Distributed Analysis of Vertically Partitioned Sensor Measurements under Communication Constraints

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Marco Stolpe

Dortmund

2017

Tag der mündlichen Prüfung:	31. Januar 2017
Dekan:	Prof. Dr.-Ing. Gernot A. Fink
Gutachter:	Prof. Dr. Katharina Morik
	Prof. Dr. Jakob Rehof
	Prof. Dr. Ulf Brefeld

Acknowledgements

This thesis couldn't have been created without support. First of all, I'd like to express my gratitude to my supervisor, Prof. Dr. Katharina Morik. Finishing this thesis would have been impossible without her continuous encouragement and never ending trust in my person. Also, I'd like to thank the other reviewers of this thesis, Prof. Dr. Jakob Rehof and Prof. Dr. Ulf Brefeld, for taking the time of reading it and writing the final reviews. I thank my colleagues at the Artificial Intelligence Group for the many interesting collaborations, discussions and all the fun we had. I also received great support by the members of the SFB 876, who always had an open ear, patiently listened to my questions and gave valuable advice. Especially, I thank our partners in project B3, Daniel Lieber in particular, for the intensive collaboration on data acquisition, storage and the preprocessing of value series. Also, I thank our industrial partner for providing data about the processing of steel blocks. I further recognize that my research would not have been possible without the financial support by the Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876, project B3.

I thank Dr. Kamalika Das and Dr. Kanishka Bhaduri for the insightful discussions about distributed data mining here at our chair and the many collaborations following. I appreciate the time they spent with me in countless phone calls and during the night before our ECML submission. Further, I'd like to thank other guests, like Giorgio Patrini, for their interesting talks and the discussions we had about them afterwards.

Finally, I acknowledge the support from my family, in particular my wife, who endured my absence while I was writing on this thesis or thought about its content. I thank you very much, Dörte, for all your patience and encouragement throughout these years!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Every day, data is generated by humans using devices as diverse as personal computers, company servers, electronic consumer appliances or mobile phones and tablets. Due to tremendous advances in hardware technology over the last few years, nowadays even larger amounts of data are automatically generated by devices and sensors, which are embedded into our physical environment. They measure, for instance,

- machine and process parameters of production processes in manufacturing,
- environmental conditions of transported goods, like cooling, in logistics,
- temperature changes and energy consumption in smart homes,
- traffic volume, air pollution and water consumption in the public sector or
- pulse and blood pressure of individuals in healthcare.

The collection and exchange of data is enabled by electronics, software, sensors and network connectivity, that are embedded into physical objects. The infrastructure which makes such objects remotely accessible and connects them, is called the *Internet of Things (IoT)*. In 2010, already 12.5 billion devices were connected to the IoT [Eva11], a number about twice as large as the world's population at that time (6.8 billion).

The IoT revolutionizes the Internet, since not only computers are getting connected, but physical things, as well. The IoT can thus provide us with data about our physical environment, at a level of detail never known before in human history [Ora15b]. Understanding the generated data can bring about a better understanding of ourselves and the world we live in, creating opportunities to improve our way of living, learning, working, and entertaining [Eva11]. Especially the combination of data from many different sources and their automated analysis may yield new insights into existing relationships and interactions between physical entities, their environment and users. This facilitates to optimize their behavior. Automation of the interplay between data analysis and control can lead to new types of applications that use fully autonomous optimization loops. Examples will be shown, indicating their benefits.

However, IoT's inherent distributed nature, the resource constraints and dynamism of its networked participants, as well as the amounts and diverse types of data are challenging even the most advanced automated data analysis methods known today. In particular, the IoT requires a new generation of distributed algorithms which are resource-aware and intelligently reduce the amount of data transmitted and processed throughout the analysis chain.

Distributed data analysis algorithms developed for the IoT can be divided into two main types. The first type targets data centers created for high performance cloud computing. The second type targets pervasive systems consisting of small devices connected in wireless networks. Both kinds of environments will be part of the IoT. In pervasive systems, like wireless sensor networks, resources are much more scarce than in data centers for high performance computing. The most important difference are constraints on communication, since wireless connections have low bandwidth, and the transmission of data is known to be the most expensive operation for battery-powered devices like mobile phones or smart sensors. However, there are other examples of communication-constrained scenarios, for instance the real-time analysis of data streamed by high throughput applications, or settings in which the privacy of data needs to be preserved, not allowing for the transmission of data to other networked nodes.

A particularly challenging setting for the distributed analysis of data in an IoT context, with many relevant applications, is the vertically partitioned data scenario. Here, information about single observations is distributed over different physical nodes. The learning of accurate prediction models and prediction itself may thus require the combination of information from different nodes, necessarily leading to communication. The main question is how to design communication-efficient algorithms for the scenario, while at the same time preserving sufficient accuracy. This thesis focuses on the distributed analysis of sensor measurements in the vertically partitioned data scenario under communication constraints.

The first part, "Fundamentals", introduces important definitions, concepts, notations, algorithms and problems in so far as they serve the understanding of the second part.

Chapter 2 gives an overview of the IoT and its many applications, with a special focus on data analysis. It then explains the important differences between high performance cloud computing and much more communication-constrained settings, like pervasive distributed systems or high throughput applications. The vertically partitioned data scenario is described in more detail, together with applications. Challenges of designing communication-efficient algorithms for the scenario are stressed. This results in a list of important research questions which have driven the development of algorithms introduced in the second part of this thesis.

Chapter 3 introduces the topic of machine learning and data mining, from a theoretical perspective, as well as a practical one, giving examples of learning algorithms.

Chapter 4 gives an introduction into distributed systems and stresses the differences between parallel and distributed computing. A selection of different approaches for the distributed analysis of data is discussed in more detail, and should give a good

impression of the underlying problems, principles and techniques used by different classes of distributed algorithms. Finally, the specific challenges of the vertically partitioned data scenario are reviewed from a learning perspective.

Chapter 5 discusses the important task of preprocessing time-related sensor measurements, in the context of a smart manufacturing case study. In particular, it is described how sensor measurements assessed during a hot rolling mill process are preprocessed before analysis. The results of the case study are presented, and conclusions are drawn, which serve as a motivation for the algorithms developed in the second part of this thesis.

The second part, "Algorithms", introduces new algorithms for distributed learning in the vertically partitioned data scenario.

Chapter 6 first discusses the relatively novel problem of learning from label proportions and how it relates to smart manufacturing and issues of privacy. A new algorithm for the learning task is developed and evaluated, the Learning from Label Proportions by Clustering (LLPC) algorithm. The algorithm's performance is compared to three other state-of-the-art approaches, in terms of accuracy and running time. It can be shown that the algorithm's accuracy is similar to the accuracy of its competitors, or significantly higher in the case of larger bag sizes, where learning is more difficult. At the same time, LLPC's asymptotic running time is only linear, while the running time of its competitors is at least quadratic. The proposed algorithm comes with many other benefits, like ease of implementation and a small memory footprint. It is shortly explained how algorithms developed for the scenario may be used for the communication-efficient transmission of labels, motivating the next chapter.

Chapter 7 proposes a communication-efficient decentralized in-network classification algorithm, the Training of Local Models from (Label) Counts (TLMC). The method reduces communication by only transferring aggregated label information between nodes, namely the counts of labels. At each local node, it transforms label counts into proportions and learns from them with the previously introduced approach for learning from label proportions. Feasibility of the approach is demonstrated by evaluating the algorithm's performance in the application context of traffic flow prediction. It is shown that TLMC is much more communication-efficient than centralization of all data, but that accuracy can nevertheless compete with that of a centrally trained global STRF model.

Chapter 8 introduces a communication-efficient distributed algorithm for anomaly detection, the Vertically Distributed Core Vector Machine (VDCVM), which is based on distributing kernel calculations of the Core Vector Machine (CVM). It can be shown that the proposed algorithm communicates up to an order of magnitude less data during learning, in comparison to another state-of-the-art approach or training a global model by the centralization of all data. Nevertheless, in many relevant cases, the VDCVM achieves similar or even higher accuracy on several controlled and benchmark datasets.

Chapter 9 summarizes the thesis and draws conclusions. Further research opportunities are discussed by listing some open questions concerning the developed algorithms and learning in the vertically partitioned data scenario in general.

Part I

Fundamentals

Chapter 2

The Internet of Things (IoT)

The IoT consists of physical objects (or "things") which embed electronics, software, sensors, and communication components, enabling them to collect and exchange data. Physical things are no longer separated from the virtual world, but connected to the Internet. They can be accessed remotely, i.e. monitored, controlled and even made to act.

Ideas resembling the IoT reach back to the year 1988, starting with the field of ubiquitous computing. In 1991, Mark Weiser framed his ideas for the computer of the 21st century [Wei91]. Weiser envisioned computers being small enough to vanish from our sight, becoming part of the background, so that they are used without further thinking. Rooms would host more than 100 connected devices, which could sense their environment, exchange data and provide human beings with information similar to physical signs, notes, paper, boards, etc. Devices would need self-knowledge, e.g., of their location. Many of Weiser's original ideas can still be found in current definitions of the IoT and requirements for according devices. For example, Mattern and Floerkemeier [MF10] enumerate similar capabilities needed to bridge the gap between the virtual and physical world. Objects must be able to communicate and cooperate with each other, which requires addressability, unique identification, and localization. Objects may collect information about their surroundings and they may contain actuators for manipulating their environment. Objects can embed information processing, featuring a processor or microcontroller, and storage capacity. Finally, they may interface to and communicate with humans directly or indirectly. In a report by Verizon [Ver15], the IoT is defined as a machine to machine (M2M) technology based on secure network connectivity and an associated cloud infrastructure. Things belonging to the IoT follow the so called three "A"s. They must be *aware*, i.e. sense something. They must be *autonomous*, i.e. transfer data automatically to other devices or to Internet services. They also must be *actionable*, i.e. integrate some kind of analysis or control.

The history of the IoT itself started in 1999, with the work on Radio-frequency identification (RFID) technology by the Auto-ID Center of the Massachusetts Institute of Technology (MIT) [MF10, Eva11]. The term "Internet of Things" was first literally

used by the center's co-founder Kevin Ashton in 2002. In a Cisco white paper, Dave Evans [Eva11] estimates that the IoT came into real existence between 2008 and 2009, when the number of devices connected to the Internet began to exceed the number of human beings on earth. Many of such devices were mobile phones, after in 2007, Steve Jobs had unveiled the first iPhone at Macworld conference. Since then, more and more devices are getting connected. It is estimated that by 2020, the IoT will consist of almost 50 billion objects [Eva11].

The World Wide Web (WWW) fundamentally changed in at least four stages [Eva11]. First, the web was called the Advanced Research Projects Agency Network (ARPANET) and foremost used by academia. The second stage was characterized by companies acquiring domain names and sharing information about their products and services. The "dot-com" boom may be called the third stage. Web pages moved from static to interactive transactional applications that allowed for selling and buying products online. The "social" or "experience" web marks the current fourth stage, enabling people to communicate, connect and share information. In comparison, Internet's underlying technology and protocols have gradually improved, but didn't change fundamentally. Now, connecting billions of physical things, crossing borders of entirely different types of networks poses new challenges to Internet's technologies and communication protocols. This is why the IoT was called the first evolution of the Internet [Eva11].

As did the Internet, the IoT has the potential to change our lives in fundamental ways. Gathering and analyzing data from many different sources in our environment may provide a more holistic view on the true relationships and interactions between physical entities, enabling the transformation of raw data and information into long-term knowledge and wisdom [Eva11]. The timely identification of current trends and patterns in the data could further support proactive behavior and planning, for instance by anticipating natural catastrophes, traffic jams, security breaches, etc. The IoT may also create new business opportunities. Potential benefits for companies are improved customer and citizen experience, better operation of machines and quality control, accelerating growth and business performance, as well as improving safety and a reduction of risk. Verizon estimates that by 2025, companies having adopted IoT technology may become 10% more profitable. Other sources predict profit increases by up to 80%. It is further estimated that the number of business to business (B2B) connections will increase from 1,2 billion in 2014 to 5,4 billion by 2020 [Ver15].

Many surveys about the IoT (for instance, [AIM10, GBMP13, PK13, XHL14]) discuss IoT's underlying technologies, others [RZL13, Fle15] security and privacy issues. Data analysis' role and related challenges are only covered shortly, if at all. Some surveys [AAS13, BdDPP16, CDW+15, DMR16] mention the problem of big data analysis and propose centralized cloud-based solutions, following the paradigm of parallel high performance computing. The authors of [GBA+13], [TLV14] and [QSF+16] take a more things-centric perspective and argue for the analysis and compression of data before its transmission to a cloud. [BYX10] identify the need for decentralized analysis algorithms, in addition. [TLCY14] present existing applications of well-known data analysis algorithms in an IoT context, highlighting decentralized data analysis as open issue

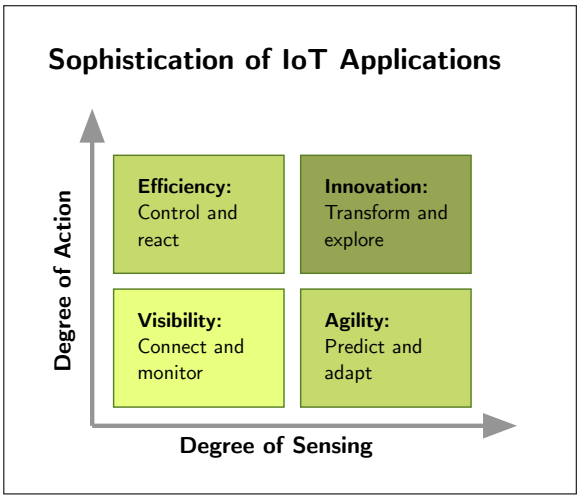


Figure 2.1: Sophistication levels of IoT applications [104]

concerning infrastructure. However, they do not address an algorithmic perspective.

To the best of our knowledge, our following survey is the first one dealing with differences between cloud-based and decentralized data analysis from an algorithmic perspective. In Sect. 2.1, we show, how advanced levels of data analysis could enable new types of applications. Section 2.2 presents the challenges of data analysis in the IoT and argues for the need of novel data analysis algorithms. Like many other authors, we see the convenience and benefits of cloud-based solutions. However, we want to move further and enable data analysis even in resource-restricted situations (Sect. 2.3). In Sect. 2.4, we argue in favor of data reduction and decentralized algorithms in highly communication-constrained scenarios which existing surveys largely neglected, so far. We focus on communication-efficient distributed analysis in the vertically partitioned data scenario, which covers common IoT use cases. Section 2.5 presents research questions and directions, many of which are dealt with in later parts of this thesis, having driven the development of communication-efficient algorithms presented there. Finally, we summarize and draw final conclusions.

2.1 Data-Driven IoT Applications

In [MBC+09, CLR10], IoT applications are categorized by their level of *information and analysis* vs. their level of *automation and control*. A similar distinction is made in [Ver15], which measures the sophistication of IoT applications by two factors, namely the *degree of action* and the *degree of sensing* (see Fig. 2.1). Applications falling into the lower left corner of the diagram in Fig. 2.1 already provide benefits given the ability to connect to and monitor physical things remotely. Giving objects a virtual identity independent of their physical location highly increases their visibility and can facilitate

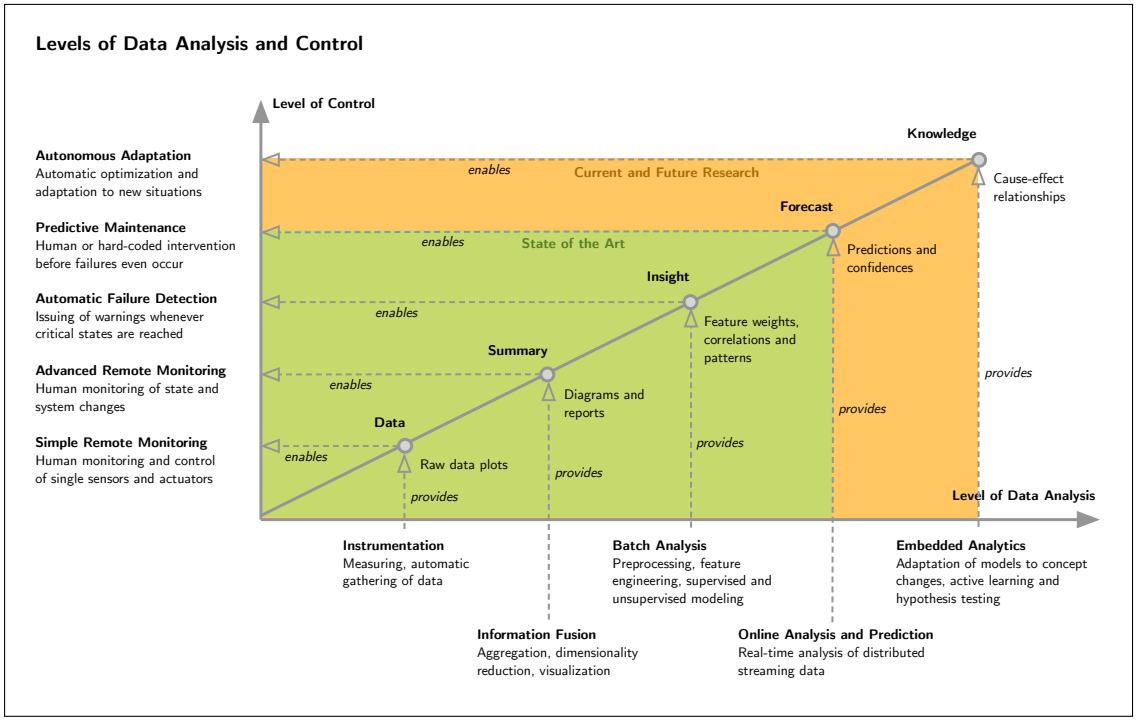


Figure 2.2: Relationship between data analysis and control

decision making based on smart representations of raw data. Applications located in the upper left corner of Fig. 2.1, in addition, use embedded actuators. Beyond pure monitoring, they enable remote control of physical things, thereby easing their management. Applications that analyze IoT generated data fall into the lower right corner of Fig. 2.1. Here, especially the combination of data from different physical objects and locations could provide a more holistic view and insights into phenomena that are only understood poorly, so far.

Though we agree with the previously presented categorizations, they don't show the dependency of advanced control mechanisms on data analysis. Data analysis could turn data into valuable information, which can then be utilized for building long-term knowledge and proactive decision making. Finally, merging analysis and control may lead to innovative new business models, products and services. We therefore propose the scheme in Fig. 2.2 which stresses the analysis. We structure the field along the dimensions of *control* and *data analysis*. The diagonal shows the milestones on the path to fully embedded analytics, which is put to good use in automatic system optimization.

The data gathered from single sensors for analysis enables simple remote monitoring applications. Here, the informed choice and placement of sensors during instrumentation depend on a well-defined analysis goal [ZSS+16, SBM16]. Advanced applications move from the observation of single sensors to the monitoring of system and process states.

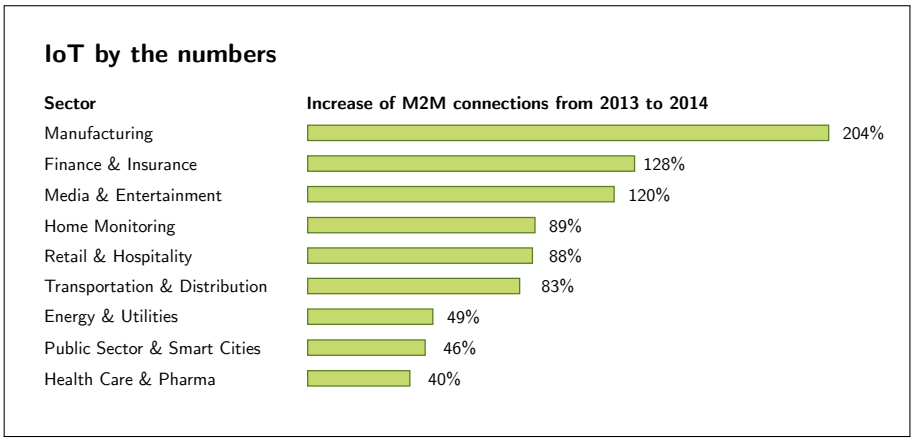


Figure 2.3: Increase of M2M connections in Verizon's network from 2013 to 2014 [104]

This monitoring is based on the visualization of summary information obtained with the help of data analysis from multiple types of sensors and devices. The batch analysis of historical records finds correlations between features and relate them to a target value. Insights gained from this step may lead, for instance, to a better understanding of critical failure conditions and their automated detection. Prediction models derived from batch analysis may also be deployed for real-time forecasts. This is current state-of-the-art.

However, depending on the amount and rate of generated measurements, their preprocessing may become infeasible. Hence, current research focuses on distributed streaming analysis methods and the intelligent reduction of data directly at the sensors and devices themselves (see Sect. 2.2.3 and Sect. 2.3.2). Data analysis which is embedded into all parts of an IoT system will finally require the real-time derivation of models and an adaptation to changes in underlying data distributions and representations. This would in turn allow for a continuous and automated monitoring of changes in correlations. The full integration of data analysis and control introduces an automated conduction of cause-effect analysis by active testing of hypotheses, moving beyond the detection of correlations. Knowledge about causal relationships may then be used to autonomously adapt the relevant parameters in new situations. Limiting models and their use to a small selection of parameters saves memory, computing, and energy resources.

Figure 2.3 shows the increase of M2M connections for different business sectors in Verizon's network from 2013 to 2014. In the following, we present examples of specific IoT applications from the sectors mentioned at the beginning: Manufacturing, transportation and distribution, energy and utilities, the public sector and smart cities, as well as healthcare and pharma. We have ordered examples of each sector according to the different levels of data analysis and control as shown in Fig. 2.2 and have identified three main application types: Predictive maintenance, sustainable processes saving resources and quality control.

2.1.1 Manufacturing

The manufacturing sector supports the development of IoT by the provision of smart products. For instance, 43 million wearable bands were shipped in 2015 [Can14], and it is estimated that 20 million smart thermostats will ship by 2023 [Nav14]. By 2016, smart products will be offered by 53% of manufacturers [Oxf13].

The sector not only produces devices, but also uses IoT technology itself. According to Fig. 2.3, the manufacturing sector is seeing the largest growth in terms of M2M connections in Verizon's network. Following the levels of Fig. 2.2, we now present types of industrial applications.

Simple remote monitoring applications increase visibility by embedding location-aware wireless sensors into products and wearables [Ver15]. This allows for a continuous tracking of persons and assets, like available stock and raw materials, on- and offsite over cellular or satellite connections. [Ver15] further mentions sensors which can detect hazards or security breaches by the instrumentation of products and wearables. Embedding sensors into production machinery will allow for the monitoring of individual machines with high granularity along the process chain. It should be added, however, that the automatic detection of such events necessarily requires an analysis and interpretation of measurements.

The aggregation of data from the same type of sensors supports the confidence in the accuracy of analysis results. Moreover, the fusion of data from different types of sensors advances remote monitoring of larger units, like systems, processes and their environment. For instance, [SBM16] visually identify and quantify different types of productions modes in steel processing by summarizing multi-dimensional sensor data with algorithms for dimensionality reduction.

Models derived from heterogenous data sources by batch analysis may provide insights into the correlations between multiple dimensions of process parameters and a target value. According to [Ver15], the timely identification of failure states can lead to less disruption and increase uptime in comparison to regular human maintenance visits and inspections. It should be added that once trained, data analysis models can often be made directly operational, and be used, for instance, for the automatic detection of critical patterns. For instance, learned models may be deployed early in the process for the automatic real time prediction of a product's final quality [SBM16], allowing for timely human intervention. Here, resources might be saved by omitting further processing of already defect products. Based on human knowledge, control parameters might be adjusted such that a targeted quality level can still be reached. In the context of maintenance, the quantity to be predicted is machine wear or failure. The timely detection of anomalies and machine wear can help with reducing unplanned downtime, increasing equipment utilization and overall plant output [SBM16, Ver15]. However, depending on the amount of generated data, batch analysis as well as preprocessing all data in real-time can be challenging [SMK+11]. Advanced applications therefore require the development of new kinds of data analysis algorithms (see Sect. 2.2.3 and Sect. 2.3.2).

Making data acquisition and analysis an integral part of production systems could

finally allow for the long time observation of changes in correlations between process parameters and target variables. The importance of manufacturing for the adoption of IoT is emphasized by the German initiative "Industrie 4.0". It fosters the integration of production processes, IoT technology and cyber-physical systems into a so called *smart factory*. In this future type of factory, products can communicate with their environment, for instance with other products, machines and humans. In contrast to fixed structures and specifications of production processes that exist today, Reconfigurable Manufacturing Systems (RMS) derive case-specific topologies automatically based on collected data [BFKR14]. Hence, production will become more flexible and customized. Reactions to changes in customer demands and requirements may take only hours or minutes, instead of days. RMS might further support the active testing of hypotheses and targeted generation of new observations. The resulting variability of large numbers of observations might then help with automatically distinguishing between random correlations of parameters and those the target variables truly depend on. Such knowledge could then be used for the automatic optimization and autonomous real-time adaptation of production processes and their parameters to new situations. The intelligent combination of data analysis and control can thereby lead to more sustainable systems which allow for major reductions in waste, energy costs and the need for human intervention [CLR10, SBM16].

2.1.2 Transportation and Distribution

The sector of transportation and distribution belongs to the early adopters of IoT. Here, according to [Ver15], important factors for the adoption of IoT technology are regulations and competition which force higher standards of efficiency and safety, as well as expectations of greater comfort and economy. From 2013 to 2014, the sector has seen a 83% increase of M2M connections in Verizon's network (see Fig. 2.3).

The instrumentation of vehicles enables simple remote monitoring applications that make it easier to locate and instruct fleets of cars, vans or trucks [Har15]. Logging driver's working hours, speed and driving behavior can improve safety and simplify compliance with regulations [Ver15]. Customers can be regularly informed about the delivery times of anticipated goods. Even containers themselves are now equipped with boards of very restricted capacities, which open up opportunities of tracing and organizing the goods in a logistic chain of storage and delivery [VRR+15].

Another example for new types of applications is the UBER smartphone app which indicates the location of passengers calling a taxi to nearby drivers and uses surge pricing to fulfill demands for more taxis.

Advanced remote monitoring applications use data analysis to aggregate data and may provide summaries of fleet movements on a larger scale, like the average number of vehicles traveling certain routes, thereby facilitating resource planning [KBL+04].

Instrumentation allows car manufacturers deeper insights into the use of their cars. Models derived by the batch analysis of data gathered from many cars could automatically be deployed inside cars to identify or predict failure conditions. These models

may also provide information about the relationships between failures and underlying causes. According to [Ver15], such information would allow to preemptively issue recalls, improve designs to iron out problems, and better target new features to driver and market preferences. Intelligence built into vehicles, like proximity lane sensors, automatic breaking, head lamps, wipers, and automated emergency calls can increase road safety [Ver15].

Advanced applications, like autonomously driving vehicles [BW16], require the embedded real-time analysis of data directly inside the vehicle. In addition, information sent by nearby infrastructure, like traffic signals, traffic signs, street lamps, road works or local weather stations might be taken into account (see also Sect. 2.1.4). For navigation, vehicles may remotely access current information on street maps.

At a larger scale, data gathered from many vehicles and infrastructure could be analyzed and used to instruct vehicles beyond their individual driving decisions. [KSG10] developed a sophisticated distributed analysis of local data from vans of a fleet, which allows to manage the overall fleet. Work orders can be allocated in real-time more efficiently, adopting to drivers, reacting to order changes, or other events. The effects on cutting fuel costs, leading to more sustainable vehicles and distribution systems has been shown [MBK12]. Similarly, through timely diagnostics, predictive analytics, and the elimination of waste in fleet scheduling, the rail industry is looking to achieve savings of 27 billion dollars globally over 15 years [EA12].

2.1.3 Energy and Utilities

In the sector of energy distribution, IoT applications range from telematics for job scheduling and routing, to bigger ones extending the life of electricity infrastructure [Ver15]. According to Fig. 2.3, the energy sector has seen an estimated growth of 49% in the number of M2M connections from 2013 to 2014.

Concerning remote monitoring, the energy sector was the first to introduce SCADA (supervisory control and data acquisition). Smart meters increase visibility by providing more granular data. Thereby they reduce the inconvenience and expense of manual meter readings or estimated bills. Further, advanced remote monitoring provides more accurate views of capacity, demand and supply over different smart homes, made possible by visualizing summary information obtained from data analysis [MK11, KMS+16, ZY16]. Based on such information, sustainability may be improved through better resource planning and cutting energy theft. According to [Ver15], in 2014, 94 million smart meters were shipped worldwide and it is predicted that by 2022, the number of smart meters will reach 1.1 billion. One target of the European Union is to replace 80% of meters by smart meters by 2020, in 28 member countries.

Beyond monitoring applications, the batch analysis of data from smart homes may help with giving recommendations for saving energy and enable more sophisticated energy management applications [ZY16]. Oil and gas companies can cut costs and increase efficiency by early predicting the failure of artificial components, local weather conditions, and the automated start up and shutdown of equipment [Ver15]. On a larger

scale, the smart grid connects assets in the generation, transmission and distribution infrastructures. Especially in recent years, energy use has become harder to predict, due to a decentralization of energy production. The prediction of wind power [THK16] and photovoltaic power [WLK16] is important in order to better understand grid utilization. Data analysis may increase efficiency and optimize the infrastructure [KMS+16]. The embedded real-time analysis of data could enable even more sustainable distributed energy generation models in which highly autonomous systems react dynamically to changes in energy demand and distribute energy accordingly.

2.1.4 Public Sector

In the public sector, M2M connections have grown by 46% from 2013 to 2014 according to Fig. 2.3. It is estimated that by 2050, 66% of humans will live in urban areas [Uni14] and 75% of world's energy use is taking place in cities [Ver15]. The IoT promises the delivery of more effective services to citizens, like citizen's participation, controlling crime, the protection of infrastructure, keeping power and traffic running, and building sustainable developments with limited resources [Ver15]. The IoT thus enables municipal leaders to make their communities safer and more pleasant to live, and to deal better with demographic changes [Ver15].

The instrumentation of cities with sensors may lead to more sustainable resource usage by simple remote monitoring applications. For instance, currently it takes 20 minutes on average to find a parking space in London [Ver15] and 30% of congestion in cities is caused by people looking for a parking space [Sho11]. The smart city of Santander [Sma16] has instrumented, among others, parking lots. Their space utilization could be tracked and provided as information to smart phone apps. Advanced applications may also identify trends and anomalies in parking data [ZRLP14]. Similar tracking apps could support car-sharing or unattended rental programs that offer on-demand access to vehicles by the hour [Ver15]. More advanced remote monitoring applications could indicate the crowdedness of neighboring cities by aggregating data with the help of data analysis. Using real-time analysis, they might as well give direct recommendations, for instance which city to visit for more relaxed shopping.

Resource savings can also be expected from a more sustainable management of water. IBM offers an intelligent software for water management that uses data analysis for visualization and correlation detection [IBM16]. According to IBM, the software helps to manage pressure, detect leaks, reduce water consumption, mitigate sewer overflow and allows for a better management of water infrastructure, assets and operations.

Currently, up to 40% of municipal energy costs come from street lighting [Woo12]. The European Union has set a target to reduce CO_2 emissions of professional lighting by 20 million tons by 2020 [Ver15]. Predictive models obtained through data analysis enable smart streetlights that automatically adjust their brightness according to the expected volume of cars and weather conditions. In a case study it was shown that the city of Lansing, Michigan, could thereby cut the energy and maintenance costs of street lighting by 70% [Ste12, Ver15].

Further resources might be saved by using more intelligent transportation and traffic systems. Predicting traffic flow on the basis of past data that has been measured by sensors in the streets offers drivers an enhanced routing. The German government estimated a daily fuel consumption in Germany due to traffic jams of 33 millions of liter, a waste of time in the range of 13 million hours and concludes that traffic jams are responsible for an economic loss of 259 million Euro per day. For instance, the SCATS system [SCA13] provides traffic flow data for different junctions throughout Dublin city. Simple remote monitoring can provide data about the current traffic flow to individual drivers by plotting counts of cars on a digital street map. The batch analysis of traffic data could help with determining factors causing traffic jams, which in turn might be used by traffic managers to adapt the street network accordingly. For the City of Dublin, traffic forecast derived from a spatiotemporal probabilistic graphical model, was exploited for smart routing [LPBM14]. In the future, such recommendations may be as well given to autonomously driving vehicles (see also Sect. 2.1.2).

Embedding data analysis everywhere in a city and combining the data from multiple heterogenous systems and other cities may even provide larger value. Such combination could provide a holistic view of everything, like energy use, traffic flows, crime rate and air pollution [Ver15]. Correlations and relationships between seemingly unrelated variables are not necessarily obvious. For instance, according to the broken windows theory, the prevention of small crimes such as vandalism helps with preventing more serious crimes. However, critics state that other factors have more influence on crime rate. Up to now, such theories are hard to test and validate, since studies conducted by humans can only focus on a limited number of influence factors and might be biased. The instrumentation of many different cities and areas could increase the number of observations and help with obtaining more objective and statistically significant results. Long time observation of many different variables and active hypothesis testing, for instance by giving recommendations to city planners, may help with the detection of causes that underly phenomena. The insights gained may then enable better policy decisions.

2.1.5 Healthcare and Pharma

According to Fig. 2.3, healthcare has seen the smallest growth in M2M connection from 2013 to 2014. Similarly, Gartner estimates that it will take between five and 10 years for a full adoption of the IoT by health care. This slow adoption rate may be explained by strict requirements for keeping data of patients private and secure [Gla15], with the IoT posing many challenges for privacy and security (see also Sect. 2.2). Despite such difficulties, the number and possible impact of IoT applications in healthcare is large.

The instrumentation of healthy citizens as well as patients, devices or even whole hospitals with different kinds of sensors enables different kinds of remote monitoring applications. It starts with consumer-based devices for personal use. In two years, there will be 80 million wearable health devices [Gla15], like fitness trackers and smart watches. New kinds of devices are able to monitor not only the number of steps taken

or calories, but also pulse rate, blood pressure, or blood sugar levels. The aggregation of these kinds of different information requires data analysis [Faw16]. Monitoring might promote healthy behavior through increased information and engagement [KNK+15]. In addition, physicians may get more holistic pictures of their patients' life styles, which eases diagnosis [BZ11].

Monitoring can be done remotely and continuously in real time, beyond office visits, with patients staying at home [BZ11, MSDPC12, CLR10]. Emergencies can be detected early, like with breath pillows for children or Ion mobility spectrometry combined with multi-capillary columns (MCC/IMS) that can give immediate information about the human health status or infection threats [HSP+12]. In the case of chronic illnesses, practitioners get early warning of conditions that would lead to unplanned hospitalizations and expensive emergency care [CLR10, Har15, KNK+15, Gla15]. Monitoring alone could reduce treatment costs by a billion dollars annually in the US [CLR10]. According to [Gla15], estimates show a 64% drop in hospital readmissions for heart failure patients whose blood pressure and oxygen saturation levels were monitored remotely. Similarly, at-risk elderly individuals may longer stay in their own homes. Here, remote monitoring can reassure loved ones by detecting falls or whether an individual got out of bed in the morning, or whether an individual took his or her medicine [KNK+15].

Monitoring may as well help with drug management and the detection of fraudulent drugs in the supply chain, by incorporating RFID tags in medication containers and finally embedding technology in the medication itself [Har15]. In hospitals, medical equipment like MRIs and CTs can be connected and remotely monitored, helping with maintenance, replenishing supplies and reducing expensive downtime [Gla15].

While conditions based on a few measurements may be detected automatically based on hard-wired rules, the detection of more complex patterns necessarily requires the analysis of data.

Data analysis is also needed, if we want to identify critical patterns in patient's vital parameters [IFGL03, LG10] or in movements through hospitals and optimize flow [Gla15]. The analysis of multi-dimensional data is necessary for discovering dependencies between many variables, like, e.g., the duration of treatments and waiting times at other wards. Data analysis provides doctors with insights of scientific value, taking the data gathered by many individuals as population-based evidence [KNK+15]. Clinical and nonclinical data of larger population samples may help to understand the unique causes of a disease. Finally, data analysis that was directly embedded into devices like electrocardiograms (ECG) or wireless electrocardiograms (WES) could help with the detection of emergency cases in real-time [CMW+13].

2.2 Data Analysis Challenges

The previous section has given many examples of applications in diverse sectors, showing that advanced levels of control not only require the instrumentation of devices, but also an analysis of the acquired data. These examples support our view expressed in Fig. 2.2

that it is data analysis which enables advanced types of control. Unfortunately, the IoT poses new challenges to data analysis. The following sections present problems in terms of security and privacy, technical issues as well as algorithmic challenges which require research on new types of data analysis methods.

2.2.1 Security and Privacy

Despite IoT's anticipated positive effects, it also poses risks for our security and privacy. Especially sectors that deal with highly personalized information, such as healthcare (see Sect. 2.1.5), require according means for the secure and privacy-preserving processing of data. Apart from having to make existing data analysis code more secure, analysis can as well provide solutions to decrease existing threats.

Security The biggest security risk of IoT stems from its biggest benefit, namely the connection of physical things to a global network. In the past, security breaches were mostly restricted to the theft and manipulation of data *about* physical entities. However, the IoT allows for a direct control of the physical entities *themselves*, many of which belonging to critical infrastructures in sectors previously mentioned. Without security measures, malware like viruses could easily spread through many of IoT's connected networks, potentially resulting in disasters at a global scale [Fle15, Dix15].

Data analysis algorithms can be made secure by design. However, existing code bases weren't necessarily designed and implemented with security in mind. In the past, algorithms could be expected to run mostly in environments which weren't publicly accessed. Further, the way how data has been input into analysis software was relatively controlled. With the IoT, analysis code will run on devices directly exposed to an open network environment and is thus susceptible to malicious hacking attempts. It will be much harder to ensure that data originates from trustworthy sources and is in appropriate format. Hackers might gain access to sensors and other embedded devices [RZL13, Fle15, Dix15], or install rogue devices that interfere with existing network traffic [RZL13]. Hence, it becomes more and more important to make data analysis code more robust by penetration testing [Eng13] and differentiate hacking attempts from usual sensor failure. Also, legal liability frameworks must be established for algorithms whose decisions are fully automated [CLR10].

At the same time, data analysis might provide solutions for the automatic detection or even prevention of security breaches. For instance, outlier and novelty detection algorithms which examine deviations from normal behavior have already been used successfully in fields like intrusion or malware detection [BG15, BAM09].

Privacy Another of IoT's challenges is the protection of citizens' privacy. As Mark Weiser already stated in 1991, "hundreds of computers in every room, all capable of sensing people near them and linked by high-speed networks, have the potential to make totalitarianism up to now seem like sheerest anarchy" [Wei91]. Since it became known that intelligence agencies of democratic states are spying at other friendly states and

their citizens [Tap15], the topic of privacy has developed an especially high brisance. It also plays a large role in business sectors where data is highly personalized. For instance, data in healthcare must be especially protected.

One problem is that with small embedded devices vanishing from our sight, people might not even recognize that data about them is getting acquired. Further, it may not be entirely clear how data given away will be combined later on and what can then be derived from it. For instance, as research on learning from label proportions [SM11, PNCR14] suggests, information that seems harmless all by itself, like public election results, may become problematic once it is combined with data from other sources, such as social web sites.

It is important to mention, however, that several of the aforementioned benefits from data analysis can be achieved without highly personalized data [GP07]. For instance, disease research based on population-based evidence (see Sect. 2.1.5) would yield the same results with anonymized observations. If that doesn't suffice and enough samples are present, data can further be aggregated to guarantee k-anonymity [Swe02]. Related is the problem of learning from label proportions [SM11, PNCR14]. Where more privacy is needed, the challenge consists of developing distributed analysis algorithms that derive a model without exchanging individualized records between different networked nodes (for instance, see [DBK09]).

2.2.2 Technical Challenges

Technical challenges of IoT mainly concern networking technology, devices interoperability, as well as increasing the life-time and range of wireless battery-powered devices. Here, we list the technical problems that every application of data analysis has to face.

Data Understanding One envisioned scenario for the analysis of IoT generated data is that as people connect new devices to the IoT, their data is automatically getting analyzed, together with the data of other already existing devices. Data analysis being successful, however, depends much on the correct preprocessing of data, which in turn depends on the types and ranges of features of observations. This information can be estimated from the data. However, it can be difficult to assess the quality of such estimations without ground truth. For instance, outlier detection algorithms may indicate measurements which occur only seldom. However, without additional background knowledge provided by experts, it is impossible to determine automatically if values are still inside physically meaningful ranges or caused by sensor failure. Similarly, peak detection algorithms might wrongly identify noise as relevant patterns. These problems could easily be solved if manufacturers made their sensors and embedded devices queryable and provided meta data, e.g. meaningful ranges and noise levels of theirs sensors.

Standardization The ability to query sensors and devices for meta information requires standardized protocols. A similar standardization is needed for the exchange of

raw data. Especially in industry, closed systems with proprietary data formats complicate the exchange of data between distributed components and make automated data analysis unnecessarily difficult [SBM16]. Similarly important would be a standardization of user interfaces for data analysis tools. As Mark Weiser already noted in [Wei91], technology becomes unobtrusive once its user interfaces are as uniform and consistent as possible. In contrast, today the user interface of operating systems and applications often is their most distinguishing property and therefore a unique selling point. Hence, a wide adoption of common standards requires that profits made from IoT technology outweigh potential losses caused by the lacking individualization of products.

Porting existing code bases As Sect. 2.2.1 already discussed, existing code bases for data analysis must be made more robust to operate in hostile network environments. In addition, as more and more data analysis algorithms can be expected to run directly on embedded and mobile devices, existing code and related libraries need to be ported to these platforms. The implementation language of choice for embedded devices is C/C++. In contrast, much data analysis code is written in Java and Python, whose virtual machines and interpreters require too many resources to run on small embedded devices like sensors. Currently, the same algorithms must therefore be implemented in many different versions, making the reuse of existing code more difficult. Beyond modification of existing code bases, the IoT poses several challenges that require research on new algorithms, as described in the next section.

2.2.3 Algorithmic Challenges

Manual inspection of IoT generated data is possible only in simple cases. Normally, since the amount of data generated by single sensors becomes too high, the analysis needs to be fully automated. Further, the combination of data from many heterogenous sources leads to high-dimensional datasets that cannot be easily visualized or examined by humans.

Automated data analysis methods have been developed in the fields of signal processing and computer vision [Dav12], statistics [HTF09], artificial intelligence [RN13], machine learning [Mit97], data mining [HK06] and databases [GMUW13], to name just some text books. Among them are sophisticated methods that can generalize over raw data, deriving *models* that describe patterns and relationships which statistically hold on expectation also for unseen observations. Such methods will be called *learning algorithms* in the following. Unsupervised learning algorithms find general patterns and relationships in the data. Supervised algorithms find such patterns in relation to a specified target value, which at best should be given as label for each observation. The difficulty in both cases is that the model must be derived only from a given finite *sample* of the data, while the probability distribution generating the data is unknown (for a more formal definition of the problem, see [HTF09]). Many learning algorithms assume the sample to be given as a single batch which can be processed in a random access

fashion, potentially making several passes over the data. Observations are assumed to have a relatively homogenous structure and fixed representation.

The IoT poses new challenges to data analysis. At the data generating side, devices are often highly resource-constrained in terms of CPU power, available main memory, external storage capacity, energy and available bandwidth. Algorithms working at the data generating side must take these constraints into account. Also the underlying data distribution may change which is known as *concept drift* [ŽPG16]. For instance, due to wear, the accuracy of sensors may decrease.

At the receiving side, e.g. a data center, the combination of data from many different sources may create huge masses of heterogenous data. It is estimated that in total, the IoT will generate 4.4 trillion GB by 2020 [Ora15a]. Hence, the problem consists of having to analyze *big data* [MW14, Ora15b], which is characterized by large *volume* (terabytes or even petabytes of data), *heterogenity* (different sources and formats) and *velocity* (speed of generated data). High volume and velocity prohibit several passes over the data, and thus require new types of algorithms. In addition to the big data problem, the analysis of IoT data are distributed and asynchronous. Just to illustrate an effect of this particular setting, let us look at IoT devices dynamically entering or leaving the network. This contradicts an assumption underlying almost all data analysis approaches, namely that the representation of observations, e.g. the number of features, does not change over time.

2.3 Distributed Data Analysis

The requirements of algorithms for the analysis of IoT generated data are largely determined by the hardware and network environment in which they are expected to run. Depending on volume and rate of data generation, as well as the particular analysis problem, data must either be already preprocessed and analyzed at the generating side, on network middleware or sent to a data center. Each scenario comes with its own set of advantages and disadvantages, constraints and particular challenges. Based on specifications found on websites of cloud providers and manufacturers, we have compiled a list of computing environments and device's properties for a quick and easy comparison in Fig. 2.4.

The current focus is on the centralization of data in the cloud and its analysis by high performance computing [GBMP13, Bur14, CDW+15, Ora15b, DMR16]. Cloud computing allows for highly scalable distributed systems that solve tasks in parallel by means of virtualization. Virtual instances of nodes in a network are independent from the particular physical nodes they run on. Hence, new instances can easily be added and removed depending on current computational demands. Computation follows the paradigm of parallel computing in so far as modern frameworks shield programmers as much as possible from the intricate details of distributed systems. For instance, the scheduling and execution of code, the creation of threads or processes, synchronization as well as message passing are handled automatically. Failures that can occur in distributed

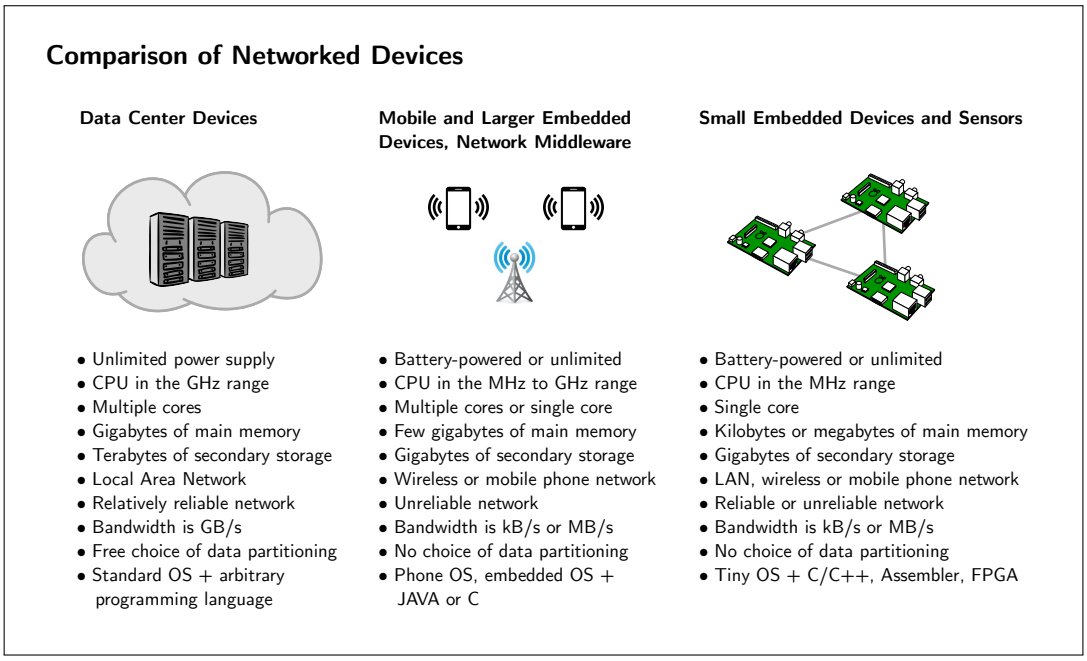


Figure 2.4: Comparison of computing environments and device types

systems are taken care of by redundancy and the automatic rescheduling of processes. The main task for programmers is to divide their problem into smaller subproblems which can be worked on in parallel. How and where code is executed is mostly transparent, giving the impression of a single big machine instead of many nodes.

As more and more devices are getting connected, existing network hardware and infrastructure will no longer suffice to handle the expected network traffic [MSDPC12, CLR10, Bur14, Ora15b, Com15]. Whenever the rate of data generation is higher than available bandwidth, data must be analyzed on the generating devices themselves or at least be reduced *before* transmission into the cloud [GBA+13, TLV14, QSF+16]. In the following, algorithms that process or analyze data directly where it is acquired will be called *decentralized*. In case they need another node for coordination, data and computation are at least split between local nodes and the coordinator. Decentralized algorithms which need no coordinator and exchange information only with local peer nodes will be called *fully decentralized*. Ideally, decentralized analysis algorithms should exchange less information than all data between nodes.

The next section presents the ideas and constraints of current cloud-based data analysis approaches in more detail, while the following section discusses the need for decentralized data analysis algorithms in more communication-constrained scenarios.

2.3.1 Data Centers and Cloud Computing

One option for the analysis of IoT generated data is its centralization at a data center. Cloud computing solutions are offered by different service providers. They allow for an easy and cost-efficient upscaling of computing and storage resources. Depending on the rate of data generation, there exist two different models of data processing: Data may either be stored and analyzed as a batch, or it must be processed directly as a stream.

Batch analysis Huge data masses which do not fit in one server require the distribution of data over different connected storage devices. This is, for instance, accomplished by saving chunks of arriving data in a distributed file system such as HDFS [SKRC10]. Once the data is stored, it can be analyzed as a batch by distributed algorithms that solve tasks cooperatively. Each machine in a data center may have multiple cores, which algorithms can exploit for parallel execution. CPUs are in the gigahertz (GHz) range and main memory has several gigabytes. Machines are usually connected in a local area network (LAN) where connections are relatively reliable. Technologies such as InfiniBand and 100 Gigabit Ethernet allow for high bandwidths which are comparable to direct main memory accesses. Reading from dynamic random access memory (DRAM) can be about one order of magnitude faster than reading from external storage mediums, like solid-state drives (SSDs). A reorganization of data would therefore be an expensive operation. Hence, it is desirable to read data from disk only once. This can be achieved by moving code to the machine storing the data and executing it locally.

The distributed batch analysis of data is currently supported by different frameworks. Hadoop [Whi11] is a popular framework. It follows the map and reduce paradigm known from functional programming, where the same code is executed on different parts of the data and the results are then merged. Map and reduce is especially well-suited for problems that are data parallel. This means that tasks can work independently from each other on different chunks of data, reading it only once, without synchronization or managing state. The paradigm lends itself well for data analysis algorithms which process subsets of observations or features only once. Some algorithms for counting, preprocessing and data transformation fall into this category.

More advanced data analysis algorithms, especially learning algorithms, often require the combination of data from different subsets. They also need to make several passes over the data, and synchronize shared model parameters. For instance, the k-Means clustering algorithm [Mac67] repeatedly assigns observations to a globally maintained set of centroids. Similarly, many distributed optimization algorithms used in data analysis maintain a globally shared set of model parameters (see also [BPC+11]). In map and reduce, distributed components are assumed to be stateless. One way to maintain state between iterations would be to access, for instance, a database server which is external to the Hadoop framework. However, this would require the unnecessary and repeated transmission of state over the network. For the implementation of stateful components, lower level frameworks like the Message Passing Interface (MPI) [Arg15]

or ZeroMQ [Hin13] are usually better suited. These frameworks allow for long running stateful components and full control over which data is to be sent over the network.

Distributed variants of well-known data analysis algorithms, like k-Means clustering [Mac67] and random forests [Bre01], have been implemented in the Apache mahout [The15b] framework that works on top of Hadoop. However, the framework contains only few algorithms, as research on distributed data analysis algorithms for high performance computing is still ongoing.

Analysis of streaming data Whenever batch processing isn't fast enough to provide an up-to-date view of the data, it must be processed as a stream [Com15, Boc15]. The Lambda architecture by Marz [MW14] is a hybrid of batch and stream processing. The batch layer regularly creates views on historical data. The speed layer processes current data items which come in while batch jobs are running, and creates up-to-date views for this data. Both views are combined at a service layer, which provides a single view on the data to users. A disadvantage of the Lambda architecture is that algorithms must be designed and implemented for different layers. Kreps [Kre14] therefore proposed the Kappa architecture, in which all data is treated as a stream.

Several frameworks support the development of streaming algorithms (for one framework and an overview, see [Boc15]). Related analysis algorithms are still an active area of research [Gam10] and are currently implemented in different frameworks [BHKP10, DFMB15, The15a].

The centralization of all data in the cloud offers several benefits. The often complicated network infrastructure needed for distributed computing as well as the corresponding machines are fully managed by the provider. Due to providers' expert knowledge, security risks might decrease. Customers pay only for those services they really use, such that it becomes easier and less costly to accommodate to spikes in network traffic. As long as the data analysis algorithms to be executed and their components can be fully parallelized, scalability is just a matter of adding new machines.

However, the centralization of all data also poses risks for privacy and may have disadvantages. In the case of data theft, all data may suddenly become accessible. Further, the cloud itself poses a single point of failure. Whenever data is generated at a higher rate than can be transmitted, either due to a limited bandwidth or high latency, the cloud can become a bottleneck for real-time analysis and control. Such cases require the local processing and reduction of data directly at the data generating side, as argued for in the next section.

2.3.2 Communication-constrained Scenarios

A central analysis of IoT generated data requires its transmission over a network. However, due to technical limitations, the transmission of *all* data to a central location, like a data center, is not always possible. Either the data generating devices themselves

Table 2.1: Data transfer rates of different technologies

Technology	**Rate**		**Type**
EDGE	29.6	kB/s	Mobile Phone
UMTS 3G	48.0	kB/s	Mobile Phone
LTE	40.75	MB/s	Mobile Phone
802.15.4 (2.4 GHz)	31.25	kB/s	Wireless
Bluetooth 4.0	3.0	MB/s	Wireless
IEEE 802.11n	75.0	MB/s	Wireless
IEEE 802.11ad	900.0	MB/s	Wireless
Solid-state drive (SSD)	600.0	MB/s	Storage
eSATA	750.0	MB/s	Peripheral
USB 3.0	625.0	MB/s	Peripheral
VDSL2	12.5	MB/s	Broadband
Ethernet	1.25	MB/s	Local Area
Gigabit Ethernet	125.0	MB/s	Local Area
100 Gigabit Ethernet	12.5	GB/s	Local Area
Infiniband EDR 12x	37.5	GB/s	Local Area
PC4-25600 DDR4 SDRAM	25.6	GB/s	Memory

are highly communication-constrained, or the available bandwidth is too limited. Moreover, there exist cases where privacy concerns, security concerns, business competition or political regulations prohibit the centralization of all data.

Communication-constrained devices One of mobile devices' biggest constraint is that they are battery powered. Devices having much less computational power, like embedded devices or smart sensors, can be battery powered as well, even if they aren't mobile. Sending and receiving data is known to be one of the most energy draining operations on mobile devices [CH10] and smart sensors [LSFB15]. Hence, communication must be traded off against computation.

Limitations of bandwidth There exist several scenarios in which the available bandwidth does not suffice to transmit all data to a central location. IoT generated data may stem from devices that are connected wirelessly. Table 2.1 shows typical transfer rates for different kinds of network technologies and bus systems. It becomes apparent that wireless networks provide much lower bandwidths than LANs which are used in data centers. For instance, ZigBee networks based on IEEE 802.15.4, a specification for personal area networks consisting of small, low-power digital radios, have a data transmission rate of only 31.25 kB/s. Mobile devices, like smartphones or tablets, are relatively pow-

erful in terms of computation and available main memory (see also Fig. 2.4). They easily may generate data at higher rates than can be transmitted over mobile telephone interfaces. Other applications, like those in earth science [ZRDZ07] or telescopes in physics [BBB+15], produce masses of data whose transmission over satellite connections is in the range of years. Masses of data are also generated by high throughput applications, like Formula One racing [Sti14], which require a real-time analysis of large amounts of data [Com15]. Similarly, analysis and control in manufacturing can have real-time constraints [SMK+11, SBM16]. In cases where reaction times lie in the range of a few seconds, it seems risky to send production parameters first into the cloud for preprocessing and analysis, which then computes an answer. Depending on latency, which can be high with Internet based services, the answer may come too late. Finally, bandwidth becomes more limited with more network participants. With the IoT, those will likely increase as more and more devices are getting connected to the same network segments [Ora15b]. According to [MSDPC12], "how to control the huge amount of data injected into the network from the environment is a problem so far mostly neglected in the IoT research".

Privacy concerns and regulations Privacy concerns and regulations may entirely prohibit the transmission of data to a central location. Or, privacy-preserving algorithms may transmit data, but not the original records. Further, network usage might be constrained by political or business regulations, such that data cannot be centralized. Other issues concern security and fail-safe operation. Centralized systems pose single points of failure. The more control is depending on data and its analysis, the more important it is to guarantee its delivery. In the cloud computing scenario, service provider and client may secure their end points, but usually have no control over the transmission of packets in between. A smart factory sending all its data into the cloud, depending on a timely analysis for real-time operation, might come to a complete standstill in case of a network failure. Even if the cloud is not available, continuous local operation should at least be possible.

In all of the aforementioned cases, data must be directly analyzed on the generating devices themselves and be reduced before transmission (see also [BYX10, GBA+13, Com15, QSF+16]). For instance, as shown in [LSFB15], the reduction of data before transmission with the help of autoregressive models reduced the energy consumption of smart sensors (MEMS) by factors up to 11. Similar reductions could be achieved with edge mining [GBA+13], whose authors argue purely in favor of local data preprocessing. However, local transformations and models may not suffice to capture dependencies between highly correlated measurements from different sensors. In such cases, decentralized algorithms are needed which build a global model based on messages exchanged between peer nodes or with a coordinator node. Such algorithms will necessarily need to be designed differently from distributed algorithms running in a data center. There, network technology allows for transfer rates resembling those of main memory accesses. Moreover, it may be freely decided how data is getting stored and partitioned across

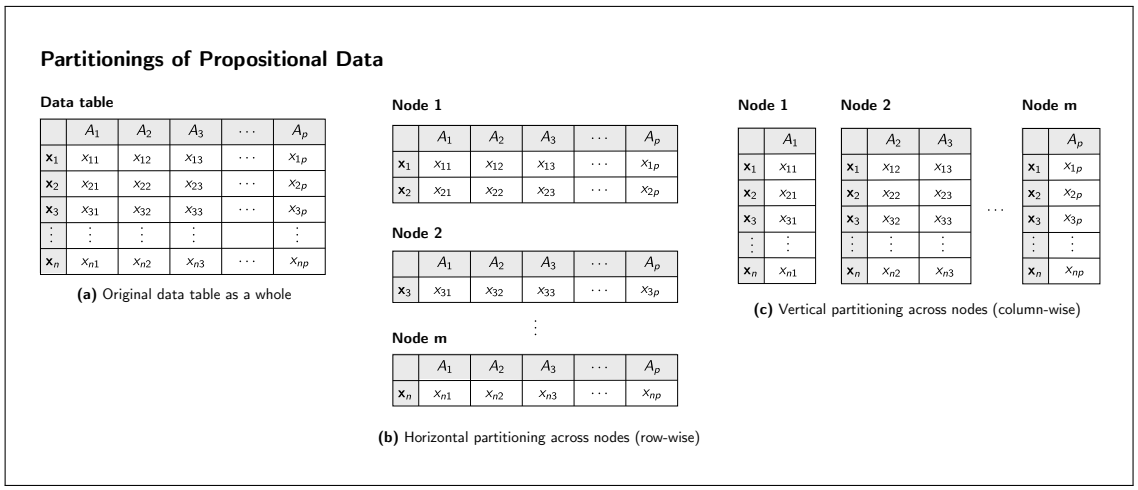


Figure 2.5: Common types of data partitioning

machines. New storage and compute nodes may be dynamically added to the network, based on demand. However, on the data generating side, the kind of data partitioning as well as the network structure are usually application dependent and given as fixed. Especially the type of data partitioning can have a large influence on learning and the amount of data that needs to be communicated, as shown in the following section.

2.4 Types of Data Partitioning

Data for learning is often given as a sample S of n observations, i.e. $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. For the following discussion, w.l.o.g. it is assumed that observations are represented in *propositional* form, i.e. described by a finite set of p different *features* $A_1, \ldots, A_p$ (also called *attributes*). Feature values are stored in columns of a data table, with one observation per row (see Fig. 2.5a). In distributed settings, data from this table may be spread across nodes in two different ways [CSH00b].

Horizontal partitioning In the *horizontally partitioned data* scenario (see Fig. 2.5b), data about observation, i.e. rows of the data table, are distributed across nodes $j = 1, \ldots, m$. All observations share the *same features*.

Horizontally partitioned sets of observations may be seen as skewed subsamples of a dataset that would result from centralizing and merging all observations. Hence, the distributed learning task consists of building a global model from such local samples, with as few communication between nodes as possible. Observations may be assumed to be independent and identically distributed, which for instance is exploited by learning algorithms that merge summary information independently derived from each subsample. In general, there exist many distributed learning algorithms for the scenario (for

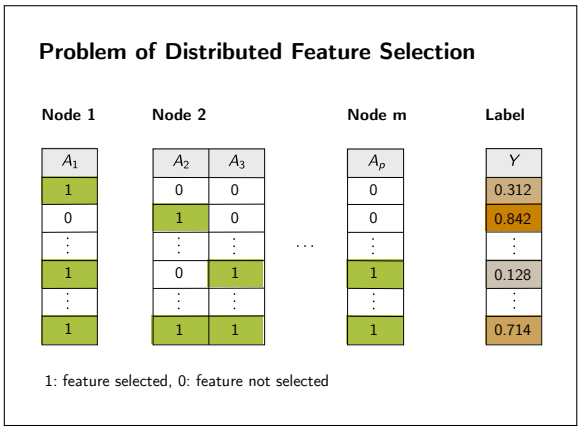


Figure 2.6: Which features provide most information about the target concept?

instance [DBK09, MK11, BGS+12, KBK+14]), though only few algorithms are truly suited for small devices (for a more detailed treatment, see [BS13, SBD15]). Communication costs for the scenario are well understood in the sense that bounds have been established for different classes of learning problems [BBFM12, ZDJW13]. For instance, [BBFM12] show that a distributed perceptron, which is a linear classifier, can find a consistent hypothesis in at most $O(k(1 + \alpha/\gamma^2))$ rounds of communication, k being the number of nodes, supposed that data is α-well-spread and all points have margin at least γ with the separating hyperplane.

An example task for learning in the horizontally partitioned data scenario is link quality prediction in wireless sensor networks (WSNs). We may assume that factors influencing link quality are the same across different wireless sensor nodes, i.e. recorded features provide information about the same underlying concept to be learned. However, the distributions of observations may differ for different parts of the network. For instance, in certain parts the link quality could be better than in other parts. The question is how to learn a global model which represents the distribution over all observations across nodes, without having to transfer all observations to a central node.

Vertical partitioning In the *vertically partitioned data* scenario (see Fig. 2.5c), feature values of observations, i.e. columns of the data table, are distributed across nodes $j = 1, \ldots, m$. Shared is only the index column, such that it is known which features belong to which observation. This might require a continuous tracking of objects, which in the IoT would be realized through globally unique identifiers for each entity.

The columns distributed over nodes constitute subspaces of the whole instance space. These subspaces and their individual components (e.g. features), in supervised learning including the target label, have a dependency structure that is usually unknown before learning. Learning in the scenario may thus be seen as a combinatorial problem of exponential size: Which subset of features provides the most information about the

target concept (see also Fig. 2.6)? In supervised learning, this is also known as the *feature selection* [SJ15] problem, whereas in unsupervised learning similar problems occur in *subspace clustering* [KKZ09]. Several techniques have been developed to tackle the exponential search space [KJ97]. Most of them are highly iterative and assume that features can be freely combined with each other. In a decentralized setting, however, such combination requires the costly transmission of column information between nodes in each iteration step and is thus prohibited. Hence, current approaches [DBV11, LSM12, SLM15] circumvent such problems by making explicit assumptions on the conditional joint dependencies of features, given the label.

In the context of the IoT, learning in the vertically partitioned data scenario is relevant and common. The problem occurs whenever a state or event is to be detected or predicted, based on feature values assessed at different nodes. What exactly constitutes a single observation then is application dependent. A common use case are spatiotemporal prediction models, which use measurements of devices at different locations. Measurements may be related to each other by the time interval in which they occur. The following list gives examples of applications:

- In manufacturing, one is interested in predicting the final product quality as early as possible [SMK+11, SBM16], based on process parameters and measurements at different production steps. Similarly, the optimization of process flow could benefit from a prediction of the time it takes to assemble a product, based on the current filling of queues and machine parameters at different locations on a shop floor. In both cases, a single observation consists of features, like sensor measurements and machine parameters, that are distributed and assessed at different locations. Depending on the granularity of control to be achieved, predictions must be either given after minutes, seconds or maybe also milliseconds. The more time-constrained the application, the more it might benefit from decentralized local processing.

- Products are assembled from parts delivered by different suppliers [WBX14]. Optimal planning and scheduling of assembly steps depend on a correct and continuous estimation of parts' delivery times. Those again are determined by production and transportation parameters of individual suppliers. For instance, the delivery of a particular part might be delayed due to the maintenance of a single production unit at one supplier. Assembly time of a product is thus a global function depending on local information (features) from different suppliers, i.e. observations for learning this function are vertically partitioned. Even if it was technically feasible to centralize the raw production and transportation data from all suppliers for analysis, it would be unnecessary if the global function depended only on a few local features. Moreover, due to privacy concerns, it is unrealistic that suppliers would provide raw data about their processes. Hence, a decentralized algorithm is needed that derives a global model from local data, at the same time preserving privacy.

- The smart grid requires a continuous prediction of energy demand [MK11, KMS+16], based on local information about energy usage at different smart homes [ZY16]. Here, observations might represent the whole state of the energy grid, and consist of vertically partitioned features at different locations describing local states. Instead of centralizing raw meter readings from ten thousands of households, communication could be spared by an aggregation of local data or a combination of predictions from locally trained models.

- Centralized traffic management systems analyze traffic based on data from a hard-wired mesh of distributed presence sensors [SCA13]. While easy to design, centralized systems pose a single point of failure in case of an emergency. With the addition of new sensors, they may become a bottleneck, due to limited bandwidth. Further, the maintenance of hard-wired sensors can be expensive in case of failure, due to required construction work. A more decentralized system could consist of cheap wireless sensors. Those may be attached to existing infrastructure, like traffic lights, signs and street lights. Traffic lights may then adjust themselves, based on the prediction of traffic flow at neighboring junctions. The flow measurements at each individual junction can be interpreted as vertically partitioned features of a single observation describing the current state of all sensors. The learning task is to derive prediction models from these distributed flow measurements, without transmission of all data to a central server [SLM15].

- In healthcare, diagnoses of illnesses depend on many factors, like a patient's health care records, parents' illnesses and current health parameters such as pulse, blood pressure, measurements from a blood sample, an electroencephalogram or other specialized information. With IoT technology, even more data becomes available through fitness trackers or dieting apps (see also Sect. 2.1.5). The features describing a single patient are thus distributed over different locations, like several physicians, medical centers, and now even devices or social websites. The centralization of all data poses a threat to patients' privacy. Hence, the learning task is to derive a global model for diagnosis from local data, without transmission of raw data between locations. The features of diagnoses from different geographical locations over certain time intervals could then be combined to predict, for instance, epidemics and their spread at a larger scale (see also [MZDM16]). Again, the features from different locations over the same time intervals constitute vertically partitioned observations.

2.5 Research Questions

The number of communication-efficient distributed data analysis methods for the vertically partitioned is much smaller than those for horizontally partitioned data. There are many open research questions, which mainly concern the relationship between accuracy and communication costs. Therefore, we first define how communication costs

are measured and what it means for an algorithm to be communication-efficient. Then, an overview of typical components that vertically distributed algorithms may consist of is given. It is shown that the schema is general enough to cover common designs of distributed algorithms. Finally, open issues and research questions are formulated that concern communication-efficient learning.

2.5.1 Communication Costs and Efficiency

In most publications on distributed data analysis, *communication costs* are the total payload transmitted measured in bits, i.e. excluding meta data, like packet headers. The authors of [GBA+13] argue for a measurement of communication costs by the number of transmitted packets. Although the number of packets in certain cases might be a more exact measure than the payload in bits, it is highly dependent on chosen network protocols and the underlying network technology. Similar to measuring the run-time of algorithms in seconds, it would make the comparison of results from different publications very difficult. A fair comparison would require building the exact same network with the same hardware and configuration. A solution could be network simulators, however, there doesn't seem to exist a commonly agreed on standard between different scientific communities. At least for batch transmissions of data, the number of packets to be sent is proportional to the payload in bits. From there, we follow the argumentation in [GBA+13] that a reduction of packets may reduce congestion and collisions on networks with large amounts of traffic. This in turn reduces the number of acknowledgements and retransmissions, which should enable better use of available bandwidth (i.e. higher transmission rates or more network participants).

Central analysis requires the transmission of all data (or at least all preprocessed data) to the coordinator node. We define a learning method to be *communication-efficient* if less data than the whole dataset (optionally after local preprocessing) is exchanged between local nodes and an optional coordinator node. Method A is called *more communication-efficient* than method B, if A is communication-efficient and its communication costs are less than those of B.

The amount of data communicated per observation during learning may differ from the amount communicated when making an actual prediction. It should be noted that in the vertically partitioned data scenario, at least *some* data must be communicated for detecting a global state or predicting a global event. Further, the supervised learning of local models may require the transmission of label information from a coordinator. This is different from a horizontal partitioning of data, where each local node contains all the necessary information (i.e. feature values and often also the label).

2.5.2 Distributed Setting and Components

Figure 2.7 gives an overview of the setting in the vertically partitioned data scenario and the distributed components that algorithms may be designed of. Given are $m + 1$ networked nodes $j = 0, \dots m$, where nodes $1, \dots, m$ are called *local nodes* and $j = 0$ de-

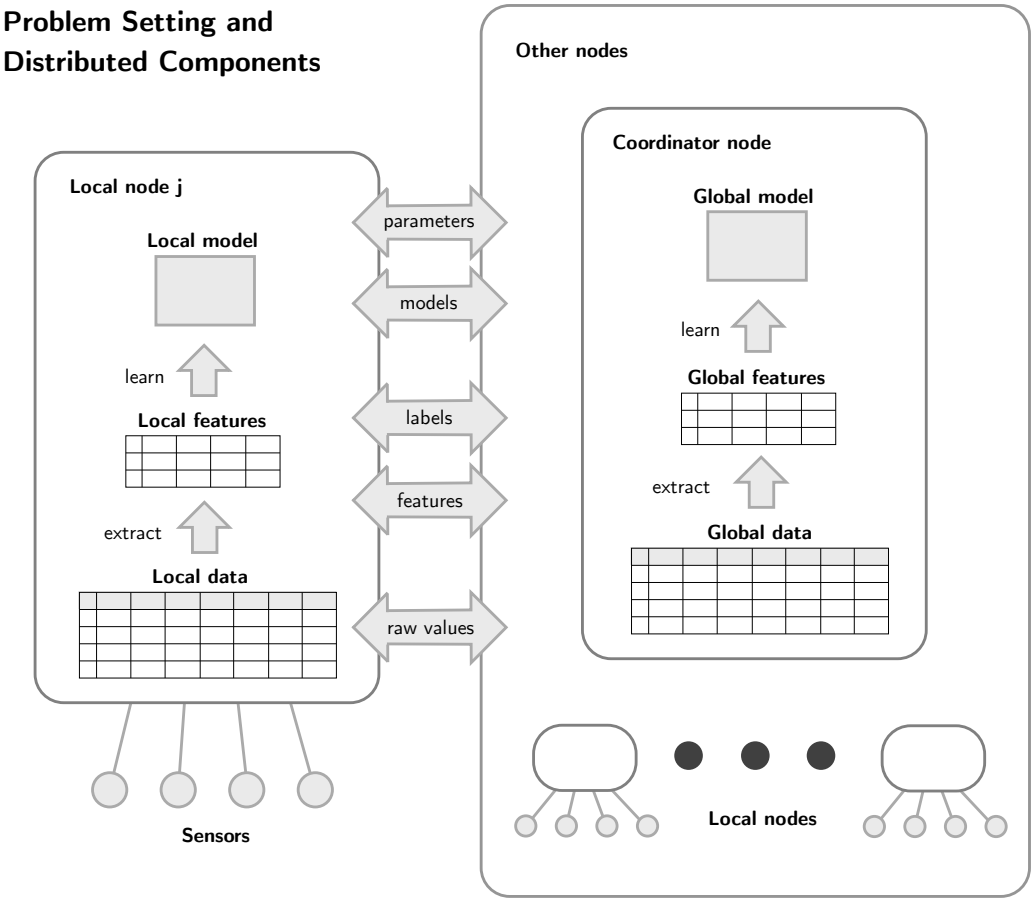


Figure 2.7: Problem setting and distributed components

notes a *coordinator node*. No assumptions are made on network topology or technology. Further, "local" and "coordinator" are to be understood as *roles* that physical nodes can have, and may change depending on context.

Each local node acquires raw values, like sensor measurements. Those may be locally preprocessed and transformed into features for learning. It is assumed that the features of the same observations are vertically partitioned across the local nodes. Distributed components of learning algorithms may do further local calculations on such features, and might build or update local models. Once or iteratively, depending on algorithm, local nodes will either transmit raw values, features, models or predictions of such models to other nodes, which in turn may preprocess the received data, and do further calculations on them, like build or update a global model, fuse predictions, etc. The setting as described is general enough to cover the following common approaches for designing

distributed algorithms:

Central analysis Each local node transmits all of its raw values to a coordinator node for further analysis. This may include the stages of preprocessing, feature extraction and model building. This is in principle what cloud-based data processing proposes [GBMP13, Bur14, CDW+15, Ora15b, DMR16]: While the coordinator may consist itself of distributed components and solve the analysis problem in parallel, from the perspective of local nodes it looks like a single machine where all data is getting centralized. The design is not decentralized, as data and processing aren't split between local nodes and coordinator node, but all processing is done at the coordinator node.

Local preprocessing, central analysis Local nodes preprocess raw values and transform them into a representation for learning. The representations are sent to a coordinator, which builds a global model based on them. While according to our former definition, this design is decentralized, its form is very rudimentary, as most of the processing is still done at the coordinator. Depending on the processing capabilities of local nodes and the particular learning task, such a design might be the only viable option. The design fits ideas mentioned in [GBA+13, TLV14, QSF+16], whose authors' propose to reduce data locally before sending it to the cloud for analysis. Privacy-preserving Support Vector Machine (SVM) algorithms like [MWF08, YLG09] also follow this design, but are not necessarily communication-efficient.

Model consensus Local nodes iteratively try to reach consensus on a set of parameters among each other (peer-to-peer), or on a set of parameters they share with a coordinator node. At the end, each local node (or only the coordinator) has a global model. As [BPC+11] demonstrate, many existing analysis problems can be cast into a consensus problem and then be solved, for instance, with the Alternating Direction Method of Multipliers (ADMM). Algorithms of this sort are working fully decentralized, but are working iteratively and may transmit more than the original data, depending on their convergence properties.

Fusion of local models Each local node preprocesses its own data and builds a local model on it. Such models are then transmitted to a coordinator node or to peer nodes, which fuse them to a global model. These algorithms are working decentralized, as data and the load of processing are shared among all nodes. In the vertically partitioned data scenario, using a global model usually requires the transmission of feature values of observations whenever a prediction is to be made. An example algorithm would be [HMM16].

Fusion of local predictions Each local node preprocesses its own data and builds a local model on it. Whenever a prediction is to be made, only the predictions are transmitted from local nodes, and fused at the coordinator or other peer nodes

according to a fusion rule. This could be, for instance, a majority vote over predictions. Local nodes each transmit only one value during prediction, but a fusion rule may not be as accurate as a global model, depending on data distribution and learning task. Examples would be [LSM12, SLM15].

While aforementioned approaches are common, there exist hybrids also covered by the setting shown in Fig. 2.7. For instance, in [DBV11] local models are used to detect local outliers, which are then checked against a global model that was derived from a small sample of all data.

According to our previous definition, the examples of distributed algorithms given above all learn from vertically partitioned data in a decentralized fashion. However, not all are communication-efficient. Apart from the two mentioned privacy-preserving SVMs which might send more data than the whole dataset, the model consensus based algorithms may send more data as well, depending on the number of iterations during optimization. The design of communication-efficient decentralized algorithms in the vertically partitioned data scenario leaves many open research questions of which some are presented in the following.

2.5.3 Open Questions

Despite first successes in the development of communication-efficient algorithms for the vertically partitioned data scenario, there are still many open research questions left:

- Data analysis knows many different kinds of tasks, like dimensionality reduction, classification, regression, clustering, outlier detection or frequent itemset mining. How does the task influence communication costs when the data is vertically partitioned? And how does the design change with the task?
- As first results suggest, the accuracy of communication-efficient algorithms in the vertically partitioned data scenario very much depends on the data being analyzed. What influence have different data distributions on the communication costs and accuracy of algorithms? How is the design of algorithms affected?
- What are bounds on communication, i.e. how much information must be at least and at most communicated to learn successfully from vertically partitioned data? What trade-off has to be made between communication and accuracy?
- How can the supervised learning of local models be made more communication-efficient in cases where labels do not reside on the local nodes, but must first be transmitted to them? For instance, how can we learn from aggregated label information?
- Many existing data analysis algorithms can easily work on different numbers of observations, but expect the number of features to be fixed. How can algorithms that

work on observations with features from different sensors deal with the dynamic addition and removal of sensors, i.e. features?

Beyond those questions, there are open issues concerning distributed data analysis algorithms in general, i.e. also those that work on horizontally partitioned data or in the cloud. For instance, methods for feature selection, the optimization of hyper parameters and validation are highly iterative and work on different subsets of features and observations in each iteration. How can we adapt these algorithms in such a way that the same data isn't repeatedly sent over the network or read from external storage? As the previous questions demonstrate, there is still a lot of research to do before data analysis and the IoT will become seamlessly integrated.

2.6 Summary

After a short introduction to the IoT, it was argued for data analysis being an essential part of it. By giving examples from different sectors, it was shown that already remote monitoring applications may benefit from a summarization of data with the help of data analysis. Complex applications require more advanced and autonomous control mechanisms. These in turn depend on advanced data analysis methods, like those that can analyze data in real-time, adapt to changing concepts and representations and test hypotheses actively. Beyond security, privacy and technical problems, especially algorithmic challenges need to be tackled before such advanced applications will become a reality.

Distributed cloud-based algorithms follow the paradigm of parallel high performance computing. The cloud might seem like the most convenient and powerful solution for the analysis of IoT generated big data, which is expected to have large volume, high velocity and high heterogeneity. However, without substantial advances in network technology, bandwidth will become more and more scarce with each new device getting connected. The transmission of all data into the cloud can already be infeasible, due to limited energy, bandwidth, high latency or due to privacy concerns and regulations. Communication-constrained applications require decentralized analysis algorithms which at least partly work directly on the devices generating the data, like sensors and embedded devices. A particularly challenging scenario is that of vertically partitioned data, which covers common IoT use cases, but for which not many data analysis algorithms exist so far. The main research question is how to design communication-efficient decentralized algorithms for the scenario, while at the same time preserving the accuracy of their centralized counterparts.

Chapter 3

Basic Principles and Methods

This chapter introduces basic notations, principles and methods which are needed for a better understanding of the rest of this thesis. The first sections explain important terms and principles from the fields of machine learning and data mining. Among them are a more formal view on learning, how learning algorithms should be evaluated, how to construct or choose a good model or hypothesis, and how many observations are needed for learning. Further, the relationship between prediction error and model complexity is elaborated upon. The CRISP-DM model provides an overview of the process of data mining as a whole.

Then, in the sections following, supervised methods are described which derive prediction models from a set of observations whose target values are provided by a teacher. The problem of outlier and anomaly detection, together with according methods, is introduced next. Afterwards, we have a look at the unsupervised problems and methods of cluster analysis and dimensionality reduction. The discussion will be restricted to methods either used for the analysis of datasets in experimental sections of this thesis, or modified in algorithmic sections. In addition, whenever meaningful, it is explained how concepts and methods relate to earlier introduced problems of data analysis in an IoT context and the vertically partitioned data scenario.

3.1 Machine Learning and Data Mining

According to Tom M. Mitchell, "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E" [Mit97].

The kinds of tasks in machine learning can be broadly divided into the three categories of *supervised learning*, *unsupervised learning* and *reinforcement learning* [RN13]. In this thesis, we focus only on the first two kinds:

Supervised learning The goal of *supervised learning* is to learn a rule that maps inputs to outputs, based on example inputs and their desired outputs (which are

also called *labels*). The labels are assumed to be given by an oracle or teacher. In practice, this means they come either from domain experts, or that they are automatically generated. If labels are discrete, one also speaks of a *classification* task. If labels are numeric, one speaks of *regression*. The performance of supervised learning algorithms is usually measured by the deviation of predicted output value and the given label. The deviation (or *error*) should be minimized. For a more precise characterization of error, see Sect. 3.1.2.

