PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

Mining and Learning in Sequential Data Streams: Interesting Correlations and Classification in Noisy Settings

Katsiaryna Mirylenka

Advisor:

Prof. Themis Palpanas	Paris Descartes University, France

Thesis Committee:

Prof. Alfons Kemper	Technische Universität München, Germany
Prof. Minos Garofalakis	Technical University of Crete, Greece
Prof. Yannis Velegrakis	Università degli Studi di Trento, Italy

April 2015

Acknowledgements

My PhD benefited greatly from the contribution of many people who helped me to realize who I am and supported my work with excellent advice.

First of all, I would like to express deep gratitude to my scientific advisor Professor Themis Palpanas for the chance that he gave me, for countless life and professional lessons, and for his continuous help and support in different steps of my PhD life.

I also thank my teacher of mathematics Galina Yulianovna Guscha, and my first scientific advisor Elena Nikolaevna Orlova for showing me the beauty and the power of mathematical techniques.

I am very grateful to my internship mentors – Divesh Srivastava, Graham Cormode, and Vassilis Christophides – for the great opportunities they opened to me, and for the very useful insights on how to do research.

I thank my dearest friends and colleagues, who encouraged me during the years of my PhD, especially all the Belarusian community in Trento, and in particular our friends Volha and Aliaksandr.

I am very grateful to my wonderful parents and brothers for their unconditional love and support and for being a great example of how one should pursue their studies and reach their goals. My particular gratitude goes to my husband Daniil for very interesting and productive discussions, for being always there for me, and for his strong love and care.

Abstract

Sequential data streams describe a variety of real life processes: from sensor readings of natural phenomena to robotics, moving trajectories and network monitoring scenarios. An item in a sequential data stream often depends on its previous values, subsequent items being strongly correlated. In this thesis we address the problem of extracting the most significant sequential patterns from a data stream, with applications to real-time data summarization and classification and estimating generative models of the data.

The first contribution of this thesis is the notion of Conditional Heavy Hitters, which describes the items that are frequent conditionally – that is, within the context of their parent item. Conditional Heavy Hitters are useful in a variety of applications in sensor monitoring, analysis, Markov chain modeling, and more. We develop algorithms for efficient detection of Conditional Heavy Hitters depending on the characteristics of the data, and provide analytical quality guarantees for their performance. We also study the behavior of the proposed algorithms for different types of data and demonstrate the efficacy of our methods by experimental evaluation on several synthetic and real-world datasets.

The second contribution of the thesis is the extension of Conditional Heavy Hitters to patterns of variable order, which we formalize in the notion of Variable Order Conditional Heavy Hitters. The significance of the variable order patterns is measured in terms of high conditional and joint probability and their difference from the independent case in terms of statistical significance. The approximate online solution in the variable order case exploits lossless compression approaches. Facing the tradeoff between memory usage and accuracy of the pattern extraction, we introduce several online space pruning strategies and study their quality guarantees. The strategies can be chosen depending on the estimation objectives, such as maximizing the precision or recall of extracted significant patterns. The efficiency of our approach is experimentally evaluated on three real datasets.

The last contribution of the thesis is related to the prediction quality of the classical and sequential classification algorithms under varying levels of label noise. We present the "Sigmoid Rule" Framework, which allows choosing the most appropriate learning algorithm depending on the properties of the data. The framework uses an existing model of the expected performance of learning algorithms as a sigmoid function of the signal-to-noise ratio in the training instances. Based on the sigmoid parameters we define a set of intuitive criteria that are useful for comparing the behavior of learning algorithms in the presence of noise. Furthermore, we show that there is a connection between these parameters and the characteristics of the underlying dataset, hinting at how the inherent properties of a dataset affect learning. The framework is applicable to concept drift scenarios, including modeling user behavior over time, and mining of noisy time series of evolving nature.

Keywords
streaming data; online algorithms; conditional heavy hitters; Markov chain modeling; event mining; finding interesting correlations; network data analysis; time series; Variable Markov Models; classification; sequential classifiers; classifier evaluation; handling noise; concept drift

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last decades there has been a dramatic explosion in the availability of streaming measurements that naturally appear from different sources, such as sensor networks of human activities, position updates of moving objects in location-based services, mobile calls, climatological processes, financial transactions and others. Such data sources produce billions of possibly noisy and temporally correlated values that arrive at a high rate, thus, making the tasks of pattern extraction, generative model estimation and sequential classification challenging.

In this chapter, we introduce three research directions that correspond to the problems of *i)* detecting interesting correlations, *ii)* estimating variable order generative models for sequential data streams, and *iii)* analyzing the performance of learning algorithms in noisy settings. Section 1.1 presents the general motivation for each direction. Then, in Section 1.2, we discuss the problem definitions and the contributions of this thesis. The structure of the thesis is given in Section 1.2.4. The publications based on the research made in this thesis are listed in Section 1.2.5.

1.1 Motivation

Due to its importance, the data stream model has received much attention in different research communities, such as data bases, data mining, networking, robotics and theory. There are many research problems that are addressed there, for example, data stream summarization [58], anomaly detection [21], data stream clustering [59] and classification [2]. In the majority of these works, adjacent observations of a data stream are treated as independent. In the recent works [35, 36] it has been demonstrated that it is beneficial to take into account the correlations between the neighboring observations for data modeling and related tasks that are based on similarity measurement among the parts of a data stream, such as pattern extraction, motif discovery, clustering and classification.

Streaming data are often noisy. Although the machine learning and data mining com-

munities have extensively studied the behavior of classifiers in different settings[62, 112], the effect of labeling noise on the classification task is still an open problem.

1.1.1 Detecting Interesting Correlations in Fast Data Streams

Within applications that generate large quantities of data, it is often important to identify particular entities that are associated with a large fraction of the data items [34, 84]. For example, in a network setting, we often want to find which users are responsible for sending or receiving a large fraction of the traffic. In monitoring updates to a large database table, it is important to know which attribute values predominate, for query planning and approximate query answering purposes. This notion has been abstracted as the idea of "heavy hitters" or "frequent items". Considerable effort has been spent on finding algorithms to track the heavy hitters under a variety of scenarios and data arrival models [7, 20, 25, 33, 78, 85, 88, 93].

However, the concept of heavy hitters can on occasion be quite a blunt one. Consider the network health monitoring scenario. It is well-known that in any measurement, there will be some destinations that are globally popular (search engines, social networks, video providers); likewise, so will be certain users (large organizations behind a single IP address, heavy downloaders and filesharers). As a result, tracking the heavy hitters within this data is not always informative, as they reveal only knowledge which is relatively slow changing, and not actionable. Rather, we would like to find which are sources or destinations that are significantly *locally* popular. That is, find those (source, destination) pairs where the destination is a heavy hitter amongst the connections of the same source.

Another application area is in security, in the intrusion detection domain. Here, a large number of different actions are observed, and the goal is to sift for unusual patterns of activity. The canonical approach is based around association rule and frequent itemset mining. These methods identify subsets of activities whose joint occurrence frequency exceeds some given threshold. While popular, this methodology has its limitations. The enormous search space implied by all possible combinations of actions typically requires a lengthy off-line search to identify the patterns of interest. While there are some on-line algorithms, these still require substantial resources to track sufficient statistics for the potentially frequent subsets. As a result, this kind of mining tends to be costly, and to deliver results significantly after the event.

Essentially, these two approaches (frequent items and frequent itemsets) fall at two ends of the spectrum: the frequent items approach is not rich enough to identify behavior of interest, while the frequent itemsets are potentially too rich, and too costly to find. In this thesis, we propose an intermediate goal, which focuses more on correlations between items than simple heavy hitters, but is lightweight enough to permit efficient streaming algorithms. We call this concept "Conditional Heavy Hitters", and work to provide

meaningful definitions and a suite of algorithmic approaches to find them.

1.1.2 Estimating Variable Order Generative Models for Sequential Data Streams

Conditional Heavy Hitters, introduced in the previous section, correspond to sequential patterns of fixed order (length). If a data stream contains correlated subsequences of varying length, Conditional Heavy Hitters are not enough to capture the peculiarities of the data stream.

There have been few works devoted to online statistical and probabilistic modeling of data streams, where correlations among the values are taken into account, for example, estimation of regression models [57] and Hidden Markov Models (HMMs); but they reflect only linear dependency among the observations or only first order correlations [63, 118]. Currently, estimation of more sophisticated probabilistic models, such as high order HMMs, Conditional Random Fields (CRFs), or Variable Order Markov Models (VMMs) requires much space in order to keep the exponentially large number of parameters of high orders, which is infeasible for large alphabets and streaming settings. Conditional Heavy Hitters, described in Chapter 3, are capable of approximately estimating the parameters of a high-order Markov chain with large alphabet, but keep the order of the model fixed.

In Chapter 4 we address the problem of online discovery of Variable Order Conditional Heavy Hitters (VOCHH) for data streams whose elements can be represented as symbols from large finite alphabets.

Specifically, we consider data that can be abstracted as pairs of temporally sorted symbols, which we refer to as a parent and a child. Parent consists of previously seen symbols, while a child usually corresponds to the most recent symbol. The fundamental concept of Conditional Heavy Hitters of variable order is to find those parent-child pairs, where the probability of the child is high, conditioned on the parent and the frequency of the parent-child pair is statistically significantly different from the i.i.d case of the distribution of the symbols. Variable order comes from the fact that parents can consist of variable number of symbols.

Streaming data from moving trajectories, sensors, web search, network monitoring, human activities, mobile wireless networks and text processing can naturally be modeled using variable order contexts, thus, they can be efficiently analyzed using VOCHH. VOCHH can be used to estimate the most important transition probabilities of the data generation models with markovian properties, such as Markov Models, HMMs, CRFs and others, in streaming environment. In other words, VOCHH extract the largest probabilities of moving from one state to another.

Even though any variable order Markov model can be described using a fixed-length Markov Chain of a sufficiently high order, the number of parameters that would need to

be stored is exponentially high. At the same time, to estimate all high order transition probabilities of a Markov Chain of a fixed order, much larger training datasets are needed. The variable order models estimate the parameters for only a small subset of meaningful contexts, thus, allocating space and using training datasets much more efficiently, which is crucial for streaming processing.

1.1.3 Analysis of Performance of Learning Algorithms in Noisy Settings

Conditional Heavy Hitters and VOCHH are useful for estimation of the generative model of the data stream in real time. The models can be exploited in the task of data stream classification. As streaming data often contain noise, it is important to study the behavior of classifiers in noisy settings.

Transforming vast amounts of collected – possibly noisy – data into useful information through the processes of clustering and classification has important applications in many domains. One example is preference detection of the customers for marketing. Another example is detection of negligent "providers" for Mechanical turk and reduction of their behavior on task results. The machine learning and data mining communities have extensively studied the behavior of classifiers in different settings (e.g., [62, 112]), however, the effect of noise on the classification task is still an important and an open problem.

The importance of studying noisy data settings is augmented by the fact that noise is very common in a variety of large scale data sources, such as sensor networks and the Web, and Machine learning algorithms, both classical and developed for sequential data, perform differently in the settings with varying levels of labeling noise. Thus, there rises a need for a unified framework aimed at studying the behavior of learning algorithms in the presence of noise, depending on the specifics of each particular dataset.

In Chapter 5, we study the effect of training set label noise[1] on a classification task. This type of noise is common in cases of *concept drift*, where a target concept shifts over time, rendering previous training instances obsolete. Essentially, in the case of concept drift, feature noise causes the labels of previous training instances to be out of date and, thus, equivalent to label noise. Drifting concepts appear in a variety of settings in the real world, such as the state of a free market, or the traits of the most viewed movie.

1.2 Problem Description and Contributions

1.2.1 Detecting Interesting Correlations in Fast Data Streams

In order to find interesting correlations, we model data that can be abstracted as pairs of items, which we refer to as *parent* and *child* items. The central concept of Conditional Heavy Hitters is to find those parent-child pairs that are most frequent, relative to the

[1] Throughout the rest of this thesis we will use the term *noise* to refer to the label noise, unless otherwise indicated.

frequency of the parent. The reason for referring to such heavy hitters as "conditional" is by analogy to conditional probabilities: essentially, we seek children whose probability is high, conditioned on the parent. These should be distinct from the parent-child pairs which are overall most frequent, since these can be found by using existing heavy hitter algorithms. While this is a natural goal, it turns out that there are several ways to formalize this, which we discuss in more detail in Chapter 3. Thus, the first challenge is to formalize the notion of Conditional Heavy Hitters. As we consider streaming data that consists of symbols from possibly large alphabets, the second challenge is to accurately extract Conditional Heavy Hitters in real time with limited space budget. To this end, we developed several streaming algorithms that use pruning based on the value of the conditional probability. The core structure of the algorithms, which keeps estimated statistics about the frequencies of paren-child pairs, depends on the simple characteristics of data. The characteristics correspond to the expected size of possible parents that have at least one heavy hitter child. If the expected size of parents is small we call such a dataset "sparse", otherwise, we refer to it as to a "dense" dataset.

Contributions. The main contributions of this research direction include:

- Definition of the concept of Conditional Heavy Hitters, which can be applied in a variety of settings;
- Theoretical and empirical comparison of Conditional Heavy Hitters with other notions of interesting elements studied in the data stream literature, such as frequent itemsets, association rules, and correlated heavy hitters;
- Development and description of several streaming algorithms for retrieving Conditional Heavy Hitters and analysis of their applicability for data with varying characteristics;
- The algorithms developed are evaluated on a mixture of real and synthetic datasets. We observe that certain algorithms can retrieve the conditional heavy hitters with high accuracy while retaining a compact amount of historical information. Different algorithms achieve the best results depending on simple characteristics of the data: essentially, whether the number of Conditional Heavy Hitters is comparable to the number of parents, or whether it is much lower.

1.2.2 Estimating Variable Order Generative Models for Sequential Data Streams

Studies [41, 74, 82, 104] show that for various domains, such as language modeling, correction of corrupted text, speech recognition, machine translation, protein models and models for web page prediction, VMM provide better fit than fixed order models and use much less space, which is crucial to our problem of modeling data from large alphabets.

In terms of statistical learning theory, the generalization guarantee (bias-variance tradeoff) of estimation of VMM depends on the degree of the variability of the con-

texts. This means that if the maximum order of the context is chosen, then, the fixed order model of this order leads to the smallest bias and the largest variance. The cause of the largest variance is the lack of evidence for estimating the parameters of the fixed-order model with enough confidence. Other words, fixed order model leads to the overfitting. On the other hand, using the smallest context information leads to underfitting, when bias is the largest and variance is the smallest. The main challenge of this work is to devise proper notion of variable order Conditional Heavy Hitters and develop online methods of their extraction in order to balance the bias-variance tradeoff in limited memory settings. To achieve this, we focus on the variable order modeling aimed at keeping all the important contexts that correspond to the significant dependencies of the data, and keeping the shortest possible contexts for the independent observations.

Our approach for VOCHH discovery exploits the variations of well known *lossless compression* algorithms [13] that are used for building an efficient representation of VMMs. Lossless algorithms with the best accuracy guarantees and at the same time the best performance originate in suffix trees or their analogs [22], [13]. After training they can be used for different purposes of data analysis:

1. extraction of essential connections between a symbol and a context – in order to find out the highest conditional probabilities of various orders.
2. forecasting – to predict future states of a data stream given a sequence of previous states;
3. similarity matching – to compare several parts of a data stream or several data streams based on their representation structures. This can be further used for classification and clustering tasks.

Lossless compression algorithms have several limitations: tree structures keep the information of all seen subsequences, thus, in case of large alphabets and, consequently, large number of states, the trees can grow up to the length of the stream and exceed memory budget. The algorithms that involve pruning, such as Probabilistic Suffix Trees, have running time quadratic in the length of the stream, thus making online usage impossible. Other pruning strategies proposed in the literature are either too simplistic, for example, based on the frequency of a subsequence, or require much time to reconstruct the representation tree like in Probabilistic Suffix Trees.

In this work we develop and assess several algorithms that adopt the trie structure of Prediction by Partial Match (PPM) algorithm – one of the best performing lossless compression algorithms. We transform the PPM trie into the VOCHH trie by using online pruning strategies based on the values of conditional probabilities, frequencies of the sequential patterns and their statistical significance. The developed algorithms are suitable for various settings if either higher precision or recall of the VOCHH extraction is preferred. On the other hand, their aim is to minimize the LogLoss of the estimated variable order model.

Contributions. Main contributions of this thesis along this research direction include:

- Definition of the concept of Conditional Heavy Hitters of Variable Order.
- Several streaming algorithms for retrieving VOCHH.
- Analysis of the developed algorithms, including quality guarantees with regard to the VOCHH estimation.
- The developed algorithms are evaluated on three real-world datasets. We observe that the algorithms can retrieve VOCHH with high accuracy while using restricted memory budget.

1.2.3 Analysis of Performance of Learning Algorithms in Noisy Settings

Giannakopoulos and Palpanas [53] have shown that the performance[2] of a classifier in the presence of noise can be effectively approximated by a *sigmoid* function, which relates the signal-to-noise ratio in training data to the expected performance of the classifier. We call this fact the "Sigmoid Rule". In Chaptercha:srf we examine how much added benefit we can derive from the sigmoid rule model, by studying and analyzing the parameters of the sigmoid in order to detect the influence of each parameter on the learner behavior. Based on the most prominent parameters, we define several dimensions characterizing the behavior of the algorithm, which can be used to construct criteria for the comparison of different learning algorithms. We name this set of dimensions the *"Sigmoid Rule" Framework (SRF)*. The dimensions of the "sigmoid rule" are useful for comparing algorithms, and can take into account different requirements of the user. We also study, using SRF, how dataset attributes (i.e., the number of classes, features and instances and the fractal dimensionality [39]) correlate with the expected performance of the classifiers in varying noise settings.

In this work we also show that the "Sigmoid Rule" is equally applicable to the problem of sequential classification, where classifiers of a different nature (for example, based on Hidden Markov Models) are considered. Sequential classifiers are useful for datasets where samples rely on causal phenomena, based on the sequence of appearance. We consider datasets where instances are the points of time series, causing the relationship between adjacent points. Using SRF, we additionally study the influence of the dataset characteristics on the performance of sequential classifiers. The dataset characteristics in this case are the number of classes, features and correlation order (a measure of how many recent instances strongly correlate with the next instance).

Contributions. In summary, we make the following contributions.

- We define a set of intuitive criteria based on the SRF that can be used to compare the behavior of learning algorithms in the presence of noise. This set of criteria provides both quantitative and qualitative support for learner selection in different settings.

[2]When we write *performance* of an algorithm, we mean the classification accuracy.

- We demonstrate that there is a connection between the SRF dimensions and the characteristics of the underlying dataset, using both a correlation study and regression modeling based on linear and logistic regression. In both cases we discovered statistically significant relations between SRF dimensions and dataset characteristics. Our analysis shows that these relations are stronger for the sequential classifiers than for the non-sequential, traditional classifiers, where the instances are considered to be independent.
- Our results are based on an extensive experimental evaluation, using 10 synthetic and 31 real datasets from diverse domains. The heterogeneity of the dataset collection validates the general applicability of the SRF for both non-sequential and sequential learners.

1.2.4 Thesis Structure

The thesis contains the following chapters:

Chapter 2 surveys background and the related work relevant to the research directions of this thesis. In more detail, we discuss the specifics of streaming model, consider relevant notions of interesting patterns proposed in the literature, survey lossless compression algorithms and discuss the methods that are used for the analysis of classifiers in noisy settings.