Unsupervised learning The task of *unsupervised learning* is to find some structure in the given inputs, without getting any direct help (e.g. labels) from a teacher. A performance measure is needed which implicitly tells a learning algorithm what kind of structure is to be found and how well the algorithm has solved the task. The final evaluation, i.e. if patterns found are meaningful, is usually done by domain experts.

There are hybrid learning tasks that are neither purely supervised nor unsupervised. For instance, in the case of outlier and anomaly detection, it might be known that a subset of the given input examples belongs to a certain normal class, while the remaining examples are unlabeled and could be anomalies or not. In semi-supervised learning, the label is only known for a few of the given input examples. A novel kind of learning task is that of learning from label proportions, where only statistical information about the labels for groups of example inputs is given (see Sect. 6).

3.1.1 Instances, Concepts and Labeled Examples

The space X of possible inputs is called the *instance space*. A single element $x \in X$ will be called an *instance* or *example*, independent of learning task (supervised or unsupervised). For example, X might represent the set of all people, described by attributes such as age, height and weight. A single instance then is a person with a specific age, height and weight. It is assumed that the instances in X are generated according to an underlying probability distribution D.

Let $c \in C$ be a *concept* from a set C of possible concepts over X. Each c is a subset of X, and can be thought of as a boolean-valued function $c : X \to \{0, 1\}$. If x is a *positive example* of c, $c(x) = 1$, and if x is a *negative example*, $c(x) = 0$. For example, one concept might be the subset of "old people", whereas another concept might be the subset of "overweight people".

We can now specify the task of *supervised learning* more formally. The task is to learn a *target concept* $c \in C$, based on a *sample* sequence $S = \langle (x_i, y_i) \rangle$, $i = 1, \ldots, n$, where each element x_i is an element of the instance space, drawn according to D, and $y_i = c(x_i)$ is a *label* which indicates if x_i belongs to the target concept or not. A single pair of instance and label is called a *labeled example*, and the label is assumed to be provided by an oracle or teacher. Instances are assumed to be drawn *independent and identically distributed* (i.i.d.), i.e. that they are all drawn from the same distribution D and mutually independent. For instance, when observations are generated by an

automatic process, it must hold that the generation of previous observations has no influence on the generation of subsequent observations.

In unsupervised learning, the concept to be learned is unknown. Especially, no labels are given. Intrinsic performance measures rather describe the structure of patterns that might be interesting. For instance, unsupervised clustering algorithms as described in Sect. 3.4 group examples based on their similarity. Depending on the similarity measure used, a clustering algorithm may either detect the concept of "old people", "overweight people" or an entirely different concept which doesn't necessarily need to be interesting. If the detected concepts are meaningful or not can usually only be decided by domain experts, which makes unsupervised learning much more explorative than supervised learning.

The following discussion of how to evaluate the performance of learning algorithms and how to construct good hypotheses will mainly focus on binary, and later on non-binary supervised learning tasks. The specifics of evaluating unsupervised learning algorithms are discussed in the according sections on outlier and anomaly detection, clustering and dimensionality reduction.

3.1.2 Training and Test Error

During *training*, a learner L may choose from a set of hypotheses H for learning the target concept c from S. After training, L outputs one $h \in H$, which is an estimate of c. A hypothesis h is also called a *model* for c. We want to maximize the performance of h. This means that h should approximate the target concept c as good as possible. This is equivalent to minimizing the *true error* of h, which is the expected error rate of h on arbitrary instances drawn at random from X, according to the underlying probability distribution D. More formally [Mit97]:

Definition 3.1.1 (True error) The *true error* $err_D(h)$ of hypothesis h with respect to target concept $c \in C$ and distribution D is the probability P that h will misclassify an instance x drawn at random according to D:

$$err_D(h) := P_{x \in D}[c(x) \neq h(x)] \tag{3.1}$$

The challenge of supervised learning is that the probability distribution D is unknown, and that h must be derived from a finite sample S of labeled examples. In contrast, the true error is measured over the whole instance space X, which can be even infinite. Without knowing D, the true error cannot be calculated directly, but only be estimated from the performance of h on examples in the given sample S [Mit97]:

Definition 3.1.2 (Sample error) The *sample error* $err_S(h)$, i.e. the error of hypothesis h on *all* instances in sample S, is the fraction of misclassified examples in S:

$$err_S(h) := \sum_{x_i \in S} [y_i \neq h(x_i)] \tag{3.2}$$

It can be shown that the sample error (also called the *training error*) is usually a bad estimate of the true error [Mit97, HTF09]. Imagine an algorithm that simply memorizes the mapping between inputs and outputs in S by rote learning. It would assign the correct label to each $x \in S$, that is, it would achieve zero sample error. However, on new instances which differ only slightly from the instances in S, it couldn't perform any better than random guessing. Rote learning or random guessing are usually not what we mean when speaking of learning. What we want is that a learner L derives a hypothesis $h \in H$ that generalizes well over the instances in sample S, i.e. that performs well also on previously unseen instances.

A much better estimate of the true error can be achieved by evaluating hypothesis $h \in H$ on an independent *test set*. Under the assumption that all we have are the labeled examples in S, a test set can be randomly sampled from S. Learner L can then be trained on the remaining examples, which is also called the *training set*, and the performance of the resulting hypothesis h can be evaluated on the test set. The larger the test set, the more accurate the estimate will be. For small samples S, there exist special techniques, like cross validation (see Sect. 3.1.7), which make most economic use of S for training and testing.

It should be noted that if we would know distribution D, we could construct an optimal classifier, called the *Bayes classifier*. It assigns that class to an object which is most probable. This decision rule is also known as the *maximum-a-posteriori* (MAP) criterion and can be shown to be optimal for cost measures that create costs only in the case of misclassifications, like the error measures defined above. The notion of an optimal classifier is important in so far as no other classifier can achieve better prediction performance, and even the optimal classifier cannot achieve zero true error, depending on D. As such, it can be used as a baseline in cases where D is known. For instance, imagine data that is generated by two different gaussian distributions. Data generated by these distributions will necessarily overlap, and is not separable. We therefore cannot expect the Bayes classifier or any other classifier to achieve zero error rate over arbitrary independent test samples.

3.1.3 PAC Learnability

So far, it has been discussed how to measure the performance of learning algorithms, but not how to choose or construct a good hypothesis. For instance, if there were any heuristics or even proven methods for devising a good hypothesis, maybe we could spare ourselves the difficult empirical estimation of the true error. Up to now, it isn't even clear what kinds of concepts can be learned at all. Further, as indicated by Mitchell's definition of learning, we'd expect a learning algorithm to perform better with more experience, i.e. more training examples. However, it is unclear how many examples are necessary and sufficient for learning. A field that tries to answer these kinds of questions is *computational learning theory*. It has introduced the notion of *PAC learnability*, which will be shortly described in the following.

The treatment of PAC learnability in this thesis is central to understand the relationship between instance, model and sample complexity, as well as the bias variance trade-off explained in Sect. 3.1.6. PAC learnability also provides a theoretical foundation and motivation for the support vector machine (SVM) described in Sect. 3.2.5, whose 1-class variant is adapted for distributed learning in Chap. 8. The SVM is based on the structural risk minimization principle, which in turn is closely related to a measure of complexity, the Vapnik-Chervonenkis (VC) dimension, described in this section. Chapter 6 discusses a theoretical work on learning from label proportions which has proven bounds based on the concept of PAC learnability. The following presentation bases on [Mit97].

PAC Learnability PAC learnability requires the hypothesis output by a learning algorithm to be *probably approximately correct*. There is a nonzero probability that a randomly drawn sample doesn't represent the overall distribution D well. *Probably correct* means that we allow a learner to fail on this kind of misleading samples, with a probability bounded by some constant δ. In exchange for more training examples, this constant can be made arbitrarily small. We further must allow a learner to be only approximately correct, because there may be multiple hypotheses leading to zero sample error. We therefore only require that the true error of a learner L is bounded by some constant ε, which can be made arbitrarily small.

Definition 3.1.3 (PAC learnable) Consider a concept class C defined over a set of instances X of length p and a learner L using hypothesis space H. C is *PAC-learnable* by L using H if for all $c \in C$, distributions D over X, ε such that $0 < \varepsilon < 1/2$, and δ such that $0 < \delta < 1/2$, learner L will with probability at least $(1 - \delta)$ output a hypothesis $h \in H$ such that $err_D(h) \leq \varepsilon$, in time that is polynomial in $1/\varepsilon$, $1/\delta$, p and $size(c)$.

The definition points to two quantities that weren't discussed so far, namely the size p of instances in X and the encoding length $size(c)$ of $c \in C$. They relate to the fact that instances, as well as concepts and hypotheses, are represented by words from different languages.

Representation and Hypothesis Language Instances can be thought of as being represented by words from a so called *representation language* L_X, whereas concepts and hypotheses are represented by words from a *hypthesis language* L_H. For example, people described by attributes such as age, height and weight could be represented by three-dimensional real-valued vectors, i.e. $X = \mathbb{R}^3$. Here, the alphabet Σ_X might consist of digits, a character for the decimal point and a separator character for denoting vector components. Hypotheses could be, for instance, axis-parallel hyper rectangles, represented as pairs of coordinates describing two opposite corners, whose description is based on a corresponding alphabet Σ_H. Although L_X and L_H are not the same

languages, they are necessarily related. That is, it must be possible to map concepts and hypotheses represented by words from L_H to subsets of the instance space X.

The quantity p measures the length of words from L_X. For instance, in the example above, p could be chosen as the number of vector components, i.e. $p = 3$. The inherent complexity of concepts and hypotheses is measured by $size(c)$. For instance, pairs of corners, each represented by three-dimensional vectors, would have $size(c) = 6$.

Efficient PAC learnability requires learning algorithms to finish in polynomial time. This means that the time to process words from L_X and L_H must be polynomial in their description length. A further requirement is that instances $x \in X$ can be mapped to concepts $c \in C$ in polynomial time. That the time for learning is polynomial in $1/\varepsilon$ and $1/\delta$ indirectly means that the sample complexity $n(\varepsilon, \delta)$, i.e. the number of instances required for learning, which depends on ε and δ, must be polynomial, too.

Actually, to show that a class C of target concepts is PAC-learnable, one first proves that each target concept in C can be learned from a polynomial number of training examples and then demonstrates that the processing time per instance is bounded by a polynomial.

Sample Complexity of Consistent Learners A learner is consistent if it outputs hypotheses that perfectly fit the training examples in S, whenever possible. A general bound can be proven for the number of training examples that are *sufficient* to consistently learn *any* target concept in H [Mit97]:

$$n(\varepsilon, \delta) \geq \frac{1}{\varepsilon}(\ln |H| + \ln(1/\delta)) \tag{3.3}$$

Here, $n(\varepsilon, \delta)$ grows linearly in $1/\varepsilon$ and logarithmically in $1/\delta$. It also grows logarithmically in $|H|$.

The bound usually overestimates the number of training examples required for learning. However, it points to an important general relationship between sample complexity and the size of the hypothesis space: The more hypotheses a learner can choose from, the more training examples are needed to guarantee that a hypothesis is probably approximately correct. Intuitively, if the sample size is staying constant, a larger hypotheses space means that more hypotheses are consistent with the given training examples. Therefore, the probability to choose a wrong hypothesis regarding the true error increases. In turn, more training examples are needed to falsify wrong hypotheses.

Hypothesis Space Complexity and VC Dimension Another notion is that the larger the space of different hypotheses, the more complex they can be. More complex hypotheses allow for a more fine granular adaptation to the training examples, but can lead to *overfitting*. This means they may adapt to patterns which are sample specific and do not generalize over the whole instance space. In other words, more complex hypotheses allow for the differentiation of ever smaller subsets of X. This idea has lead to a measure for the complexity of hypotheses in H, the Vapnik-Chervonenkis dimension (VC dimension, or $VC(H)$, for short).

Let $T \subseteq H$ be a subset of instances. Each hypothesis $h \in H$ partitions T into the two subsets $\{x \in T | h(x) = 1\}$ and $\{x \in S | h(x) = 0\}$.

Definition 3.1.4 (Shattering) We say that H *shatters* T if every possible partitioning of S into two subsets can be represented by some hypothesis from H.

There are $2^{|T|}$ possible partitionings, but H may only represent some of these.

Definition 3.1.5 (VC dimension) The Vapnik-Chervonenkis dimension, or *VC dimension*, $VC(H)$, of a hypothesis space H defined over instance space X is the size of the largest finite subset of X shattered by H. If arbitrarily large finite sets of X can be shattered by H, then $VC(H) := \infty$.

In other words, the VC dimension measures the number of distinct instances from X that can be discriminated using H.

Assume that $VC(H) = d$. To shatter a set of d instances, H must contain 2^d distinct hypotheses. Hence, $2^d \leq |H|$, and $d = VC(H) \leq \log_2 |H|$. Therefore, whenever $|H|$ is finite, $VC(H) \leq \log_2 |H|$. This observation usually allows for tighter bounds on sample complexity (see below). Apart from that, VC dimension also allows for a characterization of sample complexity given infinite hypothesis spaces, in contrast to the general bound of (3.3).

Sample Complexity and VC Dimension Now that the complexity of H can be measured by $VC(H)$, tighter bounds on the number of samples required for learning can be derived. The first bound is an upper bound, specifying the number of training examples that are *sufficient* to learn any target concept in C probably approximately correct, for any ε and δ [Mit97]:

$$n(\varepsilon, \delta) \geq \frac{1}{\varepsilon} \left(4 \log_2(2/\delta) + 8 VC(H) \log_2(13/\varepsilon)\right) \tag{3.4}$$

The new bound grows log times linear in $1/\varepsilon$, instead of linearly. More important, the $\ln |H|$ term has been replaced by $VC(H)$, where $VC(H) \leq \log_2 |H|$), as shown earlier.

It is also possible to obtain a lower bound [Mit97] on the number of training examples that are *necessary* for learning. For any concept class C with $VC(C) \geq 2$, any learner L, and any $0 < \varepsilon < \frac{1}{8}$, and $0 < \delta < \frac{1}{100}$, there exists a distribution D and target concept in C such that if L observes fewer examples than

$$\max \left[\frac{1}{\varepsilon} \log(1/\delta), \frac{VC(C) - 1}{32\varepsilon} \right], \tag{3.5}$$

then with probability at least δ, L outputs a hypothesis h with $err_D(h) > \varepsilon$. In other words, no learner can PAC-learn every target concept in any C with fewer training examples.

Though it can be difficult to derive the VC dimension of arbitrary concept classes, it could be derived successfully for important classes like separating hyperplanes. It plays an important role in choosing hypotheses according to the principle of structural risk

minimization (see Sect. 3.1.5), which is closely related to the support vector method (SVM) explained in Sect. 3.2.5.

3.1.4 Supervised Function Learning

Up to this point, the concepts to be learned were subsets of the instance space X, which can be thought of as a boolean-valued function $c : X \to \{0, 1\}$. The corresponding hypotheses were defined similarly as $h : X \to \{0, 1\}$, indicating if a particular instance $x \in X$ belongs to the target concept to be learned or not. The decision was thus binary.

In the following, we'll move from binary indicator functions to the supervised learning of general functions $f : X \to Y$, where Y may be either a set $Y = \{Y_1, \ldots, Y_l\}$ of discrete labels (classification), or a real value, i.e. $Y \subseteq \mathbb{R}$ (regression). Function $f(x)$ will denote the true function to be learned, while $\hat{f}(x)$ denotes the function estimated from the data.

Definition 3.1.6 (Supervised function learning) The task of *supervised function learning* aims at deriving a function $\hat{f} : X \to Y$ from a sample $S = \langle (x_i, y_i) \rangle_{i=1,\ldots,n}$ of n labeled examples $(x_i, y_i) \in X \times Y$, drawn i.i.d. from an unknown joint probability distribution $P(X, Y)$, such that the expected risk

$$R_{\text{exp}} = \int \ell(y, \hat{f}(x)) dP(x, y)$$

is minimized. Here, ℓ is a convex *loss function* $\ell : Y \times Y \to \mathbb{R}_0^+$ which measures the cost of assigning the wrong label to individual observations.

For classification, a common loss function is the *0-1 loss*.

Definition 3.1.7 (0-1 loss) Given a function $\hat{f}$ and discrete label space Y, the *0-1 loss* ℓ_{01} counts misclassified examples as one and correctly classified examples as zero:

$$\ell_{01}(y, \hat{f}(x)) = \begin{cases} 1 & \text{if } \hat{f}(x) \neq y \\ 0 & \text{if } \hat{f}(x) = y \end{cases} \tag{3.6}$$

For the 0-1 loss, the expected risk is the same as the true error $err_D(h)$ of a hypothesis $h \in H$ (see Def. 3.1.1).

3.1.5 Empirical vs. Structural Risk Minimization

Like the true error, the expected risk cannot be calculated explicitly, since the joint distribution of examples and labels is unknown. What can be directly calculated is the empirical risk

$$R_{\text{emp}} = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \hat{f}(x_i)), \ (x_i, y_i) \in S$$

which measures the loss of function $\hat{f}$ on the examples in sample S. The empirical risk is thus analogous to the sample error (see Def. 3.1.2), and, for classification with 0-1 loss, it is the same.

Intuitively, given two functions with the same empirical risk R_{emp}, the less complex function should generalize better and thus overfit less likely. This idea, which is in accordance with the bounds presented in Sect. 3.1.3, gives rise to the definition of structural risk.

Definition 3.1.8 (Structural risk) For a function class $\hat{f}(x, \boldsymbol{\gamma})$ with parameter vector $\boldsymbol{\gamma}$, the *structural risk* (also called *regularized risk*) is defined as

$$R_{\text{reg}}(\boldsymbol{\gamma}) = R_{\text{emp}}(\boldsymbol{\gamma}) + \lambda\Omega(\boldsymbol{\gamma}) \ ,$$

where Ω is a strictly monotonic increasing function which measures the *capacity* of function class $\hat{f}(x, \boldsymbol{\gamma})$ depending on parameter vector $\boldsymbol{\gamma}$. The trade-off between the empirical training error and the capacity is managed by λ.

The capacity can, for example, be measured by the VC dimension (see Def. 3.1.5), which yields a probabilistic bound for the regularized risk. It can be shown that after having seen n examples, the structural risk is upper bounded with probability $1 - \mu$, where η is the capacity [Vap95]:

$$R_{\text{reg}}(\boldsymbol{\gamma}) \leq R_{\text{emp}}(\boldsymbol{\gamma}) + \sqrt{\frac{\eta\left(\log\left(\frac{2n}{\eta}\right) + 1\right) - \log\left(\frac{\mu}{4}\right)}{n}} \tag{3.7}$$

The bound explains much better why, and in which cases, minimizing the empirical risk does not suffice, i.e. why the sample error can be a bad estimate for the true error (see also Sect. 3.1.2). The VC bound is getting smaller and smaller with growing sample size, if we assume the capacity of the function class to be constant. In other words, if enough observations are given or capacity is low, the sample error is a good estimate of the true error. Intuitively, one can say that with a limited number of hypotheses, it becomes more difficult to match the increasing variance of a large number of observations. Therefore, if n/η is large, one may follow the principle of *empirical risk minimization* (ERM) [Vap95]. However, if a learner is allowed to choose arbitrarily complex hypotheses, it can always adapt to such variability and match the given training examples with zero sample error. Therefore, whenever only a small number of observations is given, we need a way to control the capacity of the function class.

The idea behind the principle of *structural risk minimization* (SRM) [Vap95] is to fit hypotheses from function classes of increasing capacity, $\eta_1 < \eta_2 < \ldots$, and to choose the hypothesis as optimal which minimizes the structural risk $R_{\text{reg}}(\boldsymbol{\gamma})$, i.e. the empirical risk *and* the VC bound. Derivation of the VC dimension can be difficult, depending on function class. Nevertheless, the VC dimension has been derived for some function classes, of which an important one is the class of separating hyperplanes. As we will see, the support vector machine (SVM) described in Sect. 3.2.5 follows the principle of

structural risk minimization, controlling capacity by maximizing the margin between observations lying on two sides of a separating hyperplane, at the same time minimizing the number of observations lying in the wrong half space.

3.1.6 Bias and Variance

In the last section, we formulated the intuition that given two functions with the same empirical risk R_{emp}, the less complex function should generalize better and thus overfit less likely. This intuition is backed up by the proven bounds of computational learning theory discussed in Sect. 3.1.3, which state that all else staying the same, a larger or more complex hypothesis space will lead to higher error probability. In the field of *statistical learning theory* [HTF09], choosing a function which is complex enough to reflect the target concepts to be learned, but does not overfit the training data, is known as the problem of trading off bias against variance.

Let $\mathbb{E}$ denote statistical expectation and Var denote variance. Given a function $y = f(x) + \epsilon$, where $\mathbb{E}(\epsilon) = 0$ and $\text{Var}(\epsilon) = \sigma_\epsilon^2$, the expected prediction error of a regression fit $\hat{f}(X)$ at input point $x = x_0$ can be expressed as follows when using squared-error loss [HTF09]:

$$
\begin{aligned}
err_D(x_0) &= \mathbb{E}[(Y - \hat{f}(x_0))^2 | X = x_0] && (3.8)\\
&= \sigma_\epsilon^2 + [\mathbb{E}\hat{f}(x_0) - f(x_0)]^2 + \mathbb{E}[\hat{f}(x_0) - \mathbb{E}\hat{f}(x_0)]^2 && (3.9)\\
&= \sigma_\epsilon^2 + \text{Bias}^2(\hat{f}(x_0)) + \text{Var}(\hat{f}(x_0)) && (3.10)\\
&= \text{Irreducible Error} + \text{Bias}^2 + \text{Variance} && (3.11)
\end{aligned}
$$

The irreducible error cannot be controlled. The bias term is the squared difference between the true mean $f(x_0)$ and the expected mean estimate. The expectation averages over the randomness in the training data. The variance term describes the expected variance, i.e. noise, of our predictions.

The more we adapt to the training data, the more we tend to increase model complexity. Adapting too much may thus lead to overfitting, i.e. the model will not generalize well. The variance of our predictions will increase, i.e. the true error will be larger. As we decrease model complexity, the squared bias tends to increase, and variance tends to decrease. However, if the model is not complex enough to cover the target concept, it will *underfit* and may have large bias. The goal of learning is thus to trade off bias with variance, i.e. to choose model complexity, such that the true error is minimized.

3.1.7 Validation and Model Selection

The previous sections have introduced the problem of choosing an optimal model, given different parameterized functions. Structural risk minimization, from the field of computational learning theory, depends on the capacity of a class of functions, and provides

a principled way for choosing an optimal model based on the upper bound of VC confidence. The principle works without having to know the true error. In statistical learning theory, the same problem is posed as having to trade off bias with variance, such that the true error is minimized.

Whenever the capacity is hard to determine, there are two different ways to select a good model, i.e. trade off bias with variance in an optimal way. The first may be to assess models analytically, as done, for instance, by methods such as AIC, BIC or MDL (for a more detailed discussion, see [HTF09]). The second is to estimate the true error empirically based on independent test sets. The advantage of the second approach is that it works as a black-box approach, and may be used for arbitrary learners. Especially, it allows for the selection of models from different learners. All experiments in this thesis therefore follow the empirical approach of model validation and selection.

As we have seen in Sect. 3.1.2, an empirical approach for obtaining a better estimate of the true error (or R_{emp}) is to evaluate the performance of a hypothesis on a hold-out test set. In the case where we can vary the complexity of models based on a trade-off parameter, we must be careful not to mix the sets used for testing and parameter optimization (i.e. model selection), as this would introduce a bias into the estimation. The given sample S is therefore usually divided into three different subsets [HTF09]: A training set, a validation set and a test set. For model selection, we use the training and validation set. The quality of the finally chosen model is then evaluated on the hold-out test set. However, in practice, obtaining instances and corresponding labels can be costly. Especially in cases where all that is given is sample S, the validation and test set will need to be subsets of S. At the same time, we want to use as many instances as possible for training. Therefore, if S is small, we need to use examples for training and testing as economically as possible. One technique do to so is cross validation.

Cross Validation *Cross validation* [Koh95] is an estimation technique which divides sample S into k disjunct and independent subsamples $S_1, \ldots, S_k \subseteq S$ of about equal size. These subsamples are also called *folds*, which is the reason why we speak of k-fold cross validation. The cross validation is *stratified* if the original label ratios in S are maintained in each of the folds. For estimation, in each of k rounds, a different fold is left out for testing, while a combination of the remaining $k-1$ folds is used for training. The total estimated performance is the average of the sample error (or another performance value) over all k folds. A special case of cross validation is *leave-one-out*. Here, in each fold, a learning algorithm is trained on all training examples, except for one that is left out for testing, i.e. $k = n$. It can be shown that cross validation provides a good estimate of the true error $err_D(h)$. In this thesis, models are select by cross-validation, if not indicated otherwise.

It should be noted that, in a particular application, we are usually not interested in an estimate of the true error (also called *expected test error* in [HTF09]), but in the so called conditional *test error* or *generalization error* [HTF09], given a fixed sample S for

training:

$$err_{SD}(h) = P_{x \in D}[c(x) \neq h(x)|S] \tag{3.12}$$

That is, before deployment of a classifier in a particular application, we would like to use as many training examples as possible from a fixed sample S at hand, and then estimate the performance of the resulting classifier. Cross validation doesn't estimate the conditional test error, given a fixed training set, but instead averages over different training and test sets. Unfortunately, according to [HTF09], it does not seem possible to estimate conditional error effectively, given only the information in the same training set. However, for this thesis, the discrepancy between true error and test error doesn't play a role, since we are not interested in choosing the best method for a particular application problem, but in the general performance of methods over different domains and datasets.

3.1.8 The CRISP-DM Process

According to the *Cross Industry Standard Process for Data Mining* (CRISP-DM) [She00], the process of data mining consists of the following common steps:

Business Understanding First of all, the data analysis problem has to be understood from a business perspective. From there, a more concrete formulation of the data mining problem can be derived, and particular objectives defined.

Data Understanding Data for analysis needs to be collected and understood, and its quality must be assessed. What data types do occur? Is the data structured or unstructured? Are there any values or features missing? What is the size of the data? If it is stored in a database, how many relational tables and records are there? If the data is unstructured, how many documents, images or whatever are there and what is their size?

Data Preparation Once it is clear how the raw data looks like, it must be decided how to prepare and transform the data for the modeling step, which comes next. For instance, raw data spread over different relational database tables might somehow need to be transformed into single observations. Missing values have to be replaced, and the data might need to be normalized. Subsets of observations and features have to be selected.

Modeling In this phase, different learning algorithms and models have to be evaluated, and their parameters need to be adjusted accordingly. At the end, the model which is best from a data analysis perspective is chosen and put to evaluation in the next step.

Evaluation Here, it is checked if the model chosen also meets constraints and needs from a business perspective.

Deployment The insights gained into the data by the modeling step must be somehow presented in the given business context, for instance in the form of reports. Maybe it is also necessary to integrate or connect the data preparation and modeling steps with existing software, such that they can be executed repeatedly.

In Chap. 2 on the IoT, many different business use cases for data analysis have been described (business understanding) and it was discussed how data in the vertically partitioned scenario might look like (data understanding). The rest of this thesis loosely orients itself on the CRISP-DM phases as well. In Chap. 5, the form of IoT generated data will be discussed in more depth and different techniques will be presented how to transform the given raw data into a format for modeling. All steps have been applied and evaluated in a case study, except for deployment. The chapter on learning from label proportions, Chap. 6, is in a sense a mixture of data preparation and modeling: Given only the proportions of labels for groups of observations, the individual labels need to be reconstructed (data preparation) for learning (modeling). However, many referenced algorithms do not reconstruct the labels first, but derive a model directly. The remaining chapters then describe learning algorithms for the vertically partitioned data scenario that may be used in the modeling step.

3.1.9 Propositional Representation

As discussed for PAC learning, observations are words from a representation language, whereas hypotheses are words from a hypotheses language. Which languages, i.e. which representations, to choose, depends on the particular learning problem. Hence, in addition to the problem of model selection for a specific representation and hypotheses language, we have the problem of selecting the best languages for a learning problem. Having to search for the best representation every time a new problem occurs would be cumbersome. Over time, the fields of machine learning and data mining have therefore established different canonical representations for common types of learning problems. Such representations might not be optimal in each particular case, but work sufficiently well on average. The advantage of canonical representations is that they ease the development of learning algorithms, which can expect observations to have similar form.

A common representation of observations is the *propositional representation*. Observations are described by a fixed number of p *features* or *attributes* $A_1, \ldots, A_p$, which represent properties of entities. We have already seen examples, like people being described by their age, height and weight. The values of attributes can be either numerical, ordinal, or categorical. A set of observations with their attributes forms a so called *data table*, in which observations are stored in rows, and their attribute values in corresponding columns. The propositional representation is thus closely related to the representation of records in relational database tables.

Many learning algorithms, like those presented in the next sections, work on observations given in propositional form. Distance based methods are more independent from the particular representation. However, many accompanying distance measures,

like Euclidean distance, also expect observations to be in propositional form. In the rest of this thesis, whenever we assume instances to be given in propositional form, we'll write $\mathbf{x} \in X$ instead of $x \in X$. The individual feature values will be denoted as $\mathbf{x}[j]$, where j means the value of feature A_j.

In all cases where the raw data is represented differently, it must first be brought into propositional form. In an IoT context, the raw data given are oftentimes time-related sequences of real-valued measurements from one or different sensors. How to represent time-related or spatiotemporal information in an optimal way for learning can be a non-trivial problem [Mor00]. In Chap. 5, we show how series of sensor measurements may be transformed into propositional form in the context of a smart manufacturing case study.

3.2 Supervised Learning Methods

The following sections give an overview and short description of supervised learning methods. The methods were either applied in the experimental sections of this thesis, or used and modified in the algorithmic sections.

The reasoning behind the selection for empirical evaluation is that chosen methods cover different popular model classes, making different assumptions about the data. The k-Nearest Neighbor method is distance-based and can model non-linear decision boundaries. Naïve Bayes is a probabilistic classifier making specific assumptions on the dependency structure between target value and attributes. This kind of assumption also plays a large role for communication-efficient learning in the vertically partitioned data scenario, as discussed in Sect. 4.5. Decision tree models are highly interpretable and easy to understand, but can only model axis-parallel decision boundaries. Random forests improve on the accuracy of decision trees by combining several of them in an ensemble. The technique is similar, but not equivalent, to the learning of local models in a vertically distributed setting. The support vector method (SVM) has a strong theoretical foundation, following the principle of structural risk minimization, and can be shown to have good generalization performance. The SVM allows for the modeling of linear as well as non-linear decision boundaries by the use of kernel functions. Kernel functions have been developed for a large variety of domains, making the SVM quite general. Running the SVM on vertically partitioned data is therefore desirable. However, as will be discussed in Sect. 4.5, the development of communication-efficient distributed versions is non-trivial, depending on kernel function.

3.2.1 k-Nearest Neighbors

Given an observation x to classify, *k-nearest neighbors* (k-NN) [Aha92] finds a set $N_k(x)$ of k labeled observations in the training set which are most similar to x and assigns the majority class (label) to x. To arrive at a unique decision, k should be an odd number.

For regression, k-NN averages the output values of all k nearest neighbors:

$$\hat{f}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i \tag{3.13}$$

In $\mathbb{R}^p$, the similarity (or dissimilarity) of observations can easily be measured by a metric, like Euclidean distance. According similarity measures can also be defined for ordinal or nominal values.

The parameter k trades off bias against variance. If k is high, k-NN has high bias, and low variance. Predictions are less noisy, but boundaries between classes are less distinct. If k is small, bias is low and variance is high. Actually, setting $k = 1$ comes closest to rote learning. k-NN performs well in many cases. A disadvantage is that the model is instance-based and consists of all training examples, which can be a large set. The time needed for prediction can be reduced by the creation of spatial index structures. However, those do usually not perform well in higher dimensions.

Application of kNN in the vertically partitioned data scenario is rather non-trivial. Independent distance calculations on different nodes, i.e. on local features only, may find entirely different sets of k-nearest neighbors. At best, local nodes could send the partial sums of all instances to a central node. This still has communication costs in the order of the sample size, $O(nm)$, where m is the number of nodes. However, this would need to be done for each instance to be classified. In comparison, other methods which are not instance-based, need to communicate only $O(m)$ values per prediction.

3.2.2 Naive Bayes

Naïve bayes [JL95] is a probabilistic classifier. Given an instance $\mathbf{x} \in X$ in propositional form, it determines probabilities $P(Y_v \,|\, \mathbf{x})$ for each of l classes, and decides which class to assign, based on the estimated probabilities. The conditional probability $P(Y_v \,|\, \mathbf{x})$ can be expressed using Bayes' theorem as

$$P(Y_v \,|\, \mathbf{x}) = \frac{P(Y_v)P(\mathbf{x} \,|\, Y_v)}{P(\mathbf{x})} \text{ for } v = 1, \ldots, l\,. \tag{3.14}$$

For the decision, only the numerator is needed. It is equivalent to the joint probability model $P(Y_v, \mathbf{x}[1], \ldots, \mathbf{x}[p])$. Using the chain rule for a repeated application of the definition of conditional probability, it can be rewritten as

$$\begin{aligned}
P(Y_v, \mathbf{x}[1], \ldots, \mathbf{x}[p]) &= P(\mathbf{x}[1], \ldots, \mathbf{x}[p], Y_v) \\
&= P(\mathbf{x}[1] \,|\, \mathbf{x}[2], \ldots, \mathbf{x}[p], Y_v) \cdot P(\mathbf{x}[2], \ldots, \mathbf{x}[p], Y_v) \\
&= P(\mathbf{x}[1] \,|\, \mathbf{x}[2], \ldots, \mathbf{x}[p], Y_v) \cdot P(\mathbf{x}[2] \,|\, \mathbf{x}[3], \ldots, \mathbf{x}[p], Y_v) \cdot \\
&\quad\; P(\mathbf{x}[3], \ldots, \mathbf{x}[p], Y_v) \\
&= \ldots \\
&= P(\mathbf{x}[1] \,|\, \mathbf{x}[2], \ldots, \mathbf{x}[p], Y_v) \cdot P(\mathbf{x}[2] \,|\, \mathbf{x}[3], \ldots, \mathbf{x}[p], Y_v) \cdot \ldots \\
&\quad\; P(\mathbf{x}[p-1] \,|\, \mathbf{x}[p], Y_v) \cdot P(\mathbf{x}[p] \,|\, Y_v) P(Y_v)
\end{aligned}$$

Naïve Bayes makes the "naïve" assumption that each feature A_i is conditionally independent of every other feature A_j, $i \neq j$, given class Y. Therefore,

$$P(\mathbf{x}[j] \,|\, \mathbf{x}[j+1], \ldots, \mathbf{x}[p], Y_v) = P(\mathbf{x}[j] \,|\, Y_v)\,. \tag{3.15}$$

The conditional distribution over class Y_v can then be expressed as

$$P(Y_v \,|\, \mathbf{x}[1], \ldots, \mathbf{x}[p]) = \frac{1}{Z} P(Y_v) \prod_{j=1}^{p} P(\mathbf{x}[j] \,|\, Y_v) \tag{3.16}$$

$Z = P(\mathbf{x})$ is a constant scaling factor dependent only on $\mathbf{x}[1], \ldots, \mathbf{x}[p]$ and constant if the feature values are known.

The individual probabilities can be estimated feature-wise. For numerical features, one often makes a Gaussian assumption. The data is first segmented by class, and then the mean and variance are estimated from the given feature values. For nominal feature values, their relative frequencies are determined by counting.

The final decision rule for classification is often the MAP criterion, picking the most probable hypothesis, which is also used in the optimal Bayes classifier (see Section 3.1.2):

$$\hat{f}(x) = \underset{v \in \{1,\ldots,l\}}{\operatorname{argmax}} \; P(Y_v) \prod_{j=1}^{p} P(\mathbf{x}[j] \,|\, Y_v)\,. \tag{3.17}$$

As will be discussed in Sect. 4.5, the conditional independence assumption plays an important role for communication-efficient learning in the vertically partitioned data scenario.

3.2.3 Decision Tree Induction

Decision trees classify instances in propositional form according to tests on attributes along their nodes and branches. For features with nominal values, internal nodes ask for the values of features, while outgoing arcs represent the different possible values of features, i.e. test outcomes. For numerical features, nodes may represent binary tests which ask if a feature value is greater (or smaller) than some threshold value. The arcs are labeled with the two possible test outcomes, "yes" and "no". The leafs of a decision tree represent the outcome of all tests along a path, which is the class to be assigned. For classification, instances may take different paths through the tree, depending on their particular feature values and test outcomes, and are finally assigned the class of the leaf they end up in.

The learning task of decision tree induction consists of finding a tree that minimizes the true error, i.e. the classification error on an independent test set, given a sample S of instances. The search space over all possible trees is huge. For nominal features, it is at least finite, but already exponential in the number of features. For numerical features, the number of possible threshold values and therefore tests is even infinite. Hence, decision trees are induced by heuristic strategies.

A greedy heuristic for the induction of decision trees is the *top-down induction of decision trees* (TDIDT). Each decision rule along a path splits instance space X into ever smaller axis-parallel regions. A set of instances is thus split into ever smaller subsets. The greedy heuristic works recursively. In each step, it is decided which attribute, and, for numerical features, which threshold value, would provide the best split of instances, as measured by some quality criterion. Tests on the same numerical features may appear repeatedly along a path, but with different threshold values. The recursion either stops when no more attributes are available, when a node's associated subset contains only instances of the same class, or when the split would yield an empty subset. The last nodes are then turned into a leaf, which is labeled with the majority label of instances contained in it's associated subset.

A popular criterion for the quality of a split, the *information gain*, comes from information theory. Let P_i be the probability for choosing item i from a discrete set X of l items. The *entropy* $I(X)$ is then defined as

$$I(X) = -\sum_{i=1}^{l} P_i \log_2 P_i \tag{3.18}$$

Entropy is a measure of unpredictability of information content. Low entropy means that an event is fairly predictable. For instance, if the probability that item i is getting chosen is one, and all other events have probability zero, the event that item i will be chosen is fully predictable. If, on the other hand, all probabilities P_i are similar, it is much harder to predict which item will be chosen. Entropy is therefore high.

In the context of decision trees, entropy can be thought of as a measure of unpredictability of the class label. For a set of instances, P_i is the probability that an instance has class label i, and for a concrete set, can be estimated by relative frequencies. An attribute with k different values splits a set of instances into k different subsets. The total entropy over all subsets is then the sum of the subset's individual entropies. The information gain is the difference between the entropy of instances with and without being split by a particular attribute. The attribute with the highest information gain is chosen for the split, as it provides most information about the class label. One can say that with each test along a path, more information is gained about an instance's class label. What cannot be avoided is that, since the algorithm is greedy, another choice of parent attributes might have led to an even bigger information gain.

Information gain is highest if each subset resulting from a split is pure, i.e. it only consists of instances having the same class. There are other criterions that measure purity, like the Gini impurity. The most complex decision tree has only one instance in each of its leafs, which are pure. To avoid overfitting, different techniques exist. The tree can be pruned from the leafs, resulting in smaller trees. The maximum depth of trees can be limited. In comparison to k-NN, where model complexity (and thereby the bias variance trade-off) is controlled by only a single hyper parameter, the complexity of decision trees might be controlled by several hyper parameters.

Given that there are different types of attributes, impurity measures, pruning techniques, and ways to build trees, there exist different algorithms for the induction of decision trees. Popular algorithms are ID3 [Qui86], CART [BFOS84], and C4.5 [Qui93]. Experiments in this thesis use the decision tree implementation from RapidMiner. With modifications, decision trees can also be used for regression [BFOS84].

3.2.4 Random Forests

Random forests by [Bre01] combine many decision trees for classification in a so called *ensemble*. As discussed, deep decision trees may easily overfit the training data, i.e. they have low bias and high variance. Random forests try to reduce this variance by averaging over the results of multiple deep decision trees. For classification, random forests make a majority vote over the classes, as predicted by each individual tree. For regression, the mean of outputs is taken.

In a procedure called *tree bagging*, the training set is sampled randomly with replacement T times, and a single tree is fitted to each sample. Variance is decreased by averaging over the predictions of all trees, without increasing the bias, as long as trees are uncorrelated. Bootstrap sampling (see [Koh95, HTF09]) ensures that trees are trained on different training sets, and therefore decorrelated.

Despite the bootstrap sampling procedure, highly relevant features may still be selected in different trees and may lead to correlation among trees. Therefore, random forests combine tree bagging with feature bagging, by choosing from a random subset of features in each recursive step of decision tree induction.

In many experiments over standard datasets, random forests perform similarly well as support vector machines, or even better. Random forests were therefore used for comparison in some experiments in this thesis.

3.2.5 Support Vector Method (SVM)

Large margin approaches, like the *Support Vector Method* (SVM) [Vap95, Vap99], have a strong theoretical foundation as they follow the structural risk minimization principle (see Sect. 3.1.5). The number of possible solutions and therefore the capacity of the function class is reduced by maximizing a margin between a decision function and the nearest data points.

Support Vector Classification Let observations be vectors consisting of p real-valued components, i.e. $\mathbf{x}_i \in X \subseteq \mathbb{R}^p$. Let further constrain Y to two values, -1 and +1, which is the problem of *binary classification*, and for simplicity assume that all observations are linearly separable. Then there must exist a hyperplane

$$H = \{h | \langle \mathbf{w}, h \rangle + b = 0\}$$

with $\mathbf{w}$ being normal to H, bias b with $|b|/||\mathbf{w}||$ being the perpendicular distance of H to the origin and $||\mathbf{w}||$ being the Euclidean norm of $\mathbf{w}$, such that

$$\forall_{i=1}^{n}\, y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 0,$$

i.e. that all observations of a particular class are lying in the same half space as given by H. Given $\mathbf{w}$ and b, observations $\mathbf{x}$ may be then be classified by function

$$\hat{f}(\mathbf{x}, \mathbf{w}, b) = \mathrm{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b)\ .$$

The parameters $\mathbf{w}$ and b define the position and orientation of H and can be seen as the parameter vector $\boldsymbol{\gamma}$ of function $\hat{f}$.

There are infinitely many hyperplanes which correctly separate positive and negative training examples and thereby minimize the empirical risk R_{emp}. However, there is only one hyperplane which also minimizes the structural risk R_{reg}. This hyperplane separates positive and negative observations with the largest possible *margin*, which is defined as the perpendicular distance of points closest to the hyperplane. For normalized $\mathbf{w}, b$ such that points $\mathbf{x}_i$ closest to the hyperplane satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$, the margin is given by $1/||\mathbf{w}||$. Instead of maximizing $1/||\mathbf{w}||$, we may as well minimize $\frac{1}{2}||\mathbf{w}||^2$. Furthermore, one can allow for non-separable data points that lie inside the margin or even in the wrong half space by introducing slack variables ξ_i and minimizing the sum of such errors.

Primal SVM Problem for Non-separable Data The primal optimization problem for non-separable data then becomes

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & \forall_{i=1}^{n}\ :\ y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i\ . \end{aligned} \tag{3.19}$$

In terms of structural risk minimization, parameter C in (3.19) trades off the empirical risk (the sum over all slack variables) against the structural risk (the size of the margin). This relationship can be even easier seen when replacing (3.19) by the *hinge function notation*

$$\min_{\mathbf{w}} \quad \sum_{i=1}^{n} [1 - y_i(\langle \mathbf{w}, \mathbf{x} \rangle + b)]_+ + \lambda||\mathbf{w}||^2\,, \tag{3.20}$$

where $\xi_i = [1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)]_+$ is the so called *hinge loss function*. It can be shown that this problem is a quadratic optimization problem with inequality constraints. Such problems are sometimes easier to solve by introducing Lagrange multipliers α_i, μ_i, $i = 1, \ldots, n$ for each inequality constraint, resulting in the Lagrangian

$$L_P(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \frac{1}{2}||\mathbf{w}||^2 - C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i(y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 + \xi_i) - \sum_{i=1}^{n} \mu_i \xi_i \tag{3.21}$$

Dual SVM Problem for Non-separable Data By setting the partial derivatives of L_P for $\mathbf{w}$ and b to zero and inserting the solutions $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$ and $0 = \sum_{i=1}^{n} \alpha_i y_i$ into (3.21), one obtains the dual SVM problem for non-separable data

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ \text{s.t.} \quad & \forall_{i=1}^{n} : 0 \leq \alpha_i \leq C \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{3.22}$$

The solution $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$ is a linear combination of data points for which $0 \leq \alpha_i \leq C$. Such data points are also called *support vectors* (SVs), as they sufficiently determine the computed hyperplane.

Support Vector Regression (SVR) For real-valued outputs $y_i \in \mathbb{R}$, the primal SVM problem can be stated like

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C \left(\sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \xi_i' \right) \\ \text{s.t.} \quad & \forall_{i=1}^{n} : \langle \mathbf{w}, \mathbf{x}_i \rangle + b \leq y_i + \epsilon + \xi_i' \quad \text{and} \\ & \qquad \langle \mathbf{w}, \mathbf{x}_i \rangle + b \geq y_i - \epsilon - \xi_i \,. \end{aligned}$$

The dual formulation then contains two αs, one for each ξ_i and ξ_i'.

Kernel Functions For better or non-linear separation of observations, it may help to map them to another space, called *feature space*, by a transformation function $\Phi : X \to \mathcal{H}$, sometimes also called a *feature map*. Space $\mathcal{H}$ has usually higher dimension than X. Hence, an explicit calculation of the dot product in (3.22) on the mapped observations can become quite time-consuming. However, it can be shown that for certain mappings Φ, there exist kernel functions $k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$ with $k : X \times X \to \mathbb{R}$ which correspond to dot products in $\mathcal{H}$. For this to work, $\mathcal{H}$ has to be an inner product (Hilbert) space. Often, replacing $\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$ by $k(\mathbf{x}, \mathbf{x}')$ allows for a much more efficient computation of the dot product. This replacement is also known as the *kernel trick*. Apart from efficiency, the use of kernels has another advantage. Since the dual problem only depends on values of the dot product, but not on the observations themselves, instance space X may not only consist of real-valued vectors, but arbitrary objects like strings, trees or graphs which have an associated similarity measure. Further, there exist kernel functions with $\mathcal{H}$ being infinite.

Popular kernel functions are, for instance, the

$$\textbf{Polynomial Kernel} \quad k(\mathbf{x}, \mathbf{x}') = (\kappa \langle \mathbf{x}, \mathbf{x}' \rangle + \delta)^d \,, \tag{3.23}$$

$$\textbf{RBF Kernel} \quad k(\mathbf{x}, \mathbf{x}') = e^{-\frac{||\mathbf{x}-\mathbf{x}'||^2}{2\sigma^2}} \text{ and} \tag{3.24}$$

$$\textbf{Sigmoid Kernel} \quad k(\mathbf{x}, \mathbf{x}') = \tanh(\kappa \langle \mathbf{x}, \mathbf{x}' \rangle - \delta) \,. \tag{3.25}$$

As will become clear in Sect. 4.5, non-linear classification by the use of kernel functions is often difficult to achieve in distributed settings, especially in the vertically distributed scenario.

Solvers There exist several methods that solve the SVM problem. Interior point methods [BV04] replace the constraints with a barrier function. This results in a series of unconstraint problems which can be solved efficiently with Newton or Quasi-Newton methods. However, the general methods have a cubic run-time and quadratic memory requirements. More popular approaches are chunking and decomposition methods [OFG97, Pla99, Joa99], which work on a subset of dual variables at a time. Finally, gradient methods like Pegasos and SVM-perf iteratively update the primal weights. Their convergence rate is usually $O(1/\varepsilon)$.

3.3 Outlier and Anomaly Detection

The task of outlier or anomaly detection can have different purposes. One purpose is the cleansing of data in the data preparation step of the CRISP-DM model (see Sect. 3.1.8). Here, outliers are often seen as undesirable infrequent patterns in the data, which deviate much from the overall distribution D of observations. The reason why they are undesirable is that many performance measure, like for instance the squared quadratic loss in regression, or distance measures, like Euclidean distance, are highly sensitive to strong deviations in patterns. That is, outliers have much influence on the distance calculation, although they do not occur often. Since they can skew analysis results, such outliers are usually regarded as undesirable and not meaningful and should be excluded from the modeling step. In other cases, however, outliers can be meaningful and interesting. For instance, in production settings, failure patterns occur infrequently, and may deviate much from the overall data distribution, but shall be predicted and described by a model. Thus, the purpose of outlier or anomaly detection in the modeling step of CRISP-DM is to derive a model-based characterization of outliers. This characterization may yield insights, for instance, into how to adapt the process such that no or less failures occur in the future.

A popular method for the detection of anomalies is the support vector data description (SVDD) method or 1-class SVM. Like the support vector classifier, it comes with a strong theoretical foundation and is highly flexible, due to the use of kernel functions. It will be described in the following, together with an efficient approximation method, the core vector machine (CVM). In the context of this thesis (see Chap. 8), the CVM will be adapted to work in the vertically partitioned data scenario, and its communication-efficiency will be empirically demonstrated for a number of cases.

3.3.1 Support Vector Data Description (SVDD) and 1-class SVM

Supervised classifiers are trained on two or more classes. Their accuracy may suffer if the distribution of observations over classes is highly imbalanced. For instance, this may

happen in applications where unusual events and therefore data about them is scarce, like faults in machines, quality deviations in production processes, network intrusions or environmental catastrophes. In all such cases, many positive examples are available, but only few or even no examples of the negative class.

The task of data description, or 1-class classification [MKH93], is to find a model that well describes the observations of a single class. The model can then be used to check whether new observations are similar or dissimilar to the previously seen data points and mark dissimilar points as anomalies or outliers. Support Vector Data Description (SVDD) [TD04] computes a spherical boundary around the data. The diameter of the enclosing ball and thereby the volume of the training data falling within the ball are user-chosen. Observations inside the ball are classified as normal whereas those outside the ball are treated as outliers or anomalies.

Primal SVDD Problem Given a sample of observations $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq X$ that all belong to the same class, the *primal SVDD problem* is to find a minimum enclosing ball (MEB) with radius R and center $\mathbf{c}$ around all data points $\mathbf{x}_u \in S$:

$$\min_{R,\mathbf{c}} \quad R^2 \,:\, ||\mathbf{c} - \mathbf{x}_u||^2 \leq R^2,\ u = 1, \ldots, n$$

Dual SVDD Problem Similar to the previously presented support vector methods, kernel functions may be applied whenever observations are arbitrary objects or the decision boundary in the original space is non-spherical. The *dual SVDD problem* after the kernel transformation then becomes

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{u=1}^{n} \alpha_u k(\mathbf{x}_u, \mathbf{x}_u) - \sum_{u,v=1}^{n} \alpha_u \alpha_v k(\mathbf{x}_u, \mathbf{x}_v) \\ \text{s.t.} \quad & \forall_{u=1}^{n} \,:\, \alpha_u \geq 0,\ \sum_{u=1}^{n} \alpha_v = 1\,. \end{aligned} \tag{3.26}$$

The primal variables can be recovered using

$$\mathbf{c} = \sum_{u=1}^{n} \alpha_u \Phi(\mathbf{x}_u), \quad R = \sqrt{\boldsymbol{\alpha}^T \mathrm{diag}(\mathbf{K} - \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha})}$$

where $\mathbf{K} = (k_{uv})$ with $k_{uv} = k(\mathbf{x}_u, \mathbf{x}_v)$ is the $n \times n$ kernel matrix. Support vectors are data points for which $\alpha_u > 0$. An observation $\mathbf{x}$ belongs to the training set distribution if its distance from the center $\mathbf{c}$ is smaller than radius R, where distance is expressed by the set of support vectors SV and the kernel function:

$$||\mathbf{c} - \Phi(\mathbf{x})||^2 = k(\mathbf{x}, \mathbf{x}) - 2 \sum_{u=1}^{|SV|} \alpha_u k(\mathbf{x}, \mathbf{x}_i) + \sum_{u,v=1}^{|SV|} \alpha_u \alpha_v k(\mathbf{x}_u, \mathbf{x}_v) \leq R^2$$

It can be shown [TKC05] that for kernels $k(\mathbf{x}, \mathbf{x}) = \kappa$ (κ constant) that map all input patterns to a sphere in feature space, (3.26) can be simplified to the optimization problem (where $\mathbf{0} = (0, \ldots, 0)^T$ and $\mathbf{1} = (1, \ldots, 1)^T$)

$$\max_{\boldsymbol{\alpha}} \quad -\boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \; : \; \boldsymbol{\alpha} \geq \mathbf{0}, \, \boldsymbol{\alpha}^T \mathbf{1} = 1 \tag{3.27}$$

Whenever the kernel satisfies $k(\mathbf{x}, \mathbf{x}) = \kappa$, any problem of the form (3.27) is an MEB problem. For example, Schölkopf [SPST$^+$01] proposed the 1-class ν-SVM that, instead of minimizing an enclosing ball, separates the normal data by a hyperplane with maximum margin from the origin in feature space. If $k(\mathbf{x}, \mathbf{x}) = \kappa$, the optimization problems of the SVDD and the 1-class ν-SVM with $C = 1/(\nu n)$ are equivalent, and yield identical solutions.

3.3.2 Core Vector Machine (CVM)

Bădoiu and Clarkson [BC02] have shown that a $(1 + \varepsilon)$-approximation of the MEB can be computed with constant time and space requirements. Their algorithm only depends on ε, but not on the dimension p or the number of training examples n. In [TKC05], this algorithm has been adopted for kernel methods like the SVDD.

Let $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq X$ be a sample of instances again. For an $\varepsilon > 0$, the ball $B(\mathbf{c}, (1 + \varepsilon)R)$ with center $\mathbf{c}$ and radius R is an $(1 + \varepsilon)$-approximation of the MEB(S), the minimum enclosing ball that contains all data points of S. A subset $\mathcal{S} \subseteq S$ is called the *core set* of S if the expansion of MEB($\mathcal{S}$) by the factor $(1 + \varepsilon)$ contains S.

The core vector machine (CVM) algorithm shown in Alg. 1 starts with an empty core set and extends it consecutively by the furthest point from the current center in feature space until all data is contained in an approximate MEB. The algorithm uses a modified kernel function $\tilde{k}$ for the reason that optimization problem (3.26) yields a hard margin solution, but can be transformed into a soft margin problem [KSBM00] by introducing a 2-norm error on the slack variables, i.e. by replacing $C \sum_{i=1}^n \xi_i$ with $C \sum_{i=1}^n \xi_i^2$, and replacing the original kernel function k with a new kernel function $\tilde{k} : \tilde{\varphi} \to \tilde{\mathcal{F}}$, where

$$\tilde{k}(\mathbf{x}_u, \mathbf{x}_v) = k(\mathbf{x}_u, \mathbf{x}_v) + \frac{\delta_{uv}}{C}, \quad \delta_{uv} = \begin{cases} 1 : u = v \\ 0 : u \neq v \end{cases} \tag{3.28}$$

The new kernel again satisfies $\tilde{k}(\mathbf{x}, \mathbf{x}) = \tilde{\kappa}$ with $\tilde{\kappa}$ being constant.

The furthest point calculation in step 2 takes $O(|\mathcal{S}_t|^2 + n|\mathcal{S}_t|)$ time for the t^{th} iteration. However, as is mentioned by Schölkopf [SS02], the furthest point obtained from a randomly sampled subset $S' \subset S$ of size 59 already has a probability of 95% to be among the furthest 5% points in the whole dataset S. By using this *probabilistic speed-up* strategy, i.e. determining the furthest point on a small sampled subset of points in each iteration, the running time for the furthest point calculation can be reduced to $O(|\mathcal{S}_t|^2)$. As shown in [TKC05], with probabilistic speed-up and a standard QP solver, the CVM reaches a $(1+\varepsilon)^2$-approximation of the MEB with high probability. The total number of iterations is bounded by $O(1/\varepsilon^2)$, the running time by $O(1/\varepsilon^8)$, and the space

Algorithm 1 Core Vector Machine (CVM) algorithm [TKC05]

S: training set, consisting of n examples
$\mathcal{S}_t \subseteq S$: core set of S at iteration t
$\mathbf{c}_t$: center of the MEB around $\mathcal{S}_t$ in feature space
R_t: radius of the MEB

1. **Initialization:** Uniformly at random choose a point $\mathbf{z} \in S$. Determine a point $\mathbf{z}_a \in S$ that is furthest away from $\mathbf{z}$ in feature space, then a point $\mathbf{z}_b \in S$ that is furthest away from $\mathbf{z}_a$. Set $\mathcal{S}_0 := \{\mathbf{z}_a, \mathbf{z}_b\}$ and the initial radius

$$R_0 := \frac{1}{2}\sqrt{2\tilde{\kappa} - 2\tilde{\kappa}(\mathbf{z}_a, \mathbf{z}_b)}$$

2. **Furthest point calculation:** Find $\mathbf{z} \in S$ such that $\tilde{\phi}(\mathbf{z})$ is furthest away from $\mathbf{c}_t$. The new core set becomes $\mathcal{S}_{t+1} = \mathcal{S}_t \cup \{\mathbf{z}\}$. The squared distance of any point from the center in $\tilde{\mathcal{F}}$ can be calculated using the kernel function

$$||\mathbf{c}_t - \tilde{\phi}(\mathbf{z}_\ell)||^2 = \sum_{\mathbf{z}_u, \mathbf{z}_v \in \mathcal{S}_t} \alpha_u \alpha_v \tilde{k}(\mathbf{z}_u, \mathbf{z}_v) - 2 \sum_{\mathbf{z}_u \in \mathcal{S}_t} \alpha_u \tilde{k}(\mathbf{z}_u, \mathbf{z}_\ell) + \tilde{k}(\mathbf{z}_\ell, \mathbf{z}_\ell)$$

3. **Termination check:** Terminate if all training points are inside the $(1+\varepsilon)$-ball $B(\mathbf{c}_t, (1+\varepsilon)R_t)$ in feature space, *i.e.* $||\mathbf{c}_t - \tilde{\phi}(\mathbf{z})|| \leq R_t(1+\varepsilon)$.

4. **MEB calculation:** Find a new MEB($\mathcal{S}_{t+1}$) by solving the QP problem

$$\max_{\alpha} -\alpha^T \tilde{\mathbf{K}} \alpha \ : \ \alpha \geq \mathbf{0}, \alpha^T \mathbf{1} = 1, \ \tilde{\mathbf{K}} = [\tilde{k}(\mathbf{z}_u, \mathbf{z}_v)]$$

   on all points of the core set. Set $R_{t+1} := \sqrt{\tilde{\kappa} - \alpha^T \tilde{\mathbf{K}} \alpha}$.

5. $t := t + 1$, then go to step 2.

complexity by $O(1/\varepsilon^4)$. The running time and resulting core set size are thus constant and *independent* of the size of the whole dataset.

3.4 Cluster Analysis

The task of cluster analysis is to divide a given sample of observations into groups (or clusters), such that the similarity of observations inside clusters is maximized, and the similarity of observations between clusters is minimized.

The above formulation is general enough to cover particular variants of the problem. For instance, partitional clustering algorithms [Mac67] divide a given sample S of

observations into disjunct groups, while probabilistic [DLR77, WFH11] and fuzzy techniques allow for overlapping clusters. Some algorithms measure similarity with the help of a metric [Mac67], while others use the density of points [EKSX96, Han05]. Subspace clustering algorithms [KKZ09] group observations based on their similarity on a subset of features, while projectional and correlation clustering accounts for data sets that are rotated. Some algorithms respect outliers, others do not. In this thesis, the focus is on partitional clustering. It should be noted that the results of other types of clustering algorithms usually can easily be transformed into a partitioning, except for those of subspace clustering algorithms, where groups of instances depend on different subsets of features.

Partitional clustering algorithms divide a given sample S of observations into clusters $\mathcal{C} = \{C_1, \ldots, C_k\}$, such that $C_i \subseteq S$ and for all $i \neq j$, $C_i \cap C_j = \emptyset$. The similarity (or dissimilarity) of observations can, for example, be measured by their distance $d : X \times X \to \mathbb{R}_0^+$, which is usually assumed to be a metric.

Given a distance measure, we can describe the task of partitional clustering more formally. First, the dissimilarity of observations inside clusters, over all clusters, can be measured by summing up their distances.

Definition 3.4.1 (Within cluster scatter) The *within cluster scatter* $W(\mathcal{C})$ over all clusters can be expressed as

$$W(\mathcal{C}) = \frac{1}{2} \sum_{i=1}^{k} \sum_{x \in C_i} \sum_{x' \in C_i} d(x, x') \,. \tag{3.29}$$

Similarly, we can measure the dissimilarity of observations between different clusters by summing up their distances accordingly.

Definition 3.4.2 (Between-cluster scatter) The *between-cluster scatter* $B(\mathcal{C})$ is the total sum of distances between observations of different clusters, over all clusters:

$$B(\mathcal{C}) = \frac{1}{2} \sum_{x \in C_i} \sum_{x' \in C_j} d(x_i, x_j) \; \forall i \neq j \,. \tag{3.30}$$

Further, we can express the total scatter of all observations, by summing up the distances between all observations.

Definition 3.4.3 (Total scatter) The *total scatter* $T(S)$ of all observations in a sample S is given as

$$T(S) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} d(x_i, x_j) \,. \tag{3.31}$$

The total scatter can be written in terms of the within cluster scatter and between-cluster scatter as follows:

$$T(S) = W(\mathcal{C}) + B(\mathcal{C}) \,. \tag{3.32}$$

This matches an intuition behind our informal description of the task of cluster analysis: An optimal grouping can either be achieved by maximizing the similarity of observations inside clusters or by minimizing the similarity of observations between clusters. Both criterions are equivalent. The problem of partitional clustering can thus be seen as a combinatorial optimization problem, trying out all possible partitionings of S and choosing that partitioning which minimizes, for instance, the within cluster scatter. However, the number of possible combinations grows fast with higher n and k [DH73, JW92]. For most practical cases, trying out all combinations would be infeasible. Algorithms that solve the problem of cluster analysis therefore cannot be exact, but can at best approximate the optimal solution.

The number of clusters k is a user-specified parameter. Which choice of k is optimal or meaningful is highly domain dependent. Without any information about class membership (i.e. labels) or domain-specific external quality measures, the automatic choice of k is a difficult problem. Especially, the within cluster scatter $W(\mathcal{C})$ cannot be used for optimization, since the costs decrease with higher k, leading to the trivial solution $k = n$. Quality measures like the Davies Bouldin or silhouette index are more independent of k. However, the partitionings they produce don't necessarily reflect the way in which an expert would have grouped the observations. At best, there exist heuristics, like the gap heuristic, which can facilitate the choice of k.

Proper choice of distance measure can be difficult as well. Especially the weighting of features can have a large influence on clustering results [HTF09]. On the one hand, larger ranges of values may lead to higher natural weightings of features and may require the equalization of weights by proper normalization of all data points. On the other hand, features might have different importance, which might get lost during normalization. Supervised settings allow for the automatic selection or weighting of relevant features. In an unsupervised setting, however, automatic selection or weighting of features is much more difficult and must be based on additional assumptions of what is interesting.

3.4.1 k-Means Clustering

The k-Means clustering algorithm tries to minimize the within cluster scatter $W(\mathcal{C})$ directly, in a greedy fashion, by iterative descent. The within cluster scatter can be written as

$$W(\mathcal{C}) = \frac{1}{2}\sum_{i=1}^{k}\sum_{x\in C_i}\sum_{x'\in C_i} d(x,x') \tag{3.33}$$

$$= \sum_{i=1}^{k} n_i \sum_{x\in C_i} d(x,\bar{x}_i) \tag{3.34}$$

where $\bar{x}_i$ is the mean vector of the i-th cluster and n_i is the number of observations in this cluster. Assigning all n observations to the k clusters such that within each clus-

Algorithm 2 Lloyd's algorithm

1: **procedure** kMeans(S, k)
2: $C_1, \ldots, C_k := \emptyset$
3: Choose initial centroids $\mathbf{c}_1, \ldots, \mathbf{c}_k$
4: **repeat**
5: $\forall \mathbf{x} \in S : \; C_j := C_j \cup \{\mathbf{x}\}$ with $j := \operatorname{argmin}_{1 \leq i \leq k} ||\mathbf{x}_i - \mathbf{c}_i||^2$
6: $\forall 1 \leq i \leq k : \; \mathbf{c}_i := \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$
7: **until** no assignment change
8: **return** $\mathcal{C} = \{C_1, \ldots, C_k\}, \{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$
9: **end procedure**

ter, the average dissimilarity between observations and the cluster mean is minimized, minimizes the whole criterion.

For the k-Means algorithm to work, it must be possible to calculate the arithmetic mean of observations. Further, the mean calculation must be compatible with the chosen dissimilarity measure, in the sense that the costs of a single cluster are minimized by the cluster mean. This is, for instance, the case for observations represented as points in $\mathbb{R}^p$ and the quadratic Euclidean distance as dissimilarity measure.

Algorithm 2 shows an iterative descent algorithm for the problem, given the aforementioned instance space and distance measure. At the beginning, initial cluster centers are chosen (line 3). There exist different strategies for that choice. A basic variant draws k different cluster centers uniformly at random from the given sample of observations. More sophisticated methods, like k-Means++, try to place cluster centers far apart, in dense regions of the data space. The intuition is that different clusters should be maximally dissimilar, following (3.32). Next, all observations are assigned to their closest cluster centers, according to the given distance measure (line 5). Then, cluster centers are calculated anew, based on the current assignment of observations to clusters (line 6). The last two steps are repeated in an alternating fashion, until assignments (or the total costs $W(\mathcal{C})$) do not change anymore.

For a random initialization of cluster centers, it can be shown that the algorithm is guaranteed to stop in a local minimum. In each step, the costs $W(\mathcal{C})$ are monotonically decreasing. Since the quality of solutions is dependent on the particular initialization, it is recommended to start the algorithm several times with different random choices of cluster centers and take the solution with lowest costs $W(\mathcal{C})$. For k-Means++ in $\mathbb{R}^p$ and quadratic Euclidean distance, it can be shown that the expected costs deviate from the globally optimal solution by a factor which is logarithmic in k.

It should be noted that k-Means can be seen as a partitional variant of expectation maximization (EM) clustering [DLR77, WFH11]. EM clustering assumes the data to be generated by a mixture of k gaussians, estimating their means and covariance structure. Thereby, EM can account for elliptically formed clusters. In comparison, k-Means does only estimate the cluster means. The resulting partitioning corresponds to a centroidal Voronoi tessellation.

Chapter 4

Distributed Data Mining

All methods that were presented in the previous sections originally have been developed to run on a single machine and process sample S as one batch of data. As already described in Sect. 2.2.3, the IoT requires new types of distributed data analysis algorithms. On the one hand, there is a need for algorithms which follow the paradigm of cloud-based parallel high performance computing. On the other hand, there exist highly communication-constrained IoT scenarios which require more decentralized solutions.

After a short introduction to distributed systems in Sect. 4.1 and sensor node clustering in Sect. 4.2, the sections following give an overview of existing distributed data analysis algorithms from the fields of distributed data mining and wireless sensor networks. Both fields are huge, which is the reason why only a selection of references and methods can be presented here. Methods have either been included to illustrate and discuss important and generic principles of distributed data analysis, or they are direct competitors of the algorithms developed in this thesis. To the best of our knowledge, in this regard, our list includes all relevant methods and algorithms. Especially, we will pay close attention to distributed versions of the SVM. The algorithms are first grouped by the type of data partitioning they work on, i.e. horizontal (Sect. 4.3) or vertical (Sect. 4.4). Then, we follow the categorization introduced in Sect. 2.5.2, distinguishing between different kinds of distributed approaches, from centralized to decentralized. In each subsection, methods and algorithms are loosely ordered from supervised to unsupervised. Based on a summary of algorithmic approaches in the vertically partitioned data scenario, challenges of the scenario are discussed in more detail in Sect. 4.5.5.

4.1 Distributed Systems

A distributed system consists of components that can be located on physically different networked machines, communicating and coordinating their actions by passing messages. This makes distributed processes different from parallel processes or threads of execution, which can usually share the same address space in main memory for communication. The distinction is elaborated upon in Sect. 4.1.1. To understand each other, all processes

in distributed systems must adhere to a predefined set of message formats and there have to be regulations when to exchange which types of messages. A set of message formats and a set of rules for the exchange is also known as a *protocol* in network terminology. Sect. 4.1.2 deals with a common model for layers of protocols, known as the OSI reference model. Moreover, it is possible to make a distinction between several *communication types*, which are introduced in Sect. 4.1.3, like synchronous vs. asynchronous communication. Networks can have different *topologies*, each coming with their own advantages and disadvantages. Those are discussed in Sect. 4.1.4. Finally, Sect. 4.1.5 stresses the difference between high performance and pervasive computing, which is highly relevant for this thesis as already explained in Sect. 2.3. Most of the following material is based on [TvS06], which is recommended for further details.

4.1.1 Parallel vs. Distributed Computing

Parallel and distributed systems share common properties, as they allow for the concurrent execution of different processes. However, both types of systems are also fundamentally different.

Parallel programs are assumed to run on different processors, but on the same machine. The components of parallel algorithms usually have access to shared memory segments, which can be used for the coordination of different processes or threads of execution. Communication is only as costly as reading from or writing to main memory. Critical conditions, like program exceptions or hardware failures, will usually leave the system in a well-defined state, which may also include shutdowns of the whole system. Moreover, parallel processes share the same system clock.

In comparison, distributed components run (or at least are expected to run) on different physical entities. By definition, they cannot share the same memory and clock, but have a local memory and clock. The only way to coordinate themselves is by exchanging messages with each other. The time needed for sending and receiving messages over physical network communication links is usually several orders of magnitudes higher than the time to access main memory, except for the most advanced types of underlying network technology (see also Tab. 2.1 again). On battery-powered embedded or mobile devices, communication usually also consumes lots of energy and thus doesn't come for free. Further, in distributed systems partial failures, i.e. failures of individual components, are common and sometimes hard to distinguish from valid system states. For instance, a sender may have difficulties to determine if other components have received a message or not. A message may have been dropped due to failure of communication lines that are not even directly linked to the sender, it may get lost due to a hardware failure at the receiving end, or the receiver got the message, but cannot acknowledge it in the required time frame, caused by a high computational load. It might also occur that the message has been delivered, but the acknowledgement is getting lost on its way back to the sender. In each case, it is hard for the sender to determine the state of the receiving end, leading to difficult synchronization problems. Similar problems may be caused by differing system clocks of different physical entities.

Considering the differences of parallel and distributed systems, distributed algorithms usually must be designed to be much more fault-tolerant and autonomous than parallel algorithms. Moreover, in certain cases, communication costs must be taken explicitly into account and possibly traded off for computation. With the advent of pervasive systems, another limiting factor is the consumption of energy. Due to all of the aforementioned differences, in Section 4.3 and Section 4.4 we will only focus on distributed versions of data analysis algorithms instead of parallel versions that require a shared memory implementation.

4.1.2 Layered Protocols

The communication between different parties necessarily requires standards for how messages are to be exchanged and the format of messages. In other words, each party must speak the same language. A definition of the rules and formats for the exchange is also called a *protocol*. Protocols that can be negotiated dynamically when opening a connection are also called *connection oriented*, while *connectionless* protocols don't require any kind of additional negotiation.

Standards are usually needed for different levels of abstraction. For example, on the physical network layer, it must be agreed upon how many volts should be used to signal the bits 0 and 1. On a somewhat higher level, it must be defined how the length or end of a message should be encoded in bits. On an even higher level, every party must encode numbers like integer and real values or strings in exactly the same way. These data types must first be encoded by the sender and decoded by the receiver and then be translated into the machine's internal encoding for such types. Furthermore, for a reliable communication between two end points, it must be decided how unique sequence numbers are to be generated or when to retransmit messages.

The *Open Systems Interconnection Reference Model* (OSI model) developed by the International Standards Organization (ISO) has identified seven different layers usually involved in network communication. Such layers are organized in a so called *protocol stack*. Messages are passed from top to bottom of the stack by the sender, sent over the physical communication medium, and then passed from bottom to top by the receiver. Each layer may add or remove information like *headers* and *trailers* to and from messages of the previous layer. Such information may contain meta data, like the length of a message or the physical network address of a target device. While the originally developed protocols for the layers were never widely used, the OSI model itself builds the foundation of today's internet protocols, like TCP and IP. In the following, the layers and their jobs are discussed in somewhat more detail.

Physical, Data Link and Network Protocols Protocols on the *physical layer* define how bits are to be transmitted from one end point to the other, i.e. the electrical, mechanical and signaling interfaces. It also defines how many bits per second can be sent and if a two-way communication is possible or not. The *data link layer* groups bits into units called *frames*. It also appends a checksum to each frame, such that the receiv-

ing end can calculate and check if any errors during the transmission have occurred. If not, the sender is asked to retransmit the frame. Therefore, a unique *sequence number* must be assigned to each frame. The *network layer* decides how machines in different networks should be addressed and how to route messages over possibly several nodes between these networks. Currently, the connectionless *Internet Protocol* (IP) is the most widely used network protocol.

Transport Protocols Protocols on the *transport layer* define how information can be reliably sent from one end point to the other. The basic idea is that not every application needs to handle their own error recovery, but to provide the functionality of a reliable communication mechanism to all applications that need it. The transport layer therefore splits messages given to it by the application layer into frames, assigns sequence numbers and continuously keeps track of which messages have been received, which should be retransmitted and if all frames of a message have been received. In cases where the transport protocol is built on top of connection-oriented network services, all frames will arrive in the correct sequence. However, if built on top of a connectionless protocol, the transport layer also must put back all message frames into order once they are received. The *Transmission Control Protocol* (TCP) is an internet protocol and currently the de facto standard for reliable connection-oriented communication, whereas the *Universal Datagram Protocol* (UDP) is a connectionless transport protocol for applications. Other transport protocols exist, like the *Real-Time Transport Protocol* (RTP) for real-time data transfer. It should be noted that this protocol just specifies the package formats for real-time data, but doesn't provide any mechanisms for guaranteeing the delivery of the data.

Higher-Level Protocols Of the higher level protocols, in practice only the *application layer* is ever used. Nevertheless, the concept of a *session* can be important in the context of middleware systems. The *presentation layer* should usually deal with the interpretation of structured information, like records of data, but at least in the Internet protocol suite, no protocol has been specified for this layer.

Application-specific protocols also define the format of messages and rules for how to exchange them, however, they usually are more oriented on queries or functions that should be performed by another host and how to specify the corresponding parameters and return values. For example, the *Hypertext transfer protocol* (HTTP) defines how to ask another host for web pages based on *Uniform Resource Locators* (URLs), the *Post Office Protocol* (POP) defines how emails can be downloaded from a corresponding server and the *File Transfer Protocol* (FTP) defines how to upload and download files to and from a remote host.

4.1.3 Communication Types

Whenever two entities communicate, one can differentiate between different types of communication.

Discrete vs. Streaming In some applications, it is only necessary to exchange single, relatively short messages between applications. For example, a common paradigm for client-server applications is to call procedures or functions on a remote server and get their result as a response, also known as *remote procedure call* (RPC). The parameters for the call must first be translated into a bit string for communication, whose size is usually limited. Similarly, the response is also a bit string of limited size, which must be translated into the return value on the receiver's end. Once the messages are exchanged, the procedure call is finished. A similar type of communication happens in message queuing systems, where the distributed processes are even more decoupled.

In comparison, the continuous transmission of data over a communication channel is called *streaming*. Examples would be copying a large file over the network or the transmission of live video data. While the copy operation produces a finite stream, a live video stream potentially could also be infinite. Moreover, copying a file over the network does not necessarily need to happen in real time, while for a video stream, a certain transmission rate must be guaranteed.

Point to Point vs. Broadcast If a network node sends data to another explicitly specified network node, the communication is called *point to point*. In many cases, for point to point communication, a dedicated connection between the nodes is established. Sending data to several or all nodes in a network at once, potentially not even specifying their network identifiers, is called *multicast* or *broadcast* communication.

The underlying network topology might differ from a logical topology and naming system that are established on top of it (see Sect. 4.1.4). A similar distinction has to be made regarding broadcast communication. A broadcast on the logical layer might be realized by opening several individual point to point connections on the physical layer. For example, broadcasting data to other nodes in a peer to peer network that works on top of the internet usually involves sending messages between routers that might be connected by dedicated cables. Similarly, what looks as a point to point connection on the logical layer, might be realized as true broadcast communication on the physical layer. For example, nodes connected to each other by a hub or a wireless network necessarily receive all or some messages which are sent to other nodes, requiring a filtering of messages based on the recipient's identifier.

Since one of the goals of distributed systems is achieving transparency, how point to point or broadcast communication is realized on the physical layer is often shielded from the developer and end user. Nevertheless, it sometimes can be beneficial to raise its algorithmic realization to the middleware layer of distributed applications. Moreover, in distributed pervasive systems like wireless sensor networks, how broadcast communication is realized on the physical layer and the network topology play such an important role for performance that oftentimes they need to be explicitly considered.

Transient vs. Persistent The communication between two or more nodes is called *transient* if both participants need to be connected to each other or to the network

at the time of communication. In contrast, a *persistent* communication between nodes ensures that messages to other nodes are intermediately stored and delivered to them once they connect again to the network. Thereby, the communication between nodes becomes decoupled in the sense that not every participant must stay online all the time. An example for a persistent communication mechanism is the mail system, where emails to other recipients are stored on intermediate mail servers and delivered to the recipients once they connect to the mail server.

Reliable vs. Unreliable The exchange of messages between physically different network nodes can fail for several reasons. First of all, the hardware of the receiving nodes or of intermediate nodes that route messages might fail. Other failures might be caused by bugs in the software, like an infinite loop or wrong pointer arithmetic. Then, a high process load on a node could render this node not responding. In such cases, it can also happen that message queues run full and messages are automatically discarded. Sometimes, it is also hard to automatically distinguish between these different types of faults. Nevertheless, many applications require a *reliable* communication between nodes, i.e. some guarantee that messages have reached their destination. Additional requirements can be that messages must arrive at the receiving end in order and at most only once. The first problem can usually be solved by sending acknowledgements back to the sender and the second one by giving each message a unique sequence number. Although the TCP protocol provides a reliable point to point connection between nodes, it is only transient and not robust against failure of any of the communication participants. Therefore, depending on the purpose of a distributed system, it can be necessary to provide additional mechanisms for reliable communication on the middleware or application layer. Moreover, since the actions to be performed in case of failures are often highly application dependent, developers and end users sometimes must be prepared to handle communication related errors. It should also be noted that realizing a reliable broadcast between nodes in a network can be a difficult problem to solve.

In other cases, it might not be necessary to guarantee the delivery of messages at the receiver, however. *Unreliable* communication doesn't need to be a problem. Especially in real time systems, for example, the resending of already outdated information often makes no sense at all or can be restricted to a fixed time frame.

Synchronous vs. Asynchronous The communication between two or more nodes is called *synchronous* if the sender waits for an answer before continuing with its execution. An example would be calling a function on a remote host, waiting for the computation to finish and getting the result. In contrast, calling the function *asynchronously* would mean sending the request to another host, continuing with other tasks and getting somehow notified about the result, e.g. by registering a handler. It should be noted that some algorithms necessarily require some form of synchronization. Hybrid algorithms may also require both types of communication.

4.1.4 Topologies

The network topology determines the structure of a network, i.e. the nodes and how they are linked to each other. Formally, it can be described as a graph $\mathcal{G} = (P, E)$ where P is a set of different nodes and E is a set of edges between them. While a *physical topology* describes the location of different types of devices and media and the layout of the physical cabling between then, a *logical topology* describes the way data passes through the network and its nodes. The logical topology must not necessarily match the physical topology, potentially also leading to different distances and transmission rates between nodes. For instance, nodes in an Ethernet network connected to a repeater hub physically form a star topology, though logically the network has a bus topology.

The following subsections shortly explain eight common network topologies, as well as their advantages and disadvantages.

Point to Point The *point to point topology* consists only of two nodes and a link between them. The link can either be *permanent* (dedicated), like a cable connecting two different end points, or *dynamically configured*, by circuit-switching or packet-switching technologies. A direct link between two nodes is the fastest possible connection, as data doesn't need to travel over any other node. The number of nodes that are passed until a message reaches its destination is called the number of *hops*, i.e. that the number of hops for a point to point connection is always zero.

Bus The *bus topology* is mostly suited for local networks, also called *local area networks* (LANs). Each node in the network is connected to a single cable. All nodes receive the same signal. If data is only to be sent to a single node, all nodes must ignore the signal that was not intended for them. This can be achieved by creating an addressing or naming scheme for the nodes. The main advantage of a bus technology is its low cost, since only a single cable is used. Moreover, the number of hops for each message is zero. However, a disadvantage is that the cable can become a single point of failure. If the cable is somehow damaged and breaks, the entire network is down. Moreover, the management of nodes connected by a single cable, like adding new nodes to the cable, can be impractical.