Chapter 3 is devoted to the problem of Detecting Interesting Correlations in Fast Data Streams. It describes the notion of Conditional Heavy Hitters and developed algorithms together with the analysis of their quality guarantees, as well as the experimental evaluation of the algorithms.

Chapter 4 presents our results along the direction of Estimating Variable Order Generative Models for Sequential Data Streams. We devise optimal pruning strategy to minimize LogLoss of the variable order model and describe its online variations. We also discuss the developed algorithms for VOCHH discovery and their quality guarantees. Finally, we experimentally evaluate the algorithms using three real datasets.

Chapter 5 contains the description of the Sigmoid Rule Framework, including the criteria that can be used to choose the optimal classifier in the noisy settings for a particular dataset. We also present the analysis of dependence of classifier's performance on the dataset characteristics. The analysis is done for both classical and sequential classifiers.

Chapter 6 presents concluding remarks and directions of future work.

1.2.5 Publications

Research presented in this dissertation resulted in the following publications:

- [35] Michele Dallachiesa, Besmira Nushi, Katsiaryna Mirylenka and Themis Palpanas. *Similarity Matching for Uncertain Time Series: Analytical and Experimental*

Comparison. Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Querying and Mining Uncertain Spatio-Temporal Data. ACM, 2013: 8–15 (QUeST '2011).

- [92] Katsiaryna Mirylenka, George Giannakopoulos Themis Palpanas. *SRF: A Framework for the Study of Classifier Behavior under Training Set Mislabeling Noise.* Advances in Knowledge Discovery and Data Mining. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012. 109-121 (PAKDD'2012).
- [37] Michele Dallachiesa, Besmira Nushi, Katsiaryna Mirylenka and Themis Palpanas. *Uncertain Time-Series Similarity: Return to the Basics.* Proc. VLDB Endow. VLDB Endowment. 2012 (VLDB'2012).
- [90] Katsiaryna Mirylenka, Graham Cormode, Themis Palpanas and Divesh Srivastava. *Finding Interesting Correlations with Conditional Heavy Hitters.* Proceeding of IEEE 29th International Conference on Data Engineering(2013): 1069–1080 (ICDE'2013).
- [89] Katsiaryna Mirylenka, Graham Cormode, Themis Palpanas and Divesh Srivastava. *Conditional heavy hitters: detecting interesting correlations in data streams.* The VLDB Journal. Springer Berlin Heidelberg. 2015: 1-20.

Chapter 2

Background and Related Work

In this thesis we address the problem of mining and learning in sequential data streams along the directions of finding important patterns of the fixed and variable order, and analysis of classifiers in noisy settings. We propose methods of finding important patterns and a framework for classifier analysis. In this chapter we provide the necessary background and related work for the formulated problems and the proposed methods. In Sections 2.1 we describe the data stream model. In Sections 2.2, 2.3 and 2.4 we consider relevant notions of interesting patterns proposed in the literature, survey lossless compression algorithms and discuss the methods that are used for the analysis of classifiers in noisy settings.

2.1 Data Streams and Their Specifics

The recent development of tools that generate and monitor data online has led to the necessity of processing of huge quantities of data in very short time. This gave rise to a new kind of data model that is called a *data stream*.

Data stream can be defined as an unbounded sequence (x_1, x_2, ..., x_n, ...) that is indexed on the basis of the time of its arrival at the receiver. Babcock et al. [8] point out some fundamental properties of a data stream system such as:

(a) data elements arrive continuously, sometimes with a different rate;

(b) there is no limit on the total number of points in the data stream;

(c) once processed, the element of a data stream can not be retrieved unless it was explicitly stored in the memory;

(d) the amount of the available memory is less than the size of the data stream;

(e) there is a limited processing time that can be devoted to each element in the stream. This time can be so small that some values may have to be thrown away (load shedding [1]).

In other words, a data stream is an ordered sequence of elements that needs to be processed online because otherwise it will be lost.

2.2 Detecting Interesting Correlations in Fast Data Streams

The notions of heavy hitters and frequent itemsets have been heavily studied in the database and data mining literature. Interest in finding the heavy hitters in streams of data goes back to the early eighties [17, 93], where simple algorithms based on tracking items and counts were developed. Thanks to the interest in algorithms for streams of data, improved methods have been developed over the course of the last decade. These include variants of methods that track items and corresponding estimated counts [40, 85, 88], and randomized "sketch" methods, capable of handling negative weights [25, 33]. These methods can all provide the guarantee that, given a parameter ϵ, they can find all items in a stream of length n that occur more than ϵn times, while maintaining a summary of size $O(1/\epsilon)$. Equivalently, they estimate the frequency of any given item with additive error ϵn. For further details and empirical comparison of methods, see the surveys [34, 84].

The heavy hitters are a special case of frequent itemsets: they are the frequent 1-itemsets. Additionally, all larger frequent itemsets consist of subsets of the heavy hitters. There has been much work dedicated to finding frequent itemsets (and their variations) in the off-line setting, often starting from the Apriori [3] and FP-Tree algorithms [64]. These concepts have been adapted to work over streams of data, resulting in algorithms such as FUP [24], and FP-stream [55]. A limitation of finding frequent itemsets is that the number of possibly frequent itemsets can become very large, meaning that the algorithm either has to track information about many candidates, or else aggressively prune the retained data, and risk missing out on some frequent itemsets. In formalizing Conditional Heavy Hitters, one aim is to form a compromise between heavy hitters (which are simple and for which space/accuracy tradeoffs can be provided) and frequent itemsets (which are much more complex, and for which no tight space guarantees are provided). Additional background on itemset mining in streams is given by Yu and Chi [127].

Several other variations of heavy hitters on streams have been proposed in the literature. Where the stream is time-varying, it is sometimes of interest to monitor only the heavy hitters within a recent time window, or with some other time-decay [32, 38, 79, 109]. The notion of hierarchical heavy hitters says that when items fall in a hierarchy (or combination of hierarchies), it is interesting to find nodes in the hierarchy that are heavy from aggregating their descendants [31]. Lastly, correlated aggregates consider streams of tuples, and ask for aggregates on some attributes, for the subset of tuples that meet some other conditions [52]. In this regard, our concept of Conditional Heavy Hitters can be seen as a generalized version of a kind of correlated aggregate, albeit one that has not been studied previously.

Our notion of Conditional Heavy Hitters is related to models of (temporal) correlation in data, as captured by Markov chains. That is, given a sequence of items, the kth order transition probabilities are defined as the (marginal) probability of seeing each character, given the history of the k prior characters. In our terminology, setting the child as the new character and the parent as the concatenation of the k previous characters means that finding the Conditional Heavy Hitters maps on to finding the high transition probabilities in this Markov chain. The importance of considering correlations has been recently motivated within several domains [35, 37, 80]. There has been much prior work on capturing correlations in data via different Markov-style models, such as homogeneous Markov chains of high order, hidden Markov Models [12], Bayesian networks [97] and others [80, 119]. However, fitting these increasingly complex models requires a lot of CPU and I/O time and multiple passes over the data, and hence it is infeasible to estimate them in a streaming setting. For example, the simple Mixture Transition Distribution [101] aims to approximate the transition probabilities with a smaller number of parameters, but requires multiple iterations over the data to do so. By focusing on the Conditional Heavy Hitters, we also identify a small number of parameters to describe the distribution, but can recover these efficiently in a single pass over the data.

Most related to this work is the paper of Lahiri and Tirthapura [77] which considers the problem of 'correlated heavy hitters' over a stream of tuples (a, b). Here, (a, b) is a correlated heavy hitter if a is a simple heavy hitter (frequency exceeds ψ) in a sequence of single-dimensional records and b is a heavy hitter in the subset of tuples where a appears. We discuss the similarity and differences of this definition to Conditional Heavy Hitters in Sections 3.2 and 3.3.

2.3 Estimating Variable Order Generative Models for Sequential Data Streams

As we have discussed before, Variable Order Conditional Heavy Hitters (VOCHH) can be seen as the main elements of the transition matrix of a Variable Order Markov Model, as they have highest conditional probability values and high support. Currently, lossless compression algorithms are the most efficient algorithms that can estimate VMMs. According to Bunton [22] in the hierarchy of model classes, most of the lossless compression algorithms lie in the FSMX (Finite State Machines X class of models), thus, these algorithms provide asymptotically optimal solutions with respect to the Markovian sources with Variable Order. The algorithms that belong to FSMX are based on suffix trees, and include general suffix trees [113], count suffix trees [66], probabilistic suffix trees [104]. FSMX also include algorithms that can be represented via suffix tress while possibly having different core structures, such as Prediction by Partial Matching (PPM) [27] and

its derivatives PPM-C [94] and PPM* [28]. In this section we consider theoretical and empirical properties of these algorithms together with another non-stochastic popular lossless compression algorithm LZ78 [129]. We discuss the PPM algorithm in detail in Section 2.3.4 because we use it as a foundation for our methods of VOCHH discovery.

2.3.1 Online Compression Algorithms

There are several types of compression algorithms [22]: (a) lossy and lossless; (b) online and multiple-pass; (c) stochastic and Ziv-Lempel-like. As we are interested in efficient and accurate VOCHH estimation for large finite-alphabet sources in streaming settings we focus on lossless, online and stochastic algorithms. Due to the very large source alphabets it is not feasible to keep the counts of all seen sequences in a tree. The least informative nodes and sequences should be pruned. Thus, eventually we cannot guarantee lossless compression. On the other hand, we cannot use lossy algorithms as they are developed for particular domains such as image or video compression or even text compression where prior knowledge about the data source is heavily exploited [121]. Currently, online stochastic modeling techniques arguably provide the best data compression among techniques known according to both theoretical and empirical studies [22], [13]. These results belong to the performance of the PPM family of techniques which we are going to discuss in more detail later in the chapter.

The work [13] surveys the most popular compression methods and evaluates the methods by their prediction accuracy for three different datasets of sequential nature, namely english text, music and protein (amino-acid) sequence datasets. The methods estimate VMM from a sequence of observed states. They build a model, based on a tree or a collection of trees in a way that, when we look up a "context" (preceding states), we get a prediction or a probability distribution for the next state.

General features of the compression algorithms according to Begleiter [13] are the following:

- A tree that is used to represent the estimated VMM is constructed during a training phase.
- Given the tree, conditional probabilities of any order (if the tree is not bounded) can be calculated. If the order is bounded and a query is longer than the maximum depth of the tree, there are several tricks to calculate the probability distribution, for example, the largest parent subsequence is used to estimate the probability of the sequence.
- The methods provide lossless compression of data. Having a good lossless algorithm, data series can be efficiently encoded with a small number of bits, proportional to the entropy of prediction, which also means high prediction quality of the model. Thus, lossless compression can be considered as summarizaion of the data series.

Lossless compression algorithms usually do not use any pruning and do not assume

uniformity of VMMs (in Section 4.1 we discussed uniform and nonuniform models where the variable length depends only on a context or on a full sequence consisting of a context and a symbol).

The algorithms that perform the best according to the study [13] are the following:

1. PPM-C – the Prediction by Partial Matching, Method C, proposed in [94].

2. For the protein classification problems the best algorithm is LZ-MS [96], which is a modification of the Lempel-Ziv compression algorithm LZ78 [129]. It is shown in the work [10] that the running time of LZ78 can be improved if it is represented via suffix tree.

These algorithms and their master algorithm – the general suffix tree algorithm [113] – together with their variations are described in Table 2.1. It is also shown in the study [83] that usage of the prefix and suffix structures is the most beneficial for sequential pattern mining.

In the description of the algorithms we use the following notation:

- Training dataset consists of n strings (or trajectories) of total length N;
- D – the maximum order of the model, which shows the maximum length of dependency among the neighboring observations. The order of the model corresponds to the length of the context or a parent that is used for modeling time correlations;
- m – the length of the query string [1], that is the string whose probability we need to find.
- z – the number of occurrences of this string in the tree,
- r – the number of resulting LZ78 factors for the corresponding algorithm.
- Σ – a finite alphabet defining the possible values of the sequence elements.

The full VOCHH-related notation is listed in Appendix A.3.

According to the comparison and accuracy results provided in [13] and [22], PPM-C is the best performer, while LZ78 and PST, in addition to the lower accuracy, have larger space and time requirements for the calculation of conditional probabilities.

VOCHH can be extracted using the tree structures obtained after training a lossless compression algorithm. Many lossless compression algorithms build either a suffix or a prefix tree (trie) in linear time, and can thus be used for real time applications, if space pruning is implemented efficiently.

[1] We use the terms *string*, *sequence* and *trajectory* interchangeably in the thesis, unless specified otherwise.

2.3.2 Pruning

As mentioned above, in the case of large alphabets and bounded space, the main issue is to efficiently prune the nodes that are not promising in a sense that they have low potential to become Conditional Heavy Hitters. Pruning strategies are not naturally embedded into the lossless compression algorithms, although several pruning strategies, such as *least recently used* or *least frequently used*, are described in the litereture. The most popular and the easiest strategy is to fully rebuild a tree when memory budget is spent [49]. One of the reason why full model reconstruction is preferred is that for other strategies there is a need to keep additional pointers to track the least recently/frequently used sequences.

Study of the VMM usage for predicting web pages navigation [41] introduces the confidence-based pruning, which checks if there is a difference in conditional probabilities of several children given a parent. If all possible children are equiprobable, it is safely assumed that elimination of such a parent will not change the model. This strategy is not useful in the case of very large alphabets as it is almost impossible to observe a parent with all possible states following it.

In the work [74], in which the back off model is used, the quality of a model after pruning is measured by the improved average KL divergence, which consists of the sum of terms:

$$d(y^k, \sigma) = Pr_{full}[y^k, \sigma] \log \frac{Pr_{full}[\sigma|y^k]}{Pr[escape|y^k] P_{full}[\sigma|y^{k-1}]}$$

for those sequences (y^k, σ) that were not observed in a training set. σ is a symbol from Σ, and probabilities $Pr_{full}[\cdot]$ are defined by the suffix tree. The work [74] uses another kind of pruning strategy in order to minimize the improved KL divergence with the full model: it aims to remove those elements from the full model that have the lowest values of $d(y^k, \sigma)$. All the nodes with the lowest values of the criteria are successively removed until the tree reaches the predefined size. If the node is decided to be removed, all its successor nodes are also removed. Due to heavy computation this pruning strategy can only be used offline.

As VMMs of high orders have large number of parameters, all related works consider mainly *offline* pruning strategies: when an exact model is already built (or all exact probabilities are estimated) some parts of the model or some probabilities with the smallest impact on the model are discarded. Even strategies like "least frequently used" are usually assessed in the offline fashion, though they can be used online as well.

2.3.3 Probabilistic Suffix Trees versus PPM-C

Probabilistic Suffix Trees. The most significant work in machine learning domain that is also a lossless compression algorithm is a variation of a suffix tree algorithm, which is called Probabilistic Suffix Tree (PST). PST estimates Probabilistic Suffix Automata (a

Table 2.1: Lossless compression algorithms and their properties.

#	Name	Main structure	Running time		Space	Pruning		Applications	Real-time usage	Interesting facts
			Training	Search	requirments	Apriori	Aposteriori			
1.0	Suffix trees	suffix tree	$O(n+N)$	$O(m+z)$	$O(N)$	truncation: max string length is defined - space complexity $O(min(N\|\Sigma\|^D + Dn)$	strings with characteristics that can be expressed as a non-decreasing function, can be pruned from the tree, without increasing space or time complexity, for example suffix frequency	text-editing, free-text search, computational biology	yes, small alphabets	all the algorithms below are the variation of Suffix trees, also different PPM and LZ78 algorithms can be implemented via suffix tree structure
1.1	Count suffix trees	suffix tree	$O(n+N)$	$O(m+z)$	$O(N)$	specific to text, exploit linguistical features : regarding only the suffixes that are useful in a linguistic sense - syllabification, stemming, and non-word detection		text	yes, small alphabets	for frequency based pruning, strategies for probability estimation: maximum overlap (MO) and others from work [66] can be used
1.2	PST	suffix tree + pruning during construction	$O(D \cdot N^2)$	$O(D)$	$O(D \cdot N)$, though lower than traditional case	symbols are pruned if marginal probability i threshold, ratio to the sequence with shorter suffix has comparable probability			no, due to quadratic time	accuracy is not very good, but when backoff model for zero-frequency problem is used, the performance is better (closer to PPM) [72]
2.0	PPM	trie, for bounded order model	$O(N)$	$O(\|parent\|^2)$	$O(D \cdot N)$		frequency-based pruning, conditional probability based pruning possible online	text, DNA	yes, small alphabets	can be represented as a suffix tree, best performer
2.1	PPM*	trie, with unbounded order model	$O(N)$	$O(\|parent\|^2)$	$O(N)$, much larger space requirements than 2.0	no	no	text, DNA	yes, small alphabets	can be represented as a suffix tree, unbounded keeps all information about the sequence, linear in space, bad compression as keeps everything, overestimates local Markov model
2.2	PPMC	trie with inexact frequency counting	$O(N)$	$O(\|parent\|^2)$	$O(D \cdot N)$	Frequency or probability based pruning cannot be used as only the counters of the leaves are exactly kept		text, DNA	yes, small alphabets	very fast as only leaf nodes are updated
3	LZ78	trie	$O(N)$	$O(n\sqrt{N} + r \log N)$	$O(n)\sqrt{N}+r$	no	no	text	yes, small alphabets	not stochastic approach

subclass of Probabilistic Finite Automata that is used to model transitions between the finite number of states). The PST algorithm has PAC-style performance guarantees [104]: for some small δ and ϵ it is proven that with the probability $1 - \delta$ the Kullback-Leibler divergence between a true and an estimated distributions, after normalization by the length of the string, is less than ϵ.

The PST algorithm is not a real time algorithm, as it first extracts all the subsequences bounded by the maximal order D with their exact counts, then calculates the maximal likelihood estimates for all transitional probabilities, and finally uses some refinements to keep only significant ones. These refinements can be seen as offline pruning strategies. Each candidate subsequence (y, x), which corresponds to a new node or a sequence of nodes, is examined for two conditions:

1. joint probability of the context and the symbol should be larger than the predefined threshold. This threshold can be converted to a frequency threshold;
2. the context y is meaningful if there is a symbol σ such that $P(\sigma|y) > threshold$;
3. the context y contributes additional information for predicting σ relative to its longest suffix y' (if $y = (x_1, x_2, ..., x_k)$ then its longest suffix $y' = (x_2, x_3..., x_k)$). This means that the ratio of their conditional probabilities is significantly different from one:

$$\frac{P(\sigma|y)}{P(\sigma|y')} \geq 1 + \varepsilon \quad \text{or} \quad \frac{P(\sigma|y)}{P(\sigma|y')} \leq \frac{1}{1+\varepsilon}, \tag{2.1}$$

where $\varepsilon > 0$.

All the thresholds depend on ϵ, δ, the size of the alphabet, and length of the strings used for training. The procedure for building PST has quadratic time complexity ($O(D * N^2)$), and the space complexity $O(D * N)$, where N is the length of the training data, and $O(D)$ is the time required to find $P(\sigma|y)$.