Star In networks with a *star topology*, the nodes are connected to a central device with a point to point connection. The central device usually can either be a hub or switch. If a node wants to communicate with other nodes, all data it sends is first transmitted to the central device and than to one or more of the other connected nodes. Additional nodes can easily be added to a star topology. However, as with the bus topology, the central hub or switch can also become a single point of failure. A hybrid topology, also called *hierarchical star topology*, can be created by connecting several hubs or switches. In a *distributed star topology*, such hubs or switches are connected in a linear fashion, with no central connection point. In a star topology, the number of hops depends on how many switches are passed by a message.

Linear and Ring If the nodes are connected in a chain, such that the end nodes are not also connected to each other, the nodes form a *linear topology*. One also says that the nodes are *daisy chained*. In this type of network, each node is a critical link between two other nodes. In comparison, the nodes in a *ring topology* form a circle. Data is sent along the ring, into one direction, until it reaches its destination. Each node on the ring can receive data from the previous node and send data to the next node. The advantage of a ring topology is that if one of the nodes fails, the data can still be sent into the other direction along the ring and reach its destination. A disadvantage of both topologies is that they do not scale to a large number of nodes, since the average number of message hops increases linearly with each additional node.

Mesh In a *mesh topology*, several nodes are connected to each other directly by point to point links. The nodes are *fully connected* if they form a complete graph, i.e. each node is directly connected with each other node. The main advantage of a fully connected network are its performance (i.e. the number of hops for each message is zero) and the redundancy of the links: Even if some of the links are failing, there is a high chance that the network will remain at least connected. However, a disadvantage is that the number of links grows quadratically with the number of nodes. This make fully connected topologies impractical for large networks. A network is *partially connected* if at least some of the nodes are connected to more than one other node in the network. Even a partial connection of the nodes provides at least some redundancy, though the complexity is not as high as it would be if the nodes were fully connected. Also the required number of message hops to reach another node is usually small.

Tree If nodes are organized in a hierarchy, i.e. that there is a top level or root node and every node has one or several child nodes, the topology is called a *tree topology*. All links between nodes are point to point links. A tree topology must be at least three levels deep, otherwise it would be a star topology. The number of child nodes is also called the *branching factor*. The advantages of a tree topology are that the number of links between nodes is linear in the number of nodes, that the network is scalable (the number of message hops is restricted by the depth of the tree) and that additional nodes can easily be added to the network. Its main disadvantage is that if any node fails, parts of the network become disconnected. It also should be noted that in a tree topology, nodes that are higher up in the hierarchy usually will have a higher communication load than nodes on the periphery. For example, under the assumption that each node communicates with each other node with equal probability, on expectation the top level node would receive about half of the network traffic.

Hybrid If two different basic network topologies are connected to each other, the resulting topology is called a *hybrid topology*. An example would be a *star ring network*, where the central devices (hubs or switches) of several star networks are connected to each other in a ring topology.

4.1.5 High Performance vs. Pervasive Computing

Despite the differences between parallel and distributed systems, algorithms designed for running in data centers often follow the paradigm of parallel *high-performance computing*, as explained in Sect. 2.3.1 on cloud computing. This can be partly justified by the control exerted over such systems, since machines in a cluster or cloud operate in a well-defined and closed environment. In particular, it may be assumed that communication lines are reliable and have a high bandwidth which resembles that of direct memory accesses. Also, the network topology stays the same during the run of a distributed algorithm. The computational resources per node usually may be assumed to be at least as high as that of contemporary hardware and furthermore, each node potentially may use an unlimited amount of energy.

In comparison, *pervasive distributed systems* consist of low cost and low powered small devices that communicate with each other in an ad-hoc fashion over sometimes highly unreliable communication lines. An example would be *wireless sensor networks* (WSNs) [DP10]. WSNs have a plethora of applications, like earth sciences, forest fire detection, air pollution monitoring, oceanographic applications, system health monitoring, or greenhouse monitoring, to name a few. WSNs may vary widely in their topology from simple star or ring network to complicated multi-hop networks. Especially with mobile devices, the network topology might change continuously. Each node works autonomously using its own battery power. The nodes are thus constrained in terms of sensing capability, computational power and transmission ability. In addition, wireless networks have much lower bandwidth than, say, hardwired local area networks.

Sending all raw data to a central base station for post analysis incurs high communication costs, at the same time running the risk of delayed analysis. Needed are novel types of distributed algorithms which can analyze the data as much as possible in situ, i.e. directly at the local nodes where measurements occur. At most they are allowed to communicate in local environments, with their nearby peer neighbors or next bigger nodes. Further, for the reduction of energy, communication channels should be intelligently shut down whenever possible. Hence, algorithms for pervasive distributed systems must be differently designed than their centralized counterparts or algorithms for high-performance computing. The predominant concern often is not speed of computation, but to increase the network's life-time for monitoring purposes, and finding a solution with satisfying accuracy at all, given limited resources.

The next section discusses shortly how techniques from the field of data analysis, like cluster analysis (see Sect. 3.4), can be used for grouping sensor nodes, such that the total energy consumption needed for communication is reduced. The topic of sensor node clustering is included as it demonstrates the severe technological constraints under which algorithms in pervasive distributed systems must operate. In addition, node clustering algorithms use techniques which might be relevant for the development of distributed data analysis algorithms that target such restricted environments.

4.2 Distributed Clustering of Sensor Nodes

Continuous monitoring as well as intermittent querying of sensor networks involves transmitting data from individual sensor nodes, the *sources*, to a single node, the *sink*. Communication costs increase with higher distance r between sensor nodes, as ground reflections from short antenna heights may cause a drop-off of the radio signal power by r^4 [PK00]. Therefore, hierarchical, tiered *multi-hop* architectures with shorter distances between relaying nodes are usually more energy-efficient than letting all sensors communicate directly with some base station [EGPS01].

The sensor nodes in tiered multi-hop networks form clusters, which can be hierarchical. Certain nodes in each cluster are designated as *cluster heads*. Cluster heads fulfill special roles, like relaying signals from local nodes in their cluster to other cluster heads or a base station. They also can manage and restrict network access as well as the life cycle of local nodes, or reduce the amount of data transmitted by aggregating and pre-processing the signals from sensor nodes in their cluster.

Manual placement of sensors and routing through pre-determined paths are only feasible for very small networks. However, typical applications of sensor networks, like environmental monitoring, disaster management or military surveillance missions envision hundreds or even thousands of sensor nodes [AY07], possibly deployed randomly, e.g. dropped by a helicopter. The network is usually left unattended for long periods of time and batteries can't be recharged. While some setups utilize mobile sensors, sensor nodes are usually assumed to operate stationary after deployment. Nevertheless, the network could change over time, since battery-operated sensors may run out of energy and harsh environmental conditions can damage network components. In these scenarios, algorithms are needed that cluster sensor nodes and determine cluster heads dynamically, forming the infrastructure in an ad-hoc manner. Also, they must be able to reconfigure the network when necessary.

Clustering algorithms that have been developed for WSNs mainly differ in their assumptions on the given *network components*, the desired *topology*, and in the *goals* they try to achieve. These in turn influence the used methodologies and running times.

Regarding *network components*, clustering can become more constraint in heterogenous networks where cluster heads have a higher capacity than sensor nodes. Here, the available number of high capacity components will determine the maximum number of cluster heads and therefore the number of clusters. Moreover, if communication costs between cluster heads and sensor nodes are to be minimized, a stationary location of cluster heads will lead to a static assignment of sensors to clusters, except for cases where cluster heads fail and the network needs to be reconfigured. In comparison, in more homogenous networks, also regular sensor nodes can become cluster heads. Clustering algorithms for these networks are usually more dynamic, as they need to continuously balance the energy consumption across all nodes, based on their residual energy. Several algorithms achieve this, for instance, by a regular rotation of cluster heads.

The required *topology* is largely dependent on the given distances between sensor nodes, cluster heads and base stations. Depending on the placement of nodes, the

network topologies that need to be considered can reach from fixed 1-hop [HCB02] over fixed k-hop [YYYA06] to fully adaptive architectures [DHC05]. An important objective is that network components remain connected, i.e. that sensor nodes are able to reach their cluster heads and that cluster heads can reach a base station. Other objectives like minimizing the intra-cluster energy-consumption may need to be traded off with the goal of components staying connected, for example in cases where an energy-optimal cluster head could no longer reach its base station. Taking into account several—possibly contradicting—quality criteria thus turns clustering in WSNs into a multi-objective optimization problem.

The main *goals* that cluster algorithms for WSNs try to achieve are maximal network longevity, connectivity and fault-tolerance. Extending the operational life-time of a WSN requires load-balancing strategies that prevent premature exhaustion of subsets of sensor nodes and cluster heads. The goal of maintaining connectivity is concerned with ensuring that the most important network components can reach each other, possibly putting constraints on the clustering. Fault-tolerance deals with the failure of network components and can be achieved by redundancy, rotating roles of network components as well as re-clustering.

As a survey article [AY07] shows, the clustering algorithms for sensor nodes are quite diverse and hard to categorize. Moreover, there already exist more algorithms than can sufficiently be presented here, even in summary. Therefore, only two algorithms are focused on in more detail. The algorithms were chosen as examples for demonstrating how the same network topology can be achieved by entirely different means and with different running times. At the end of this section, the reader is then pointed to further algorithms.

Hierarchical Control Clustering The clustering scheme introduced in [BK01] forms a hierarchical multi-hop network topology, where the number of layers is determined automatically. The original problem statement is that given an undirected graph $G = (V, E)$ and a positive integer k with $1 \leq k \leq |V|$, for each connected component clusters $V_1, \ldots, V_l$ with $V_i \subseteq V$ should be found such that (1) all vertices are part of a cluster, (2) all subgraphs induced by V_i are connected, (3) cluster size is bounded by $k \leq |V_i| < 2k$, (4) two clusters should only have few common vertices and (5) each vertex belongs to a constant number of clusters. After demonstrating that requirement (5) could be violated in general graphs, the problem is restricted to *bounded disk graphs*, as they are usually given in WSNs. For $R_{\min}$ and $R_{\max}$ being the minimum and maximum transmission radius over all nodes, (u, v) is an edge in G if and only if $R_{\min} \leq d(u, v) \leq R_{\max}$. The algorithm then guarantees that no node is a member of more than $O(\log(R_{\max}/R_{\min}))$ clusters. Furthermore, to fulfill requirement (4), it is necessary to allow a single cluster in G to have a size smaller than k.

The distributed algorithm consists of two phases: *cluster creation* and *cluster maintenance*. The cluster formation process can be started by an arbitrary node in the network, which becomes the root node of a Breadth-First-Search (BFS) tree. The ini-

tiator with the least node ID takes precedence. Every t units of time, each node u broadcasts a *tree discovery* message to nodes that are in its transmission radius. The message contains a source ID, parent ID (initially not set), the ID of the root node, a sequence number and the shortest (known) hop-distance to the root, r. A node v will make u its parent and update its hop-distance if the route through u to r is shorter. The root ID and sequence number are used to distinguish between multiple instances of the cluster creation phase. Next, for *cluster formation*, the sent messages are extended by additional fields representing *subtree size* and *node adjacency*. The size information is aggregated bottom up. When the subtree size of a node w crosses the size parameter k, it forms clusters on its subtree $T(w)$. If $|T(w)| < 2k$, a single cluster containing $T(w)$ is created. Otherwise, children subtrees will be appropriately partitioned, using the node adjacency information. The cluster assignments are propagated to the relevant nodes by *cluster assignment* messages. Once clusters have been formed for $T(w)$, w does not include information about these nodes in subsequent messages. Nodes send a *terminate cluster* message down their subtrees if subtree sizes have not changed for a fixed amount of time. At the end, only the cluster assignments need to be maintained, while the BFS information is unimportant. During cluster maintenance, a sensor node joining the network may either be assigned to an existing nearby cluster V_i, if $|V_i| < 3k - 1$, or clusters are split, like in the cluster creation phase. If existing nodes leave the network, clusters can become disconnected. However, the number of remaining connected components is bounded, since no node is a member of more than $O(\log(R_{\max}/R_{\min}))$ clusters (see above). The connected components are either made clusters of their own or, if their size is $< k$, their nodes will try to join a neighboring cluster. The same is true in cases of link outages and network partitions.

The algorithm converges in $O(n)$ steps, where n is the number of sensor nodes. In principle, it can work with mobile sensor nodes and recover from network failures. It achieves the self-organization of sensor nodes into a multi-hop network and reduces transmission distances, since parent nodes are chosen by the shortest known hop-distance to the root.

DWEHC In [DHC05], a distributed weight-based energy-efficient hierarchical clustering protocol (DWEHC) has been proposed. The key idea is to elect cluster heads not only based on distances from a node to all its neighbors, but also take into account the residual energy of nodes. A basic observation here is that for devices with similar antenna heights, the transmitter power required by distance r is r^α. For three nodes s, r and d, a direct transmission from s to d takes power $||sd||^\alpha + c$, while relaying transmission through a node r takes power $||sr||^\alpha + c + ||rd||^\alpha + c$. In cases where $||sd||^\alpha + c > ||sr||^\alpha + c + ||rd||^\alpha + c$, relaying is more efficient. The neighbors $N_{\alpha,c}(s)$ of a node s are defined as the set of nodes that lie in the transmission range of s and need no relaying. The weight $W(s)$ is then calculated as

$$W(s) = \left(\sum_{u \in N_{\alpha,c}(s)} \frac{(R-d)}{6R} \right) \times \frac{E_{\text{residual}}(s)}{E_{\text{initial}}(s)}$$

where R is the cluster range (the farthest distance nodes can be from their cluster heads) and $E_{\text{initial}}(s)$ and $E_{\text{residual}}(s)$ are the initial and residual energy of node s respectively. The average number of neighboring nodes can be shown to be at most six [DHC05]. Intra-cluster communication will be at minimum when the transmission graph contains the shortest paths between all pairs of nodes in the cluster.

The protocol starts with each node u broadcasting its (x, y) coordinates, establishing its local neighborhood $N_{\alpha,c}(u)$, calculating its weight $W(s)$ and broadcasting it. A node s sets $level(s) = -1$, indicating that it hasn't joined any cluster yet. In the *cluster generation* phase, the following procedure is repeated for a fixed number of six iterations. Let i be the iteration number. A node s first checks if it is assigned to a cluster. If not, it will become a temporary cluster head if its weight is largest among it neighbors, otherwise the neighbor with the largest weight is chosen as a temporary head for s. The ID of the temporary head is broadcasted to all neighbors of s. A node becomes a real cluster head only if a percentage of $(6 - i)/6$ nodes elect the node as their temporary cluster head. In this case, the information is broadcasted to all neighbors, including the (x, y) coordinates, and the level of s is set to 0. There are three cases in which a node doesn't become a cluster head, but a child node:

1. When $level(s) = -1$, node s receives a broadcast message from its neighbor n, including the (x, y) coordinates of its cluster head h_n. If $||sh_n|| < R$, s chooses h_n as its cluster head. $level(s) := level(n) + 1$ and the distance of s from its cluster head is set to $||sn|| + ||nh_h||$.

2. If s receives a message from neighbor n and $level(s) \neq -1$, node s has already chosen its cluster head. If n is assigned to a different cluster head h_n whose distance from s is in cluster range R and the previously calculated distance to its current cluster head is greater than $||sn|| + ||nh_n||$, then h becomes the new cluster head of s and $level(s) := level(n) + 1$.

3. If s receives a message from neighbor n, $level(s) \neq -1$ and the cluster heads of s and n are the same, it is checked whether the distance of node s to its neighbor n is less than the previously calculated distance. If it is, s will choose n as its parent and set $level(s)$ and the new distance as in the second case.

For finalization, the cluster generation is run a last (seventh) time. Afterwards, each node is either a cluster head or a child node.

In comparison to hierarchical control clustering, DWEHC converges in a constant number of iterations. Moreover, it not only respects the distance between nodes, but also their residual energy. In contrast to previously proposed protocols like LEACH [HCB02], DWEHC doesn't require knowledge about the network size, density or homogeneity or

about the number of levels, like HEED [YF04]. While cluster topologies generated by HEED may not achieve minimum energy consumption in intra-cluster communication, it was shown empirically that the energy savings of DWEHC outperform those or HEED. Also, DWEHC produces more well-balanced clusters and a better distribution of cluster heads, resulting in higher energy savings for inter-cluster communication.

Further Reading Hierarchical node clustering and DWEHC are only representatives of several distributed clustering algorithms that have been developed for WSNs. The survey article [AY07] gives a thorough summary of many additional algorithms. For example, other clustering approaches that have a linear convergence rate are LCA [BE81], CLUBS [NC98], RCC [NC98] and EEHC [BC03]. Approaches with a constant number of iterations are, for example, LEACH [HCB02], HEED [YF04], MOCA [YYYA06], EECPL [BA10] or N-LEACH [TSV12].

4.3 Horizontally Distributed Data Analysis Algorithms

In comparison to algorithms that work on the level of sensor nodes, distributed data analysis algorithms work on the level of data acquired by such nodes. The sensor measurements which constitute a single observation can be partitioned either horizontally or vertically (see also Sect. 2.4).

Let's assume that observations are in propositional representation (see Sect. 3.1.9). Then, observations can be thought of as being stored in a fixed-size $n \times p$ data matrix $\mathbf{D}$, whose rows store the p feature values of n observations in S. In the *horizontally partitioned data scenario* [CSH00b], each node stores only a subset of observations. This means for $j = 1, \ldots, m$ nodes, we have samples $S_j \subseteq S$ and $S = S_1 \cup \cdots \cup S_m$ with $S_u \cap S_v = \emptyset$ ($u \neq v$ for $u, v = 1, \ldots, m$) and corresponding data matrices $\mathbf{D}_1, \ldots, \mathbf{D}_m$. For supervised learning, we may further assume a label to be stored with each observation. At each node j, such labels are stored in a corresponding $(p+1)$th label column $\mathbf{Y}_j$.

The following subsections present a selection of algorithms for the scenario and discuss different principle design approaches for distributed algorithms, from centralized to more decentralized architectures.

4.3.1 Local Preprocessing, Central Analysis

Distributed algorithms can balance load and may reduce communication costs by preprocessing data locally, then sending the new representation to a central coordinator for further processing.

Incremental Least Squares SVM [DP06] introduces a distributed SVM based on a slightly different formulation of the SVM, the so called Least Squares SVM [SV99]. In the least squares formulation, the inequality constraints of the original primal SVM problem (see Sect. 3.2.5) are replaced by equality constraints and a 2-norm error, leading

to the unconstraint optimization problem

$$\min_{\mathbf{w},b} \quad \frac{1}{2}||\mathbf{w}||^2 + \frac{\lambda}{2}||\mathbf{1} - \mathbf{Y}(\mathbf{D}\mathbf{w} - \mathbf{1}b)||^2\,,$$

where $\mathbf{Y}$ is a diagonal matrix with $\mathbf{Y}_{ii} = y_i$. Setting the gradient w.t.r. $\mathbf{w}$ and b to zero, instead of a quadratic optimization problem one obtains a system of $(p+1)$ linear equations

$$[\mathbf{w}\, b]^T = \left(\frac{1}{\lambda}\mathbf{I}^\circ + \mathbf{E}^T\mathbf{E}\right)^{-1} \mathbf{E}^T\mathbf{Y}\mathbf{1}\,,$$

where $\mathbf{E} = [\mathbf{D} - \mathbf{1}]$ and $\mathbf{I}^\circ$ denotes a $(p+1) \times (p+1)$ diagonal matrix whose $(p+1)$-th diagonal entry is zero and the other diagonal entries are 1.

As the authors show, it is possible to solve this system of linear equations incrementally:

$$[\mathbf{w}\, b]^T = \left(\frac{1}{\lambda}\mathbf{I}^\circ + \sum_{j=1}^{m} \mathbf{E}_j^T\mathbf{E}_j\right)^{-1} \sum_{j=1}^{m} \mathbf{E}_j^T\mathbf{Y}_j\mathbf{1}\,. \tag{4.1}$$

In the distributed version of their algorithm, each node j computes the local sums $\mathbf{E}_j^T\mathbf{E}$ and $\mathbf{E}_j^T\mathbf{Y}_j\mathbf{1}$ independently from each other and communicates them to a central coordinator. In other words, the original data is first preprocessed locally and reduced to sums, which the coordinator can then sum up to globally solve the linear system of equations (4.1).

The algorithm can speed up computations, because the sums involved in solving the linear system of equations can be computed in parallel over different nodes j. With a linear kernel, the algorithm is communication-efficient if $n > p^2$, i.e. it sends less data than the original dataset. For cases where $p^2 > n$, the authors applied the Sherman-Morrison-Woodbury formula to the linear system of equations, resulting in a $n \times n$ instead of a $(p+1) \times (p+1)$ matrix. For non-linear kernels, the algorithm usually is not communication-efficient, since the original data matrix $\mathbf{D}$ is replaced by the kernel matrix $\mathbf{K}$, resulting in an n-dimensional weight vector $\mathbf{w}$ and thus a system of n linear equations.

Distributed Outlier Detection The authors of [SLMJ07] present a non-parametric statistical technique to identify global outliers in WSNs. The method first derives data histograms locally at each node and sends such statistics to a central coordinator (a base station). The central coordinator uses the histograms to infer a data distribution and filter out the non-outliers. The identification of outliers is achieved by a fixed threshold distance or the rank among all outliers. One of the major drawbacks of this technique is the ability to process only one dimensional data.

[SPP+06] and [PPKG03] use kernel density estimation for outlier detection. They fit kernel densities at each observation instead of comparing all raw observations. Outlier are then identified by user defined thresholds. The techniques achieve high accuracy

in terms of quality of estimation and high detection rate, while having low memory consumption and a small number of transmitted messages.

In statistical approaches, the task is to model the probability distribution of the data using parametric or non-parametric approaches and then tag as outliers those data points which do not fit the modeled distribution. Two local techniques for the identification of outlying sensors are presented in [WCD+07]. Spatial correlation of the readings existing among neighboring sensor nodes is used to detect bad sensors. Each node computes the distance between its own reading and the median reading of its neighboring sensors. One might say that each neighboring sensor provides a summary statistic, i.e. preprocessed data. A node is considered as an outlying node if the absolute value of the distance is sufficiently large compared to a user defined threshold. Accuracy of these outlier detection techniques is relatively low, since they ignore the temporal correlation of sensor readings.

Distributed Clustering USP2P (P2P k-Means Clustering Based on Uniform Node Sampling) improves on a distributed k-Means clustering technique [BGM+06]. It selects s nodes randomly uniformly by a random walk strategy [DK07] to update centroids in each iteration. For a static network, USP2P provides an accuracy guarantee. Communication costs are upper bounded by $O(Ms\log(m))$, where M denotes the maximum allowed number of iterations by source node, s is the random walk length and m is the number of nodes.

4.3.2 Model Consensus

Consensus algorithms iteratively exchange information between nodes until all nodes have converged to the same values on a set of distributed (shared) variables. Such algorithms can work with local nodes and a central coordinator, but may also work in a peer-to-peer fashion. Basic consensus algorithms exist for the calculation of averages, sums, max/min, etc. Communication costs mainly depend on the number of variables on which consensus should be reached and the number of iterations until convergence.

Probabilistic Gossip-based Classification Probabilistic gossip based protocols are simple in their implementation and asymptotically guarantee convergence. They reach consensus on a set of variables by using gossip protocols in which a node randomly chooses another node and exchanges information with it. This process continues for some iterations. It can be shown that the error reduces exponentially at each iteration. Due to their simplicity, such protocols have been used extensively for many distributed algorithms in WSNs.

In [CFSZ11], an algorithm for distributed and classification in WSNs is proposed. The data is modeled as

$$x_i = \theta + y_i + \nu_i$$

where x_i's are the measurements at each sensor node, $\theta \in \mathbb{R}$ is the common (shared) unknown parameter, $y_i \in \{0, 1\}$ are the unknown discrete terms which denote the class label of each node, and ν_i's are zero mean i.i.d. Gaussian random variables with finite variance. The goal of each node is to estimate θ and y_i. Parameter θ is estimated by a consensus algorithm for maximum likelihood estimation over all nodes. Class labels are inferred using a gossip based protocol. The paper further proposes an EM algorithm for the case in which the y_i's are assumed to be i.i.d. Bernoulli trials. Experimental results demonstrate that the proposed methods have convergence rates which are similar as those of existing methods, but are more robust in various situations, like in the presence of outliers.

Distributed Decision Trees [BWGK08] have proposed a decision tree learning algorithm which can build the same tree on networked nodes in an asynchronous fashion. The main building block of the algorithm is the scalable distributed majority voting protocol first discussed in [WS04]. Given a pair of real numbers a_i and b_i at each node, this algorithm decides if $\sum_i a_i > \sum_i b_i$ in a very communication efficient fashion, without needing a node to exchange messages even if a_i and b_i are changing. Based on this protocol, first, the authors show that comparison of two features can be accomplished by concurrently running 4 majority votes. The next step is to choose top 1-out-of-k attributes and this can be easily accomplished by running the previous comparison per attribute pair. Finally, the tree can be built asynchronously by performing this 1 out of k comparison for each level of the tree. First of all, this algorithm is guaranteed to converge to the globally correct solution. Extensive experimental results also show that the algorithm is communication-efficient, even when the data is changing.

Alternating Direction Method of Multipliers (ADMM) The Alternating Direction Method of Multipliers (ADMM) extensively described in [BPC+11] is a method for consensus optimization which can be applied to a large variety of learning problems. The approach followed in general solves the problem

$$\begin{aligned} \min_{\mathbf{v}} \quad & f_1(\mathbf{v}) + f_2(\boldsymbol{A}\mathbf{v}) \\ \text{s.t.} \quad & \mathbf{v} \in \mathcal{P}_1, \boldsymbol{A}\mathbf{v} \in \mathcal{P}_2 \,, \end{aligned}$$

where $f_1 : \mathbb{R}^{p_1} \to \mathbb{R}$ and $f_2 : \mathbb{R}^{p_2} \to \mathbb{R}$ are convex functions, $\boldsymbol{A}$ is a $p_2 \times p_1$ matrix, while $\mathcal{P}_1 \subset \mathbb{R}^{p_1}$ and $\mathcal{P}_2 \subset \mathbb{R}^{p_2}$ denote non-empty polyhedral sets. The problem is made separable by introducing an auxiliary variable $\boldsymbol{\omega} \in \mathbb{R}^{p_2}$:

$$\begin{aligned} \min_{\mathbf{v},\boldsymbol{\omega}} \quad & f_1(\mathbf{v}) + f_2(\boldsymbol{\omega}) \\ \text{s.t.} \quad & \boldsymbol{A}\mathbf{v} = \boldsymbol{\omega}, \mathbf{v} \in \mathcal{P}_1, \boldsymbol{\omega} \in \mathcal{P}_2 \end{aligned}$$

Let $\boldsymbol{\alpha} \in \mathbb{R}^{p_2}$ denote the Lagrange multipliers corresponding to the constraint $\mathbf{A}\mathbf{v} = \boldsymbol{\omega}$. The augmented Lagrangian is

$$\mathcal{L}(\mathbf{v}, \boldsymbol{\omega}, \alpha) = f_1(\mathbf{v}) + f_2(\boldsymbol{\omega}) + \alpha^T(\mathbf{A}\mathbf{v} - \boldsymbol{\omega}) + \frac{\eta}{2}||\mathbf{A}\mathbf{v} - \boldsymbol{\omega}||^2 \,,$$

where $\eta > 0$ controls how much equality constraints may be violated. ADMM minimizes $\mathcal{L}$ in an alternating fashion, i.e. first for the primal variable $\mathbf{v}$ and then for the auxiliary variable $\boldsymbol{\omega}$. After each iteration, it updates the multiplier vector $\boldsymbol{\alpha}$. With t denoting the current iteration, the ADMM iterates at $t+1$ are given by

$$\begin{aligned}\mathbf{v}^{t+1} &= \underset{\mathbf{v}\in\mathcal{P}_1}{\operatorname{argmin}}\, \mathcal{L}(\mathbf{v}, \boldsymbol{\omega}(t), \boldsymbol{\alpha}^t),\\ \boldsymbol{\omega}^{t+1} &= \underset{\boldsymbol{\omega}\in\mathcal{P}_2}{\operatorname{argmin}}\, \mathcal{L}(\mathbf{v}^{t+1}, \boldsymbol{\omega}, \boldsymbol{\alpha}^t),\\ \boldsymbol{\alpha}^{t+1} &= \boldsymbol{\alpha}^t + \eta(\boldsymbol{A}\mathbf{v}^{t+1} - \boldsymbol{\omega}^{t+1})\end{aligned}$$

The first two optimization problems may be solved on different processors or machines. In a distributed setting, their results must be communicated over the network, such that each node can update its multiplier vector $\boldsymbol{\alpha}$. It can be proven that after a finite amount of iterations, the iterates $\boldsymbol{\alpha}^t$ will converge to the globally optimal solution $\boldsymbol{\alpha}^*$ of the dual problem.

Distributed Consensus SVM In [FCG10], the standard SVM problem is divided into a set of decentralized convex optimization problems which are coupled by consensus constraints on the weight vector $\mathbf{w}$. Thereby the SVM problem is cast into an ADMM formulation. The network is modeled by an undirected graph $G(P, E)$, where vertices $P = \{1, \ldots, m\}$ represent nodes and the set of edges E describes communication links between them. The graph is assumed to be connected and each node j only communicates with nodes in a one-hop neighborhood $\mathcal{N}_j \subseteq P$. Each node $j \in P$ stores a sample $S_j = \{(\mathbf{x}_{j1}, y_{j1}), \ldots, (\mathbf{x}_{jn_j}, y_{jn_j})\}$ of labeled observations, where $\mathbf{x}_{ji}$ is a $p \times 1$ vector from $\mathbb{R}^p$ and $y_{ji} \in \{-1, +1\}$ is a binary class label. The original primal SVM problem (see Sect. 3.2.5) is cast into the distributed ADMM framework by putting consensus constraints on the weight vectors $\mathbf{w}_j, \mathbf{w}_l$ and bias variables b_j, b_l of each node j and its one-hop neighboring nodes $l \in \mathcal{N}_j$:

$$\begin{aligned}\min_{\{\mathbf{w}_j, b_j, \xi_{ji}\}} \quad & \frac{1}{2}\sum_{j=1}^{m} ||\mathbf{w}_j||^2 + mC\sum_{j=1}^{m}\sum_{i=1}^{n_j}\xi_{ji} & \\ \text{s.t.} \quad & y_{ji}(\langle \mathbf{w}_j, \mathbf{x}_{ji}\rangle + b_j) \geq 1 - \xi_{ji} & \forall j \in P, i = 1, \ldots, n_j\\ & \xi_{ji} \geq 0 & \forall j \in P, i = 1, \ldots, n_j\\ & \mathbf{w}_j = \mathbf{w}_l, b_j = b_l & \forall j \in P, l \in \mathcal{N}_j\,.\end{aligned}$$

For ease of notation, the authors define the augmented vector $\mathbf{v}_j := [\mathbf{w}_j^T, b_j]^T$, the augmented matrix $\mathbf{X}_j := [[\mathbf{x}_{j1}, \ldots, \mathbf{x}_{jn_j}]^T, \mathbf{1}_j]$, the matrix $\mathbf{Y}_j := \operatorname{diag}([y_{j1}, \ldots, y_{jn_j}])$ of diagonal labels, and the vector of slack variables $\boldsymbol{\xi}_j := [\xi_{j1}, \ldots, \xi_{jn_j}]^T$. $\boldsymbol{\Pi}_{p+1}$ is a $(p+1) \times (p+1)$ matrix with zeros everywhere except for the $(p+1), (p+1)$-st entry. It follows that $\mathbf{w}_j = (\mathbf{I}_{p+1} - \boldsymbol{\Pi}_{p+1})\mathbf{v}_j$ for $[\boldsymbol{\Pi}_{p+1}]_{(p+1)(p+1)} = 1$. With these vector and

matrix notations, the problem can be rewritten as

$$\begin{aligned}
\min_{\{\mathbf{v}_j,\boldsymbol{\xi}_j,\boldsymbol{\omega}_{ji}\}} \quad & \frac{1}{2}\sum_{j=1}^{m}\mathbf{v}_j^T(\mathbf{I}_{p+1}-\boldsymbol{\Pi}_{p+1})\mathbf{v}_j + mC\sum_{j=1}^{m}\mathbf{1}_j^T\boldsymbol{\xi}_j \\
\text{s.t.} \quad & \mathbf{Y}_j\mathbf{X}_j\mathbf{v}_j \succeq \mathbf{1}_j - \boldsymbol{\xi}_j && \forall j \in P \\
& \boldsymbol{\xi}_j \succeq \mathbf{0}_j && \forall j \in P \\
& \mathbf{v}_j = \boldsymbol{\omega}_{jl}, \boldsymbol{\omega}_{jl} = \mathbf{v}_l && \forall j \in P, \forall l \in \mathcal{N}_j
\end{aligned}$$

where the auxiliary variables $\{\boldsymbol{\omega}_{jl}\}$ decouple parameters $\mathbf{v}_j$ at node j from those of its neighbors $l \in \mathcal{N}_j$. The augmented Lagrangian for the problem is

$$\begin{aligned}
\mathcal{L}(\{\mathbf{v}_j\},\{\xi_j\},\{\boldsymbol{\omega}_{jl}\},\{\alpha_{jlk}\}) = {} & \frac{1}{2}\sum_{j=1}^{m}\mathbf{v}_j^T(\mathbf{I}_{p+1}-\boldsymbol{\Pi}_{p+1})\mathbf{v}_j + mC\sum_{j=1}^{m}\mathbf{1}_j^T\xi_j \\
& + \sum_{j=1}^{m}\sum_{l\in\mathcal{N}_j}\boldsymbol{\alpha}_{jl1}^T(\mathbf{v}_j-\boldsymbol{\omega}_{jl}) + \sum_{j=1}^{m}\sum_{l\in\mathcal{N}_j}\boldsymbol{\alpha}_{jl2}^T(\boldsymbol{\omega}_{jl}-\mathbf{v}_l) \\
& + \frac{\eta}{2}\sum_{j=1}^{m}\sum_{l\in\mathcal{N}_j}||\mathbf{v}_j-\boldsymbol{\omega}_{jl}||^2 + \frac{\eta}{2}\sum_{j=1}^{m}\sum_{l\in\mathcal{N}_j}||\boldsymbol{\omega}_{jl}-\mathbf{v}_l||^2 \quad (4.2)
\end{aligned}$$

where the Lagrange multipliers $\boldsymbol{\alpha}_{jl1}$ and $\boldsymbol{\alpha}_{jl2}$ correspond to the constraints $\mathbf{v}_j = \boldsymbol{\omega}_{jl}$ and $\boldsymbol{\omega}_{jl} = \mathbf{v}_l$. The quadratic terms $||\mathbf{v}_j - \boldsymbol{\omega}_{jl}||^2$ and $||\boldsymbol{\omega}_{jl} - \mathbf{v}_l||^2$ ensure strict convexity, while parameter η allows for trading off speed of convergence against approximation error.

The distributed iterations that solve (4.2) are

$$\begin{aligned}
\{\mathbf{v}_j^{t+1},\boldsymbol{\xi}_j^{t+1}\} &= \operatorname*{argmin}_{\{\mathbf{v}_j,\xi_j\}}\mathcal{L}(\{\mathbf{v}_j\},\{\boldsymbol{\xi}_j\},\{\boldsymbol{\omega}_{jl}^t\},\{\boldsymbol{\alpha}_{jlk}^t\}) \\
\{\boldsymbol{\omega}_{jl}^{t+1}\} &= \operatorname*{argmin}_{\{\boldsymbol{\omega}_{jl}\}}\mathcal{L}(\{\mathbf{v}_j^{t+1}\},\{\xi_j^{t+1}\},\{\boldsymbol{\omega}_{jl}\},\{\boldsymbol{\alpha}_{jlk}^t\}) \\
\boldsymbol{\alpha}_{jl1}^{t+1} &= \boldsymbol{\alpha}_{jl1}^t + \eta(\mathbf{v}_j^{t+1} - \boldsymbol{\omega}_{jl}^{t+1}) \quad \forall j \in P, \forall l \in \mathcal{N}_j \\
\boldsymbol{\alpha}_{jl2}^{t+1} &= \boldsymbol{\alpha}_{jl2}^t + \eta(\boldsymbol{\omega}_{jl}^{t+1} - \mathbf{v}_l^{t+1}) \quad \forall j \in P, \forall l \in \mathcal{N}_j\,.
\end{aligned}$$

For details of how to simplify these iterations further and how to formulate the corresponding dual, see [FCG10]. In each iteration, each node $j \in P$ must solve a quadratic optimization problem that is similar to the original SVM problem, but the local datasets S_j on which it needs to be solved can be considerably smaller than the whole dataset $S = S_1 \cup \ldots \cup S_m$. After each iteration, nodes must communicate their local solutions v_j to each one-hop neighbor and then update their local multiplier vectors. Casting the SVM problem into the ADMM framework guarantees that after a finite number of iterations, the local solutions v_j at each node $j \in P$ equal the global solution $\mathbf{w}, b$ that would have been found by training a centralized SVM classifier on all data S.

As long as the number of iterations is smaller than the total number of training examples n, the algorithm communicates less than the whole dataset. On the MNIST

dataset used for evaluation in [FCG10], the algorithm only needs about 200 iterations for reaching a similar error as a centralized SVM.

Even although each local optimization problem is solved in its dual formulation, only primal weight vectors are exchanged between nodes. The non-linear case is thus much harder to solve, since direct application of the Φ transformation may lead to high-dimensional weight vectors and therefore to high communication costs. Direct application of the kernel trick is not possible. The authors of [FCG10] therefore propose to enforce consensus of the local discriminants on a subspace of reduced rank. This however requires preselected vectors common to all nodes, which introduce a subset of basis functions common to all local functional spaces. The choice of such vectors isn't necessarily straightforward and potentially requires the algorithm to be run again for each new classification query. Due to its complexity, the non-linear version is not discussed here. For further details, see [FCG10].

Distributed Dual Ascend [HMS08] presents a method for distributed SVM learning based on distributed dual ascend. Let $P_j(\mathbf{d})$ with $\mathbf{d} = \mathbf{0}$ be the solution of a standard linear SVM with zero bias trained on the data S_j at node j:

$$P_j(\mathbf{d}) = \underset{\mathbf{w}[j]}{\operatorname{argmin}} \frac{\lambda}{2m}||\mathbf{w}_j||^2 + \mathbf{w}_j^T\mathbf{d} + \frac{1}{m}\sum_{(\mathbf{x}_i,y_i)\in S_j} \max\{0, 1 - y_i\langle \mathbf{w}_j, \mathbf{x}_i\rangle\}\,.$$

The proposed distributed scheme uses $\mathbf{d} \neq \mathbf{0}$ for tying together the local results $\mathbf{w}_j$ while iterating over the local solutions $P_j(\cdot)$. At the beginning, the algorithm sets $\boldsymbol{\lambda}_j^{(0)} = \mathbf{0}$ and $\boldsymbol{\mu}_j^{(0)} = \mathbf{0}$. Then, in each iteration, each node j computes updates $\boldsymbol{\lambda}_j^{(t)} \leftarrow -\boldsymbol{\mu}_j^{(t-1)} - \frac{\lambda}{m}P_j(\boldsymbol{\mu}_j^{(t-1)})$ and passes its solution to a central node, which calculates $\boldsymbol{\mu}_j^{(t)} \leftarrow -\boldsymbol{\lambda}_j^{(t)} + \frac{1}{m}\sum_{j=1}^m \boldsymbol{\lambda}_j^{(t)}$ and communicates the solution back to each local node. The final output at iteration T is $\mathbf{w}^* = -\frac{1}{\lambda}\sum_{j=1}^m \boldsymbol{\lambda}_j^{(T)}$.

The algorithm is based on principles of Fenchel Duality (see [HMS08] for further details) and thus has a linear convergence rate, i.e. it takes $O(\log(1/\varepsilon))$ iterations to get ε-close to the optimal solution. With a linear kernel, only p scalars need to be transmitted in each iteration. The authors have also extended the algorithm to non-linear kernels. There, in the worst case, all components of vector $\boldsymbol{\alpha}$ need to be exchanged, meaning that each node j must transmit n/m scalars and receive all remaining αs in each iteration. Therefore, if $T > p$, the algorithm is not communication-efficient.

Distributed Block Minimization For the linear SVM, [PSJ11] rewrite the dual SVM problem (see Sect. 3.2.5) as

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \boldsymbol{\alpha}^T\mathbf{Q}\boldsymbol{\alpha}/2 - \mathbf{1}^T\boldsymbol{\alpha} \\ \text{s.t.} \quad & \forall_{i=1}^n : 0 \leq \alpha_i \leq C \end{aligned}$$

where $\mathbf{Q}_{uv} = y_u y_v \mathbf{x}_u \mathbf{x}_v$. [YHCL10] have shown that this problem can be used with sequential block minimization (SBM), i.e. that at each iteration t, only a single block

S_j of matrix $\mathbf{Q}$ is considered. The authors show that when solving for the variables in this block, the variables from other blocks don't need to be kept in memory. Suppose that $\boldsymbol{\alpha}^t$ is a solution after t iterations and that at $t+1$, the focus is on block S_j, with $\mathbf{d}^j = \boldsymbol{\alpha}^{t+1}[j] - \boldsymbol{\alpha}^t[j]$ being the direction for components of the $\boldsymbol{\alpha}$ vector that are associated with block S_j. Then, according to [PSJ11], $\mathbf{d}_j$ may be obtained by solving the optimization problem

$$\begin{aligned} \min_{\mathbf{d}_j} \quad & \mathbf{d}_j^T \mathbf{Q}[j,j]\mathbf{d}_j/2 + (\mathbf{w}^t)^T \mathbf{U}_j \mathbf{d}_j - \mathbf{1}^T \mathbf{d}_j \\ \text{s.t.} \quad & 0 \preceq \boldsymbol{\alpha}^t[j] + \mathbf{d}_j \preceq C\,, \end{aligned} \tag{4.3}$$

where $\mathbf{Q}[j,j]$ is a submatrix of $\mathbf{Q}$ consisting only of entries associated with the training examples in S_j and $\mathbf{U}_j$ is a $p \times |S_j|$ matrix where the i-th column is the i-th example in S_j, multiplied by its label y_i. For solving the problem, all that needs to be kept in memory are the training examples in S_j and the p-dimensional vector $\mathbf{w}^t$. After solving (4.3), $\mathbf{w}^t$ is updated as $\mathbf{w}^{t+1} = \mathbf{w}^t + \sum_{\mathbf{x}_i \in S_j} \mathbf{d}_j[i] y_i \mathbf{x}_i$.

The proposed distributed block minimization (DBM) with averaging scheme is then straightforward: Instead of processing each block S_j sequentially, they are all optimized in parallel. That is, given a central coordinator and j local nodes, per iteration t each node j solves (4.3) for S_j, sends $\Delta\mathbf{w}_j^t = \sum_{\mathbf{x}_i \in S_j} \mathbf{d}_j[i] y_i \mathbf{x}_i$ to the central coordinator and sets $\boldsymbol{\alpha}^{t+1}[j] = \boldsymbol{\alpha}^t[j] + 1/m \cdot \mathbf{d}_j$. The central coordinator then computes $\mathbf{w}^{t+1} = \mathbf{w}^t + 1/m \sum_{j=1}^m \Delta\mathbf{w}_j^t$ from the deltas received by each local node. The new vector $\mathbf{w}$ must then be transmitted to each local node, before a new iteration starts. The authors also discuss another variant than averaging, using line search for updating $\mathbf{w}$.

In each iteration, the algorithm communicates $O(mp)$ values. The authors argue that for a constant number of iterations, the communication complexity becomes independent from the number of training examples n. However, they haven't provided a proof of global convergence or for the rate of convergence. Empirically, the number of iterations needed to achieve sufficient accuracy on two different datasets was only 20. On both tested datasets, one proprietary from Akamai with 79M training examples and the other a public learning to rank dataset from Microsoft with 37M training examples, the algorithm achieves a higher accuracy than LIBLINEAR-CDBLOCK (see [YHCL10]) in a much shorter time.

Distributed Consensus for Outlier Detection Nearest neighbor approaches use distance to other points to compute an outlier. The widely used definition from [KNT00] says that outliers are those points which are *very far* from their nearest neighbors, according to some threshold. Many variants of this definition have been proposed, depending on the used distance measure and threshold. One practical definition uses Euclidean distance and a user defined threshold or the number of desired outliers. Such a definition has been used in [BSG+06] to find global outliers in WSNs. Each node j determines outliers in its local dataset S_j by a set of local rules and then broadcasts them to other nodes for validation. The neighboring nodes repeat the procedure until all

sensor nodes eventually agree on the global outliers. This technique can be flexible with respect to multiple existing distance-based outlier detection techniques. It has two major advantages: (1) it can be proven that the found outliers are the same as those found by a centralized algorithm, and (2) the algorithm can easily adopt to data and network changes. One drawback, however, is that the method requires a node to broadcast all found outliers to all the other nodes for validation, which may require a large amount of communication.

The distance-based technique proposed in [ZSGL07] identifies k global outliers in continuous query processing applications of sensor networks. It overcomes the broadcast issue of [BSG+06] by adopting the structure of an aggregation tree. Each node in the tree transmits some useful data to its parent after collecting all the data sent from its children. The root node then approximates the top k global outliers and sends them to all the nodes in the network for verification. If any node disagrees on the global results, it will send extra data to the root again. This procedure is repeated until all nodes agree on the global results. A major drawback of this technique is that it requires a tree topology to be overlaid on top of the network. Hence, it is not suitable for any kind of topology.

Distributed Consensus Clustering Generally, clustering algorithms situated in and developed for peer-to-peer networks are good candidates for use in WSNs, especially those that mostly rely on local computations and communication with a limited number of nearest neighbor nodes only. [DGK09] introduced two distributed variants of the k-Means algorithm (see Sect. 3.4.1). The first variant, LSP2P (Local Synchronized-Based P2P) k-Means, is based on a more general local algorithm for mining data streams in distributed systems [WBK09]. It carries out repeated iterations of a modified k-Means at each local node and collects newly calculated centroids and cluster counts only from its immediate neighbors to produce the centroids for the next iteration. Nodes terminate if these new centroids don't differ substantially from the old ones. The algorithm requires no global synchronization and can be extended to a dynamic environment, in which nodes enter and leave the network. Communication costs are shown to be independent of the number of observations to cluster and the total amount of communication is $O(mT(k + L))$, where m is the number of nodes, T the number of iterations, k the number of clusters and L the maximum number of neighbors. It was shown empirically that the algorithm yields similar accuracy as a centralized version of k-Means, however, proving convergence or bounds on the accuracy appears to be a hard problem.

A distributed expectation maximization algorithm for clustering data from a Gaussian mixture distribution, DEM, has been introduced in [Now03]. The algorithm particularly focuses on sensor networks. DEM utilizes an incremental version of the EM algorithm [NH99]. It repeatedly cycles through all nodes in a network and performs incremental E- and M-steps at each node, using only locally stored data and summary statistics passed from the previous node. DEM is guaranteed to converge to a local maximum and, as shown empirically, often more rapidly than the standard EM algorithm.

[Gu08] proposes to estimate the global sufficient statistics for the M-step by an average consensus filter, diffusing the local sufficient statistics over the entire network by communicating only with neighboring nodes. Thereby, each node gradually gains global information, until the parameters to estimate can be accessed from any node in the network. The local communication between neighbors which is inherently parallel makes it more run-time efficient than DEM which repeatedly cycles over all nodes in the network.

4.3.3 Fusion of Local Models

Techniques for the fusion of local models first learn models on the local data at each node j. Such local models are then broadcast to peer nodes or sent to a central coordinator, which fuse models in an intelligent way.

Fusion of Decision Trees The hierarchical decision tree classification technique proposed in [CXPL10] first constructs a spanning tree over all nodes in a network. The tree is built beginning with the leave nodes of the spanning tree, first building a decision tree on local data, then sending this decision tree upstream to the parent nodes in the spanning tree. The parent nodes of the spanning tree then build a new classifier by combining all the classifiers they have received from their children. Therefore, a portion of the dataset is subsampled with the same proportion of negative and positive examples. These intermediate nodes then send the classifiers again upstream and the root node (base station) builds a single classifier which represents all data over all the nodes. A short but wide spanning tree increases the communication cost of sending the classifiers to the next node, but reduces the overall accuracy due to a smaller number of hops. In comparison, a tall and narrow tree suffers from the opposite effect. The paper presents extensive experimental results on simulated wireless networks to show that this method offers better accuracy and energy consumption compared to a baseline ensemble method. Here, meta classifiers are learned independently at each node and then (majority) voting is applied during the test phase.

Incremental SVMs There exist several distributed SVM approaches for horizontally partitioned data that are based on the exchange of SVM models, e.g. the set of support vectors. Many of such approaches are inspired by early incremental versions of the SVM which repeatedly keep only the support vectors of previous learning steps for training. More sophisticated versions have demonstrated that although support vectors are not sufficient representations of a dataset, correct results can be achieved by exchanging support vectors in multiple iterations or by keeping other relevant data points. In following, the aforementioned incremental approaches are first described and then it is discussed how to extend them to distributed methods.

Instead of learning on a single batch of data, the incremental SVM by [SHKS99] assumes a training set S to be divided into disjunct subsets $S_1, \ldots, S_m$. The training procedure works incrementally. In the initial first step $t = 1$, the algorithm trains a

SVM on set S_1, but only keeps the support vectors SV_1. At each following time step t, an SVM model $\hat{f}_t$ is trained on the union $S_t \cup SV_{t-1}$ of the current training set S_t and the support vectors SV_{t-1} found in the previous step. Empirical results on UCI datasets suggest the incremental SVM achieves a similar performance as the standard batch SVM trained on all data up to the corresponding time step t, i.e. also at the end of training. The performance drops significantly if the SVM is only trained on a subset (90%) of the determined support vectors, from which the authors conclude that the incremental SVM finds a minimal set of support vectors. The authors further point out that the incremental SVM can be seen as a lossy approximation of the chunking method [OFG97], with the incremental approach considering each support vector only once.

The incremental procedure appears plausible as long as the distributions of training examples in each subset $S_1, \ldots, S_m$ are similar to the distribution of data points in the whole training set S. In cases where the training algorithm has full control over how S is split into subsets, the aforementioned condition could be achieved by a uniform sampling of examples from S. In a distributed setting, one could use a distributed uniform sampling algorithm like the one introduced in [DK07] to adjust for the skewness.

A more specialized solution for the case where the statistical properties of each batch $S_1, \ldots, S_m$ may differ from those of S has been investigated, among others, by [RÖ1]. In contrast to detecting concept drift, the focus is on learning a single concept from all data. However, though training examples in each batch consistently represent that concept, their distributions differ. The author notes that while support vectors provide a condensed and sufficient description of the learned decision boundary, they do not represent the examples themselves. That is, in terms of empirical risk minimization, the support vectors provide an estimate of $P(X|Y)$, but not of $P(X)$. If the number of support vectors is small in comparison to the number of examples in the next batch, their influence on the decision boundary will be small. It is demonstrated how decision boundaries can differ between an SVM trained on all data and one trained on a subset of the data, with SVs from another subset added. The few support vectors are treated as mere outliers, which the SVM is known to be robust against. Therefore, [RÖ1] proposes to weight prediction errors on support vectors higher than errors on training examples in the new batch by replacing the original primal SVM objective (see Sect. 3.2.5) with

$$\min_{\mathbf{w}} \quad \frac{1}{2}||\mathbf{w}||^2 + C\left(\sum_{i \in S} \xi_i + L \sum_{i \in SV} \xi_i\right),$$

where S is the set of new training examples, SV is the set of old support vectors and $L = 2\frac{n}{|SV|}$. It is shown that the modified incremental algorithm empirically achieves a higher accuracy than the plain version proposed in [SHKS99].

Iterative Exchange of Support Vectors While [RÖ1] gives a counter example which shows that local sets of support vectors may differ strongly from the global set of support vectors, [CSH00a] includes a formal proof. Like [BC00], the authors further show that

points on the convex hull provide a sufficient statistic for SVM learning from distributed data sources. However, for higher dimensions, computing the convex hull is exponential and thus not efficient.

In [CCH05], the same authors propose a scheme that is, according to their reasoning, efficient and exact. The idea consists of exchanging support vectors iteratively with a central node. At iteration t, each local site j determines its current set of support vectors SV_j^t, based on the dataset $S_j \cup GSV^{t-1}$, where S_j is the local data stored at node j and GSV^{t-1} is the global set of support vectors j has received from the central node at the previous iteration $t-1$. Each node sends its local set of support vectors to the central node, which merges them to the global set GSV^t and communicates it to all local nodes. The authors sketch a proof which shows that after a finite number of iterations, the local sets of support vectors converge to the globally optimal set of support vectors.

Cascade SVM The Cascade SVM introduced in [GCB+05] is based on a similar idea as the previously presented incremental SVMs, which is to identify and eliminate non-support vectors as early as possible and only communicate support vectors between distributed nodes. Proposed is a hierarchical binary tree topology of cascading SVMs. At the beginning, disjunct subsets $S_1, \ldots, S_m$ of S are distributed over the leaves of the tree. An SVM is trained at each leaf and the resulting support vectors are communicated to the parent node in the next layer of the hierarchy. At the parent nodes, SVMs are trained on unions of support vectors of the previous layer. The root node communicates the finally determined support vectors to each leaf, and each leaf decides if any of its input vectors would become new support vectors. When the set of support vectors has stabilized over all leaves, the algorithm stops, otherwise the hierarchical cascade is traversed again. The algorithm thus includes a similar feedback loop as the approach proposed in [CCH05], but due its hierarchical design, it may earlier filter data points that are not in the global set of support vectors.

It is proven that the Cascade SVM converges to a set of support vectors that is globally optimal in finite time. However, no bound is given for the total number of iterations. For the standard datasets tested, like the MNIST dataset, the authors report a low number of iterations between 3 and 5. Moreover, with 16 machines in a cluster, the final number of support vectors and the size of subsets at each leaf is about 16 times smaller than the total number of 1M data points. Training time was reduced from about one day on a single machine to one hour with the distributed approach.

In practice, run-time and communication costs will largely depend on the ratio of training examples to support vectors. Often, bad choices of hyperparameters, like C or σ for the RBF kernel, result in an unnecessarily large number of support vectors. Unfortunately, optimal hyperparameters are hard to determine in advance, but must be found experimentally. Similarly, a high number of support vectors can be expected for complicated non-linear decision boundaries.

[LR06] prove and demonstrate that the iterative exchange of support vectors converges to the global optimum also in case of other network topologies. Particularly, they

train local SVMs and exchange support vectors with their ancestors and descendants in a strongly connected network. It is shown that the binary cascade proposed in [GCB+05] is a special case of a strongly connected network and that a random strongly connected network topology may lead to faster convergence. A ring topology has the slowest convergence. Further tested are synchronous and asynchronous versions of the algorithm. The synchronous implementation dominates in terms of training time, while the asynchronous version leads to less data accumulation (number of exchanged support vectors) in sparser networks.

Energy-efficient Distributed SVMs In [FBLT06], the incremental procedure proposed in [RÖ1] is brought into the context of distributed wireless sensor networks. Subsets of S are assumed to be stored at cluster heads. Such cluster heads may be determined by already existing energy-efficient clustering network protocols (see Sect. 4.2). Similar to the original incremental algorithm, models are consecutively trained on the data S_j at cluster head j and support vectors received from the previous cluster head in a chain of cluster heads. The authors regard varying distributions of observations by a different weighting of examples and support vectors, as already proposed in [RÖ1]. Empirically the algorithm is shown to be similarly accurate, but more energy-efficient than transmitting all data to a central node and training a single SVM on all data. However, in comparison to [CCH05, GCB+05], the algorithm is not guaranteed to find a globally optimal solution. Moreover, it was only evaluated on a single synthetic two-dimensional dataset consisting of two Gaussian distributions. Like the Cascade SVM, the communication costs will very much depend on the number of support vectors found. Here, it may happen that cluster heads at the end of the chain always receive more support vectors than those at the beginning. Balancing the network's total energy consumption would thus require a technique for changing the order of communication dynamically.

For solving the last problem, in [FBLT08] the same authors propose two gossip algorithms that exchange summary information between one-hop neighboring nodes. A single iteration of the minimum selective gossip algorithm (MSG-SVM) at time step t consists of training SVMs at each node, based on the currently available local information. Each node then communicates its current set of support vectors to all one-hop neighbors and all nodes update their current model at time step $t + 1$. Although the authors give no explicit stopping criterion for their algorithm, they argue that over time, all nodes will converge to the same SVM model. However, they also argue that their algorithm is sub-optimal and will not converge to the same solution as a centralized SVM trained on all data. The idea of this proof is based on the same argument as already given in [CSH00a]. It remains unclear if between iterations, nodes only keep the determined support vectors or if exchanged support vectors are added to the already available local data points. In the first case, data points that might later become support vectors could be thrown away and would thus be missed. In the second case, however, the iterative exchange of support vectors closely resembles the filtering mechanism and feedback loops of the approaches introduced in [CCH05] and [GCB+05], which both

converge to the global optimum. The second proposed strategy, the sufficient selective gossip algorithm (SSG-SVM), ensures convergence to the global optimum by exchanging points that lie on the convex hull of each class. While this algorithm might work efficiently on the synthetic two-dimensional datasets used for evaluation, it is inefficient for higher dimensions (see [CCH05]).

In [FBLT09], the authors propose to trade-off communication costs for accuracy by exchanging only a pre-determined percentage of observations between neighboring nodes. The observations to be transmitted by each node are ranked by their distance from the current determined local hyperplane. For a single type of synthetic data, consisting of two 2-dimensional Gaussians, the authors demonstrate that accuracy can be increased by transmitting slightly more observations than only the support vectors. While the algorithm allows for trading off communication costs for accuracy, it remains unclear how much accuracy decreases if much less is sent than the set of support vectors. Since the number of support vectors may be high, it appears somewhat questionable that any of the aforementioned approaches could truly work in highly energy-constrained systems like WSNs.

Distributed Outlier Detection Given examples of two kinds, an outlier detection problem can be transformed to a classification problem. This trick has been widely explored in the data mining community [CBK09]. The classification techniques presented here may thus as well be applied for outlier detection in WSNs. In [RLPB07], the 1-class SVM (see Sect. 3.3.1) is used for outlier detection. First, a local model is trained at each node. Then, points lying outside the decision boundary are labeled as outliers and sent to a central coordinator, along with the model. The local outliers are then validated and the global set is determined.

Fusion of Distributed Local Cluster Models The DMBC (Distributed Model-Based Clustering) algorithm proposed in [KKPS05] assumes a Gaussian mixture distribution. It first estimates the number of Gaussian clusters, their parameters (mean and covariance matrix) and their weights at the local nodes, using the standard EM algorithm. Then, the local parameters and weights are transferred to a central coordinator, where similar Gaussians are joined to a compact global distribution. The similarity is measured as the *mutual support* between two clusters C_u, C_v, which in addition to their mean vectors also considers the variance of the clusters. For high dimensional data, DMBC assumes attributes to be independent of each other, resulting in a reduction of the $p \times p$ covariance matrices to p-dimensional variance vectors. For m nodes and a maximum number of local clusters k, the total communication costs are thus bounded by $O(mk)$. It was shown empirically, for varying numbers of clusters and nodes, that the clustering found by DMBC is highly similar to a central clustering, as measured by the Rand Index.

Further algorithms exist, like a distributed version of density-based clustering [JKP04], spectral clustering [SSB12] and solutions specialized on particular applications, like spa-

tial [MS06] and time series clustering [YG08].

Only few algorithms for data clustering developed so far are truly resource-aware and consider, for example, the residual energy of nodes or the CPU utilization explicitly. An exception is ERA-cluster, proposed in [PGR07], which is based on the concept of microclusters [AHWY03]. It can automatically adapt its sampling rate and the number of examined microclusters based on the current battery, memory and CPU utilization as measured by a resource monitor. EDISKCO [HMS09] solves the k-center clustering problem and can also determine outliers. It works incrementally and only needs a single pass over the input observations, without storing them. The local nodes keep a special heap structure for storing their local k centers and z outliers, sorted according to cluster counts. If a new point doesn't fit the current clustering, a request for increasing the radius is sent to a coordinator. The coordinator replies with the biggest radius it has received from all other nodes. The local nodes maintain their heap such that the effect of the most l dense clusters which appeared in history solutions is kept, but also such that space is left for establishing new clusters if there is a new trend in the input stream. The coordinator receives the local solutions C_j, radii R_j and radius increase requests from the nodes. It continuously performs the Furthest Points algorithm on the solutions C_j and keeps the largest radius received from all nodes. The base station (server side) rotates the coordinator according to an estimate of the residual energy in each node. EDISKCO determines a $(4+\varepsilon)$-approximation of the optimal global clustering. Empirically, it was shown that the algorithm outperforms the centralized Global Parallel Guessing algorithm that was proposed in [CMZ07], with regard to accuracy as well as energy consumption.

4.3.4 Fusion of Local Predictions

Instead of fusing local models, it also possible to train local models across nodes $j = 1, \ldots, m$ and then fuse only their predictions for final classification. A popular fusion rule is majority vote, or in the case of probabilistic classifiers, the maximum a posteriori (MAP) criterion (see comments on the optimal Bayes classifier in Sect. 3.1.2).

Collaborative Target Classification A classic application of WSNs is multi-vehicle tracking and classification. Collaborative techniques bolster the inference of one node using the posterior of the other node. If one node can validate a hypothesis, then it makes sense to use it for subsequent inferencing rather than starting from scratch for each node. In [MNR02], the idea is used for the identification and classification of vehicle types from a convoy of vehicles. Using confidence boosting, which uses the posterior of one node to do inference on the next node, the classification accuracy increases by 7%, while a collaborative data driven approach boosts the accuracy by 9%. Finally, the paper shows how collaborative mining techniques can help in identifying and isolating the effects of multiple vehicles which is itself a very hard problem due to signal interference.

A similar approach is discussed in [DS03], introducing the concept of collaborative signal processing (CSP). Two forms of CSP are discussed in the paper: (1) *data fusion*: which exchanges low dimensional feature vectors between the correlated nodes for op-

timal network performance, and (2) *decision fusion*: which exchanges likelihood values among the independent nodes. The latter one is preferred in WSNs due to its low computational and communication overhead. The paper studies CSP algorithms for single target classification based on multiple acoustic signals measured at different nodes. One of the ways sensor networks can save power is by using a region-based processing instead of all nodes communicating to each other. A manager node is assigned to each region, coordinating the communication among the nodes in its region and across regions. In this model, single target classification consists of the following steps: (1) *target detection and classification*: the first step is to use CSP algorithm to detect the region in which the target is, and designate it as the active region, (2) *target localization*: this step is used by the manager nodes to localize the target using the energy detected at each node, (3) *target location prediction*: past estimates are used by the manager nodes to predict future values, and (4) *active location determination*: when the target becomes close to any other region, that region is designated as the new active region and this process is repeated. The paper studies three classifiers: An optimal maximum likelihood classifier, a data averaging classifier that treats all measurements as correlated, and a decision-fusion classifier that treats each observation as independent. Experimental results on DARPA SensIT program data show that the sub-optimal decision fusion classifier is the most attractive model in a WSN context.

Distributed Vehicle Classification The application of vehicle classification in WSNs is discussed in [DH04]. Each sensor is equipped with a microphone or a geophone. Upon detection of a vehicle in the vicinity of the sensor, the on-board processor first extracts features in the frequency domain using a Fast Fourier Transform (FFT). The next step is to use a local classifier at each node to generate a preliminary hypothesis about the observation using only the data present at that node. The authors have experimented with three classifiers: A k-NN based classifier, a maximum likelihood classifier, and an SVM classifier. The local decision, together with the estimated probability of being a correct decision is transmitted to a central coordinator, which can then use MAP to compute the final classification. Extensive experimental results show that the MAP estimate with the nearest neighbor as the local classifiers works well in vehicle classification.

Distributed Outlier Detection The technique presented in [BHL07] detects outliers in WSNs for ecosystem monitoring applications. The method exploits the spatiotemporal distribution of data to find outliers. The basic idea is to compare the measurement of one sensor with those in the spatial vicinity and also with its measurements back in time. Then, if the deviation of these values are greater than a user defined threshold (based on a statistical significance test), a sensor detects an outlier. The obvious drawback of this method is the choice of the outlier.

4.3.5 Summary

In the previous sections, we have seen that distributed data analysis algorithms for the horizontally partitioned data scenario use different techniques and architectures to achieve their goals, which are either speed improvement or the reduction of communication costs.

Distributed algorithms that preprocess data locally and send the new representation to a central coordinator share the work load of preprocessing and can be communication-efficient. How much is communicated depends very much on the preprocessing techniques used. The least squares regression SVM sends only $O(p^2)$ values (partial sums) per node, however, only for the linear kernel. Statistical outlier detection techniques represent local data compactly by histograms or the median of sensor readings. The accuracy of such reductions depends then very much on the underlying data distribution.

Consensus algorithms share the work load during training and can be communication-efficient if the number of shared variables and iterations T are not too large. For instance, the distributed consensus SVM transmits only $p+1$ scalars in each iteration, and its total communication costs in experiments on the MNIST dataset are considerably smaller than, for instance, those of the Cascade SVM which transmits about 60k p-dimensional support vectors. Regarding convergence rates from a theoretical point of view, ADMM is known to have slow convergence, while the dual ascend approach converges in $O(\log 1/\varepsilon)$ steps to a global optimum. Distributed block minimization has no proof for the convergence rate. However, all algorithms show good performance in practice, with regard to prediction performance as well as the number of iterations. The communication costs of the clustering algorithms presented are similarly independent of the number of training examples, and mostly depend on convergence rate. It should be noted that in the case of highly limited resources, iterations could be artificially restricted to a fixed number, though it might come at the expense of accuracy or losing guarantees.

The communication costs of techniques which fuse models trained on local data obviously depend on the complexity of such models. As explained in Sect. 3.1.5 on the structural risk minimization principle and in Sect. 3.1.6 on the bias variance trade-off, more complex models don't necessarily minimize the true error. The complexity of models thus depends on how well they generalize, which in turn is highly dependent on the underlying data distribution. Non-linear decision boundaries in many cases require more complex hypotheses for their description. In the dual SVM formulation, more complex models consist of a large number of support vectors. Therefore, distributed SVM algorithms which exchange support vectors iteratively among nodes, like the Cascade SVM, aren't necessarily communication-efficient, especially for wrongly chosen hyperparameters. In comparison, the presented methods for fusing cluster models estimate only a fixed number of parameters per cluster. They are thus communication-efficient if the number of clusters is not too large. However, for high-dimensional data, DMBC needs to assume that features are independent from each other, since otherwise, covariance matrices would become too large for transferal.

Methods which fuse only the predictions of local models trained on horizontally

partitioned datasets are most communication-efficient during training. Since all models can be trained independently from each other, no data needs to be transmitted at all. The communication costs for the prediction step depend on how many observations are to be predicted. Depending on model size, if only a few observations need to be predicted, it might be most beneficial to broadcast them to all local nodes, receive predictions for them and apply the fusion rule. However, this is not communication-efficient, since all feature values need to be transmitted for each observation. If the local models are small enough, a better strategy can be to broadcast local models once to all other nodes or a central coordinator, such that local models and the decision rule can be applied directly where observations are acquired.

Remarks should be made on how the skewness of local datasets S_j, i.e. their deviation from the global data distribution of S (see also Sect. 2.4), influences accuracy and communication costs. Training of a global model on condensed representations of local data shouldn't be influenced by the skewness, since accuracy and communication costs only depend on the quality of representation. In the case of consensus and model fusion techniques, it depends. If model parameters estimate the data distribution, they might be looked at as condensed data representations, and skewness shouldn't have any influence again. If model parameters describe a decision boundary, however, the deviance of locally determined boundaries increases with the skewness of local datasets. For the original incremental SVM [SHKS99] this can lead to low accuracy. For iterative methods, like the Cascade SVM or distributed consensus SVM, it means that the number of iterations increases, as more information needs to be exchanged until consensus on a global model is reached. Communication costs increase. The accuracy of methods which fuse predictions of local models may suffer if local datasets represent the global data distribution badly. A solution might be to give such bad predictors lower weight during fusion. Weights can be determined, for instance, by assessing the accuracy of local classifiers on a hold-out test sample.

Finally, it should be noted that only very few algorithms exist which explicitly take the resource limitations of sensor nodes or WSNs into account. From the selection of algorithms, only ERA-cluster and EDISKCO account for the current battery, memory and CPU utilization or the residual energy of nodes. EDISKCO further uses similar techniques as sensor node clustering, like the rotation of nodes's responsibilities, to save energy and increase the network's overall lifetime. Incorporating techniques from sensor node clustering or integrating data analysis with such algorithms could thus provide for many new research opportunities.

4.4 Vertically Distributed Data Analysis Algorithms

In the *vertically partitioned data scenario* [CSH00b], each node stores only partial information about observations, i.e. subsets of their features $A_1, \ldots, A_p$, but for all observations. Let $\mathbf{x}[j] \in \mathbb{R}^{p_j}$ denote a vector which contains p_j features of observation $\mathbf{x}$ available at node j. Columns of the data matrix $\mathbf{D}$ are thus split over the nodes, i.e.

each node j stores a $n \times p_j$ submatrix $\mathbf{D}[j]$ whose rows consist of vectors $\mathbf{x}_1[j], \ldots, \mathbf{x}_n[j]$. An individual feature q stored at node j can then be denoted by $\mathbf{x}[j][q]$.

As already discussed in Sect. 2.4, distributed learning in the vertically partitioned data scenario can be particularly challenging, since each $\mathbf{D}_j$ constitutes a subspace of the entire data matrix $\mathbf{D}$ and local features all by themselves might not contain enough information about the target concept.

There exist some algorithms for the scenario, presented in the following. Only few are communication-efficient. Algorithms developed in the context of privacy-preserving data mining preprocess data locally and combine the new representations at a central coordinator (see Sect. 4.4.1). Consensus models iteratively reach consensus on a set of variables, in this case the predictions of labels (see Sect. 4.4.2). Hybrid solutions combine local and global models (see Sect. 4.4.4).

After having presented all methods, their properties will be summarized in Sect. 4.5.5 and particular challenges of the scenario will be discussed in relation to the development of new algorithms, like those that are going to be developed in this thesis.

4.4.1 Local Preprocessing, Central Analysis

One way to deal with vertically partitioned data is to process each dataset locally and send the new representation to a central coordinator for further analysis. Transforming data into a new representation is especially important when its privacy needs to be preserved, as it is not allowed to communicate the original raw data between nodes.

Privacy Preserving SVMs [YVJ06, MWF08, YLG09] present privacy-preserving SVMs that are mainly based on the communication of kernel matrices. A central observation in each work is that entries of the $n \times n$ kernel matrix $\mathbf{K}$ are separable in the sense that

$$k(\,[\,\mathbf{E}\,\mathbf{F}\,]\,,\,[\mathbf{G}\,\mathbf{H}]^T\,) \;= k(\,\mathbf{E},\,\mathbf{G}^T\,) \,+\, k(\,\mathbf{F},\,\mathbf{H}^T\,) \quad \text{or} \tag{4.4}$$

$$k(\,[\,\mathbf{E}\,\mathbf{F}\,]\,,\,[\mathbf{G}\,\mathbf{H}]^T\,) \;= k(\,\mathbf{E},\,\mathbf{G}^T\,) \,\odot\, k(\,\mathbf{F},\,\mathbf{H}^T\,) \tag{4.5}$$

where $k : \mathbb{R}^{n\times p} \times \mathbb{R}^{p\times n} \to \mathbb{R}^{n\times n}$ denotes the kernel function for whole matrices, $+$ denotes standard addition and $\odot$ denotes the Hadamard componentwise product of two matrices with same dimensions. In [MWF08] it is shown that the linear dot product kernel $k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$ satisfies (4.4), while the RBF kernel (3.24) satisfies (4.5). Moreover, separability can be extended to polynomial kernels (3.23).

In a distributed setting, $\mathbf{D}[j]$ is the $n \times p_j$ data matrix whose rows consist of the (partial) training examples S_j at each local node j and $\mathbf{D}$ the $n \times p$ data matrix for the whole dataset S. Given kernel matrices $\mathbf{K}_1, \ldots, \mathbf{K}_m$ with entries of the linear kernel for data matrices $\mathbf{D}[1], \ldots, \mathbf{D}[m]$ at m different nodes, the global kernel matrix $\mathbf{K}$ for $\mathbf{D}$ can be calculated as

$$\mathbf{K} = \mathbf{K}_1 + \cdots + \mathbf{K}_m = \mathbf{D}[1]\mathbf{D}[1]^T + \cdots + \mathbf{D}[m]\mathbf{D}[m]^T\,.$$

In [YVJ06], it is proposed that each local node j first calculates its local kernel matrix $\mathbf{K}_j$. Each node might then send $\mathbf{K}_j$ to a central coordinator, which builds $\mathbf{K}$ and trains a centralized SVM on the full kernel matrix as usual. According to the authors, this scheme preserves the privacy of each local data matrix $\mathbf{D}[j]$, since it doesn't reveal the original attribute values. For added privacy, i.e. not even revealing the entries of the local kernel matrices, the authors propose an extended scheme with a secure addition mechanism (for details, see [YVJ06]). There, m nodes communicate in a ring topology where each node sends an $n \times n$ matrix to the next node and then back to the first node.

A slightly different approach is followed in [MWF08]. There, it is proposed to replace the standard kernel by a reduced kernel $k(\mathbf{D}, \mathbf{B}^T) : \mathbb{R}^{n \times p} \times \mathbb{R}^{p \times \tilde{n}} \to \mathbb{R}^{n \times \tilde{n}}$, where $\tilde{n} < n$ and $\mathbf{B}$ is a random matrix. The $\tilde{n}$ columns of the random matrix are privately generated in m blocks corresponding to the m nodes which hold the corresponding feature values in their local data matrices $\mathbf{D}[1], \ldots, \mathbf{D}[m]$. Each node communicates its reduced local kernel matrix to a central coordinator, which reconstructs the global (reduced) kernel matrix $\mathbf{K}^r$ according to (4.4) or (4.5) and then trains a centralized SVM based on $\mathbf{K}^r$ as usual. It is empirically shown for several standard datasets that learning with the reduced kernel matrix achieves a similar error rate as a centralized SVM trained on the full kernel matrix.

The authors of [YLG09] propose a similar scheme as [YVJ06] and [MWF08], but argue that the secure addition procedure proposed in [YVJ06] or the reduced kernel are not necessary for the preservation of privacy. Instead, local kernel matrices could be sent directly to a central coordinator.

While all of the aforementioned approaches preserve the privacy of each local dataset, according to their authors, only [MWF08] may improve the total run-time, due to a reduced kernel matrix. None of the approaches is communication-efficient for most practical purposes. The data matrix $\mathbf{D}[j]$ at node j consists of $n \times p_j$ real values and each kernel matrix $\mathbf{K}_j$ of $n \times n$ (or $n \times \tilde{n}$) values. Only if $p_j > n$ (or $p_j > \tilde{n}$), less data is sent than transmitting the original data matrices $\mathbf{D}[j]$ to a central node. However, usually $p_j \ll n$, especially since the total number of features p is split among m different nodes.

Ridge Regression with Random Projections In [HMM16] propose DUAL-LOCO, which is a dual variant of their distributed ridge regression algorithm for vertically partitioned data, LOCO [HMMK14]. The algorithm uses random projections to reduce the number of features which need to be transmitted to worker nodes.

For random projection, the authors use the Subsampled Randomized Hadamard Transform (SRHT). The projection matrix has the form $\mathbf{\Pi} = \sqrt{\tau/\tau_{subs}}\mathbf{XHS}$, with $\mathbf{S} \in \mathbb{R}^{\tau \times \tau_{subs}}$ being a subsampling matrix, $\mathbf{X} \in \mathbb{R}^{\tau \times \tau}$ being a diagonal matrix whose entries are drawn independently from $\{-1, 1\}$ and $\mathbf{H}$ being a normalized Walsh-Hadamard matrix. The product between $\mathbf{\Pi}^\top$ and some vector $\mathbf{u} \in \mathbb{R}^\tau$ can be computed in $O(\tau \log \tau)$ time, while never constructing $\mathbf{\Pi}$ explicitly.

Random projections have been used before to reduce the dimensionality of the data

Algorithm 3 The DUAL-LOCO algorithm

procedure DUALLOCO($\mathbf{D}, \mathbf{Y}, m, \tau_{subs}, \lambda$)
  Partition $\mathbf{D}$ into m submatrices $\mathbf{D}[1], \ldots, \mathbf{D}[m]$ of equal dimension τ.
  for worker $1, \ldots, m$ **do** ▷ in parallel
    Compute and send random features $\mathbf{D}[j]\Pi[j]$.
    Receive random features and construct $\tilde{\mathbf{D}}[j]$.
    $\tilde{\boldsymbol{\alpha}}_j \leftarrow$ LOCALDUALSOLVER($\tilde{\mathbf{D}}, \mathbf{Y}, \lambda$)
    $\hat{\boldsymbol{\beta}}_j = -\frac{1}{n\lambda}\mathbf{D}[j]^\top \tilde{\boldsymbol{\alpha}}_j$.
    Send $\hat{\boldsymbol{\beta}}_j$ to central coordinator.
  end for
  Construct solution vector $\hat{\boldsymbol{\beta}} = \left[\hat{\boldsymbol{\beta}}_1, \ldots, \hat{\boldsymbol{\beta}}_m\right]$
end procedure

before performing regression. The disadvantage of using random projections in the primal formulation of ridge regression is that the solution vector is in the compressed space and therefore hard to interpret. Instead of solving the primal problem, the authors propose to solve the dual optimization problem

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^n} -\sum_{i=1}^{n} f_i^*(\alpha_i) - \frac{1}{2\eta\lambda}\boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha}\,, \tag{4.6}$$

where f^* is the conjugate Fenchel dual of f, $\lambda > 0$ and $\mathbf{K} = \mathbf{D}\mathbf{D}^T$. Using the squared loss function $f_i(u) = \frac{1}{2}(y_i - u)^2$, one obtains $f_i^* = \frac{1}{2}\alpha^2 + \alpha y_i$. The primal solution has then the form

$$\boldsymbol{\beta}^*(\boldsymbol{\alpha}^*) = -\frac{1}{n\lambda}\mathbf{X}^\top \boldsymbol{\alpha}^*\,. \tag{4.7}$$

By defining $\tilde{\mathbf{K}} = (\mathbf{D}\Pi)(\mathbf{D}\Pi)^\top$, we obtain the dual in projected space as

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^n} -\sum_{i=1}^{n} f_i^*(\alpha_i) - \frac{1}{2\eta\lambda}\boldsymbol{\alpha}^\top \tilde{\mathbf{K}}\boldsymbol{\alpha}\,. \tag{4.8}$$

Under mild assumptions on the loss function, the solution of this problem, $\tilde{\boldsymbol{\alpha}}$, can be mapped back to the original space as $\tilde{\boldsymbol{\beta}}(\tilde{\boldsymbol{\alpha}}) = -\frac{1}{n\lambda}\mathbf{D}^\top \tilde{\boldsymbol{\alpha}}$, which is a good approximation of the solution to the original primal ridge regression problem.

The procedure proposed by the authors is shown in Alg. 3. For the derivation of the local solver, see [HMM16]. The matrices of random features $\tilde{\mathbf{D}}[j]$ communicated to other nodes have a dimension of τ_{subs} and are therefore much smaller than the original local data matrices. The algorithm has therefore communication costs of $O(m\tau_{subs})$.

The authors can show that the recovery error between the solution of DUAL-LOCO and the solution of the primal optimization problem is bounded. As we will see in Sect. 4.5, a problem for distributed learning from vertically partitioned data is taking into account the conditional dependencies of features residing at different nodes, given the label. The authors demonstrate in experiments that their algorithm does respect such dependencies.

4.4.2 Model Consensus

As already discussed in Sect. 4.3.2, consensus models exchange information between nodes until the values of shared variables converge. In the following, first a distributed algorithm for least squares regression is presented. Then, it is discussed how the SVM with a linear kernel can be cast into an ADMM formulation.