Several additional offline pruning criteria for PST were used in the work [82]. The first two pruning strategies from PST are adopted as they are, while the last strategy that checks if a node brings additional information compared to its parent is reformulated. In the previous work [104], difference from a parent suffix is checked separately for each symbol using the equation 2.1, while in the paper [82] the difference is checked using Kullback-Leibler divergence between distributions $\hat{P}(\cdot|\sigma y)$ and $\hat{P}(\cdot|y)$. If the difference is smaller than a threshold, then the node is pruned. This criterion is efficient only for small alphabets, as the Kullback-Leibler divergence is calculated when distribution over all the symbols is defined.

Another pruning criterion adopted in the work [82] is based on Akaike's Information Criterion (AIC), which should be minimized by the best model. AIC seeks the balance between the complexity of the model and its fit. Complexity of the model is represented by the number of its parameters, while its fit is calculated using the likelihood function of the model. The pruning is done on two levels. First, maximal model depth D is chosen by

testing fixed order Markov Chains of different orders, then model with the minimal AIC is considered as a starting point for further pruning. On the next level, each parent-child pair is considered in order to choose the model based on either the parent or the son. According to the results on the protein data [82], using the AIC criterion results in the significantly smaller size of the PST, as well as the improved quality of prediction. AIC still needs offline processing due to the multiple passes over the data.

In most of the lossless compression algorithms that are used for prediction[2] so called "zero-frequency problem" is faced. Zero-frequency problem corresponds to those cases where the sequence for which the probability should be predicted has not been seen in the training dataset. To address zero-frequency problem ($P[\sigma|y] = 0$), the PST approach uses a small, user-defined probability to be assigned to it: $P[\sigma|y] = p_{min}$. This part is additional and it does not participate in PAC learning guarantees.

Interestingly, though the PST algorithm has good theoretical properties, it did not perform well in the empirical studies [13], [72], which compared different lossless compression algorithms for the purpose of building VMM models. In fact, PST was not among the three best performers.

Prediction by Partial Matching. PPM and especially its variation PPM-C perform the best. PPM-based algorithms can be represented as a suffix tree [22] and PST and PPM keep essentially the same information as PST, but in different data structures. The main difference between these two algorithms is the implementation of zero-frequency problem for sequences that have not been seen in a training set. While PST uses minimal probability threshold, PPM uses a complex mechanism of escape and exclusion, which has very good empirical performance, despite having no theoretical guarantees. This escape mechanism in also called back-off technique. The technique can be formulated as Equation 2.3.4, where $Pr(escape|y)$ is different for different implementations, but they depend on the number of symbols seen after y – Σ_y, a context y and probabilities of seen symbols.

The back-off technique helps to define a model completely on demand when an estimation of the probability of an unseen sequence is needed. Parameter $Pr(escape|s)$ is needed in order to get distributions to sum to unity. Thus, in PPM algorithms the probability of unobserved symbols is not fixed and depends on the number of observed symbols and the frequency of a context, while in PST this minimal probability is the same for any context. It is demonstrated in the work [72] that usage of similar mechanism for PST improved its performance greatly.

As we focus on the streaming settings, we build our approach based on the best performing PPM algorithm, as it has training time complexity $O(n)$ and has the potential for online pruning. The main challenge of our approach is to tackle PPM space complexity O(D*n), which is infeasible for the streams with large alphabets. For prediction we use

[2]The equivalence of problems of lossless compression and prediction of discrete sequences is proven in the work [48]

the best performing back-off model implemented in PPM-C version of PPM algorithm which is described below.

2.3.4 Prediction by Partial Marching

In this section we describe in more detail the PPM compression algorithm and its PPM-C variation that we will use as the basis of VOCHH estimation. The main parts of the algorithm are the following [13]:

Input:

(1) D – the maximal order of dependency (maximum context of VOCHH parent), and, at the same time, the maximal depth of the tree which contains the model information;

(2) *Escape* and *exclusion* mechanisms, which are used to solve the zero-frequency problem for the contexts which were not seen during training. There are various PPM algorithms depending on different escape and exclusion mechanisms. In general, escape mechanism works as follows, for all the contexts y with the order $k \leq D$ we need to allocate a certain probability mass $\hat{P}_k(escape|y)$ for all symbols that did not appear after the context y. The remaining probability mass $1 - \hat{P}_k(escape|y)$ is shared among symbols with non-zero frequencies. The probability mass allocation differentiates PPM algorithms, nevertheless, all algorithms follow the recursive equation following [13]:

$$\hat{P}(x|y) = \begin{cases} \hat{P}_k(x|y), \text{if } Fr(y,x) \neq 0 \\ \hat{P}_k(escape|y) \cdot \hat{P}r(x|y'), \text{ otherwise.} \end{cases} \tag{2.2}$$

The *exclusion* mechanism is based on the fact that probabilities of observed symbols given a context can be estimated more accurately if they are calculated using just the observed set of symbols rather than all possible symbols.

Learning phase:

PPM algorithm constructs a trie T based on the data stream $s = (x_1, x_2, ..., x_N)$. Each node in T corresponds to a symbol from the alphabet Σ and has a counter. The depth of the trie T is bounded by the order D and is less or equal to $D + 1$. The root node of T corresponds to an empty symbol. The algorithm incrementally parses the training sequence, one symbol at a time. Each symbol and its context of the length up to D define a path in T. The counter of any node with the corresponding path (y, x) is $Fr(y, x)$, where y is the parent or the context of a path, and x is the child or the last symbol; The learning phase requires linear time $O(N)$ to be finished. The space required for the worst case trie is $O(D \cdot N)$.

There are several types of trie updates that are used in PPM variations:

1. Full updates, when all counts of all sequences including the intermediate sequences in a trie are updated;

2. Update Exclusion (used in PPM-C) – only the frequencies that correspond to the maximum suffix-tree frontier are updated. The frequencies are updated if it is a new symbol of a parent, or it is a maximum frontier (length) symbol;
3. MaxOrderUpdates – counts are updated only when a new symbol is a maximum order state. For the descendants all the counts are smaller than real, similarly to the previous case.

The last two types of updates lead to faster training, but make the trie inconsistent for pruning, as all the non-leaf nodes have counts smaller than their real frequencies.

As in the best-performer – PPM-C algorithm – no pruning was implemented, during the training phase not all counts of the nodes corresponding to a training path were updated, only the leaf nodes have valid up-to-date counts. As we would like to implement further pruning, we use the standard PPM full update procedure.

Testing/prediction phase:

When a trie T is built (the building can be continuous and a testing sequence can be included in the trie) it encodes the probability distribution of variable order of the training dataset. Probabilities $\hat{P}(x|y)$, $|y| \leq D$ can be calculated by traversing the trie along the longest suffix of y, denoted as y', such that (y', x) are fully contained in T as a path from the root to a leaf. Then the counters along the path are used to calculate the probabilities using the following equations:

$$\hat{P}_k(x|y) = \frac{Fr(y,x)}{|\Sigma_y| + \sum_{x'' \in \Sigma_y} Fr(y,x'')}, \quad if \quad x \in \Sigma_y; \tag{2.3}$$

$$\hat{P}_k(escape|y) = \frac{|\Sigma_y|}{|\Sigma_y| + \sum_{x'' \in \Sigma_y} Fr(y,x'')}, \quad otherwise, \tag{2.4}$$

where $|y| = k$, Σ_y is the set of symbols seen so far in the context of y. The straightforward computation of $\hat{P}(x|y)$ using T requires $O(|y|^2)$ time [13]. This escape mechanism is considered among the best [13] in practice. At the same time, according to Witten and Bell [122] there is no principle justification among different escape mechanisms.

2.3.5 Model Validation: LogLoss

Here we discuss the goodness of fit methods for estimated VMM models, which are used to see how well the built model describes initial data. As we already mentioned before, the compression algorithms for VMM estimation are validated using LogLoss on the testing dataset [13]. LogLoss is defined as follows:

$$l(\hat{P}, x^k) = -\frac{1}{k} \sum_{i=1}^{k} \log \hat{P}(x_i|x_1, x_2, ..., x_{i-1}). \tag{2.5}$$

The best model has minimum LogLoss, as a small LogLoss means that the probability that a sequence was generated by a model is high, thus, the model fits the data well. For

LogLoss computation there is no need to assess the full probability distribution from the trie including the cases that are not encoded in the trie. The conditional probability is estimated only for the current testing sequence. LogLoss which is also called cross-entopy loss or logistic regression loss is defined on the estimated probabilities, and is widely used to evaluate probability outputs of prediction and classification tasks, for example, it is used as one of the loss optimization functions by the winners of the KDD Cup Orange Challenge in 2009 [42] and KDD Cup 2004 [116] and for model evaluation in the work [128]. The authors of the work [107] show that if models are well calibrated, LogLoss is the highest-performing selection metric if little data is available for model selection.

There is a direct connection between LogLoss, lossless compression, and entropy. The quantity $-\log_2 Pr(x_t|x_1, ..., x_{t-1})$ is called "self-information", it is the compression or code length of symbol x_t and it is measured in bits per symbol. The expected value of self-information is the entropy. So, LogLoss is the average of the self-information for each symbol in a test sequence. The average LogLoss is the average compression rate of a sequence, given the estimated distribution $\mathbb{P}$ [13]. The equivalence of the LogLoss minimization and compression is shown in the works [103], [102]. This validation fits the case where we are interested in the prediction of the next points given the context.

Sometimes, for the model validation, similarity measures between several distributions, such as Kullback-Leibler divergence or Jensen-Shannon divergence, are used. For their calculation it is required to estimate all the probabilities of the model, even those probabilities which are not estimated because not all possible contexts are seen during the training phase. Since we are interested in detecting only strong correlations , the measures above cannot do us justice, since they value the detection of weak correlations equally. As we would like to concentrate only on the largest conditional probabilities given the variable order contexts, validation metrics similar to logloss are more preferable.

2.3.6 Comparison with Other Notions of Interesting Correlations

Among the notions that describe elements of interest in the streaming data series are Heavy Hitters, Frequent Itemsets, Correlated Heavy Hitters, Association Rules and others that are compared with the Conditional Heavy Hitters of the fixed order and discussed in Section 3.3. Association Rules is the closest notion to the Conditional Heavy Hitters of Variable Order, as both of them focus on the pairs that have high frequency and high conditional probability of occurrence. At the same time, a parent in the pair has variable length.

The main differences between Association Rules and VOCHH are the following:

1. In Association Rule world there is no single global sequence, the patterns are discovered in terms of subsequences. This also means that there is no single generative model of the data;

2. Sequential Association Rules do not try to find Markov style patterns, though there is sequentiality there. In general, Association Rules are used to find the correlation phenomena among the items, but they are not devised to make direct predictions [106].
3. There may be time gaps in the timestamps of items in the rule. In one transaction there is no sequential relationship between the items. In contrast, the VOCHH approach assumes that all the observations are sequential.

Studies devoted to the discovery of Association Rules can be divided into four main branches:

(a) Original Association rules, which operate through the thresholds on support and confidence [3], as discussed in Section 3.3;
(b) Streaming algorithms for the online discovery of Association Rules [67];
(c) Sequential Association Rules, that take into account order of the itemsets [65, 106]. The order might be with the gaps in the timestamps, thus the parts of the rule may happen not in the adjacent transactions;
(d) Association Rules with statistical foundation, where statistical significance of the rule or the set of rules is taken into account [61, 73].

Our VOCHH approach can be seen as a product of all the branches of Association Rules but with Markov sequentiality and a single generative model.

2.4 Analysis of Performance of Learning Algorithms in Noisy Settings

Since 1984, when Valiant published his theory of learnable [114], setting the basis of Probably Approximately Correct (PAC) learning, numerous different machine learning algorithms and classification problems have been devised and studied in the machine learning community. Given the variety of existing learning algorithms, researchers are often interested in obtaining the best algorithm for their particular tasks. The algorithm selection is considered to be a part of the meta-learning domain [56].

According to the No-Free-Lunch theorems (NFL) described in the work [125] and proven in the works [123, 124], there is no overall best classification algorithm, given the fact that all algorithms have equal performance on average over all datasets. Nevertheless, we are usually interested in applying machine learning algorithms to a specific dataset with certain characteristics, in which case we are not limited by the NFL theorems. As mentioned in the work [5], the results of NFL theorems hint at comparing different classification algorithms on the basis of dataset characteristics, which is the aim of our work in this research direction.

Concerning the measures of performance that help distinguish among learners, in [5]

the authors compared algorithms on a large number of datasets, using measures of performance that take into account the distribution of the classes, which is a characteristic of a dataset. The Area Under the receiver operating Curve (AUC) is another measure used to assess machine learning algorithms and to divide them into groups of classifiers that have statistically significant difference in performance [18]. In all the above studies, the analysis of performance has been applied for the datasets without noise, while we study the behavior of classification algorithms in noisy settings. In our study we use the fact shown by G. Giannakopoulos and T. Palpanas [53, 54] about concept drift, which illustrates that a sigmoid function can accurately describe performance in the presence of varying levels of label noise in the training set. In an early influential work [75], the problem of *concept attainment* in the presence of noise was also indicated and studied in the STAGGER system. In this thesis, we analytically study the sigmoid function to determine the set of parameters that can be used to support the selection of the learner in different noisy classification settings.

The behavior of machine learning classifiers in the presence of noise is also considered in the work [70]. The artificial datasets used for classification were created on the basis of predefined linear and nonlinear regression models, and noise was injected in the features, instead of the class labels as in our case. Noisy models of non-markovian processes using reinforcement learning algorithms and the methods of temporal difference are analyzed in the paper [98]. In the work [26], the authors examine multiple-instance induction of rules for different noise models. There are also theoretical studies on regression algorithms for noisy data [111] and works on denoising, like [81], where a wavelet-based noise removal technique was shown to increase the efficiency of four learners. Both noise-related studies [81, 111] dealt with noise in the attributes. However, we study label-related noise and do not consider a specific noise model, which is a different problem. The label-related noise corresponds mostly to the concept drift scenario, as was also discussed in the introduction 1.2.3.

Paper [95] presents a study on the effect of different types of noise on the precision of supervised learning techniques. They address this issue by systematically comparing how different degrees of noise affect supervised learners that belong to different paradigms. They consider 4 classifiers, namely the Naïve Bayes classifier, the decision tree classifier, the IBk instance-based learner and the SMO support vector machine. They test them on a collection of data sets that are perturbed with noise in the input attributes and noise in the output class, and observe that Naïve Bayes is more robust to noisy data than the other three techniques. The performance was evaluated on 13 datasets. However, no general framework was proposed for modelling the effect of noise on the performance of the classifiers. In contrast, we propose the Sigmoid Rule Framework for comparing the behavior of classifiers. We also introduce different dimensions that can be used to formulate the criteria for comparing the algorithms depending on user needs. Moreover,

we also apply this framework to sequential classifiers, which were not considered in above studies.

Sequential data and especially time series data collected from sensors are often affected by noise. That is, the quality of the data may be decreased by errors due to limitations in the tolerance of the measurement equipment. Sequential classification based on Markov chains with noisy training data is considered in the work of Dupont [44]. The paper shows that smoothed Markov chains are very robust to classification noise. The same paper discusses the relation between the classification accuracy and the test set perplexity that is often used to measure prediction quality. However, unlike our work, no specific formalization is proposed for analyzing the effect of different noise levels on the performance of the classifier. Development of such formalization is becoming increasingly important as sequential and time series data classification are applied in diverse domains, such as financial analysis, bioinformatics and information retrieval. For example, a recent survey [126] considers various methods of sequential classification in terms of methodologies and application domains. The methods for sequential classification we focus on are introduced in the works [99], [108] and [46], which describe Hidden Markov models, Conditional Random Fields, and Recurrent Neural Networks correspondingly. The techniques described in this thesis, such as Conditional Heavy Hitters from Chapter 3, allow estimating sequential models in real time. The efficiency of such techniques enables the applicability of sequential classification to sensor data analysis and moving trajectories.

Chapter 3

Streaming Conditional Heavy Hitters

In this chapter we tackle the problem of finding interesting correlations in data streams. We have introduced the notion of Conditional Heavy Hitters in Section 4.5.1. In this Chapter, first, we discuss the motivating applications of Conditional Heavy Hitters in Section 3.1 and describe necessary preliminaries in Section 4.5.1. Then, in Section 3.3, we discuss similarities and differences of Conditional Heavy Hitters with other notions of interesting elements studied in the data stream literature, such as frequent itemsets, association rules, and correlated heavy hitters. In Section 3.4 we present several streaming algorithms for detecting Conditional Heavy Hitters and analyze their applicability for data with varying characteristics. In Section 3.5 the developed algorithms are evaluated on a mixture of real and synthetic datasets. The algorithms can detect Conditional Heavy Hitters with high accuracy while retaining a small amount of historical information. Furthermore, these algorithms are able to process the streams arriving at a high speed. In Section 3.5.6 we also show the ability of extracted Conditional Heavy Hitters to accurately estimate the Markov Model of moving trajectories.

The work presented in this chapter has been published as a conference paper [90] and as a journal paper [89].

3.1 Motivating Applications

Equipped with the concept of Conditional Heavy Hitters, we can apply it to a variety of settings:

– In a network monitoring setting, we can look for the conditional heavy hitters over packets within the network, where for each packet the source address is considered as the parent, and the destination address as the child. In this case the Conditional Heavy

Hitters identify those destinations which occur most frequently for their corresponding sources. This can be useful in identifying local trends, shifts in popularity, and traffic planning, especially when several sources are found to share the same child as a conditional heavy hitter.

– When modeling many real systems and processes, it is common to use a Markov chain to capture the transition behavior between states. However, many of the systems have large numbers of possible states (e.g. modeling traffic flow in a large city), and so it can be costly to maintain complete statistics. Instead it suffices if we can just measure the most important observed transition probabilities from state to state from a stream of state occupancies. That is, we identify the large probabilities of moving from one state to another – these are exactly the Conditional Heavy Hitters, with the parent being the current state, and the child being the transition taken. Thus, tracking the Conditional Heavy Hitters over a long sequence of observations of state transitions can capture the essential parameters of the Markov chain.

– Caching and pre-fetching are widely used in Web-based systems. Their aim is to reduce the time latencies perceived by users when navigating the Web. Advanced caching and pre-fetching policies use probability graphs, access trees and markov-chain models [100], in order to predict the least/most probable objects to be accessed next, or in the near future. In this case Conditional Heavy Hitters can efficiently capture the main transitions from previous sequence of objects to the next, thus providing existing policies with the most relevant real-time probabilities to enable them to make better predictions.

– Within a database management system or data warehouse, there are a large number of transactions which affect the overall data distribution. Such systems commonly keep statistics on individual attributes, to capture the number of distinct values, frequent items, and so on. Given the large number of columns, it is infeasible to keep detailed statistics on all combinations of columns. However, a suitable compromise is to keep some summary statistics on pairs of columns which commonly co-occur (in join paths, say). Finding the Conditional Heavy Hitters within such pairs captures information about the correlations between them, and can allow improved selectivity estimation.

3.2 Preliminaries

To allow our definition of Conditional Heavy Hitters to be generally applicable, we assume that the input can be modeled as a stream of pairs of adjacent (parent, child) values $(\mathbf{p}, c)$. A parent $\mathbf{p}$ can be a single symbol, or a sequence of adjacent symbols in the stream, while a child is a single symbol.

Definition 1 (Frequencies)**.** *Given a stream of (parent, child) pairs whose ith element is* $(\mathbf{p}_i, c_i)$*, the frequency of a parent* $\mathbf{p}$*,* $f_{\mathbf{p}}$*, is defined as*

$$f_{\mathbf{p}} = |\{i : \mathbf{p}_i = \mathbf{p}\}|.$$

The frequency of a (parent, child) pair, $f_{\mathbf{p},c}$*, is defined as*

$$f_{\mathbf{p},c} = |\{i : \mathbf{p}_i = \mathbf{p} \wedge c_i = c\}|.$$

From these frequencies, we can define (empirical) probabilities associated with items and pairs.

Definition 2 (Probabilities)**.** *Given a stream of n (parent, child) pairs, the empirical probability of a parent* $\mathbf{p}$*,* $\Pr[\mathbf{p}]$*, is defined as* $\Pr[\mathbf{p}] = f_{\mathbf{p}}/n$*. The joint probability of a parent-child pair,* $\Pr[\mathbf{p}, c]$*, is defined as* $\Pr[\mathbf{p}, c] = f_{\mathbf{p},c}/n$*. The conditional probability of a child given a parent,* $\Pr[c|\mathbf{p}]$*, is defined as*

$$\Pr[c|\mathbf{p}] = \frac{\Pr[\mathbf{p}, c]}{\Pr[\mathbf{p}]} = \frac{f_{\mathbf{p},c}}{f_{\mathbf{p}}}.$$

We can now define a first notion of conditional heavy hitters.

Definition 3 (Conditional Heavy Hitter)**.** *We say that a pair* $(\mathbf{p}, c)$ *is a conditional heavy hitter with respect to a threshold* $0 < \phi < 1$ *if* $\Pr[c|\mathbf{p}] \geq \phi$*.*

This definition has the advantage of clarity and simplicity. However, on further consideration, there are some drawbacks associated with this formulation. Firstly, observe that when a parent is rare, it is more likely to generate Conditional Heavy Hitters. As a clear example, consider the case of a parent $\mathbf{p}$ that occurs only once in the stream. Then we have $f_{\mathbf{p},c} = f_{\mathbf{p}} = 1$ for the associated child c, and so $\Pr[c|\mathbf{p}] = 1$, making it automatically a conditional heavy hitter. While this is a valid application of the definition—the (empirical) conditional probability of this child truly is 1—we might still object that this is not a particularly significant association, due to the limited support of this item. Second, for related reasons, the total number of conditional heavy hitters meeting this definition can be large. Specifically, in an extreme case a given parent $\mathbf{p}$ can have $\Theta(1/\phi)$ distinct children which are all conditional heavy hitters. So if there are $|P|$ distinct parents, there can be a total of $\Theta(|P|/\phi)$ distinct Conditional Heavy Hitters—a very large amount—many of which may be infrequent and uninformative.

A natural way to avoid these issues is to place an additional constraint on the frequency of the parent, thus limiting the number of parents which can contribute to Conditional Heavy Hitters. One solution would be to additionally require that $\Pr[\mathbf{p}] > \psi$ for $(\mathbf{p}, c)$ to form a conditional heavy hitter (similar to [77]). Certainly, this has the desired effect: the number of conditional heavy hitters can now be at most $\Omega(1/\phi\psi)$, and parents with very small count can no longer contribute Conditional Heavy Hitters. However, we argue

that this definition is overly restrictive: it restricts attention to only those parents who are ψ-heavy hitters, and so misses those pairs which may have a significant correlation despite a lower total frequency. Other formulations, such as requiring $\Pr[(\mathbf{p}, c)] > \phi\psi$ have similar drawbacks. Consequently, we set up a different requirement as our goal.

Definition 4 (Popular conditional heavy hitter)**.** *A pair* $(\mathbf{p}, c)$ *is a* popular *conditional heavy hitter if it is a conditional heavy hitter with respect to* ϕ*, and it ranks among the top-*τ *of the Conditional Heavy Hitters, ordered by* $f_{\mathbf{p},c}$*.*

This says that we seek parent-child pairs that are conditional heavy hitters, with a preference to those that have a higher occurrence within the observed data. In realistic data sets, we may expect that there will be many Conditional Heavy Hitters with large $f_{\mathbf{p},c}$ values, which will ensure that we avoid the trivial case of $f_{\mathbf{p},c} = f_{\mathbf{p}} = 1$. Consequently, this represents a workable definition, that avoids this unwanted case, while also avoiding ruling out interesting cases.

Given this definition, it is not possible to provide algorithms which guarantee to always find exactly those items counted as popular Conditional Heavy Hitters while also using small space, as shown by the following lemma:

Lemma 1. *Any (randomized) one-pass algorithm which promises to find all popular Conditional Heavy Hitters must use* $\Omega(\min(n, |P|))$ *space, where* n *is the length of the stream, and* $|P|$ *is the number of distinct parents.*

Proof. Consider a stream of items, $x_1, x_2, \ldots x_n$, and suppose we have an algorithm which finds popular Conditional Heavy Hitters. From this stream, we generate a new stream of parent-child pairs, $(x_1, 0), (x_2, 0), \ldots (x_n, 0)$. Then each distinct pair is a conditional heavy hitter: $P[0|p] = 1$. Thus, the algorithm must find the top-τ most frequently occurring parents. But observe that these correspond exactly to the top-τ most frequently occurring items in the original stream. It has previously been shown that accurately solving this problem requires space $\Omega(\min(n, U))$ over a set of U possible items, even just to find the most frequent item [6], which gives the claimed lower bound. □

In some cases, $|P|$ is not so large, and so we can look for algorithms which use this much space. In other cases, $|P|$ may be very large, and this amount of space is impractical. However, this bound should not cause us to abandon hope of finding methods which are effective in practice. The kinds of sequences which are used to construct the worst-case examples in the lower-bounds are very artificial, where all items occur only once or twice within the data, forcing any correct algorithm to keep enough information to distinguish which items occur more than once. Realistic data is more varied, and so there is more evidence spread throughout the stream to help identify the conditional heavy hitters.

Our goal will be to design algorithms which allow us to estimate the conditional probability of parent-child pairs accurately. That is, the goal is find an estimate $\widehat{\Pr}[c|\mathbf{p}]$ that

accurately estimates $\Pr[c|\mathbf{p}]$. From this, we will be able to find candidate conditional heavy hitters. Having the candidate Conditional Heavy Hitters we can also extract the popular Conditional Heavy Hitters. Our experimental study evaluates the ability of these algorithms to find such Conditional Heavy Hitters.

3.3 Elements of Interest in a Data Stream

The problem of detecting "interesting" elements in data streams has attracted a lot of attention in the recent years. There can be many different interpretations of what makes an element of interest, varying across different applications. Consequently, several different notions have been proposed that are relevant to our study. In each case, there has been at least one paper describing algorithms for each of the following notions

(a) frequent items, or Heavy Hitters [34, 84];

(b) Frequent Itemsets [64], which are the foundation of

(c) Association Rules [3];

(d) Correlated Heavy Hitters [77];

(e) elements with the highest Pointwise Mutual Information (PMI) [45], and

(f) Conditional Heavy Hitters [90].

In this section, we formally consider each notion, or problem definition and study the relationships among them. For consistency with the rest of the thesis, we consider streams of pairs $\mathbf{p}$ and c.

Notion 1. The *Frequent Itemsets* (of size two) are the $(\mathbf{p}, c)$-pairs that occur in the data stream more often than a given support threshold: $f_{\mathbf{p},c} > \phi_0$, $\phi_0 > 0$.

Note that *Heavy Hitters* (HHs) of the stream can be considered as frequent itemsets of size one. Conceptually, the frequent itemsets are equivalent to the heavy hitters if every $(\mathbf{p}, c)$-pair is modeled as a single element.

Notion 2. A $(\mathbf{p}, c)$-pair, or implication $\mathbf{p} \Rightarrow c$, is an *Association Rule* [3] if $f_{\mathbf{p},c} > \phi_1$ (support), $\phi_1 > 0$, and $\Pr[c|\mathbf{p}] > \phi_2$ (confidence), $0 < \phi_2 < 1$.

Notion 3. *Correlated Heavy Hitters* [77] are the pairs $(\mathbf{p}, c)$, where $f_{\mathbf{p}} > \phi_3$, $\phi_3 > 0$ and $\Pr[c|\mathbf{p}] > \phi_2$, $0 < \phi_2 < 1$.

Notion 4. A pair $(\mathbf{p}, c)$ is considered to have the highest *Pointwise Mutual Information* [45] if it is among the top-σ ($\sigma \in \mathbb{Z}_{>0}$) pairs ranked using $PMI(\mathbf{p}, c) = \log \dfrac{\Pr[\mathbf{p}, c]}{\Pr[\mathbf{p}] \Pr[c]}$.

Notion 5.1 A pair $(\mathbf{p}, c)$ is a *Conditional Heavy Hitter* [90] if $\Pr[c|\mathbf{p}] > \phi_2$, $0 < \phi_2 < 1$.

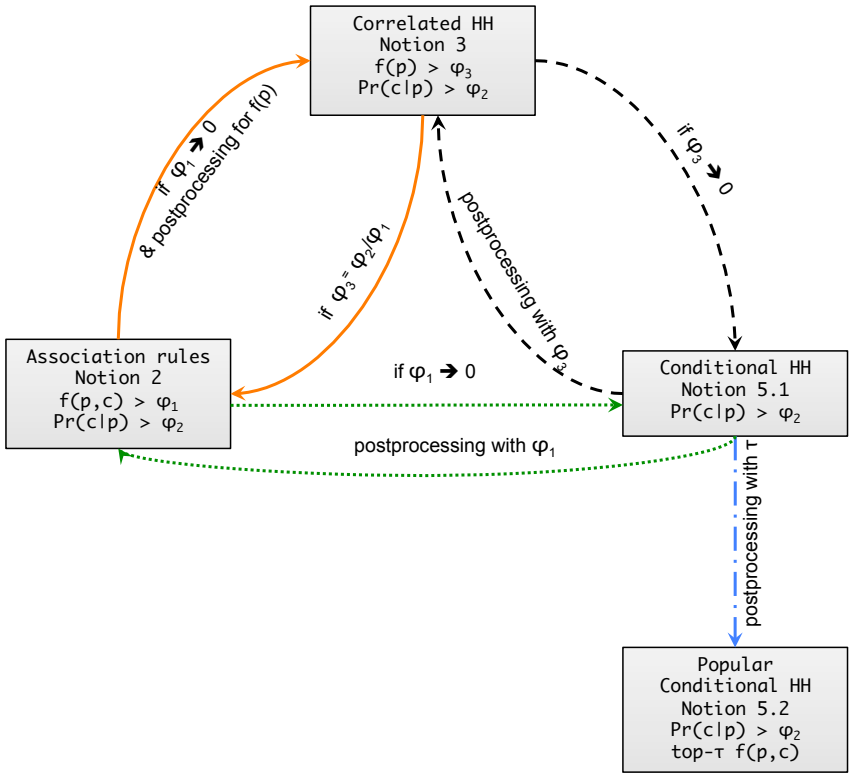


Figure 3.1: Comparison of various notions of interesting data stream elements.

Notion 5.2 The pair $(\mathbf{p}, c)$ is considered to be a *Popular Conditional Heavy Hitter* [90] if it is a conditional heavy hitter and it ranks among the top-τ ($\tau \in \mathbb{Z}_{>0}$) of the highest Conditional Heavy Hitters sorted by $f_{\mathbf{p},c}$.

All the thresholds, ϕ_i, $i = 0, 1, 2, 3$, σ, and τ, are user-defined.

We note that we consider frequent itemsets and association rules only for the case of pairs of items. The general problem of frequent itemsets is not considered here as it involves additional challenges; mainly that of pruning non-promising itemsets of varying length, creating an exponentially large search space. Moreover, we point out that Notion 1 (frequent itemsets, or heavy hitters) and Notion 4 (pointwise mutual information) are very different from the others, because both these notions treat the two elements of the pair equally, thus, not taking into account the sequential nature of their relationship within the data stream. Notion 1 suffers this limitation due to its simplicity, while Notion 4 treats $\mathbf{p}$ and c symmetrically and finds those pairs where both elements occur together more often than if their occurrences were independent. Therefore, in the following we elaborate on the other three notions, which consider the sequential nature of the pair. Figure 3.1 sketches these notions and the relations between them.

Association rules (Notion 2) and correlated heavy hitters (Notion 3) are very similar as they both consider the conditional probability of the child given the parent. Notion 2

has an additional constraint on the frequency of the parent-child pair, while Notion 3 has a constraint on the frequency of the parent. Note that given the output of any algorithm for association rules, where $\phi_1 > \varepsilon$ with ε sufficiently close to zero, we can filter out those items where $f_{\mathbf{p}} > \phi_3$ and obtain the same results as correlated heavy hitters.

Lemma 2. *The output of the correlated heavy hitters algorithm produces the output of the association rules algorithm if* $\phi_3 = \phi_2/\phi_1$.

Proof. If the pair $(\mathbf{p}, c)$ is a correlated heavy hitter, it satisfies the condition $\Pr[c|\mathbf{p}] > \phi_2$. To qualify as an association rule the pair $(\mathbf{p}, c)$ should also satisfy the condition:

$$f_{\mathbf{p},c} > \phi_1 \tag{3.1}$$

Consider the definition of the conditional probability $\Pr[c|\mathbf{p}] = \Pr[\mathbf{p}, c]/\Pr[\mathbf{p}] = f_{\mathbf{p},c}/f_{\mathbf{p}}$. We then have $f_{\mathbf{p},c} = \Pr[c|\mathbf{p}] \cdot f_{\mathbf{p}}$. Since $(\mathbf{p}, c)$ is a correlated heavy hitter, $f_{\mathbf{p}} > \phi_3$ and $f_{\mathbf{p},c} = \Pr[c|\mathbf{p}] \cdot f_{\mathbf{p}} > \phi_3 \cdot \phi_2$. If $\phi_3 = \phi_2/\phi_1$ then condition (3.1) is satisfied. □

The transformations needed to derive association rules from correlated heavy hitters and vice versa are depicted by the solid (orange) arrows in Figure 3.1.

Association rules (Notion 2), correlated heavy hitters (Notion 3) and Conditional Heavy Hitters (Notions 5.1 and 5.2) all use a threshold for the conditional probability. (Note that it is possible to use the same algorithms for Conditional Heavy Hitters and popular Conditional Heavy Hitters; in the latter case, additional ordering and pruning by the frequencies of parent-child pairs is applied.) Conditional heavy hitters do not have any additional conditions on the frequencies of the elements, hence by filtering the results based on the frequencies of parents, or the frequencies of parent-child pairs, the results of Conditional Heavy Hitters may be transformed to correlated heavy hitters or association rules, respectively. Likewise, if the thresholds on frequencies ϕ_1 and ϕ_3 are set to a sufficiently small value $\varepsilon > 0$, the results of association rules and correlated heavy hitters algorithms can be used to find the Conditional Heavy Hitters. These relationships are shown using the dashed (black) and dotted (green) arrows in Figure 3.1.

These connections show that a solution set for certain of the notions identified above can be manipulated to provide solutions for others also. However, there are overheads in simply trying to apply an algorithm for one notion to solve another: there are additional time and space costs, and it may be necessary to modify the parameters used substantially to obtain the desired output. Furthermore, the quality guarantees offered by a particular algorithm are specific to the notion and thresholds that it targets. In most cases, these guarantees do not translate into guarantees for different notions. For example, if we use the algorithm of correlated heavy hitters for Conditional Heavy Hitters by setting $\phi_3 = 0$, the space guarantees of the algorithm (which depend on ϕ_3) no longer hold. In Section 3.5.2, we further study the relationships among heavy hitters, Conditional Heavy

Table 3.1: Main characteristics of the proposed algorithms

Algorithm	Parents	Summary structure	Eviction order
GlobalHH	all	global	parent-child frequency
ParentHH	all	local	parent-child frequency
CondHH	all	global	conditional probability
FamilyHH	partial	global	parent-child frequency
SparseHH	partial	global	conditional probability

Hitters, correlated heavy hitters and association rules by experimentally evaluating their behavior in different settings.

The focus in the rest of this paper is on Conditional HH, which share the threshold for conditional probability with AR and Correlated HH, but do not use any additional thresholds. The guaranties for Conditional HH are focused on the accuracy of estimation of the conditional probability, rather than on the other costs, while the other approaches do not take this into consideration and focus more on a set of pairs, which satisfy the corresponding constraints.

3.4 Algorithms for Conditional Heavy Hitters

In this section, we describe a variety of algorithms to help us identify the Conditional Heavy Hitters within a stream of data. These are summarized in Table 3.1. We begin with algorithms for the traditional heavy hitters problem, and adapt these to identify those which are conditional heavy hitters.

3.4.1 GlobalHH Algorithm

Our first algorithm aims to identify all parent-child pairs that occur frequently, so that we can extract the subset that are Conditional Heavy Hitters. For this task, we make use of existing algorithms to find the heavy hitters from a stream of items. The *SpaceSaving* algorithm by Metwally *et al.* [88] has been widely used for this problem. Given an amount of space s, it guarantees to find all items in a stream of length n which occur more than n/s times (and, moreover, to provide an estimate of their frequency which is accurate up to an n/s additive error). For streams which obey a frequency distribution that follows a long-tail distribution, formalized as a Zipfian distribution, it further guarantees to provide accurate recovery of the head of the distribution. The algorithm behaves very well in practice, finding accurate estimates of frequencies of items across a variety of data sets [34, 84]: it exhibits the most stable behavior among all heavy hitter algorithms.

The algorithm works as follows: it maintains a collection SS of k items and associated counts. For simplicity, assume that the structure is initialized with k arbitrary items with

count 0. For each item x seen in the stream, if it is currently stored in the collection, the associated count $\hat{f}_x$ is incremented. Otherwise, we find the item with the current smallest count in the collection, and replace it with the new item, then increment its count. At any time, we can estimate the frequency of any item x with the associated count in the collection $\hat{f}_x$ if the item is stored, and 0 otherwise.

We formalize the GlobalHHalgorithm in pseudocode in Algorithm 1. Given each $(\mathbf{p}, c)$ pair in the stream, we insert it into the SS structure (lines 3-13). We also separately maintain information on the frequency of each parent (lines 1, 4). In this first algorithm, we assume that there is sufficient space to store information on all parents, which means we have the exact $f_{\mathbf{p}}$ values. If the SS structure is full (line 8), we eliminate the element with the lowest count from the structure and store its frequency in variable m (line 9). Then, a new element is inserted in SS with frequency $m + 1$ (line 11).

To identify the Conditional Heavy Hitters and the popular Conditional Heavy Hitters, we iterate over all items in the SS structure in decreasing order of their counts. For each stored $(\mathbf{p}, c)$ pair, we compute its estimated $\hat{f}_{p,c}$ value, which is contained in the SS structure and denoted as $SS[(\mathbf{p}_i, c_i)]$ in Algorithm 1. Then, we calculate the corresponding conditional probability $T[(\mathbf{p}_i, c_i)] = SS[(\mathbf{p}_i, c_i)]/f_{\mathbf{p}_i}$, and test whether $T[(\mathbf{p}_i, c_i)] > \phi$. If this condition is true, we output this pair as a conditional heavy hitter. The top-τ such pairs are the popular Conditional Heavy Hitters. We refer to this algorithm as the *GlobalHH* algorithm, since it is based on finding the parent-child pairs which are global heavy hitters.

The SS structure is implemented as a min heap. Since its operations, namely Insert and Delete Minimal element, can be implemented with $O(\log s)$ time complexity, the running time of the GlobalHH algorithm for processing each new element of the stream is $O(\log s)$.