Co-Regularised Least Squares Regression In the context of a semi-supervised learning setting, the authors of [BGSW06] consider the problem of *M-view learning*. The goal is to find M functions from different Hilbert spaces $\mathcal{H}_v$ such that the error of each function on sample S and the disagreement between the functions on the unlabeled data is small. In the problem posed, $M \geq 1$, and the instances described by different views may differ. Given M finite sets of training instances $S_v \subseteq X$, $|\cup_{v=1}^{M}|$ labels $y(x) \in \mathbb{R}$, and a finite set of unlabeled instances $S \subseteq X$, $\hat{\mathbf{f}} = (\hat{f}_1, \ldots, \hat{f}_M) \in \mathcal{H}_1 \times \cdots \times \mathcal{H}_M$, $f_v : X \to \mathbb{R}$ functions shall be found that minimize

$$Q(\hat{\mathbf{f}}) = \sum_{v=1}^{M} \left[\sum_{x \in S_v} \ell(y(x), \hat{f}_v(x)) + \nu ||f_v(\cdot)||^2 \right] + \lambda \sum_{u,v=1}^{M} \sum_{z \in S} \ell(\hat{f}_u(z), \hat{f}_v(z)) \,, \tag{4.9}$$

where the norms are in respective Hilbert spaces and λ is a parameter weighting the influence of pairwise disagreements between labels. By means of the representer theorem, solutions can be expressed in terms of kernels as

$$\hat{f}_v^* = \sum_{x \in S_v \cup S} c_v(x) k_v(x, \cdot) \,, \tag{4.10}$$

where $k_v(\cdot, \cdot)$ is the reproducing kernel of the Hilbert space $\mathcal{H}_v$. The vector of functions $(\hat{f}_v(x_1), \hat{f}_v(x_2), \ldots)^\top_{x_i \in S_v \cup S}$ can then be written as $\mathbf{K}_v c_v$, and $||\hat{f}_v(\cdot)||^2$ as $c_v^t \mathbf{K}_v c_v$, where $[\mathbf{K}_v]_{ij} = k_v(x_i, x_j)$ and $[c_v]_i = c_v(x_i)$. $\mathbf{K}_v$ forms a strictly positive definite kernel matrix. Further, in the following, $\mathbf{y}_v = (y(x_1), y(x_2), \ldots)^t_{x_i \in S_v}$.

With the squared loss $\ell(\mathbf{y}, \hat{\mathbf{y}}) = (\mathbf{y} - \hat{\mathbf{y}})^2$, which turns the optimization problem into a ridge regression problem (also known as regularized least squares regression), it is possible to rephrase (4.9) as the exact non-parametric coRLSR problem. However, the problem has cubic run-time complexity not only in the number of labeled examples, but also unlabeled examples. Since last number is assumed to be large, solving the problem would need much time. In the following, only the formulation of a semi-parametric approximation to the problem is presented, as formulated by the authors of [BGSW06], which leads to a faster algorithm.

Given for each view $v \in \{1, \ldots, M\}$ a strictly positive definite matrix $\mathbf{L}_v \in \mathbb{R}^{n_v \times n_v}$, where n_v is the number of training instances in view v, and an arbitrary matrix $\mathbf{U}_v \in \mathbb{R}^{m \times n_v}$. For fixed $\lambda, \nu \geq 0$, the *semi-parametric coRLSR* optimization problem is to minimize

$$Q(\mathbf{c}) = \sum_{v=1}^{M} \left[||\mathbf{y}_v - \mathbf{L}_v c_v||^2 + \nu c_v^t \mathbf{L}_v c_v \right] + \lambda \sum_{u,v=1}^{M} ||\mathbf{U}_u c_u - \mathbf{U}_v c_v||^2 \tag{4.11}$$

Algorithm 4 Distributed CoRLSR

procedure DISTCORLSR($\mathbf{L}, \mathbf{U}$)
  repeat
    for each view v sequentially **do**
      $c_v \leftarrow \mathbf{G}_v^{-1}\left[\mathbf{L}_v^t y_v + 2\lambda \mathbf{U}_v^t \sum_{u \neq v} \hat{\mathbf{y}}_u\right]$
      $\hat{\mathbf{y}}_v \leftarrow \mathbf{U}_v c_v$
      send $\hat{\mathbf{y}}_v$ to all
    end for
  until convergence
end procedure

over $\mathbf{c} = (c_1, \ldots, c_M) \in \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_M}$. For details on the form of $\mathbf{L}_v$ and $\mathbf{U}_v$, see [BGSW06]. The authors proof that this problem can be solved in time $O(M^3 n^2 m)$, where $n = \max_v n_v$.

The authors then show that the semi-parametric coRLSR problem can be solved with an iterative distributed algorithm, using block coordinate descent, which iteratively only communicates the predictions of each site for the unlabeled data (see Alg. 4, for matrix $\mathbf{G}_v$, see [BGSW06]). The idea behind this approach is that different companies may have similar prediction problems, but don't want to share the data or the predictions on them. What they could share safely, however, is appropriately generated synthetic data, exchanging their predictions on this set of unlabeled data, increasing prediction performance.

It is shown that the algorithm converges in $\lceil \log_\delta 1/\varepsilon \rceil$ iterations to achieve an error reduction factor of at least ε, where Δ is a factor depending on the largest and smallest eigenvalue of the Hessian. Therefore, communication costs are $O(Mm\lceil \log_\Delta 1/\varepsilon \rceil)$ for broadcasting the labels, which are scalar values.

If we take the views to be subspaces of the whole data matrix $\mathbf{D}$, the distributed algorithm learns from vertically partitioned data.

Vertically Distributed Consensus SVM [BPC+11] casts the vertically distributed SVM problem into the ADMM framework. The general problem of model fitting on vertically partitioned data is posed in terms of structural risk minimization (see Sect. 3.1.5) as

$$\min_{\{\mathbf{w}[j]\}} \quad l\left(\sum_{j=1}^{m} \mathbf{D}[j]\mathbf{w}[j] - \mathbf{y}\right) + \sum_{j=1}^{m} r_j(\mathbf{w}[j])\,,$$

where $\mathbf{y} = (y_1, \ldots, y_n)$ is the vector of all labels, l measures the loss and $\mathbf{w}[j]$ is a partial weight vector whose dimension corresponds to the number of features p_j stored at node j. Multiplication of the local $n \times p_j$ data matrix $\mathbf{D}[j]$ with the partial weight vector $\mathbf{w}[j]$ results in a vector of dimension n, which consists of the local predictions at node j for the partial observations stored at j. The loss over all nodes should be minimized. The

regularization function $r(x)$ is assumed to be separable. For solving with ADMM, the authors introduce an auxiliary variable $\mathbf{z}_j$, resulting in the optimization problem

$$\begin{aligned} \min_{\{\mathbf{w}[j]\}} \quad & l\left(\sum_{j=1}^{m} \mathbf{z}_j - \mathbf{y}\right) + \sum_{j=1}^{m} r_j(\mathbf{w}[j]) \\ \text{s.t.} \quad & \mathbf{D}[j]\mathbf{w}[j] - \mathbf{z}_j = 0,\ j = 1, \ldots, m\,. \end{aligned}$$

For the SVM problem in particular, the distributed ADMM iterates are

$$\begin{aligned} \mathbf{w}^{t+1}[j] &= \operatorname*{argmin}_{\mathbf{w}[j]} \left(\frac{\eta}{2}||\mathbf{D}[j]\mathbf{w}[j] - \mathbf{D}[j]\mathbf{w}^t[j] - \bar{\mathbf{z}}^t + \overline{\mathbf{D}\mathbf{w}}^t + \mathbf{u}^t||^2 + \lambda||\mathbf{w}[j]||^2\right) \\ \bar{\mathbf{z}}^{t+1} &= \operatorname*{argmin}_{\mathbf{z}} \left(\mathbf{1}^T(m\bar{\mathbf{z}} + \mathbf{1})_+ + \frac{\eta}{2}||\bar{\mathbf{z}} - \overline{\mathbf{D}[j]} - \overline{\mathbf{D}\mathbf{w}}^{t+1} - \mathbf{u}^t||^2\right) \\ \mathbf{u}^{t+1} &= \mathbf{u}^t + \overline{\mathbf{D}\mathbf{w}}^{t+1} - \bar{\mathbf{z}}^{t+1}\,, \end{aligned}$$

where the bar denotes averaging, η allows for trading off speed of convergence against approximation error and λ controls the structural risk. Updates of the weight vector $\mathbf{w}$ require solving local ridge regression problems at each node. The $\bar{\mathbf{z}}$ updates can be shown to split to the component level, i.e. they can be run on each node independently from each other (for details, see [BPC+11]). What needs to be communicated to other nodes in each iteration is vector $\overline{\mathbf{D}\mathbf{w}}$, which is the average of predictions over all nodes.

The communication costs will depend on the total number of iterations T. In comparison to the horizontally distributed consensus SVM [FCG10] (see Sect. 4.3.2), consensus is not to be reached on p components of the weight vector $\mathbf{w}$, but on n predictions after applying the partial weight vectors $\mathbf{w}[j]$ to local data. Therefore, in each iteration n scalar values need to be communicated by each node. If $Tm > p$, already more data would be transmitted than sending all data to a central node, which is not communication-efficient. Unfortunately, the authors provide no empirical evaluation of how many iterations the algorithm needs to reach a sufficient accuracy on different datasets. In general, however, ADMM is known to have a slow convergence rate.

4.4.3 Fusion of Local Predictions

As already seen for the horizontally partitioned case, local models may be trained independently on each node, with their predictions being fused for final prediction. The training of local models is among the most communication-efficient methods, since only a single scalar needs to be sent from each node during prediction.

Separable SVM [LSM12] solves the primal SVM problem locally at each node with stochastic gradient descent (SGD). The global optimization problem consists of learning a weighting for the combination of local predictions. While [LSM12] addresses the tasks of 1-class learning, binary classification and regression, the following discussion is

restricted to binary classification. The primal optimization problem to solve is denoted in hinge function notation as

$$\min_{\mathbf{w}\in H} \quad \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{n}\sum_{u=1}^{n} \max\{0, 1 - y\langle \mathbf{w}, \Phi(\mathbf{x}_u)\rangle\} \,, \tag{4.12}$$

where feature mapping $\Phi : \mathbb{R}^p \to H$ induces a positive semidefinite kernel $k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}')\rangle$. (Here, without loss of generality, the intercept b is ignored). The kernel function is split across nodes by definition of a *composite* kernel k, which is a conic combination of local kernels $k_j : \mathbb{R}^{p_j} \times \mathbb{R}^{p_j} \to \mathbb{R}$ defined on the partial feature vectors $\mathbf{x}_i[j]$, $i = 1, \ldots, n$ stored at node j:

$$k(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{m} \mu_j^2 k_j(\mathbf{x}[j], \mathbf{x}'[j]) \,.$$

With Lagrange multipliers $\boldsymbol{\alpha}$, the optimization problem becomes

$$\begin{aligned} \min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \quad & \frac{\lambda}{2}\sum_{j=1}^{m}\mu_i^2 \sum_{u=1}^{n}\sum_{v=1}^{n} \alpha_u\alpha_v k_j(\mathbf{x}_u[j], \mathbf{x}_v[j]) \\ & + \frac{1}{n}\sum_{u=1}^{n} \max\left\{0, 1 - y_u \sum_{j=1}^{m}\mu_j^2 \sum_{v=1}^{n}\alpha_v k_j(\mathbf{x}_u[j], \mathbf{x}_v[j])\right\} . \end{aligned} \tag{4.13}$$

Problem (4.13) points to the fundamental difficulty of distributing SVMs in the vertically partitioned data scenario: All optimization variables $\alpha_1, \ldots, \alpha_n$ are coupled with each node $j = 1, \ldots, m$, and therefore cannot be split over the nodes. In [LSM12], separability of the problem is achieved by two different means. The first observation is that the terms $||\mathbf{w}||^2$ and $\langle \mathbf{w}, \Phi(\mathbf{x}_u)\rangle$ in the primal problem (4.12) become separable over the components of $\mathbf{w}$ if $\mathbf{w}$ and $\Phi(\mathbf{x}_u)$ are in a finite dimensional space. The authors therefore propose to replace local feature mappings Φ_j with approximate mappings φ_j which can be directly constructed using the technique of random projections (for details, see [LSM12]). The second observation is that the hinge loss can be upper bounded as follows:

$$\max\left\{0, \sum_{j=1}^{m}\mu_j(1 - y\langle \mathbf{w}[j], \varphi_j(\mathbf{x}[j])\rangle)\right\} \leq \sum_{j=1}^{m}\mu_j(1 - y\langle \mathbf{w}[j], \varphi_j(\mathbf{x}[j])\rangle) \,. \tag{4.14}$$

Summing up the inequalities (4.14) over training examples $u = 1, \ldots, n$, the local objective solved by each node $j = 1, \ldots, m$ becomes

$$\min_{\mathbf{w}[j]} \quad \frac{\lambda}{2}||\mathbf{w}[j]||^2 + \frac{1}{n}\sum_{u=1}^{n}\mu_j(1 - y\langle \mathbf{w}[j], \varphi_j(\mathbf{x}_u[j])\rangle).$$

A global classifier may then be constructed by combining local predictions, i.e. $\langle \mathbf{w}, \varphi(\mathbf{x})\rangle = \sum_{j=1}^{m}\mu_j\langle \mathbf{w}[j], \varphi(\mathbf{x}[j])\rangle$. Hence, for each test point, m scalars $\langle \mathbf{w}[j], \varphi(\mathbf{x}[j])\rangle$ need to be transmitted.

Approximating the non-separable hinge loss by separable upper bounds necessarily leads to a gap in accuracy. In addition to the local objectives, which are solved by SGD, [LSM12] therefore poses a central quadratic optimization problem for finding optimal weights $\mu_1, \ldots, \mu_m$. The problem is solved iteratively in an alternating fashion. Per iteration, each node j solves its local objective and transmits predictions for all n observations to the central node. The central node finds optimal weights μ_j and transmits them back to the corresponding local nodes. The loop stops after a user-specified number of iterations or if the central objective cannot be further improved. The algorithm has been evaluated on synthetic data and five standard datasets. While on one dataset, prediction accuracy could be improved by six percentage points with the central optimization, improvement on the other datasets was marginal. The use of random projections and a composite kernel reduces accuracy in the range of 5.5 to 27.8 percentage points. However, the method is highly communication-efficient, since without the central optimization, *no* data needs to be transmitted during training. It is also communication-efficient during application, since each node only transmits a single scalar value per test point, instead of p_j feature values.

4.4.4 Hybrid Methods

Hybrid methods combine different techniques and architectures for solving distributed problems. The approach for anomaly detection presented in the following combines local models with a global sampling strategy.

Distributed 1-class SVM [DBV11] introduces a synchronized distributed anomaly detection algorithm based on the 1-class ν-SVM (see also Sect. 3.3.1). A local 1-class model is trained at each local node and points identified as local outliers are sent to a central coordinator, together with a small sample of all observations. A global model trained on the sample at the central coordinator is then used to decide if the local outlier candidates are true global outliers or not. The method cannot detect outliers which are global due to a combination of attributes from different local nodes. However, the algorithm shows good performance if global outliers are also local outliers. Moreover, in the application phase, the algorithm is highly communication-efficient, since the number of outlier candidates is often only a small fraction of the data.

A drawback of the method is that the fixed-size sampling approach gives no guarantees or bounds on the correctness of the global model. Moreover, during training, no other strategies than sampling are used for a reduction of communication costs. The algorithm is therefore extended and improved on in this thesis (see Sect. 8).

4.4.5 Related Approaches

There are several approaches which seem to be somehow related to distributed learning in the vertically partitioned data scenario, but either do not fit it exactly, or are not distributed, or do not respect communication costs. They are subsumed in this section.

Co-Training The semi-supervised learning technique of *co-training* [BM98] assumes sample S to consist of a small number of labeled observations, and a large number of unlabeled observations. It further assumes that the training set can be divided into two different feature sets, which are also called views. For learning to work, the views must be conditionally independent, given the label. Moreover, each view must be sufficient, such that the class of an instance can be accurately predicted from each view alone.

Two classifiers $\hat{f}_1$ and $\hat{f}_2$ are first independently trained on the labeled instances from each view. The main idea of co-training then is to use both classifiers iteratively to label unlabeled observations. In the original algorithm, in each of T iterations, a smaller sample S' is drawn from S. Then, both classifiers are allowed to label p positive and n negative examples from S' they are most confident about, independently of each other. Each example labeled this way is then added to the set of labeled examples and S' is replenished by drawing $2p + 2n$ examples from U at random. The classifiers are trained anew on the now increased set of labeled examples.

While co-training itself isn't distributed, it is making assumptions about the two feature sets which are closely related to the ability to learn communication-efficient in the vertically partitioned data scenario. For instance, the labels assigned to the unlabeled observations could as well be exchanged between nodes. In [BS04], linear classifiers are casted into a probabilistic framework, developing a co-EM version of the Support Vector Machine. In experiments it is shown that the number of iterations needed for learning is relatively low, perhaps about 30 iterations. Keeping the number of iterations low is highly relevant in distributed learning from vertically partitioned data, when predictions for all observations are exchanged between nodes. Note that the original co-training algorithm does not even label all observations, but only those which classifiers are confident about. The distributed least squares regression approach [BGSW06] presented in Sect. 4.4.2 is also making use of co-training. However, there the labels for all unlabeled observations are exchanged between nodes, in each iteration.

Co-training may be also seen as a form of multi-view learning. In [Bre15], the author investigates the performance of the co-trained SVM and co-EM SVM on text classification problems, in cases where the independence assumption is violated. It is shown that even with random attribute splits, co-training can be beneficial for text classification. Further, the error correlation coefficient Φ^2 of the initial classifiers is identified as a measure which might benefit multi-view learning.

Multiple Kernel Learning (MKL) In *multiple kernel learning* (MKL) [BZB04], one considers a combination of t kernels

$$k(\mathbf{x}, \mathbf{x}') = \sum_{o=1}^{t} \beta_o k_o(\mathbf{x}, \mathbf{x}'),\ \beta_o \geq 0,\ \sum_{o=1}^{t} \beta_o = 1\,. \tag{4.15}$$

The main idea is that multiple kernels may perform better than just one kernel, since each kernel might specialize on different structures in the data, or certain kernels are better suited for particular kinds of data. For instance, RBF-kernels with different γ

values could take into account structures in the data at different levels of granularity. Similarly, time series, images and video sequences may each require different types of kernels. The β_o weight the kernels according to their importance and should be optimized automatically.

Using the standard formulation of the SVM with multiple kernels leads to a semi-definite program (SDP) [LCB+02]. This is much harder to solve than the standard SVM problem. For normalized kernels, i.e. $k_o(\mathbf{x}, \mathbf{x}) = 1$, the problem can be reduced to a quadratically constrained quadratic program, which can be solved more efficiently. The formulation can be modified such that it leads to further improvements, resulting in a semi-infinite linear program [SRSS06], a quadratic program, or faster interleaved optimization using ℓ_p-norms [KBSZ11].

The general formulation allows for kernels which are calculated over all features, or only (potentially overlapping) subsets of features. Of course, in a vertically partitioned data scenario, it would be also possible to have kernels whose calculations are restricted to features of each local node. In fact, applications which use multiple kernels suited for different kinds of data make heavy use of this property. Here, each kernel can be said to have a different view on a single observation, which might be described, for instance, by text data and image data.

However, separable kernel functions do not automatically lead to a separable objective function. As shown in [LSM12], the objective function of MKL which leads, for instance, to a quadratic program, is not separable over the nodes. All α variables are coupled with each node, such that optimization cannot be easily distributed. While it cannot be excluded that objective functions in MKL could be modified in a similar way as demonstrated in [LSM12], to the best of our knowledge, there are no distributed algorithms for MKL.

Bagging and Boosting Bagging and boosting (see [HTF09]) are both approaches from ensemble learning. In *bagging*, the predictions of different regression or classification models are combined according to some fusion rule. The technique of bagging has already been described in the context of random forests, which combine tree bagging with feature bagging. As we will see in Sect. 4.5.4, especially the technique of feature bagging may lead to high communication costs in the vertically partitioned data scenario, because feature values from different nodes are randomly drawn. A technique which draws samples of whole observations instead would be more comparable to the horizontally partitioned data scenario.

In *boosting*, a set of t weak learners is combined in the following form:

$$\hat{f}(x) = \sum_{o=1}^{t} \beta_o \hat{f}_o(x) \tag{4.16}$$

As long as performance of each weak learner is slightly better than random guessing, total performance can be boosted to that of a strong learner. The idea is to produce a sequence of weak classifiers, where each is applied to a differently weighted version

of the same sample. More accurate classifiers are getting assigned higher β_o weights. At intermediate steps, observations that were misclassified by the previous classifier have their weights increased for training the current classifier, and weights of correctly classified observations are decreased. Thereby, over several iterations, observations which are difficult to classify gain more and more influence. The next classifier must pay more attention to correctly classify these observations, whereby total performance over time is boosted. A popular boosting algorithm is AdaBoost.M1, but there are more recent boosting algorithms, such as LPBoost.

Though boosting combines the predictions of different learners, it is unclear how it would perform in the vertically partitioned data scenario. The point is that algorithms like AdaBoost.M1 assume that each classifier has access to the full sample. In comparison, in the vertically partitioned data scenario, local weak classifiers would only see subspaces of the whole data matrix $\mathbf{D}$. As will be explained in Sect. 4.5.1, conditional dependencies between feature sets of different nodes, given the label, can make learning difficult. Therefore, it is not even clear if all classifiers in the ensemble could really perform better than random guessing. Moreover, it is assumed that all weak classifiers have access to the same weights. In a distributed setting, such weights would need to be transmitted to the next local node in each iteration. However, it is unclear what such weights could mean for observations described only by a subset of features. In summary, one might say that boosting was not created with a distributed setting in mind and somehow doesn't seem to fit the vertically partitioned data scenario. A similar observation has been made in [LSM12].

4.4.6 Summary

As our previous discussion of distributed data analysis methods for the vertically partitioned data scenario suggests, communication-efficiency is much harder to achieve than for horizontally partitioned data.

The privacy-preserving SVM algorithms have a communication complexity of $O(n^2)$, and thus communicate more than the entire data if $n > p$. Consensus approaches have a communication complexity of $O(Tmn)$, which is not efficient if $Tm > p$. For instance, if 100 feature values are partitioned over 20 nodes, such algorithms could maximally run five iterations before they transmit more than the original data. In comparison, the algorithm for horizontally partitioned data communicates $O(Tp)$ values and is no longer efficient if $T > n$. The hybrid method for outlier detection is communication-efficient during training, as it samples from all data. During prediction, it is highly communication-efficient, since only outlier candidates are sent to a central coordinator, instead of each observation. However, its accuracy may suffer in cases where being a global outlier depends on a combination of features from different local nodes. Using random projections to reduce the number of features transmitted is communication-efficient, but in comparison the sampling approach, it transmits a reduced feature set for all observations.

The next section describes the challenges and difficulties of the vertically partitioned data scenario in more depth, by relating it to different learning primitives.

4.5 Challenges of Learning on Vertically Partitioned Data

Section 2.5.3 presented open questions of learning in the vertically partitioned data scenario. Such questions mainly concerned the design of algorithms for different learning tasks and the trade-off to be made between accuracy and communication costs. In this section, we describe the challenges of distributed learning in the vertically partitioned data scenario from the perspective of primitives that often occur in data analysis algorithms, like the ones presented in Chap. 3.

4.5.1 Estimation of Probabilities

As discussed in Sect. 3.2.2 on the Naïve Bayes classifier, one way to look at learning on propositional data with a categorical class variable is to think about it as estimating the probability $P(Y \mid X)$. In the same section, we shortly strived the assumption of features being conditionally independent, given the class. In this section, we look at the notion of conditional independence in more depth, and assess its meaning for distributed learning in the vertically partitioned data scenario.

Definition 4.5.1 (Conditional independence) In probability theory, two events A and B are *conditionally independent* given a third event C if and only if

$$\begin{aligned} &P(A \cap B \mid C) = P(A \mid C) \cdot P(B \mid C) \Leftrightarrow \\ &P(A \mid B \cap C) = P(A \mid C) \Leftrightarrow \\ &P(B \mid A \cap C) = P(B \mid C) \end{aligned}$$

In other words, A and B are conditionally independent given C if and only if, given the knowledge that C occurs, knowledge of whether A occurs provides no information on the likelihood of B occurring, and knowledge of whether B occurs provides no information on the likelihood of A occurring. Two examples might illustrate the notion of conditional independence further.

Example 4.5.1 Let us assume we want to decide about a person being a woman or man, based on the features hair-length and body size. If we already know a person to be a woman, the probability of her having long hair is high, based on the number of women having long hair in comparison to the number of men. Similarly, there is a high probability that she is smaller than 170 cm. We wouldn't assume the probability of her having long hair to change if we knew in addition that she is smaller than 170 cm, already knowing that the person is a woman. The reason is that both properties seem not to depend on each other, given the class.

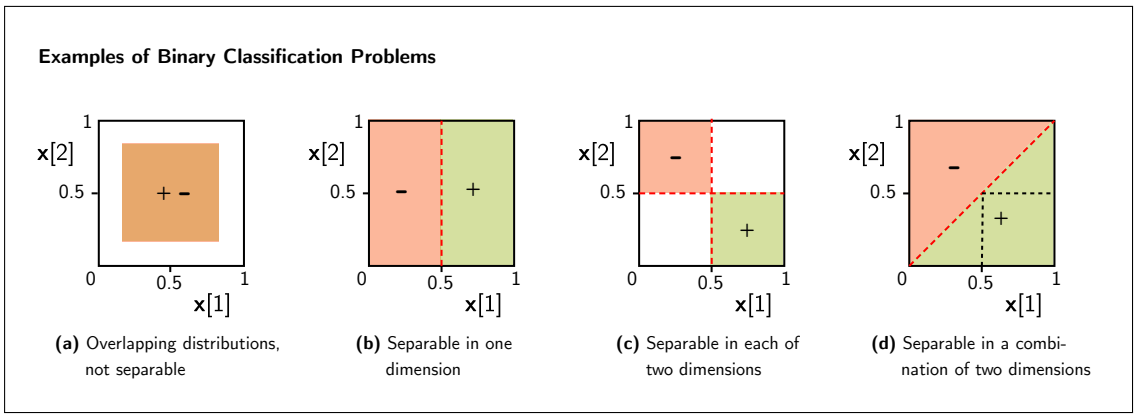


Figure 4.1: Examples of binary classification problems

It should be noted that conditional independence of features doesn't mean that taking into account the values of all features during prediction couldn't give us more evidence about the class. For this reason, in the Naïve Bayes classifier, we multiply the conditional probabilities of all features, given the class. However, it isn't necessary to consider the conditional probabilities of features, given the class and *other* features. Each feature provides information about the class on its own, independent of the values of other features. This means that even if we had access to only one feature, we should perform as well or better than random guessing in assigning the correct class label.

Example 4.5.2 Let us now consider the problem of deciding if a person is overweight, based on the features of body weight and size. The underlying rule of thumb says that someone is overweight if the body weight is higher than size, minus 100 cm. If we know that a person is overweight and has a body weight of 80 kg, then we also know that this person must be smaller than 180 cm. However, when all we know is that the person is overweight, the person could as well be taller than 180 cm. In other words, the probability for being smaller than 180 cm changes with additional information about other features. This means that body weight and size are not conditionally independent, given the label. In turn, to determine if someone is overweight, we need to consider both features in combination. Each individual feature alone doesn't give us enough information to become much better than random guessing, except in those cases where the body weight of a person is exceptionally high.

It can now be understood why the Naïve Bayes classifier described in Sect. 3.2.2 makes a "naïve" assumption. There are cases where the assumption doesn't hold. In Fig. 4.1, the problem is illustrated further for different binary classification problems.

In Fig. 4.1(a), examples of the positive class follow exactly the same distribution as those of the negative class. The conditional independence assumption holds, since

knowledge about the class and the value of feature $\mathbf{x}[1]$ provides no information about the value of feature $\mathbf{x}[2]$ (or the other way around). However, in terms of the optimal Bayes decision rule (see Sect. 3.1.2), the probability that a given observation belongs to the positive class equals the probability that this same observation belongs to the negative class, and no classifier could get any better than random guessing.

In Fig. 4.1(b), positive examples can be separated from negative examples by the rules "$y = +$ if $\mathbf{x}[1] < 0.5$" and "$y = -$ if $\mathbf{x}[1] > 0.5$". Given only values of the second attribute $\mathbf{x}[2]$, one couldn't do any better than random guessing again. The features are conditionally independent, given the label, since the probabilities $P(\mathbf{x}[1] < 0.5 \,|\, +) = 1$ and $P(\mathbf{x}[1] > 0.5 \,|\, -) = 1$ don't change if we know the value of $\mathbf{x}[2]$. That is, feature $\mathbf{x}[2]$ doesn't add any further information. Similarly, we don't get any information about the value of $\mathbf{x}[2]$, if in addition to the class label we'd know the value of $\mathbf{x}[1]$.

In Fig. 4.1(c), positive and negative examples can already be separated if we only know the value of $\mathbf{x}[1]$ or $\mathbf{x}[2]$ (or both). Again, the features are conditionally independent, given the label, since the label alone already determines the range of values for each feature. That is (probabilities of zero are excluded, but are analogous):

$$P(\mathbf{x}[1] < 0.5 \,|\, \mathbf{x}[2] < 0.5, -) \;=\; P(\mathbf{x}[1] < 0.5 \,|\, -) \;=\; 1 \tag{4.17}$$

$$P(\mathbf{x}[1] > 0.5 \,|\, \mathbf{x}[2] > 0.5, +) \;=\; P(\mathbf{x}[1] > 0.5 \,|\, +) \;=\; 1 \tag{4.18}$$

$$P(\mathbf{x}[2] < 0.5 \,|\, \mathbf{x}[1] > 0.5, +) \;=\; P(\mathbf{x}[2] < 0.5 \,|\, +) \;=\; 1 \tag{4.19}$$

$$P(\mathbf{x}[2] > 0.5 \,|\, \mathbf{x}[1] < 0.5, -) \;=\; P(\mathbf{x}[2] > 0.5 \,|\, -) \;=\; 1 \tag{4.20}$$

In Fig. 4.1(d), however, features are *not* conditionally independent, given the label. Looking at the regions of positive and negative examples, one can see, for instance, that $P(\mathbf{x}[1] < 0.5 \,|\, +) = 0.25$, but $P(\mathbf{x}[1] < 0.5 \,|\, \mathbf{x}[2] < 0.5, +) = 0.\bar{3}$. In other words, given the label value, feature $\mathbf{x}[2]$ provides additional information about the value of $\mathbf{x}[1]$. Further, knowing only the value of $\mathbf{x}[1]$ or $\mathbf{x}[2]$ all by itself doesn't suffice to separate positive and negative examples with zero error. However, if the values of both $\mathbf{x}[1]$ and $\mathbf{x}[2]$ are known, examples can be perfectly separated from each other by a diagonal line.

Now that we understand conditional independence of all features, given the label, we can assess its meaning for the vertically partitioned data scenario. We loosen the requirement in the sense that conditional independence doesn't need to hold for individual features, but only for subsets of features stored at different nodes. That is, we assume that all features in the subset of features stored at some node j are conditionally independent from features stored at all other nodes, given the label.

Whenever the assumption holds, subsets of features stored at different nodes can be treated independently from each other during estimation. This means that we can train local models independently of each other, per node, without transmitting any additional information between nodes. Individual estimations or decisions of local models on the class label can then be merged by some fusion rule, like the MAP criterion or a majority vote, with an according weighting of local decisions. Conditional independence of features, given the class label, thus gives a justification for the architectural design of distributed algorithms that combine the predictions of different local classifiers. They are

highly communication-efficient during training, since no information has to be exchanged between nodes at all. They are also communication-efficient when making predictions, since only a single scalar (the prediction) needs to be transmitted for each observation, instead of all feature values. However, if independence of features, conditioned on the class, does not hold, accuracy might suffer. In this case, at least some information about features residing at different nodes would need to be exchanged. In the following, we'll discuss what might be exchanged in the context of distance and kernel functions, as well as determining splitting points like they occur in decision tree induction.

4.5.2 Distance Functions

How does the calculation of distances, as they occur in algorithms like k-NN, k-Means clustering or outlier detection, relate to the vertically partitioned data scenario? As we can see, Euclidean distance would be separable across features and nodes, except for the final square root operation:

$$d(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{j=1}^{p} (\mathbf{x}[j] - \mathbf{x}'[j])^2} \tag{4.21}$$

However, since many algorithms are based on comparing relative distances only, the absolute value doesn't play any role, meaning that we can get rid off taking the square root:

$$d(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{p} (\mathbf{x}[j] - \mathbf{x}'[j])^2 \tag{4.22}$$

The individual elements of the sum can now be calculated independently from each other, feature-wise, on different nodes, and be centrally summed up. An interesting question is if also comparisons between distances can be made independently from each other, i.e. feature-wise, and their results be finally merged somehow.

Consider the outlier detection problem in Fig. 4.2. In this case, a point $\mathbf{x}$ is to be called an outlier if it lies outside a minimum enclosing ball around all normal points, i.e. if its distance from center $\mathbf{c}$ is bigger than some radius R. Can we decide if point x is an outlier or not only by looking at each dimension separately, and combining results later? The global outlier shown in Fig. 4.2(a) can be correctly identified as one by looking at each dimension separately, since it is also an outlier in each dimension. In Fig. 4.2(b) and Fig. 4.2(c), the global outlier is an outlier in at least one of two dimensions. If we see only one dimension, what can we do? One strategy might be to designate the point as an outlier candidate, and check it later on both dimensions. This is what the distributed 1-class ν-SVM by [DBV11] does: Send only outlier candidates to a central coordinator and check them against a global model that was trained on a sample of all data. However, we would still overlook the point in Fig. 4.2(d) being an outlier, since it is neither an outlier in the first dimension, nor in the second. As the example shows, it is not possible to determine all outliers correctly by looking at dimensions separately from each other.

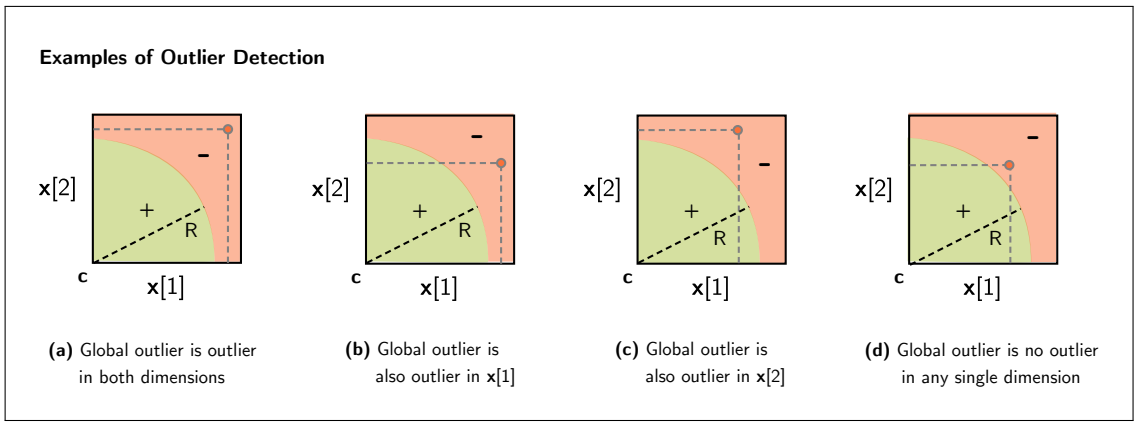


Figure 4.2: Examples of outlier detection

As long as we are interested in an exact algorithm, we might hope to distribute the calculation of partial sums at least. That is, we might calculate the partial sums of Euclidean distance, for instance, at each local node, and then sum them up centrally. We could then run our algorithm centrally, with distributed distance calculations. For each distance calculation, we would need to transmit at most one scalar per node, instead of all attribute values. Unfortunately, many data analysis algorithms require a quadratic number of distance calculations. For instance, the classification of new observations with k-NN requires the calculation of distances to all other existing n observations. This number might be reduced by the use of spatial index structures and sparing calculations by exploiting the triangle inequality of metric distance functions. However, those might not work well in higher dimensions.

4.5.3 Kernel Functions

Kernel functions are similar to distance functions in so far as the scalar product may be seen as a measure of similarity between observations. However, kernel functions have additional requirements to fulfill, as explained in Sect. 3.2.5. As can be seen, the partial sums of the linear kernel

$$k(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{p} \mathbf{x}[j] \cdot \mathbf{x}'[j] \tag{4.23}$$

can be calculated separately from each other at each node, like the partial sums of Euclidean distance. As we have seen in previous sections, it is much harder to develop distributed variants of algorithms when using non-linear kernels. In the vertically partitioned data scenario, where features of observations are distributed over different nodes, non-linear kernels are especially challenging. As explained in Sect. 3.2.5, kernel functions are associated with a mapping $\phi : X \times X \to \mathcal{H}$, which in some cases, can explicitly

represented. For instance, the explicit feature map for the polynomial kernel is given as

$$\begin{aligned}\phi(x) = \langle & x_n^2, \ldots, x_1^2, \sqrt{2}x_n x_{n-1}, \ldots, \sqrt{2}x_n x_1, \\ & \sqrt{2}x_{n-1}x_{n-2}, \ldots, \sqrt{2}x_{n-1}x_1, \ldots, \sqrt{2}x_2 x_1, \\ & \sqrt{2c}x_n, \ldots, \sqrt{2c}x_1, c\rangle\end{aligned}$$

What we see here is that the explicit feature map of the polynomial kernel combines features from different nodes. In fact, non-linearity implies that different variables are combined with each other. Can we somehow circumvent the problems of the explicit representation, by using the implicit kernel function, and separating its calculation across nodes? In other words, does the kernel trick maybe also help to spare us communication costs?

Interestingly enough, non-linear kernels like the RBF kernel can be divided across nodes:

$$\begin{aligned}k(\mathbf{x}, \mathbf{x}') &= e^{-\gamma||\mathbf{x}-\mathbf{x}'||^2} \\ &= e^{-\gamma(\mathbf{x}[1]-\mathbf{x}'[1])^2+\ldots+-\gamma(\mathbf{x}[p]-\mathbf{x}'[p])^2} \\ &= e^{-\gamma(\mathbf{x}[1]-\mathbf{x}'[1])^2} \cdot \ldots \cdot e^{-\gamma(\mathbf{x}[p]-\mathbf{x}'[p])^2}\end{aligned}$$

This property is used by the privacy-preserving SVMs described in Sect. 4.4.1, which combine the entries of kernel matrices from different local nodes. Unfortunately, kernel matrices have quadratic size. As it seems, with kernels, we run into similar problems as with distance functions. In Chap. 8 we will further see that kernel functions may appear in contexts which makes them inseparable.

4.5.4 Calculation of Splitting Points

Decision trees (see Sect. 3.2.3) require the recursive determination of optimal splitting points, based on the calculation of quality criteria defined on subsets of instances that would result from the split. How to make decision trees work in the vertically distributed scenario focused on in this thesis is not trivial. The combination of features from different nodes on the same path through a decision tree necessarily requires communication. It would need to be communicated which instances belong to which subset, given the previous split. It is unclear how to do so efficiently. It seems that at least $O(n)$ values would need to be sent, for each split that crosses the border of nodes along a tree's path. In comparison, in the horizontally partitioned data scenario, distributed decision tree algorithms only need to communicate the model to other nodes.

The procedure of feature bagging in random forests (see Sect. 3.2.4) looks highly similar to the training of local models in the vertically partitioned data scenario and combining their results by majority vote. However, both procedures are not equivalent. Though local features may be seen as random subsets of all features, the original algorithm of feature bagging has the chance to draw and combine features from different nodes in each recursion step, for each tree. Without change of algorithm, this global

sampling of features may lead to high communication costs. However, whenever we can make a conditional independence assumption on the features, given the label, we might restrict the training of trees to local features only (see also Sect. 4.5.1).

4.5.5 Summary and Outlook

As the previous discussion has shown, the development of communication-efficient distributed data analysis algorithms for the vertically partitioned data scenario isn't trivial. The privacy preserving SVM algorithms are accurate, but need to send quadratic kernel matrices. This is only communication-efficient if $p > n$. In the vertically partitioned data scenario, the iterative nature of consensus algorithms leads to high communication costs, since per iteration, already $O(mn)$ scalars are transmitted. Much better suited for the scenario are one pass algorithms, which either sample from the data or reduce it in a single step, like the presented approach for distributed ridge regression does, using random projections. The distributed 1-class ν-SVM, which is also based on such techniques, is guaranteed to be communication-efficient. However, they come with a problem: The number of points to sample and the size of a reduced subspace are user-defined and usually unknown before learning.

In the next Chap. 5, the vertically partitioned data scenario is motivated by a case study from smart manufacturing.

Chapter 5

Preprocessing Case Study

Digital sensors attached to IoT devices deliver continuous streams of real-valued measurements, i.e. series of values. Other time-related data, like parameters or states of devices, may be represented as value series as well. Before the modeling step (see Sect. 3.1.8), the stream of raw measurements must be prepared for further processing, which includes steps of data cleansing, alignment, the replacement of missing values, smoothing, normalization and segmentation. It then has to be transformed and must be brought into a representation which matches the learning task and the format of inputs expected by accompanying learning methods.

In this chapter, we deal with the transformation of value series for learning in the context of a case study from steel processing. In the following Sect. 5.1, the case study and its goals are described in more detail. In Sect. 5.2, we state the problem of value series representation and preparation more formally and, based on the case study, give a concrete example of how series of sensor measurements may look like. Further, it is explained how the problem relates to distributed learning in the vertically partitioned data scenario. Then, in Sect. 5.3, a selective overview of standard methods from the huge field of value series preprocessing is given, ordered by data preparation steps and different representations for learning to choose from. In Sect. 5.4, it is described more specifically how value series have been preprocessed in the context of the given case study, and Sect. 5.5 presents results from the modeling step, which follows preprocessing. The chapter finishes with a short summary, drawing conclusions and giving an outlook on the second part of this thesis in Sect. 5.6.

5.1 Real-Time Quality Prediction in a Hot Rolling Mill Process

In project B3 of the Collaborative Research Center SFB 876[1], the Artificial Intelligence Group LS 8 and the APS chair at the institute for production systems (IPS) at TU Dortmund University research new data mining and machine learning techniques for

[1] http://sfb876.tu-dortmund.de/SPP/sfb876-b3.html

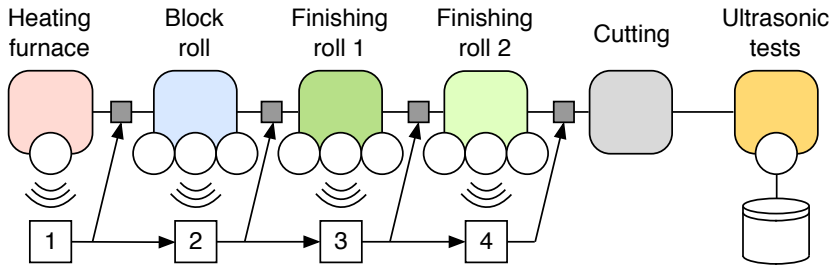


Figure 5.1: Hot rolling mill process with prediction (white squares) and decision/control modules (grey squares)

smart manufacturing. As discussed in Sect. 2.1.1, the manufacturing sector is adopting IoT technology at a fast pace. Embedding data analysis directly into the process chain and integrating it with control may lead to more sustainable systems, allowing for major reductions in waste, energy costs and the need for human intervention.

The particular focus of the case study is on interlinked production processes, more specifically on a hot rolling mill process. Here, steel blocks move through a process chain as the one shown in Fig. 5.1. Already casted blocks are first heated for up to 15 hours in five different heating zones of a furnace. They are then rolled at the block roll and the first finishing roll. The rolling in the second roll is optional. Each block usually moves back and forth through a single roll for several times, where each of the rolling steps takes only about a few seconds. The blocks are finally cut into smaller bars (also called rods) whose quality is assessed by ultrasonic tests several days later.

Different sensors attached along the process chain provide online measurements about how a steel block is currently processed. For example, in the furnace, every five minutes sensors measure the air temperature in each of the five zones. From such measurements, the core temperature of the blocks can be estimated by an already existing mathematical model. At each roll, sensors provide measurements such as rolling force, rolling speed and the height of the roll, with 10 values per second. Additional signals provide meta information about the process itself, like the current number of rolling steps. The ultrasonic test results indicate the number of bars tested and, for each bar, the amount of material containing defects, though not their exact position. Moreover, due to technical reasons, most often it is not possible to reconstruct which of the final bars belonged to which of the cut steel blocks. Learning from such statistical information about labels leads to a relatively novel kind of learning problem (see Chap. 6).

According to the current technical state of the art, it is impossible to assess the physical quality of hot steel blocks or smaller bars at intermediate steps of the process chain. The blocks first must cool down before their final quality can be tested. In cases where some of the blocks are, for example, already wrongly heated, energy, material and human work force are wasted if blocks below a desired quality threshold nevertheless move through the whole process chain. The goal is therefore the identification of quality-

related patterns in the sensor data, and to predict the final quality of steel blocks as early as possible during the running process, in real-time. Energy savings are already to be expected if, depending on the predictions, blocks with defects could be sorted out of the process early enough. For one thing, all of the following processing steps could be spared. For another thing, blocks might be reinserted into the heating furnace while still being hot, sparing the energy needed for a complete reheating. A reinsertion into the heating furnace might even be entirely spared if, depending on the predictions, parameters of subsequent processing stations could be adjusted such that the aimed-at final quality level would still be reached. Concepts for the integration of prediction models with control have already been developed by our project partners [KLD13].

However, before anything can be predicted at all, the continuous stream of measurements must be preprocessed, and brought into a format which is suitable for analysis. In the next section, data and problem will be described more formally. It will be further made clear how the given data and learning task relate to distributed learning in the vertically partitioned data scenario.

5.2 Problem Definition

In an IoT setting, there can exist a large variety of data sources. For instance, data may be generated by sensors which measure conditions such as pressure, humidity, temperature, gas, acceleration, force or light. Sensors can either be stand-alone devices, like MEMS [LSFB15], or be embedded into other devices and things, like cars, home appliances or smartphones. Hence, data may also stem directly from such devices, and reflect their internal states, or events, like the information that a button has been pressed. Another data source can be human generated data, like social network messages. In the mentioned smart manufacturing case study, all types of data are present. The rolling temperature, for instance, is measured by a sensor. The height of the roll is an internal machine parameter (or state). A human operator may decide to rotate a steel block along its longer axis during the rolling process, which is indicated by a binary signal. The question is how to unify such heterogenous data types for learning.

What all of the aforementioned measurements, states, or signals have in common is that they vary over *time*. A series of time-related values is called a *time series*. More generally, one can also speak of a *value series*, as long as the sequence of values has an inherent order. For instance, values of spectra can usually be ordered according to some physical quantity. In the next Sect. 5.2.1, we give a definition of value series and describe the format of data as given in the case study. In Sect. 5.2.2, we define the problem of finding a good representation, i.e. a mapping between the raw data and observations in propositional format. Then, in Sect. 5.2.3, some additional transformations are listed which might be needed to bring value series into a proper format for learning. Finally, in Sect. 5.2.4, the relationship between the case study and learning in the vertically partitioned data scenario is discussed.

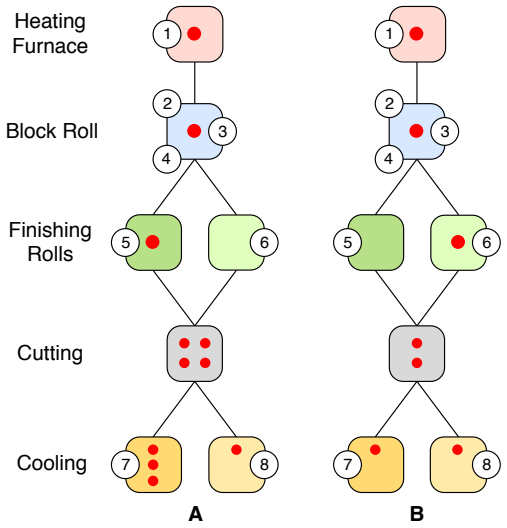


Figure 5.2: Routes that blocks A and B could take through some process chain

5.2.1 Value Series

In [MM05], a general definition of value series is given as a mapping from natural numbers to vectors with an index and value dimension. Here, we use a simpler definition as a list of values instead, which is more in accordance with the data recorded by the technical systems in our case study. The definition is more simple in so far as it ignores the real-valued index dimension, assuming that all values are sampled at equidistant time points, which is the case in the described hot rolling mill process.

Definition 5.2.1 (Value series) A *value series* is an ordered list of values $\mathbf{v} = [v_1, v_2, \ldots, v_p]$ of arbitrary length p. An individual element v_q of the list will be denoted as $\mathbf{v}[q]$. If a value at position q of the list is missing, we assign the special value "?", i.e. $\mathbf{v}[q] =?$.

In our case study, a sample $\mathcal{S}$ of historical data about the processing of steel blocks over a certain time period is then a set of tuples $\{(b_u, j_u, c_u, s_u, r_u, p_u, \mathbf{v}_u)\}_{u=1,\ldots,n_\mathcal{S}}$, where b_u is an ID for the steel block, j_u is an ID for the machine (or processing station) where the data has been assessed, c_u is an ID for the particular data source (i.e. sensor), called the *channel*, s_u is the absolute time point when recording has been started, r_u is the resolution with which the values have been sampled, and $\mathbf{v}_u$ is the according series of values with length p_u. Absolute time and resolution might be given in seconds, where $r_u = 1$ means, for instance, that values were sampled each second, $r_u = 10$ all ten seconds, and $r_u = 0.1$ each tenth of a second. While historical data is no streaming data, the above notation is general enough to cover the streaming case: If we set $p_u = 1$ and $r_u = 0$, i.e. we allow for an infinite resolution, we may assume that the elements of

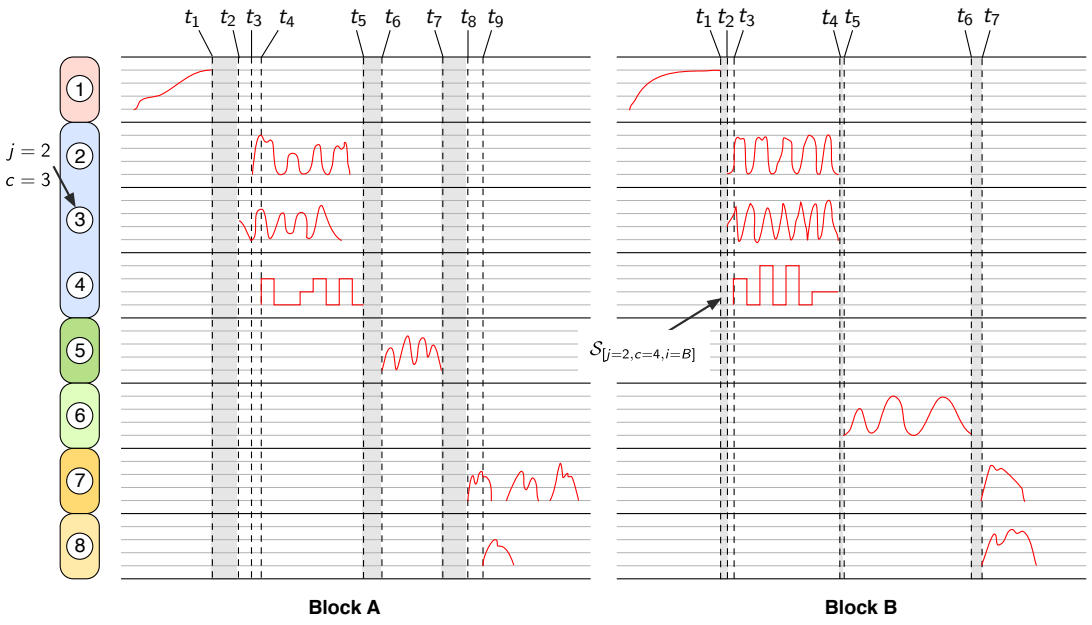


Figure 5.3: Sensor measurements for blocks A and B

$\mathcal{S}$, which are now single values, arrive in sequence such that $s_1 \leq s_2 \leq \ldots$, i.e. the values arrive ordered (or at least stamped) by the time point when they have been assessed. Subsets of $\mathcal{S}$ may be created based on a given time interval, where

$$\mathcal{S}_{[t_s - t_e]} := \{ (b_u, j_u, c_u, s_u, r_u, p_u, \mathbf{v}_u) \mid s_u \geq t_s \wedge s_u + r_u \cdot p_u \leq t_e \} \tag{5.1}$$

Other subsets can be defined in a similar way. For instance, $\mathcal{S}_{[j]}$ denotes all data for machine j, the subset of all tuples where $j_u = j$. $\mathcal{S}_{[c]}$ denotes all data for channel c, the subset of all tuples where $c_u = c$. $\mathcal{S}_{[i]}$ denotes all data for steel block i, the subset of all tuples where $b_u = i$. Such filters may also be combined. $\mathcal{S}_{[j,t_s-t_e]}$, for instance, is the subset of all tuples belonging to machine j, restricted to the time interval $[t_s, t_e]$.

Example 5.2.1 We illustrate process chains and associated sensor measurements by an example. Figure 5.2 shows how two different steel blocks, A and B, might have moved through several processing stations: a heating furnace, a block roll, two finishing rolls, a station for cutting the blocks, and two cooling stations. Each station, except for the cutting station, has sensors attached. Steel block A is first heated up, rolled and then moves through the first finishing roll. It is then cut into four parts. Three of such parts move through the first cooling station while one moves through the second one. Steel block B moves through the second finishing roll instead, is cut into two parts afterwards, and such parts then move through the cooling stations in parallel.

The measurements that might have been recorded by each sensor over time are shown in Fig. 5.3. The repeating patterns in many of the value series, for instance at

the rolls, represent different rolling steps, i.e. a single steel block could move several times through a roll. For instance, according to the measurements of sensor 2, steel block A moved four times through the block roll and steel block B five times. Since A moved through the first finishing roll, there are no measurements for the second roll, while for B, there are values for the second roll, but none for the first.

During time intervals $[t_1, t_2]$, $[t_5, t_6]$ and $[t_7, t_8]$, steel block A has moved from one processing station to the other, as well as steel block B during time intervals $[t_1, t_2]$, $[t_4, t_5]$ and $[t_6, t_7]$. For steel block A, the recording at the block roll starts with sensor 3, while sensor 2 has a time lag of $t_3 - t_2$ and sensor 4 a lag of $t_4 - t_2$. Similar lags occur during the processing of steel block B. Sensor 4 is an example of a sensor whose values are not continuous, but discrete.

5.2.2 Problems of Representation

As the previous example shows, the data about the processing of a single steel block, like block A, consists of value series from different machines and sensors. Before we come to the problem of early quality prediction in real-time, and the vertically partitioned data scenario, let's assume for a moment that we are given all the data at once and want to learn a prediction model for the quality of rods finally cut from each steel block. For instance, we may be interested in finding correlations between patterns in the value series and the given quality information. Then, it becomes apparent that the given data deviates much from the canonical propositional representation of observation that has been discussed in Sect. 3.1.9. Instead of a fixed-length vector of feature values, here the processing of a single steel block is described by a whole set of value series with different lengths, from different machines and sensors over different points in time. The raw data as given cannot be used in the modeling step, as it does not fit the input representation of many learning algorithms.

There are two options now. Either we invent new learning algorithms which are tailored specifically to (collections of) value series data, or we try to transform the data into propositional form such that it can be handled by most standard learning algorithms. Let $\phi_t : \mathcal{X} \to X$ be a function which maps observation $z \in \mathcal{X}$ from instance space $\mathcal{X}$, represented in some language, to a new observation $\mathbf{x} \in X$ from instance space X, represented in propositional form. In a supervised learning setting, in analogy to supervised function learning (see Sect. 3.1.4), in the following we define the task of finding or learning a good representation.

Definition 5.2.2 (Supervised Representation Learning) Given a sample $Z = \langle (z_i, y_i) \rangle_{i=1,\ldots,n}$ of n labeled examples $(z_i, y_i) \in \mathcal{X} \times Y$, drawn i.i.d. from an unknown joint probability distribution $P(\mathcal{X}, Y)$, the task of *supervised representation learning* consists of finding functions $\phi_t : \mathcal{X} \to X$ and $\hat{f} : X \to Y$, such that observations $x_i = \phi_t(z_i)$ with $x_i \in X$ are in propositional form and the expected risk

$$R_{\text{exp}} = \int \ell(y, \hat{f}(\phi_t(z)))dP(z, y)$$

is minimized. Here, ℓ is a convex *loss function* $\ell : Y \times Y \to \mathbb{R}_0^+$ which measures the cost of assigning the wrong label to individual observations.

It is important to note that the definition of the task as given above is general enough to cover also the case where the input examples z_i are already given in propositional form. We may then find a new representation, i.e. a new set of attributes with according values, which is better suited for learning.

Example 5.2.2 In our case study, each observation z_i is a collection of value series for steel block i, i.e. $z_i = \mathcal{S}_{[i]}$. Given function ϕ_t, we may create a new sample $S = \langle(\phi_t(z_i), y_i)\rangle$ for deriving a prediction model $\hat{f}$ by supervised function learning.

As the example suggests, finding a good representation of raw data, which is sometimes also called *feature extraction*, can be seen as an iterative two step process. First, some ϕ_t is chosen, and a new sample S from $\mathcal{S}$ is created by applying ϕ_t to each observation in $\mathcal{S}$. Then, the best model $\hat{f}$ is selected by training different learning algorithms and evaluating them on a hold-out test set. If performance is sufficient enough according to some quality criterion, ϕ_t and $\hat{f}$ are taken, otherwise, the search continues.

Several studies, for instance [MK05, Mor99], have shown that changing the data representation changes the ranking of learning methods substantially. Each learning algorithm favors different features. For a model's quality, feature extraction can therefore be more important than the particular learning method. The given definition reflects the fact that there is not one best method, but one best pair of method and representation.

In the case of value series, finding a good representation is often a manual, highly heuristical ad hoc process. There exist only few approaches which have automated the process, one of them being method trees [MM05]. Building trees for feature extraction follows a genetic programming approach, and has been successfully evaluated in the context of music genre classification. Unfortunately, the method is hard to apply in our case study, since our observations are not single value series, but sets of value series. Currently, another strand of research focuses on the automatic building of deep representations consisting of hierarchies of features [BCV13]. Deep learning has also been discussed in the context of feature extraction from time series [LKL14]. However, many of the approaches referred to are unsupervised, while we want to find a good representation based on the given labels.

Finding good representations for learning in a fully automated fashion remains a hard problem. One challenge is complexity from a computational point of view. Concatenation of various operators for preprocessing leads to non-convex optimization problems. The evaluation of fitness in a genetic programming approach like method trees may take a long time, due to the evaluation of one or several classifiers. For this reason, even if the search space is structured, not many combinations can be tried. Another challenge is complexity from a learning point of view. With more preprocessing operators to choose from, capacity of the function class increases. As discussed in Sect. 3.1.3, this may easily lead to overfitting, requiring an even larger number of trials to minimize the true error as estimated from a hold-out test set.

In the literature, at least three popular ways to represent value series (not necessarily in propositional form) can be found:

Raw The value series are left in raw form, comparing them based on distance measures or kernel functions which have been specifically developed for value series. Another way is to map the raw values to positions in a fixed-length propositional vector by according scaling operations.

Numerical Numerical features are extracted from the value series and then mapped to a fixed-length vector.

Symbolic Value series are transformed into a symbolic representation, from which in turn numerical features can be extracted (or not, depending on the approach).

In Sect. 5.3.2, a selective overview of several standard approaches is given which transform value series into one of the previously mentioned representations.

5.2.3 Problems of Preparation

While choice of representation can have large influence on learning, and therefore is one of the most important steps in preprocessing, data quality is another important aspect. For instance, features extracted from value series that contain missing values, or values lying outside meaningful ranges, have certainly lower quality than features extracted from an accordingly cleansed value series. In comparison to the function ϕ_t, which maps observations from one representation to another, data preparation steps usually don't change representation, but only the values of observations. In the context of the discussed case study, the following data preparation steps for the given value series could be identified:

Cleansing Faulty sensor readings must be identified and handled accordingly, for instance by replacing them with new values. Further, parts of the value series which are definitely irrelevant for learning, like those where no processing happened, are to be stripped off.

Imputation Missing sensor readings must be identified and replaced accordingly.

Alignment Value series might be shifted, due to missing or faulty synchronization of clocks. They then have to be shifted such that parts with the same meaning are properly aligned with each other. Similarly, they may come at different resolutions and therefore must be scaled accordingly along the time dimension.

Smoothing Value series whose sensor readings contain lots of noise should be smoothed before further processing.

Normalization While scaling due to different resolutions along the time dimension is called alignment in the following, normalization means the analogous scaling and shifting of data along the value dimension.

Segmentation Value series may contain distinct parts or patterns from which features should be extracted, like the different rolling steps, for instance. Segmentation means the division of value series into such meaningful intervals.

The different steps are explained from a more general perspective in Sect. 5.3.1, where also references to existing literature are given. In Sect. 5.4, it is explained in more detail how value series data has been prepared in the context of the case study.

5.2.4 Learning from Vertically Partitioned Value Series

The learning task in our case study is not just to find correlations between patterns in value series and related quality information, but to predict the final quality of the rods which are cut from each steel block after processing as early as possible, in real-time. This means that at a given time point t, only partial information about observations is given.

Example 5.2.3 If processing of block A has started at t_0, the data available for prediction shortly after A has left the furnace is $\mathcal{S}_{[t_0-t_1,i=A]}$, which is the data of sensor $c = 1$ in the furnace ($j = 1$) (see Fig. 5.3). At a later point in time, for instance t_9, we have seen the data $\mathcal{S}_{[t_0-t_9,i=A]}$, i.e. the data of sensors $c = 1, \ldots, 5$ during the heating and all rolling steps, and we also got a bit of data about the cooling ($j = 5$, $c = 7$).

For simplicity, let us assume we want to predict the final quality of the rods after a steel block has left a processing station. Then we may learn separate prediction models $\hat{f}_j$ for each processing station, based on data from the previous processing stations and j itself. This could be done in several ways. All measurements could be sent to a central server for analysis. Or, maybe the quality could already be predicted sufficiently just from combining the predictions of local models, i.e. $\hat{f}_1, \ldots, \hat{f}_j$, directly trained at each processing station. Maybe we could just send predictions from the previous node, and build a classifier chain. Or, due to conditional dependencies between processing steps, given the quality, some more information might need to be exchanged between processing stations. In each case, we are in the vertically partitioned data scenario, since information about the same observation (the processing of a single steel block) is partitioned over different networked machines. This means we have all of the aforementioned design options to distribute components across nodes.

Though the situation in the given case study doesn't match some problems of learning in the vertically partitioned data scenario exactly, due to processing's sequential nature, and though it isn't as communication-constrained as the scenarios of WSNs we have previously discussed, due to the use of high bandwidth connections, it is nevertheless constraint by the time available for making predictions. The ultimate goal is to fully

integrate data analysis with control, such that decisions about the further processing of steel blocks can be made within a few milliseconds, based on the timely predictions of trained models. If we increase the rate of making predictions, for instance predicting quality after each rolling step, or even during rolling steps, we may also have to increase the rate with which values are sampled. In other words, we have to process more data in less time. The question then is what kind of data to process locally, and what kind of data to send to a central server or directly to the next processing station. The problem here isn't necessarily bandwidth, but latency. The more packets we send, the more probable collisions become, especially on a central bus connecting machines. Depending on when packets arrive at the central server, or at the processing station, it might be already too late for making a proper control decision. We think the tighter the feedback loop between prediction and control modules becomes, the more we will arrive at similar questions as those in learning from vertically partitioned data, namely which data is needed at which networked node to make an accurate prediction, and which information should be communicated to match the given real-time constraints.

5.3 Standard Methods of Value Series Preprocessing

The following subsections give an overview of the steps which are typically involved in the preparation of value series (Sect. 5.3.1) and transforming them into a suitable format for learning (Sect. 5.3.2).

5.3.1 Data Preparation

Before value series can be brought into a format that is appropriate for learning, like the propositional representation, they usually must be processed by following the data preparation steps identified in Sect. 5.2.3. These are described in more detail here.

Cleansing The first step of *cleansing* should get rid of parts of value series which are not needed for learning and subsequent preprocessing steps. Depending on the application, one may only be interested in measurements that indicate certain events. For instance, in a production setting consisting of different processing steps, parts need a certain amount of time to move from one processing station to the next. The particular measurements assessed during such time intervals are usually irrelevant. If at all, the only information that matters is the time a part needed to move from one station to the next. As reasonable as stripping off irrelevant readings might sound, the task can be non-trivial. Due to quantization errors caused by the sampling of analog signals and noise, sensors may deliver different values even if the quantity that is measured didn't change. Simple solutions may mark changes as relevant only if the amount of change exceeds a certain threshold.

In real production environments, sensors might provide also wrong readings or can fail entirely. The quality of a sensor depends on how much its measurements are influenced by quantities it wasn't designed for. Many sensors are sensitive to changes

in temperature. They can thus provide wrong readings [NRC+09]. Further, the older sensors get, there can be a drift in sensor readings. Faulty sensor readings can only be handled if they are detected. Such detection is easiest in cases in which sensor readings lie outside physically meaningful ranges, as defined by accompanying meta data. But there are also non-trivial cases, in which faulty readings overlap with the normal data, requiring the automatic detection of faulty patterns. If such patterns cannot be defined based on knowledge about the underlying hardware [JAF+06], or based on visual inspection, they might be derived automatically by supervised learning methods (see Sect. 3.2). If the faults are highly irregular or not frequent, it is difficult to learn their detection on the basis of given training examples. Models for the detection of anomalies in production settings often describe only the normal data, marking patterns as anomalies that deviate from the learned description (see Sect. 3.3). Nevertheless, the correct definition of parameters, like threshold values, remains difficult with only a few negative examples. Moreover, it can be difficult even for domain experts to identify such negative examples correctly.

Alignment The time series of different sensors may have different *resolutions*, *lengths* and *offsets*. Depending on the prediction task and methods, it can be necessary to scale and shift time series correctly before they are further processed, such that parts of different value series with corresponding meaning can be compared, for instance, with according distance measures. In other words, the value series must be *aligned*. Correct alignment is especially important for the subsequent step of segmentation, where indices designating the intervals of segments may be derived from one value series, and applied to another. Here, not taking into account the different resolutions, lengths and offsets of value series may easily lead to incorrect segmentation results.

Imputation Once faulty readings or missing values are detected, there are different possible ways to handle them. A simple strategy for the replacement of single or only a few faulty values is to replace them by their predecessor value or based on ARMA (auto-regressive moving average) models [BJR94]. Replacement of missing values is also called *imputation*. In other cases, faulty values can be imputed based on prediction models that were trained on other existing values. However, if many relevant values are missing or whole sensors fail, the quality of the predictions may either decrease or it might become impossible to provide a prediction at all. The challenge here is to estimate the confidence of predictions correctly, since it is not always clear how missing or faulty values in the raw sensor data will influence later preprocessing and model building steps. Another challenge is that different types of sensors may require different strategies for the handling of faults and that knowledge about the best strategy is often scarce.

Smoothing Finally, even correctly working sensors usually have some level of *noise*, which may also be introduced by the production process itself, like sensors moving due to vibrations. If the underlying noise model is known, it should be used. Otherwise, mea-

surements can be filtered and *smoothed*, for instance with the same ARMA techniques used for the replacement of missing values. However, finding the correct parameters for filtering is not necessarily trivial, since it also interacts with subsequent preprocessing and model building steps.

Normalization Depending on application and learning task, it can become necessary to scale and shift value series also along their value dimension, for better comparison. This is especially important in cases where the focus is not on the differences of values, but form. For instance, in our case study, large differences in rolling force might skew the results of distance calculations which should assess the similarity of rolling force patterns. Scaling and shifting value series along the value dimension is known as *normalization*.

A common normalization for value series is the Z-transformation, recommended in [KK03, RCM+12]. A value series $\mathbf{v}$ is called *z-normalized* if

$$\mu := \frac{1}{p}\sum_{j=1}^{p}\mathbf{v}[j] = 0 \quad \text{and} \quad \sigma^2 := \frac{1}{p}\sum_{j=1}^{p}(\mathbf{v}[j]-\mu)^2 = 1\,. \tag{5.2}$$

Segmentation Value series may contain parts with different meanings, and which therefore should be treated differently in subsequent processing and feature extraction steps. Finding such parts and dividing a value series into according intervals along the index dimension is known as the preprocessing step of *segmentation*. The *automatic segmentation* of value series can be a difficult task, since it depends on the correct detection of patterns.

Many segmentation techniques have been developed for the preprocessing of image data. There, the borders of segments usually indicate significant changes in basic properties of the pixels, e.g. their color. Once the segments are determined, features can be extracted from them, like their average, minimum and maximum color, gradients or textural features. SIFT features [Low04], which are translation, rotation and scale invariant, have almost become a standard for the meaningful description of images. The authors of [CRWS12] propose a salient features approach for the segmentation of one-dimensional value series, transferring ideas from image segmentation and the extraction of SIFT features. Salient points in the value series are points which deviate much from their surrounding values, and may be used for segmentation. Then, from each segment, characterizing features are extracted. The method determines salient points at different resolutions, allowing for a description of value series at different levels of granularity. Another promising approach not only automatically divides value series into their segments, but also clusters them, following the Minimum Description Length (MDL) principle [RKLE12].

5.3.2 Choice of Representation

Section 5.2.2 already discussed the problem of finding a good representation of value series in general, and especially in propositional format. In the following, the most

common representations of value series found in the literature are presented and shortly discussed.

Representation by Raw Values The direct handling of value series with learning algorithms that are heavily based on attributes in the propositional representation of observations usually doesn't make much sense. For instance, imagine the application of decision trees to value series. Methods for the induction of decision trees recursively determine splitting points, i.e. attributes and their values, according to a given quality criterion. In the case of value series, this would result in choosing specific points along the index dimension, which can be time. Taking an example from the presented case study, this might then result in a rule which states that whenever the rolling force after ten seconds of processing exceeds a specific threshold value, the final rods cutted from a steel block will have low quality. Though the rule might sound sensible in itself, the question is what happens when the value occurs only slightly shifted at ten seconds and 100 milliseconds, maybe due to alignment errors. Depending on the form of value series, this rule might then no longer apply, resulting in an incorrect classification. Moreover, we might be not just interested in single time points, but the *form* of value series over longer intervals. With learning algorithms expecting observations in propositional representation, it therefore makes much more sense to extract according features, as presented in the following subsections.

However, the use of raw values can make sense with distance-based methods. If the value series are correctly aligned, normalized and have the same length, their form may be compared by standard Euclidean distance. For value series of differing lengths, more specialized distance measures have been developed, like Dynamic Time Warping (DTW) [MÖ07] or Longest Common Subsequence (LCSS) [DGM97] distance.

Numerical Representation Aggregation and summarization methods for value series reduce the amount of raw data as much as possible, while at the same time trying to keep its most important characteristics. The amount of feasible reduction depends on the prediction task. The simplest type of aggregation is the calculation of summary statistics, like minimum and maximum values, the mean, median, standard deviation, percentiles or histograms. According to our experience, such simple global features can already be sufficient for prediction purposes (see also Sect. 5.5 and [SBM16]).

More sophisticated methods try to represent a given time series as the combination of a (usually fixed number) of basis series, like the Discrete Fourier Transform (DFT) [FRM94] or the Discrete Wavelet Transformation (DWT) [MVW00]. The approach of mathematical models presented in [KKP+09] approximates time series Y by a model $Y = f(X, \vec{\alpha}) + \epsilon$, where f is an arbitrary mathematical function and X a fixed set of basis functions. The basis functions can be derived, for example, by clustering (see Sect. 3.4), while the coefficients $\vec{\alpha}$ can be determined by least squares, such that the random error ϵ is minimized. Instead of representing value series by their raw values, they can then be represented by a fixed-length coefficient vector. A disadvantage is that

such coefficients are usually much harder to interpret than the aforementioned simpler summary statistics.

All of the aforementioned methods can also be extracted from smaller sized windows which are shifted over value series. On top of them, the same methods might be used again, being recursively applied. Together with the aforementioned data preparation steps, there are then myriads of possible ways to preprocess value series and extract meaningful features from them, as already discussed in Sect. 5.2.2. In addition, almost all methods can be used with different parameterizations. Instead of trying and combining all such methods and their parameters manually, the authors of [MM05] propose method trees to learn promising combinations of preprocessing methods and their parameters. Basis transformations, filters, mark-ups and a generalized windowing cover elementary methods that can be combined in the form of a method tree. The tree applies the operators (nodes) in a breadth-first manner, thus transforming a value series. The root of each tree represents a windowing function, while the children of each parent node form operator chains consisting of basis transformations, filters and a finishing functional. Learning the feature extraction tree is done by a genetic programming algorithm. In each iteration, the algorithm generates a new population of method trees, by mutation and crossover operators that change and combine respective subtrees. Preprocessing results in a fixed number of attributes being extracted, i.e. that observations are transformed into propositional format. Therefore, the fitness of each method tree can be determined by an arbitrary inner classifier. Complexity is reduced by limiting the number of possible preprocessing operations and parameters that can be used, and structuring the search space accordingly. The approach has been used successfully for the classification of music by genre or the personal music taste, but should be applicable to value series from other domains.

Symbolic Representation The symbolization of value series bridges the gap between numerical methods and those that work on symbols, like frequent item set mining [AIS93] or text processing. For example, frequencies of symbols, sequences or words are length-invariant features that already have been used successfully in areas such as text classification or intrusion detection, where documents and records may have different lengths and numbers of words.

Symbolic Aggregate Approximation (SAX) [LKWL07] first determines the elements of a sequence $C = (c_1, \ldots, c_n)$ by piece-wise aggregate approximation and maps them to a new sequence C' with $w < n$:

$$c'_i = \frac{n}{w} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} c_j$$

The elements c'_i are then discretized by mapping them to a fixed number of symbols, preserving the upper bounded Euclidean distance between all series. In [SM13], value series are first symbolized with SAX and then transformed into a new propositional representa-

tion according to the vector space model known from text processing. A gradient-based approach for the symbolization of streaming sensor data has been proposed in [MW99].

Once value series are symbolized, several algorithms working on individual symbols or sequences of symbols can be applied. For instance, the KRIMP method [SVvL06] compresses a database of binary transactions by code tables. An open research question is if such code tables could also be used as a condensed representation of value series. The AprioriAll algorithm [AS95] finds *frequent sequential patterns* in transactions of items, e.g. subsequences in symbolized value series. General Sequential Patterns (GSP) [SA96] extend the previously mentioned approach by respecting constraints on the patterns. Similarly, the WINEPI algorithm [MTV97] can find *frequent episodes* in event sequences.

Algorithms for *motif discovery*, like the probabilistic approach introduced in [CKL03], not necessarily require a symbolic representation of value series, but try to find frequently reoccurring subsequences directly. An interesting new direction is the supervised discovery of motifs, like the shapelet approach [YK09], which can also take given class labels into account.

5.4 Preprocessing of Value Series from Production

As the example in Sect. 5.2.1 has shown, the "features" of observations in our case study are series of sensor measurements, machine states and events which describe how a single steel block has been processed over time at different processing stations. Most of the approaches referred to in Sect. 5.3 work on *univariate* value series. For instance, several existing classification approaches, like the automated building of method trees, assume that each observation in a sample is a single, one-dimensional value series. Alignment and segmentation operations can either work on each observation independently, or take all observations in the sample into account.

The data as given in the presented case study, however, differs much from the aforementioned format of observations. In the context of production processes, each observation is not a single value series, but a *set* of value series which may have all different lengths and offsets, which can overlap in time, which may contain different numbers of segments at different levels of granularity and resolutions, and which may stem from different machines and sensors. Though such a set of value series could be interpreted as being a single *multivariate* value series, observations are not simply given in matrix form, with each series in a row being correctly aligned already. Instead, each observation has to be brought into this format first. For instance, all value series related to the processing of block i, recorded at the same processing station j, i.e. $\mathcal{S}_{[i,j]}$, have to be correctly aligned for proper segmentation, before feature extraction. Thereby the measurements from different sensors (channels) must each be treated differently. The preprocessing of data from production processes is thus highly domain dependent and individualized, and not directly covered by any standard approach. Of course, preprocessing modules can build on existing methods and techniques presented in Sect. 5.3. However, the exact combination of such methods and their parameterization depends on many different

Algorithm 5 Preprocessing of Value Series from Production Processes

```
1: procedure PreprocessValueSeries(S)
2:     S := ∅                                          ▷ create new sample in propositional format
3:     for i ← 1, n do                                 ▷ for each product i (steel block)
4:         for j ← 1, m do                             ▷ for each production step j (processing station)
5:             for c ← 1, k do                         ▷ for each sensor c (channel)
6:                 v := v ∈ S_[i,j,c]                  ▷ take value series
7:                 v := GlobalPreparation_{j,c}(v)     ▷ clean, impute, align, etc.
8:                 x_i := GlobalFeatures(v)            ▷ extract global features
9:                 G := Segmentation_j(v)              ▷ segment series
10:                for v_g ∈ G do                      ▷ for each segment v_g
11:                    v_g := LocalPreparation_{j,c}(v_g)   ▷ clean, impute, align, etc.
12:                    x_i := x_i ⋈ LocalFeatures(v_g)      ▷ extract and join local features
13:                end for
14:                x_i := x_i ⋈ AggregateFeatures(x_i)      ▷ aggregates of local features
15:            end for
16:        end for
17:        S := S ∪ {x_i}                              ▷ add new observation to sample
18:    end for
19: end procedure
```

factors like processing station, sensor type, the format of the steel block produced, etc. In the end, the data from different preprocessing modules must all be brought together in a single data table to learn from.

Algorithm 5 gives an overview of the procedure we propose for the preprocessing of value series from production processes. The full procedure and its components have been implemented as different processes calling each other in the data mining software RapidMiner [MWK+06]. This has been done in close collaboration with our project partners, who provided the necessary domain knowledge to implement the highly specialized series preparation operations. As can be seen by the indices attached to procedure names, such operations depend very much on the particular processing step j and type of sensor c. The rest of the process is kept as generic as possible. Especially, the global and local feature sets extracted easily can be extended by additional features, without having to touch any of the more specialized preparation operations. The only requirement is that the number of features extracted is fixed, such that at the end, all observations added to the new sample S are in propositional form and have the same number of attributes. We think that the process is general enough to cover a whole lot of production scenarios, for instance also discrete manufacturing settings, except for the domain-specific preparation operations, of course. The procedure has been designed such that the value series from a single processing station j can be preprocessed independently of those from other stations, which allows for a local execution per node j in the vertically partitioned data scenario.

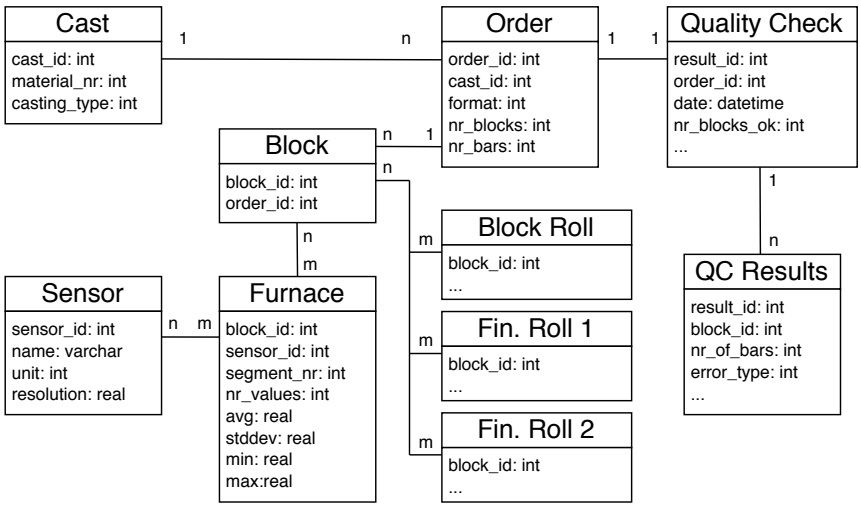


Figure 5.4: Database schema for hot rolling mill case study

The implementation in RapidMiner consists of about 100 processes for the preprocessing of 17 different sensor types, attached to four different processing stations. Of such processes, 42 have about 40 elementary operators. Complex data mining processes cannot be realized by scripting anymore. They need a principled development process. In appendix A it is explained how elementary data structures and operators of RapidMiner can be mapped to well-known constructs from structured programming languages, giving a better idea of how the processes in the case study have been realized.

In the following sections, it is explained in a bit more detail how the value series have been preprocessed. Section 5.4.1 explains the first step, namely how data has been assessed and stored. Then, in Sect. 5.4.2 the more specialized steps of data preparation are described, like cleansing and segmentation. The process of feature extraction and the different possible representations tested are explained in Sect. 5.4.3.

5.4.1 Data Assessment and Storage

During the period of one year, over one billion measurements from 30 different sensor types have been recorded during the processing of about 10,000 steel blocks, together with according quality information. Among the readings are the air temperature for each furnace zone, the rolling speed, force, position and temperature, which domain experts consider to be the most relevant quality-related parameters. For validation purposes and guaranteeing the reproducibility of results, all data has been stored in a single SQL database.