Lemma 3. *Given space $O(s+|P|)$, the GlobalHH algorithm guarantees that each candidate $(\mathbf{p}, c)$ pair output will have $\Pr[c|\mathbf{p}] \leq \widehat{\Pr}[c|\mathbf{p}] \leq \Pr[c|\mathbf{p}] + \frac{1}{s \Pr[\mathbf{p}]}$. When the distribution of $(\mathbf{p}, c)$ pairs follows a Zipfian distribution with parameter $z > 1$, the error bound is improved to $\frac{1}{s^z \Pr[\mathbf{p}]}$.*

Proof. Since the $f_{\mathbf{p}}$ values are found exactly, the uncertainty in the estimated conditional probability, $\widehat{\Pr}[c|\mathbf{p}]$ is due to the error from the SS algorithm. This guarantees that our estimate of $f_{\mathbf{p},c}$ is an overestimate by at most n/s for arbitrary streams. We output $\hat{f}_{p,c}/f_{\mathbf{p}}$, which overestimates by at most $\frac{n}{sf_{\mathbf{p}}} = \frac{1}{s \Pr[\mathbf{p}]}$. Therefore, we have the bound stated. This guarantees to overestimate the conditional probability, and so will ensure good recall. Alternately, we could provide an underestimate of the conditional probability by using a lower bound on the estimate of $f_{\mathbf{p},c}$. In this case, we ensure good precision,

Algorithm 1 GlobalHH for Conditional Heavy Hitters

Input: Data stream $D = \{(\mathbf{p}_i, c_i), i = 1, 2, ...\}$.
Output: SS - SpaceSaving structure of length s for parent-child pairs

```
1: f_{p_i} = 0, i = 1, 2, ..., |P| - keeps the frequency of each parent
2: m = 0 - highest frequency of a pair removed from SS
3: for each element (p_i, c_i) of D do
4:    f_{p_i} = f_{p_i} + 1
5:    if (p_i, c_i) ∈ SS then
6:       SS[(p_i, c_i)] = SS[(p_i, c_i)] + 1
7:    else
8:       if |SS| ≥ s then
9:          m = SS.deleteMin()
10:      end if
11:      SS.insert((p_i, c_i), m + 1)
12:   end if
13: end for
```

but do not guarantee recall. For streams with Zipfian frequency distribution, the error bound is tightened to ns^{-z} [88], improving the error bound to $\frac{1}{s^z \Pr[\mathbf{p}]}$ as claimed. □

3.4.2 ParentHH Algorithm

Our second algorithm takes a parent-centric view of the problem. Again, making the assumption that it is feasible to retain information about each distinct parent observed, we consider the case of keeping information about the set of children associated with each parent. That is, we keep a separate instance of the SS structure for each distinct parent. Clearly, this can use a lot of space, but will allow very accurate recovery of Conditional Heavy Hitters. We call this the *ParentHH* algorithm, since it retains heavy hitter information for each parent.

Algorithm 2 describes ParentHH. For each parent $\mathbf{p}$ observed in the stream, we maintain an instance of the $SS_{\mathbf{p}}$ structure of size $s/|P|$, dedicated to the children c that arrive as part of a pair for this $\mathbf{p}$ (line 2). For each pair $(\mathbf{p}, c)$ that arrives, we insert c in the corresponding $SS_{\mathbf{p}}$ (line 7).

The output of the ParentHH algorithm, T, which is the set of the s highest Conditional Heavy Hitters, is calculated using the SS_P structure in the following way: for each parent $\mathbf{p}_i$ the corresponding $SS_{\mathbf{p}_i}$ is retrieved. All the pairs $(\mathbf{p}_i, c_i)$, where $c_i \in SS_{\mathbf{p}_i}$ are placed in the answer set T with the corresponding conditional probabilities $T[(\mathbf{p}_i, c_i)]$ are set equal to $SS_{\mathbf{p}_i}(c_i)/f_{\mathbf{p}_i}$. In order to recover the Conditional Heavy Hitters given a threshold ϕ, we consider the set T in decreasing order of the conditional probabilities, and output $(\mathbf{p}, c)$ as an estimated conditional heavy hitter if $\hat{f}_{p,c}/f_{\mathbf{p}} \geq \phi$. The first τ such pairs are popular Conditional Heavy Hitters.

Algorithm 2 ParentHH for Conditional Heavy Hitters

Input: Data stream $D = \{(\mathbf{p}_i, c_i), i = 1, 2, ...\}$.

Output: $SS_{\mathbf{p}_i}$ - SpaceSaving structure of length $s/|P|$ for parent-child pairs assigned to each parent $\mathbf{p}_i$, $i = 1, 2, ..., |P|$.

1: $f_{\mathbf{p}_i} = 0, i = 1, 2, ..., |P|$ - keeps the frequency of each parent
2: $m_{\mathbf{p}}$ - highest frequency of a pair removed from $SS_{\mathbf{p}}$
3: $m_{\mathbf{p}_i} = 0, i = 1, 2, ..., |P|$
4: **for** each element $(\mathbf{p}_i, c_i)$ of D **do**
5: $\quad f_{\mathbf{p}_i} = f_{\mathbf{p}_i} + 1$
6: $\quad$ **if** $c_i \in SS_{\mathbf{p}_i}$ **then**
7: $\quad\quad SS_{\mathbf{p}_i}(c_i) = SS_{\mathbf{p}_i}(c_i) + 1$
8: $\quad$ **else**
9: $\quad\quad$ **if** $|SS_{\mathbf{p}_i}| \geq s/|P|$ **then**
10: $\quad\quad\quad m_{\mathbf{p}_i} = SS_{\mathbf{p}_i}.deleteMin()$
11: $\quad\quad$ **end if**
12: $\quad\quad SS_{\mathbf{p}_i}.insert(c_i, m_{\mathbf{p}_i} + 1)$
13: $\quad$ **end if**
14: **end for**

The reintroduction strategy in this algorithm is similar to the GlobalHH algorithm considered above, but in this case, the $SS_{\mathbf{p}}$ structure and reintroduction frequencies $m_{\mathbf{p}}$ are specific for each distinct parent. In our implementation $SS_{\mathbf{p}}$ is realized using a heap, and all heaps are kept in a hash table with a parent ID as a key. Since in this case we consider a fixed amount of parents, the operations on the hash table take constant time and the time required by the ParentHH algorithm for processing a new element from the stream is $O(\log s)$.

Lemma 4. *Given space $O(\min(s, n))$ (for $s > |P|$), the ParentHH algorithm guarantees that each candidate $(\mathbf{p}, c)$ pair output will have*

$$\Pr[c|\mathbf{p}] \leq \widehat{\Pr}[c|\mathbf{p}] \leq \Pr[c|\mathbf{p}] + \frac{|P|}{s}.$$

Proof. From the $SS_{\mathbf{p}}$ structure, we obtain an estimate of $f_{\mathbf{p},c}$ which has error proportional to the number of items passed to the structure, which is $f_{\mathbf{p}}$, the number of occurrences of $\mathbf{p}$. So the amount by which $\hat{f}_{p,c}$ overestimates is at most $f_{\mathbf{p}}|P|/s$. When we estimate $\widehat{\Pr}[c|\mathbf{p}]$, the error is $(f_{\mathbf{p}}|P|/s)/f_{\mathbf{p}} = |P|/s$. The space bound follows immediately: it is bounded by n, since each item in the stream can increase the number of tuples stored by at most a constant amount. Using this overestimate favors recall, at the cost of precision. It is possible to instead use an underestimate of $f_{\mathbf{p},c}$, by subtracting an appropriate amount. In this case, we obtain good precision, but without guarantees on recall. □

Clearly, this algorithm provides accurate estimated conditional probabilities, but at a cost: we devote up to $s/|P|$ space for each parent, which seems excessive for parents that

turn out to be relatively infrequent (and hence their children are unlikely to appear as true Conditional Heavy Hitters).

3.4.3 CondHH Algorithm

Our third algorithm also keeps a summary structure similar to the previous algorithms, but with a different goal. Instead of using the absolute frequency to determine which items to retain detailed information for, we use their (estimated) conditional probability. Since this aligns with the overall goal, it may lead to more accurate behavior. We call this algorithm *CondHH* since it treats the conditional probability as a first class citizen.

In the CondHH algorithm (which is written out in pseudocode in Algorithm 3), we keep a collection of $(\mathbf{p}, c)$ pairs in the CSS (line 3) structure, along with a count for each pair. The algorithm proceeds similarly to GlobalHH: for each $(\mathbf{p}, c)$ pair that arrives, it checks if it is stored in CSS, and if so, it increases its associated count. If it is not stored, then it evicts some $(\mathbf{p}, c)$ pair from CSS, and replaces it with the new pair. Under the GlobalHH semantics, we would apply the SS algorithm, and evict the pair with the least frequency. But in the CondHH algorithm, we find the pair with the lowest conditional probability, i.e. with the smallest value of $\widehat{\Pr}[c|\mathbf{p}] = \hat{f}_{p,c}/f_{\mathbf{p}}$, and evict it (line 20). The algorithm also keeps track of the maximum value of $\hat{f}_{p,c}$ for all children of parent $\mathbf{p}$ that have been deleted so far; this value is stored in $m_{\mathbf{p}}$ (line 2). When we need to remove an old pair $(\mathbf{p}', c')$ from the data structure in order to insert a new pair, we update $m_{\mathbf{p}'}$ if needed, and insert the new pair $(\mathbf{p}, c)$ with an estimated count $\hat{f}_{p,c} = m_{\mathbf{p}} + 1$ (line 22).

The set of Conditional Heavy Hitters and their probabilities can then be calculated on demand as follows: for each $(\mathbf{p}_i, c_i) \in CSS$ its conditional probability is $T[(\mathbf{p}_i, c_i)] = CSS[(\mathbf{p}_i, c_i)]/f_{\mathbf{p}_i}$. The Conditional Heavy Hitters and the popular Conditional Heavy Hitters are then computed in the same way as in the algorithms described earlier.

Directly implementing this algorithm could be slow, due to the need to find the item with the lowest $\widehat{\Pr}[c|\mathbf{p}]$ on each eviction. However, this can be made fast by keeping the data in an appropriate data structure. Specifically, we can index the stored parent-child pairs in a two-level data structure. For each parent, we keep its children stored in sorted ascending order of $\hat{f}_{p,c}$. This can be maintained efficiently using data structures based on doubly-linked lists as described in [88]. Then the parents are stored in sorted order of $\min_c(\hat{f}_{p,c})/f_{\mathbf{p}}$, i.e. the estimated conditional probability of their least frequent child, via a standard data structure such as a heap or search tree if we need also fast search. This structure means that we can quickly find the parent-child pair with the smallest overall estimated conditional probability, based on the observation that for each parent we only need to consider the probability of its least frequent child.

Whenever a child frequency is increased, we can quickly update the estimated $\hat{f}_{p,c}$, and ensure that the sortedness condition on the children is maintained. This update also

Algorithm 3 CondHH for Conditional Heavy Hitters

Input: Data stream $D = \{(\mathbf{p}_i, c_i), i = 1, 2, ...\}$.

Output: CSS - Conditional SpaceSaving structure of length s for parent-child pairs

1: $f_{\mathbf{p}_i} = 0, i = 1, 2, ..., |P|$ - keeps the frequency of each parent
2: $m_{\mathbf{p}_i} = 0$ - highest frequency of a pair removed from CSS per parent $\mathbf{p}_i$, $i = 1, 2, ..., |P|$
3: **for** each element $(\mathbf{p}_i, c_i)$ of D **do**
4: $f_{\mathbf{p}_i} = f_{\mathbf{p}_i} + 1$
5: **if** $(\mathbf{p}_i, c_i) \in CSS$ **then**
6: $CSS[(\mathbf{p}_i, c_i)] = CSS[(\mathbf{p}_i, c_i)] + 1$
7: **else**
8: **if** $|CSS| \geq s$ **then**
9: $m_{p_j}^{candidate} = CSS.deleteMinCondProb()$
10: $m_{p_j} = \max(m_{p_j}, m_{p_j}^{candidate})$
11: **end if**
12: $CSS.insert((\mathbf{p}_i, c_i), m_{\mathbf{p}_i} + 1)$
13: **end if**
14: **end for**

affects $f_{\mathbf{p}}$, and so may also require us to move the parent around to restore the heap property on $\min_c(\hat{f}_{p,c})/f_{\mathbf{p}}$. Likewise, whenever a child of $\mathbf{p}$ is removed from this structure (because it is chosen for eviction), its successor in the sorted order of $\hat{f}_{p,c}$ becomes the new least frequent child of $\mathbf{p}$, which may also require restoring the heap property. At the same time we can update $m_{\mathbf{p}}$. Thus, implementing this algorithm requires a constant number of pointer operations ($O(1)$), and a constant number of heap or tree operations per update (each taking time $O(\log s)$), leading to an overall time complexity of $O(\log s)$ per element.

Lemma 5. *The CondHH algorithm guarantees that each candidate* $(\mathbf{p}, c)$ *pair output will have*

$$\Pr[c|\mathbf{p}] \leq \widehat{\Pr}[c|\mathbf{p}] \leq \Pr[c|\mathbf{p}] + \frac{m_{\mathbf{p}} + 1}{f_{\mathbf{p}}}.$$

Proof. For each parent $\mathbf{p}$, $m_{\mathbf{p}}$ is an upper bound on the maximum value of $\hat{f}_{p,c}$ of a $(\mathbf{p}, c)$ pair that has been deleted. Inductively, this is also an upper bound on any $f_{\mathbf{p},c}$ deleted $(\mathbf{p}, c)$ pair: this is true initially when $m_{\mathbf{p}} = f_{\mathbf{p},c} = 0$ for all $(\mathbf{p}, c)$ pairs, and is maintained by every operation. Therefore, we have that whenever any new pair is inserted with $\hat{f}_{p,c} = m_{\mathbf{p}} + 1$, we have that $0 \leq f_{\mathbf{p},c} \leq \hat{f}_{p,c} = m_{\mathbf{p}} + 1$, and hence (while it remains in the data structure) $0 \leq \hat{f}_{p,c} - f_{\mathbf{p},c} \leq m_{\mathbf{p}} + 1$. Consequently we have $0 \leq \widehat{\Pr}[c|\mathbf{p}] - \Pr[c|\mathbf{p}] \leq (m_{\mathbf{p}} + 1)/f_{\mathbf{p}}$. As with the previous algorithms, this tends to overestimate the true conditional probability, leading to higher recall, but weaker precision. This can be reversed by manipulating the estimate of $\hat{f}_{p,c}$, by subtracting $m_{\mathbf{p}} + 1$, to obtain a lower bound on $f_{\mathbf{p},c}$ and hence $\Pr[c|\mathbf{p}]$. □

Note that in general this bound might not be very strong: we may see cases where $m_{\mathbf{p}} + 1 = f_{\mathbf{p}}$, and so we do not obtain a useful guarantee. However, in practice we expect to obtain useful guarantees for the popular Conditional Heavy Hitters.

3.4.4 FamilyHH Algorithm

All the algorithms proposed above make the assumption that we can track detailed information for each parent (such as $f_{\mathbf{p}}$, heavy hitter children for each $\mathbf{p}$, etc.). However, this assumption is not always reasonable. For example, in the network traffic example in Section 1.1.1, the number of parents is equal to the number of possible children (both are equal to the number of IP addresses, which is 2^{32} under IPv4). In some cases the number of parents actually observed in the data will be small enough to track exactly. But in general, we should also provide algorithms for when this is not so.

Our next algorithm generalizes GlobalHH, and so keeps sparse information about both parents and children by maintaining just the heavy hitter parents, and the heavy hitter parent-child pairs. So instead of tracking $f_{\mathbf{p}}$, we will instead use $\hat{f}_{\mathbf{p}}$, an approximate version of the frequency.

The resulting algorithm is called FamilyHH as it keeps track of reintroduction frequencies, one for the parent elements, $m_{\mathbf{p}}$, and one for the parent-child pairs, m_c. FamilyHH, shown in Algorithm 4, uses two separate instances of the SS data structure, namely, $SS_{\mathbf{p}}$ for the parents with space t, and SS_c for the parent-child pairs with space s (lines 4–1). The insertion and eviction mechanisms are similar to the ones presented in the previous algorithms. When a parent or a child is evicted from the corresponding structure reintroduction frequencies are updated by its current frequency plus one (lines 9, 11 and 17, 19 correspondingly for a parent and a child).

In order to identify the candidate Conditional Heavy Hitters on demand, we iterate over the heavy hitter parent-child pairs in decreasing order of frequency, and from each find $\widehat{\Pr}[c|\mathbf{p}] = \hat{f}_{p,c}/\hat{f}_{\mathbf{p}}$, or fill in the set T with the corresponding probabilities $T[(\mathbf{p}_i, c_i)]$ set equal to $SS_c[(\mathbf{p}_i, c_i)]/SS_{\mathbf{p}}[\mathbf{p}_i]$.

Both $SS_{\mathbf{p}}$ and SS_c are implemented as heaps, which leads to $O(\log(t+s))$ running time complexity for processing every element in the data stream.

Lemma 6. *The FamilyHH algorithm guarantees that each candidate* $(\mathbf{p}, c)$ *pair output will have*

$$\widehat{\Pr}[c|\mathbf{p}] = \Pr[c|\mathbf{p}] \pm 1/(\min(s,t)\Pr[\mathbf{p}]).$$

Proof. From the properties of the heavy hitters algorithm, we have that $f_{\mathbf{p}} \leq \hat{f}_{\mathbf{p}} \leq f_{\mathbf{p}} + n/t$

Algorithm 4 FamilyHH for Conditional Heavy Hitters

Input: Data stream $D = \{(\mathbf{p}_i, c_i), i = 1, 2, ...\}$.
Output: SS_c - SpaceSaving structure of length s for parent-child pairs

```
SS_p - SpaceSaving structure of length t for parents
m_c = 0 - highest frequency of a pair removed from SS_c
m_p = 0 - highest frequency of a parent removed from SS_p
for each element (p_i, c_i) of D do
    if p_i ∈ SS_p then
        SS_p[p_i] = SS_p[p_i] + 1
    else
        if |SS_p| ≥ t then
            m_p = SS_p.deleteMin()
        end if
        SS_p.insert(p_i, ++m_p)
    end if
    if (p_i, c_i) ∈ SS_c then
        SS_c[(p_i, c_i)] = SS_c[(p_i, c_i)] + 1
    else
        if |SS_c| ≥ s then
            m_c = SS_c.deleteMin()
        end if
        SS_c.insert((p_i, c_i), ++m_c)
    end if
end for
```

and $f_{\mathbf{p},c} \leq \hat{f}_{p,c} \leq f_{\mathbf{p},c} + n/s$. Consequently, we have

$$\begin{aligned}
\widehat{\Pr}[c|\mathbf{p}] = \frac{\hat{f}_{p,c}}{\hat{f}_{\mathbf{p}}} &\leq \frac{f_{\mathbf{p},c} + n/s}{f_{\mathbf{p}}} = \Pr[c|\mathbf{p}] + \frac{1}{s \Pr[\mathbf{p}]} \\
\widehat{\Pr}[c|\mathbf{p}] = \frac{\hat{f}_{p,c}}{\hat{f}_{\mathbf{p}}} &\geq \frac{f_{\mathbf{p},c}}{f_{\mathbf{p}} + n/t} = \frac{\Pr[c|\mathbf{p}]}{1 + n/(f_{\mathbf{p}} t)} \\
&\geq \Pr[c|\mathbf{p}](1 - \frac{n}{f_{\mathbf{p}} t}) = \Pr[c|\mathbf{p}] - \frac{\Pr[c|\mathbf{p}]}{t \Pr[\mathbf{p}]} \\
&\geq \Pr[c|\mathbf{p}] - \frac{1}{t \Pr[\mathbf{p}]}
\end{aligned}$$

Based on this analysis, we choose to set $t = s$, so that the error bound is $\Pr[c|\mathbf{p}] \pm 1/(s \Pr[\mathbf{p}])$. □

This estimate may sometimes overestimate, and sometimes underestimate, depending on the errors of the component estimates. We can make it always overestimate or always underestimate by scaling these values appropriately.