Figure 5.4 shows an excerpt of the database schema, representing the most important tables and relationships. The steel blocks resulting from a single cast can be divided according to different customer orders. Steel blocks from a single order are usually

inserted into the furnace together. For each order, quality information about the bars that were cut from each block is available. Each row consists of the test results for several bars. In only a few cases it is possible to relate the bars back to the steel block they originally were cut from, based on the last two digits of their ID. For the few cases were such tracking is possible, our project partners have introduced a weighted sum calculation that derives a single label from multiple types of quality information available for all bars [KLD13].

A tool developed in the Java programming language allows for reading in the raw data delivered in different files and formats and transforming them into the shown database schema. Once imported, sensor measurements can be exported based on filters written in SQL. Exported are several CSV files, where each contains all measurements recorded by a particular sensor, at a particular processing station, for a single steel block. The individual CSV files are then read in by the previously described RapidMiner process, which preprocesses them as described in the following two subsections.

5.4.2 Data Preparation Steps

At first, all value series are correctly aligned. The alignment step mainly consists of scaling operations, to bring value series of different resolutions to the same length. After scale up, values between two original values were inserted by a simple linear interpolation.

Afterwards, the value series are cleansed. Irrelevant parts where no processing happened are cut away, as discussed in Sect. 5.3.1. The operation is highly dependent on processing station and sensor type. For instance, at the block roll, intervals with a rolling force appearing too low for any processing to happen are marked as irrelevant. The same intervals are then cut away also from other value series at this processing station. At other rolls, the height of the roll is a better indicator for no processing happening. This type of cleansing must also be done for individual segments, before the extraction of local features. Then, measurements lying outside meaningful value ranges are marked as outliers and replaced by their predecessor value. Temperature measurements from the block roll could not be used in many cases, because the sensor was defect. Value series from this sensor therefore could not be processed any further.

The technical system from which value series are received already replaces missing values, or takes the average of values in cases where the queried resolution is lower than during recording. Therefore, no imputation is necessary. No value series is smoothed, but many are normalized such that their values are in the $[0, 1]$ range.

The value series are either segmented based on domain knowledge, or based on a received signal which indicates the switch to another rolling step. In case of the heating furnace, for instance, the five different heating zones make up natural borders for the segments. Similarly, individual rolling steps seem to be natural divisions for all series stemming from the three different rolls. At the block roll, a change in the signal counting the number of rolling steps directly indicates the beginning of a new division. At the finishing rolls, due to the aforementioned signal not being available, the rolling force can be used accordingly, as longer segments with zero force indicate the period of no

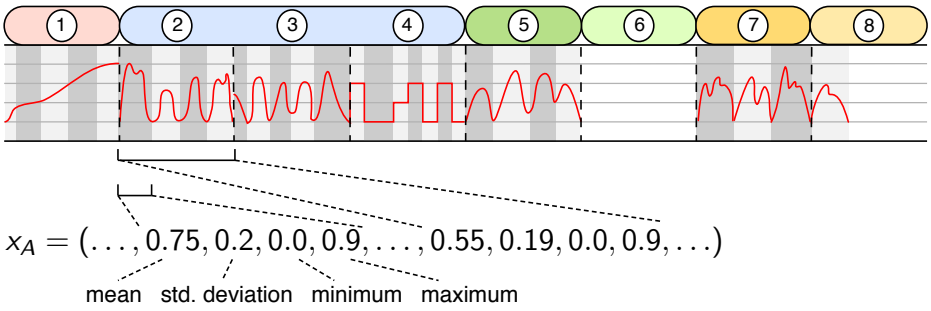


Figure 5.5: Segmentation of value series and encoding of descriptive statistics about these segments in a fixed-length feature vector. Alternating gray values indicate the segments.

processing between rolling steps. It should be noted here that in practice, even seemingly simple tests like the ones described are not always easy to implement. For example, the rolling force sensor will catch vibrations of the roll, even without any processing happening. Therefore, it will not deliver values exactly equaling zero, but values that oscillate around zero instead. In such cases, it sometimes can be difficult to manually devise global thresholds that separate valid signals from background noise.

5.4.3 Choice of Representation and Features

The following subsections describe three different types of representations to which the value series from our case study can be mapped. How performance, i.e. accuracy and run-time, changes with representation is evaluated in Sect. 5.5.

Propositional Representation and Extracted Features The types of features extracted from the value series in our case study are *global* features, *local* features and *aggregates* of local features. Global means that features are extracted from the value series as a whole, and local means that features are only based on individual segments. Aggregates are features over combinations of local features, for instance over the features from two consecutive segments.

Global and local features are simple statistics. Summary statistics are the mean, the standard deviation, minimum and maximum values, the length of the series, and the area under the curve. Other features are value differences between start and end point of the series, and histograms. Aggregate features are calculated across segments, and describe, for instance, the difference between the means of two consecutive segments, the mean and standard deviation over the means of segments, or the mean of differences between the means of segments, etc. Thereby the value series can be represented at different levels of granularity.

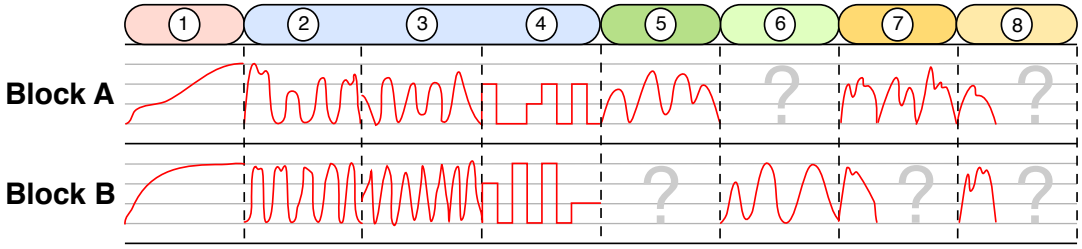


Figure 5.6: The value series from different sensors as a single fixed-length vector

In the end, features extracted from all value series associated with a single steel block end up at predefined positions in a fixed length feature vector (see Fig. 5.5). The biggest advantage of such an approach is that it is multivariate in the sense that features from different value series and their parts, at different levels, may be combined in a highly interpretable manner. For example, a classification rule that is formed based on such features may read like "Predict the rods cut from a steel block as defect if it was heated less than one hour at 900 degree Celsius and the maximum rolling force in the first rolling step exceeds the value of 10,000". In Sect. 5.5 it will be shown that the features allow for a meaningful interpretation of observations in terms of coarse grained patterns, like processing modes. At the same time, the up to 60,000 raw series values from each steel block are reduced to about 2,000 features. For the modeling steps described in Sect. 5.5, this set of features is even further reduced to only 218 features, since features about individual segments seem not to be correlated with quality.

Mapping of Series Values to a Fixed-Length Vector As explained in Sect. 5.3.2, raw value series can also be compared with the help of different distance measures. One popular distance measure is Euclidean distance, however, it only works with properly aligned series of fixed length. The procedure shown in Alg. 5 can be used to output the raw series values after data preparation, before extraction of the previously described features. Once such value series are obtained, they can be projected to appropriate (predefined) positions of a fixed-length vector, as shown in Fig. 5.6.

In theory, the resulting vectors may now be used with all kinds of distance based methods, like k-NN, k-Means clustering, or the SVM with RBF-kernel. However, it is unclear how to handle certain parts of the fixed-length vector. The first question that arises is which values to assign to portions where no processing happened (the question marks in Fig. 5.6), which could be, for instance, optional processing steps. A simple approach might be to fill the missing portions with zeros or the last recorded value. However, filling with zero values can easily lead to problems with several distance measures. For example, how similar are two value series, where steel block A moved through a different finishing roll than did steel block B? When filling with zeros, both series would be marked as highly dissimilar by Euclidean distance, although both blocks could

well lead to a similar final quality of the steel blocks. In such a case, the desired correspondence between similar feature vectors and similar labels would be lost. Reserving the same portion for both finishing rolls (sensors 5 and 6) in the fixed-length vector seems to solve the problem, but it doesn't take into account that both finishing rolls might have somewhat different properties, e.g. value scales, which usually requires a careful normalization. Moreover, the solution would not be transferable to a situation where parts of a steel block are processed in parallel, like at the cooling stations (sensors 7 and 8). There are other methods to fill the missing portions, like inserting the mean over all value series, however, this would introduce some kind of "ghost" processing which never really happened. Similar problems result from different numbers of rolling steps and according segments.

Although filling missing parts with zeros or the mean doesn't look too promising, the fixed-length vector representation of raw values has been used for comparisons in the modeling step.

Concatenation of Value Series Instead of mapping all series values to a fixed-length vector by rescaling, another option is to use distance measures that can handle value series with different lengths, like DTW or LCSS. In principle, there are two approaches for transforming the original time series appropriately. The first approach simply concatenates all value series belonging to the processing of a single steel block. The resulting series might then be compared with one of the aforementioned distance measures. Given data about the processing of two steel blocks, A and B, the second approach calculates distance values for each value series of each sensor independently and then sums them up to a total distance.

The described approach in principle leads to similar problems as the previous one, in the sense that certain parts of the value series with entirely different processing of the steel blocks are hard to compare. Nevertheless, the approach of concatenating all value series has been compared with the extraction of simple statistics as features.

5.5 Data Analysis and Prediction Results

In the modeling step, a goal was to find out what can be predicted at all, knowing how steel blocks were processed at *all* processing stations. If quality cannot be predicted sufficiently with data from all processing stations, it is unlikely that it could be predicted early with less information available.

For modeling, the feature vectors of 470 processes for which the relation between steel blocks and the bars cut from them could be established were first analyzed with different learning methods, like Naïve Bayes, Decision Trees, k-NN and the SVM (see Sect. 3.2). It soon turned out that including features about the individual segments decreases accuracy in comparison to only including global information about the value series and segments. Features of individual segments were therefore excluded for the following analysis, resulting in 218 remaining features. However, even with the reduced feature

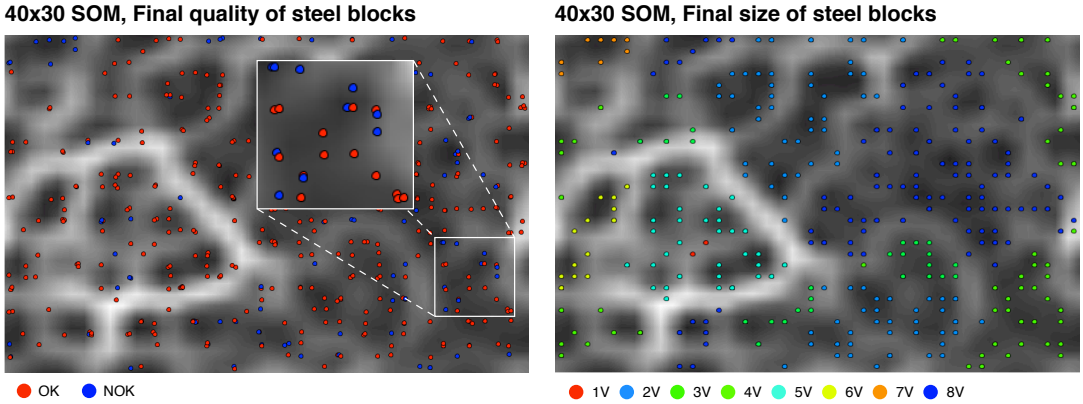


Figure 5.7: Similarity relationships between feature vectors

set, none of the classifiers mentioned before could reach a significantly better prediction accuracy than the baseline, which predicts the majority label.

For getting a better impression of the data, the feature vectors were mapped to a two dimensional *Self-Organizing Map* (SOM) [Koh89] and colored according to different types of meta information (see Fig. 5.7). Points lying close to each other on the map have similar feature vectors. The shading indicates a weighted distance between the points, where lighter shades represent a larger distance.

In the SOM on the left hand side, the points represent the feature vectors of production processes and their color the final quality of the resulting steel bars as discretized values, “okay" (OK) and “not okay" (NOK). In many cases, processes leading to a low final quality of the bars are lying very close to processes resulting in a high quality (see also the zoomed area in Fig. 5.7), meaning they have highly similar feature vectors. As it seems, the features extracted so far do not suffice to distinguish well between low and high quality processes, explaining the previously mentioned prediction results.

In comparison, the SOM on the right hand side of Fig. 5.7 shows the final size of the resulting steel bars. Here, processes resulting in the same size form large continuous areas on the SOM, i.e. their feature vectors are similar. As it seems, the features extracted are thus highly correlated with distinct operational modes for the different bar sizes produced. The hypothesis could be verified by training a decision tree on features of the first finishing roll (see Fig. 5.8). The accuracy as estimated by a 10-fold cross validation is 90%, while k-NN (k=11) even achieves 97%. Most important for the decision is the position of the roll (sensor 501). Domain experts have verified that the results reflect the real modes of operation in the rolling mill.

Through concatenating raw series values and comparing them by using DTW distance, or projecting them to a fixed length vector, using Euclidean distance for comparison, processing modes could be predicted with similar accuracy. An advantage of the distance based approach on raw values is that it requires no segmentation of value series, which is highly domain specific and requires lots of expert knowledge. However,

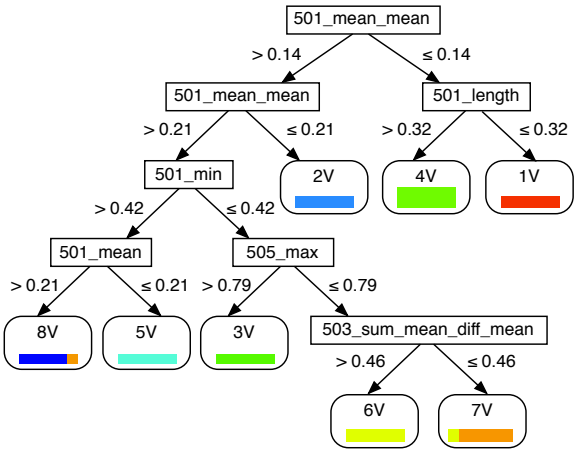


Figure 5.8: Decision tree for predicting the final size of steel bars

using distances calculated on raw values also comes with two disadvantages. The first is that the number of raw values is much higher than the number of extracted features (60,000 vs. 218 in our case study). Given that distance based learning methods often have quadratic running time, using raw values leads to a much higher running time. The second disadvantage is that distances between raw series values can be hard to interpret. In contrast, the statistics we extract are easier to interpret. Especially, the decision tree has reduced the 218 features to an even smaller set of relevant features, which could be discussed with the domain experts.

The description and prediction of operational modes has value in its own right. For the first time, it has become possible to quantify deviations from the targeted processing, which could be used, for instance, for automatic monitoring purposes. This kind of information was previously unknown, and has been made available by a combination of the proposed feature extraction approach and different kinds of learning methods, like SOMs, decision trees and k-NN.

To improve on quality prediction, other feature extraction methods have been evaluated, like coefficients of the Discrete Fourier Transform (DFT) and the transformation of value series into word vectors through symbolization. No extraction method tried has lead to significantly better results than those presented so far. One hypothesis is that the rolling alone cannot explain the differences in final quality. Therefore, the next step is to combine data from the hot rolling process with data from melting, which comes *before* rolling.

5.6 Summary, Conclusions and Outlook

IoT devices generate data which is time-related. At the beginning of this chapter, it was shown how real-world data may look like, based on a hot rolling mill case study

from the field of smart manufacturing. It has been made clear that values series have a much different form than observations in propositional format. This has lead to a more formal definition of value series and the general problem of feature extraction and representation learning, which is a hard task. Several standard approaches for the preprocessing of value series have been presented. However, not many of them are directly applicable in our case study, since each observation is not a single value series, but a set of value series. This difference has lead to the development of a preprocessing algorithm for value series from production processes. The algorithm is highly modularized and generic concerning the process of feature extraction. The only domain-specific parts are data preparation processes which depend on the particular machine and sensor. Unfortunately, for a given application, each machine and sensor may require a different kind of data preparation, resulting in a large number of such processes. In the context of the case study, all processes have been implemented in RapidMiner. Segmentation is based on the signals of specific value series. The developed feature extraction process extracts global features from the whole value series, local features from each segment, and aggregates over local features. Features are mainly statistics which are easy to calculate and interpret. In the modeling step, such features were used with different classifiers to predict the quality of rods finally cut from each steel block. While quality was hard to predict, through the use of SOMs, decision trees and k-NN it has become possible to identify and quantify operational modes. The quantification of deviations from targeted processing is information made available by data analysis, which previously didn't exist in this form. It could be used, for instance, for the automatic monitoring of processes.

As the results demonstrate, data analysis methods are able to detect meaningful patterns in production processes. As the results also show, however, finding exactly those features which are relevant for the prediction task is not always straightforward. The next step is to gather data from the melting process, which comes before rolling, and combine it with the sensor data which is already getting recorded.

One problem mentioned, but not having been discussed in more detail so far, is how to deal with cases where quality information is not available for individual steel blocks, but whole charges of blocks. In this case, we need to learn a model for predicting individual labels based on summary statistics about the labels. This relatively novel kind of learning problem is called *learning from label proportions*. The learning task is defined and explained in more detail in the next Chap. 6, where a new algorithm for the problem is going to be developed. Learning from aggregate data is an interesting new field of research which has high relevance for smart manufacturing processes. In fact, also data about the melting process will be aggregated over several charges of steel blocks, while the information is needed for single steel blocks, leading to problems of tracking object identity again.

Another problem mentioned is that the tighter the feedback loop between data analysis and control becomes, the less time will be available for making predictions. Considering that preprocessing takes also time, we expect to end up with similar questions as in distributed learning from vertically partitioned data, namely which data to process locally, and which data to send to a central server or the next machine for further

processing. Two communication-efficient algorithms for the scenario will be introduced in Chap. 7 and Chap. 8, after discussing learning from label proportions. It will be shown that both problems are even related, in the sense that learning from label counts may reduce the communication between networked nodes. While sequential interlinked production processes don't match the problems of distributed learning in the vertically partitioned data scenario exactly, the learning task is relevant in its own right. For instance, production processes in discrete manufacturing are much more parallelized and constraint, with many parts being concurrently processed and assembled. Similarly, processes in logistics are highly distributed, concurrent and much more communication-restraint, due to the use of small devices and wireless network technology. Further, we will see how communication-efficient algorithms could lead to much more robust and fault-tolerant traffic prediction systems. In general, the trend is to instrument more and more devices with wireless sensors, requiring a run-time efficient automatic preprocessing of sensor data and communication-efficient distributed algorithms which can learn from the preprocessed data.

Part II

Algorithms

Chapter 6

Learning from Label Proportions

In a supervised learning scenario, we learn a mapping from input to output values, based on labeled examples. Can we learn such a mapping also from groups of unlabeled observations, only knowing, for each group, the proportion of observations with a particular label? Solutions have real world applications:

- In smart manufacturing settings like the one presented in the previous case study, quality information is sometimes only given in statistical form, for samples of products. Can we derive a model based on this information which assigns the correct quality information to individual products?
- After democratic elections, the percentages of parties which were elected in each district are published. Governmental agencies may obtain additional information about people living in each district. Is it possible to reconstruct who voted for which party, based on such information?
- In distributed settings, like the vertically partitioned data scenario, we might like to reduce communication costs by transmitting only aggregated label information between nodes. Can we learn a model that is sufficiently accurate in assigning class labels to individual instances, only based on aggregated label information?

The problem of learning from label proportions not only deviates from that of supervised learning, where we learn from individually labeled training examples, but also from many other learning settings known in machine learning and data mining. It is different from *semi-supervised learning* [CSZ06], where we are given at least some examples that are labeled. It is not strictly *unsupervised learning*, since we are given at least *some* additional information about labels. It is different from *anomaly and outlier detection*, where we might know about observations that belong to a normal class. It comes close to *multiple instance learning* [WEH11], where whole bags of observations are either labeled as positive or negative. However, learning from label proportions is not exactly the same problem, since we are not given binary information on each bag, but real-valued statistical information about the labels in each bag.

In the following Sect. 6.1, the task of learning from label proportions is defined more formally and illustrated with more concrete examples. Then, Sect. 6.2 gives an overview of related work. Since the problem is relatively novel in machine learning research, it isn't as well understood from a theoretical point of view as other learning tasks. Nevertheless, as will be shown, some bounds have been proven. In Sect. 6.3, we discuss the difficulty of the problem from a more Bayesian perspective. Afterwards, Sect. 6.4 defines loss functions for the scenario. Section 6.5, introduces a clustering approach and variants that minimize aforementioned loss functions. The approach possesses many positive properties which existing state-of-the-art methods don't share to the same extent. In Sect. 6.6, we compare the algorithm's prediction performance and run-time to other existing methods. Finally, Sect. 6.7 gives a short summary, concludes, and gives an outlook on learning from label proportions in the context of vertically partitioned data.

6.1 The Problem of Learning from Label Proportions

To the best of our knowledge, [MCO07] were the first who formulated both the classification and regression tasks of the problem in a more formal way. We extend their problem definition to multi-class problems and relate it to the unknown joint distribution $P(X, Y)$ from which all observations and labels are drawn.

Definition 6.1.1 (Learning from label proportions) Let X be an instance space and Y be a set of categorical class labels $Y_1, \ldots, Y_l$. Let $P(X, Y)$ be an unknown joint distribution on the instances and class labels. In the setting of *learning from label proportions*, we are given a sample of unlabeled observations $S = \langle x_1, \ldots, x_n \rangle$ with x_i from instance space X, drawn i.i.d. from $P(X, Y)$ and then having their label $y_i \in Y$ removed. Furthermore, we are given a partitioning of S into h disjunct bags $B_1, \ldots, B_h$. For each bag B_u and label Y_v, we are also given the proportions $\pi_{uv} \in [0, 1]$ of that label in bag B_u. Only based on this information, we seek a function (model) $\hat{f} : X \to Y$ that predicts $y_i \in Y$ for an observation $x_i \in X$ drawn i.i.d. from P, such that the expected risk

$$R_{\text{exp}} = \int \ell(y, \hat{f}(x)) dP(x, y)$$

is minimized. Here, ℓ is a convex *loss function* $\ell : Y \times Y \to \mathbb{R}_0^+$ which measures the cost of assigning the wrong label to individual observations.

The given label proportions π_{uv} can more conveniently be written as a $h \times l$ matrix $\mathbf{\Pi} = (\pi_{uv})$, where the values in a row $\mathbf{\Pi}_{u,\cdot} = (\pi_{u1}, \ldots, \pi_{ul})$ sum up to one. The frequency count μ_{uv} of observations with label $Y_v \in Y$ in bag B_u can easily be reconstructed by multiplying the label proportion π_{uv} with bag size $|B_u|$.

The proportion $\eta(\mathbf{\Pi}, Y_v)$ of label Y_v over the whole sample S can then be calculated from $\mathbf{\Pi}$ as the sum of the frequency counts for bag u, divided by the total number of observations n:

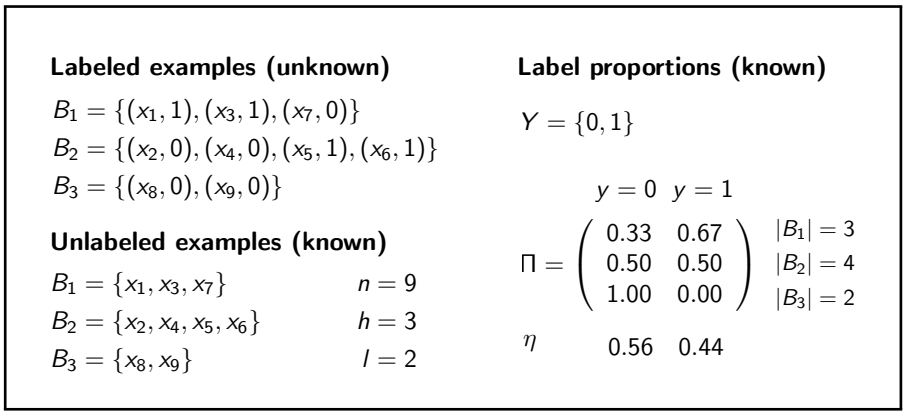


Figure 6.1: Example for given bags of observations, a label proportion matrix, and related notations

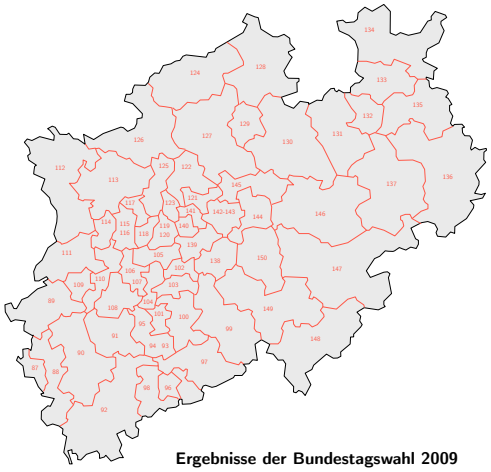
$$\eta(\mathbf{\Pi}, Y_v) = \frac{1}{n} \sum_{u=1}^{h} \mu_{uv} \,. \tag{6.1}$$

In the following, the problem of learning from label proportions is illustrated by giving different examples and applications, like the preservation of privacy in democratic free elections and tracking objects in smart manufacturing.

Example 6.1.1 (Label proportion matrix) Figure 6.1 gives an example of the notations previously introduced, the division of observations into disjunct bags, and the label proportion matrix as derived from the original (now unknown) labels.

Example 6.1.2 (Democratic free elections) What can be learned from aggregated information plays an important role in the field of privacy-preserving data mining. For instance, consider democratic free elections. On the one hand, there is a demand for *privacy*. It must remain secret what each individual citizen has voted for. On the other hand, there is a demand for *transperancy*. It must be made clear how final sums in the total election results were calculated from sums that stem from individual electoral districts. Adhering to the demand for transparency, in Germany the individual election results from all 299 districts are published in daily newspapers, usually on the next day after the election. Is it possible to derive what individual citizens have voted for, based on aggregated information about the votes per district?

The problem can be turned into the task of learning from label proportions if in addition to the public election results, information about individuals in each district can be obtained. Then, the districts can be seen as a division of all individuals $x \in S$ into disjunct bags $B_1, \ldots, B_h$, with the election results per district being the proportion (or frequency count) of individuals having voted for each party (see Fig. 6.2).



Ergebnisse der Bundestagswahl 2009
für die Wahlkreise 100-105

Kreis	Anzahl	CDU	SPD	Grüne	FDP	Die Linke
			...			
100	150,609	48.4%	27.8%	10.1%	8.8%	6.5%
101	166,669	50.0%	26.7%	8.3%	8.6%	5.2%
102	146,460	35.5%	37.1%	9.3%	7.8%	8.1%
103	145,089	35.8%	34.3%	9.3%	8.3%	10.5%
104	162,192	39.0%	34.3%	8.1%	8.7%	8.4%
105	150,715	33.8%	44.5%	7.1%	7.8%	5.6%
			...			

Figure 6.2: Election results from 2009 for the districts 100-105

Given that people provide more and more information about themselves in public social networks or business contexts and data is gathered even without their knowledge by governmental intelligence agencies, research on learning from label proportions has a high actuality for citizen's privacy. The biggest problem here is that information which looks harmless all by itself, and which has been gathered in entirely different contexts, like business transactions and during elections, could be easily brought together and used for malicious purposes.

Example 6.1.3 (Tracking of object identity) In smart manufacturing, it can be difficult to track products through the whole process chain. For instance, in the hot rolling mill case study presented in Sect. 5, steel blocks are too hot to be stamped or to be equipped with RFID chips. Once cut to smaller rods, tracking object identity, i.e. which rods belonged to which block in which customer order (or charge), can become a big technical and logistic challenge (see Fig. 6.3). In the scenario, quality labels are usually given as percentages for whole charges, but not for individual blocks. The task is to learn a model which predicts the final quality of individual blocks, only based on the aggregated label information per charge. Formulated like this,

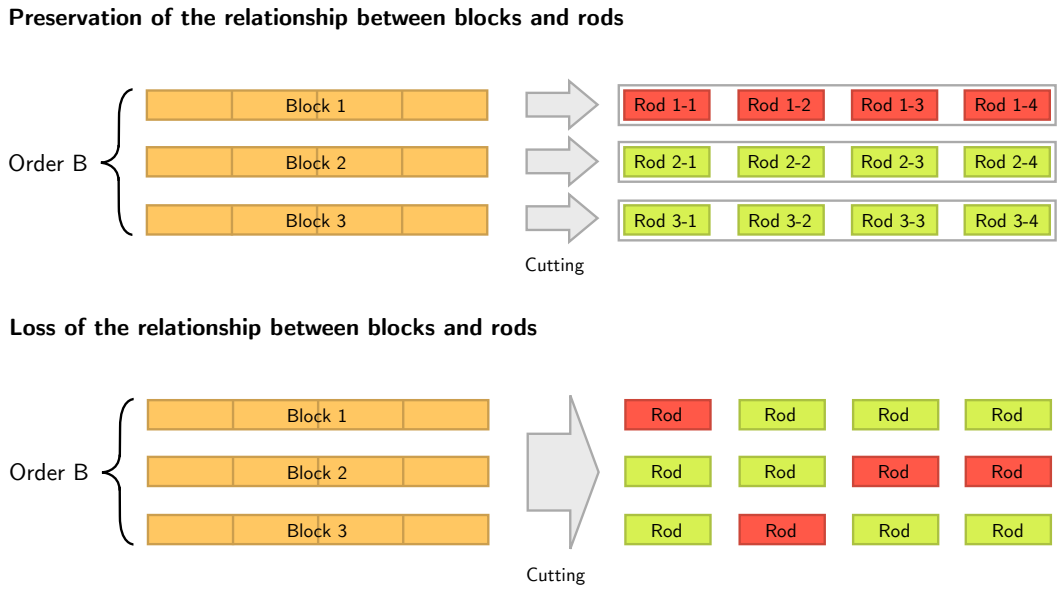


Figure 6.3: Relationship between blocks and rods getting lost

the problem maps directly to that of learning from label proportions, with customer orders or charges dividing the set of all steel blocks into disjunct bags, and proportions of quality labels given for each bag.

6.2 Related Work

When starting the work on learning from label proportions in 2010, only a few publications on the topic were available. Since then, more papers have been published and a workshop on the more general topic of learning and privacy with incomplete data and weak supervision has been held at the annual conference on Neural Information Processing Systems (NIPS) in 2015. The following subsections present related work, in so far as it seems relevant regarding the clustering approach going to be developed in this chapter.

Related Semi-Supervised Methods There are some approaches which seem similar to the scenario of learning from label proportions, but are in truth semi-supervised learning tasks. For instance, the authors of [DKS02] first cluster the given data with SOMs and then label the resulting clusters. However, labeled observations are given, which are usually not available when learning from label proportions. In [DBBE99], the k-Means optimization problem is adapted to respect labeled data. Again, this is a semi-supervised setting, with labeled observations.

Basic Methods To the best of our knowledge, the authors of [KdF05] were the first who introduced the problem of learning from label proportions. They propose a probabilistic model trained by an efficient Markov-Chain-Monte-Carlo (MCMC) sampling algorithm. The authors of [MCO07] were the first who defined the problem of learning from aggregate values for regression and classification tasks in a more formal way. They modified well-known methods like k-NN [Aha92], backpropagation neural networks [Mit97] and the linear SVM [Vap99] to respect the given label proportions. Their experimental results focus on regression tasks, while we are mainly interested in classification.

Mean Map The Mean Map method we use for comparisons in Sect. 6.6 has been proposed in [QSCL09]. It estimates the conditional class probability $P(Y|X,\theta)$ by conditional exponential models, using a joint feature map $\phi : X \times Y \to \mathcal{H}$ that maps observations $\mathbf{x} \in \mathbb{R}^p$ and labels $y \in Y = \{Y_1, \ldots, Y_v\}$ into a new feature space $\mathcal{H}$ with kernel $k((\mathbf{x}, y), (\mathbf{x}', y'))$, and normalization function g:

$$P(Y|X,\boldsymbol{\theta}) = \exp(\,\langle \phi(X,Y), \boldsymbol{\theta}\rangle - g(\boldsymbol{\theta}|X)\,)\,. \tag{6.2}$$

The parameter θ can then be estimated by solving a convex maximization problem for the conditional log-likelihood $\log P(Y|X,\boldsymbol{\theta})$:

$$\log P(Y|X,\boldsymbol{\theta}) = \sum_{i=1}^{n} [\langle \phi(\mathbf{x}_i, y_i), \boldsymbol{\theta}\rangle - g(\boldsymbol{\theta}|\mathbf{x}_i)] = n\langle \boldsymbol{\mu}_{XY}, \boldsymbol{\theta}\rangle - \sum_{i=1}^{n} g(\boldsymbol{\theta}|\mathbf{x}_i)\,. \tag{6.3}$$

It becomes apparent then that the conditional log-likelihood can be expressed in terms of the so called *mean operator* defined as

$$\boldsymbol{\mu}_{XY} := \mathbb{E}_{XY}[\phi(\mathbf{x}, y)]\,. \tag{6.4}$$

Although the authors derive their method for arbitrary feature maps $\phi(\mathbf{x}, y)$, for ease of notation let us assume the special case where $\phi(\mathbf{x}, y)$ factorizes into $\Psi(\mathbf{x}) \otimes \varphi(y)$. Let's further assume that $\Psi(\mathbf{x}) = \mathbf{x}$. Then, the mean operator can be expanded into its bag-wise and label-wise components as follows:

$$\begin{aligned}
\boldsymbol{\mu}_{XY} &= \mathbb{E}_{XY}[\phi(\mathbf{x}, y)] && (6.5)\\
&= \sum_{u=1}^{h} \frac{|B_u|}{n} \mathbb{E}_{XY}[\phi(\mathbf{x}, y)|u] && (6.6)\\
&= \sum_{u=1}^{h} \frac{|B_u|}{n} \sum_{v=1}^{l} \varphi(Y_v) P(Y_v|u) \mathbb{E}_{XY}[\mathbf{x}|Y_v, u] && (6.7)\\
&= \sum_{u=1}^{h} \frac{|B_u|}{n} \sum_{v=1}^{l} \varphi(Y_v) \pi_{uv} \mathbb{E}_{XY}[\mathbf{x}|Y_v, u]\,. && (6.8)
\end{aligned}$$

Algorithm 6 Mean Map algorithm

procedure MEANMAP($\mathbf{\Pi}$,S,$\mathcal{B}$,Y,λ)
  for $u \leftarrow 1, h$ **do**
    $\mathbb{E}_{XY}[\mathbf{x}|u] \leftarrow \frac{1}{|B_u|} \sum_{\mathbf{x} \in B_u} \mathbf{x}$
  end for
  $\mathbb{E}_{XY}[\mathbf{x}|y] \leftarrow (\mathbf{\Pi}^\top \mathbf{\Pi})^{-1} \mathbf{\Pi}^\top \mathbb{E}_{XY}[\mathbf{x}|u]$
  $\boldsymbol{\mu}_{XY} \leftarrow \sum_{v=1}^{l} P(Y_v) \mathbb{E}_{XY}[\mathbf{x}|Y_v]$
  Solve the minimization problem

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[\sum_{i=1}^{n} g(\boldsymbol{\theta}|\mathbf{x}_i) - n\langle \boldsymbol{\mu}_{XY}, \boldsymbol{\theta} \rangle + \lambda ||\boldsymbol{\theta}||^2 \right]$$

  return $\boldsymbol{\theta}^*$
end procedure

The only unknown quantities in the above formulation are the $2h$ p-dimensional vectors $\mathbb{E}_{XY}[\mathbf{x}|y,u]$. These vectors are the solution of a linear system which can be expressed by means of the law of total probability as follows:

$$\mathbb{E}_{XY}[\mathbf{x}|u] = \sum_{v=1}^{l} \pi_{uv} \mathbb{E}_{XY}[\mathbf{x}|Y_v, u] \tag{6.9}$$

The expectations $\mathbb{E}_{XY}[\mathbf{x}|u]$ on the left-hand side of the linear system can be estimated from the data, without knowing the labels, and can be calculated as the average over the feature vectors in each bag. The system has to be solved separately for each of the p dimensions, such that the total number of equations is $h \times p$. However, since there are $2h \times p$ unknowns, the system is underdetermined.

The authors of Mean Map turn the system of equations into a well-formed system with $2p$ unknowns by making a *homogeneity assumption*, which states the conditional independence of feature vectors from bags, given the label:

$$\forall u \ : \ \mathbb{E}_{XY}[\mathbf{x}|y,u] = \mathbb{E}_{XY}[\mathbf{x}|y] \,. \tag{6.10}$$

In other words, the feature vectors to be expected, given an arbitrary label $Y_v \in Y$, should be the same as the expected feature vectors when we know label Y_v and the bag B_u they are stemming from. Estimation of the mean operator then simplifies to

$$\hat{\boldsymbol{\mu}}_{XY} = \sum_{v=1}^{l} P(Y_v) \mathbb{E}_{XY}[\mathbf{x}|Y_v] \,, \tag{6.11}$$

with

$$\mathbb{E}_{XY}[\mathbf{x}|y] = (\mathbf{\Pi}^\top \mathbf{\Pi})^{-1} \mathbf{\Pi}^\top \mathbb{E}_{XY}[\mathbf{x}|u] \tag{6.12}$$

being the solutions of the linear system of equations as found by pseudo-inversion. The quantity $P(Y_v)$ can be empirically estimated by taking the average over all proportions given for label Y_v. Once $\boldsymbol{\mu}_{XY}$ is estimated, the parameter vector $\boldsymbol{\theta}$ can be derived by standard methods for maximum likelihood estimation. All aforementioned steps of the Mean Map method are shown in Alg. 6.

In [QSCL09], Mean Map is compared to kernel density estimation, discriminative sorting, and MCMC [KdF05]. Mean Map outperformed the related techniques in terms of accuracy.

Laplacian Mean Map Mean Map's homogeneity assumption is quite restrictive: It assumes that the feature vectors in each bag are similarly distributed as the feature vectors over all bags, given the label. This assumption doesn't necessarily hold, for instance, in the case of democratic free elections: Knowing which party a person has voted for might give us an idea about the person's income, but knowing in addition the region or district this person is coming from would certainly also influence our decision. In [PNCR14], the homogeneity assumption is therefore relaxed to

$$\forall u, u' \quad \text{if} \quad u \approx u' \quad \text{then} \quad \mathbb{E}_{XY}[\mathbf{x}|y,u] \approx \mathbb{E}_{XY}[\mathbf{x}|y,u']\,. \tag{6.13}$$

In other words, whenever bags are similar to each other, it is assumed that also their feature vectors are similarly distributed, given the label. The similarities between bags are domain-specific, and assumed to be given as parameters $v_{u,u'} \geq 0$. Another approach would be, for instance, to measure the similarity of bags by the distance of their mean feature vectors, as is also proposed by the authors.

The authors of [PNCR14] restrict themselves to a non-kernelized version of Mean Map, dropping the joint feature map ϕ, and further to binary classification problems for which $Y = \{-1, +1\}$. In the following, the proportion of positively labeled observations in bag u will be denoted as π_u, while $1 - \pi_u$ is the proportion of negatively labeled observations. Let $\boldsymbol{\Pi} = [\mathrm{DIAG}(\boldsymbol{\pi})|\mathrm{DIAG}(\mathbf{1}-\boldsymbol{\pi})]^\top \in \mathbb{R}^{2h\times h}$ be a matrix with proportions for each label and bag, $\mathbf{b}_u^y = \mathbb{E}_{XY}[\mathbf{x}|y,u]$ denote the $2h$ unknown p-dimensional vectors and $\mathbf{b}_u = (1/|B_u|)\sum_{\mathbf{x}\in B_u}\mathbf{x}$ denote the h averages over all observations in bag u.

As in the original Mean Map method, $\boldsymbol{\mu}_{XY}$ can be retrieved from the $2h$ bag-wise, label-wise unknown averages $\mathbf{b}_u^y$:

$$\boldsymbol{\mu}_{XY} = \frac{1}{2}\sum_{u=1}^{h}\frac{|B_u|}{n}\sum_{v=1}^{l}(2\pi_u + Y_v(1-Y_v))\mathbf{b}_u^{Y_v}\,. \tag{6.14}$$

The system of linear equations in matrix form can then be stated as

$$\mathbf{B} - \boldsymbol{\Pi}^\top\mathbf{B}^{\pm} = \mathbf{0}, \tag{6.15}$$

where $\mathbf{B} = [\mathbf{b}_1|\mathbf{b}_2|\ldots|\mathbf{b}_n]^\top \in R^{h\times p}$ and $\mathbf{B}^{\pm} \in R^{2h\times p}$ is the matrix of unknowns

$$\mathbf{B}^{\pm} = \left[\mathbf{b}_1^{+1}|\mathbf{b}_2^{+1}|\ldots|\mathbf{b}_n^{+1} \mid \mathbf{b}_1^{-1}|\mathbf{b}_2^{-1}|\ldots|\mathbf{b}_n^{-1}\right]^\top\,. \tag{6.16}$$

Algorithm 7 Laplacian Mean Map (LMM)

1: **procedure** LMM($\mathbf{\Pi}$,S,$\mathcal{B}$,Y,λ,γ,$\mathbf{V}$)
2: $\quad \mathbf{B}^{\pm} \leftarrow \left(\mathbf{\Pi}\mathbf{D}_{\boldsymbol{\omega}}\mathbf{\Pi}^{\top} + \gamma\mathbf{L}\right)^{-1}\mathbf{\Pi}\mathbf{D}_{\boldsymbol{\omega}}\mathbf{B}$
3: $\quad \boldsymbol{\mu}_{XY} \leftarrow \frac{1}{2}\sum_{u=1}^{h}\frac{|B_u|}{n}\sum_{v=1}^{l}(2\pi_u + Y_v(1-Y_v))\mathbf{b}_u^{Y_v}$
4: $\quad \boldsymbol{\theta}^* \leftarrow \operatorname{argmin}_{\boldsymbol{\theta}}\sum_{i=1}^{n} g(\boldsymbol{\theta}|\mathbf{x}_i) - n\langle\boldsymbol{\mu}_{XY},\boldsymbol{\theta}\rangle + \lambda||\boldsymbol{\theta}||^2$
5: **end procedure**

As we have already seen for Mean Map, with $h \times p$ equations and $2h \times p$ unknowns, the system is underdetermined. Instead of making the aforementioned restrictive homogeneity assumption (6.10), however, the authors of [PNCR14] encode their relaxed assumption (6.13) into the following regularized least squares minimization problem:

$$\begin{aligned}\underset{\mathbf{b}_u^{+1},\mathbf{b}_u^{-1}}{\operatorname{argmin}} &\sum_u (\mathbf{b}_u - \pi_u\mathbf{b}_u^{+1} + (1-\pi_u)\mathbf{b}_u^{-1})^2 \\ &+\gamma\sum_{u,u'} v_{u,u'}\left[(\mathbf{b}_u^{+1} - \mathbf{b}_{u'}^{+1})^2 + (\mathbf{b}_u^{-1} - \mathbf{b}_{u'}^{-1})^2\right]. \end{aligned} \tag{6.17}$$

The second part of the minimization problem enforces that the estimates for $\mathbf{b}_u^y$ are close together whenever the bags are similar, as given by the similarities $v_{u,u'}$. How much this is enforced can be controlled by the regularization strength parameter $\gamma \geq 0$.

The problem can be rewritten in matrix form by the Laplacian of the symmetric matrix $\mathbf{V} \in \mathbb{R}^{h\times h}$ whose entries consist of the similarities between bags. The Laplacian is defined as $\mathbf{L}_a = \mathbf{D} - \mathbf{V}$, where $\mathbf{D}$ is a diagonal matrix such that $\mathbf{D}_u = \sum_{u'} v_{u,u'}$. The Laplacian can be seen as the adjacency matrix of the graph induced by the similarities $v_{u,u'}$. By setting

$$\mathbf{L} = \varepsilon\mathbf{I} + \left[\begin{array}{c|c} \mathbf{L}_a & \mathbf{0} \\ \mathbf{0} & \mathbf{L}_a \end{array}\right] \in \mathbb{R}^{2h\times 2h}, \tag{6.18}$$

the optimization problem becomes

$$\mathbf{B}^{\pm} = \underset{\mathbf{X}\in\mathbb{R}^{2h\times p}}{\operatorname{argmin}}\ \operatorname{tr}\left((\mathbf{B}^{\top} - \mathbf{X}^{\top}\mathbf{\Pi})\mathbf{D}_{\omega}(\mathbf{B} - \mathbf{\Pi}^{\top}\mathbf{X})\right) + \gamma\operatorname{tr}\left(\mathbf{X}^{\top}\mathbf{L}\mathbf{X}\right), \tag{6.19}$$

with $\operatorname{tr}(\cdot)$ denoting the trace of a matrix and $\mathbf{D}_{\omega} = \operatorname{DIAG}(\boldsymbol{\omega})$ being a bias matrix with non-negative elements on the diagonal $\boldsymbol{\omega}$. Such entries allow for different weightings of the importance of linear equations. As the authors show, the solution to the stated optimization problem can be obtained in closed form:

$$\mathbf{B}^{\pm} = \left(\mathbf{\Pi}\mathbf{D}_{\boldsymbol{\omega}}\mathbf{\Pi}^{\top} + \gamma\mathbf{L}\right)^{-1}\mathbf{\Pi}\mathbf{D}_{\boldsymbol{\omega}}\mathbf{B}. \tag{6.20}$$

The aforementioned steps are summarized in Alg. 7, the Laplacian Mean Map (LMM) algorithm.

In [PNCR14], LMM is compared to Mean Map, Inverse Calibration (Invcal) and the $\propto$SVM (for an explanation, see next subsections). Further, LMM is compared to the

Alternating Mean Map (AMM) algorithm which improves on the solution of LMM and has been proposed in the same work (for further details, see [PNCR14]). On simulated data, where the homogeneity is violated, it can be shown that LMM and AMM perform significantly better than Mean Map. On ten small datasets from the UCI standard repository [AN07] and for bigger datasets from the same repository, LMM and AMM outperformed Mean Map, Invcal, and the $\propto$SVM in terms of prediction performance and run-time.

It should be noted that in [PNCR14], not only LMM and AMM, but also Mean Map clearly outperformed the prediction performance of Invcal in almost all cases. This seems to contradict the results of the original paper on Invcal [Rüp10], and our own findings presented in Sect. 6.6.2, where Invcal outperforms Mean Map in many cases, on the same datasets. One possible explanation might be that Invcal has only been tested with the linear kernel in [PNCR14], while the results in [Rüp10] and our results are based on trying Invcal (but also Mean Map) with linear and non-linear kernels. Since LMM and AMM in comparison to the other methods are not kernelized, and can only handle linear decision boundaries, LMM and AMM aren't direct competitors of the clustering approach that will be introduced in Sect. 6.5. We therefore have decided to compare only to Mean Map, Invcal and AOC Kernel K-Means (see below), which are more powerful, as they can handle also non-linear decision boundaries.

Inverse Calibration The author of [Rüp10] proposes the Inverse Calibration (Invcal) method. The regression SVM (SVR) (see Sect. 3.2.5) is converted into a probabilistic classifier by applying a scaling function σ to the outputs $\hat{y} = \hat{f}(x)$, such that $\sigma(\hat{y})$ is a good estimate for $p = P(y = 1|\mathbf{x})$. In learning from label proportions, we are given no labels y_i or estimates p_i for individual observations $\mathbf{x}_i \in S$. Instead, in the case of binary classification, we are given the label proportions π_u for each bag u, i.e. the proportions of positively labeled examples in bag u. According to the author, it is only required that $\hat{f}$ predicts $y_u = \sigma^{-1}(\pi_u)$ well on average:

$$\forall u \ : \quad \frac{1}{|B_u|} \sum_{\mathbf{x} \in B_u} (\mathbf{w}\mathbf{x} + b) \approx y_u \tag{6.21}$$

In other words, the predictions of the classifier should approximate the given label proportions well, for each bag u. These constraints are integrated as auxiliary conditions into the standard SVR optimization problem. The main idea is to restate the primal

SVR problem as

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\left(\sum_{u=1}^{h}\xi_i + \sum_{u=1}^{m}\xi'_u\right) \\ \text{s.t.} \quad & \forall_{u=1}^{h} : \xi_u, \xi'_u \geq 0 \\ & \forall_{u=1}^{h} : \frac{1}{|B_u|}\sum_{\mathbf{x}\in B_u}(\mathbf{w}\mathbf{x}+b) \geq y_u - \varepsilon_u - \xi_u \\ & \forall_{u=1}^{h} : \frac{1}{|B_u|}\sum_{\mathbf{x}\in B_u}(\mathbf{w}\mathbf{x}+b) \leq y_u + \varepsilon_u - \xi'_u\,. \end{aligned}$$

As a large margin method, the formulation allows for the reduction of model complexity, while the class probability estimates for B_u are kept close to the given label proportions π_u, with ε_u being the maximum tolerable error. The primal problem can be transformed into its dual, and then solved with a standard solver for quadratic optimization.

It is shown empirically over twelve standard datasets from the UCI repository that Invcal significantly outperforms Mean Map in terms of prediction accuracy.

$\propto$SVM The $\propto$SVM proposed in [YLK+13] explicitly models the labels of individual observations. Invcal treats the mean of each bag as some kind of super-instance, and gives each bag a regression label that corresponds to the label proportions. In comparison, the $\propto$SVM is based on the idea that the label proportions, as calculated from labels assigned to individual observations, should match the given label proportions as close as possible. This criterion is encoded as an additional term into the primal problem of the standard SVM as follows:

$$\begin{aligned} \min_{\mathbf{Y},\mathbf{w},b} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\ell(y_i, \langle\mathbf{w},\mathbf{x}_i\rangle + b) + C_\pi\sum_{u=1}^{h}\ell_\pi(\hat{\pi}_u(\mathbf{Y}), \pi_u) \\ \text{s.t.} \quad & \forall_{i=1}^{n} : y_i \in \{-1, 1\}\,. \end{aligned}$$

Here, π_u denotes the given proportion of positively labeled observations in bag u (binary classification), while $\hat{\pi}_u(\mathbf{Y})$ is the proportion of positively labeled observations as calculated based on $\mathbf{Y}$. The task is to find a vector of labels $\mathbf{Y}$ such that the loss ℓ over individual observations is minimized, but also the loss ℓ_π over label proportions. Since the vector of labels $\mathbf{Y}$ is not given, but part of the optimization, one may say that minimizing over the standard loss ensures that observations lying on the same side of the hyperplane will also be assigned the same label, depending on the particular values of the trade-off parameters C and C_π.

While the formulation seems intuitive, the optimization problem is an NP-hard non-convex integer programming problem. The authors propose two different efficient algorithms for solving it, one based on an alternating optimization strategy, and another based on convex relaxation. For further details, see [YLK+13].

In experiments [YLK+13], the $\propto$SVM has been compared to Mean Map and Invcal. The $\propto$SVM outperforms both methods in terms of accuracy on several datasets from

the UCI standard repository. However, the authors do not report which significance test they used. It should be noted that in [PNCR14], results are not always in favor of the $\propto$SVM in comparison to Invcal, even on the same datasets. Moreover, Mean Map outperformed the $\propto$SVM in many cases, while Invcal outperformed Mean Map in [Rüp10].

AOC Kernel K-Means The main idea of AOC (Aggregate Output Classification) Kernel k-Means [CLQZ09] is to cluster the observations in S in such a way that clusters correspond to classes, and the assignment of observations to clusters (classes) matches the given label proportions. The authors present variants of k-Means and Kernel k-Means [DGK04], which is a kernelized version of the original k-Means algorithm.

For clustering, we may assume that $k = l$ and that cluster membership is given by a $k \times n$ matrix $\mathbf{T}$, with $\mathbf{T}_{vi} = 1$ if observation $\mathbf{x}_i$ is in cluster v and all other elements being zero. The difference of Kernel k-Means to k-Means is that the cluster centers $\mathbf{c}_1, \ldots, \mathbf{c}_k$ can no longer be written in explicit form, but have to be expressed in terms of a kernel function $k(\mathbf{x}, \mathbf{x}')$ which is induced by a given feature map ϕ. The distance calculations then become

$$||\phi(\mathbf{x}_i) - \mathbf{c}_v||^2 = k(\mathbf{x}_i, \mathbf{x}_i) - 2\frac{\sum_{j=1}^{n} \mathbf{T}_{vj} k(\mathbf{x}_i, \mathbf{x}_j)}{\sum_{j=1}^{n} \mathbf{T}_{vj}} + \frac{\sum_{j=1}^{n}\sum_{j'=1}^{n} \mathbf{T}_{vj}\mathbf{T}_{vj'} k(\mathbf{x}_j, \mathbf{x}_{j'})}{\sum_{j=1}^{n}\sum_{j'=1}^{n} \mathbf{T}_{vj}\mathbf{T}_{vj'}}, \tag{6.22}$$

which means that we need to calculate the distance of point $\mathbf{x}_i$ not only to the cluster mean $\mathbf{c_v}$, but that we compare $\mathbf{x}_i$ to all points in cluster v in terms of the kernel function (the last term in (6.22) is constant).

Let $\mathbf{A}$ be a $h \times n$ matrix where individual matrix elements denote if observation $\mathbf{x}_i$ belongs to bag u ($\mathbf{A}_{ui} = 1$) or not ($\mathbf{A}_{ui} = 0$). Let further $\mathbf{\Pi}$ be a $h \times l$ matrix ($k = l$) containing the label counts (instead of proportions) of observations labeled as Y_v in bag u. Taking into account the label proportions, the authors propose to solve the following minimization problem:

$$\min_{\mathbf{T}} \quad \sum_{v=1}^{l}\sum_{i=1}^{n} \mathbf{T}_{vi}\mathbf{D}_{vi} + \lambda \ell(\mathbf{A}, \mathbf{\Pi}, \mathbf{T}) \tag{6.23}$$

$$\text{s.t.} \quad \mathbf{D}_{vi} = ||\phi(\mathbf{x}_i) - \mathbf{c}_v||^2 \tag{6.24}$$

$$\mathbf{T}_{vi} \in \{0, 1\} \quad \forall v, i \tag{6.25}$$

$$\sum_{v=1}^{l} \mathbf{T}_{vi} = 1 \quad \forall i = 1, \ldots, n, \tag{6.26}$$

where $\mathbf{D}$ is a $l \times n$ square distance matrix between observations and clusters. The first term in the objective function, $\mathbf{T}_{vi}\mathbf{D}_{vi}$, is the same as in the original objective function of Kernel k-Means, when written as a matrix factorization problem. However,

the second term $\ell(\mathbf{A}, \mathbf{\Pi}, \mathbf{T})$ is a loss function which measures the deviance between the label proportions that would result from the current assignment of observations to clusters (classes), and the given label proportions. While the authors have proposed two different loss functions, ℓ_1 and ℓ_2, here we present only the second one, which is a version of quadratic loss:

$$\ell_2(\mathbf{A}, \mathbf{\Pi}, \mathbf{T}) = \frac{1}{2}\text{tr}\left((\mathbf{A}\mathbf{T}^\top - \mathbf{\Pi})(\mathbf{A}\mathbf{T}^\top - \mathbf{\Pi})\right) . \tag{6.27}$$

In that way, the authors try to find a good clustering $\mathbf{T}$, i.e. assignment of observations to clusters (classes), such that the within cluster scatter (see Sect. 3.4) is minimized, but at the same time also the given label proportions are matched as good as possible. The trade-off between the two criterions can be controlled by parameter λ.

Although the two loss functions proposed in [CLQZ09] are convex for continuous $\mathbf{T}$, the elements of $\mathbf{T}$ are discrete, i.e. that standard tools for convex optimization cannot be used. The authors therefore propose to solve the problem with an EM-like alternating updating algorithm (see [DLR77]). First, $\mathbf{D}$ is calculated according to the current $\mathbf{T}$, and then a sub-optimization problem is solved to assign each observation to a cluster. These steps are repeated until convergence. The authors show that the sub-optimization problem for ℓ_1 can be solved efficiently with linear programming, and for ℓ_2 leads to a standard quadratic optimization problem.

The authors have compared their method with the k-NN and neural network variants for learning from label proportions (see basic methods [MCO07] above), on two datasets from the UCI standard repository. In both cases, AOC Kernel k-Means outperformed k-NN and neural networks in terms of accuracy. Moreover, AOC Kernel k-Means had lower run-time than the neural networks.

The presented approach, called AOC-KK in the following, shares similarities with the clustering approach (LLPC) developed in Sect. 6.5, but has some fundamental differences. The first is that AOC-KK restricts the number of clusters to the number of classes, while LLPC allows for classes being represented by more than one cluster. This allows for a better control of bias vs. variance (see Sect. 3.1.6), by changing k. Another difference is that AOC-KK combines the loss over label proportions with the original Kernel k-Means objective in the same objective function, while LLPC first clusters observations as usual, and then tries to find a good assignment of labels to the resulting clusters. LLPC thus has the advantage that it can be used with arbitrary partitional clustering algorithms, while the approach taken in [CLQZ09] works only with k-Means and Kernel K-Means. LLPC is compared to AOC-KK with loss ℓ_2 in Sect. 6.6.

Theoretical Results In an unpublished work [YKJC14], the task of learning from label proportions has recently be cast into the PAC learnability framework (see Sect. 3.1.3). The driving question is: Can we learn from label proportions at all and if we can, under which circumstances? In their work, the authors prove that under certain conditions, the labels of individual observations can be predicted well when the label proportions per bag (called bag proportions) can be predicted well. The generalization error over

bag proportions in turn can be bounded by the empirical proportion error if the number of bags is large in relation to the VC dimension of the underlying hypothesis class H.

For the discussion in this section, we adapt the original problem setting (see Sect. 6.1) slightly. Let us assume that bags $b = \{(\mathbf{x}_i, y_i)\}$ with labeled observations are generated i.i.d. according to an underlying distribution D. Let $S = \langle b_1, \ldots, b_h \rangle$ be a finite sample of bags, and $\pi(b)$ be a function which calculates the label proportions from a given bag b. We want to derive a function $\hat{f} \in H$, being element of hypothesis class H, which minimizes loss ℓ over individual observations, only based on the bags in S, with their labels removed, and the label proportions $\pi(b)$ calculated from them. Let $\hat{\pi}(b)$ be a function which calculates the label proportions from a bag b, not based on the given labels, but based on applying function $\hat{f}$ to each $x_i \in b$ instead, taking the predicted labels $\hat{f}(x_i)$. The result of $\hat{\pi}(b)$ may be seen as the *model-based label proportions* calculated from bag b. Let ℓ_π be a loss function which measures the deviation between the model-based label proportions $\hat{\pi}(b)$ and the given label proportions $\pi(b)$, as already defined in a similar way for the $\propto$SVM. The sample error err_S^π of $\hat{f}$ and the true error err_D^π of $\hat{f}$ over bags can then be defined as follows:

$$err_S^\pi(\hat{f}) := \frac{1}{h} \sum_{u=1}^{h} \ell_\pi(\hat{\pi}(b_u), \pi(b_u)) \,, \qquad err_D^\pi(\hat{f}) := P_{b \in D}\, \ell_\pi(\hat{\pi}(b), \pi(b)) \tag{6.28}$$

The sample error (or empirical error) measures how well we match the given label proportions of bags in S with hypothesis $\hat{f} \in H$, while the true error measures how well we generalize, i.e. how well we can *predict* the label proportions of previously unseen bags, drawn i.i.d. from D, based on the learned hypothesis $\hat{f} \in H$.

Similar to supervised learning, the question arises how well the empirical error reflects the true error. As Sect. 3.1.5 has shown, if sample size is large enough in relation to the capacity of the hypothesis class, the second term of the structural risk is small. Risk is then largely determined by the empirical error, which justifies the empirical risk minimization principle. Let H_π be a bag proportion hypothesis class associated with $\hat{\pi}(b)$, the function which predicts the label proportions of a given bag b. Based on the idea that the capacity of the bag proportion class H_π is dependent on the capacity of hypothesis class H for individual observations, the authors of [YKJC14] bound the covering number of H_π by the covering number of the VC dimension of H. The authors then prove the following theorem on the generalization error of learning bag proportions:

Theorem 1. *For any $0 < \delta < 1$, $0 < \varepsilon < 1$, $h \in H$, with probability at least $1 - \delta$, $err_D^\pi(\hat{f}) \leq err_S^\pi(\hat{f}) + \varepsilon$, if*

$$h \geq \frac{64}{\varepsilon^2}(2VC(H)\ln(12r/\varepsilon) + \ln(4/\delta)) \,, \tag{6.29}$$

where $VC(H)$ is the VC dimension of hypothesis class H for individual observations, h is the number of training bags, and r is the bag size.

It follows that the generalization error of learning bag proportions is bounded if there are enough bags for training in relation to the VC dimension of hypothesis class H. Sample complexity further depends logarithmically on the bag size r.

A bag is called $(1-\beta)$-pure if at least a fraction of $1-\beta$ of all observations in this bag have the same label. The authors then state and prove the following two lemmas:

Lemma 6.2.1 Let $\hat{f}$ be a hypothesis satisfying $P_{b\in D}(|\hat{\pi}(b)-\pi(b)| \leq \varepsilon) \geq 1-\delta$ for some $0 < \varepsilon, \delta < 1$. Assume that the probability that a bag is $(1-\beta)$-pure is at least $1-\rho$ for some $0 < \beta, \rho < 1$. Then for that bag, the probability that $\hat{f}$ classifies correctly at least a fraction $(1-2\beta-\varepsilon)$ of its instances is at least $(1-\delta-\rho)$.

Lemma 6.2.2 There exists a distribution $\mathcal{D}$ over all bags of size r and a learner $\hat{f}$ such that $\hat{\pi}(b) = \pi(b)$, each bag is $(1-\beta)$-pure, but $\hat{f}$ misclassifies a fraction 2β instances of each bag.

The first lemma implies that the probability for classifying instances in a bag correctly increases with purity of the bag. The authors generalize this result to the case where several bags are $(1-\beta)$-pure. The second lemma means that in extreme cases where all label proportions are equal (i.e. they are the least pure), it can happen that a hypothesis achieves zero bag proportion error, but nevertheless classifies all instances incorrectly.

In the rest of their work, the authors give further results on bounding the true error of predicting individual instances by making additional assumptions on the distribution of bags. For further details, see [YKJC14].

Other Works The authors of [HGInL13] apply a structural EM strategy to learn Bayesian network classifiers from label proportions. They compare their method to Mean Map and report lower error rate of their method for four of seven domains. However, significance of results is not reported. In [FZY+14], a generative classifier called DNLP is learned from label proportions by following a deep belief network approach. The authors compare their method to Mean Map and Invcal on several standard datasets from the UCI repository. In terms of prediction performance, they report no significant differences. However, the run-time of DNLP is much lower than that of Mean Map and Invcal. In [FT15], convolutional neural networks (CNN) are combined with probabilistic graphical models trained by an EM approach to learn from label proportions in the context of ice and open water classification from image data. Their algorithm shows good performance in the context of the mentioned application, but isn't evaluated on other domains.

6.3 Difficulty of the Problem

In the following, we discuss the difficulty of learning from label proportions from a more Bayesian perspective and relate it to different kinds of better known learning tasks, like supervised, semi-supervised and unsupervised learning.

For the supervised learning scenario, as we have already seen in Sect. 3.1.2, the optimal classifier that could be constructed if distribution $P(X, Y)$ was known is called the *optimal Bayes classifier*. To repeat, it is based on the idea that from a Bayesian perspective, a prediction model can be obtained from estimating the conditional class density $P(Y \mid X)$. Applying Bayes theorem, one recognizes that $P(Y \mid X)$ may also be estimated from other unknown densities—the class-conditional density $P(X \mid Y)$ and the class prior density $P(Y)$:

$$P(Y \mid X) = \frac{P(X \mid Y) \cdot P(Y)}{P(X)} \tag{6.30}$$

Here, $P(X)$ doesn't necessarily need to be known or estimated, since it can be calculated from $P(X \mid Y)$ and $P(Y)$. $P(Y)$ may be estimated directly from the data, if the number of data points is high enough. Moreover, if the joint distribution $P(X, Y)$ is known, as is assumed by the optimal Bayes classifier, all other quantities can be derived from it. For a given observation $x \in X$ to classify, the optimal Bayes classifier would predict the most probable class, which is also known as the MAP criterion. Here, optimal means that the Bayes classifier is the best classifier over all possible classifiers for the given data.

Best Case With respect to learning from label proportions, the class prior $P(Y_v)$ for label Y_v can be estimated as $\eta(\mathbf{\Pi}, Y_v)$, the proportion of Y_v over sample S. This is done, for instance, by the Mean Map method presented in Sect. 6.2. Finding a good estimate for $P(X \mid Y)$, however, is at least as difficult as in the supervised scenario and equates to it if

1. each bag B_u only contains observations from a single class (i.e. $\exists v : \pi_{uv} = 1$ and $\forall z \neq v : \pi_{uz} = 0$) and
2. at least l bags contain observations from different classes.

In other words, if $\pi_{uv} = 1$ appears in a row, all observations in the corresponding bag must have the same label, which thus can be assigned to all observations in this bag. If each bag only contains observations from a single class, this equates to all observations being labeled and thus the supervised learning scenario. We may then choose from many well-known supervised classifiers for learning. From the perspective of learning from label proportions, where usually less information about labels is given than in the supervised learning scenario, this may therefore be called the *best case*. Our intuition matches lemma 6.2.1, which has only been recently proven and states that the probability

of classifying instances correctly increases with the purity of the bags. When each bag only contains examples from the same class, each bag is as pure as possible.

If only some bags (instead of all bags) are pure, the scenario resembles a semi-supervised learning scenario: Some observations can be considered to be labeled, and the whole rest of observations can be considered being unlabeled. However, the situation doesn't equate the semi-supervised learning scenario exactly, since there is additional information given about the label proportions in bags of unlabeled observations. In the following Sect. 6.4, it will be shown how to take labeled observations into account, while at the same time respecting the given label proportions.

Worst Case In the *worst case*, all label proportions π_{uv} in matrix $\mathbf{\Pi}$ are equal, i.e. least pure, and labels can only be guessed correctly with probability $1/l$. If sample size is large enough, the worst case can only occur if also all class priors $P(Y_v)$ are equal, and the label does not depend on the bag, i.e. $P(Y|u) = P(Y)$. Otherwise, we can estimate $P(Y)$ from the data and at least predict the class that has highest probability to occur (i.e. the majority class). In this case, the probability for predicting the correct label would be higher than $1/l$.

In this context, lemma 6.2.2 points to an important difference between supervised learning and learning from label proportions. PAC learnability implies that the true error of a hypothesis is bounded by ε for all distributions D over X with probability at least $1 - \delta$ (see Sect. 3.1.3). The hypothesis can be worse if a randomly drawn sample doesn't represent the overall distribution D well. In the scenario of learning from label proportions, however, it may not only happen that a given sample leads to a bad hypothesis, but also the way in which observations are distributed over the bags. How likely is the worst case? Of course, it is hard to argue over all possible datasets and analysis tasks, which is unknown. It is worth to mention, however, that all results from [YKJC14] are based on probabilities, and not on relative frequencies. So even if the probabilities that an observation has a specific label are all equal over the bags, for a concrete sample we may expect the proportions in matrix $\mathbf{\Pi}$ to deviate slightly from each other, depending on the size of bags and how we sample. Slight deviations of the proportions may already help with making a correct decision about class labels. For instance, the well-known Iris dataset contains the same number of 50 observations for each of three different classes. With the clustering approach developed in Sect. 6.5, and sampling observations randomly uniformly into bags, in almost all cases a hypothesis can be found which classifies at least 96% of the observations correctly on average. It's only when we force all relative frequencies, i.e. proportions of observations in matrix $\mathbf{\Pi}$, to be equal or very close to each other, that prediction performance collapses to random guessing. Hence, the worst case is especially important in the context of privacy-preserving data mining, where we might have some control over the formation of bags and want to make the reconstruction of original labels as difficult as possible. Relative frequencies may further approach the true underlying probabilities with large sized bags, following the law of large numbers known from probability theory. In general, a small

number of large sized bags can make the problem more difficult, according to theorem 1.

Average Case The interesting question is what performance we can expect in cases where the label proportion matrix $\mathbf{\Pi}$ has a form which is different from best and worst case. The intuition is that the problem becomes more difficult the more similar the label proportions are over the bags. This idea follows the notion of entropy from information theory, which was already introduced in the context of decision trees (see Sect. 3.2.3): Bags which are more "pure", i.e. contain more instances of the same class, seem to provide more information. Lemma 6.2.1 shows that our intuition is correct, namely that the probability of classifying individual instances correctly increases with the purity of bags. It is important to realize, however, that lemma 6.2.1 only upper bounds the *probability* for misclassifying a fraction of individual observations incorrectly. Therefore, in practice, it is possible that cases occur where we perform well, despite label proportion matrix $\mathbf{\Pi}$ having high entropy, or badly, despite $\mathbf{\Pi}$ having low entropy. For instance, in [PNCR14], the information content of $\mathbf{\Pi}$ is measured by the criterion of Gini entropy. LMM and AMM sometimes performed well on domains where corresponding label proportion matrices had high entropy (i.e. low information content). How might this be explained?

While it is hard to tell the exact reason for a particular case, in general we note that not only the labels provide information for learning, but also the observations. Observations in bags with low information content in terms of labels may nevertheless provide information about the underlying distribution of observations, $P(X)$. As results from semi-supervised learning show [CSZ06], getting more information about $P(X)$ by taking also unlabeled observations into account can increase prediction performance when only a few labeled examples are given. So even in cases where the overall entropy of $\mathbf{\Pi}$ is high, we might arrive at a well-performing classifier if at least some of the bags provide sufficient information about labels. Conversely, even if a bag has high information content in terms of labels, learning might not profit from it. In terms of PAC learnability, we may say that the sample we got for learning doesn't represent the underlying distribution D well. For instance, the sample might only contain observations being located far away from the true decision border, providing not sufficient information for deriving a good hypothesis. Here, we can establish an interesting connection to learning from horizontally partitioned data (see Sect. 4.3 and Sect. 4.3.5): Subsamples of S, here the bags, can be skewed and may deviate much from the overall data distribution.

As the discussion shows, for practical cases, it is hard to find a measure of problem difficulty. While it is easy to measure the entropy of $\mathbf{\Pi}$, it is difficult to measure how well bags reflect the overall data distribution given a concrete sample S, without knowing the underlying data distribution—which is the crux of learning!

6.4 Loss and Risk

The biggest difference between supervised learning and learning from label proportions is that in the supervised setting, the kind of quantity we want to minimize is the loss over individual observations, while in learning from label proportions, all we can minimize directly is the loss between model-based and given proportions, although in truth we want to minimize the same quantity as in supervised learning. It is not exactly clear how the quantity that *can* be minimized relates to the quantity that *should* be minimized. Especially, many different label assignments may lead to the same label proportions and therefore to the same loss calculated on them.

There are different possible ways to define loss functions over label proportions. The first we define measures the quadratic deviation between the label proportions as being derived from a previously trained prediction model $\hat{f}(X)$ and the given label proportions. Applying the trained model to a set of observations $x_i \in S$, the resulting label proportions can be calculated by counting the number of observations x_i with $\hat{f}(x_i) = Y_v$, in each bag for each label $Y_v \in Y$ and dividing such counts by the size of their respective bag. This leads to a new matrix $\mathbf{\Gamma}_{\hat{f}}$, containing the model-based label proportions:

$$\mathbf{\Gamma}_{\hat{f}} = (\gamma_{uv}^{\hat{f}}), \quad \gamma_{uv}^{\hat{f}} = \frac{1}{|B_u|} \sum_{x \in B_u} I(\hat{f}(x), Y_v), \quad I = \begin{cases} 1 : \hat{f}(x) = Y_v \\ 0 : \hat{f}(x) \neq Y_v \end{cases} . \tag{6.31}$$

Similarly to defining a loss function for individual observations, it is now possible to define a loss function over individual matrix entries, for example by taking as loss the squared error $(\pi_{uv} - \gamma_{uv}^{\hat{f}})^2$. The total deviance between $\mathbf{\Pi}$ and $\mathbf{\Gamma}_{\hat{f}}$ can then be defined as the average squared error over all matrix entries:

$$\ell_{MSE}(\mathbf{\Pi}, \mathbf{\Gamma}_{\hat{f}}) = \frac{1}{hl} \sum_{u=1}^{h} \sum_{v=1}^{l} (\pi_{uv} - \gamma_{uv}^{\hat{f}})^2 \tag{6.32}$$

The average squared error ℓ_{MSE} doesn't take into account the relative group and class sizes, nor can it catch the situation where two hypotheses $\hat{f}_1$ and $\hat{f}_2$ appear indistinguishable from each other, because the total error sum over all matrix entries is the same. In practice, it can make sense to measure the error between $\mathbf{\Pi}$ and $\mathbf{\Gamma}_{\hat{f}}$ by $\ell_{\mathbf{\Pi}}$, which we define as the geometric mean of two different error measures ℓ_{weight} and ℓ_{prior} which deal with the previously mentioned disadvantages:

$$\ell_{\mathbf{\Pi}}(\mathbf{\Gamma}_{\hat{f}}) = \sqrt{\ell_{weight}(\mathbf{\Pi}, \mathbf{\Gamma}_{\hat{f}}) \cdot \ell_{prior}(\mathbf{\Pi}, \mathbf{\Gamma}_{\hat{f}})} \quad \text{with} \tag{6.33}$$

$$\ell_{weight}(\mathbf{\Pi}, \mathbf{\Gamma}_{\hat{f}}) = \frac{1}{hl} \sum_{u=1}^{h} \sum_{v=1}^{l} \eta(\mathbf{\Pi}, Y_v) \frac{|B_u|}{n} (\pi_{uv} - \gamma_{uv}^{\hat{f}})^2 \quad \text{and} \tag{6.34}$$

$$\ell_{prior}(\mathbf{\Pi}, \mathbf{\Gamma}_{\hat{f}}) = \frac{1}{l} \sum_{v=1}^{l} \Big(\eta(\mathbf{\Pi}, Y_v) - \eta(\mathbf{\Gamma}_{\hat{f}}, Y_v) \Big)^2 \tag{6.35}$$

ℓ_{weight} weights the squared error of individual matrix entries by their relative group and class size. ℓ_{prior} measures how well a chosen hypothesis matches the class priors, as estimated by $\eta(\mathbf{\Pi}, Y_v)$. The choice to include the prior in the loss function has been made based on empirical evaluations and a close examination of the label proportion matrices which have lead to misclassifications. What we have observed in our experiments has now also a theoretical justification. As recently shown in [YKJC14], whenever a hypothesis matches the class priors and observations in bags are distributed i.i.d., the probability to misclassify a fraction of individual observations is bounded.

Moreover, if in addition to the label proportions, the true labels $y(x)$ of a subset $T \subseteq S$ of observations $x \in T$ are given, error criterion (6.33) can be easily extended to include the average loss ℓ_T over these labeled training examples:

$$\ell_{\mathbf{\Pi}} = \sqrt[3]{\ell_{weight} \cdot \ell_{prior} \cdot \ell_T} \quad \text{with} \quad \ell_T = \frac{1}{|T|} \sum_{x \in T} \ell(y(x), \hat{f}(x)) \tag{6.36}$$

Algorithms which optimize over $\ell_{\mathbf{\Pi}}$ can thereby easily consider also labeled observations in addition to the given label proportions.

The idea of matching the given label proportions well has now also found support from theory. Theorem 1 and lemma 6.2.1 together give a justification for minimizing the deviation between the model-based and given label proportions in terms of empirical risk minimization: If hypothesis $\hat{f}$ matches the given bag proportions well, the risk of misclassifying a fraction of individual observations is bounded. As discussed in Sect. 3.1.5 and Sect. 6.2, empirical risk minimization only works in cases where the number of bags is large enough in relation to the VC dimension of the chosen hypothesis class. If we allow for arbitrarily complex hypotheses, we can always construct one which performs well on the given training set, but doesn't generalize well. In supervised learning, this comes close to memorizing the mapping between examples and labels by rote learning. Similarly, in learning from label proportions, we can always match the given label proportions, for instance by randomly sampling exactly μ_{uv} many observations from each bag B_u and assigning them label Y_v. It is unlikely, however, that a random labeling will be the correct one in terms of individual label assignments. Thus, even solutions which fit the given label proportions optimally can be bad solutions regarding the true error we want to minimize.

Especially in cases where the number of bags is small, there is a high risk to choose a hypothesis which doesn't generalize well. It is therefore important to control the capacity of our hypothesis class. Although the authors of [YKJC14] don't explicitly mention it, based on their work it should be possible to derive a VC bound for learning from label proportions, such that structural risk minimization can be applied. In cases where it is hard to derive the VC dimension of a hypothesis class for model selection and validation, the true bag proportion error might be estimated from a hold-out test set or by a black box approach like bag-wise cross-validation. Unfortunately, though an estimate of the true bag proportion error might help with model selection, it doesn't give us any more concrete information about the true error over individual observations other

than that the probability of misclassifying a fraction of them is bounded. In contrast, estimating the true error from a hold-out test set in supervised learning gives us a much more interpretable value. In the experimental Sect. 6.6, we have therefore used label proportions to learn and optimize hyperparameters, but test sets of labeled observations to evaluate the performance of algorithms in terms of prediction accuracy.

6.5 Learning from Label Proportions by Clustering

As discussed in the previous sections, the goal in learning from label proportions is to find a hypothesis $\hat{f} \in H$ which predicts the proportions of previously unseen bags as well as possible, which in turn bounds the risk of misclassifying individual observations, as shown in [YKJC14]. The authors pose this problem in terms of empirical risk minimization. However, if we allow for arbitrarily complex hypotheses, we can always match the given label proportions. Especially, if we tried all different possible labelings of observations exhaustively, we could always find a set of labelings which minimizes one of the previously introduced loss functions. We would expect only few of such labelings to also minimize the empirical loss over individual observations, i.e. we somehow need to control the capacity of our hypothesis class.

The approach for learning from label proportions proposed in the following works under the assumption that observations lying close together in regions of the input space also share the same class label. It first forms clusters of similar observations using an arbitrary partitional clustering algorithm and according distance measure (see Sect. 3.4). Instead of trying all possible labelings of observations, the algorithm heuristically tries different labelings of clusters, such that a loss function over label proportions is minimized. The capacity of the hypothesis space can be controlled by varying the number of clusters k. A small number of clusters leads to high bias, but low variance. A larger number of clusters allows for ever smaller divisions of sample S, and therefore leads to low bias, but high variance.

The assumption that clusters represent classes is not necessarily correct. As shown in [HTF09], especially the weighting of features can have an enormous influence on clustering results. In fact, one advantage of supervised methods over unsupervised ones is that they can determine the relevance of features in relation to the target variable. We therefore allow for a certain flexibility in distance measures. Such measures should respect weights $\mathbf{w}_j \in [0, 1]$ for each feature A_j, as given by a vector $\mathbf{w} = (w_1, \ldots, w_p)$. Usually, such weights are specified by a domain expert. In the clustering approach introduced in the following, however, the relevance weights can be approximated automatically by an evolutionary strategy, based on one of the loss functions defined in Sect. 6.4 (or other loss functions for learning from label proportions). In the next Sect. 6.5.1, the accompanying optimization problem is stated. Then, in Sect. 6.5.2, an approach for solving it is described. The algorithm can be used with different labeling strategies presented in Sect. 6.5.3. The approach's run-time is analysed in Sect. 6.5.4, while Sect. 6.5.5 explains how to classify new examples, based on a set of labeled clusters.

6.5.1 Optimization Problem

Let the vector $\boldsymbol{\lambda}_{\mathcal{C}} = (\lambda_1, \ldots, \lambda_k)$ with $\lambda_j \in Y$ represent a labeling for a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$. Let $\hat{f}_{\boldsymbol{\lambda}_{\mathcal{C}}} : X \to Y$ be a mapping that returns the label λ_i for a given observation $x \in C_i$. Given a clustering $\mathcal{C}$, we search for a labeling $\boldsymbol{\lambda}_{\mathcal{C}}$ of the clusters that minimizes the error, according to some error measure $\ell_{\boldsymbol{\lambda}_{\mathcal{C}}}(\boldsymbol{\Pi}, \boldsymbol{\Gamma}_{\hat{f}_{\boldsymbol{\lambda}_{\mathcal{C}}}})$, between the model-based label proportions $\boldsymbol{\Gamma}_{\hat{f}_{\boldsymbol{\lambda}_{\mathcal{C}}}}$ and the known label proportions $\boldsymbol{\Pi}$:

$$\min_{\boldsymbol{\lambda}_{\mathcal{C}}} \ell_{\boldsymbol{\lambda}_{\mathcal{C}}}(\boldsymbol{\Pi}, \boldsymbol{\Gamma}_{\hat{f}_{\boldsymbol{\lambda}_{\mathcal{C}}}}) \tag{6.37}$$

The error measure could be, for instance, the average squared error ℓ_{MSE} or a combined error measure like $\ell_{\boldsymbol{\Pi}}$.

Let $q_{\mathbf{w}}$ be a function which is able to assess the quality of a clustering based on a similarity measure that respects feature weights. We are trying to solve the optimization problem

$$\min_{\mathbf{w}} \ell_{\boldsymbol{\lambda}_{\mathcal{C}}}(\boldsymbol{\Pi}, \boldsymbol{\Gamma}_{\hat{f}_{\boldsymbol{\lambda}^*_{\mathcal{C}}}}), \quad \boldsymbol{\lambda}^*_{\mathcal{C}} = \operatorname*{argmin}_{\boldsymbol{\lambda}_{\mathcal{C}^*}} \ell_{\boldsymbol{\lambda}_{\mathcal{C}}}(\boldsymbol{\Pi}, \boldsymbol{\Gamma}_{\hat{f}_{\boldsymbol{\lambda}_{\mathcal{C}^*}}}), \quad \mathcal{C}^* = \operatorname*{argmax}_{\mathcal{C}} q_{\mathbf{w}}(\mathcal{C}), \tag{6.38}$$

i.e. we are searching for a clustering $\mathcal{C}^*$ which maximizes $q_{\mathbf{w}}$ and whose labeling $\boldsymbol{\lambda}^*_{\mathcal{C}}$ minimizes $\ell_{\boldsymbol{\lambda}_{\mathcal{C}}}$, for all possible weight vectors $\mathbf{w}$. As formulated, with arbitrary functions $q_{\mathbf{w}}$ and $\ell_{\boldsymbol{\lambda}_{\mathcal{C}}}$, the problem is non-convex. Since we want to allow for flexibility in the choice of such functions, in the following we approximate solutions by an evolutionary strategy.

6.5.2 The LLPC Algorithm

The LLPC (Learning from Label Proportions by Clustering) algorithm solves problem (6.38) by an evolutionary strategy. For each weight vector $\mathbf{w}$, the sub-optimization problem of maximizing $q_{\mathbf{w}}$ is solved by an inner clustering algorithm, where the particular $q_{\mathbf{w}}$ depends on the algorithm. The only prerequisite for the clusterer is that it returns disjunct clusters and respects different feature weights. The sub-optimization problem (6.37) is independent from the clusterer and currently can be solved by different labeling strategies, of which two are introduced in Sect. 6.5.3.

In more detail, LLPC (see Alg. 8) takes a clustering algorithm *clusterer*, a labeling algorithm *labeler* and an error measure $\ell_{\boldsymbol{\lambda}_{\mathcal{C}}}$ as parameters, in addition to $\boldsymbol{\Pi}$, S, $\mathcal{B} = \{B_1, \ldots, B_h\}$ and $Y = \{Y_1, \ldots, Y_l\}$, which are related to the task of learning from label proportions, and a set of parameters *evo* related to the evolutionary learning strategy. LLPC then approximates the optimal weight vector and returns $\mathbf{w}^*$, as well as the related clustering $\mathcal{C}^*$ and labels $\boldsymbol{\lambda}^*_{\mathcal{C}}$ for the clusters.

We use the evolutionary strategy described in [Mie08]. The evolutionary strategy starts with a random population P of normalized weight vectors, $\mathbf{w}_j \in [0, 1]$. For each individual in P, the clustering algorithm *clusterer* is called. The clusters are labeled according to the given labeling algorithm *labeler* and the fitness is evaluated by criterion $\ell_{\boldsymbol{\lambda}_{\mathcal{C}}}$. If the fitness is higher than the best fitness seen so far, the newly found clustering,

Algorithm 8 The LLPC algorithm

procedure LLPC($\mathbf{\Pi}$,S,$\mathcal{B}$,Y,*clusterer*,k, *labeler*,$\ell_{\boldsymbol{\lambda}_\mathcal{C}}$,*evo*)
  $best\_fit := -\infty$; $generation := 0$
  Randomly initialize a population P of $psize$ normalized weight vectors
  while $generation < maxgen$ **do**
    for $\mathbf{w} \in P$ **do**
      $\mathcal{C} := clusterer(\ S,\ k,\ \mathbf{w}\ )$
      $(\boldsymbol{\lambda}_\mathcal{C},\ err) := labeler(\ \mathcal{C},\ \mathcal{B},\ \mathbf{\Pi},\ Y,\ \ell_{\boldsymbol{\lambda}_\mathcal{C}}\ )$
      if $best\_fit < -err$ **then**
        $best\_fit := -err$; $\mathcal{C}^* := \mathcal{C}$; $\boldsymbol{\lambda}_\mathcal{C}{}^* := \boldsymbol{\lambda}_\mathcal{C}$; $\mathbf{w}^* := \mathbf{w}$
      end if
    end for
    $generation := generation + 1$
    if $generation < maxgen$ **then**
      $Pcopy := P$
      Gaussian mutation of weights in $Pcopy$ with variance $mutvar$
      $Pchildren :=$ Uniform crossover on $P \cup Pcopy$ with probability $crossprob$
      $P :=$ Tournament selection with size $tournsize$ on $P \cup Pcopy \cup Pchildren$
    end if
  end while
  return $\mathcal{C}^*$, $\boldsymbol{\lambda}_\mathcal{C}{}^*$, $\mathbf{w}^*$
end procedure

labeling and weight vector are memorized as the new best ones. In each generation, the weight values in a copy of P are mutated by a Gaussian distribution and, with a certain probability, exchanged with P by a crossover operator. The individuals then take part in a tournament and only the best ones are kept in the next generation. This process is repeated until the maximum number of generations as specified by the user is reached.

Using an evolutionary strategy as a wrapper has the advantage that it is not necessary to integrate the error measure $\ell_{\boldsymbol{\lambda}_\mathcal{C}}$ into the optimization problem of the inner clustering algorithm, like it was done in AOC-KK, for instance. The clustering algorithm can thus be treated as a black box and easily exchanged, without any further adaptation. It should also be noted again that in contrast to AOC-KK, LLPC allows for classes being represented by more than just one cluster ($k > l$). Thereby LLPC allows for ever smaller divisions of sample S, i.e. parameter k may be seen as a control parameter that trades off bias against variance, as previously discussed.

The free choice of clustering algorithm allows for respecting different kinds of data distributions. For example, LLPC was already run successfully with k-Means [Mac67], Kernel k-Means [DGK04], EM clustering [DLR77, WFH11], DBSCAN [EKSX96], PROCLUS [AWY+99] and Support Vector Clustering (SVC) [BHHSV02], without modification. Moreover, LLPC can be used with different error measures, for instance with criterion (6.36) that can respect individually labeled examples.

LLPC may therefore be looked at as a meta-algorithm for learning from label proportions, which allows for the use of different clustering algorithms, labeling strategies and loss functions. In a further step, one might also exchange the evolutionary strategy. For instance, it might be adapted to not only minimize $\ell_{\boldsymbol{\lambda}_\mathcal{C}}$ over weight vector $\mathbf{w}$, but also over hyperparameters like k in the case of k-Means clustering, or C and the RBF kernel γ in the case of SVC.

6.5.3 Labeling Strategies

The following two labeling algorithms solve the sub-optimization problem (6.37) and can be used as the *labeler* in LLPC (see Alg. 8).

Exhaustive Labeling As long as k can be restricted to a small number and $l = 2$ for a binary classification problem, trying l^k possible labelings for a clustering $\mathcal{C}$ is no problem. In experiments (see Sect. 6.6), good solutions often were found for $6 \leq k \leq 12$. For each labeling, we need to calculate $\text{err}_{\boldsymbol{\lambda}_\mathcal{C}}$. For error measures like ℓ_{MSE} or $\ell_{\boldsymbol{\Pi}}$, this takes linear time in the number of observations n. In case of the aforementioned error measures, the calculations only involve basic operations like count, addition, multiplication and division.

Local Search with Multistarts For cases where the number of clusters $k > 12$ or the number of labels $l > 2$, a local search that is started multiple times with different random combinations of labels is proposed (see Alg. 9). The local search greedily improves on the current labeling of clusters by trying all possible labels at each component of the labeling vector $\boldsymbol{\lambda}_\mathcal{C}$. Fitness measures how well the model-based label proportion matrix $\boldsymbol{\Gamma}_{\hat{f}}$, as calculated from the current labeling, matches the given label proportions $\boldsymbol{\Pi}$. If the fitness improves, the search starts from the first component of the labeling vector $\boldsymbol{\lambda}_\mathcal{C}$, again. Otherwise, it resets the label at the current position *kpos* to the label of the best (local) solution found so far. Returned is the best labeling found over all starts of the different greedy searches.

In each iteration, the greedy search runs until no further improvement is possible (which is a stopping criterion). Moreover, at each step of the algorithm, the fitness either improves or is staying the same. Therefore, each search finds a local minimum. Since the number of searches is finite, the returned labeling vector is also locally minimal. In comparison to the exhaustive labeling strategy, it cannot be guaranteed that a globally optimal solution is found. However, as will be shown in Chap. 7, in the context of a real-world application like traffic flow prediction, LLPC_{lsm} provides good performance, while having much lower run-time than the exhaustive search.

6.5.4 Run-time Analysis

The user-specified parameters *maxgen*, *psize* and *tournsize* in LLPC are constants. They do not depend on the number of observations n and limit the number of iterations of

Algorithm 9 Labeling of clusters by local search with multistarts

function LocalSearchMultiStart($\mathcal{C}, \mu, \mathbf{\Pi}, k, Y, \ell_{\boldsymbol{\lambda}_\mathcal{C}}, starts$)
  $best = -\infty$
  for $iteration \leftarrow 1, starts$ **do**
    $\boldsymbol{\lambda}_\mathcal{C}, {\boldsymbol{\lambda}_\mathcal{C}}^{\text{bestIter}} \leftarrow (\lambda_1, \ldots, \lambda_k)$ with $\lambda_j \in Y$ chosen uniformly at random
    $start, bestIter \leftarrow -\ell_{\boldsymbol{\lambda}_\mathcal{C}}(\mathbf{\Pi}, \mathbf{\Gamma}_{\hat{f}_{\boldsymbol{\lambda}_\mathcal{C}}})$ ▷ Calculate initial fitness, based on error measure
    $improving \leftarrow$ true
    while $improving$ **do**
      for $kpos \leftarrow 1, k$ **do** ▷ At each position ...
        for $lpos \leftarrow 1, |Y|$ **do** ▷ ... try all labels ...
          $\lambda_{kpos} \leftarrow Y_{lpos} \in Y$
          $fitness \leftarrow -\ell_{\boldsymbol{\lambda}_\mathcal{C}}(\mathbf{\Pi}, \mathbf{\Gamma}_{\hat{f}_{\boldsymbol{\lambda}_\mathcal{C}}})$ ▷ Calculate fitness, based on error measure
          if $fitness > bestIter$ **then**
            ${\boldsymbol{\lambda}_\mathcal{C}}^{\text{bestIter}} \leftarrow \boldsymbol{\lambda}_\mathcal{C}$; $start, bestIter \leftarrow fitness$
            break ▷ Leave both **for** loops
          else
            $\lambda_{kpos} \leftarrow \lambda_{kpos}^{\text{bestIter}}$ ▷ Reset to best label found at $kpos$ so far
          end if
        end for
      end for
      if $bestIter = start$ **then** ▷ Nothing better found
        $improving \leftarrow$ false
      end if
    end while
    if $bestIter > best$ **then**
      $best \leftarrow bestIter$; ${\boldsymbol{\lambda}_\mathcal{C}}^{\text{best}} \leftarrow {\boldsymbol{\lambda}_\mathcal{C}}^{\text{bestIter}}$ ▷ Remember best solution
    end if
  end for
  return ${\boldsymbol{\lambda}_\mathcal{C}}^{\text{best}}, -best$
end function

the evolutionary strategy to be constant. As discussed in Sect. 6.5.3, the asymptotic run-time of the labeling strategies is linear in n, as k and l are constants and the evaluation of $\text{err}_{\boldsymbol{\lambda}_\mathcal{C}}$ usually takes linear time. The asymptotic run-time of LLPC will otherwise depend on the used cluster algorithm. For example, if we allow for approximate solutions and limit the number of iteration steps, k-Means has linear run-time. Hence, overall LLPC has linear run-time. However, when used with an algorithm like Kernel k-Means, the run-time of LLPC can also become quadratic, for instance.

6.5.5 Generating a Prediction Model

The LLPC algorithm returns labeled clusters of sample S. It is then possible to assign labels to individual observations $x_i \in S$ with $\hat{f}_{\boldsymbol{\lambda}_\mathcal{C}}$. The question is how to predict the labels of new observations, i.e. how to transform a clustering into a prediction model.

In the case of clustering algorithms which return a model-based description of clus-

ters, like k-Means which returns cluster means, one can simply use the model to assign new observations to a cluster, and then predict the cluster's corresponding class label. For instance, in the case of k-Means, one can assign new observations to their closest cluster mean and predict the corresponding class label, by applying function $\hat{f}_{\boldsymbol{\lambda}_C}$. Whenever a clustering algorithm is purely descriptive, i.e. in cases where it only returns a clustering of S, but no model to assign new observations to clusters, one may use a nearest neighbors approach like k-NN for classification.

In general, one option for getting a prediction model after running LLPC is to train a standard classifier like Naïve Bayes [JL95] or a SVM [Vap99], based on the now labeled observations. Taking this approach, LLPC may be looked at as a preprocessing step before modeling, in which the missing labels of observations in sample S are restored, based on the given label proportions.

6.6 Evaluation

In this section, the LLPC algorithm is compared to three state-of-the-art methods for learning from label proportions: The Mean Map [QSCL09] method, Inverse Calibration (Invcal) [Rüp10] and AOC Kernel k-Means (AOC-KK) [CLQZ09].

LLPC is written in Java and has been implemented in the form of several operators in RapidMiner. All results are based on using Fast k-Means [Elk03] as inner clustering operator, which is a variant of k-Means utilizing the triangle inequality for faster distance calculations. As distance measure, we have used the weighted Euclidean distance

$$d_{\mathbf{w}}(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{p} (\mathbf{w}[j]\mathbf{x}[j] - \mathbf{w}[j]\mathbf{x}'[j])^2 \,. \tag{6.39}$$

In all experiments, we used the exhaustive labeling strategy (see Sect. 6.5.3) with loss function $\ell_{\boldsymbol{\Pi}}$ (see Sect. 6.4).

AOC-KK has been implemented using a combination of Java, RapidMiner and Matlab. For Mean Map and Invcal, R scripts were used which have been provided by the author of Invcal [Rüp10].

6.6.1 Prediction Performance Experiments

The prediction performance (accuracy) of LLPC, AOC-KK, Invcal and Mean Map has been assessed on the eight UCI [AN07] data sets shown in Table 6.1. Each possible value of a nominal feature has been mapped to a binary numerical feature with values 0 or 1. Numerical features were normalized to the $[0, 1]$ interval. Table 6.1 shows the number of features p after this preprocessing step.

In each single experiment, the accuracy has been assessed by a 10-fold cross-validation. For learning from label proportions, we have partitioned the training set of a particular fold into bags of size σ, by uniform sampling of observations. While such uniform sampling might not reflect the way in which bags are formed in a real-world setting, it

Dataset	n	p	Dataset	n	p
CREDITA	690	42	SONAR	208	60
VOTE	435	16	DIABETES	768	8
COLIC	368	60	BREAST CANCER	286	38
IONOSPHERE	351	34	HEARTC	303	22

Table 6.1: UCI data sets used for the experiments

allows for a more homogeneous interpretation of results across different datasets than domain-specific sampling based on feature values. We tried several bag sizes σ: 2, 4, 8, 16, 32, 64 and 128 (with the last bag smaller than σ, if necessary). The label proportions were calculated and the individual labels removed. In each fold, the accuracy of the learned prediction model has then been calculated on a labeled test set.

The kernel methods Mean Map, Invcal and AOC-KK have been tested with the linear kernel, polynomial kernels of degree 2 and 3 and radial basis kernels ($\gamma = 0.01$, 0.1 and 1.0). LLPC has been tested for cluster sizes $k \in [2, 12]$. As parameters for the evolutionary strategy, we used $maxgen = 10$, $psize = 25$, $mutvar = 1.0$, $crossprob = 0.3$ and $tournsize = 0.25$. Running LLPC with k-Means provides a prediction model consisting of cluster means with associated class labels. The same is true for AOC-KK. However, the cluster methods also assign labels to each observation in sample S, allowing for a subsequent training of other classifiers, as described in Sect. 6.5.5. Based on such labeled examples, we have trained models for Naïve Bayes [JL95], k-NN [Aha92], decision trees [Qui86], random forests [Bre01], and the SVM [Vap99] with linear and radial basis kernel. See Sect. 3.2 for a further explanation of these methods. The model parameters of each method have been optimized by a grid or evolutionary search.

The combination of all datasets, bag sizes, classifiers, their variants and parameters results in a total of 13.216 experiments: 672 for Mean Map and Invcal, 2.688 for AOC-KK and 9.856 for LLP. For bag sizes 16, 32, 64 and 128 on the datasets COLIC and SONAR, and for bag size 128 on CREDITA, we conducted additional experiments with LLP for $maxgen = 5$ and $psize = 100$. In some cases, we got a better prediction accuracy. All experiments took about three weeks. They were run in parallel on up to six machines with an AMD Dual-Core or Quad-Core Opteron 2220 processor and a maximum of 4 GB main memory.

6.6.2 Prediction Performance Results

Figure 6.4 contains plots of the highest achieved accuracies for all data sets and bag sizes, based on the best performing models of LLPC, AOC-KK, Invcal and Mean Map, over all conducted experiments. LLPC shows a higher accuracy than Invcal for many bag sizes on the data sets CREDITA, VOTE, COLIC, SONAR and BREAST CANCER. On CREDITA, VOTE, IONOSPHERE, SONAR and DIABETES, the variance of accuracy between bag sizes

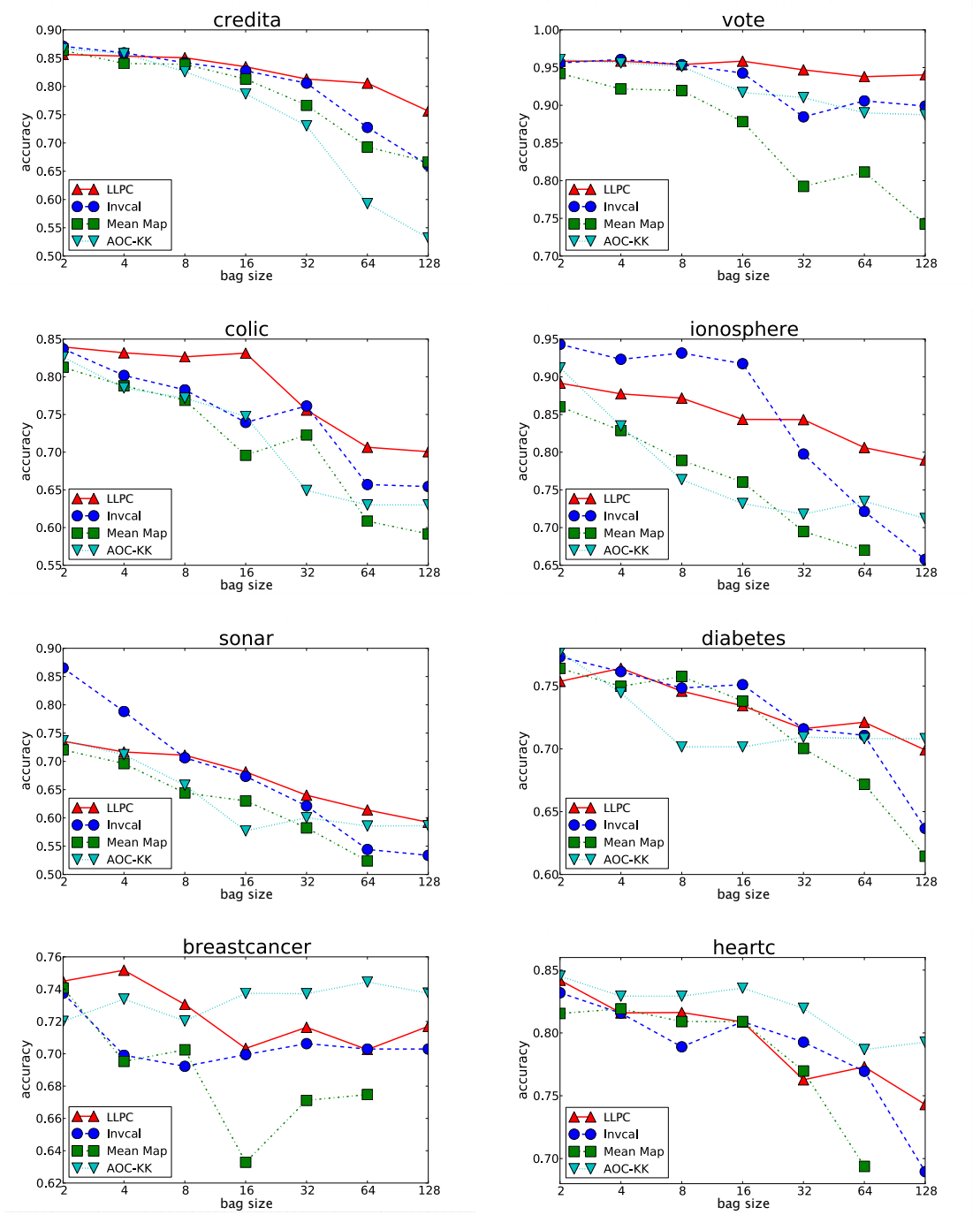


Figure 6.4: Highest accuracies for all data sets and bag sizes, over all 13.216 runs of LLPC, AOC-KK, Invcal and Mean Map (plus the additional runs of LLPC with $maxgen = 5$ and $psize = 100$). Some values for Mean Map and bag size 128 are missing in the plots, due to an error in the R script.

σ	2	4	8	16	32	64	128
						Average Ranks	
LLPC	2.500	**1.875**	**1.500**	**1.875**	**1.625**	**1.375**	**1.375**
AOC-KK	**2.000**	2.750	3.000	2.875	2.625	2.375	2.000
Invcal	**2.000**	**1.875**	2.375	2.125	2.125	2.275	2.625
Mean Map	3.500	3.500	3.125	3.125	3.625	3.875	-
		Differences, $CD_{<128}$=1.4317, CD_{128}=0.98					
AOC-KK	-0.500	0.875	**1.500**	1.000	1.000	1.000	0.625
Invcal	-0.500	0.000	0.875	0.250	0.500	1.000	**1.250**
Mean Map	1.000	**1.625**	**1.625**	1.250	**2.000**	**2.500**	-

Table 6.2: Average ranks of classifiers by bag size, and their difference to LLPC's rank, based on the best models for each data set and bag size. Positive difference values indicate a better performance of LLPC. Highest ranks and significant differences (higher than CD) at the 10%-level are marked in bold.

is smaller for LLPC in comparison to the other methods. Mean Map performs worse than LLPC and Invcal in many cases. The performance of AOC-KK varies, depending on the data set. It shows good performance on BREAST CANCER and HEARTC, but not on the others. Except for the BREAST CANCER and VOTE data set and a few other accuracy values, the overall accuracy of *all* methods decreases with a larger bag size. The results thus confirm the theory, as given by theorem 1: With larger sizes of bags, without increasing the size of sample S, learning becomes more difficult.

6.6.3 Statistical Significance

For the comparison of multiple classifiers over multiple data sets, Demsar [Dem06] proposes the Friedman test, which is a non-parametric equivalent of ANOVA. Used is the adjusted version, with a test statistic distributed according to the F-distribution (see [Dem06]). The Friedman test ranks the classifiers for each data set separately. Under the null-hypothesis, the average ranks of the classifiers should be equal. In case of a critical difference, the null-hypothesis can be rejected. According to the Friedman test, we have found significant differences for all group sizes. One can then proceed with a post-hoc test. We have decided for the two-tailed Bonferroni-Dunn test (again, see [Dem06]), which is for comparing a single classifier (here, LLPC) to all others.

Table 6.2 shows the average ranks of the compared classifiers and their difference to LLPC's rank. Each rank was calculated based on the best performing models (including the standard classifiers), over all conducted experiments. The table also shows the critical difference (CD) values for the Bonferroni-Dunn test. The CD for $\sigma = 128$ is different, because Mean Map was not included in the comparison, due to missing values. LLPC has the highest rank in six cases, for $\sigma > 2$. At the 10%-level, LLPC is significantly

σ	2	4	8	16	32	64	128
						Average Ranks	
LLPC	2.375	2.375	**2.000**	2.250	2.250	**1.750**	**1.375**
AOC-KK	3.750	3.250	3.125	2.875	2.875	2.375	2.125
Invcal	**2.125**	**1.625**	2.125	**1.750**	**1.625**	2.000	2.500
Mean Map	2.750	2.750	2.750	3.125	3.250	3.875	-
		Differences, $CD_{<128}$=1.4317, CD_{128}=0.98					
AOC-KK	1.375	0.875	1.125	0.625	0.625	0.625	0.750
Invcal	-1.000	-0.750	0.125	-0.500	-0.625	0.250	**1.125**
Mean Map	0.125	0.375	0.750	0.875	1.000	**2.125**	-

Table 6.3: Average ranks of classifiers by bag size, and their difference to LLPC's rank. Ranks are based on the best performing models of Invcal and Mean Map and the best performing cluster mean models of LLPC and AOC-KK. Positive difference values indicate a better performance of LLPC. Highest ranks and significant differences (higher than CD) at the 10%-level are marked in bold.

better than AOC-KK for $\sigma = 8$, better than Invcal for $\sigma = 128$ and better than Mean Map for $\sigma = 4, 8, 32$ and 64. In all other cases, LLPC performs equivalently.

The ranks in Tab. 6.3 are based on different models than those in Tab. 6.2. For LLPC and AOC-KK, included were only the best performing cluster mean models. They were compared to the best performing models of Invcal and Mean Map, i.e. to different kernels. The cluster mean models perform significantly better than Mean Map for $\sigma = 64$ and better than Invcal for $\sigma = 128$. In all other cases, they show an equivalent prediction performance, but are faster to train and apply.

Concerning the performance and significance of the other classifiers, which were trained based on the LLPC and AOC-KK cluster models, decision trees performed significantly better than Invcal for $\sigma = 128$, better than Mean Map for $\sigma = 64$ and better than AOC-KK for $\sigma = 4$ and $\sigma = 32$. Naïve Bayes, k-NN and random forest had a performance similar to the cluster mean models. The linear SVM and the SVM with radial basis kernels showed no significant differences to Invcal, Mean Map or AOC-KK.

6.6.4 Run-time Comparison

For an empirical run-time comparison of LLPC, Invcal, Mean Map and AOC-KK, we have generated random data for a two Gaussian mixture classification problem (10.000 observations and 10 features, with values normalized to $[0, 1]$). Then, the average run-time for training and the accuracy of the classifiers over 10 folds of a cross-validation has been assessed for different samples of the data, with varying sizes (see Fig. 6.5). The bag size for learning from label proportions has been $\sigma = 16$ for all runs. A radial basis kernel with $\gamma = 0.1$ has been used for the kernel methods. LLPC has been run with the

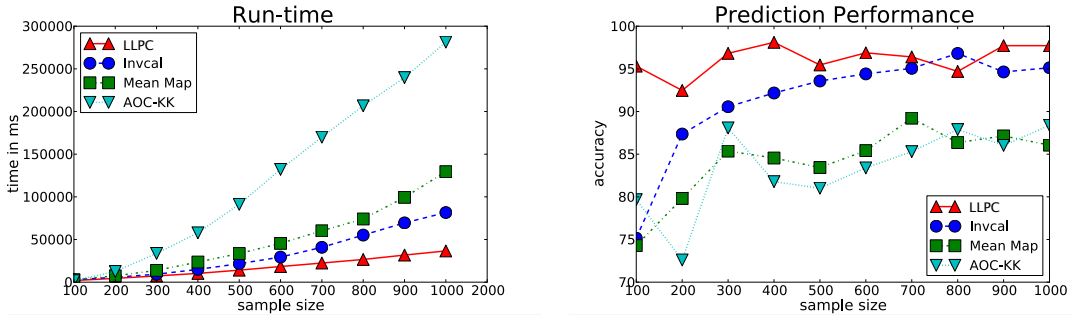


Figure 6.5: Average run-time and accuracy of 10-fold cross-validations with LLPC, Invcal, Mean Map and AOC-KK on several samples of random data. The data was generated for a two Gaussian mixture classification problem ($n = 10000$, $p = 10$, feature values normalized to $[0, 1]$).

exhaustive labeling strategy and Fast k-Means ($k = 6$), with parameters $maxgen = 3$, $psize = 25$, $mutvar = 1.0$, $crossprob = 0.3$ and $tournsize = 0.25$ for the evolutionary optimization. Both LLPC and AOC-KK used the cluster mean model for prediction.

LLPC shows a high prediction performance for all sample sizes. Moreover, LLPC has the lowest run-time. However, since the methods are implemented in different programming languages (Java, Matlab, R), one should not compare the absolute times, but the slope of the curves. The curve of LLPC's run-time is very flat and almost a straight line, while the slopes of the other curves indicate run-times that are faster growing.

6.7 Summary, Conclusions and Outook

Learning from label proportions has relevance for real world applications, like guaranteeing the privacy of democratic free elections, or the reconstruction of labels for objects that are hard to track, like in smart manufacturing.

Presented has been a novel approach for learning from label proportions, the Learning from Label Proportions by Clustering (LLPC) algorithm. The approach is general enough to accommodate for the use of different clustering algorithms, labeling strategies and loss functions. With k-Means as the inner clustering algorithm and a constant number of iterations, LLPC has only linear worst-case training time and its cluster mean models are small and fast to apply. This makes LLPC especially well-suited for running on resource-constrained devices, as they occur in many of the aforementioned IoT applications (see Sect. 2.1). In comparison to state-of-the-art methods, which need more training time, the cluster mean models show a significantly higher or equivalent prediction accuracy in the conducted experiments. By training other classifiers on the labeled clusters, the highest achieved accuracy of LLPC is significantly higher for even more bag sizes. Here, LLPC has the highest average rank for all $\sigma > 2$. In addition, LLPC has other beneficial properties that, to the best of our knowledge, other approaches don't

possess all at once: (1) LLPC can handle non-linear decision boundaries, depending on choice of clustering algorithm, (2) multiple classes, (3) additionally given labeled observations, and (4) it can weight the relevance of features.

Beyond the tracking of objects in smart manufacturing, learning from label proportions can also be related to distributed learning in an IoT context. So far, the missing of labels for individual observations has been looked at as a problem. However, as the empirical results presented demonstrate, in several cases accuracy is relatively high and stable even for growing sizes of bags. The question arises if approaches for the learning from label proportions can be combined with distributed approaches, such that only aggregated label information needs to be communicated between nodes. In the following Chap. 7, a fully decentralized algorithm for learning from vertically partitioned data is introduced that only exchanges label counts between nodes. In the context of traffic flow prediction, the algorithm reduces communication costs, while maintaining an acceptable high accuracy.

Chapter 7

Decentralized Training of Local Models from Label Counts

Traffic flow prediction allows intelligent control of traffic lights and other traffic signals. Individual mobility benefits from predictions as well, since they allow for proactive, smart decisions on individual travel plans like the avoidance of likely traffic hazards [NZC+15, LPBM14]. In an IoT context, more and more cities are getting equipped with smart sensors (see also Sect. 2.1.4) which, for instance, measure traffic flow, but also provide other kinds of valuable information about traffic, like parking lot utilization.

The task of traffic flow prediction focused on in this chapter can be stated informally as follows: Given the current traffic flow measurements and windows of previous measurements from all sensors at time point t, predict the traffic flow measurements at all sensors at some future time point $t + r$, where r is the prediction horizon.

Centralization of all data and control can easily account for global traffic patterns and relationships. However, a single point of failure poses high security risks facing natural disasters or intended devastation. The server-side collection further may become a bottleneck for real-time processing and is thus not scalable. The maintenance of cable networks is costly regarding materials and construction work. Moreover, the area of traffic management systems is often limited by the political area of homogeneous network regulations. In contrast, cheap battery-powered wireless presence sensors that work mostly autonomously could be easily attached to existing infrastructure like traffic lights, signs or buildings, increasing coverage. Restricting the exchange of measurements to local neighborhoods of topologically close sensors would lead to highly decentralized systems. Decentralization could make traffic control more robust facing disasters, as failures affect only locally confined parts of the whole system. For instance, traffic lights could become more autonomous, basing their operation on traffic flow predictions received from nearby sensors. Furthermore, costly construction work could be spared. Also, restricting communication to local neighborhoods might increase response time and can avoid the bandwidth bottlenecks of centralized systems.

Decentralized wireless networks consisting of small devices pose their own challenges

by putting severe constraints on data analysis tasks, as already discussed in Sect. 2.3.2. Challenges involve questions of streaming data, dynamic network changes, concept drift, and communication-efficient analysis by distributed components. Interpreting the sensor measurements (or windows of measurements) over all sensors as a single observed state from which we'd like to predict future measurements, one may say that the measurements are vertically partitioned over the local sensor nodes. As we have seen in Sect. 4.5, communication-efficient distributed learning in the vertically partitioned data scenario is particularly challenging, especially if we want to take conditional dependencies between features, given the label, into account.

In the following, a decentralized spatiotemporal in-network learning algorithm for the vertically partitioned data scenario is proposed. For the training of local models, it exchanges only space-time aggregated label counts between topologically close sensors. Thereby it reduces communication costs almost by an order of magnitude in comparison to a fully centralized analysis.

Section 7.1 gives a short overview of the state-of-the-art in traffic flow prediction. Then, Sect. 7.2 defines the learning task focused on in this chapter more formally. Section 7.3 discusses the advantages and disadvantages of an analysis of traffic flow measurements in local neighborhoods of sensors in comparison to a centralized, global analysis. Next, Sect. 7.4 proposes to exchange only aggregates of labels between nodes, reducing communication costs. The final distributed algorithm for the decentralized training of local models from label counts is introduced in Sect. 7.5, together with an analysis of its communication costs. The algorithm is evaluated on traffic flow data from the city of Dublin in Sect. 7.6. Finally, the chapter is summarized and conclusions are drawn in Sect. 7.7, further giving an outlook on the following chapter.

7.1 Related Work

In the field of traffic flow prediction, most literature describes processes on central servers. There are two major ways to model traffic: using a simulation [RN06] or applying an imputation model, trained on previous sensor measurements. Models are required for the estimation of traffic flow at locations not being observed at all. Such imputation is not the focus of our study, but the prediction of traffic flow at sensor locations. We point the interested reader to methods of simulation (e.g. cellular automaton [RN06]) and model-based imputation (e.g. [LXMW12]). Most learning-based traffic flow prediction methods analyze time series, where a popular model is based on auto-regressive integrated moving average (ARIMA) [AC79].

In [LTT06], the traffic flow prediction of k-NN for a particular location is improved by weighting measurements by their temporal distance to the prediction time. The Kriging approach described in [WK09] also aims at taking spatial relations into account. Improvements to k-NN non parametric regression are made in [MHK+08]. Their proposed algorithm is a spatial k-nearest neighbor approach that incorporates geometric distances for estimation of an unknown segment: the closer a measured segment is to an unmea-

sured one, the higher its impact. We utilize this idea to build local models based on their closest sensors. More complex approaches investigate neural networks or support vector machines for estimating a regression function for traffic forecast. Recently, an application of a Gaussian Markov Model was proposed in [SLMM14], and more advanced graphical models, namely Spatio-Temporal-Random-Fields (STRFs), were applied to traffic modeling in [PLM13]. We use the latter as a baseline method for evaluation (see Sect. 7.6).

The few existing distributed approaches for traffic flow prediction combine sketches of neighboring sensors to get probabilistic estimates of the number of vehicles co-occurring at different locations. Instead of counting and re-identifying individual vehicles, the approach introduced in the following only exchanges aggregated quantities between nodes.

7.2 Problem Setting

Here, the problem setting looked at is defined more formally. Given are m sensor nodes $j = 1, \ldots, m$. It is assumed that each node j delivers an infinite series of real-valued flow measurements $\ldots, v^{t-1}(j), v^t(j), v^{t+1}(j), \ldots$, where t denotes the current time step t and $t - 1/t + 1$ denote next and previous ones, assuming a constant sample rate. Associated with each sensor is a spatial location. Labels are assumed to be discretized flow measurements, since decisions on traffic signals often base on discrete flow categories, achieved by a mapping $\rho : \mathbb{R} \to Y$ of raw measurements to categories $Y = \{Y_1, \ldots, Y_l\}$.

Let each node j provide a dataset $D(j)$ for supervised offline learning, containing n pairs $(x_i(j), y_i(j))$ of training examples $x_i(j) \in \mathbb{R}^p$ and labels $y_i(j) \in Y$. Each $x_i(j)$ is created by sliding a window of size p with step size one over the stream of measurements at node j. When recording from time step s, observations $x_i(j) = [v^{s+i-1}(j), \ldots, v^{s+i-2+p}(j)]$ are windows of measurements, and labels $y_i(j)$ are discretized measurements $\rho(v^{s+i-2+p+r}(j))$ lying r time steps ahead.

For each node j, we want to predict the traffic flow category $y(j)$ at node j with horizon r correctly, given the current measurement windows $x(1), \ldots, x(m)$. For learning a corresponding model $\hat{f}$, available are all datasets $D(1), \ldots, D(m)$. Interpreting measurement windows as features of single observations x_i, the data is *vertically partitioned*, since each node only stores partial information about x_i, i.e. a subset of its features.

7.3 Global vs. Local Analysis

In a fully centralized data analysis setting, we would transfer all datasets to a central node or data center. There, we could then learn a *global model* $\hat{f}$ over all sensor locations and their measurements. For instance, probabilistic graphical models like the Spatio-Temporal-Random-Fields (STRFs) introduced in [PLM13] explicitly model the assumed dependency structure between measurements at different locations and over different time points. Thereby, they may also take into account conditional dependencies of traffic flow measurements over different locations. For instance, such models may capture the

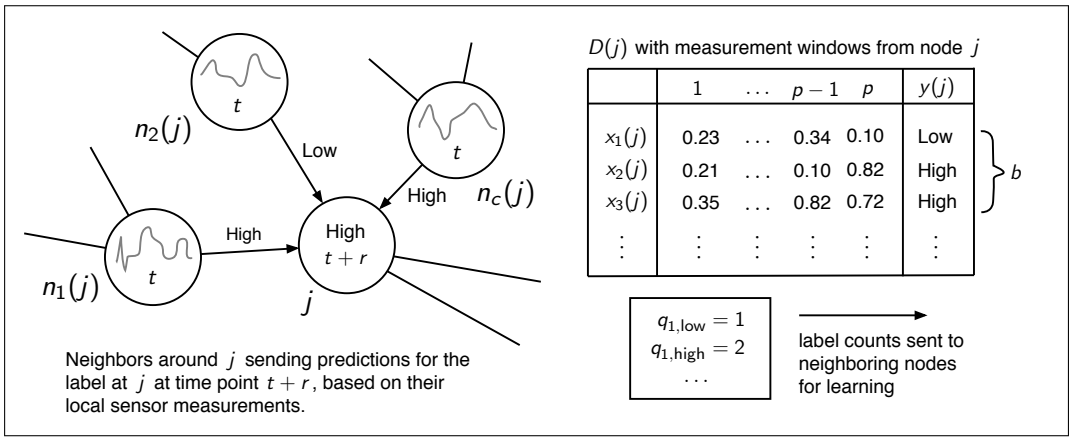


Figure 7.1: Distributed local models in a restricted neighborhood

situation in which traffic flow in the inner city depends on a combination of traffic flow on several highways leading to that city, and the flow on such highways influences each other. While the ability to model such dependencies is an advantage of global models, it comes at the cost of needing to transfer all data to a central node or data center, with all the disadvantages mentioned at the beginning of this chapter. The question is if we really need to consider the traffic flow at all sensor locations at once.

In the case of traffic flow, it is important to observe that the flow at a particular location doesn't change *immediately* with changes of traffic flow at other locations, especially those far away. It usually takes some time for changes to propagate through the street network, since each car has a physical speed limit. Further, the flow of traffic obeys additional rules. For instance, if we consider a particular junction, cars which reach that junction certainly must have crossed one of the other junctions directly connected to it. Hence, the question arises if taking into account dependencies between traffic flow of far away locations is truly necessary, since they seem to have only marginal influence. If it was possible to infer the traffic flow at node j only based on the traffic flow at connected neighboring junctions, we could restrict learning to local neighborhoods of topologically close sensors. As mentioned in Sect. 4.2 on clustering sensor nodes, the power needed to transmit data wirelessly increases with distance, and is d^α for distance d. Therefore, restricting wireless communication to topologically close sensors would be especially beneficial for the battery-powered sensors we assume.

Based on the aforementioned considerations, for every node j, we restrict learning to j and c neighboring nodes with indices $n_1(j), \ldots, n_c(j)$. Given datasets at j and its neighbors only, i.e. $D(j), D(n_1(j)), \ldots, D(n_c(j))$, we now want to learn a *local* model $\hat{f}(j)$ for node j that, given current windows $x(j), x(n_1(j)), \ldots, x(n_c(j))$ of sensor readings at node j and its neighboring nodes, predicts the traffic flow category $y(j)$ at node j with horizon r correctly. This situation is depicted in Fig. 7.1, showing node j with its neighbors and an exemplary dataset.

As discussed in Sect. 4.5 on the challenges of learning from vertically partitioned data, the architectural design of distributed algorithms in the scenario heavily influences their inferential abilities and communication costs. For learning that is restricted to local neighborhoods, we consider two different design choices.

Transmission of Measurements The first option is to transfer all datasets from neighbors to j, join them to a single dataset, and combine the resulting collection of observations with j's labels. Based on this data, we can then learn $\hat{f}(j)$ at j. By restricting communication and learning to local neighborhoods, we already have achieved some of our goals. The system has become more robust to failures, and avoids the bottleneck problems of transmitting all data to a central node. However, transmission of all sensor measurements still has high communication costs in the order of $O(np)$, during training as well as in the prediction phase. That is, for prediction, we would continuously need to exchange all raw measurements between nodes. Considering that we assume local sensor nodes to be battery-powered, this would be highly energy-draining.

Transmission of Labels As second option, we consider sending labels $y_i(j)$ from j to each of its local neighbors. We may then learn local models $\hat{f}_j(q)$ at nodes $k = j, n_1(j), \ldots, n_c(j)$, i.e. at node j and its neighbors, based on datasets $D_j(q) = \{(x_i(q), y_i(j))\}_{i=1,\ldots,n}$. Model $\hat{f}(j)$ could then, for instance, be a majority vote over predictions at j and predictions received from its neighbors. From the perspective of j's neighboring nodes, one may also say that each neighbor learns a model for the traffic flow at node j, based on the discretized sensor measurements $y_i(j)$ it receives from node j, and its own measurement windows. Communication costs during training and model application are in the order of $O(n)$, i.e. they have become independent of the number of features p, which is here the size of measurement windows. In practice, we would expect much lower communication costs in the prediction phase. The reason is that only information about the traffic flow category is sent, which can be expected to change less often than raw flow measurements. Much communication might be saved by restricting transmission to changes of category.

While taking the second design option looks more beneficial, what do we lose? The answer is that we lose inferential power. The first design allows us to respect conditional dependencies between traffic flow measurements at local neighboring nodes, given the label $y(j)$, because we have transmitted all feature values from neighboring nodes to j. In contrast, the second design option corresponds to the learning of independent local models, without exchanging feature values, but only combining predictions from each local model. Here, as discussed in Sect. 4.5.1, we have to make a conditional independence assumption on the traffic flow values from different neighboring nodes, given the label. In other words, we would need to assume that neighbors of j contribute independently to the traffic flow at node j. In a densely connected street network, this assumption is unlikely to hold. For instance, imagine a sensor measuring traffic flow at some junction j and sensors at two connected neighboring junctions $n_1(j)$ and $n_2(j)$,

with traffic flowing in both directions. Since label $y(j)$ itself denotes traffic flow, in the following, let's only focus on dependencies between measurements. Knowing the traffic flow values $x(n_1(j))$ and $x(n_2(j))$ at j's neighbors, we want to derive probabilities for the traffic flow at node j itself. If the only way to reach $n_1(j)$ from $n_2(j)$ is over j, we can estimate the conditional probabilities $P(x(j) \,|\, x(n_1(j)))$ and $P(x(j) \,|\, x(n_2(j)))$ independently from each other and multiply the result at j, since the traffic flow from both neighboring junctions should contribute independently to the flow at j. However, if $n_1(j)$ and $n_2(j)$ are connected somehow, we can assume that traffic at both junctions influences each other. Therefore, we would need to combine (i.e. communicate) traffic flow measurements from both junctions, and estimate $P(x(j) \,|\, x(n_1(j)), x(n_2(j)))$.

However, since our goal is the design of communication-efficient algorithms, for the sake of lower communication costs, we may ignore aforementioned dependencies and sacrifice accuracy. Therefore, from now on, we follow the second design approach. Unfortunately, communication costs during training are still in the order of $O(n)$. This points to an unsolved problem of training local models in the vertically partitioned data scenario, namely that labels are not available at local nodes, but first need to be exchanged between local nodes, or to be transferred from a central node. In the following, an idea is presented how to exchange only aggregated label information, namely counts of labels, between nodes, thereby saving communication. We can then transform such counts into proportions and choose from the many different methods for learning from label proportions presented in Chap. 6.

7.4 Communication of Label Counts

Given a partitioning of observations $x_1, \ldots, x_n$ into batches B_u, $u = 1, \ldots, h$ and label proportions π_{uv} for each batch u and class Y_v, algorithms for *learning from label proportions* (see Chap. 6) learn model $\hat{f} : X \to Y$ that assigns labels to individual observations. Instead of sending each individual label from j to its neighbors, communication may thus be saved by only sending the counts of labels per batch.

A simple partitioning of dataset $D(j)$ at each node j into b-sized batches is a division over consecutive time intervals. Node j counts how often each class occurs in each batch and sends these counts in a $h \times l$ matrix $\mathbf{Q}(j)$, $h = \lceil n/b \rceil$, to its neighboring nodes. These transform $\mathbf{Q}(j)$ into a label proportion matrix $\mathbf{\Pi}(j)$, yielding the original problem of learning from label proportions at neighboring nodes of j.

In principle, arbitrary algorithms that learn from label proportions (see Chap. 6) could be used as classifiers at each node $j = 1, \ldots, m$. The clustering approach developed in Sect. 6.5 seems to be especially well-suited for learning on small devices. As demonstrated, it has much lower run-time than other approaches, like Mean Map or Invcal. Moreover, it has a very small memory footprint and can easily handle multiple classes, like the flow categories in traffic flow prediction. For traffic flow prediction, we abstain from the evolutionary optimization of attribute weights as described in Sect. 6.5.2. Also, with $l > 2$, an exhaustive search over all labelings would consume too much time. In our

Algorithm 10 Distributed Training of Local Models from Label Counts

```
1: procedure TLMC
2:     for j ← 1, m do                                   ▷ in parallel over nodes
3:         divide D(j) into batches B_1, ..., B_h
4:         calculate label counts for each batch and store them in Q(j)
5:         send Q(j) to nodes n_1(j), ..., n_c(j)
6:         for q ∈ {j, n_1(j), ..., n_c(j)} do           ▷ in parallel over nodes
7:             calculate Π(j) from Q(j)
8:             train LLPC_lsm model f̂_j(q) at node q, on D(q)
9:         end for
10:    end for
11: end procedure
```

experiments presented in Sect. 7.6, we therefore rely on the local search with a multi-start strategy (see Alg. 9). The modified version of LLPC, i.e. without the evolutionary optimization of attribute weights, using a local search strategy, will be called LLPC_{lsm} in the following.

7.5 Distributed Training of Local Models from Label Counts

For the whole distributed learning algorithm, called Training of Local Models from Counts (TLMC), see Alg. 10. The algorithm starts after local preprocessing (windowing) at each node j. At the end, each node stores $c + 1$ different models, for itself and each of its neighbors. All models are *local* in the sense that learning is restricted to local neighborhoods and windows of local raw measurements. Moreover, the algorithm works fully *in-network*, as no central coordinator is needed for local synchronization and learning between peer nodes. The algorithm is communication-efficient, since it exchanges less data between nodes than sending all data to a central node or data center, as argued for in the next subsection.

It is important to note that although TLMC has been motivated by the application of traffic flow prediction, the algorithm and underlying principles are not restricted to it. In fact, TLMC is suited for all distributed classification tasks on vertically partitioned data in which we may assume conditional independence of features, given the label, or are ready to sacrifice accuracy in exchange for lower communication costs. Neighborhoods of nodes can be defined depending on application and network topology.

Analysis of Communication Costs Each node j transmits a matrix $\mathbf{Q}$ to each of its neighboring nodes, consisting of counts for each label $Y_v \in Y$ and batch. Such counts may be assumed to be integers. The maximum value of each integer is b, which means we need to reserve at most $\lceil \log_2 b \rceil$ bits to encode the count for each label. The number of batches, given n observations, is $\lceil n/b \rceil$. The total number of bits z_{cnt} which are needed

to encode matrix $\mathbf{Q}$ is therefore

$$z_{\text{cnt}} = \left\lceil \frac{n}{b} \right\rceil |Y| \lceil \log_2 b \rceil \quad . \tag{7.1}$$

Given m nodes, the total payload transmitted in bits is thus $O(mz_{\text{cnt}})$, if we assume that label counts are broadcast to neighboring nodes. For topologically close wireless sensors, this assumption is not unrealistic. All payloads reported in Sect. 7.6 base on this assumption.

In comparison, the total payload transmitted by sending all measurements to a central node or data center is $mpn \cdot 64$ or $mpn \cdot 32$, depending on the precision of floating point numbers we choose (double or single precision). In many learning settings, it can be assumed that $p \geq |Y|$. Moreover, the number of bits needed to encode the label counts, which are integer values, can be assumed to be smaller than the number of bits needed to encode floating point numbers. This means we have $n > \lceil n/b \rceil$, $p \geq |Y|$ and $32 > \lceil \log_2 b \rceil$, and therefore $np \cdot 32 > \lceil n/b \rceil |Y| \lceil \log_2 b \rceil$.

Example 7.5.1 For $m = 100$, $n = 4096$, $b = 256$ and $p = |Y| = 4$, when sending all measurements with single precision to a central node, the transmitted payload would be about 52 MB. In comparison, when sending label counts, one order of magnitude less data would need to be transmitted, namely only about 13 KB. Even taking into account that windows can be created at the central node, setting $p = 1$, we would still transmit one order of magnitude more data when sending all data to a central node, namely about 13 MB instead of 13 KB.

Example 7.5.2 Local models may save much communication also in the prediction phase, where we cannot send label counts, but must send individual labels instead. As discussed in Sect. 7.3, communication can be especially saved in cases where the number of features p is high. Consider the case where instead of presence sensors, we are dealing with slightly more powerful wireless devices with cameras attached. The cameras are capturing $n = 15$ images per second, each image consisting of 800x600 grey values, i.e. $p = 480,000$. Let's assume we want to fuse the data from $m = 10$ cameras for binary event detection ($|Y| = 2$). Monitoring, based on a central global model, might require the continuous transmission of all image data, leading to a data transmission rate of 72 MB/s. In comparison, since label information can be encoded in only one bit per image, 20 Byte/s would need to be transmitted when fusing the predictions of local models at some central node. This is 3.6 million times less data to be sent. Looking into Tab. 2.1 in Sect. 2.3.2 shows that the transmission of all image data would require advanced wireless network technologies, while the transmission of binary event labels could be easily realized even over a slow EDGE connection. Even when sending vectors of extracted features to a central node, where for instance $p = 16$, the transmission of labels would still be more communication-efficient.

Adaptation to Other Settings As mentioned, TLMC and its underlying principle are not restricted to the application of traffic flow prediction. In fact, Alg. 10 may be easily adapted to different settings and network topologies in the vertically partitioned data scenario. For instance, one special case would be having only a single "local" neighborhood, comparable to the setup shown in Fig. 2.7 in Sect. 2.5.2. Node $j = 0$ would play the role of a central coordinator storing labels of observations, but no data about the observations themselves. All neighboring nodes may play the role of local nodes $j = 1, \ldots, m$, storing the vertically partitioned features of observations. Instead of looping over all nodes, for training, we focus on the inner loop of the algorithm. The central coordinator would inform all local nodes about batch size b and send according label counts. Each local node could then transform the counts into label proportions and learn an according model, restricted to its own local subset of features. In the prediction phase, local nodes might then send their local predictions to the central coordinator, which combines them by an according fusion rule, like a majority vote, for making the final prediction.

It should also be noted that the algorithm might be adapted to settings where nodes enter and leave the network dynamically. In comparison to a global model, which might need to be completely retrained, here only local models of nearby sensors would need to be retrained. Although dynamically changing neighborhoods could pose technical challenges, they are no problem from the view point of data analysis.

7.6 Experiments

TLMC is evaluated on data of the city of Dublin. The Sydney Coordinated Adaptive Traffic System (SCATS) provides information on vehicular traffic at over 750 fixed sensor locations as spatiotemporal time series [McC14]. Sensors are attached to junctions over the city of Dublin. The used data[1] is a snapshot from January 2013, consisting of tuples encoding the location of the observation, an index for the junction, the arm and the lane number at which the sensor is located at, as well as the aggregated vehicle count at sensor location since last measurement. A time stamp denotes the recording time.

For simplicity, data is getting aggregated at junction level. This step makes prediction harder, since information on movement directions is getting lost. The measured traffic flow at all arms of a junction is summarized by their mean value, resulting in 339 sensor locations. To be independent of traffic light affected fluctuations, traffic flow is aggregated over 15 minute intervals, resulting in 2,976 time slices. After filtering out sensor nodes with only zero or missing values, the final dataset of 296 sensor locations can be obtained. For each sensor node, its c-nearest sensor nodes ($c = 6$) are determined, based on the Euclidean distance between sensor nodes' WGS84 coordinates. For each sensor node j, a dataset $D(j)$ is created by moving a fixed-length window ($p = 5$) over the measurements and storing the windows as training examples, together with labels gained by discretizing sensor measurements into five different ranges 0-5, 5-30,

[1] Data is publicly available at http://dublinked.ie .

Table 7.1: Prediction results for STRF and kNN, time slices of 30 min.

Global STRF model, 1 day, acc. 78.11%							
true/pred.	**0**	**1-5**	**6-20**	**21-30**	**31-60**	**61-485**	**precision**
0	3.09%	0.19%	0.04%	0.02%	0.01%	0.00%	94.30%
1-5	0.01%	2.32%	1.83%	0.01%	0.00%	0.00%	55.60%
6-20	0.34%	0.57%	44.71%	7.37%	0.31%	0.09%	83.80%
21-30	0.12%	0.00%	4.49%	20.71%	2.64%	0.05%	73.90%
31-60	0.16%	0.00%	0.22%	3.28%	7.15%	0.11%	65.50%
61-485	0.00%	0.00%	0.06%	0.01%	0.05%	0.13%	53.00%
recall	83.30%	77.10%	87.10%	65.90%	70.50%	34.00%	
Local kNN models, 1 month, acc. 85.73% +/- 6.99%							
true/pred.	**0**	**1-5**	**6-20**	**21-30**	**31-60**	**61-485**	**precision**
0	0.35%	0.07%	0.04%	0.03%	0.03%	0.02%	66.12%
1-5	0.23%	1.64%	0.43%	0.01%	0.00%	0.00%	71.06%
6-20	0.04%	1.02%	10.89%	2.24%	0.45%	0.03%	74.24%
21-30	0.01%	0.01%	1.04%	3.53%	1.29%	0.04%	59.79%
31-60	0.05%	0.01%	0.29%	2.04%	14.98%	2.09%	76.97%
61-485	0.14%	0.00%	0.04%	0.08%	2.52%	54.34%	95.12%
recall	43.10%	59.71%	85.53%	44.54%	77.73%	96.15%	

30-60, 60-150 and 150-260, shifting the label column by the prediction horizon $r = 1$ for correct alignment. This means we want to predict the traffic flow category for the next 15 minutes, based on five previous time slices (75 minutes) of measurements at j itself and each of j's six neighboring nodes. Each local dataset $D(j)$ thus consists of 2,971 real-valued examples with 5 attributes.

Local models are trained for each of the 296 sensor nodes and their nearest topological neighbors. k-NN with $k = 15$ is used as a supervised baseline learner which receives all individual labels, not just label counts. For learning from aggregated label counts, LLPC_{lsm} is used, with k-Means as inner clustering algorithm ($k = 15$, 50 different random starting points, 500 iterations at maximum) and the local search with multi-start strategy (150 starts). Different batch sizes $b = 25, 50, 75$ and 100 are tried. The accuracy of each method is assessed by a 10-fold cross validation, i.e. all models are trained and evaluated for different hold-out sets 10 times. In total $296 \times 7 \times 10 = 20,720$ models for k-NN need to be evaluated and $296 \times 7 \times 10 \times 4 = 82,880$ models trained and evaluated for LLPC_{lsm}. The evaluation has been done offline in parallel on different machines (about 36 CPU cores) and needed about a week.

The first question to answer is how the prediction accuracy of local models compares to much more sophisticated models, like the STRF described in [PLM13]. Graphical models may capture the whole joint-distribution between observations and labels. In comparison, local models naïvely assume that features from different nodes are conditionally independent, given the label. It can be expected that prediction performance decreases whenever this assumption doesn't hold.

The upper part of Tab. 7.1 shows the confusion matrix for an STRF trained on time

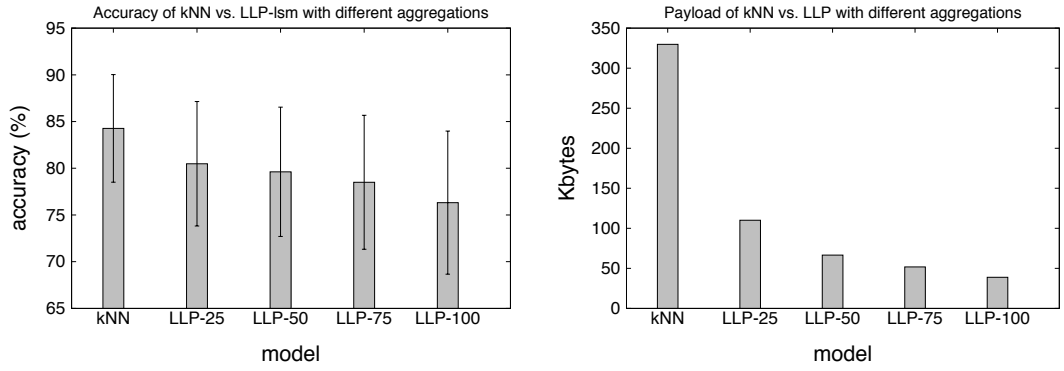
Table 7.2: Prediction results for kNN and LLPC_{lsm} ($b = 50$), slices 15 min., 1 month

Local kNN models, 84.27% +/- 5.76%						
true/pred.	**0-5**	**5-30**	**30-60**	**60-150**	**150-260**	**precision**
0-5	4.96%	1.08%	0.02%	0.01%	0.00%	81.70%
5-30	2.16%	32.26%	2.89%	0.09%	0.01%	86.25%
30-60	0.09%	2.57%	26.41%	3.35%	0.01%	81.44%
60-150	0.07%	0.08%	2.94%	20.45%	0.31%	85.75%
150-260	0.00%	0.00%	0.01%	0.05%	0.19%	75.98%
recall	68.05%	89.64%	81.86%	85.40%	37.10%	
Local LLPC_{lsm} ($b = 50$) models, 79.62% +/- 6.92%						
true/pred.	**0-5**	**5-30**	**30-60**	**60-150**	**150-260**	**precision**
0-5	4.47%	1.64%	0.22%	0.11%	0.00%	69.39%
5-30	2.65%	30.43%	3.46%	0.19%	0.01%	82.82%
30-60	0.09%	3.81%	25.05%	4.04%	0.02%	75.90%
60-150	0.07%	0.10%	3.52%	19.48%	0.31%	82.96%
150-260	0.00%	0.01%	0.01%	0.13%	0.19%	56.24%
recall	61.35%	84.55%	77.64%	81.37%	36.39%	

slices of 30 minutes over one day (the results originate from the study in [LPBM14]), while the lower part shows results for locally trained kNN models, also for time slices of 30 minutes, but one month. Further, for better comparison with [LPBM14], we have adapted discretization ranges accordingly. Despite the differences, one can see that the principal form of both tables is similar: Most observations are classified correctly, and therefore counted on the diagonal of the matrix. In total, the local kNN models even achieve a higher accuracy than the STRF, which might be explained by the STRF not having seen as many examples. Considering that local models cannot capture joint dependencies between features from different nodes, however, they perform quite well and their results look sensible.

A reevaluation of the previously shown discretization ranges, by plotting the distribution of traffic flow values over a whole month, suggest that an adaptation of ranges and time slices of 15 minutes may be more beneficial for the task of traffic flow prediction. Table 7.2 therefore shows the prediction results of k-NN and LLPC_{lsm} models ($b = 50$) for the adapted ranges and time slices of 15 minutes, evaluated over a whole month. Again, both matrices show a similar form and many observations are correctly predicted, being counted on the diagonal of the matrix. LLPC_{lsm} trained on aggregated label information has a lower recall than k-NN evaluated on all labels, but the relative differences in recall across columns are similar. With LLPC_{lsm}, precision suffers when predicting lowest or highest discretization ranges. Although the total accuracy of LLPC_{lsm} is lower by about five percentage points, its general performance in the setting is still comparable to the performance of the much more complex STRF model.

Figure 7.2 shows the trade-off between accuracy and payload sent for k-NN and LLPC_{lsm} trained on differently sized batches of aggregated labels. Besides the average accuracy over all 10-fold cross-validations at each node, the bars in Fig. 7.2(a) also depict



(a) Accuracy (b) Payload

Figure 7.2: Trade-off between accuracy and payload sent for kNN and LLPC_{lsm}

the standard deviation of accuracy over all nodes. LLPC_{lsm}'s accuracy decreases with larger batch sizes, with a steeper decrease for the largest batch size 100. The standard deviation of LLPC_{lsm} is slightly larger than that of k-NN. Figure 7.2(b) shows the total payload transmitted by all nodes for the training of local models, under the assumption that messages can be broadcast to neighboring nodes (otherwise numbers would only change by the same constant factor for all models). The k-NN models at j's neighboring nodes and j itself use all available labels, while LLPC_{lsm} models are trained on aggregates of j's labels. When using LLPC_{lsm} with a batch size of 25 aggregated labels, nodes need only send a third of the payload to their neighboring nodes in comparison to k-NN sending and using all labels. For LLPC_{lsm} with $b = 50$, only about a fifth of all available label information needs to be transmitted. LLPC_{lsm} with $b = 100$ even decreases the payload sent by a factor of about 8.5. Experiments show that sending aggregated label counts instead of all labels saves communication, while accuracy is still in the order of STRF's performance.

However, training a global model, like the STRF, requires the centralization of all sensor measurements. In total, 28 MB would need to be transmitted, assuming a central windowing, i.e. $p = 1$. In comparison, k-NN transmits only 330 KB using all labels, and LLPC_{lsm} with $b = 50$ transmits only 67 KB using aggregated label counts. In other words, local models with a supervised classifier like k-NN transmit 85 times less data than would be required for training a global model at some central node or data center, and LLPC_{lsm} with $b = 50$ transmits 418 times less data, yet maintaining an accuracy which is comparable to a global model. In fact, the label counts transmitted by LLPC_{lsm} with $b = 100$ would fit into a single SMS per node, for a whole month.

Finally, Fig. 7.3 compares the prediction accuracies of LLPC_{lsm}, $b = 50$ and k-NN for different sensor nodes, mapped to the junctions on a street map of Dublin. In general, LLPC_{lsm} performs slightly worse than k-NN, which can be seen by nodes on the upper

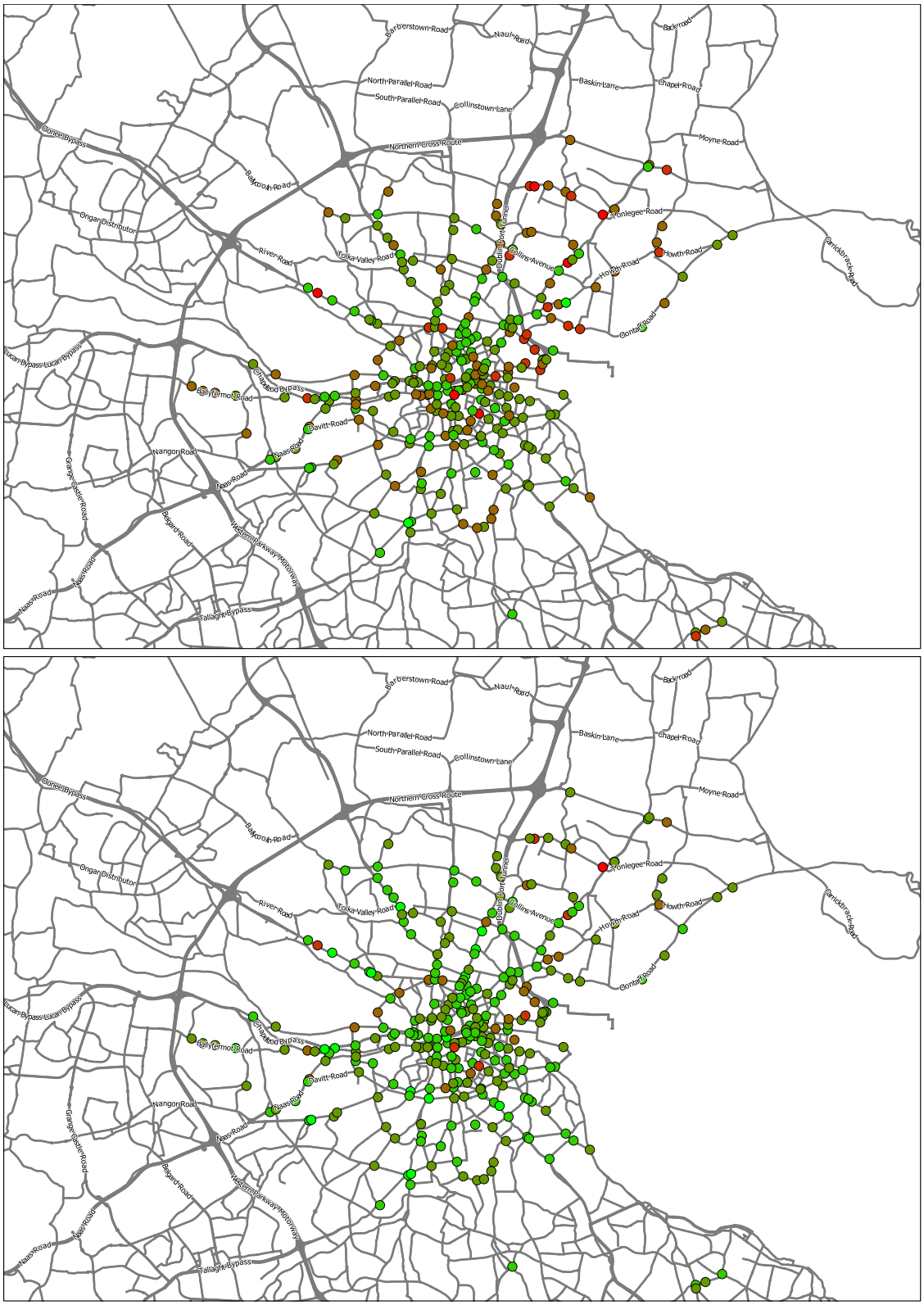


Figure 7.3: Accuracy of LLPC_{lsm}, $b = 50$ (upper map) in comparison to k-NN (below), color ranges from red - low accuracy (50%), till green - high accuracy (99%) (best viewed in color)

map having a slightly darker color than those on the map below. Nevertheless, there are still many junctions for which the traffic flow is predicted quite well with LLPC_{lsm}. Some locations have bad performance in both plots, the map reveals that these are locations of parking areas, e.g. inner-city parking houses and recreational areas where many vehicles stay for a long period of time.

7.7 Summary, Conclusions and Outlook

Complex global models which can capture the whole joint probability distribution of observations and labels, like STRFs, allow for answering many different types of queries, based on a single model. However, transmission of all data to a central node or data center may lead to high communication costs for training and prediction. Further, centralization can easily create single points of failure, and the use of cabled networks for the transmission of sensor measurements may lead to high maintenance costs due to construction work. A decentralized system consisting of cheap, battery-powered presence sensors would be much more flexible. However, distributed data analysis in such a setting, especially in the vertically partitioned data scenario, is very challenging.

An approach has been proposed which restricts learning to local neighborhoods of topologically close sensors. At each node, a local model is trained, based on label counts received from a fixed number of neighboring nodes. Here, learning from label proportions, as introduced in Chap. 6, solves a problem which has been largely neglected in the existing literature so far, which is the communication-efficient transmission of label information to local nodes.

Scalable traffic flow prediction then involves a trade-off to be made between the accuracy of models and the amount of communication between networked nodes. Using local models, we assume conditional independence of features, given the label. This assumption usually does not hold in a dense street network. Nevertheless, the approach of training local models from label counts, TLMC, has been successfully evaluated on traffic flow data from the city of Dublin and shows the feasibility of the proposed approach. Though accuracy drops by five percentage points in comparison to local k-NN models, which use all labels, LLPC_{lsm}, which aggregates labels in bags of size $b = 50$, still shows an accuracy comparable to a centrally trained global STRF model. At the same time, LLPC_{lsm} transmits about five times less data than k-NN, and 418 times less data that would need to be sent when centralizing all data. The data sent by LLPC_{lsm} with bag size $b = 100$ would even fit into a single SMS, containing the label counts for a whole month.

The results achieved demonstrate that the training of local models and combining their predictions can lead to highly communication-efficient distributed algorithms for the vertically partitioned data scenario which are nevertheless sufficiently accurate. Especially, in comparison to consensus algorithms, labels need to be exchanged only once during training, and not iteratively. Restricting communication to topologically close sensors is beneficial for wireless communication, where power consumption increases

with transmission distance. With adaptations, the approach might also respect dynamically changing neighborhoods. TLMC makes use of LLPC_{lsm}, which should be especially well-suited for small resource-constrained devices when using k-Means as inner clustering algorithm. The number of k-Means' iterations can be restricted such that it has linear running time, and its operations consist mostly of distance calculations. Those are based on simple arithmetic operations, like addition and multiplication. The local search with multi-start strategy for the labeling of clusters is fast and achieved sufficient accuracy in the conducted experiments. Moreover, k-Means has a small memory footprint of $O(kp)$, since its model consists of only k centroids which are p-dimensional.

Chapter 8

Vertically Distributed Core Vector Machine

In Sect. 2.1, it has been discussed how more advanced IoT applications could be realized by fully embedding data analysis into all parts of an IoT system. The automation of processes which integrate data analysis and control also requires the automatic preprocessing of data, which includes steps of data cleansing and the detection of abnormalities. For instance, observations of physical processes suffer from instrument malfunction and noise. In contrast, during modeling, rare events are not to be excluded, since they can be the most interesting findings. For instance, faults in production processes occur relatively seldom, but can have a large influence on the final quality of products. The earlier such anomalies are detected, the more options remain to improve on the final quality.

In production, features about the processing of single products are usually assessed at different machines, which resembles the vertical partitioning of data across networked nodes. Each machine may generate a large number of process parameters. Although bandwidth is high, due to the use of local area networks, the time for making decisions can be limited. Local analysis and a reduction of data before transmission could enhance response times and reduce collisions and retransmissions over central buses. In discrete manufacturing and logistics, more and more products are instrumented with wireless sensors. Similarly, monitoring applications in earth science, like prediction of rare volcanic outbursts, may build on vertically partitioned measurements from wireless networked sensors. Intrusion detection needs to detect anomalies in large amounts of network traffic [GKRB09]. In physics, the detection of rare astronomical events could be improved by combining the data from several telescopes. However, these produce masses of data [BBB+15] whose transmission over satellite connections would take several years. In all aforementioned cases, communication is limited. The automatic detection of outliers or rare events in such scenarios therefore requires new kinds of communication-efficient distributed anomaly detection algorithms.

As shown in Sect. 4.5.2, outlier detection in the vertically partitioned data scenario can be especially challenging, since outliers may be determined by combinations of fea-

ture values from different nodes. In this chapter, a new algorithm for anomaly detection on vertically partitioned data is proposed. It aggregates the data directly at the local storage nodes using RBF kernels. Only a fraction of the data is communicated to a central node. Through extensive empirical evaluation on controlled datasets, it is demonstrated that the method can be an order of magnitude more communication-efficient than state-of-the-art methods, achieving a comparable accuracy.

The rest of this chapter is organized as follows. In Sect. 8.1, work is shortly discussed which is related to the task of outlier detection and wasn't mentioned already in Chap. 4. The distributed task for the vertically partitioned data scenario is defined more formally in Sect. 8.2. In Sect. 8.3, the approach presented in [DBV11] is shortly reviewed, and then improved on by developing a distributed version of the Core Vector Machine (CVM) algorithm (see Sect. 3.3.2). The new approach is evaluated on several controlled and standard datasets in Sect. 8.4. The chapter is shortly summarized and conclusions are drawn in Sect. 8.5.

8.1 Related Work

Outlier or anomaly detection [CBK09] is the task of identifying abnormal or inconsistent patterns in a dataset (see also Sect. 3.3). It is a well studied problem in the fields of data mining, machine learning, and statistics. Outliers can be detected using *unsupervised*, *supervised*, or *semi-supervised* techniques [HA04, CBK09]. In some cases, outliers are looked at as undesirable data points which might disturb analysis. However, in other cases, their identification and further analysis can be crucial to many tasks such as fraud and intrusion detection [CFPS99], climate pattern discovery in Earth sciences [ZRDZ07], quality control in manufacturing processes [HSSK06], and adverse event detection in aviation safety applications [DMSO10].

In the field of distributed anomaly detection, researchers have mainly focused on the horizontally partitioned data scenario. In [LA05], the PBay algorithm is proposed, where a master node first splits the data into separate chunks for each processor. Then the master node loads each block of test data and broadcasts it to each of the worker nodes. Each worker node then executes a distance based outlier detection technique using its local database and the test block. A parallel version of the basic nested loop algorithm is presented in [HC02]. However, it is not communication-efficient, since it requires the dataset to be exchanged among all the nodes. A distributed algorithm for mixed attribute datasets is presented in [OGP06]. The algorithm introduced in [ABLS10] is a distributed distance-based outlier detection algorithm based on the concept of solving set which can be viewed as a compact representation of the original dataset. The solving set is such that by comparing any data point to only the elements of the solving set, it can be concluded if the point is an outlier or not. More recently, a distributed method using an efficient parallel pruning rule has been proposed in [BMG11]. The authors of [BGS+12] present a distributed algorithm for detecting outliers in streams.

To the best of our knowledge, only few communication-efficient algorithms have been

proposed for the vertically partitioned scenario. This task is particularly challenging, if the analysis depends on a combination of features from more than one node. To the best of our knowledge, the only anomaly detection algorithm for the scenario is the vertically distributed 1-class SVM introduced in [DBV11]. It trains one global 1-class model at a central coordinator, reducing communication costs by sampling from the local nodes. Then, it trains local 1-class models and reduces communication with a central coordinator by a pruning rule. However, it comes with the disadvantage that certain kinds of outliers might not be detected and the size of the global sample must be user-specified. In the following section, the problem setting focused on is defined more formally. Then, the distributed 1-class SVM is reviewed shortly and it is explained how it can be improved using the Core Vector Machine (CVM) algorithm (see Sect. 3.3.2).

8.2 Problem Setting

The problem setting described in Sect. 4.5 is shortly reviewed here. We assume that there is a single underlying distribution $P(X)$ of observations consisting of p features, but that each node stores only partial information about observations, i.e. subsets of their features $A_1, \ldots, A_p$. Let $\mathbf{x}[j] \in \mathbb{R}^{p_j}$ denote a vector which contains p_j features of observation $\mathbf{x}$ available at node j. Columns of the data matrix $\mathbf{D}$ are then split over the nodes, i.e. each node j stores a $n \times p_j$ submatrix $\mathbf{D}[j]$ whose rows consist of vectors $\mathbf{x}_1[j], \ldots, \mathbf{x}_n[j]$ with $\mathbf{x}_i[j] \in \mathbb{R}^{p_j}$. In the following, an individual feature q stored at node j will be denoted by $\mathbf{x}[j][q]$.

Given data matrices $\mathbf{D}[1], \ldots \mathbf{D}[m]$ stored at m different local nodes in a network, the task is to detect global outliers in the data, without communicating *all* data (or even more) to a central node or between nodes. In other words, the algorithm should be communication-efficient. Here, *global* means that the observation deviates somehow from the underlying distribution of (p-dimensional) observations. As explained in Sect. 4.5.2, the development of communication-efficient algorithms for the detection of global outliers is difficult, since outlier patterns may be determined by a combination of feature values stored at different nodes, requiring communication.

8.3 Vertically Distributed CVM (VDCVM)

In [DBV11], a synchronized distributed anomaly detection algorithm for vertically partitioned data has been proposed. It is based on the 1-class ν-SVM (see Sect. 3.3.1). The distributed version is called VD**S**VM in the following. The algorithm works as follows.

At each local node $1, \ldots, m$, a local 1-class model is trained. Points identified as local outliers are sent to a central coordinator node $j = 0$, together with a small sample of all observations. At the central coordinator, a global 1-class model is trained, based on the received sample. The global model is then used to decide if the outlier candidates sent from the local nodes are global outliers or not. During prediction, only outlier candidates are sent and then checked against the global model. While the algorithm is

highly communication-efficient during training and especially in the prediction phase, it comes with two drawbacks.

The first drawback is that the algorithm can miss certain types of global outliers. Although the authors of [DBV11] have shown that the probability to detect such global outliers correctly increases with growing dimensions, missing such outliers can still happen. The problematic situation is depicted in Fig. 4.2(d) (see Sect. 4.5), where the global outlier is no outlier in any of the dimensions. Especially, it should be noted that the number of features p is a parameter which cannot be controlled, but is given. Stated in terms of the 1-class SVM, the problem is that instead of a single global 1-class model, we have several local models which were trained on entirely different subspaces of the whole data matrix $\mathbf{D}$, and therefore do not agree on the same set of support vectors. The situation is thus similar to the Separable SVM presented in the previous chapter, where also local models are trained. To reach consensus on predictions between local models, approaches like the Alternating Direction Method of Multipliers (ADMM) iteratively exchange averages of such predictions and adapt local models accordingly (see Sect. 4.4.2). However, none of the presented consensus methods was made specifically for the detection of outliers.

The second drawback of the approach presented in [DBV11] is that the accuracy of the global model depends on the number of sample points, which is user-specified. Unfortunately, how many observations are needed for learning a well-performing model in practice depends on the dimension p and underlying distribution $P(X)$, which is unknown. Therefore, for a user it is very difficult to specify an according sample size in advance.

As we have seen in Sect. 3.3.2, the Core Vector Machine (CVM) algorithm uses a probabilistic speedup strategy to reach a $(1+\varepsilon)$-approximation of the minimum enclosing ball (MEB) around all normal observations with high probability. It only samples as many observations as needed to reach such approximation. Proposed are therefore two modifications of the VDSVM:

1. The standard 1-class ν-SVM at the central coordinator should be replaced by the CVM algorithm. The CVM can then sample as many observations as needed from the local nodes, potentially saving communication. It should be noted that by using the CVM, we remain communication-efficient, since the CVM cannot sample more observations as given. In fact, the number of iterations, which is the size of the core set, is bounded by some constant.

2. Instead of training local models at each node, the furthest point calculation of the original CVM algorithm (see Alg. 1) can be replaced by a distributed computation over local nodes. As will be shown, this modification can lead to a substantial reduction of communication costs during training, depending on the total number of iterations.

In Sect. 8.3.1, it is first shown how the furthest point calculation of the CVM can be replaced by a distributed computation over local nodes, such that the amount of data

which needs to be sent to a central coordinator is reduced. Section 8.3.2 presents the full algorithm, the Vertically Distributed Core Vector Machine (VDCVM). The algorithm's properties, like communication costs, are analyzed in Sect. 8.3.3.