3.4.5 SparseHH Algorithm

Our final, most involved, algorithm also keeps only approximate information on the set of parents. We call this the SparseHH algorithm, as it keeps only sparse information on the parents. For the parent-child pairs, we make use of the CSS structure from the CondHH algorithm, which attempts to retain those pairs with the highest conditional probability. However, now that we are retaining only a subset of the parents, we need to modify this slightly. We will aim to maintain frequency information on only those parents that have an active child in the data structure. Now, when we come to insert a new parent-child pair p, c into CSS, we must decide what bound on the frequency to use. We suggest some possibilities for this "reintroduction strategy":

- *Global.* Maintain a global value m on the max (estimated) frequency of any (p, c) pair that has been deleted so far.
- *Ancestor.* If there is some reason to believe that parents with similar labels (e.g. in a hierarchy) have similar frequency, then we can maintain a small number g of different groups of parents, and retain for each the maximum frequency of any $(\mathbf{p}, c)$ deleted that came from that group. For example, if the $\mathbf{p}$ values are drawn from a hierarchy, we can choose a high level in the hierarchy, and create groups based on this.
- *Hash Partition.* For other cases, we can create a random partitioning of parents into g groups based on a hash function, and maintain the maximum frequency of any $(\mathbf{p}, c)$ pair belonging to that group. In the case of a single group, this becomes equivalent to the global strategy.
- *Bloom Filter.* We can keep a compact summary of items deleted with high values of $\hat{f}_{p,c}$, say, in the form of a Bloom Filter [19]. That is, when we delete a pair with frequency $\hat{f}_{p,c}$, we compute an index from this as $i = \lceil \log_b \hat{f}_{p,c} \rceil$ for a parameter b (for concreteness, consider $b = 2$). Then we insert $(\mathbf{p}, c)$ into a Bloom Filter indexed by i. When a new pair $(\mathbf{p}, c)$ is inserted into the data structure, we scan through the Bloom Filters, testing if $(\mathbf{p}, c)$ is present in each. If the test indicates it is in the ith Bloom Filter, then we instantiate $\hat{f}_{p,c} = b^i$. Note that Bloom Filters may lead to false positives: in this case, we will result in an overestimate of the frequency. This may lead to false positives in the estimated conditional heavy hitters, but will ensure that we do not underestimate the conditional probability of any pair.

Depending on the circumstances, any of these reintroduction strategies may be better, and indeed, we can run multiple of these in parallel, and choose the one that gives the smallest estimated value of $\hat{f}_{p,c}$ each time. SparseHH is described in Algorithm 5. The reintroduction data structures for parents (line 2) and parent-child pairs (line 3) follow one of the strategies listed above, and get updated according to this strategy every time there is an eviction (lines 10–11), or an insertion (lines 16–17). As in the previous algorithms the set of potential heavy hitters is computed on demand and consists of all the pairs

Algorithm 5 SparseHH for Conditional Heavy Hitters

Input: Data stream $D = \{(\mathbf{p}_i, c_i), i = 1, 2, ...\}$.
Output: CSS - SpaceSaving structure of length s for parent-child pairs

```
1: f_ap - frequencies of parents which are active in CSS structure
2: R_p - reintroduction frequency for a parent
3: R_c - reintroduction frequency for a pair CSS
4: for each element (p_i, c_i) of D do
5:   if (p_i, c_i) ∈ CSS then
6:     f_ap(p_i) = f_ap(p_i) + 1
7:     CSS[(p_i, c_i)] = CSS[(p_i, c_i)] + 1
8:   else
9:     if |CSS| ≥ s then
10:      CSS.deleteMinCondProb()
11:      Update f_ap, R_c and R_p
12:    end if
13:    if p_i ∉ f_ap then
14:      f_ap(p_i) = R_p(p_i)
15:    end if
16:    CSS.insert((p_i, c_i), R_c[(p_i, c_i)] + 1)
17:    f_ap(p_i) = f_ap(p_i) + 1
18:  end if
19: end for
```

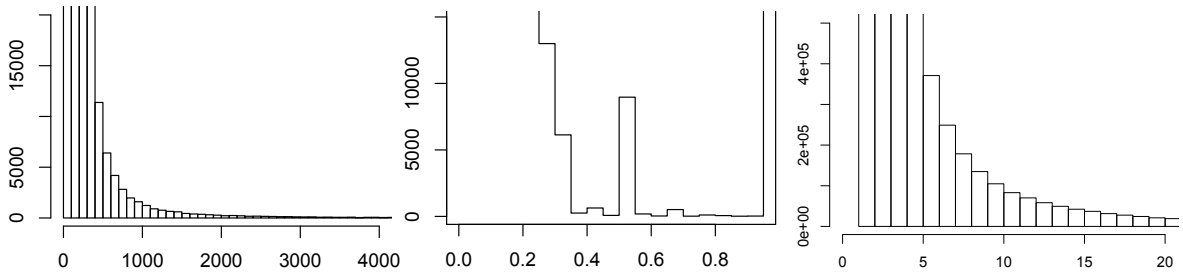
$(\mathbf{p}_i, c_i) \in CSS$ with their conditional probabilities $T[(\mathbf{p}_i, c_i)] = CSS(\mathbf{p}_i, c_i)]/f_{ap}(\mathbf{p}_i)$.

The R_c and $R_{\mathbf{p}}$ structures can be implemented with data structures that support the search, insert and delete operations have (average) complexity of $O(1)$, such as those based on hash tables. The CSS structure, similarly to the CondHH algorithm, can be implemented using a double linked list and a heap, or a balanced search tree structure. Therefore, the overall running time of SparseHH is $O(\log s)$ per element.

There are several other implementation choices for SparseHH:

- Different reintroduction strategies may offer either upper or lower bounds on estimated counts; upper bounds favor recall, while lower bounds favor precision.
- We divide the memory available to the algorithm into two pieces: the memory used for the main tracking of counts (which in turn is split into information kept for parents and for approximate (p, c) frequencies); and the memory used for estimating counts when an item is introduced into the structure. We use a parameter ρ to describe this split: a ρ fraction of the available memory is given to the main structure, and $1 - \rho$ to the reintroduction structure.

We compare these choices empirically in Section 3.5.



(a) Parent frequencies (b) Conditional heavy hitter probabilities (c) Parent-child pair frequencies

Figure 3.2: Descriptive statistics for WorldCup'98 data.

3.4.6 Discussion

We have proposed a variety of algorithms. They fall into two main classes: those that keep some information about each parent (GlobalHH, ParentHH, and CondHH); and those that do not (FamilyHH and SparseHH). Each algorithm aims to accurately approximate the conditional probability of pairs, based on different priorities for what information to retain given limited space. Among these, we are most interested in the behavior of CondHH and SparseHH, since these most directly capture the nature of conditional heavy hitters by focusing on the (estimated) conditional probability of items. Meanwhile, GlobalHH, ParentHH and FamilyHH are based on the raw frequencies of items. A priori, it is unclear which algorithm will perform best for the task of retrieving Conditional Heavy Hitters from a stream, so we will compare them empirically to determine the relative performance.

3.5 Experimental Results

All our experiments were conducted on a single 2.67GHz core of a Linux server with a large total amount of available memory. In evaluating the quality of our algorithms for recovering conditional heavy hitters, we make use of several measures of accuracy:

- The Precision and Recall of the recovered conditional heavy hitter pairs relative to the "true" set that are greater than a threshold $\Pr[c|\mathbf{p}] \geq \phi$ (Definition 3);
- The Precision of the top-τ popular conditional probabilities (Definition 4)[1];
- The Average Precision for the popular conditional probabilities, where the average is taken over all top-r sets of popular Conditional Heavy Hitters, for $r = 1, 2, ..., \tau$.

[1]Note that when restricting output to have size exactly τ, precision and recall are identical, so we do not duplicate this measurement.

3.5.1 Data Analysis and Experimental Setup

We applied the above algorithms for several real and artificial datasets, namely (1) simulated Markov chains, to estimate the largest elements of the matrix of transition probabilities; (2) requests made to the World Cup 1998 Web site to detect Conditional Heavy Hitters between clientID and objectID of the requests; (3) GPS trajectories of taxis in San Francisco to detect the most probable position of the vehicle taking into account two previous positions. We describe the datasets in more detail in the following sections. ϕ was chosen in each case according to the characteristics of the data in order to have reasonable number of Conditional Heavy Hitters. We distinguish between cases where the data is sparse—few parents have conditional heavy hitter children—and dense—most parents have conditional heavy hitter children.

Markov chain artificial data. As discussed in the Introduction, one application of finding the Conditional Heavy Hitters is to model the transition probabilities of a Markov chain. The goal is then to estimate the entries in this transition probability matrix, by finding the large values (and assuming the rest to be uniform). To model a Markov chain of order k, we concatenate the k most recent observations together to form a parent, and take the next observation as the corresponding child. In general, given an alphabet A, it is not feasible to track all the $|A|^{k+1}$ transition probabilities exactly, due to the high resource costs to do so. Hence, we instead use our algorithms to find and estimate the highest and the most important elements of transition probability matrix. In our experiments we use an alphabet size $|A| = 10^3$ and model a Markov chain of order $k = 2$. This means there are one million parents and one billion possible parent-child pairs.

We use two types of generation process for the data. The first case generates "dense" sequences so that each parent (P) has exactly one "heavy" child (C_h) with conditional probability chosen (randomly) to be greater than 0.6. The rest of the probability mass is uniformly distributed among the other possible edges. More formally, $\forall P \in A \times A$, $\exists c_h \in A$, such that $\Pr[c_h|\mathbf{p}] \geq 0.6$ while $\Pr[c_i|\mathbf{p}] = \dfrac{1 - \Pr[c_h|\mathbf{p}]}{|A| - 1}$, where $c_i \in A$, $c_i \neq c_h$. In this setting, there are 1 million conditional heavy hitters out of 1 billion possibilities.

The second generation process produces a "sparse" sequence with a predefined number of Conditional Heavy Hitters that is smaller than the number of parents. We identify a subset of parents to have one or more heavy children. We determine the number of heavy children n_c for a "heavy" parent by picking n_c from a truncated normal distribution with mean 3 and standard deviation 2. In our experiments we created a total of 200K conditional heavy hitters, so on average, only 1 in 15 parents has conditional heavy hitter children. Each "heavy" parent shares a total transition probability equal to 0.8 among its conditional heavy hitter children. The rest of the probability mass 0.2 is divided uniformly among the other edges. More formally, if a parent $\mathbf{p}$ is chosen to be heavy, then we pick n_c

children C at random, and set their transition probabilities $\Pr[c \in C|\mathbf{p}] = 0.8/n_c$, while for the others $\Pr[c \notin C|\mathbf{p}] = 0.2/(|A| - n_c)$. We set the number of Conditional Heavy Hitters to recover as the true number of Conditional Heavy Hitters, i.e. 200K.

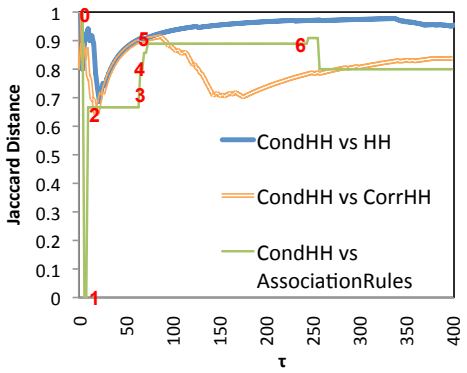
Taxicab GPS data. The Taxicab data consists of about 20 million GPS points for a fleet of taxis, collected over the course of a month, obtained from cabspotting.org. To go from the fine-grain GPS locations to streams of values, we performed pre-processing to clip the data to a bounded region and coarsen to a grid. The region of the measurements is restricted to a rectangle in the area of San Francisco, with latitude in the range $[37.6...37.835]$, which covers 26km and longitude in the range $[-122.52... - 122.35]$, which covers 15km. This clipping was performed to remove a few incorrect readings which were far outside this region.

This space was partitioned into 10,000 rectangles using a 100×100 grid. Given the readings within this grid, we proceeded to define trajectories from the data as a sequence of grid cells occupied by the same cab. We considered a new trajectory to begin if there was a gap of more than 30 minutes between successive observations. Following this definition, we extracted 54,308 trajectories. We model the trajectory data as a second order Markov chain, on the grounds that knowing the previous two steps is likely to be indicative of where the next step will take us. A first order model would only have the previous location, and so would not capture in what direction the vehicle was traveling. This model generates around 160,000 distinct parents and a million distinct parent-child pairs; our experiments with finer grids (omitted for brevity) had even higher dimensionality. With a default ϕ value of 0.8, we observed 63,721 conditional heavy hitters. This means that about 2 out of every 5 parents have conditional heavy hitter children, so we consider this a dense dataset.

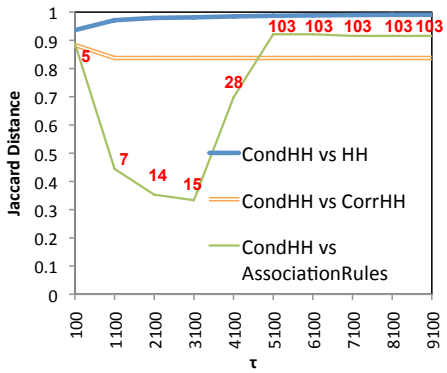
Worldcup'98 data. The Worldcup data[2] contains information about the requests made to the World Cup Web site during the 1998 tournament. Each request contains a ClientID (a unique integer identifier for the client that issued the request) and an ObjectID (a unique integer identifier for the requested URL). We are interested in finding Conditional Heavy Hitters between ClientID and ObjectID pairs, where ClientID is treated as the parent, and ObjectID as the child. That is, we are interested in detecting (ClientID, ObjectID) pairs where the requested child is particularly popular for that user.

We used data from day 41 to day 46 of the competition. The total number of records in this period is around 105 million; the number of distinct parent-child pairs is around 59 million; and the number of distinct parents is 540K. In this data we look for the Conditional Heavy Hitters that have a probability of occurrence greater then $\phi = 0.25$. The total number of such Conditional Heavy Hitters is in excess of fifty thousand. About 1 in 10 parents has a conditional heavy hitter child, making this data relatively sparse.

[2] `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`



(a) τ from 1 to 400.



(b) τ from 100 to 9100.

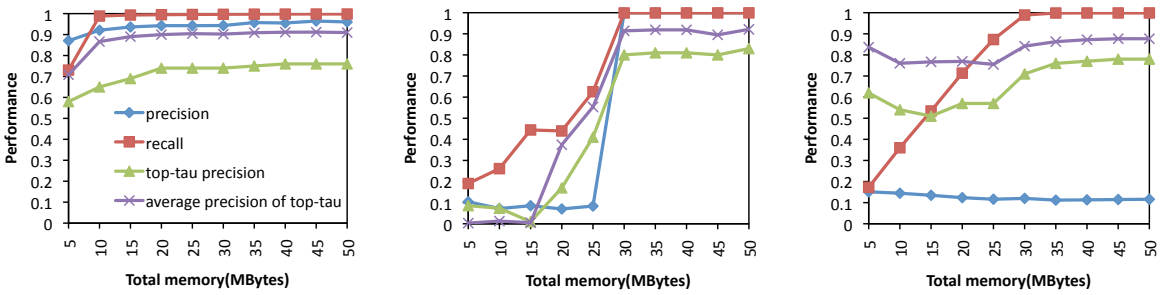
Figure 3.3: Jaccard distance for top-τ conditional (CondHH) and simple (HH), CondHH and correlated(CorrHH) heavy hitters, and CondHH and Association Rules with red numbers correspond to r - the number of retrieved frequent item sets which are triples.

The frequency distributions are skewed: there are many parents and parent-child pairs that are found only once or a small number of times in the dataset (Figure 3.2a and Figure 3.2c). Although most of the parent-child pairs occur once—52 million out of the 59 million distinct pairs—still, there are many pairs that occur a greater number of times. The distribution of probabilities of Conditional Heavy Hitters is shown in Figure 3.2b, and shows that there is sufficient probability mass associated with higher conditional probabilities.

3.5.2 Comparison with Association Rules, Simple and Correlated Heavy Hitters

In the first set of experiments, we compare Conditional Heavy Hitters to simple heavy hitters, correlated heavy hitters and association rules, and demonstrate that Conditional Heavy Hitters constitute a distinct set of elements that cannot be identified by existing methods. In these experiments, we computed the top-τ heavy hitters, correlated heavy hitters, and Conditional Heavy Hitters, and then computed the Jaccard distances between the Conditional Heavy Hitters and the other three approaches. We recall that the Jaccard distance between two sets X and Y is defined as $1 - \frac{|X \cap Y|}{|X \cup Y|}$. The distance is 0 if $X = Y$, and the closer the distance gets to 1, the more the two sets are different (distance 1 means that the sets are disjoint).

Since the set of Conditional Heavy Hitters and the set of association rules cannot be directly compared, we performed the experiment as follows. We first identified the top-τ frequent itemsets of sizes two and three combined, as both are needed in order to compute Conditional Heavy Hitters that model a Markov chain of order 2 (i.e., the parent consists



(a) Hash Partition strategy (b) Ancestor strategy (c) Bloom Filter strategy

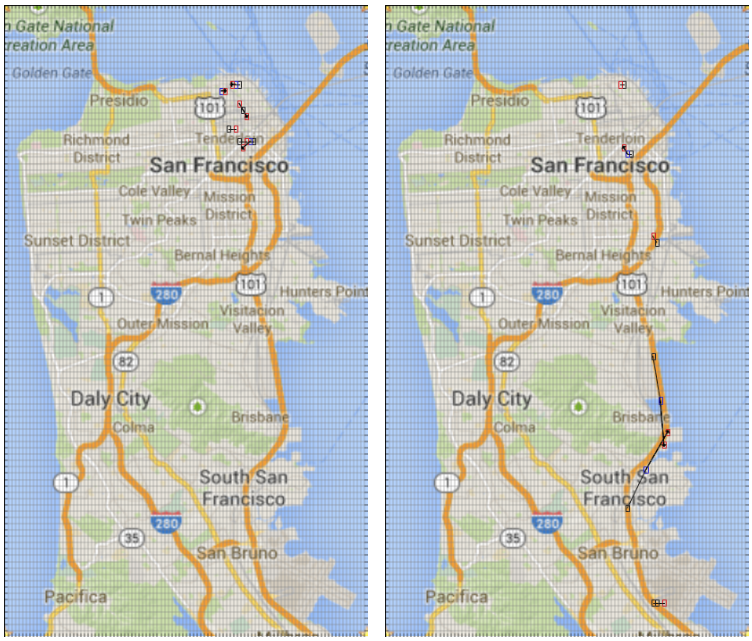
Figure 3.4: SparseHH accuracy for conditional heavy hitter recovery on Worldcup data under different reintroduction strategies.

of two elements of the data stream). From these τ itemsets we were able to compute r Conditional Heavy Hitters, and we compared those to the set of top-r Conditional Heavy Hitters, which were extracted from the τ results computed by our approach. The results of this comparison on the WorldCup'98 dataset are shown in Figure 3.3 (the plot is broken in two pieces to aid readability: Figure 3.3a shows the results for small values of τ, while Figure 3.3b plots the trends as τ increases to large values). The numbers (in red) marked on the "CondHH vs AssociationRules" curve denote the value of r for the different experiments.

These graphs show that the set of Conditional Heavy Hitters is very different from the sets produced by the other approaches. This is especially true when we compare traditional heavy hitters to Conditional Heavy Hitters, indicating that the two definitions are truly describing distinct phenomena. Likewise, there is little similarity between the results found for correlated heavy hitters and Conditional Heavy Hitters. We also observe that the number r of parent-child pairs discovered by association rules is very low compared to τ in all cases. Although the most frequent parent-child pairs are similar to the most popular Conditional Heavy Hitters for some of the small values of r, in general, the set of association rules is very different from that of Conditional Heavy Hitters. Thus, current approaches for finding association rules cannot help in retrieving Conditional Heavy Hitters, for which we need new algorithms in order to efficiently identify.

Utility of Conditional HH. We now study the elements that are found as Conditional Heavy Hitters and interpret them in a domain where the semantics are known. We compare to the elements found as correlated heavy hitters, and show that there is value in both sets discovered, but Conditional Heavy Hitters can provide more useful insights than correlated heavy hitters. For this experiment we use the taxicab GPS data, and compare the top-25 popular Conditional Heavy Hitters with the top-25 correlated heavy hitters. Following Notion 3, correlated heavy hitters are sorted in descending order of the

parent frequency. There is some overlap between these two sets, indicating items that are reported as significant under both definitions. However, there are 14 elements found outside the overlap: 7 specific to each definition. We plot each of these sets of 7 elements overlaid on the San Francisco area maps from which they are drawn, in Figure 3.5.

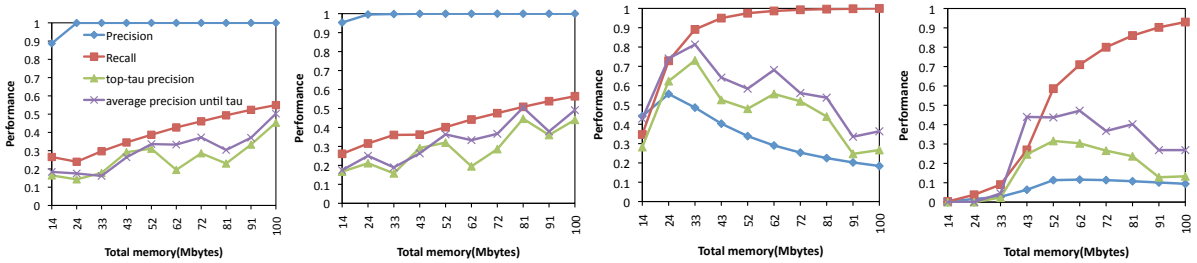


(a) Correlated HH (b) Conditional HH

Figure 3.5: Unique correlated and Conditional Heavy Hitters not detected by the other technique.

The plots represent the San Francisco region, where the taxicab data was collected from, discretized into a 100×100 grid. Here, a parent is defined as two successive positions (this helps to establish direction of travel, for example), and the child is the subsequent location. In some cases, two out of the three cells intersect. We plot first parent cells (the first in a sequence) using a black border, second parent cells (the second in a sequence) using a blue border, and child cells (the third in a sequence) using a red border. There is a line between the first parent cell and the second parent cell, and an arrow between a second parent cell and a child, if they do not overlap. Figure 3.5a shows the correlated HH that are not found with the conditional HH definition, while Figure 3.5b shows the conditional HH that are not detected as correlated HH.

We interpret these results based on our study of features on the map such as highways and tourist attractions – note that the algorithms do not possess any such knowledge. We observe that the unique correlated HH are primarily short trajectories concentrated around the city center, indicating slow moving traffic around popular points of interest. Meanwhile, the conditional HH are found around the city center but also on the highways further outside the city. Of particular interest are trajectories around the airport (SFO),



(a) Favor precision, $\rho = 0.9$. (b) Favor precision, $\rho = 0.95$. (c) Favor recall, $\rho = 0.5$. (d) Favor recall, $\rho = 0.9$.

Figure 3.6: Accuracy on sparse synthetic data using SparseHH.

which show journeys that turn off the highway to go to the airport.

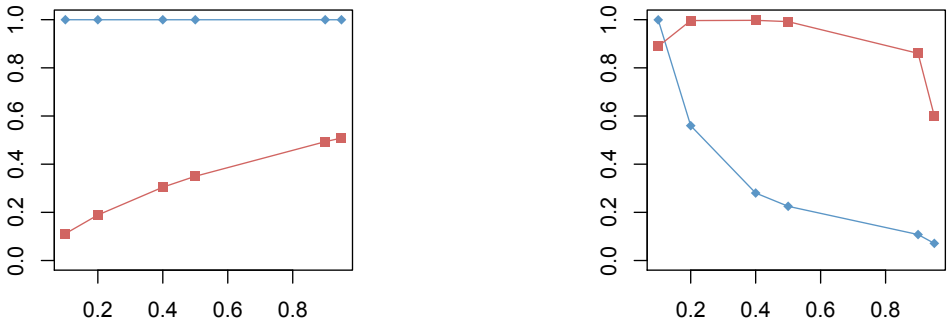
Although similar, the two definitions emphasise different aspects. Correlated heavy hitters are ones which have high support for the parent. In this instance, they have shown that it is common to make slow progress in the heart of the city (parent), in which case it is quite common to continue making slow progress in the same direction (child). While helpful in identifying "pinch points" in the traffic, this does not provide much unexpected information. Conditional heavy hitters place more emphasis on having a high conditional probability of the child, given the parents. In this instance, they highlight that traffic travelling south on highway 101 is likely to stay on (rather than take an exit), and also that traffic on the highway close to the airport is very likely to go to the airport. This highlights the importance of the airport as a destination from the highway. Such insights can be of greater use to traffic planners and city architects in understanding typical journey and behavior around intersections. We obtain similar results when we look at different sized sets, such as the top-20 and top-30 sets. These results demonstrate that the conditional HH concept can reveal interesting and meaningful patterns, distinct from those found by correlated HH.

3.5.3 Parameter Setting for SparseHH

The SparseHH algorithm has several parameters and choices that affect its performance. Here, we investigate how to set these parameters before comparing with other algorithms.

Choice of reintroduction strategy. We compare the different choices of reintroduction strategy: hash partitioning, ancestor, and Bloom Filter. Figure 3.4 shows the accuracy over the Worldcup data, where we set $\tau = 100$ and $\phi = 0.25$ to define the (popular) conditional heavy hitters.

Here, the ratio of memory allocated to the main structure, ρ, was set to 0.9, with the remainder used to help reintroduce items to the data structure. We observe that the hash partitioning strategy performs the best across all metrics (Figure 3.4a). The ancestor



(a) Favor precision

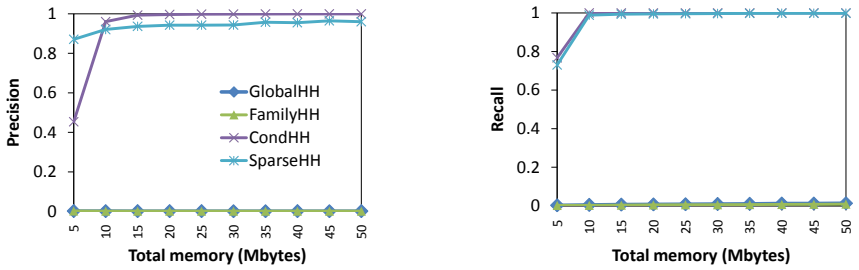
(b) Favor recall

Figure 3.7: Precision (blue diamonds) and Recall (red squares) of SparseHH variations on sparse synthetic data as ρ varies.

strategy can obtain good results, but only when a larger total amount of memory is made available (Figure 3.4b). The Bloom Filter strategy, while achieving high recall, always has very poor precision (Figure 3.4c). Based on this and other results, we adopt the Hash partitioning strategy as the method of choice for SparseHH: while the Ancestor method is sometimes competitive, this can be seen as a special case of the Hash partition method with a structured choice of hash function, so we do not further distinguish these methods.

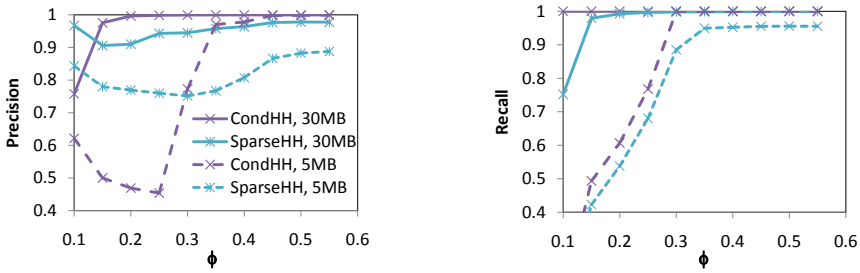
Choice of memory ratio. As noted in Section 3.4.5, we can adjust the estimated counts in the algorithm to give either upper or lower bounds, and hence to favor precision or to favor recall. We compared the impacts of this choice in our experiments, shown in Figure 3.6 for the sparse synthetic data. We set $\phi = 0.05$, sufficient to distinguish the conditional heavy hitters from the other pairs. In the plots, we pick a representative selection of parameter settings, as we vary the ratio ρ that governs the division of memory between the main and reintroduction structures, and whether the algorithm favors precision or recall. Across these, we first observe that this choice does indeed behave as advertised: favoring precision obtains near perfect precision, while favoring recall allows recall to grow as total memory increases. However, when we favor precision, recall tends to improve as we allocate more memory (Figures 3.6a and 3.6b), while favoring recall tends to cause precision to drop off as more memory is used (Figures 3.6c and 3.6d).

To investigate this further, we fix the available memory, and vary the ratio ρ. The results on the same data are shown in Figure 3.7. We observe that when we favor precision, the precision is always near perfect (Figure 3.7a). The benefit of giving more memory to the main structure outweighs the loss from reducing space for the reintroduction strategy, so a large ρ value gives the best recall. Contrarily, favoring recall has generally good recall, but gets the best precision when almost all of the memory is turned over to the reintroduction strategy (Figure 3.7b). Still, it is hard to obtain both good recall and good precision from this strategy: although we see some good behavior for very small values



(a) Precision (b) Recall

Figure 3.8: Precision and Recall on the WorldCup data.



(a) Precision (b) Recall

Figure 3.9: Accuracy as ϕ varies on Worldcup data

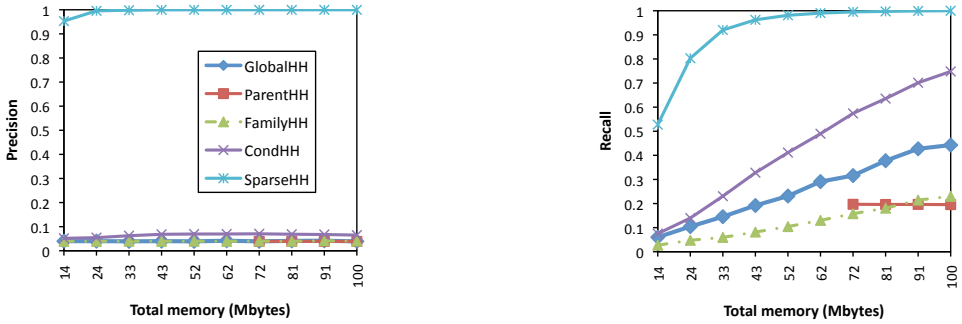
of ρ here, this was not stable across other datasets. Consequently, we conclude that it is preferable to favor precision, and adopt this with $\rho = 0.9$ as the default in all other experiments.

3.5.4 Performance on Sparse Data

We now compare all the proposed algorithms, initially on sparse data, and subsequently on more dense data.

World Cup Data. We present results for recovering (ClientID, ObjectID) Conditional Heavy Hitters from the (relatively sparse) Worldcup data. In other experiments, we also looked for correlations on other attribute combinations, such as (ServerID, ObjectID). The results there were broadly similar, and so are omitted for brevity.

Figure 3.8 shows results on precision and recall for recovering the Conditional Heavy Hitters for this data. Here, the CondHH and SparseHH (using Hash partition reintroduction) methods perform the best for both precision and recall. These two algorithms both make use of an eviction strategy that picks the parent-child pair with the lowest (estimated) conditional probability to be deleted from the main structure. This observation suggests that such a pruning strategy can be effective at retaining the most promising pairs in memory. For this data, the number of parents is not so large, and so it is feasible to retain information on all parents. Thus, CondHH is not penalized for this choice here,



(a) Precision (b) Recall

Figure 3.10: Accuracy on sparse synthetic data as memory varies

although we see examples later where there are too many parent items to track effectively. These methods also achieved high top-τ precision, over 0.8, indicating over 80% agreement between the top 100 reported Conditional Heavy Hitters and the true most popular heavy hitters. On this data, we observe that other approaches suggested—GlobalHH FamilyHH and ParentHH (omitted from the plots)—are unable to provide useful results: although they provide accuracy guarantees as a function of the space available, it turns out that these guarantees do not become useful until much more memory is available. In this case, the successful algorithms (CondHH and SparseHH) are able to achieve near-perfect precision and recall using less than 10% of the memory required to represent the data exactly.

Figure 3.9 shows the accuracy of CondHH and SparseHH as we vary ϕ, the threshold for defining a conditional heavy hitter. We see that for large ϕ values and moderate memory (30MB), CondHH is preferable, and achieves near-perfect precision and recall. As ϕ is decreased, there are more Conditional Heavy Hitters to recover, and when memory is constrained to only 5MB (the dashed lines), recall necessarily falls: the algorithms are unable to retain information about all Conditional Heavy Hitters. However, in the low ϕ, low memory setting, SparseHH is able to maintain higher precision, while the precision of CondHH falls off.

Pruning Strategy With Combined Eviction Criteria. We now evaluate the effect of using a combined eviction criteria, based on both the conditional probability and the frequency of the parent-child pairs, for both of which we fix thresholds. When the memory budget is reached the eviction strategy works as follows:

1. evict the pair for which both thresholds do not hold;
2. otherwise, evict the pair for which the threshold on the conditional probability does not hold;
3. otherwise, evict the pair for which the threshold on the parent-child frequency does

not hold;

4. otherwise, evict the pair with the lowest conditional probability.

This eviction strategy was implemented for both IP (Intermediate Parent) and Hash reintroduction strategies of SparseHH, and was tested on the WorldCup data. The experiments were done using the same settings as for the prior reintroduction strategies, and for different frequency thresholds to check their influence on the results. The new strategy does not lead to significant differences: precision and recall are the same (and up to 10% smaller for small memory budgets), while top-τ and average τ precision are 10% higher (and up to 18% for small memory budgets) for the modified versions. We tried with values 10, 100 and 1000 as the frequency thresholds used for eviction, and observed little sensitivity of the results; the reason is that low conditional probability is still an important criterion of eviction. The changes in the Hash version of SparseHH algorithm are even less pronounced, though the general trend is the same: precision and recall results are a bit lower while top-τ and average top-τ precisions are higher for the modified algorithm.

Sparse Synthetic Data. We now compare all the algorithms on the truly sparse synthetic data, for a stream of length 10^8. This data has a much smaller number of Conditional Heavy Hitters compared to the number of parent items. Consequently, we expect the algorithms which try to keep information on all parents to perform poorly here, since this will occupy most of their available resources.

This conjecture is confirmed in Figure 3.10: only SparseHH is able to obtain both good precision and good recall for the range of memory provided. It also has accuracy as measured by top-τ precision and average precision up to τ: both around 0.9 (plots omitted for space reasons). Among the other algorithms, CondHH shows the best improvement in recall as more memory is made available, with GlobalHH and FamilyHH improving more slowly (Figure 3.10b). The ParentHH algorithm can only produce results when enough memory is available to keep a (very small) summary structure for each parent—in this case, above 72MB. Interestingly, the precision performance of all algorithms apart from SparseHH is very poor: much more memory is needed before these can achieve good precision (Figure 3.10a). This is in part because even the highest amount of memory shown in Figure 3.10 represents less than 5% of the space to record the exact statistics for the given data. In terms of the original application, of approximating the Markov chain transition matrix, the results are also strong: the L_1 difference between the distributions is about 0.01, where 0 would be perfect recovery, and 1 represents the worst case. We conclude that over sparse data, the SparseHH algorithm has the best performance and is the method of choice.

In terms of the time cost of the algorithms, Figure 3.11 shows that there is little systematic variation as a function of the size of the summary structure. The simpler

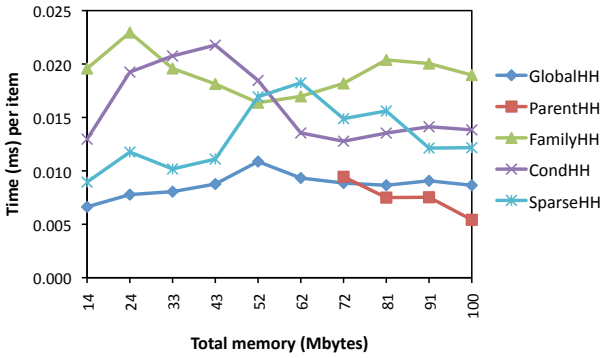
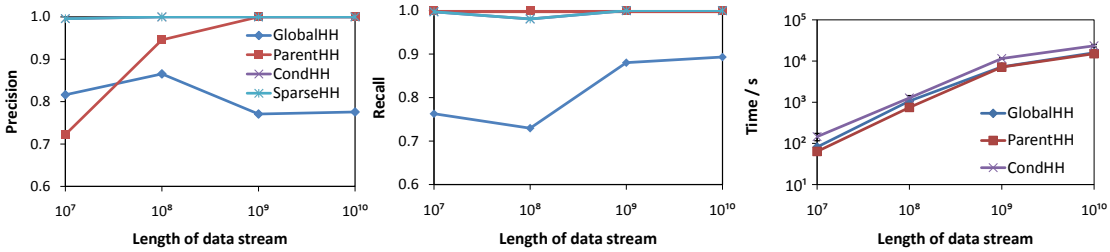


Figure 3.11: Time cost of algorithms on sparse synthetic data as memory varies



(a) Precision (b) Recall (c) Time

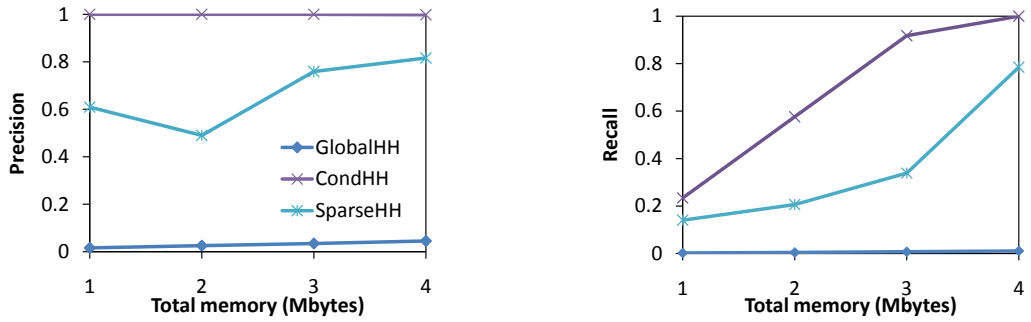
Figure 3.12: Accuracy and timing results for algorithms on dense synthetic data

GlobalHH and ParentHH algorithms are the faster ones, but all algorithms have performance measured in the hundreds of thousands of updates per second to process 10^8 items.