8.3.1 Distributed Furthest Point Calculation

In any iteration t, the original CVM algorithm (see Alg. 1) with probabilistic speedup draws a fixed-sized sample of data points from the whole dataset. Let V_t denote the sample drawn at iteration t and $|V_t|$ denote its size. From the sample, the CVM determines the point $\mathbf{z}_t$ furthest away from the current center $\mathbf{c}_t$ in feature space by calculating the squared distance $||\mathbf{c}_t - \tilde{\phi}(\mathbf{z}_\ell)||^2$ for each sample point $\mathbf{z}_\ell \in V_t$:

$$||\mathbf{c}_t - \tilde{\phi}(\mathbf{z}_\ell)||^2 = \sum_{\mathbf{z}_u, \mathbf{z}_v \in \mathcal{S}_t} \alpha_u \alpha_v \tilde{k}(\mathbf{z}_u, \mathbf{z}_v) - 2 \sum_{\mathbf{z}_u \in \mathcal{S}_t} \alpha_u \tilde{k}(\mathbf{z}_u, \mathbf{z}_\ell) + \tilde{k}(\mathbf{z}_\ell, \mathbf{z}_\ell) \tag{8.1}$$

Since $\tilde{k}(\mathbf{z}_\ell, \mathbf{z}_\ell) = \tilde{\kappa}$ is constant and the sum $\sum_{\mathbf{z}_u, \mathbf{z}_v \in \mathcal{S}_t} \alpha_u \alpha_v \tilde{k}(\mathbf{z}_u, \mathbf{z}_v)$ does not depend on the sampled points, the furthest point calculation at iteration t can be simplified to

$$\mathbf{z}^{(t)} = \text{argmax}_{\mathbf{z}_\ell \in V_t} \left[- \sum_{\mathbf{z}_u \in \mathcal{S}_t} \alpha_u \tilde{k}(\mathbf{z}_u, \mathbf{z}_\ell) \right] \tag{8.2}$$

Separability by Linear Kernel Let $\mathbf{z}[j]$ denote the components of vector $\mathbf{z}$ stored at node j. With the linear dot product kernel $k(\mathbf{z}_u, \mathbf{z}_v) = \langle \mathbf{z}_u, \mathbf{z}_v \rangle$, the sum in (8.9) could be written as

$$\mathbf{z}^{(t)} = \text{argmin}_{\mathbf{z}_\ell \in V_t} \sum_{\mathbf{z}_u \in \mathcal{S}_t} \alpha_u \langle \mathbf{z}_u, \mathbf{z}_\ell \rangle = \text{argmin}_{\mathbf{z}_\ell \in V_t} \sum_{j=1}^{m} \sum_{\mathbf{z}_u \in \mathcal{S}_t} \alpha_u \langle \mathbf{z}_u[j], \mathbf{z}_\ell[j] \rangle \tag{8.3}$$

Since the dot product kernel multiplies each component j of the $\mathbf{z}_\ell$ and $\mathbf{z}_u$ vectors independently, at each node j we can calculate the partial sum

$$v_\ell(j) = \sum_{\mathbf{z}_u \in \mathcal{S}_t} \alpha_u \langle \mathbf{z}_u[j], \mathbf{z}_\ell[j] \rangle \tag{8.4}$$

for all random indices $\ell \in I_t$ and send these partial sums back to the coordinator. The coordinator then aggregates the sums and determines the index $\ell_{\max} \in I_t$ of the furthest point:

$$\ell_{\max} = \text{argmin}_{\ell \in I_t} \sum_{j=1}^{m} v_\ell(j) \tag{8.5}$$

Each local node thus only transmits a *single* numerical value, the partial sum $v_\ell(j)$, which is a scalar, for each point of the random sample, instead of sending *all* attribute values of the sampled points to the central coordinator node. Similar to using local models, which send only a prediction per observation, communication between local

nodes and central coordinator has become independent of the number of features p_j stored at each node j.

In each case, the linear kernel ensures *separability* of the furthest point calculation (8.9) over local nodes. However, the CVM requires the kernel $\tilde{k}(\mathbf{x}, \mathbf{x})$ to be constant. The linear kernel does not fulfill $\tilde{k}(\mathbf{x}, \mathbf{x}) = \tilde{\kappa}$. Can we replace the linear kernel by the RBF kernel $k(\mathbf{z}_u, \mathbf{z}_v) = e^{-\gamma||\mathbf{z}_u - \mathbf{z}_v||^2}$, which fulfills the requirement and is usually used with the CVM? Unfortunately, as the following subsection shows, the standard RBF kernel makes the furthest point calculation non-separable over nodes.

Non-separability by RBF-Kernel In Sect. 4.5.3, it has been shown that the RBF kernel is separable over vector components:

$$k(\mathbf{x}, \mathbf{x}') = e^{-\gamma||\mathbf{x}-\mathbf{x}'||^2} \tag{8.6}$$

$$= e^{-\gamma(\mathbf{x}[1]-\mathbf{x}'[1])^2+\ldots+-\gamma(\mathbf{x}[p]-\mathbf{x}'[p])^2} \tag{8.7}$$

$$= e^{-\gamma(\mathbf{x}[1]-\mathbf{x}'[1])^2} \cdot \ldots \cdot e^{-\gamma(\mathbf{x}[p]-\mathbf{x}'[p])^2} \tag{8.8}$$

This property has been used by the privacy-preserving SVMs presented in Sect. 4.4.1, which first transfer local kernel matrices to a central coordinator node and then use the Hadamard componentwise product to multiply the entries of such kernel matrices, resulting in the global kernel matrix. Despite separability, such methods need to communicate quadratic kernel matrices, which is not communication-efficient.

While the RBF kernel itself is separable, in case of the furthest point calculation it appears in the following sum:

$$-\sum_{\mathbf{z}_u \in S_t} \alpha_u e^{-\gamma||\mathbf{z}_u - \mathbf{z}_\ell||^2} \tag{8.9}$$

Following (8.8), this sum can be rewritten as follows

$$-\sum_{\mathbf{z}_u \in S_t} \alpha_u e^{-\gamma||\mathbf{z}_u - \mathbf{z}_\ell||^2} = -\sum_{\mathbf{z}_u \in S_t} \alpha_u \left(e^{-\gamma(\mathbf{z}_u[1]-\mathbf{z}_\ell[1])^2} \cdot \ldots \cdot e^{-\gamma(\mathbf{z}_u[p]-\mathbf{z}_\ell[p])^2} \right) \tag{8.10}$$

What we see here is that, although each individual factor could be computed independently from each other across local nodes, the products in the sum cannot. The products must be calculated before summation, and the factors in each summand base on information from different local nodes. The sum has as many summands as points in the core set at iteration t. Of course, we could transmit $s|S_t|$ ($s = |V_t|$) scalars to the central node, which could sum them up for each point in S_t. However, as with the privacy-preserving SVMs, this would lead to quadratic communication costs, which is not communication-efficient.

Separability by Combination of RBF-Kernels Since the standard RBF kernel makes the furthest point calculation non-separable over nodes, leading to high communication costs, we propose to use a combination of RBF kernels, as it was already

introduced in the previous chapter:

$$k(\mathbf{z}_u, \mathbf{z}_v) = \sum_{j=1}^{m} e^{-\gamma_j ||\mathbf{z}_u[j] - \mathbf{z}_v[j]||^2} \tag{8.11}$$

For the combination of RBF kernels, $\tilde{k}(\mathbf{z}_u, \mathbf{z}_v) = \tilde{\kappa}$, since each summand is constant, and a sum of constants is constant again. With a combination of RBF kernels, the furthest point calculation finally becomes

$$\mathbf{z}^{(t)} = \text{argmin}_{\mathbf{z}_\ell \in V_t} \sum_{j=1}^{m} \sum_{\mathbf{z}_u \in \mathcal{S}_t} \alpha_u e^{-\gamma_j ||\mathbf{z}_u[j] - \mathbf{z}_\ell[j]||^2}, \tag{8.12}$$

which is separable over the local nodes.

8.3.2 The VDCVM Algorithm

Based on our previous discussion, in this section, we introduce the distributed components of the Vertically Distributed Core Vector Machine (VDCVM) and accompanying operations. The components are shown in Fig. 8.1. The *Coordinator* communicates with *Worker* components at each local node which have direct access to a local data repository. This means that local nodes can access the values of all locally stored features directly, without any network communication. While the termination check and the QP optimization (steps 3 and 4 of the CVM algorithm, see Alg. 1) are still done centrally by the *Coordinator*, the sampling and furthest point calculations are combined in a single step and done in parallel by the *Worker* components, as described in the following.

During initialization, the *Coordinator* retrieves user-defined parameters, like C and $\gamma_1, \ldots, \gamma_m$ and the number s of points to sample in each iteration, from the client, and meta information attached to the local data matrices from all *Data Repository* components, like the total number of rows n and the numbers of columns p_j. The coordinator initializes its internal data structures. The components of an initial (center)

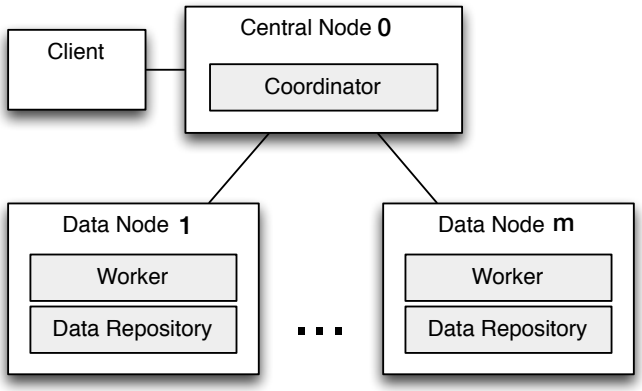


Figure 8.1: Distributed components of the VDCVM

Algorithm 11 Operations of the VDCVM *Coordinator*

on *workerInitialized*:
  if received message from all workers **then**
    Determine random index set I_0 for data points.
    Send *getPartialSums(I_0)* to all workers.

on *getPartialSumsAnswer($v_\ell(j)\ \forall \ell \in I_t$)*:
  Store partial sums received from worker j.
  if received message from all workers **then**
    $\ell_{\max} = \text{argmin}_{\ell \in I_t} \sum_{j=1}^{m} v_\ell(j)$
    Send *getData(ℓ_{max})* to all repositories.

on *getDataAnswer($\mathbf{z}^t[1 \dots p_j]$)*:
  Store attribute values received from repository j.
  if received message from all repositories **then**
    Construct furthest point $\mathbf{z}^t$ from attribute values.
    if $||\mathbf{c}_t - \phi(\mathbf{z}^t)|| \leq (1 + \varepsilon) \cdot R_t$ **then**
      Return model to client.
    else
      $\mathcal{S}_{t+1} := \mathcal{S}_t \cup \{\mathbf{z}^t\}$.
      Calculate new MEB($\mathcal{S}_{t+1}$). (Solve QP problem.)
      $R_{t+1} := \sqrt{\tilde{\kappa} - \alpha^T \tilde{\mathbf{K}} \alpha}$.
      Determine random index set I_{t+1} for data points.
      Send *getPartialSums(I_{t+1}, αs,ℓ_{max})* to workers.
      $t := t + 1$.

Algorithm 12 Operations of the VDCVM *Worker*

on *getPartialSums(I_t, αs, ℓ_{max})*:
  if $t > 0$ **then** store new αs.
  if $t > 0$ **then** $S_{t+1} := S_t \cup \{\mathbf{z}_{\ell_{\max}}[1 \dots p_j]\}$.
  Calculate $v_\ell(j)\ \forall \ell \in I_t$ — see (8.4) and (8.5)

point $\mathbf{z}$ are all set to 0.5, under the assumption that feature values of all observations are normalized to the $[0, 1]$ range. Thereby, $\mathbf{z}$ does not need to be sampled from the network. Then, the constant $\tilde{\kappa}$ is precomputed. Finally, C, γ_j and $\tilde{\kappa}$ are sent to each local node j, for $j = 1, \dots, m$. Each *Worker* in turn initializes its own data structures. It signals being finished by sending a *workerInitialized* message to the *Coordinator*, which can then start with the first iteration.

The main part of the *Coordinator* beyond initialization is shown in Alg. 11. The indices I_t of $|V_t| = s$ random data points are sampled and sent to the workers in a request for the partial sums v_ℓ. When the *Coordinator* has received all partial sums,

it can calculate the index $\ell_{\max}$ of the furthest point $\mathbf{z}^t$ and ask the repositories for its feature values. If the termination criterion is fulfilled, it returns the model (i.e. the core set points). Otherwise, the coordinator goes on with solving the QP problem and calculates the new radius R_{t+1}. It then determines a new random index set I_{t+1} and requests the next partial sums from the workers. It furthermore transmits all updated α values and the index of the previously determined furthest point, $\ell_{\max}$.

Each *Worker* (see Alg. 12) gets the local components of point $\mathbf{z}^t$ by its furthest index $\ell_{\max}$ and updates its own local core set accordingly. Based on the updated αs it received, it then calculates $v_\ell(j)$ for all random indices $\ell \in I_t$ received from the *Coordinator*, according to (8.5). The partial sums are then sent back to the *Coordinator* which continues with the main algorithm.

8.3.3 Analysis of Run-Time and Communication Costs

The VDCVM performs exactly the same calculations as the original CVM algorithm. It therefore inherits all properties of the CVM, including the constant bound on the total number of iterations (see Sect. 3.3.2) and the probabilistic $(1+\varepsilon)^2$-approximation guarantee for the calculated MEB.

Regarding communication costs, we assume that messages with same content can be broadcast to all nodes, that observation's indices are represented by 4 bytes and real numbers by 8 bytes. The total number of bytes transferred (excluding initialization and message headers) when sending all p attributes of n points in a sample to a central server for training (as does VD**S**VM) is

$$z_{\text{central}}(n) = n \cdot 4 + n \cdot p \cdot 8$$

In contrast, the bytes transferred by VD**C**VM up to iteration T are

$$z_{\text{VDCVM}}(T) = [T \cdot s \cdot 4 + T \cdot s \cdot m \cdot 8] + [T \cdot 4 + T \cdot p \cdot 8] + \left[\frac{T(T+1)}{2} \cdot 8\right]$$

The coordinator at the central node first broadcasts s index values of observations for which partial sums should be calculated to all local nodes (first summand in first bracket). It then receives different partial kernel sums for each observation, from m local nodes (second summand in first bracket). Then, the index value of the furthest point is broadcast to all local nodes (first summand in second bracket) and the coordinator receives its p feature values (second summand in second bracket). The total number of αs transmitted is quadratic in the number of iterations (last bracket).

Although the transmission of αs leads to quadratic communication costs over all iterations, there is a big difference to the quadratic costs which would result from using the original RBF kernel and transferring $s|S_t|$ scalars in each iteration (see equation (8.10) in the previous section). With the RBF kernel, communication costs would be quadratic in the total number of examples sampled over iterations, i.e. $O((Ts)^2)$. In comparison, by using a combination of RBF kernels, which makes the furthest point calculation separable over local nodes, communication costs are quadratic in the number of core set points,

Table 8.1: Maximum number of observations up to which the VDCVM is more communication-efficient than sending all n observations to a central node, for different numbers of nodes m and attributes p, $s = 59$.

p	m=1	m=2	m=5	m=10	m=25
10	61,478	54,516	33,630	0	-
25	164,138	157,176	136,290	101,480	0
50	335,238	328,276	307,390	272,580	168,150
100	677,438	670,476	649,590	614,780	510,350
250	1,704,038	1,697,076	1,676,190	1,641,380	1,536,950
500	3,415,038	3,408,076	3,387,190	3,352,380	3,247,950

i.e. $O(T^2)$. In each iteration, just a single core set point is added. This means that for a combination of RBF kernels, the costs are growing slower by a constant factor s, which is the number of points to sample in each iteration. This is a standard parameter of the CVM and user-specified.

Due to the quadratic communication costs, there exists a number of iterations from which on the VDCVM would become less communication-efficient than transmitting each single observation to the central coordinator node. This break even point T_{worse} can be calculated by setting $z_{\text{central}}(n)$ with $n = Ts$ equal to $z_{\text{VDCVM}}(T)$, solving for T:

$$T_{\text{worse}} = 2 \cdot p \cdot (s - 1) - 2 \cdot m \cdot s \tag{8.13}$$

This means that, given a fixed number of nodes m and sample size s, communication costs largely depend on the number of features stored over the nodes. Table 8.1 contains values of $s \cdot T_{\text{worse}}$, i.e. the maximum number of observations that can be analyzed with the VDCVM, while still being more communication-efficient than transmitting all observations to a central node, for different numbers of local nodes m and numbers of attributes p.

The first point to note is that the maximum number of observations that can be analyzed in a communication-efficient way with the VDCVM already exceeds the size of half of the datasets found in the original CVM paper [TKC05]. If we set $s = 150$, i.e. allow for more observations being sampled in each iteration, we could even handle all datasets listed in the CVM paper communication-efficiently, except for checkerboard, which has only two dimensions. Note that all such datasets are considered to be large scale, since the CVM itself has been developed for large scale datasets.

The second point to note is that the table shows the *maximum* number of observations that can be handled communication-efficiently, but not the actual number which would *need to be sampled* to reach a $(1 + \varepsilon)$-approximation of the MEB with high probability. In practice, the number of observations actually being sampled can be much lower, depending on the difficulty of the domain. For instance, when running the experiments described in Sect. 8.4, much smaller sample sizes were needed for datasets where outliers are easy to separate from the normal observations.

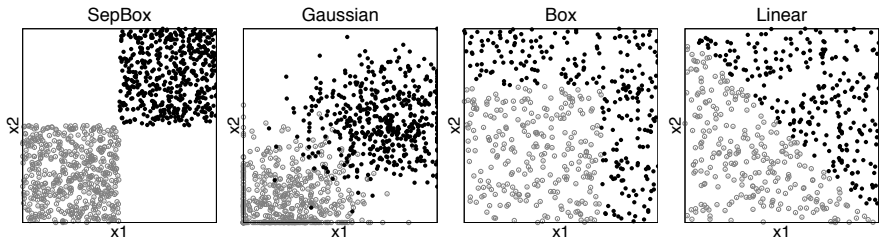


Figure 8.2: Generated normal data (grey) and outliers (black) in two dimensions.

8.4 Experimental Evaluation

In this section, we demonstrate the performance of the VDCVM on a variety of datasets and compare it to the VDSVM and a single central 1-class ν-SVM model. In 1-class learning, the ground truth about the outliers is often not available. For a systematic performance evaluation of the algorithms, synthetic data containing known outliers was therefore generated. In addition, the methods also have been evaluated on three real world datasets with known binary class labels.

Synthetic Data Figure 8.2 visualizes the generated datasets for two dimensions. The points were generated randomly in a unit hypercube of m dimensions (for $m = 2, 4, 8, 16, 32, 64$). The different types of data pose varying challenges to the algorithms when vertically partitioned among network nodes, according to the discussion in Sect. 4.5. The easiest scenario should be the one in which each attribute reveals all information about the label, represented by `SepBox`. For `Gaussian`, the means $\mu_{+,-}$ and standard deviations $\sigma_{+,-}$ of two Gaussians were chosen randomly and independently for each attribute (with $\mu_{+,-} \in [0.1, 0.9]$ and $\sigma_{+,-} \in [0, 0.25]$). If the Gaussians overlap in each single dimension, they may nevertheless become separable by a combination of attributes. Moreover, with more attributes, such an overlap becomes more improbable. In the `Box` dataset, an outlier is a point for which $\exists\, \mathbf{x}[i] > \rho$ with $\rho = 0.5^{(1/m)}$ (i.e. the normal data lies in half the volume of the m-dimensional unit hypercube). Separation is only given by all dimensions in conjunction. The same is true for the `Linear` dataset, where the normal data is separated from the outliers by the hyperplane $h = \{\mathbf{x} \,|\, \mathbf{x}/||m|| - 0.5||m|| = 0\}$.

Real World Data All real world data was taken from the CVM authors' web site[1]. The `letter` dataset consists of 20,000 data points for the 26 letters of the latin alphabet, each represented by 16 attributes. For the experiments, 773 examples of the letter `G` were taken as normal data and 796 of letter `T` extracted as outliers. The KDDCUP-99 data consists of 5,209,460 examples of network traffic described by 127 features. The task is to differentiate between normal and bad traffic patterns. The extended MIT face

[1] `http://c2inet.sce.ntu.edu.sg/ivor/cvm.html`

Table 8.2: Number of data points (total, training, test and validation set)

Dataset	Total	Training	Test		Validation	
			normal	outliers	normal	outliers
Random datasets	60,000	20,000	10,000	10,000	10,000	10,000
`letter`	1,000	400	150	150	150	150
`kddcup-99`	60,000	20,000	10,000	10,000	10,000	10,000
`face`	20,000	10,000	2,500	2,500	2,500	2,500

dataset contains 513,455 images consisting of 19x19 (361) grey scale values. The task is to decide if the image contains a human face or not.

8.4.1 Experimental Setup

VDCVM was implemented in Java using the Spread Toolkit[2]. VDSVM was implemented in Python using LibSVM. Also the results for the central 1-class ν-SVM model were obtained with LibSVM.

Table 8.2 shows that 60,000 points were generated for each of the random datasets. From each of the real-world datasets, only a random sample was taken (column *Total*), because the LibSVM would have had problems to handle datasets with such a large number of observations. The samples were randomly split further into independent sets for training, validation (i.e. parameter optimization) and testing, with sizes as shown (see also Sect. 3.1.7 on validation). The central and local VDSVM models were trained on the whole training set, while VDCVM was allowed to sample up to the same amount of data. The methods require different parameters γ and ν (or C), since VDSVM uses a standard RBF kernel and the 1-norm on its slack variables, while VDCVM uses the 2-norm and a combination of local kernels. For VDSVM, 75 random parameter combinations were tested, and for VDCVM 100 combinations, alternatingly conducting a local and global random search. All error rates reported were obtained from a single run on the test set, with parameters tuned on the validation set.

8.4.2 Experimental Results

The plots in Figure 8.3 compare the performance of VDCVM to a single central 1-class model with standard RBF kernel and to VDSVM, i.e. local 1-class models which communicate only outlier candidates to the central coordinator for testing. The error rates are averaged over the results obtained for different numbers of nodes (2, 4, 8, 16, 32), with error bars indicating the standard deviation.

Prediction Performance on Synthetic Data All methods, including the central 1-class model, show high error rates trying to separate outliers from normal data for

[2] `http://www.spread.org`

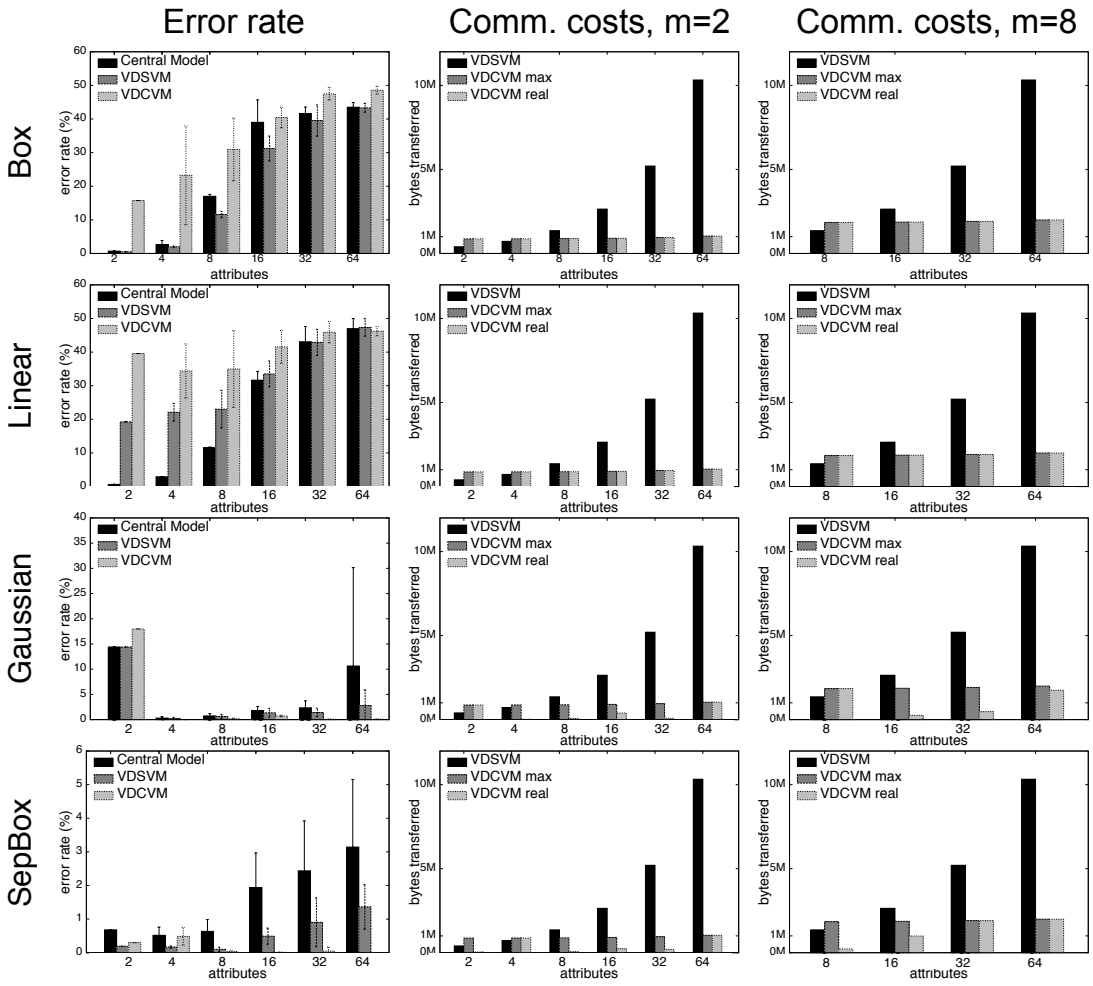


Figure 8.3: Performance of VDCVM, VDSVM and a central model with standard RBF-kernel on the generated datasets. Note that communication costs for central model and VDSVM are the same, due to equal sample size.

the `Linear` and `Box` datasets in higher dimensions. It could not be verified that the VDSVM performs well with higher dimension, as stated in [DBV11]. In low dimensions, the combined RBF-kernel has worse error rates than a standard RBF-kernel and VDSVM's ensemble of local classifiers. However, the error rates of VDSVM are still high compared to those of the central 1-class model. On the datasets whose attribute values provide more information about the label locally, i.e. `Gaussian` and `SepBox`, VDCVM shows similar or even better prediction performance than its competitors. This might be explained by the lower number of support vectors it achieves, due to small core set size. Such less complex models might generalize better than more complex ones. Another

Table 8.3: Results on real world datasets (p: attributes, m: nodes, *err*: error rate in %, *bytes*: amount of bytes transferred).

Dataset	p	m	Central model		VDSVM		VDCVM	
			err	Kbytes	err	Kbytes	err	Kbytes
`letter`	16	2	10.000	54	9.333	54	6.500	**9**
		4	11.000	54	9.333	54	10.500	**16**
`kddcup-99`	127	2	0.285	20,401	0.220	20,401	0.000	**1,206**
		4	0.450	20,401	0.290	20,401	0.002	**1,526**
`face`	361	2	6.220	29,006	7.900	29,006	4.940	**808**
		4	5.580	29,006	6.880	29,006	5.100	**969**

explanation could be that better hyperparameters were found during tuning.

Communication Costs on Synthetic Data With growing dimension, VDCVM becomes more and more communication-efficient than sending the full sample to a central node for analysis. For the largest dimension $p = 64$ and $m = 2$, it communicates about 10 times less data than the other methods. For `Gaussian` and `SepBox`, which are easier to separate than the other datasets, VDCVM sampled less observations than maximally allowed in several cases (compare *VDCVM max* to *VDCVM real*).

Performance on Real-World Data All methods achieve similar error rates on the real world datasets (see Table 8.3), with VDCVM sometimes performing a bit better. Again, this might be explained by the less complex models it produces, or by better hyperparameters found with the random tuning strategy. VDCVM transmits less data than its competitors. For instance, in case of the `face` dataset vertically partitioned over four nodes, the payload transmitted by VDCVM is 30 times smaller. Note that the `face` dataset has already been reduced from 513,455 observations to 20,000, such that LibSVM could handle it properly. Sending the whole dataset would result in a payload of about 186 MB, considering that grey scale values can be encoded in a single byte. With four nodes, VDCVM achieves an error rate as low as 5.1%, but communicates about 191 times less data between local nodes and the central coordinator as if the whole face dataset would have been centralized for analysis.

Number of Iterations vs. Communication Costs The plots in Figure 8.4 show how the number of transmitted bytes grows with the number of iterations, for a fixed number of features, and from when on VDCVM becomes less communication-efficient than VDSVM. Note that communication costs are shown on a log scale. As shown in the left figure for 10 features, the crossover occurs at 1,000 iterations (corresponding to about 59,100 observations). The right figure plots the transmitted bytes for 500 features. Here, the VDCVM is at least an order of magnitude more communication-efficient for

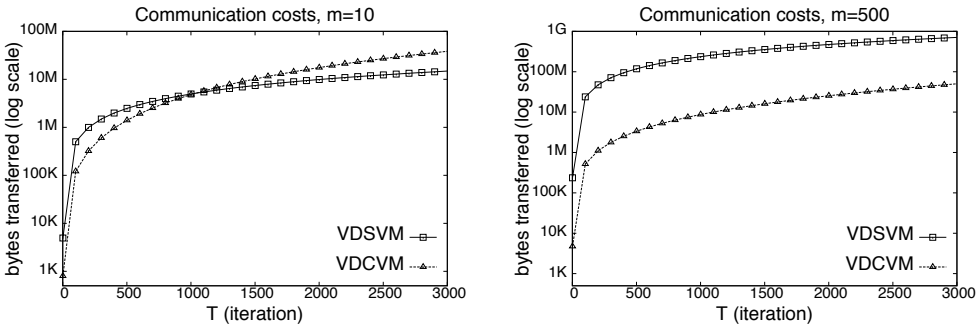


Figure 8.4: Bytes transferred (log scale) by VDSVM and VDCVM with a growing number of sample size and iterations (T), for 10 (left) and 500 (right) attributes, $s = 59$, $m = 1$.

all plotted iterations of the algorithm. In general, the more attributes are stored at each local node, the more can be saved in comparison to centralizing all data.

8.5 Summary and Conclusions

The VDSVM, a distributed version of the 1-class ν-SVM for vertically partitioned data, has two drawbacks. The first is that it might miss global outliers which are not also local outliers in at least one dimension. Here, the problem is that local models are trained on different subspaces of the whole data matrix, which may result in largely differing local models in terms of support vectors. The second drawback is that the size of a sample the VDSVM transmits to a central coordinator node must be specified by the user. Specification of the size in advance is difficult, since sample complexity depends on the underlying data distribution, which is unknown. The approach proposed in this chapter is to replace the central model by a CVM model, which samples only as many observations as needed to reach a $(1+\varepsilon)$-approximation of the minimum enclosing ball (MEB). Further, it has been proposed to replace the training of local models by a distributed furthest point calculation over the local nodes. Since there exists just a single model managed at the coordinator, there exists also only one set of support vectors.

As has been demonstrated, a communication-efficient distributed computation of the furthest point cannot be achieved for arbitrary kernel functions. The linear kernel is not normalized, and therefore cannot be used with the CVM. The RBF-Kernel, which is usually used with the CVM, leads to an expression for the furthest point calculation which is not separable over local nodes. It would lead to communication costs that are asymptotically quadratic in the number of observations n. Therefore, it has been proposed to use a combination of RBF-Kernels. This allows for the development of a more communication-efficient distributed algorithm, called the Vertically Partitioned Core Vector Machine (VDCVM).

Analysis of VDCVM's communication costs shows that they are asymptotically quadratic in the number of iterations T, where $T \ll n$. Hence, the VDCVM with a combination of RBF-Kernels is more communication-efficient than using the standard RBF-Kernel. Nevertheless, due to the transmission of updated α-values in each iteration, there is a break even point where the VDCVM becomes less efficient than sending individual observations instead of partial sums. However, as long as the number of features stored per node is high enough, communication costs grow only slowly with iterations. Therefore, in many cases, the maximum number of observations the VDCVM can handle in a communication-efficient manner corresponds to large datasets of about a million observations. As demonstrated in the experiments, the number of observations sampled in practice can be even much lower than this theoretical maximum. For instance, on the extended MIT face dataset, the VDCVM reaches an error rate as low as 5.1%, but communicates about 191 times less data than needed when centralizing all data for analysis.

In all experiments conducted, the VDCVM is more communication-efficient than its direct competitors, the central 1-class ν-SVM model and the VDSVM, except for very low numbers of attributes. In many cases, the savings in communication do not come at the expense of accuracy. To the contrary, there are some cases where the VDCVM performs even slightly better than the other tested algorithms. This might be either explained by CVM's models being less complex, or a better tuning of hyperparameters. Nevertheless, there are some cases where the VDCVM performs worse in terms of accuracy, namely on the synthetic datasets which contain conditionally dependent features, given the label. As it seems, the combined RBF-Kernel cannot capture such dependencies. However, it should be noted that at least in higher dimensions, no method performs well, even not the 1-class ν-SVM with standard RBF-kernel.

Finally, some comments should be made on the suitability of the VDCVM for highly resource-constrained settings, like pervasive distributed systems. Continuous transmission of data may consume lots of energy, while the connection must be kept up. With the VDCVM, depending on application, the local nodes could send their partial sums infrequently in short bursts, whenever they have gathered s sample points. In between, the sensors would not need to stay connected to the central node, saving resources. Once s points are gathered, sensors could forget about such points, keeping only a single point which belongs to the core set. Though the algorithm's operation is not exactly that of a streaming algorithm, since the memory consumption is not constant, it comes close to (infrequent) streaming. The memory footprint on all nodes is relatively low at least, considering that only the core set needs to be stored, which is usually much smaller than the total number of observations. In terms of CPU and battery power, local nodes need not to be powerful, since computations only consist of kernel calculations. The central coordinator node, however, needs more resources, since solving the central QP problem can be very demanding. In a wireless sensor network with hierarchical topology, maybe it could be operated on cluster heads, which are more powerful than sensor nodes. Following ideas from sensor node clustering (see Sect. 4.2), the coordinator could be rotated over cluster heads, for fairer distribution of energy usage. Due to the small size of the

core set, it could be transferred from one cluster head to the next. Prediction after training is also communication-efficient, since only partial sums need to be sent to the central coordinator. These are single scalars per node and observation, and independent of the number of dimensions.

Based on aforementioned considerations, the VDCVM may have good chances to be applicable in resource-constrained pervasive environments, in contrast to other support vector machine algorithms developed for the vertically partitioned data scenario (see Sect. 4.4). Especially, such algorithms have either high communication costs, like the privacy-preserving SVMs, or they would need to solve QP problems directly on sensor nodes, like consensus algorithms or the VDSVM. They are thus much more demanding than the VDCVM, which uses only kernel calculations at local nodes, which could be sensors. In fact, the VDCVM has already been run successfully fully distributed on three Raspberry Pis connected wirelessly, with one Raspberry Pi being the central coordinator node and the other two devices being the local nodes.

Chapter 9

Summary, Conclusions and Questions

The following subsections first summarize this thesis, then draw conclusions and finally give an outlook on future research opportunities.

9.1 Summary

Opportunities for Data Analysis in the Internet Of Things This thesis started with giving an overview of the IoT and its many applications. It stressed several kinds of applications which can only be enabled with the help of data analysis. Here, highly sophisticated applications would embed data analysis fully into an IoT system, and integrate analysis as much as possible with control. In this way, it would be possible to create fully closed optimization loops, where data analysis builds models based on what's really there, and gives control the necessary information to adapt parameters of running processes to the changing circumstances, enabling the system as a whole to reach its goals. As discussed, being able to adapt to a changing environment in real-time could lead to optimized systems which are much more sustainable than current ones, reducing waste and saving valuable resources like energy.

Cloud Computing vs. Communication-constrained Settings Applications in the IoT are of a highly distributed nature. Currently, there exist mainly two different types of distributed systems. The first type of systems are data centers following the paradigm of parallel high-performance computing. The second type are pervasive distributed systems, which consist of small devices connected in a wireless network. Such systems are much more communication-constrained than devices operating in a data center, where network technology achieves bandwidths comparable to main memory accesses. For battery-powered devices, transmitting data is one of the most expensive operations in terms of energy, and wireless connections today have a much lower band-

width than local area networks. However, pervasive distributed systems, like wireless sensor networks, are not the only kind of systems which suffer from severe communication constraints. Another example are high throughput applications which require the real-time analysis of continuous streaming data, like Formula One racing, or those where data masses are so huge that they cannot be transmitted over existing communication lines, like applications in physics or earth sciences.

The Horizontally Partitioned Data Scenario Today, there exist many distributed algorithms for the horizontally partitioned data scenario, where subsets of observations are distributed over the nodes. There are some algorithms which are communication-efficient, like consensus algorithms, which for instance solve the primal SVM problem by an iterative exchange of weights between nodes. The number of weights exchanged is usually much smaller than the number of observations. Another communication-efficient algorithm is the least squares regression SVM, which transmits only $O(p^2)$ for the linear kernel. Admittedly, communication-efficient solutions for non-linear kernels seem to be much harder to obtain. But there exist other communication-efficient distributed algorithms, like for decision tree induction or clustering. Some of such distributed algorithms even take the severe resource-constraints of wireless sensor networks into account.

The Vertically Partitioned Data Scenario The vertically partitioned data scenario, which has many relevant applications in an IoT context, seems to be more challenging. Here, not observations are distributed over nodes, but features of observations. Not many communication-efficient algorithms for the scenario exist. For instance, privacy-preserving SVMs have quadratic communication costs, and consensus algorithms require the iterative transmission of predictions for all observations. Depending on the number of iterations, they might therefore transmit more data than the original dataset. As we have seen, the biggest challenge of the scenario are conditional dependencies between the features from different nodes, given a target value to be predicted. Respecting such conditional dependencies would require to look at *combinations* of features from different nodes, potentially leading to high communication costs. We arrived at the main problem of this thesis, namely how to realize communication-efficient distributed learning algorithms for the vertically partitioned data scenario.

A Hot Rolling Mill Case Study The vertically partitioned data scenario has been motivated with a case study from smart manufacturing. In a hot rolling mill, data about the processing of steel blocks is assessed by different kinds of sensors attached along the process chain. The quality of rods finally cut from the steel blocks should be predicted as early as possible, in real-time, such that unnecessary processing could be spared and resources be saved. Data about a single observation, which is the processing of a single steel block, are sets of value series partitioned over different physical machines (processing stations). The tighter the feedback loop between prediction and control becomes, the less time will be available for the preprocessing of value series and making predictions.

Apart from the data being vertically partitioned, the real-time constraints may lead to questions which are typical also for the distributed learning from vertically partitioned data. Which data should be processed locally, to match the real-time constraints, and which data must be sent to a central node, preserving accuracy? Which data can be sent to the next station in the process chain?

The preprocessing of value series poses a problem in its own right, not only in the context of production processes, but for IoT generated data in general, which is usually time-related. As discussed, finding a good representation of value series in propositional form for learning, which is the problem of feature extraction, is a hard task. Only few automated methods for the feature extraction from value series exist, and those that do exist, like method trees, assume to be given a collection of single one-dimensional value series of the same type. In comparison, observations in production processes are *sets* of highly heterogenous value series. Given the heterogenous nature of IoT devices, the value series in our case study may come close to the kind of data which will be received also in other IoT applications. Each series has its own requirements for data preparation, like cleansing, imputation, alignment, normalization, and segmentation. In the context of our case study, such processes have been implemented in RapidMiner. They are part of a proposed algorithm for the preprocessing of value series in production settings, which, except for the domain-specific subroutines, is highly generic and might be also used, with small adaptations, in other production settings or maybe IoT applications.

Features extracted in the context of the case study are simple statistics and numerical values, extracted from whole value series (global), their segments (local) and over the features of segments (aggregates). Thereby value series can be represented at different levels of granularity. All features are gathered in a single vector in propositional form, such that all observations have the same number of attributes. Based on such features, we could successfully identify and quantify operational modes, as verified by the domain experts. Quantification of deviations from targeted processing is new information which has been made available by data analysis. It can be used, for instance, for the automatic monitoring of processes. We also tried to predict the final quality of rods by training different classifiers. As it seems, quality is hard to predict, but not only with these features, but also with many other extracted types of features. It could be that information from the rolling process alone doesn't suffice to predict the final quality of rods sufficiently well. It is therefore planned to combine the data from rolling with data about the previous processing step, which is melting.

Technically, it is difficult to associate data about the melting process with the casting of individual steel blocks. The melting data is therefore aggregated information about different charges of blocks, while we would need data about individual blocks. Similar problems occur with available quality information, where often only the proportions of labels are given for whole charges of steel blocks. Aggregated label information motivates a relatively novel kind of learning task, the problem of learning from label proportions.

Learning from Label Proportions by Clustering (LLPC) The problem of learning from label proportions has not only relevance for production processes, but also for privacy-preserving data mining. For instance, we may ask what can be derived from aggregated election results given for districts, when governmental agencies can obtain data about individuals living in these districts. Such questions are, of course, closely related to privacy issues posed by the IoT, whose devices may also collect data about individuals.

A new algorithm for the task has been proposed, the Learning from Label Proportions by Clustering (LLPC) algorithm. The algorithm's performance is compared to three other state-of-the-art approaches, in terms of accuracy and running time. The algorithm's accuracy is similar to the accuracy of its competitors, or significantly higher in the case of larger bag sizes, where learning is more difficult. At the same time, LLPC's asymptotic running time is only linear, while the running time of its competitors is at least quadratic.

The proposed algorithm comes with many other benefits. It is easy to understand and implement. It can handle multiple classes, and labeled observations, if they are given. It can handle many different data distributions, by a simple exchange of inner clustering algorithm. In fact, the algorithm has already been used successfully with many different clustering algorithms, like k-Means, EM clustering, Kernel k-Means, density-based clustering, Support Vector Clustering, and projectional clustering. Similarly, the labeling strategy can be exchanged, as can be the loss function over the label proportions. The evolutionary strategy for the optimization of attribute weights should be easy to exchange as well. The algorithm follows ideas which at the time of its publication were novel and not well-explored, but recently have been proven by other authors to be valid ideas from a view point of learning theory. For instance, it has been recently proven that the ability to predict bag proportions well depends on the capacity of the used hypothesis class. In LLPC, the capacity can be controlled by changing the number of clusters, and matching the label proportions well can bound the probability of classifying individual instances incorrectly, according to theory. Further, the proposed loss function measures how well the class priors are matched. Recently it has been shown by other authors that whenever the class priors are matched, the error made on individual observations is bounded, in cases where observations, given the bag, are identically independently distributed. Also, the formulated ideas about best and worst case of the scenario are now proven by other authors. Used with k-Means, the algorithm has a small (and constant) memory footprint. Therefore, there is a chance that the algorithm might also run on resource-constrained devices as they are typical for the IoT, at least at the data generating side.

Interestingly, the problem of learning from label proportions is closely related to the communication-efficient transmission of labels in the vertically partitioned data scenario, which has been a largely neglected problem so far. The idea of reducing communication by sending label counts is realized in the algorithm that has been proposed next.

Training of Local Models from Label Counts (TLMC) In the context of the IoT, more and more cities are getting instrumented with sensors. Such smart cities may save resources based on information obtained from different kinds of monitoring applications. One important application is the prediction of traffic flow, which promises large savings in terms of fuel if traffic jams could be predicted correctly. Current traffic prediction systems are highly centralized. This poses risks in case of disasters, since centralized systems can easily become single points of failure. Moreover, the maintenance of hard-wired sensors is costly, due to related construction work. Also, central systems pose a bottleneck in terms of bandwidth. A decentralized system like the one we propose, consisting of cheap battery-powered presence sensors that might be easily attached to existing infrastructure, could be much more fault-tolerant, and easier to maintain. As we have seen, decentralized systems consisting of battery-powered sensors pose challenges for data analysis in their own right, especially if communication is constraint by the use of wireless network technology.

For the scenario proposed has been a decentralized in-network classification algorithm, the Training of Local Models from (Label) Counts (TLMC). The method reduces communication by only transferring aggregated label information between nodes in local neighborhoods of topologically close sensors. It has been discussed that exchanging only labels between nodes can be much more communication-efficient than the transmission of measurements. Thereby communication costs become independent of the numbers of features stored at each node. Even more communication-efficient is the transmission of aggregated label information, in most cases. Hence, it has been proposed to send only the counts of labels for whole batches of observations. At each local node, TLMC transforms such label counts into proportions and learns from them with the previously introduced approach for learning from label proportions.

Feasibility of the approach has been demonstrated by evaluating the algorithm's performance in the application context of traffic flow prediction. It is shown that TLMC is much more communication-efficient than centralization of all data, or sending all labels, but that accuracy can nevertheless compete with that of a centrally trained global STRF model. This is a bit surprising, since TLMC does not respect conditional dependencies between nodes, given the label. Such dependencies certainly exist in a densely connected street network. However, as it seems, the nodes nevertheless provide enough information on their own, such that conditional dependencies involving combinations of features may be ignored.

The Vertically Distributed Core Vector Machine (VDCVM) Anomaly detection is an important analysis task in many fields of application, like smart production processes, logistics, physics, earth sciences, disaster management, to name a few. In the case of production processes, anomalies or outliers can be interesting patterns that are not to be discarded, but need to be further analyzed and modeled. For instance, in production processes like the hot rolling mill process in our case study, patterns which deviate much from the usual processing may be highly correlated with a low quality of

final products. Quality deviations in production processes occur only seldom, which is the reason why the class distribution in such settings is highly imbalanced. A popular classifier for learning from such imbalanced data is the 1-class ν-SVM.

For the vertically partitioned data scenario, there exists a communication-efficient anomaly detection algorithm based on the 1-class ν-SVM. The algorithm comes with two disadvantages. The first is that it might not detect global outliers which are no local outliers in at least one dimension. The second is that the size of a sample transmitted to a central coordinator node has to be user-specified. Specifying the size of the sample in advance is difficult, since the number of observations needed to learn depends heavily on the underlying data distribution, which is unknown.

In the algorithm proposed in this thesis, the Vertically Distributed Core Vector Machine (VDCVM), the 1-class ν-SVM at the central node is replaced by the Core Vector Machine (CVM). The CVM with probabilistic speedup strategy samples only as many observations as needed to reach a $(1 + \varepsilon)$-approximation of the minimum enclosing ball (MEB) around all observations, with high probability. Sampling all feature values of observations, however, can be avoided by distributing CVM's furthest point calculation across nodes. As has been shown, this makes only sense with kernels which make the furthest point calculation separable over nodes. The standard RBF-kernel should not be used, since it would lead to quadratic communication costs in the number of observations. Therefore, it has been proposed to use a combination of RBF-kernels. Using such combination leads to communication costs which are quadratic in the size of the core set, since updated α values need to be transmitted for each core set point. The size of the core set, however, grows only slowly by a single observation in each iteration. It has been shown that the number of observations which can be analyzed, until the algorithm becomes less communication-efficient than transmitting all feature values per observation, is large enough for many applications.

In experiments it has been shown empirically that the VDCVM communicates up to an order of magnitude less data during learning, in comparison to the previously mentioned distributed state-of-the-art approach, or training a global model 1-class model by the centralization of all data. Nevertheless, in many relevant cases, the VDCVM achieves similar or even higher accuracy on several controlled and benchmark datasets. The only disadvantage of the VDCVM might be that it cannot be used with the standard RBF-kernel, which would lead to quadratic communication costs in the number of observations sampled. Instead, the VDCVM uses a combined RBF-kernel, which seems to have difficulties to capture conditional dependencies between the features of different nodes, given the label. However, it should be noted that at least in higher dimensions, also the global model of the 1-class ν-SVM with a standard RBF-kernel didn't perform well on the same controlled datasets.

Only the central coordinator node has to solve a QP problem, which is resource-demanding, while the local nodes execute simple kernel calculations. We therefore think that the VDCVM might be the first SVM algorithm for the vertically partitioned data scenario which has a chance to be realized in a truly resource-constrained setting, like wireless sensor networks. The algorithm has already been successfully run in a network

of Raspberry Pis, which are small devices, and whose resources are comparable to those of modern mobile phones. Therefore, the algorithm should be able to run on mobile phones at least, whose most expensive operation, in terms of energy, is the transmission of data, which the VDCVM reduces.

9.2 Conclusions

In this section, we shortly return to the research questions formulated at the beginning of this thesis, which have mainly driven the development of distributed algorithms for the vertically partitioned data scenario. We first take an algorithmic perspective, afterwards discuss implications of results presented for the IoT, and then come to applications which might be enabled by the developed algorithms.

Algorithmic Perspective The first questions we wanted to answer was how the learning task influences communication costs when the data is vertically partitioned, and how the design of algorithms changes with learning task. By reviewing related work, discussing the challenges of the vertically partitioned data scenario from a learning perspective, and designing two distributed algorithms for different learning tasks, the answer which might be given is the following. Communication costs do not depend so much on task, but rather on conditional dependencies of features from different nodes, given the label. These dependencies underly all learning tasks, and are therefore a general problem of learning in the scenario. Experiments on controlled datasets which simulate the difficult cases show that the problem really exists and may lead to a complete failure of algorithms which do not respect such dependencies. The distributed 1-class ν-SVM (VD**S**VM) has high error on one of the datasets, although it checks local outliers against a global model. The experiments indicate that the algorithm might miss global outliers which are no local outliers in at least one dimension. In case of the VDCVM, the combined RBF-kernel seems to have problems with capturing such dependencies. However, in experiments on the datasets from real domains, the problem did not occur. Especially, TLMC, which ignores conditional dependencies between features of different nodes, performed well on a densely connected street network, where such dependencies should exist. The phenomenon might be compared to the success of the Naïve Bayes classifier, which makes the same naïve assumption about the data, but nevertheless performs well in many relevant cases.

The second question was how data distribution affects communication costs and the accuracy of algorithms, and how the design of algorithms could change depending on distribution. The problem of making a naïve assumption has been discussed above already. Therefore, there should be made a few remarks on VDCVM's number of iterations. It could be observed that on datasets in which the outliers are easy to separate from the normal observations, the number of iterations, and therefore core set points and support vectors, was much lower in comparison to the total number of observations. On the difficult controlled datasets, where separation is hard to achieve, the VDCVM sampled

many more observations. This behavior can be easily explained by the fact that highly irregular decision borders, as they are given in the non-separable case, usually need more support vectors for their description. It should be noted that the same behavior can also occur when the data is easy to separate, but the decision border is highly non-linear. In general, non-linearities pose problems for learning in the vertically partitioned data scenario, since they involve the necessity to combine features from different nodes. For instance, use of a non-linear kernel would make the furthest point calculation of the VDCVM non-separable over the nodes, leading to quadratic communication costs in the number of observations. From the viewpoint of data distribution, highly non-linear decision borders may easily lead to situations in which observations can no longer be separated from each other with small error by looking only at single dimensions. In a single dimension, non-linear regions of normal observations and outliers can have high overlap, making separation difficult.

The third question concerned bounds on communication, and how much information must be transmitted to learn successfully from vertically partitioned data. Though no bounds have been proven in this thesis, some remarks can be made, based on the empirical results obtained and the insights gained on the problem. If the data contains conditional dependencies between features, given the class, it becomes necessary to transmit feature values of different nodes. In principle, it would suffice to transmit only those features which are dependent. Unfortunately, which dependencies exist is often not known before learning. In the worst case, *all* features might be conditionally dependent on each other, given the label. Then we have to transmit all of them, for as many observations as needed. For the number of observations to be sampled, there exist the standard bounds from learning theory. When there are no conditional dependencies, we can transmit much less data to learn successfully, as shown by the experiments conducted. Especially, with TLMC, we could obtain sufficient accuracy, in comparison to a global model, by transmitting the labels of all observations for the training of local models only once. In comparison, existing consensus methods would send predictions for all observations in several iterations until convergence. The VDCVM transmits the scalar value of a partial sum for each observation and node, and updated α values and feature values for each core set point. As long as the core set stays small, only a small number of observations are analyzed and sampled in relation to the total number of given observations. In such cases, the VDCVM transmits much less data in comparison to existing methods, which send a single value for *each* observation, maybe even repeatedly. At the same time, as experimental results evaluating the VDCVM have shown, there was no trade-off to be made concerning accuracy.

The fourth question was how the supervised learning of local models can be made more communication-efficient in cases where labels do not reside on the local nodes. Proposed has been to transmit only aggregated label information, in the form of label counts, and use methods for learning from label proportions at each local node. The feasibility of the idea has been shown by evaluating TLMC successfully in the application context of traffic flow prediction. Further experiments on other domains would need to be conducted to show the generality of the approach empirically. However, the successful

evaluation of LLPC on several standard datasets and the performance of other methods for learning from label proportions show that learning from aggregated label information can be successful also in the context of other domains.

The fifth and final question was how distributed data analysis algorithms in the vertically partitioned data scenario might handle the addition and removal of sensors. In turn, this would lead to numbers of features changing dynamically during learning, and prediction, while standard methods expect a fixed number of features. Based on our discussion of TLMC one might say that the training of local models and combining their predictions by a fusion rule should be highly robust against such changes. During learning, local models trained for failing sensors might be simply discarded. They can be trained again when sensors reenter the network. During prediction, if only a few sensors fail, it may happen that the fusion rule combines a fewer number of predictions. This might affect accuracy badly when the models of failing sensors are well-performing models. Nevertheless, making a prediction would be still possible at least.

There were other questions posed. These concerned research questions which are still open and which will be discussed in Sect. 9.3.

Implications for the IoT In the chapter about the IoT, the difference between distributed high performance computing and distributed pervasive systems has been discussed. It has been argued that there exist highly communication-constrained scenarios, not only for pervasive systems, which don't allow for the centralization of all data in the cloud. The results presented in this thesis seem highly promising for such scenarios. For settings in which conditional dependencies between features don't exist or play only a marginal role for estimation, TLMC's local models can be trained in very communication-efficient ways, with sufficient accuracy. As shown using k-NN, for higher accuracy, the labels of all observations may be transmitted, still yielding communication costs which are independent of the number of features stored at each node. VDCVM is communication-efficient for large numbers of observations, with realistic choices for the number of nodes and features. Using any of the presented methods, there were cases where communication costs could be reduced by an order of magnitude in comparison to centralizing all data. There are counter examples. On the difficult synthetic datasets, VDCVM's accuracy suffered, which means that there might be also real-world scenarios where communication-efficient learning won't work.

In situations where we are not as communication-constrained, and have the choice between centralizing all data and more local processing, it depends. If we expect many conditional dependencies between features, we might opt for reducing risk and centralize all data. However, as has been shown, in many settings not much data needs to be communicated to obtain a high prediction accuracy.

The previous observation concerns a hot topic in data mining today, which is big data. In fact, big data is expected to be especially generated by the IoT. It is discussed very much how the big data masses produced can be handled by data analysis. As experimental results of the proposed methods suggest, data analysis algorithms don't

necessarily need to be run in a data center. They could as well operate directly on small devices, which generate the data, or on slightly more powerful intermediate nodes, reducing the amount of data which needs to be transmitted to a central location. This suggests that the big data problem might be handled much better by decentralization, instead of centralization. Especially, the variety of big data, which means the heterogeneity of its data sources, might be handled much better locally. As experiences with our case study show, heterogenous value series require highly individual processing. Manufacturers of IoT devices or the machines used in production know their devices best. If they would ensure that the data coming from their sensors and machines is already in appropriate format for learning, i.e. cleansed, aligned, and segmented, for instance, the intensive work of data preparation could be spared.

Nevertheless, even with properly prepared data, there are still many open questions left concerning data analysis in decentralized systems, presented in Sect. 9.3.

Application Perspective As already pointed out at the beginning of this thesis, and throughout the text, there exist many potential applications for communication-efficient distributed algorithms. Examples reach from telescopes producing huge data masses in physics, over earth sciences transmitting image data for analysis over satellite connections, to high-throughput applications like Formula One racing and decentralized traffic prediction systems. The smart manufacturing case study presented in this thesis focuses on a hot rolling mill process, where predictions need to be made under real-time constraints.

As consultancy during data acquisition for our smart manufacturing case study has shown, the trend goes to instrumenting more and more devices in production with sensors. The hope is to extract useful information from the data recorded. As has been demonstrated, data analysis can help with the extraction of such information. Through learning from the recorded data of newly attached sensors, for the first time it has become possible to quantify deviations from targeted processing. The learned models could now be used for the automatic and continuous monitoring of such production processes.

Talking to people from the field of manufacturing also shows that the problems we tackle in our case study occur in many different kinds of production processes. The biggest problem seems to be that simulations alone no longer suffice for the planning of production processes, which have become highly individualized. There are cases which cannot be covered by simulations. What's more, existing simulations have a long running time, and cannot be used for real-time prediction or planning. Further, they are not based on what happens, but on what is assumed to happen. Simulations might therefore optimize the behavior of a production system on average, but they cannot account for situations which occur during the running process itself or react to such situations in a timely manner. Data analysis directly embedded into the process chain could therefore help with providing the necessary information to improve on existing simulations, but also with making more timely control decisions. Thereby, waste could be reduced and valuable resources like energy could be saved.

9.3 Open Research Questions

This section first lists open research questions concerning the algorithms presented in this thesis. It then treats questions concerning distributed data analysis as a whole.

Further Research on Algorithms The distributed algorithms presented all learn from batches of data. The data generated from IoT devices, however, will be continuous data. The question is if any of the proposed algorithms could be turned into a streaming version.

The VDCVM already works incrementally, but is no streaming algorithm, because it cannot handle concept drift. In theory, it has a constant memory usage, because the size of the core set is limited by a constant. However, this constant can be too large for practical cases. Can the VDCVM be turned into a full streaming version? As our discussion of the incremental SVMs suggests, at least without making any further assumptions on the data distribution, this is unlikely to work. Throwing away support vectors, which would be necessary to keep memory usage constant, might lead to insufficient accuracy.

Creating a streaming version of TLMC seems to be more promising. It would require to turn LLPC into a streaming version. There already exist streaming clustering algorithms. Most of such algorithms are based on creating summary statistics from clusters, instead of keeping individual observations. In addition to the means and radiuses of microclusters, and the number of observations stored in them, one might perhaps add additional statistics, about the number of observations stored in each bag, per cluster. Whenever a cluster is getting labeled, based on such statistics, it should still be possible to calculate the label proportions. Another interesting question for TLMC is if by dividing the observations into batches more intelligently, even further reductions of communication costs could be achieved.

It would be nice if there was some way to replace the combined RBF-kernel with the standard RBF-kernel when using the VDCVM. Maybe this could be achieved by approximating parts of the non-separable furthest point calculation, thereby making it separable. Further, there exists a generalized CVM for binary classification which can also use linear kernels. It would be interesting to see if the generalized CVM can be turned into a communication-efficient distributed algorithm for binary classification.

It has been indicated that the proposed algorithms have a chance to run on resource-constraint small devices. An interesting question is how the savings in communication translate to savings in energy. To answer that question, the algorithms would need to be implemented on small devices and their energy consumption would need to be measured.

Open Research Questions of Distributed Data Analysis The distributed algorithms which have been proposed are communication-efficient being trained on a given set of features, and with given optimal hyperparameters. Unfortunately, as we have seen in the hot rolling mill case study, the real problem which needs to be tackled is to find a good representation of raw data, like value series, for learning. Also, the performance of algorithms can depend very much on the chosen hyperparameters. For instance, in

case of the VDCVM, wrongly chosen hyperparameters may not allow for easy separation between outliers and normal data, leading to a large number of observations being sampled.

Today, it is rather unclear how to realize the processes of hyperparameter optimization or representation learning, or even only feature selection, in a communication-efficient manner. Existing wrapper approaches for both problems would repeatedly call a distributed algorithm with different hyperparameters and sets of features. Each call would lead to the new transmission of data between nodes.

The question is if knowledge about the model could help with reducing communication, at least in the case of hyperparameter optimization. For instance, changing the value of γ in the RBF-kernel in some cases may lead only to slight deviations in the set of support vectors. If we would know which support vectors will change, or could estimate that at least, maybe we could spare, for instance, the costly transmission of similar α values in the VDCVM.

The problems addressed not only concern hyperparameter optimization and representation learning, but also the evaluation of learning algorithms. Usually, in the aforementioned wrapper approaches, the true error needs to be estimated based on cross-validation or a large enough validation set. Therefore, there are also communication-efficient techniques needed for distributed validation and testing.

Appendices

Appendix A

Programming with RapidMiner

One advantage of RapidMiner over other popular data mining tools, like R and Matlab, is that average users can build simple data mining processes without having to learn a full-fledged programming language. Most power and complexity is hidden in the operators, which are implemented in the Java programming language by experts and either built in or available as additional packages.

Unfortunately, complex data mining processes sometimes require constructs which are usually known from programming, like variables or the conditional or repeated execution of subprocesses. Another need that arises is the reuse of processes. In collaborative work, not all participants are computer scientists or programmers. They have learned a structured programming language perhaps, and can write a script, if at all, but working with a full-fledged Integrated Development Environment (IDE) like Eclipse, as is almost required today for complex object-oriented languages like Java, might be entirely new to them and involves a steep learning curve. Since they may have *some* knowledge about programming, but don't necessarily want to write RapidMiner plugins in the Java programming language, it could be very helpful for them to see how what they already know maps to RapidMiner operators and other constructs.

The following sections present how the abstraction mechanisms of structured programming languages, like the subroutine concept, can be mapped to and realized by combinations of existing RapidMiner operators. It is then discussed how to build, based on such mappings, highly modular and maintainable data mining processes and libraries with RapidMiner.

A.1 Java Operators vs. RapidMiner Processes

In RapidMiner, an operator encapsulates functionality that is generic enough to be used more than once, e.g. a machine learning algorithm. Operators take inputs and deliver outputs over so called *ports*. In a graphical user interface, the output ports of operators can be connected to the input ports of other operators, thereby specifying an execution

order and data flow. The parameters of an operator, which are usually basic data types, provide additional information on the processing.

Implementing often used building blocks in a compiled language such as Java has the advantage that execution can be faster than the repeated interpretation of RapidMiner processes consisting of low level operators. However, the creation of an extension consisting of operators written in Java might not always be the most appropriate way for building reusable modules:

1. Creating a RapidMiner extension not only requires knowledge of (structured) programming, but also of fairly advanced concepts and technologies, like object oriented programming, integrated development environments, build tools, version control systems and XML. When a user already knows RapidMiner and structured programming, but not about the other mentioned technologies, having to learn them may seem disproportionate if the number of subroutines to be implemented is only small.

2. In comparison to the more direct change and test cycles of interpreted languages, the additional building steps of compiled languages such as Java may lead to an increased development time. Even though it is no longer necessary to restart RapidMiner after code changes, the code of an extension every time needs to be compiled and built using Ant. In cases where the focus is more on rapid prototyping than speed of execution, being able to change and run a RapidMiner process without any additional steps can thus be beneficial.

3. Once data mining processes are becoming themselves more complex, the question arises how common patterns can be factored out and reused. Such patterns exist on the process and not on the operator level. Therefore, mechanisms are needed which allow for the encapsulation of common patterns in parameterized subprocesses, not operators. Similarly, complex data mining processes may require control structures like loops and branches.

4. In comparison to extensions, processes consisting only of built in RapidMiner operators are more robust against changes of RapidMiner's underlying Java code, like changes in the application programming interface. Moreover, working together on a set of processes, e.g. in one of RapidMiner Analytics' shared repositories, can be easier than distributing updated versions of an extension, since all users share the same view.

The following subsections describe how operator like functionality can be realized on the process level, using exclusively already existing RapidMiner (version 5.3) operators instead of Java.

A.2 Definition of Variables with Basic Data Types

Programming languages usually allow for the definition of named variables that may hold different values of a certain data type. Such variables are often used to store (user) inputs or the intermediate results of calculations. Basic data types are, for instance, boolean values, numbers (integer and real values) and strings. In Java, variables of the aforementioned types would be defined like the following:

```
boolean isActive = True;
int counter = 0;
double sqrt2 = 1.41;
String text = "Hello World!";
```

In RapidMiner, variables having a basic type are called *macros* and can be defined by the Set Macro and Extract Macro operators.

(a) (b)

Figure A.1: The Set Macro operator

With Set Macro, a named variable can be introduced and assigned a constant initial value. The variable's data type is determined by the assigned value. For example, Fig. A.1 demonstrates the parameter settings for introducing the boolean `isActive` variable from the previous Java code example.

(a) (b)

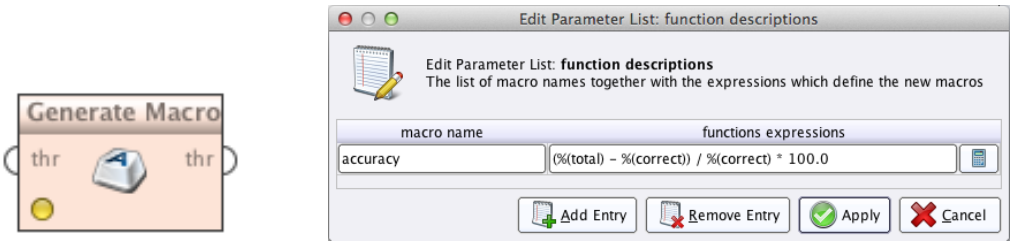
Figure A.2: The Extract Macro operator

With the Extract Macro operator, the initial value is taken from values or meta data of an *ExampleSet*, and can be for instance the number of examples (see Fig. A.2), the mean of a particular column or the value of a particular cell in the data table. The type of the variable is determined by the type of the corresponding data cell.

Often it is necessary to calculate new values from existing variables. The following code shows how the *accuracy* of a classifier could be calculated in Java from the *total* number of examples and the number of *correctly* classified examples:

```
int total = 200;
int correct = 180;
double accuracy = ( (total-correct) / (double)total ) * 100.0;
```

For such calculations, RapidMiner provides the Generate Macro operator. It assigns the result of calculations on previously defined macro values either to a new or already existing macro.



(a) (b)

Figure A.3: The Generate Macro operator

Figure A.3 shows the Generate Macro operator's dialog for the definition of macros based on function expressions. Existing macro values can be inserted into such expressions by putting the macro's name into round brackets, prefixed with a percent sign: `%(...)`. This kind of replacement almost works in any dialog's fields. For example, it is even possible to create macro names based on the values of other macros.

Once defined, macros can be deleted if necessary by the Unset Macro operator.

Variables in programming languages can have either global or local scope. Global scope means that the variable can be accessed and manipulated by all parts of a program. Local scope means that access to the variable is restricted to a certain part of the program, e.g. the subroutine in which it was defined. In RapidMiner, the scope of a macro is defined at run time by the execution order of operators: The macro is visible after the corresponding macro operators are executed. From then on, the macro's scope is global, meaning that it is visible and can be manipulated from all subsequent operators in the process in which it was defined. In processes consisting of many operators, it is thus vital to name macros carefully, since otherwise it may happen that already existing macros are unintentionally overwritten.

A.3 Definition of Variables with Complex Types

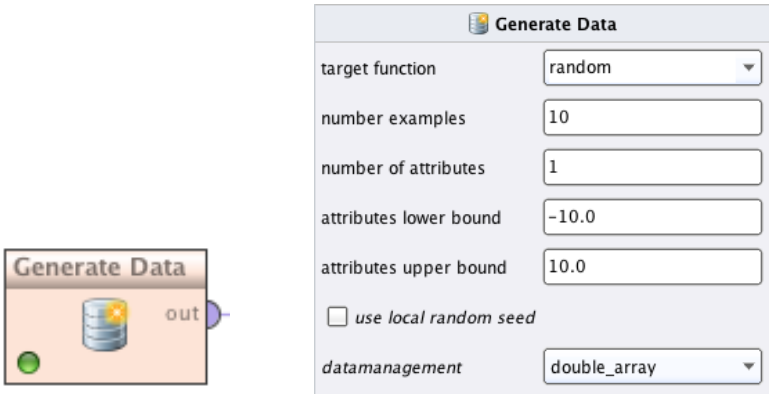
Most programming languages allow for the definition of more complex data types, like *arrays* and *records*.

A.3.1 Arrays

An *array* can be thought of as an enumeration of values with either a basic or complex data type. A particular value can be accessed by its position in the array. For example, the following Java code defines an array of 10 double values, assigns 23.5 to the second array position (indices start with 0 here) and then reads this value into another variable:

```
double[] values = new double[10];
values[1] = 23.5;
double single_value = values[1];
```

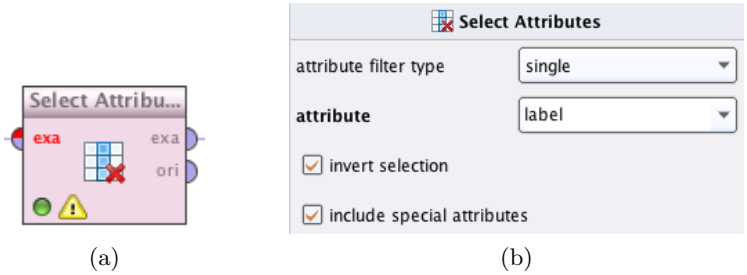
On the process level, complex data types, such as arrays, are not natively supported by RapidMiner. However, the most widely used data structure in RapidMiner, the ExampleSet, can easily be treated like an array, as shown in the following.



(a) (b)

Figure A.4: The Generate Data operator for arrays

First of all, the Generate Data operator can be used to create an ExampleSet which only consists of a single column (see Fig. A.4).



(a) (b)

Figure A.5: The Select Attributes operator for deleting the `label` attribute

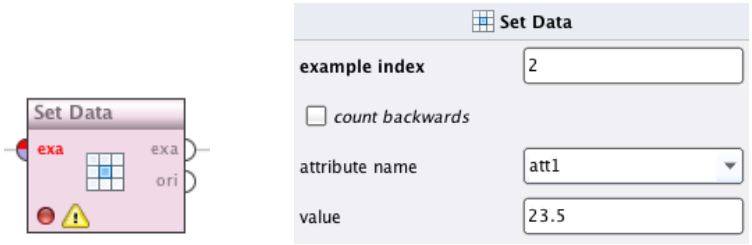
It must be followed at least by the Select Attributes operator with parameter settings as shown in Fig. A.5, because Generate Data also creates a `label` column which is not needed in the given context.

ExampleSet (10 examples, 0 special attributes, 1 regular attribute)

Row No.	att1
1	2.4676120
2	1.2924127
3	-5.589923
4	-6.350916
5	1.2414027
6	-8.767780
7	-5.189688
8	-6.224623
9	-7.153585
10	-4.695913

Figure A.6: Example Set resembling an array

Figure A.6 shows the resulting ExampleSet. Optionally the name of the column `att1` could be changed to another one, like `value`, with the Rename operator. Also the type of the column could be adapted with one of the operators in the Data Transformation →Type Conversion group.



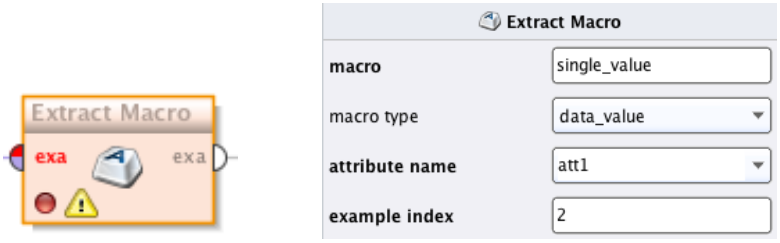
(a) (b)

Figure A.7: Setting a value with the Set Data operator

Cells of an ExampleSet can be assigned values with the Set Data operator, as shown in Fig. A.7, resembling the assignment of values to particular cells of an array. Replacement of macro names by their values also works in this case, allowing for the assignment of previously calculated values to cells of an ExampleSet.

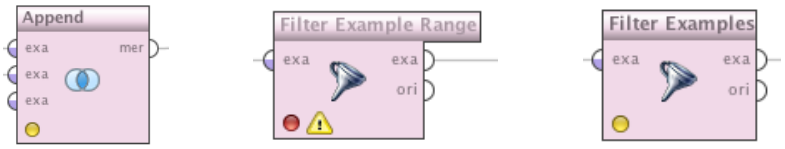
A value in the cell of an ExampleSet can be read and assigned to a macro with the previously mentioned Extract Macro operator. For instance, Fig. A.8 shows how to access the value in the second row of an ExampleSet, resembling the reading of array values at a particular array position.

Since a RapidMiner ExampleSet is a dynamic data structure, already existing ExampleSets can also be manipulated with more powerful operations. The Append operator (Fig. A.9a) concatenates the rows of two or more ExampleSets connected to its input ports



(a) (b)

Figure A.8: Reading an array value with the Extract Macro operator



(a) (b) (c)

Figure A.9: Operators for manipulating the rows of an ExampleSet

and delivers the merged rows as a new ExampleSet on its output port. The Filter Example Range operator (Fig. A.9b) selects a subset of rows, based on the indices of the first and last example. With the Filter Examples operator (Fig. A.9c) it is also possible to select rows based on other criteria, like a condition on an ExampleSet's attribute values.

A.3.2 Associative Arrays

Like an array, an *associative array* holds several elements of basic or complex types, but such elements are accessed by a symbolic key, not their position in the array. Associative arrays are sometimes also called *dictionary*, *map* or *hash table*. Some languages natively support associative arrays with a fixed number of keys already known at compile time, like Pascal and Ada (there called *records*) or C and C++ (there called *structs*). In Java, a hash table with keys of type String and values of arbitrary type can be constructed like the following:

```
HashMap<String,Object> person = new HashMap<String,Object>();
person.put( "name", "Vladimir Vapnik" );
person.put( "age", 61 );
person.put( "affiliation", "Test" );
String name = (String)person.get( "name" );
```

Values can be inserted into a HashMap with put and accessed with the get method.

As with arrays, RapidMiner doesn't provide associative arrays on the process level, but an ExampleSet consisting of only a single row resembles the data structure quite closely. As already demonstrated in the section about arrays, a new ExampleSet with the desired

number of rows (only one) and columns (resembling the keys of an associative array) can be constructed with the Generate Data operator. After construction, the attributes of the ExampleSet can be renamed with the Rename operator and their types may be changed with operators from the Data Transformation →Type Conversion group. Values can be set with the Set Data operator by setting the row number to 1 and providing the name (key) of the column to change. Similarly, values can be read and assigned to a macro with the Extract Macro operator.

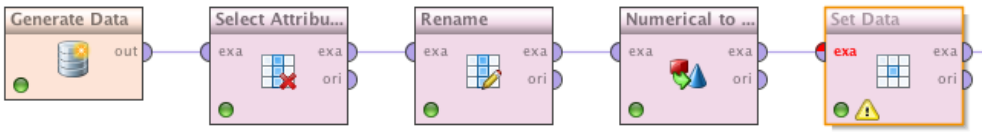


Figure A.10: Process for constructing an associative array

Table A.1: Parameters of the process in Fig. A.10

Operator	Parameter	Value
Generate Data	target function	random
	number examples	1
	number of attributes	3
Select Attributes	attribute filter type	single
	attribute	label
	invert selection	true
	include special attributes	true
Rename	old name	att1
	new name	name
	old name (2)	att2
	new name (2)	age
	old name (3)	att3
	new name (3)	affiliation
Numerical to Polynomial	attribute filter type	subset
	attributes (1)	name
	attributes (2)	affiliation
Set Data	example index	1
	attribute name	name
	value	Vladimir Vapnik
	attribute name (2)	age
	value (2)	60
	attribute name (3)	affiliation
	value (3)	TU Dortmund

Figure A.10 shows a process which constructs the same data structure as the previously shown Java code. The parameters of the operators are shown in Table A.1.

New attributes can be added to an existing ExampleSet with the Generate Empty Attribute (see Fig. A.11a) or Generate Copy (see Fig. A.11b) operators. Moreover, with the Generate Attributes operator (see Fig. A.11c) it is possible to fill the column with values that are

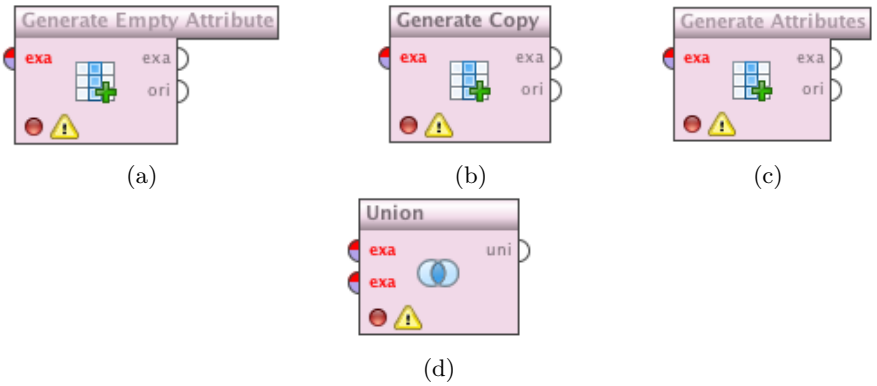


Figure A.11: Operators for manipulating the columns of an ExampleSet

calculated from other attributes in the same row. The Union operator (see Fig. A.11d) further allows to merge the columns of two or more ExampleSets connected to its input ports, delivering a new ExampleSet with the merged columns at its output port.

A.3.3 Storage and Retrieval of Complex Data Types

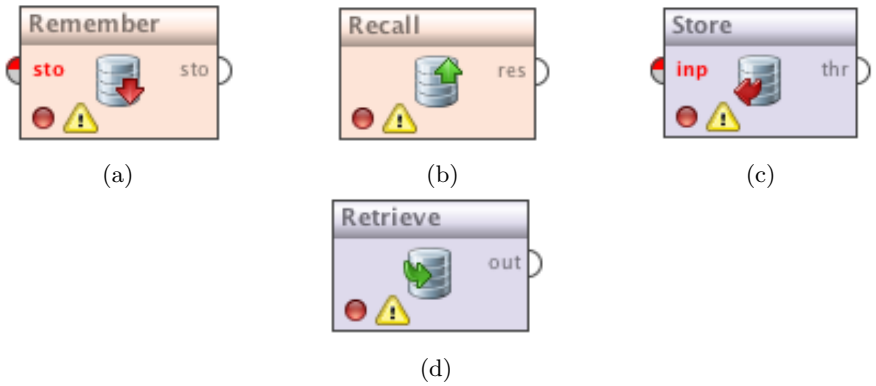


Figure A.12: Operators for storing and retrieving IOObjects

So far it was only shown how complex data structures can be created with RapidMiner, but not how they can be assigned to macros. The fact is that macros can only store values having a basic data type. Instead, complex data structures like ExampleSets, i.e. any object derived from the IOObject class, can be stored in and retrieved from main memory with the Remember (see Fig. A.12a) and Recall (see Fig. A.12b) operators. The operators take a name as a parameter under which the IOObject is to be stored and retrieved. The Recall operator in addition allows for removing the object from main memory. In comparison to a temporary storage of objects in memory, the Store (see

Fig. A.12c) and Retrieve (see Fig. A.12d) operators also allow for a persistent storage and retrieval of IOObjects under a path in a RapidMiner repository.

A.3.4 Recursive Definition of Complex Data Types

A restriction of the ExampleSet in comparison to complex data types in most programming languages is that its data cells can only contain basic types. A recursive definition of complex data structures that contain complex data structures again is not directly supported. However, since IOObjects can be stored and accessed by name, as previously explained, it is possible to store such names as references instead of the original objects in a data cell. All that is needed is a proper convention for a unique naming of objects.

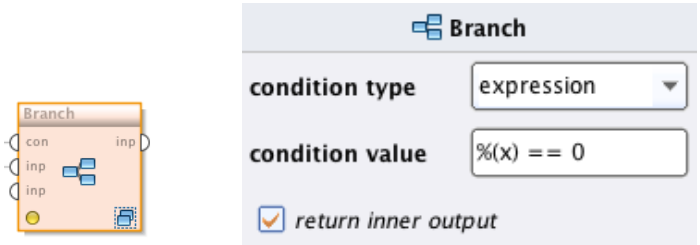
A.4 Control Structures

In addition to the definition of differently typed variables, programming languages usually allow for changing the program flow depending on the current state of such variables. Also RapidMiner contains operators which can test for conditions and change the flow of a process accordingly. In the following, the Branch and Loop operators are shortly explained.

A.4.1 Branching

A well-known construct from programming languages are branch statements like the *if-then-else* statement. For example, the following Java code tests if the value of a variable `x` equals zero. If it does, the code in curly brackets after the `if` is executed, otherwise the code in curly brackets after the `else` is run:

```
if( x == 0 ) {
   System.out.println( "x is zero!" );
} else {
   System.out.println( "x is not zero!" );
}
```



(a) (b)

Figure A.13: The Branch operator

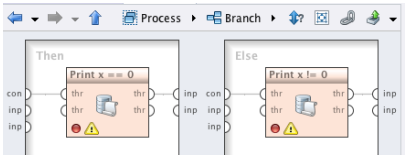
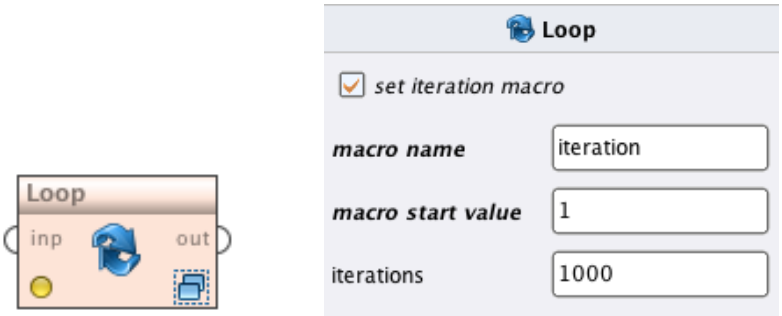


Figure A.14: Subprocess view of the Branch operator

RapidMiner provides the same functionality for processes with the Branch operator. Since expressions for the condition may also contain references to macros, as shown in Fig. A.13, the flow of processes can be changed depending on previously defined macro values. If the expression evaluates to true, the subprocess in the Then box (see Fig. A.14) is executed, otherwise that in the Else box.

A.4.2 Looping

Another important mechanism in programming languages is the so called *loop* which allows for the repeated execution of code. The behavior of the code may change in each iteration of the loop, based on the current state of variables. A condition is checked for determining when a loop should stop.



(a) (b)

Figure A.15: The Loop operator

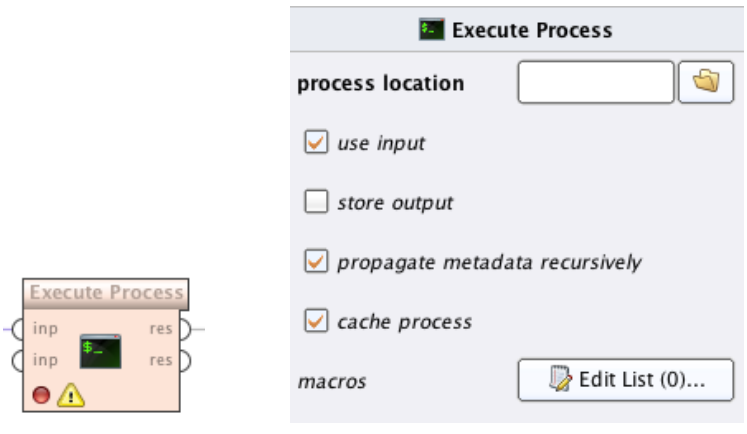
RapidMiner provides the Loop operator (see Fig. A.15) which repeatedly executes the subprocess contained in it for a specified number of times. The number of the current iteration can be automatically assigned to a macro that could then be used, for example, in a Branch operator. Other operators can loop, for instance, over examples or attributes of an ExampleSet, filenames in a directory or until an ExampleSet meets certain criteria, like containing more than a maximum number of attributes.

A.5 Subroutines

Similar to computer programs, complex data mining processes usually consist of two different kinds of parts: Those being too specialized for using them in other contexts and those whose functionality might be generalized and reused also in other processes or projects. In software engineering, the basic dictum that aims at reuse and reducing duplication of information in a program is also known as the *abstraction principle*. Benjamin C. Pierce has formulated this principle in his work "Types and Programming Languages" from 2002 as follows:

> Each significant piece of functionality in a program should be implemented in just one place in the source code. Where similar functions are carried out by distinct pieces of code, it is generally beneficial to combine them into one by *abstracting out* the varying parts.

Keeping the same type of information at a single place has the advantage that code becomes less repetitive, shorter and thereby more maintainable. The most basic mechanism of control abstraction in programming languages is a *function* or *subroutine*. Both may take arguments, also called *parameters*, whose values should be read or modified. Values can have either basic data types or complex types. A subroutine's *signature* consists of its name, the number and order of parameters and (in typed languages) their data types. As long as the signature of a subroutine does not change, errors in its control flow can usually be corrected without affecting any other parts of the code. The subroutine mechanism can therefore be seen as a first step towards more maintainable software systems.



(a) (b)

Figure A.16: The Execute Process operator

RapidMiner's Execute Process operator (see Fig. A.16) comes closest to the concept of calling a subroutine in a programming language. Parameters can be provided in

two different forms: Either as macro values or, for complex types, as IOObjects over the operator's input ports. The process to be run can either be specified as an absolute path or as a relative path to an existing entry in a RapidMiner repository (parameter process location in Fig. A.16). For performance reasons, the process may optionally be cached in main memory, avoiding a time consuming reload in case of repeated executions (e.g. in a loop).

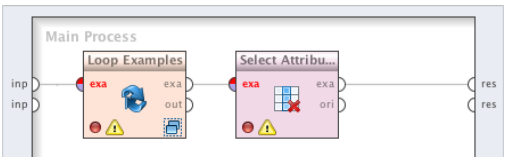


Figure A.17: Process to be called

Operators in the process to be called must connect to the input and output ports of the process (see Fig. A.17) for every input parameter and output that should be returned to the calling process.

A.6 Process Libraries

Programming languages usually have a mechanism for hierarchically grouping subroutines that deal with similar data structures or functionality into so called *modules*, *libraries* or *packages*. For example, in Java, mathematical functions like the sinus function `sin` can be found in class `Math` which resides in the `java.lang` package.

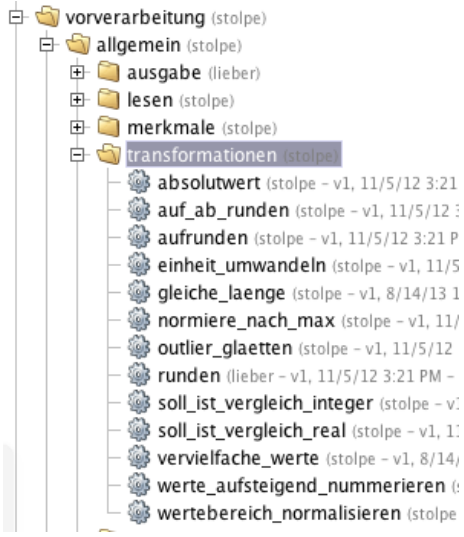


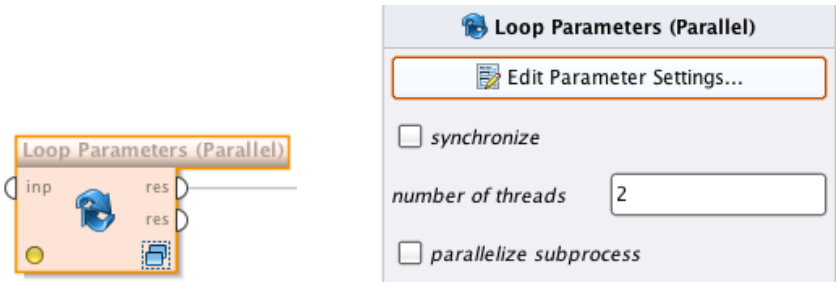
Figure A.18: Process library for time series preprocessing

Since RapidMiner processes that are called from other processes reside in standard RapidMiner repositories, they can also be grouped into a hierarchy of folders and named accordingly. For example, Fig. A.18 shows a folder structure for processes that deal with the preprocessing of time series. Processes can then be either referenced with relative paths from the current process or with absolute paths. Repositories of processes can also be easily shared with others by putting them onto a RapidMiner Analytics server.

A.7 Multithreading

Today, machines that come with more than a single CPU are no longer an exception, but the norm. Modern programming languages therefore must support the utilization of multiple CPUs or cores. A common mechanism for parallel execution is the creation of so called *threads*. Depending on the number of CPUs or cores, statements which are executed in different threads can either run fully in parallel or, if the number of CPUs is smaller than the number of threads, they are interleaved at the machine instruction level. Access to variables that are shared between threads must be synchronized. Modern programming languages usually provide several mechanisms for this synchronization, like locking or semaphores.

RapidMiner does not directly support the concept of threads or any synchronization mechanisms on the process level. However, several of its operators, like cross validation and parameter optimization, support the parallel execution of subprocesses. In the following it is shown how the Loop Parameters (Parallel) operator can be used for achieving a parallel execution of processes together with the previously introduced Execute Process operator. It is important to note that while many RapidMiner operators provide a parameter for parallelization, currently only the operators in the Parallel Processing extension are making use of the parameter and really run their subprocesses in parallel.



(a) (b)

Figure A.19: Loop Parameters

The Loop Parameters (Parallel) operator expects the number of threads as one of its parameters (see Fig. A.19). The subprocess itself does not need to be parallelized. The Edit Parameter Settings dialog (see Fig. A.20) allows for listing parameters that should be

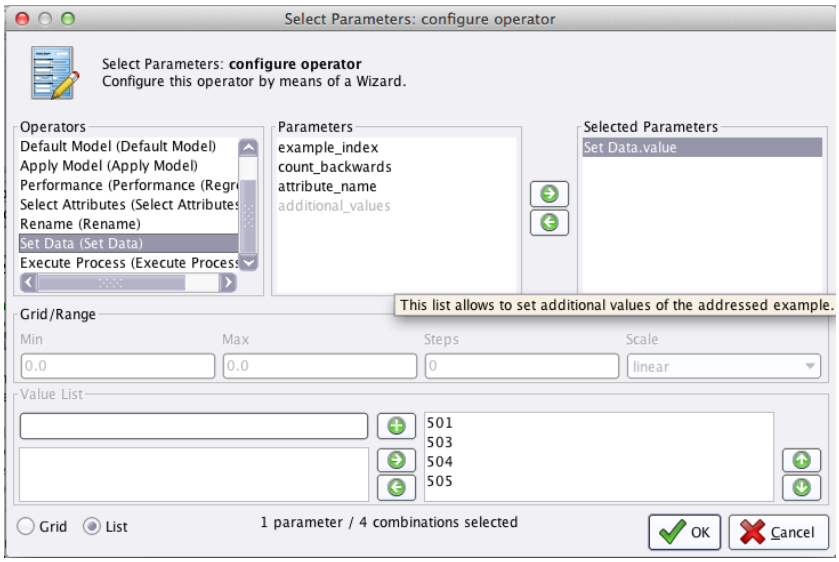


Figure A.20: The Edit Parameter Settings dialog of operator Loop Parameters (Parallel)

set for operators inside the Loop Parameters (Parallel) operator's subprocess. Here, the Set Data operator's parameter value is set to the values 501, 503, 504 and 505.

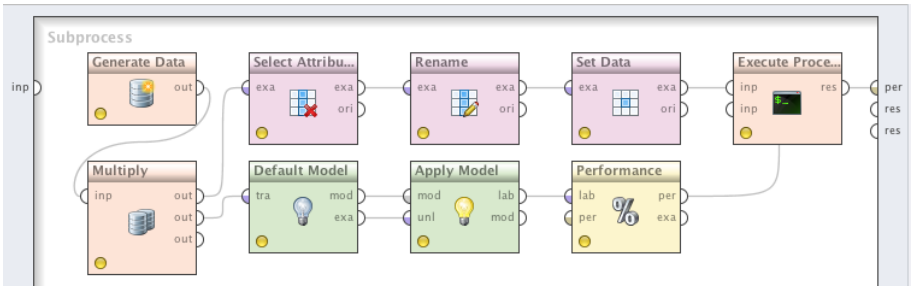


Figure A.21: A generic design for the Loop Parameters (Parallel) subprocess

Figure A.21 shows a generic design for the subprocess inside the Loop Parameters (Parallel) operator. At first, an ExampleSet consisting of only a single row is generated. It is then copied with the Multiply operator. The Select Attributes operator removes the generated label column and the Rename operator renames all attributes beginning with `att` to their proper names. The Set Data operator then assigns values to each attribute. The process thus follows the standard procedure for generating an associative array (for a direct comparison, see Fig. A.10). The associative array is then given to the process called from Execute Process. For each thread/run, the Loop Parameters (Parallel) operator inserts a different value from the list that was specified in its Edit Parameter Settings dialog into the Set Data operator's `value` parameter. This value is then given to the process that is run with Execute Process via the constructed ExampleSet. The question arises why, instead of

having to construct an ExampleSet, the value cannot simply be defined as a macro and then be given to the subprocess to be executed. The reason is that, because a macro's scope is global, access to a macro's value is not thread-safe. Since any of the threads created by the Loop Parameters (Parallel) operator could change the macro's value before the Execute Process operator is even run, its state would be undefined. The ExampleSet that is constructed instead is local to each of the threads, solving the problem of concurrent access. The called process, however, may read any value from the ExampleSet given to it with the Extract Macro operator, since macros can only be seen inside the processes they were defined in.

The operators in the lower part of the process shown in Fig. A.21 are just dummy operators without any specific function. They create an IOObject of type Performance which is delivered over the `per` port, since the Loop Parameters (Parallel) operator expects such an object and RapidMiner would not run the process otherwise.

A.8 Summary

As shown in the previous sections, RapidMiner includes several operators which can be used for achieving similar effects as mechanisms otherwise only known from full-fledged programming languages. In fact, due to the macro and loop operators, any WHILE program could be realized with RapidMiner. Since WHILE programs are known to be turing complete, RapidMiner processes are turing complete, i.e. that any computable function could be realized with RapidMiner's operators. In addition, it has been demonstrated how common elements in large data mining processes can be moved into their own processes, which then may be called with the Execute Process operator. In this way, highly complex data mining processes can be realized in a structured and principled way, only relying on RapidMiner's inbuilt operators. Thereby the steep learning curve of having to learn a full-fledged object-oriented programming language like Java and its tools can be avoided, which is an advantage in collaborations with people who are not computer scientists or programmers. Discussing what we have done with our project partners, who are mostly industrial engineers, on the level of Java code would have been very difficult. Using RapidMiner, however, after some time our project partners were able to implement some of the more specialized data preparation operations pointed to in Sect. 5.4 also on their own, creating new opportunities for more advanced collaboration.

Publications, Joint Work and Collaborations

This chapter gives an overview of publications, and in how far material from them has been included in this thesis. It further states the collaborations and contributions of other authors to this work. Whenever "the author" appears, the author of this thesis is meant.

Included in this Thesis

Distributed Support Vector Machines: An Overview [SBD15] Large parts of this publication have been included in Chap. 4. All descriptions of distributed support vector machines surveyed have been created by the author. Descriptions of the basics of distributed systems are by the author.

Sustainable Industrial Processes by Embedded Real-Time Quality Prediction [SBM16] From this publication, several parts have been included in Chap. 5. All included parts have been written by the author, and for this thesis, new material has been added. The publication [SBM16] and therefore also Chap. 5 include material which has been previously published in "Quality Prediction in Interlinked Manufacturing Processes based on Supervised & Unsupervised Machine Learning" (see below).

Distributed Traffic Flow Prediction with Label Proportions: From in-Network towards High Performance Computation with MPI [LSM15] This publication is an extended version of "Communication-efficient learning of traffic-flow in a network of wireless presence sensors" (see below). Material from [LSM15] has been included and extended in Chap. 7. The idea for the TLMC algorithm and the algorithm itself are by the author, as well as its description, and analysis of communication costs. Experiments and comparisons have been conducted by the author. The data and its description have

been provided by Thomas Liebig. Data has been preprocessed by our student assistant, Jan Czogalla. The analysis of privacy is by Thomas Liebig, as well as the bibliographic references concerning traffic and the extension of the algorithm to MPI.

Communication-efficient learning of traffic-flow in a network of wireless presence sensors [SLM15] Parts of this publication have been included in Chap. 7. For further information, see description above. Thomas Liebig provided the plots for the city map of Dublin.

Quality Prediction in Interlinked Manufacturing Processes based on Supervised & Unsupervised Machine Learning [LSK+13] Material from this publication has been included in Chap. 5. The author had the idea for the generic algorithm for preprocessing value series from production processes. Large parts of the algorithm's implementation in RapidMiner are also by the author. Daniel Lieber has provided all the necessary domain knowledge for the creation of the highly specialized data preparation operations and also created some of such operations on his own. Daniel Lieber also implemented the extraction of aggregated features over segments. Experiments have been conducted and described by the author, including results, the data has been provided by Daniel Lieber.

Distributed Data Mining in Sensor Networks [BS13] Almost all parts of this publication are included in Chap. 4 and have been slightly adapted for presentation in this thesis. The descriptions of all clustering-based approaches are by the author. The descriptions of distributed classification approaches and outlier detection algorithms are by Kanishka Bhaduri.

Anomaly Detection in Vertically Partitioned Data by Distributed Core Vector Machines [SBDM13] Chapter 8 is based on this publication. Most of the material has been included, and slightly modified and extended for this thesis. The bibliography (related work) has been provided by Kanishka Bhaduri and Kamalika Das. Also, they edited the text, mostly language-wise. The idea for the VDCVM algorithm, the analysis of its communication costs, its implementation and the experiments are all by the author.

Separable Approximate Optimization of Support Vector Machines for Distributed Sensing [LSM12] A short summary of the algorithm is given in Sect. 4.4.3. The summary in this thesis has been created by the author. Concerning the publication, the author conducted some experiments. The whole rest of the publication has been created by Sangkyun Lee, who had the idea for the algorithm and its implementation.

Learning from Label Proportions by Optimizing Cluster Model Selection [SM11] Almost all parts of this publication have been included in Chap. 6, and the material has

been heavily extended for this thesis. Especially, the local search strategy has been developed for running the TLMC algorithm, but has not been published so far. The surveys of the Mean Map and LMM/AMM algorithms are based on intensive discussions with Giorgio Patrini, the author of LMM/AMM. He also provided a tutorial manuscript for Mean Map and LMM/AMM, which the description in this thesis loosely follows.

Not Included in this Thesis

Using a Clustering Approach with Evolutionary Optimized Attribute Weights to Form Product Families for Production Leveling [BSDM13] This publication is collaborative work with our project partners. The evolutionary algorithm for optimizing attribute weights has been successfully applied in a slightly different application context from smart manufacturing, namely the clustering of products into families.

Sustainable Interlinked Manufacturing Processes through Real-Time Quality Prediction [LKD+12] In collaboration with our project partners, it is described how production processes can be made more sustainable with the help of data analysis.

Challenges for Data Mining on Sensor Data of Interlinked Processes [SMK+11] In this publication, which has been also created in collaboration with our project partners, the focus is more on the challenges of analyzing data from production processes.

Towards Adjusting Mobile Devices To User's Behavior [FJM+10a, FJM+10b, FJM+11] The work [FJM+10a] had been published at a workshop first, was then resubmitted [FJM+10b], and then extended to a book chapter [FJM+11]. The content of such publications is closely related to the topic of this thesis, namely learning on battery-powered devices like mobile phones. It hasn't been included in this thesis, however, because it doesn't deal directly with distributed learning.

Implementing Hierarchical Heavy Hitters in RapidMiner: Solutions and Open Questions [FS10] A paper about the efficient implementation of the Hierarchical Heavy Hitter algorithm in RapidMiner. Peter Fricke provided the implementation and all experimental results.

Prognosemodelle zur Ermittlung der Produkteigenschaften - Einsatz von Data-Mining-Verfahren im Walzwerk [MSD+10] A collaborative work with our project partners on the opportunities of using data analysis for predicting the quality of products in smart manufacturing applications.

Automatic Selection of Machine Learning Models for WCET-aware Compiler Heuristic Generation [LSMM10] A paper in collaboration with colleagues from our embedded systems group. Determination of the Worst Case Execution Time (WCET)

is highly relevant for real-time constraint systems. Data analysis is used to improve on the time a compiler needs to calculate the WCET.

Bibliography

[AAS13] C.C. Aggarwal, N. Ashish, and A. Sheth. The Internet of Things: A Survey From The Data-Centric Perspective. In C.C. Aggarwal, editor, *anaging and Mining Sensor Data*. Springer, Berlin, Heidelberg, 2013.

[ABLS10] F. Angiulli, S. Basta, S. Lodi, and C. Sartori. A Distributed Approach to Detect Outliers in Very Large Data Sets. In *Proc. of European Conf. on Parallel Processing (Euro-Par)*, pages 329–340, 2010.

[AC79] M.S. Ahmed and A.R. Cook. *Analysis of Freeway Traffic Time Series Data Using Box and Jenkins Techniques*. University of Oklahoma, School of Civil Engineering and Environmental Science, 1979.

[Aha92] D. Aha. Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms. *Int. J. of Man-Machine Studies*, 36(2):267–287, 1992.

[AHWY03] C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu. A Framework for Clustering Evolving Data Streams. In *Proc. of the 29th Int. Conf. on Very Large Data Bases (VLDB)*, pages 81–92, 2003.

[AIM10] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Comput. Netw.*, 54(15):2787–2805, 2010.

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 207–216, Washington, D. C., May 1993.

[AN07] A. Asuncion and D. J. Newman. UCI Machine Learning Repository, 2007.

[Arg15] Argonne National Laboratory. The Message Passing Interface (MPI) standard. `http://www.mcs.anl.gov/research/projects/mpi/`, 2015. [Online; accessed 2015-12-15].

[AS95] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proc. of the 11th Int. Conf. on Data Engineering (ICDE)*, pages 3–14, Washington, DC, USA, Mar. 1995. IEEE.

[AWY+99] C.C. Aggarwal, J.L. Wolf, P.S. Yu, C. Procopiuc, and J.S. Park. Fast Algorithms for Projected Clustering. In *Proc. of the 1999 ACM SIGMOD Int. Conf. on Management of Data*, SIGMOD '99, pages 61–72, New York, NY, USA, 1999. ACM.

[AY07] A.A. Abbasi and M. Younis. A survey on clustering algorithms for wireless sensor networks. *Comput. Commun.*, 30(14–15):2826–2841, Oct. 2007.

[BA10] F. Bajaber and I. Awan. Energy efficient clustering protocol to enhance lifetime of wireless sensor network. *Journal of Ambient Intelligence and Humanized Computing*, 1:239–248, 2010.

[BAM09] C. Bockermann, M. Apel, and M. Meier. Learning SQL for Database Intrusion Detection Using Context-Sensitive Modelling. In U. Flegel, , and D. Bruschi, editors, *Proc. of the 6th Int. Conf. on Detection of Intrusions and Malware (DIMVA)*, pages 196–205. Springer, Berlin, Heidelberg, 2009.

[BBB+15] C. Bockermann, K. Brügge, J. Buss, A. Egorov, K. Morik, W. Rhode, and T. Ruhe. Online Analysis of High-Volume Data Streams in Astroparticle Physics. In *Proc. of the European Conf. on Machine Learning (ECML), Industrial Track*. Springer, 2015.

[BBFM12] M.-F. Balcan, A. Blum, S. Fine, and Y. Mansour. Distributed Learning, Communication Complexity and Privacy. In *JMLR: Workshop and Conference Proceedings, 25th Annual Conference on Learning Theory*, pages 26:1–26.22, 2012.

[BC00] K.P. Bennett and C. Campbell. Support Vector Machines: Hype or Hallelujah? *SIGKDD Explor. Newsl.*, 2(2):1–13, Dec. 2000.

[BC02] M. Bădoiu and K.L. Clarkson. Optimal Core Sets for Balls. In *DIMACS Workshop on Computational Geometry*, 2002.

[BC03] S. Bandyopadhyay and E.J. Coyle. An Energy Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks. In *Proc. of the 22nd Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 1713–1723, Apr. 2003.

[BCV13] Y. Bengio, A. Courville, and P. Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug. 2013.

[BdDPP16] A. Botta, W. de Donato, V. Persico, and A. Pescapé. Integration of Cloud computing and Internet of Things: A survey. *Future Gener. Comp. Sy.*, 56:684–700, 2016.

[BE81] D. Baker and A. Ephremides. The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm. *IEEE Trans. on Communications*, 29(11):1694–1701, Nov. 1981.

[BFKR14] M. Brettel, N. Friederichsen, M. Keller, and M. Rosenberg. How Virtualization, Decentralization and Network Building Change the Manufacturing Landscape: An Industry 4.0 Perspective. *Int. Journ. of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering*, 8(1):37–44, 2014.

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, CA, 1984.

[BG15] A. Buczak and E. Guven. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 2015.

[BGM+06] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, and S. Datta. Clustering Distributed Data Streams in Peer-to-Peer Environments. *Information Sciences*, 176(14):1952–1985, 2006.

[BGS+12] J.W. Branch, C. Giannella, B. Szymanski, R. Wolff, and H. Kargupta. In-network outlier detection in wireless sensor networks. *Knowl. Inf. Sys.*, 34(1):23–54, 2012.

[BGSW06] U. Brefeld, T. Gärtner, T. Scheffer, and S. Wrobel. Efficient Co-regularised Least Squares Regression. In *Proc. of the 23rd Int. Conf. on Machine Learning (ICML)*, pages 137–144, New York, NY, USA, 2006. ACM.

[BHHSV02] A. Ben-Hur, D. Horn, H.T. Siegelmann, and V. Vapnik. Support Vector Clustering. *J. Mach. Learn. Res.*, 2:125–137, Mar. 2002.

[BHKP10] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis. *J. Mach. Learn. Res.*, 11:1601–1604, Aug. 2010.

[BHL07] L.M.A. Bettencourt, A.A. Hagberg, and L.B. Larkey. Separating the Wheat from the Chaff: Practical Anomaly Detection Schemes in Ecological Applications of Distributed Sensor Networks. In *Proc. of the 3rd IEEE Int. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, pages 223–239, 2007.

[BJR94] G.E.P. Box, G. Jenkins, and G.C. Reinsel. *Time Series Analysis. Forecasting and Control*. Prentice Hall, Englewood Cliffs, 3rd edition, 1994.

[BK01] S. Banerjee and S. Khuller. A Clustering Scheme for Hierarchical Control in Multi-hop Wireless Networks. In *Proc. of the 20th Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1028–1037, 2001.

[BM98] A. Blum and T. Mitchell. Combining Labeled and Unlabeled Data with Co-training. In *Proc. of the 11th Ann. Conf. on Computational Learning Theory*, pages 92–100, New York, NY, USA, 1998. ACM.

[BMG11] K. Bhaduri, B. L. Matthews, and C. Giannella. Algorithms for Speeding up Distance-based Outlier Detection. In *Proc. of the 17th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 859–867, 2011.

[Boc15] C. Bockermann. *Mining Big Data Streams for Multiple Concepts*. PhD thesis, TU Dortmund, Dortmund, Germany, 2015.

[BPC+11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, Jan. 2011.

[Bre01] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[Bre15] U. Brefeld. Multi-view Learning with Dependent Views. In *Proc. of the 30th Annual ACM Symposium on Applied Computing (SAC)*, pages 865–870, New York, NY, USA, 2015. ACM.

[BS04] U. Brefeld and T. Scheffer. Co-EM Support Vector Learning. In *Proc. of the 20th Int. Conf. on Machine Learning (ICML)*, pages 121–128, New York, NY, USA, 2004. ACM.

[BS13] K. Bhaduri and M. Stolpe. Distributed Data Mining in Sensor Networks. In C.C. Aggarwal, editor, *Managing and Mining Sensor Data*, chapter 8. Springer, Berlin, Heidelberg, 2013.

[BSDM13] F. Bohnen, M. Stolpe, J. Deuse, and K. Morik. Using a Clustering Approach with Evolutionary Optimized Attribute Weights to Form Product Families for Production Leveling. In Katja Windt, editor, *Robust Manufacturing Control*, Lecture Notes in Production Engineering, pages 189–202, Berlin, Heidelberg, 2013. Springer-Verlag.

[BSG+06] J. Branch, B. Szymanski, C. Giannella, R. Wolff, and H. Kargupta. In-Network Outlier Detection in Wireless Sensor Networks. In *Proc. of the 26th IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, page 51, 2006.

[Bur14] D. Burrus. The Internet of Things is Far Bigger Than Anyone Realizes. `http://www.wired.com/insights/2014/11/the-internet-of-things-bigger/`, 2014. [Online; accessed 2016-02-16].

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[BW16] W. Bernhart and M. Winterhoff. Autonomous Driving: Disruptive Innovation that Promises to Change the Automotive Industry as We Know It. In J. Langheim, editor, *Energy Consumption and Autonomous Driving: Proc. of the 3rd CESA Automotive Electronics Congress*. Springer, 2016.

[BWGK08] K. Bhaduri, R. Wolff, C. Giannella, and H. Kargupta. Distributed Decision Tree Induction in P2P Systems. *Stat. Anal. and Data Mining*, 1(2):85–103, 2008.

[BYX10] S. Bin, L. Yuan, and W. Xiaoyi. Research on Data Mining Models for the Internet of Things. In *Proc. of the Int. Conf. on Image Analysis and Signal Processing (IASP)*, pages 127–132, 2010.

[BZ11] N. Bui and M. Zorzi. Health Care Applications: A Solution Based on the Internet of Things. In *Proc. of the 4th Int. Symp. on Applied Sciences in Biomedical and Communication Technologies*, ISABEL '11, pages 131:1–131:5. ACM, 2011.

[BZB04] J. Bi, T. Zhang, and K.P. Bennett. Column-generation Boosting Methods for Mixture of Kernels. In *Proc. of the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 521–526, New York, NY, USA, 2004. ACM.

[Can14] Canalys. Wearable band shipments set to exceed 43.2 million units in 2015. `http://www.canalys.com/newsroom/wearable-band-shipments-set-exceed-432-million-units-2015`, 2014. [Online; accessed 2016-04-04].

[CBK09] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, Jul. 2009.

[CCH05] C. Caragea, D. Caragea, and V. Honavar. Learning Support Vector Machines from Distributed Data Sources. In *Proc. of the 20th National Conf. on Artificial Intelligence (AAAI)*, volume 4, pages 1602–1603. AAAI Press, 2005.

[CDW+15] F. Chen, P. Deng, J. Wan, D. Zhang, A. Vasilakos, and X. Rong. Data Mining for the Internet of Things: Literature Review and Challenges. *Int. J. Distrib. Sen. Netw.*, 2015:12:12–12:12, Jan. 2015.

[CFPS99] P. Chan, W. Fan, A. Prodromidis, and S. Stolfo. Distributed Data Mining in Credit Card Fraud Detection. *IEEE Intelligent Systems*, 14:67–74, 1999.

[CFSZ11] A. Chiuso, F. Fagnani, L. Schenato, and S. Zampieri. Gossip Algorithms for Simultaneous Distributed Estimation and Classification in Sensor Networks. *J. Sel. Topics Signal Processing*, 5(4):691–706, 2011.

[CH10] A. Carroll and G. Heiser. An Analysis of Power Consumption in a Smartphone. In *Proc. of the 2010 USENIX Conf. on USENIX Ann. Technical Conf. (USENIXATC)*, USA, 2010. USENIX Association.

[CKL03] B. Chiu, E. Keogh, and S. Lonardi. Probabilistic Discovery of Time Series Motifs. In *Proc. of the 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 493–498, New York, NY, USA, 2003. ACM.

[CLQZ09] S. Chen, B. Liu, M. Qian, and C. Zhang. Kernel K-means Based Framework for Aggregate Outputs Classification. In *Proc. of the IEEE Int. Conf. on Data Mining Workshops (ICDMW)*, pages 356–361, 2009.

[CLR10] M. Chui, M. Löffler, and R. Roberts. The Internet of Things. `http://www.mckinsey.com/insights/high_tech_telecoms_internet/the_internet_of_things`, Mar. 2010. [Online; accessed 2016-02-16].

[CMW+13] M. Chen, Y. Ma, J. Wang, D.O. Mau, and E. Song. Enabling Comfortable Sports Therapy for Patient: A Novel Lightweight Durable and Portable ECG Monitoring System. In *IEEE 15th Int. Conf. on e-Health Networking, Applications and Services (Healthcom)*, pages 271–273, 2013.

[CMZ07] G. Cormode, S. Muthukrishnan, and Wei Z. Conquering the Divide: Continuous Clustering of Distributed Data Streams. In *Proc. of the 23rd IEEE Int. Conf. on Data Eng. (ICDE)*, pages 1036–1045, Apr. 2007.

[Com15] F. Combaneyre. Understanding Data Streams in IoT. `http://www.sas.com/en_us/whitepapers/understanding-data-streams-in-iot-107491.html`, 2015. [Online; accessed 2016-02-23].

[CRWS12] K.S. Candan, R. Rossini, X. Wang, and M.L. Sapino. sDTW: Computing DTW Distances using Locally Relevant Constraints based on Salient Feature Alignments. *Proc. VLDB Endow.*, 5(11):1519–1530, Jul. 2012.

[CSH00a] D. Caragea, A. Silvescu, and V. Honavar. Agents that Learn from Distributed Dynamic Data Sources. In *Proc. of the Workshop on Learning Agents*, 2000.

[CSH00b] D. Caragea, A. Silvescu, and V. Honavar. Towards a Theoretical Framework for Analysis and Synthesis of Agents That Learn from Distributed Dynamic Data Sources. In *Proc. of the Workshop on Distributed and Parallel Knowledge Discovery. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2000.

[CSZ06] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[CXPL10] X. Cheng, J. Xu, J. Pei, and J. Liu. Hierarchical distributed data classification in wireless sensor networks. *Comput. Commun.*, 33(12):1404–1413, 2010.

[Dav12] E.R. Davies. *Computer and Machine Vision: Theory, Algorithms, Practicalities*. Academic Pr, Inc., 2012.

[DBBE99] A. Demiriz, K. Bennett, K.P. Bennett, and M.J. Embrechts. Semi-Supervised Clustering Using Genetic Algorithms. In *Proc. of Artificial Neural Networks in Engineering (ANNIE)*, pages 809–814. ASME Press, 1999.

[DBK09] K. Das, K. Bhaduri, and H. Kargupta. A Local Asynchronous Distributed Privacy Preserving Feature Selection Algorithm for Large Peer-to-Peer Networks. *Knowledge and Information Systems*, 24(3):341–367, 2009.

[DBV11] K. Das, K. Bhaduri, and P. Votava. Distributed Anomaly Detection Using 1-class SVM for Vertically Partitioned Data. *Stat. Anal. Data Min.*, 4(4):393–406, Aug. 2011.

[Dem06] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *The Journal of Machine Learning Research (JMLR)*, 7:1–30, 2006.

[DFMB15] G. De Francisci Morales and A. Bifet. SAMOA: Scalable Advanced Massive Online Analysis. *J. Mach. Learn. Res.*, 16(1):149–153, Jan. 2015.

[DGK04] I.S. Dhillon, Y. Guan, and B. Kulis. Kernel k-Means: Spectral Clustering and Normalized Cuts. In *Proc. of the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 551–556, New York, NY, USA, 2004. ACM.

[DGK09] S. Datta, C. Giannella, and H. Kargupta. Approximate Distributed K-Means Clustering over a Peer-to-Peer Network. *IEEE Trans. on Knowl. and Data Eng.*, 21(10):1372–1388, Oct. 2009.

[DGM97] G. Das, D. Gunopulos, and H. Mannila. Finding Similar Time Series. In *Principles of Data Mining and Knowledge Discovery*, volume 1263 of

Lecture Notes in Computer Science, pages 88–100. Springer, Berlin, Heidelberg, 1997.

[DH73] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[DH04] M.F. Duarte and Y.H. Hu. Vehicle classification in distributed sensor networks. *J. Parallel Distrib. Comput.*, 64(7):826–838, Jul. 2004.

[DHC05] P. Ding, J. Holliday, and A. Celik. Distributed Energy-Efficient Hierarchical Clustering for Wireless Sensor Networks. In *Proc. of the 1st IEEE Int. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, pages 322–339. Springer, 2005.

[Dix15] J. Dixon. Who Will Step Up To Secure The Internet of Things? `http://techcrunch.com/2015/10/02/who-will-step-up-to-secure-the-internet-of-things/`, 2015. [Online; accessed 2016-02-16].

[DK07] S. Datta and H. Kargupta. Uniform Data Sampling from a Peer-to-Peer Network. In *Proc. of the 27th IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, page 50, 2007.

[DKS02] R. Dara, S. Kremer, and D. Stacey. Clustering Unlabeled Data with SOMs Improves Classification of Labeled Real-world Data. In *Proc. of the 2002 Int. Joint Conf. on Neural Networks (IJCNN)*, volume 3, pages 2237–2242, 2002.

[DLR77] A.P. Dempster, M.N. Laird, and D.B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 39:1–38, 1977.

[DMR16] M. Díaz, C. Martín, and B. Rubio. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *Journal of Network and Computer Applications*, 2016.

[DMSO10] S. Das, B. Matthews, A. Srivastava, and N. Oza. Multiple Kernel Learning for Heterogeneous Anomaly Detection: Algorithm and Aviation Safety Case Study. In *Proc. of the 16th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 47–56, 2010.

[DP06] T.-N. Do and F. Poulet. Classifying one billion data with a new distributed SVM algorithm. In *Int. Conf. on Research, Innovation and Vision for the Future*, pages 59–66, Feb. 2006.

[DP10] W. Dargie and C. Poellabauer. *Wireless Sensor Networks: Technology, Protocols, and Applications*. John Wiley & Sons, 2010.

[DS03] A. D'Costa and A.M. Sayeed. Collaborative Signal Processing for Distributed Classification in Sensor Networks. In *Proc. of the 2nd Int. Conf. on Information Processing in Sensor Networks*, volume 2634 of *Lecture Notes in Computer Science*, pages 558–558. Springer, 2003.

[EA12] P.C. Evans and M. Annunziata. Industrial Internet: Pushing the Boundaries of Minds and Machines. `http://www.ge.com/docs/chapters/Industrial_Internet.pdf`, 2012. [Online; accessed 2016-04-04].

[EGPS01] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting The World With Wireless Sensor Networks. In *Proc. of the 27th IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 2001.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. of the 2nd Int. Conf. on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.

[Elk03] C. Elkan. Using the Triangle Inequality to Accelerate k-Means. In *Proc. of the 20th Int. Conf. on Machine Learning (ICML)*, 2003.

[Eng13] P. Engebretson. *The Basics of Hacking and Penetration Testing.* Elsevier/Syngress, 2nd edition, 2013.

[Eva11] D. Evans. The Internet of Things – How the Next Evolution of the Internet Is Changing Everything. `https://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf`, Apr. 2011. [Online; accessed 2015-11-19].

[Faw16] T. Fawcett. Mining the Quantified Self: Personal Knowledge Discovery as a Challenge for Data Science. *Big Data*, 3(4):249–266, Jan. 2016.

[FBLT06] K. Flouri, B. Beferull-Lozano, and P. Tsakalides. Training a SVM-based Classifier in Distributed Sensor Networks. In *EUSIPCO 2006*, 2006.

[FBLT08] K. Flouri, B. Beferull-Lozano, and P. Tsakalides. Distributed Consensus Algorithms for SVM Training in Wireless Sensor Networks. In *EUSIPCO*, 2008.

[FBLT09] K. Flouri, B. Beferull-Lozano, and P. Tsakalides. Optimal Gossip Algorithm for Distributed Consensus SVM Training in Wireless Sensor Networks. In *16th Int. Conf. on Digital Signal Processing*, pages 1–6, Jul. 2009.

[FCG10] P.A. Forero, A. Cano, and G.B. Giannakis. Consensus-Based Distributed Support Vector Machines. *J. Mach. Learn. Res.*, 11:1663–1707, Aug. 2010.

[FJM+10a] P. Fricke, F. Jungermann, K. Morik, N. Piatkowski, O. Spinczyk, and M. Stolpe. Towards Adjusting Mobile Devices to User's Behaviour. In *Proc. of the Int. Workshop at ECML/PKDD on Mining Ubiquitous and Social Environments (MUSE)*, pages 7–22, 2010.

[FJM+10b] P. Fricke, F. Jungermann, K. Morik, N. Piatkowski, O. Spinczyk, and M. Stolpe. Towards Adjusting Mobile Devices to User's Behaviour. In M. Atzmueller, D. Benz, A. Hotho, and G. Stumme, editors, *Lernen, Wissen & Adaptivität (LWA) – Workshop Proceedings*, pages 51–58, 2010. (resubmission).

[FJM+11] P. Fricke, F. Jungermann, K. Morik, N. Piatkowski, O. Spinczyk, M. Stolpe, and J. Streicher. Towards Adjusting Mobile Devices To User's Behaviour. In M. Atzmueller, A. Hotho, M. Strohmaier, and A. Chin, editors, *Analysis of Social Media and Ubiquitous Data*, volume 6904 of *Lecture Notes in Computer Science*, pages 99–118. Springer-Verlag, Berlin, Heidelberg, 2011.

[Fle15] D. Fletcher. Internet of Things. In M. Blowers, editor, *Evolution of Cyber Technologies and Operations to 2035*, pages 19–32. Springer International Publishing, 2015.

[FRM94] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In *Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data*, pages 419–429, New York, NY, USA, 1994. ACM Press.

[FS10] P. Fricke and M. Stolpe. Implementing Hierarchical Heavy Hitters in RapidMiner: Solutions and Open Questions. In *Proc. of the RapidMiner Community Meeting And Conference (RCOMM)*, 2010.

[FT15] L. Fan and G. Taylor. Alter-CNN: An Approach to Learning from Label Proportions with Application to Ice-Water Classification. In *Workshop on Learning and privacy with incomplete data and weak supervision at the annual conf.on Neural Information Processing Systems (NIPS)*, 2015.

[FZY+14] K. Fan, H. Zhang, S. Yan, L. Wang, W. Zhang, and J. Feng. Learning a Generative Classifier from Label Proportions. *Neurocomput.*, 139:47–55, Sep. 2014.

[Gam10] J. Gama. *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC, 1st edition, 2010.

[GBA+13] E.I. Gaura, J. Brusey, M. Allen, R. Wilkins, D. Goldsmith, and R. Rednic. Edge Mining the Internet of Things. *IEEE Sensors Journal*, 13(10):3816–3825, 2013.

[GBMP13] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Gener. Comput. Syst.*, 29(7):1645–1660, Sep. 2013.

[GCB+05] H.P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel Support Vector Machines: The Cascade SVM. In L.K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 17, pages 521–528. MIT Press, 2005.

[GKRB09] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld. Active Learning for Network Intrusion Detection. In *Proc. of the 2nd ACM Workshop on Security and Artificial Intelligence (AISec)*, pages 47–54, New York, NY, USA, 2009. ACM.

[Gla15] J. Glaser. How The Internet of Things Will Affect Health Care. `http://www.hhnmag.com/articles/3438-how-the-internet-of-things-will-affect-health-care`, Jun. 2015. [Online; accessed 2016-02-23].

[GMUW13] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Pearson Education Limited, 2nd edition, 2013.

[GP07] F. Gianotti and D. Pedreschi, editors. *Mobility, Data Mining and Privacy*. Springer, 2007.

[Gu08] D. Gu. Distributed EM Algorithm for Gaussian Mixtures in Sensor Networks. *IEEE Trans. on Neural Networks*, 19(7):1154–1166, Jul. 2008.

[HA04] V. Hodge and J. Austin. A Survey of Outlier Detection Methodologies. *A. I. Review*, 22(2):85–126, 2004.

[Han05] J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 2005.

[Har15] B. Harpham. How the Internet of Things is changing healthcare and transportation. `http://www.cio.com/article/2981481/healthcare/how-the-internet-of-things-is-changing-healthcare-and-transportation.html`, Sep. 2015. [Online; accessed 2016-02-16].

[HC02] E. Hung and D. Cheung. Parallel Mining of Outliers in Large Database. *Distrib. Parallel Databases*, 12:5–26, 2002.

[HCB02] W.B. Heinzelman, A.P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Trans. on Wireless Communications*, 1(4):660–670, Oct. 2002.

[HGInL13] J. Hernández-González, I. Iñaki, and J.A. Lozano. Learning Bayesian network classifiers from label proportions. *Pattern Recognition*, 46(12):3425–3440, 2013.

[Hin13] P. Hintjens. *ZeroMQ*. O'Reilly, USA, 2013.

[HK06] J. Han and M. Kamber. *Data Mining*. Morgan Kaufmann, 2nd edition, 2006.

[HMM16] C. Heinze, B. McWilliams, and N. Meinshausen. DUAL-LOCO: Distributing Statistical Estimation Using Random Projections. In A. Gretton and C.C. Robert, editors, *Proc. of the 19th Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, volume 51 of *JMLR: Workshop and Conference Proceedings*, pages 875–883, 2016.

[HMMK14] C. Heinze, B. McWilliams, N. Meinshausen, and G. Krummenacher. LOCO: Distributing Ridge Regression with Random Projections. In *NIPS Workshop on Distributed Machine Learning and Matrix Computations*, 2014.

[HMS08] T. Hazan, A. Man, and A. Shashua. A Parallel Decomposition Solver for SVM: Distributed dual ascend using Fenchel Duality. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, Jun. 2008.

[HMS09] M. Hassani, E. Müller, and T. Seidl. EDISKCO: Energy Efficient Distributed In-Sensor-Network K-center Clustering with Outliers. In *Proc. of the 3rd Int. Workshop on Knowl. Discovery from Sensor Data (SensorKDD)*, pages 39–48, 2009.

[HSP+12] A.-C. Hauschild, T. Schneider, J. Pauling, K. Rupp, M. Jang, J.I. Baumbach, and J. Baumbach. Computational Methods for Metabolomic Data Analysis of Ion Mobility Spectrometry Data - Rviewing the State of the Art. *Metabolites*, 2(4):733–755, 2012.

[HSSK06] J. Harding, M. Shahbaz, Srinivas, and A. Kusiak. Data Mining in Manufacturing: A Review. *J. Manuf. Sci. Eng*, 128(4):969–976, 2006.

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

[IBM16] IBM. IBM Intelligent Water: Water management software with analytics for improved infrastructure and operations. `http://www-03.ibm.com/software/products/en/intelligentwater`, 2016. [Online; accessed 2016-04-01].

[IFGL03] M. Imhoff, R. Fried, U. Gather, and V. Lanius. Dimension Reduction for Physiological Variables Using Graphical Modeling. In *AMIA 2003, American Medical Informatics Association Annual Symposium, Washington, DC, USA, November 8-12, 2003*, 2003.

[JAF+06] S.R. Jeffery, G. Alonso, M.J. Franklin, W. Hong, and J. Widom. Declarative Support for Sensor Data Cleaning. In *Pervasive Computing*, Lecture Notes in Computer Science, pages 83–100, Berlin, 2006. Springer.

[JKP04] E. Januzaj, H.-P. Kriegel, and M. Pfeifle. Scalable Density-Based Distributed Clustering. In *Proc. of the 8th European Conf. on Principles and Practice of Knowl. Discovery in Databases (OKDD)*, pages 231–244. Springer, 2004.

[JL95] G.H. John and P. Langley. Estimating Continuous Distributions in Bayesian Classifiers. In *Proc. of the 11th Conf. on Uncertainty in Artificial Intelligence*, pages 338–345, San Francisco, 1995. Morgan Kaufmann.

[Joa99] T. Joachims. Transductive Inference for Text Classification using Support Vector Machines. In *Proc. of the 16th Int. Conf. on Machine Learning (ICML)*, pages 200–209, San Francisco, CA, 1999. Morgan Kaufmann.

[JW92] R.A. Johnson and D.W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, Inc., 3rd edition, 1992.

[KBK+14] M. Kamp, M. Boley, D. Keren, A. Schuster, and I. Scharfman. Communication-Efficient Distributed Online Prediction by Decentralized Variance Monitoring. In T. Calders, F. Esposito, E. Hüllermeier, and R. Meo, editors, *Proc. of the European Conf. on Machine Learning and Principles and Practice of Knowledge Discovery (ECML/PKDD)*, pages 623–639. Springer, 2014.

[KBL+04] H. Kargupta, R. Bhargava, K. Liu, M. Powers, P. Blair, S. Bushra, J. Dull, K. Sarkar, M. Klein, M. Vasa, and D. Handy. VEDAS: A Mobile and Distributed Data Stream Mining System for Real-Time Vehicle Monitoring. In *Proc. of the SIAM Int. Conf. on Data Mining (SDM)*, chapter 28, pages 300–311. 2004.

[KBSZ11] M. Kloft, U. Brefeld, S. Sonnenburg, and A. Zien. ℓ_p-Norm Multiple Kernel Learning. *Journal of Machine Learning Research (JMLR)*, 12, 2011.

[KdF05] H. Kueck and N. de Freitas. Learning about individuals from group statistics. In *Uncertainty in Artificial Intelligence (UAI)*, pages 332–339, Arlington, Virginia, 2005. AUAI Press.

[KJ97] R. Kohavi and G.H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[KK03] E. Keogh and S. Kasetty. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371, 2003.

[KKP+09] H.-P. Kriegel, P. Kröger, A. Pryakhin, M. Renz, and A. Zherdin. *Approximate Clustering of Time Series Using Compact Model-based Description*, volume 4947 of *Lecture Notes in Computer Science*, pages 364–379. Springer, Berlin, Heidelberg, 2009.

[KKPS05] H.-P. Kriegel, P. Kröger, A. Pryakhin, and M. Schubert. Effective and Efficient Distributed Model-Based Clustering. In *Proc. of the 5th IEEE Int. Conf. on Data Mining (ICDM)*, pages 258–265, 2005.

[KKZ09] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering High-dimensional Data: A Survey on Subspace Clustering, Pattern-based Clustering, and Correlation Clustering. *ACM Trans. Knowl. Discov. Data*, 3(1):1:1–1:58, Mar. 2009.

[KLD13] B. Konrad, D. Lieber, and J. Deuse. Striving for Zero Defect Production: Intelligent Manufacturing Control through Data Mining in Continuous Rolling Mill Processes. In *Robust Manufacturing Control (RoMaC)*, Lecture Notes in Computer Science, pages 215–229, Berlin, Heidelberg, 2013. Springer.

[KMS+16] A.R. Khan, A. Mahmood, A. Safdar, Z.A. Khan, and N.A. Khan. Load forecasting, dynamic pricing and DSM in smart grid: A review. *Renew. Sust. Energ. Rev.*, 54:1311–1322, 2016.

[KNK+15] R.J Krawiec, J. Nadler, P. Kinchley, E. Tye, and J. Jarboe. No appointment necessary: How the IoT and patient-generated data can unlock health care value. `http://dupress.com/articles/internet-of-things-iot-in-health-care-industry/`, Aug. 2015. [Online; accessed 2016-02-16].

[KNT00] E.M. Knorr, R.T. Ng, and V. Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 8(3-4):237–253, Feb. 2000.

[Koh89] T. Kohonen. *Self-Organization and Associative Memory*. Springer, Berlin, 1989.

[Koh95] R. Kohavi. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1137–1143, USA, 1995. Morgan Kaufmann.

[Kre14] J. Kreps. Questioning the Lambda Architecture. `http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html`, 2014. [Online; accessed 2015-12-15].

[KSBM00] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. A Fast Iterative Nearest Ppoint Algorithm for Support Vector Machine Classifier Design. *IEEE Transactions on Neural Networks*, 11(1):124–136, 2000.

[KSG10] H. Kargupta, K. Sarkar, and M. Gilligan. MineFleet®: An Overview of a Widely Adopted Distributed Vehicle Performance Data Mining System. In *Proc. of the 16th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 37–46, 2010.

[LA05] E. Lozano and E. Acuna. Parallel Algorithms for Distance-Based and Density-Based Outliers. In *Proc. of the Int. Conf. on Data Mining (ICDM)*, pages 729–732, 2005.

[LCB+02] G.R.G. Lanckriet, N. Christianini, P. Bartlett, L.E. Ghaoui, and M.I. Jordan. Learning the Kernel Matrix with Semidefinite Programming. In *Proc. of the 19th Int. Conference on Machine Learning (ICML)*, 2002.

[LG10] V. Lanius and U. Gather. Robust online signal extraction from multivariate time series. *Comput. Stat. Data An.*, 54(4):966–975, 2010.

[LKD+12] D. Lieber, B. Konrad, J. Deuse, M. Stolpe, and K. Morik. Sustainable Interlinked Manufacturing Processes through Real-Time Quality Prediction. In D.A. Dornfeld and B.S. Linke, editors, *Leveraging Technology for a Sustainable World*, pages 393–398, Berlin, Heidelberg, 2012. CIRP, Springer-Verlag.

[LKL14] M. Längkvist, L. Karlsson, and A. Loutfi. A Review of Unsupervised Feature Learning and Deep Learning for Time-Series Modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

[LKWL07] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a Novel Symbolic Representation of Time Series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[Low04] D.G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[LPBM14] T. Liebig, N. Piatkowski, C. Bockermann, and K. Morik. Predictive Trip Planning - Smart Routing in Smart Cities. In *Proc. of the Workshops of the EDBT/ICDT Joint Conference*, volume 1133, pages 331–338. CEUR-WS.org, 2014.

[LR06] Y. Lu and V. Roychowdhury. Parallel Randomized Support Vector Machine. In W.-K. Ng, M. Kitsuregawa, J. Li, and K. Chang, editors, *Advances in Knowledge Discovery and Data Mining*, volume 3918 of *Lecture Notes in Computer Science*, pages 205–214. Springer, 2006.

[LSFB15] J. Long, J.K. Swidrak, M.Y. Feng, and O. Buyukozturk. Smart Sensors: A Study of Power Consumption and Reliability. In E. Wee Sit, editor, *Sensors and Instrumentation, Volume 5: Proc. of the 33rd IMAC, A Conf. and Exposition on Structural Dynamics*, pages 53–60. Springer, 2015.

[LSK+13] D. Lieber, M. Stolpe, B. Konrad, J. Deuse, and K. Morik. Quality Prediction in Interlinked Manufacturing Processes based on Supervised & Unsupervised Machine Learning. In *Procedia CIRP - 46th CIRP Conf. on Manufacturing Systems*, volume 7, pages 193–198. Elsevier, 2013.

[LSM12] S. Lee, M. Stolpe, and K. Morik. Separable Approximate Optimization of Support Vector Machines for Distributed Sensing. In P.A. Flach, T. De Bie, and N. Christianini, editors, *Proc. of the European Conf. on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, volume 7524, pages 387–402, Berlin, Heidelberg, 2012. Springer.

[LSM15] T. Liebig, M. Stolpe, and K. Morik. Distributed Traffic Flow Prediction with Label Proportions: From in-Network towards High Performance Computation with MPI. In *Proc. of MUD2*, volume 1392, pages 36–43. CEUR-WS, 2015.

[LSMM10] P. Lokuciejewski, M. Stolpe, K. Morik, and P. Marwedel. Automatic Selection of Machine Learning Models for WCET-aware Compiler Heuristic Generation. In *Proc. of the 4th Workshop on Statistical and Machine Learning Approaches to ARchitecture and compilaTion (SMART)*, 2010.

[LTT06] W. H. K. Lam, Y. F. Tang, and M. Tam. Comparison of two non-parametric models for daily traffic forecasting in Hong Kong. *Journal of Forecasting*, 25(3):173–192, 2006.

[LXMW12] T. Liebig, Z. Xu, M. May, and S. Wrobel. Pedestrian Quantity Estimation with Trajectory Patterns. In *Machine Learning and Knowledge Discovery in Databases*, pages 629–643. Springer, 2012.

[MÖ7] M. Müller. Dynamic Time Warping. In *Information Retrieval for Music and Motion*, pages 69–84. Springer, Berlin, Heidelberg, 2007.

[Mac67] J.B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proc. of the 5th Berkeley Symp. on Mathematical Statistics and Probability*, pages 281–297, 1967.

[MBC+09] M. May, B. Berendt, A. Cornuejols, J. Gama, F. Giannotti, A. Hotho, D. Malerba, E. Menesalvas, K. Morik, R. Pedersen, L. Saitta, Y. Saygin, A. Schuster, and K. Vanhoof. Research Challenges in Ubiquitous Knowledge Discovery. In Kargupta, Han, Yu, Motwani, and Kumar, editors, *Next Generation of Data Mining (NGDM)*, pages 131–151. CRC Press, 2009.

[MBK12] K. Morik, K. Bhaduri, and H. Kargupta. Introduction to data mining for sustainability. *Data Min. Knowl. Disc.*, 24(2):311–324, Mar. 2012.

[McC14] B. McCann. A Review of SCATS Operation and Deployment in Dublin. In *Proc. of the 19th JCT Traffic Signal Symposium & Exhibition.* JCT Consulting Ltd, 2014.

[MCO07] D.R. Musicant, J.M. Christensen, and J.F. Olson. Supervised Learning by Training on Aggregate Outputs. In *Proc. of the 7th Int. Conf. on Data Mining (ICDM)*, pages 252–261, Oct. 2007.

[MF10] F. Mattern and C. Floerkemeier. From the Internet of Computers to the Internet of Things. In K. Sachs, I. Petrov, and P. Guerrero, editors, *From Active Data Management to Event-based Systems and More*, pages 242–259. Springer-Verlag, Berlin, Heidelberg, 2010.

[MHK+08] M. May, D. Hecker, C. Körner, S. Scheider, and D. Schulz. A Vector-Geometry Based Spatial kNN-Algorithm for Traffic Frequency Predictions. In *Data Mining Workshops, Int. Conf. on Data Mining (ICDM)*, pages 442–447. IEEE Computer Society, 2008.

[Mie08] I. Mierswa. *Non-Convex and Multi-Objective Optimization in Data Mining.* PhD thesis, Technische Universität Dortmund, 2008.

[Mit97] Tom Mitchell. *Machine Learning.* Mcgraw-Hill Education Ltd, 1997.

[MK05] K. Morik and H. Köpcke. Features for Learning Local Patterns in Time-Stamped Data. In K. Morik, J.-F. Boulicaut, and A. Siebes, editors, *Local Pattern Detection: International Seminar, Dagstuhl Castle, Germany, April 12-16, 2004, Revised Selected Papers*, chapter 7, pages 98–114. Springer, 2005.

[MK11] R. Mallik and H. Kargupta. A Sustainable Approach for Demand Prediction in Smart Grids using a Distributed Local Asynchronous Algorithm. In *Proc. of the Conf. on Data Understanding (CIDU)*, pages 1–15, 2011.

[MKH93] M. Moya, M. Koch, and L. Hostetler. One-class classifier networks for target recognition applications. In *Proc. World Congress on Neural Networks*, pages 797–801. International Neural Network Society, 1993.

[MM05] I. Mierswa and K. Morik. Automatic Feature Extraction for Classifying Audio Data. *Machine Learning Journal*, 58:127–149, 2005.

[MNR02] C. Meesookho, S. Narayanan, and C.S. Raghavendra. Collaborative classification applications in sensor networks. In *In Proc. of the 2nd Sensor Array And Multichannel Signal Processing Workshop (SAM)*, pages 370–374, 2002.

[Mor99] K. Morik. Tailoring Representations to Different Requirements. In O. Watanabe and T. Yokomori, editors, *Proc. of the 10th Int. Conf. on Algorithmic Learning Theory (ALT)*, Lecture Notes in Artificial Intelligence, pages 1–12. Springer, 1999.

[Mor00] K. Morik. The Representation Race - Preprocessing for Handling Time Phenomena. In L.R. de Mántaras and E. Plaza, editors, *Proc. of the 11th Europ. Conf. on Machine Learning (ECML)*, pages 4–19. Springer, 2000.

[MS06] A. Meka and A.K. Singh. Distributed Spatial Clustering in Sensor Networks. In *Proc. of the 10th Int. Conf. on Advances in Database Technology (EDBT)*, pages 980–1000. Springer, 2006.

[MSD+10] K. Morik, M. Stolpe, J. Deuse, F. Bohnen, and U. Reichel. Prognosemodelle zur Ermittlung der Produkteigenschaften – Einsatz von Data-Mining-Verfahren im Walzwerk. *stahl und eisen*, 10:80–82, 2010.

[MSDPC12] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[MTV97] H. Mannila, H. Toivonen, and A.I. Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1(3):259–290, Nov. 1997.

[MVW00] Y. Matias, J.S. Vitter, and M. Wang. Dynamic Maintenance of Wavelet-Based Histograms. In *Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB)*, pages 101–110, San Francisco, CA, USA, 2000. Morgan Kaufmann.

[MW99] K. Morik and S. Wessel. Incremental Signal to Symbol Processing. In *Making Robots Smarter*, pages 185–198. Springer, 1999.

[MW14] N. Marz and J. Warren. *Big Data - Principles and best practices of scalable realtime data systems.* Manning, 2014.

[MWF08] O.L. Mangasarian, E.W. Wild, and G.M. Fung. Privacy-preserving Classification of Vertically Partitioned Data via Random Kernels. *ACM Trans. Knowl. Discov. Data*, 2(3):12:1–12:16, Oct. 2008.

[MWK+06] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. YALE: Rapid Prototyping for Complex Data Mining Tasks. In T. Eliassi-Rad, L.H. Ungar, M.C., and D. Gunopulos, editors, *Proc. of the 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 935–940, New York, USA, Aug. 2006. ACM Press.

[MZDM16] R. Moss, A. Zarebski, P. Dawson, and J.M. McCaw. Forecasting influenza outbreak dynamics in Melbourne from internet search query surveillance data. *Influenza and Other Respiratory Viruses*, page n/a, Feb. 2016.

[Nav14] Navigant Research. Shipments of Smart Thermostats Are Expected to Reach Nearly 20 Million by 2023. `https://www.navigantresearch.com/newsroom/shipments-of-smart-thermostats-are-expected-to-reach-nearly-20-million-by-2023`, 2014. [Online; accessed 2016-04-04].

[NC98] R. Nagpal and D. Coore. An Algorithm for Group Formation in an Amorphous Computer. In *Proc. of the 10th Int. Conf. on Parallel and Distributed Computing Systems (PDCS)*, 1998.

[NH99] R.M. Neal and G.E. Hinton. A View of the Em Algorithm that Justifies Incremental, Sparse, and other Variants. In M.I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. MIT Press, 1999.

[Now03] R.D. Nowak. Distributed EM algorithms for density estimation and clustering in sensor networks. *IEEE Trans. on Signal Processing*, 51(8):2245–2253, Aug. 2003.

[NRC+09] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen, and M. Srivastava. Sensor Network Data Fault Types. *ACM Transactions on Sensor Networks (TOSN)*, 5(3):1–29, 2009.

[NZC+15] X. Niu, Y. Zhu, Q.g Cao, X. Zhang, W. Xie, and K. Zheng. An Online-Traffic-Prediction Based Route Finding Mechanism for Smart City. *International Journal of Distributed Sensor Networks*, 2015.

[OFG97] E. Osuna, R. Freund, and F. Girosi. An Improved Training Algorithm for Support Vector Machines. In *Proc. of the 1997 IEEE Workshop on Neural Networks for Signal Processing*, pages 276–285. IEEE, 1997.

[OGP06] M. Otey, A. Ghoting, and S. Parthasarathy. Fast Distributed Outlier Detection in Mixed-Attribute Data Sets. *Data Min. Knowl. Discov.*, 12:203–228, 2006.

[Ora15a] Oracle Corporation. Energize Your Business with IoT-Enabled Applications. `http://www.oracle.com/us/dm/oracle-iot-cloud-service-2625351.pdf`, 2015. [Online; accessed 2016-02-16].

[Ora15b] Oracle Corporation. Unlocking the Promise of a Connected World: Using the Cloud to Enable the Internet of Things. `http://www.oracle.com/us/solutions/internetofthings/iot-and-cloud-wp-2686546.pdf`, 2015. [Online; accessed 2015-12-15].

[Oxf13] Oxford Economics. Manufacturing Transformation: Achieving competetive advantage in changing global marketplace. `http://www.oxfordeconomics.com/Media/Default/Thought%20Leadership/executive-interviews-and-case-studies/PTC/Manufacturing%20Transformation%20130607.pdf`, 2013. [Online; accessed 2016-04-04].

[PGR07] N.D. Phung, M.M. Gaber, and U. Rohm. Resource-aware Online Data Mining in Wireless Sensor Networks. In *IEEE Symp. on Comput. Intelligence and Data Mining (CIDM)*, pages 139–146, Apr. 2007.

[PK00] G.J. Pottie and W.J. Kaiser. Wireless integrated network sensors. *Commun. ACM*, 43(5):51–58, May. 2000.

[PK13] D. Partynski and S.G.M. Koo. Integration of Smart Sensor Networks into Internet of Things: Challenges and Applications. In *Proc. of the IEEE Int. Conf. on Green Computing and Communications (GreenCom) and IEEE Internet of Things (iThings) and IEEE Cyber, Physical and Social Computing (CPSCom)*, pages 1162–1167, 2013.

[Pla99] J.C. Platt. Fast Training of Support Vector Machines Using Sequential Minimal Optimization. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods*, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[PLM13] N. Piatkowski, S. Lee, and K. Morik. Spatio-Temporal Random Fields: Compressible Representation and Distributed Estimation. *Machine Learning*, 93(1):115–139, 2013.

[PNCR14] G. Patrini, R. Nock, T. Caetano, and P. Rivera. (Almost) No Label No Cry. In *Advances in Neural Information Processing Systems 27 (NIPS)*, pages 190–198. Curran Associates, Inc., 2014.

[PPKG03] T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Distributed Deviation Detection in Sensor Networks. *SIGMOD Rec.*, 32(4):77–82, Dec. 2003.

[PSJ11] D. Pechyony, L. Shen, and R. Jones. Solving Large Scale Linear SVM with Distributed Block Minimization. In *NIPS 2011 workshop on Big Learning: Algorithms, Systems and Tools for Learning at Scale.*, 2011.

[QSCL09] N. Quadrianto, A.J. Smola, T.S. Caetano, and Q.V. Le. Estimating Labels from Label Proportions. *J. Mach. Learn. Res.*, 10:2349–2374, Dec. 2009.

[QSF+16] Y. Qin, Q.Z. Sheng, N.J.G. Falkner, S. Dustdar, H. Wang, and A.V. Vasilakos. When things matter: A survey on data-centric internet of things. *J. Netw. Comput. Appl.*, 64:137–153, 2016.

[Qui86] J.R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

[Qui93] J.R. Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[RÖ1] Stefan Rüping. Incremental Learning with Support Vector Machines. In *Proc. of the 2001 IEEE Int. Conf. on Data Mining (ICDM)*, pages 641–642, 2001.

[RCM+12] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and Mining Trillions of Time Series Subsequences Under Dynamic Time Warping. In *Proc. of the 18th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 262–270, New York, NY, USA, 2012. ACM.

[RKLE12] T. Rakthanmanon, E. Keogh, S. Lonardi, and S. Evans. MDL-Based Time Series Clustering. *Knowl. and Inf. Systems*, 33(2):371–399, 2012.

[RLPB07] S. Rajasegarar, C. Leckie, M. Palaniswami, and J.C. Bezdek. Quarter Sphere Based Distributed Anomaly Detection in Wireless Sensor Networks. In *Proc. of the IEEE Int. Conf. on Communications (ICC)*, pages 3864–3869, 2007.

[RN06] B. Raney and K. Nagel. An Improved Framework for Large-Scale Multi-Agent Simulations of Travel Behavior. *Towards better performing European Transportation Systems*, pages 305–347, 2006.

[RN13] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2013.

[Rüp10] Stefan Rüping. SVM Classifier Estimation From Group Probabilities. In *Proc. of the 27th Int. Conf. on Machine Learning (ICML)*, 2010.

[RZL13] R. Roman, J. Zhou, and J. Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279, 2013.

[SA96] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proc. of the 5th Int. Conf. on Extending Database Technology*, volume 1057 of *Lecture Notes in Computer Science*, pages 3–17, London, UK, Mar. 1996. Springer.

[SBD15] M. Stolpe, K. Bhaduri, and K. Das. Distributed Support Vector Machines: An Overview. In S. Michaelis, N. Piatkowski, and M. Stolpe, editors, *Solving Large Scale Learning Tasks: Challenges and Algorithms*. Springer International Publishing, 2015.

[SBDM13] M. Stolpe, K. Bhaduri, K. Das, and K. Morik. Anomaly Detection in Vertically Partitioned Data by Distributed Core Vector Machines. In H. Blockeel, K. Kersting, S. Njissen, and F. Železný, editors, *Proc. of the European Conf. on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, pages 321–336, Berlin, Heidelberg, 2013. Springer.

[SBM16] M. Stolpe, H. Blom, and K. Morik. Sustainable Industrial Processes by Embedded Real-Time Quality Prediction. In *Computational Sustainability*, pages 201–243. Springer, 2016.

[SCA13] SCATS. Sydney Coordinated Adaptive Traffic System. `http://www.scats.com.au/`, 2013. [Online; accessed 2015-08-19].

[She00] C. Shearer. The CRISP-DM Model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4):13–22, 2000.

[SHKS99] N.A. Syed, S. Huan, L. Kah, and K. Sung. Incremental Learning with Support Vector Machines. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 1999.

[Sho11] D. Shoup. Free Parking or Free Markets. *Cato Unbound - A Journal of Debate*, 2011. [Online; accessed 2016-04-04].

[SJ15] U. Stanczyk and L.C. Jain, editors. *Feature Selection for Data and Pattern Recognition*. Studies in Computational Intelligence. Springer, 2015.

[SKRC10] K. Shvachko, Hairong K., S. Radia, and R. Chansler. The Hadoop Distributed File System. In *IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[SLM15] M. Stolpe, T. Liebig, and K. Morik. Communication-efficient learning of traffic flow in a network of wireless presence sensors. In *Proc. of the Workshop on Parallel and Distributed Computing for Knowledge Discovery in Data Bases (PDCKDD 2015)*, CEUR Workshop Proceedings, page (to appear). CEUR-WS, 2015.

[SLMJ07] B. Sheng, Q. Li, W. Mao, and W. Jin. Outlier Detection in Sensor Networks. In *Proc. of the 8th ACM Int. Symp. on Mobile ad hoc networking and computing (MobiHoc)*, pages 219–228, 2007.

[SLMM14] F. Schnitzler, T. Liebig, S. Mannor, and K. Morik. Combining a Gauss-Markov model and Gaussian process for traffic prediction in Dublin city center. In *Proc. of the Workshops of the EDBT/ICDT Joint Conference*, volume 1133, pages 373–374. CEUR-WS.org, 2014.

[SM11] M. Stolpe and K. Morik. Learning from Label Proportions by Optimizing Cluster Model Selection. In D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, editors, *Proc. of the European Conf. on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, pages 349–364. Springer, Berlin, Heidelberg, 2011.

[SM13] P. Senin and S. Malinchik. SAX-VSM: Interpretable Time Series Classification Using SAX and Vector Space Model. In *IEEE 13th Int. Conf. on Data Mining (ICDM)*, pages 1175–1180, Dec. 2013.

[Sma16] SmartSantanderSantander. Future Internet Research & Experimentation. `http://www.smartsantander.eu`, 2016. [Online; accessed 2016-04-01].

[SMK+11] M. Stolpe, K. Morik, B. Konrad, D. Lieber, and J. Deuse. Challenges for Data Mining on Sensor Data of Interlinked Processes. In *Proc. of the Next Generation Data Mining Summit (NGDM)*, 2011. http://www.kd2u.org/NGDM11/schedule.html.

[SPP+06] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online Outlier Detection in Sensor Data Using Non-Parametric Models. In *Proc. of the 32nd Int. Conf. on Very Large Data Bases (VLDB)*, pages 187–198, 2006.

[SPST+01] B. Schölkopf, J.C. Platt, J.C. Shawe-Taylor, A.J. Smola, and R.C. Williamson. Estimating the Support of a High-Dimensional Distribution. *Neural Comp.*, 13(7):1443–1471, 2001.

[SRSS06] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large Scale Multiple Kernel Learning. *J. Mach. Learn. Res.*, 7:1531–1565, Dec. 2006.

[SS02] B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, 2002.

[SSB12] T. Sahai, A. Speranzon, and A. Banaszuk. Hearing the clusters of a graph: A distributed algorithm. *Automatica*, 48(1):15–24, Jan. 2012.

[Ste12] W.D. Stephenson. IntelliStreets: Digital Scaffolding for 'Smart' Cities. `http://www.huffingtonpost.com/w-david-stephenson/intellistreets_b_1242972.html`, 2012. [Online; accessed 2016-04-04].

[Sti14] Jeremy Stierwalt. Formula 1 and HANA: How F1 Racing is Pioneering Big Data Analytics. `http://jeremystierwalt.com/2014/01/29/formula-1-and-hana-how-f1-racing-is-pioneering-big-data-analytics/`, 2014. [Online; accessed 2016-02-16].

[SV99] J.A.K. Suykens and J. Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[SVvL06] A. Siebes, J. Vreeken, and M. van Leeuwen. Item Sets That Compress. In *Proc. of the 6th SIAM Int. Conf. on Data Mining*, pages 395–418, 2006.

[Swe02] L. Sweeney. K-anonymity: A Model for Protecting Privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, Oct. 2002.

[Tap15] J. Tapper. Obama administration spied on German media as well as its government. `http://edition.cnn.com/2015/07/03/politics/germany-media-spying-obama-administration/index.html`, 2015. [Online; accessed: 2016-03-30].

[TD04] David M. J. Tax and Robert P. W. Duin. Support Vector Data Description. *Mach. Learn.*, 54:45–66, 2004.

[The15a] The Apache Software Foundation. Apache Flink: Scalable Batch and Stream Data Processing. `http://flink.apache.org/`, 2015. [Online; accessed 2015-12-15].

[The15b] The Apache Software Foundation. mahout. `http://mahout.apache.org/`, 2015. [Online; accessed 2016-03-30].

[THK16] N.A. Treiber, J. Heinermann, and O. Kramer. Wind Power Prediction with Machine Learning. In J. Lässig, K. Kersting, and K. Morik, editors, *Computational Sustainability*, volume 9570 of *Lecture Notes in Computer Science*. Springer, 2016.

[TKC05] I. Tsang, J. Kwok, and P. Cheung. Core Vector Machines: Fast SVM Training on Very Large Data Sets. *J. Mach. Learn. Res.*, 6:363–392, Dec. 2005.

[TLCY14] C.-W. Tsai, C.F. Lai, M.C. Chiang, and L.T. Yang. Data Mining for Internet of Things: A Survey. *IEEE Communications Surveys & Tutorials*, 16(1):77–97, 2014.

[TLV14] C.-W. Tsai, C.-F. Lai, and A.V. Vasilakos. Future Internet of Things: open issues and challenges. *Wirel. Netw.*, 20(8):2201–2217, 2014.

[TSV12] R.K. Tripathi, Y.N. Singh, and N.K. Verma. N-LEACH, a balanced cost cluster-heads selection algorithm for Wireless Sensor Network. In *National Conf. on Communications (NCC)*, pages 1–5, Feb. 2012.

[TvS06] A.S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2nd edition, 2006.

[Uni14] United Nations. World Urbanization Prospects. `http://esa.un.org/unpd/wup/Publications/Files/WUP2014-Report.pdf`, 2014. [Online; accessed 2016-04-04].

[Vap95] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[Vap99] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 2nd edition, 1999.

[Ver15] Verizon. State of the Market: The Internet of Things 2015. `http://www.verizonenterprise.com/state-of-the-market-internet-of-things/`, 2015. [Online; accessed 2015-10-22].

[VRR+15] A. K. Ramachandran Venkatapathy, A. Riesner, M. Roidl, J. Emmerich, and M. ten Hompel. PhyNode: An intelligent, cyber-physical system with energy neutral operation for PhyNetLab. In *Proc. of the Europ. Conf. on Smart Objects, Systems and Technologies, Smart SysTech*, 2015.

[WBK09] R. Wolff, K. Bhaduri, and H. Kargupta. A Generic Local Algorithm for Mining Data Streams in Large Distributed Systems. *IEEE Trans. on Knowl. and Data Eng.*, 21(4):465–478, Apr. 2009.

[WBX14] C. Wang, Z. Bi, and L. D. Xu. IoT and Cloud Computing in Automation of Assembly Modeling Systems. *IEEE T. Ind. Inform.*, 10(2):1426–1434, 2014.

[WCD+07] W. Wu, X. Cheng, M. Ding, K. Xing, F. Liu, and P. Deng. Localized Outlying and Boundary Data Detection in Sensor Networks. *IEEE Trans. on Knowl. and Data Eng.*, 19(8):1145–1157, Aug. 2007.

[WEH11] I.H. Witten, F. Eibe, and M.A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. The Morgan Kaufmann series in data management systems. Elsevier, Inc., Burlington, MA, 3rd edition, 2011.

[Wei91] M. Weiser. The Computer for the 21st Century. *Sci. Am.*, 265(9), 1991.

[WFH11] I.H. Witten, E. Frank, and M.A. Hall. *Data Mining*. Morgan Kaufmann, 3rd edition, 2011.

[Whi11] T. White. *Hadoop: The Definitive Guide*. O'Reilly, USA, 2nd edition, 2011.

[WK09] X. Wang and K.M. Kockelmann. Forecasting Network Data: Spatial Interpolation of Traffic Counts from Texas Data. *Journal of the Transportation Research Board*, 2105(13):100–108, 2009.

[WLK16] B. Wolff, E. Lorenz, and O. Kramer. Statistical Learning for Short-Term Photovoltaic Power Predictions. In J. Lässig, K. Kersting, and K. Morik, editors, *Computational Sustainability*, volume 9570 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2016.

[Woo12] E. Woods. Smart Street Lights Face Financial Hurdles. `https://www.navigantresearch.com/blog/smart-street-lights-face-financial-hurdles`, 2012. [Online; accessed 2016-04-04].

[WS04] R. Wolff and A. Schuster. Association Rule Mining in Peer-to-Peer Systems. *IEEE Trans. on Systems, Man and Cybernetics - Part B*, 34(6):2426–2438, Dec. 2004.

[XHL14] L.D. Xu, W. He, and S. Li. Internet of Things in Industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014.

[YF04] O. Younis and S. Fahmy. HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Trans. on Mobile Computing*, 3(4):366–379, 2004.

[YG08] J. Yin and M.M. Gaber. Clustering Distributed Time Series in Sensor Networks. In *Proc. of the 8th IEEE Int. Conf. on Data Mining (ICDM)*, pages 678–687, 2008.

[YHCL10] Hsiang-Fu Yu, Cho-Jui Hsieh, Kai-Wei Chang, and Chih-Jen Lin. Large Linear Classification when Data Cannot Fit in Memory. In *Proc. of the 16th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 833–842, New York, NY, USA, 2010. ACM.

[YK09] L. Ye and E. Keogh. Time Series Shaplets: A new Primitive for Data Mining. In *Proc. of the 15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 947–956, New York, NY, USA, 2009. ACM.

[YKJC14] F.X. Yu, S. Kumar, T. Jebara, and S.-F. Chang. On Learning with Label Proportions. *CoRR*, abs/1402.5902, 2014.

[YLG09] H. Yunhong, F. Liang, and H. Guoping. Privacy-Preserving SVM Classification on Vertically Partitioned Data without Secure Multi-party Computation. In *Proc. of the 5th Int. Conf. on Natural Computation (ICNC)*, volume 1, pages 543–546, Aug. 2009.

[YLK+13] F.X. Yu, D. Liu, S. Kumar, T. Jebara, and S.-F. Chang. $\propto$-SVM for Learning with Label Proportions. In *Proc. of the 30th Int. Conf. on Machine Learning (ICML)*, pages 504–512, 2013.

[YVJ06] H. Yu, J. Vaidya, and X. Jiang. Privacy-Preserving SVM Classification on Vertically Partitioned Data. In *Proc. of the 10th Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD)*, pages 647–656, Berlin, Heidelberg, 2006. Springer-Verlag.

[YYYA06] A. Youssef, M. Younis, M. Youssef, and A. Agrawala. Distributed Formation of Overlapping Multi-hop Clusters in Wireless Sensor Networks. In *IEEE Global Telecommunications Conf. (GLOBECOM)*, pages 1–6, Dec. 2006.

[ZDJW13] Y. Zhang, J. Duchi, M.I. Jordan, and M.J. Wainwright. Information-theoretic lower bounds for distributed statistical estimation with communication constraints. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 26, pages 2328–2336. Curran Associates, Inc., 2013.

[ŽPG16] I. Žliobaitė, M. Pechenizkiy, and J. Gama. An Overview of Concept Drift Applications. In Nathalie Japkowicz and Jerzy Stefanowski, editors, *Big Data Analysis: New Algorithms for a New Society*, pages 91–114. Springer International Publishing, 2016.

[ZRDZ07] J. Zhang, D. Roy, S. Devadiga, and M. Zheng. Anomaly detection in MODIS land products via time series analysis. *Geo-spatial Information Science*, 10(1):44–50, 2007.

[ZRLP14] Y. Zheng, S. Rajasegarar, C. Leckie, and M. Palaniswami. Smart car parking: Temporal clustering and anomaly detection in urban car parking. In *Proc. of the 9th IEEE Int. Conf. on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6, 2014.

[ZSGL07] K. Zhang, S. Shi, H. Gao, and J. Li. Unsupervised Outlier Detection in Sensor Networks Using Aggregation Tree. In *Proc. of the 3rd Int. Conf. on Advanced Data Mining and Applications (ADMA)*, pages 158–169, 2007.

[ZSS+16] Y. Zhao, R. Schwartz, E. Salomons, A. Ostfeld, and H.V. Poor. New formulation and optimization methods for water sensor placement. *Environmental Modelling & Software*, 76:128–136, 2016.

[ZY16] K. Zhou and S. Yang. Understanding household energy consumption behavior: The contribution of energy big data analytics. *Renew. Sust. Energ. Rev.*, 56:810–819, 2016.