3.5.5 Performance on Dense Data

Dense synthetic data. In the dense synthetic data, each parent has at least one child that is a conditional heavy hitter. We generate a stream of data from the 2nd order Markov chain of different lengths, between 10^7 and 10^{10} observations. We allocate an amount of space equivalent to twice the number of possible parent items, and evaluate their accuracy in terms of precision and recall on streams of varying lengths. With the threshold $\phi = 0.5$, Figure 3.12 shows the average time and accuracy achieved over 10 independently chosen streams. The observed standard deviation over these repetitions was very low, around 10^{-3} for all precision and recall computations.

The results show that the GlobalHH algorithm performs poorly, with only moderate precision and recall on this relatively "easy" data set (Figures 3.12a and 3.12b). The ParentHH algorithm has near perfect recall, and precision improves as the stream gets longer (and so the signal becomes easier to detect). However, again, the CondHH algorithm has the best accuracy, getting near perfect precision and recall throughout. The SparseHH



(a) Precision (b) Recall

Figure 3.13: Accuracy on Taxicab data as memory varies

algorithm had identical results to CondHH on this data. On closer inspection of the data structures, we observed that this was because SparseHH has sufficient memory to keep frequency information on all parents. Consequently, it can store the same information as CondHH and so finds the same estimated frequencies. For similar reasons FamilyHH kept the same information as GlobalHH and so is omitted from the plots. In terms of scalability, all algorithms are similar. Figure 3.12c shows that the CondHH algorithm is slightly slower in our implementation, due to the more involved data structure maintenance process. However, the difference is not substantial, and could be improved by a more engineered solution. Even here, the throughput is nearly half a million updates per second on a single core.

Taxicab data. The Taxicab data is quite dense: many parents have a conditional heavy hitter child. Figure 3.13 provides precision and recall results on this data for $\phi = 0.8$. As in other experiments, GlobalHH does not provide useful recovery of Conditional Heavy Hitters with such low memory. SparseHH achieves good precision, but CondHH has enough memory to obtain perfect precision (Figure 3.13a). The story is similar for recall: SparseHH improves as more memory is available, but is consistently dominated by CondHH, until SparseHH is given enough memory to store all parents. Moreover, for the top-τ precision the results for CondHH were much stronger, approaching 1, while SparseHH achieved only 0.25.

To better understand the relative behavior of these two competitive algorithms, Figure 3.14 shows the case as we vary ϕ, the threshold for Conditional Heavy Hitters, while holding the total memory constant at 4MB. As ϕ decreases, there are more pairs passing the threshold, and so the problem becomes harder. The precision of SparseHH tends to remain constant, while there is a more notable dip in the precision of CondHH. Interestingly, adjusting the memory available for the reintroduction strategy of SparseHH by adjusting ρ has a marked effect: putting more memory to this end improves precision, but reduces recall. We conclude that for dense data, CondHH is the method of choice,

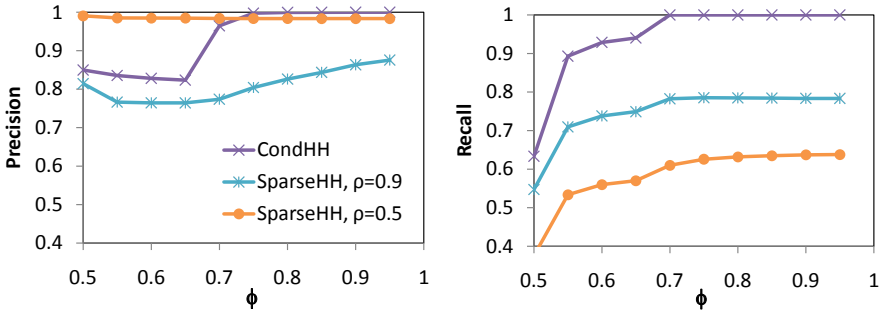
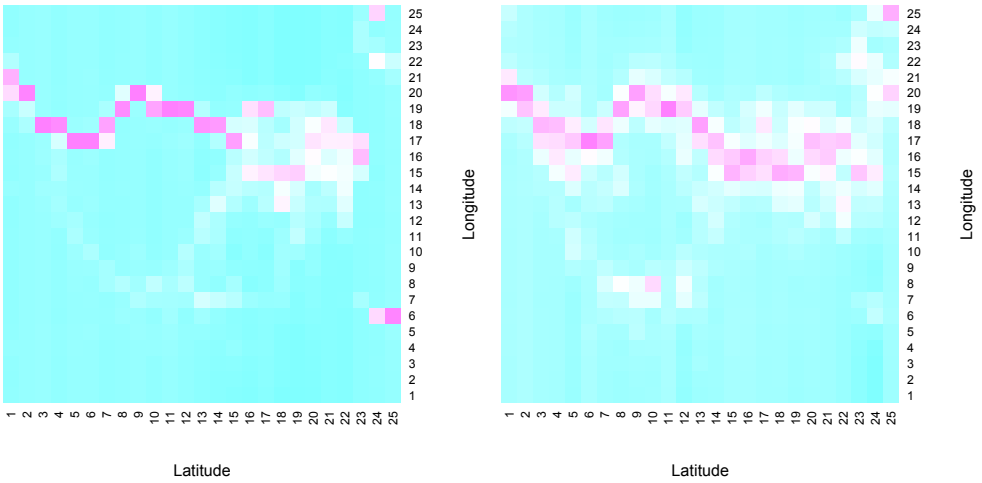


Figure 3.14: Accuracy as ϕ varies on Taxicab data



(a) Exact Markov Model

(b) Recovered Markov Model based on Conditional Heavy Hitters

Figure 3.15: Heatmaps of trajectories modeled using exact transition probability matrix and recovered matrix based on Conditional Heavy Hitters.

provided we can afford to store all parents.

3.5.6 Markov Model Estimation

In this section, we experimentally evaluate how well Conditional Heavy Hitters can approximate a Markov chain model. The premise is that the conditional probabilities we derive from the Conditional Heavy Hitters can be used to estimate the largest elements of the transition probability matrix. We computed the Conditional Heavy Hitters in the TaxiCab dataset using the CondHH algorithm, using 3MB of total memory and $\phi = 0.8$ (the corresponding precision and recall values are shown in Figure 3.13).

First, we check whether Conditional Heavy Hitters can adequately describe the generation process of the real trajectories in the Taxicab data. To study this, we compare the heatmaps of the trajectories generated by the exact Markov model and by the recovered

Table 3.2: Earth Mover's Distance between the heatmaps of trajectories.

Heatmaps	Exact markov model
Conditional HH	1.98
Supervised Random	4.13
Random	7.40

Markov model, i.e., the model approximated by the Conditional Heavy Hitters results, depicted in Figure 3.15. The heatmap indicates the number of times different trajectories have passed through a spatial cell. The 'hottest' cells are colored in red, while the 'coolest' in blue. The resulting heatmaps show that the trajectories built using Conditional Heavy Hitters accurately reflect the hot regions of the trajectories built using the exact transition probability matrix, which indicates that Conditional Heavy Hitters indeed capture the highest transition probabilities.

Earth Mover's Distance comparison. In order to quantify the distance between the two heatmaps, we employ the Earth Mover's distance (EMD)[105]. EMD measures the difference of probability masses, multiplied by the distance that the probability mass has to be moved in order to derive one probability density function from the other. In our setting, we compute EMD over the probability density functions (or the corresponding histograms) that are represented by the heatmaps. We calculated the EMD between the heatmaps of trajectories generated by the exact markov model and Conditional Heavy Hitters, as well as two baseline models, Random and Supervised Random. The *Random* model corresponds to a random assignment for the next state of a trajectory at each step. The *Supervised Random* assigns the next state in the same way as our approach does when a particular prefix (i.e., parent) is missing. That is, it distributes the probability mass among the neighborhood of this prefix, assigning slightly more mass in the direction of movement (according to the previous state). The results, reported in Table 3.2, show that Conditional Heavy Hitters are more than two times closer to the exact markov model than the Supervised Random assignment, and more than 3.5 times closer than Random.

Kernel-density comparison. We also conducted a *Kernel density based two-sample comparison (KDE)* test [43] to compare the position distributions of the trajectories generated by the exact and recovered Markov models. The null hypothesis of this test was that the distributions were equal. We set the significance level $\alpha = 0.05$, and obtained a p-value for the test equal to 0.51, which means that there is not enough evidence to reject the hypothesis that these two distributions are equal.

Prediction accuracy. Besides the heatmaps, we also assess the prediction errors of the exact and recovered Markov models. Our goal is to check whether the recovered model can be used to predict the next state of a taxicab given its two previous states. We also compare the results with the error of Random and Supervised Random models described

above. Each model defines a probability distribution for the next state given two previous states. The prediction is performed using this probability distribution.

Two kinds of errors are considered:

1. the Mean Euclidean Distance (MED) between the estimated cell $\hat{c}_i$ and true cell c_i, $i = 1, 2, ..., T$, where T is the number of predictions over taxicab trajectories, T is larger than:

$$\begin{aligned} \text{MED} &= \frac{1}{T}\sum_{i=1}^{T} \text{dist}(\hat{c}_i, c_i) \\ &= \frac{1}{T}\sum_{i=1}^{T} \sqrt{(\hat{c}_i^x - c_i^x)^2 + (\hat{c}_i^y - c_i^y)^2}, \end{aligned}$$

   where c^x and c^y are spatial coordinated of the cells in the 100×100 grid that correspond to initial longitude and latitude of trajectory points;

2. the Misclassification Error (ME), or ratio of cells which were incorrectly estimated:

$$ME = \frac{1}{T}\sum_{i=1}^{T} \mathbb{1}\{\hat{c}_i \neq c_i\}.$$

Table 3.3 summarizes the results, which are averaged over 10 runs. The differences between all pairs of errors are statistically significant according to Welch Two Sample t-test [120], for which significance level α was set to 0.01. The results show that the accuracy of the exact model compared to the accuracy of the recovered model is 1% and 5.6% higher according to MED and ME correspondingly. This was expected as the recovered model is just an approximation of the exact model. At the same time, the performance of the recovered model is 23% and 6.2% better than the performance of Random and Supervised Random models according to ME (2521% and 19% better according to MED). The prediction made with the recovered model for the states with known prefix behaves even better.

We note that both the heatmaps and the prediction errors can be improved if we better model the cases where the prefix is not among the Conditional Heavy Hitters. This can be achieved using domain knowledge, such as the road maps in the region of the taxicab dataset. Nevertheless, even without domain knowledge, we have shown that the trajectories built using the Conditional Heavy Hitters are a fairly accurate representation of the original data.

Table 3.3: Mean Euclidean Distance (MED) and Misclassification Error (ME) of different prediction models for Taxicab dataset.

Prediction Model	**MED**	**ME**
Exact Model	1.77	0.80
Recovered model	1.87	0.81
Recovered model only with known prefix	1.59	0.74
Random	47.15	1.00
Supervised Random	2.23	0.86

3.6 Summary

In this thesis, we have introduced the notion of conditional heavy hitters as a useful concept that is distinct from prior notions of heavy hitters, correlated heavy hitters and frequent itemsets. We introduced a sequence of algorithms that build on existing techniques, but target the new definition. Our empirical study demonstrated that among these, it is those that most directly target the new definition, by preferentially retaining items with high (estimated) conditional probability and pruning those with low conditional probability, that perform the best. Specifically, the SparseHH algorithm, which keeps an approximate summary of both the parent-child pairs as well as the parent items, generally performs the best across a range of sparse datasets and parameter settings. In particular, it achieves high precision and recall on the set of Conditional Heavy Hitters while retaining only 5-10% of the space of storing exact statistics. When the data is more dense and there is sufficient memory, CondHH is the preferred method. If we do not know the nature of the data in the advance, we can simply run SparseHH, since it will keep information on parents exactly while there is room, and so behave more like CondHH. Future work will identify further applications for conditional heavy hitters, and evaluate their efficacy in those settings. Our algorithms are defined in the streaming model, which captures the challenging case of high-speed arrival of data. As the scale of data increases, it will become necessary to adapt these algorithms to a distributed setting, where multiple streams are observed, and the collected summaries can be combined to give a summary of the union of all the input data.

Chapter 4

Streaming Conditional Heavy Hitters of Variable Order

In this chapter we address the problem of online discovery of Variable Order Conditional Heavy Hitters for data streams whose elements can be represented as symbols from large finite alphabets. VOCHH represent the most significant sequential patterns from a data stream and serve the tasks of real-time data summarization, inference of the generative models of the data, and online prediction of future states.

In this chapter, first, we describe the preliminaries needed to formalize the notion of VOCHH in Section 4.1. The related work for this chapter has already been discussed in Section 2.3. We obtain the optimal pruning strategy for the LogLoss minimization and discuss its online variations in Section 4.2. Then, we describe the formalization of VOCHH and the problem definition in Section 4.3. Real-time solution approach is proposed in Section 4.4, followed by the experimental evaluation in Section 4.5, in which we assess the quality of the proposed algorithms on three real datasets. Finally, we summarize the results of this chapter in Section 5.3. The notations and abbreviations used in this chapter are listed in Appendix A.3.

4.1 Preliminaries

4.1.1 Markov Models

All Markov models assume markovian property, which means that the current state of the system depends on a finite number of the previous states.

Markov Chain

Markov chain is a fixed-order Markov model[1], where the order corresponds to the length of the context or the number of the preceding states that influence the current state. Markov chains of order k, defined for the states from a finite alphabet Σ, are described using two kinds of parameters:

1. the *transition probability matrix*, which provides the conditional probabilities of moving to the next state after seeing a particular context of the length k,
2. the *initial distribution*, which defines the probabilities of the first $k-1$ elements of the chain, which cannot be derived from the transition probability matrix.

More formally, initial distribution provides the probability density functions for all the combinations of the symbols from Σ up to the order k: $P(x_i = \sigma_i)$, $P(x_i = \sigma_i, x_{i+1} = \sigma_{i+1})$,..., $P(x_i = \sigma_i, x_{i+1} = \sigma_{i+1}, ..., x_{i+k-1} = \sigma_{i+k-1})$. The matrix of transition probabilities defines all the transitions from the contexts of the length k: $P(x_i = \sigma_i | x_{i-k} = \sigma_{i-k}, x_{i-k+1} = \sigma_{i-k+1}, ..., x_{i-1} = \sigma_i)$.

Further in the thesis, for simplicity, we will refer to the initial distribution as $P(x_i)$, $P(x_i, x_{i+1})$, $P(x_i, x_{i+1}, x_{i+k-1})$ and to a transition probability matrix as $P(x_i | x_{i-k}, x_{i-k+1}, ..., x_{i-1})$.

Together, the initial distribution and the transition probability matrix define the full probability distribution $P(\cdot)$ of the high order Markov chain. Note, that the order is fixed in this case.

Variable Order Markov Model (VMM)

In the variable order case, the current state may depend on a varying number of previous states. In the formulation of the markovian property, different contexts will thus have different length (order).

When VMM is considered, the transition probability matrix describes only meaningful transition probabilities, which are defined as follows:

Definition 5 (Meaningful Transition Probabilities)**.** *For a given child x and parent y^k of length k, $k \leq D$ a transition probability $P[x|y^k]$ of VMM is called* **meaningful** *if:*

1. *$P[x|y^k] = P[x|y^j]$ for all $j > k$ (the Markov property),*
2. *k is the minimum non-negative integer satisfying the condition above.*

The initial distribution of the VMM also depends on a particular sequence, and defines the probability distribution for the few first states that cannot be derived from the transition probability matrix.

[1] In the thesis we consider only homogeneous Markov models, in which the probabilities do not change over time.

Types of Markov Sources of Variable Order

The variable part of the VMM can be defined through the variability of only the context (parent), or through the variability of the parent-child pair (full sequence which contains both the context (parent) and the following symbol (child)). These to cases correspond to the notions of the *uniform* and the non-uniform VMM respectively.

Definition 6 (Uniform VMM)**.** *Uniform Variable Markov Model is the probabilistic model with the probability density function $P(\cdot)$ that has the following characteristics: if there is a parent-child pair (y^k, x) that has meaningful transition probability as per Definition 2, then for other children from the alphabet, the transition probability given the same parent is also meaningful. k is the length of the parent y^k.*

Thus, no longer parent that includes y^k as a suffix can have meaningful transition probabilities with respect to any child. In this case, the variability of the markovian source is only in the parent.

In the second model, the *Nonuniform VMM*, variability is connected with a parent-child pair:

Definition 7 (Nonuniform VMM)**.** *Nonuniform Variable Markov Model is the probabilistic model with the probability density function $P(\cdot)$ that has the following characteristics: if there is a parent-child pair (y^k, x) that has meaningful transition probability as per Definition 2, other children from the alphabet can have meaningful transition probabilities with the longer parents that have y^k as a suffix. k is the length of the parent y^k.*

Nonuniform model says that if there a pair with the meaningful probability, a longer parent will not bring more information about the probability, but other children may have meaningful longer parents.

The second model is more general, while the first model is quite restrictive. In the experimental section we check which model corresponds to the real dataset of moving trajectories (Section 4.5.2).

If the model of a certain data stream is assumed, or proven, to be uniform, this opens the possibility to efficiently prune longer contexts that contain excessive information.

4.1.2 Pruning Strategies

In this section we consider benefits and drawbacks of possible ways of pruning the PPM trie (we introduced the PPM algorithm and data structures in Section 2.3.4). Online strategies are listed in Table 4.1.

All three strategies optimize a criteria related to the LogLoss of the model, as the LogLoss depends both on the conditional probability and on the frequency of a sequence. The higher the conditional probability and the frequency of subsequences, the lower the LogLoss.

Table 4.1: Characteristics of the online pruning strategies.

Properties	Pruning strategies		
	1	2	3
	Frequency based	Conditional Probability Based	Combination of 1 and 2
1)Description of the strategy	when the space budget is consumed we delete those leaves of the trie which have the lowest frequency	when the space budget is consumed we delete those leaves of a trie that have the lowest conditional probability given the path from a root to a leaf	this is a combination of strategies 1 and 2, when we prune those leaves of a trie, which have the lowest value of the combination of a conditional probability and a frequency
2)Benefits	Easy to implement. It is also shown that states with lower support have low prediction accuracies	If the conditional probability is small it can be neglected for the model	minimization of LogLoss
3)Additional Space Needed	Binary search tree of leaf paths (array of path values, criteria value) according to the minimal conditional probability of a path		
4)Additional operations needed	Update of the sorted leaf path structures and the trie after an element was deleted		
6)Reintroduction	Additionally, keep the maximal frequency of a deleted child in each node and reintroduce it with a new child		
7)Incremental usage	Possible		

For all the three strategies, pruning can be made in constant ($O(1)$) time, at the cost of maintaining the pointers to the leaf nodes of a trie in the order of increasing value of the pruning criterion.

This can be implemented using the sorting structures, such as the binary search tree, which is updated online with every new arriving trajectory.

Overview of other pruning strategies. The strategies described in the following are computationally expensive, and thus can only be applied periodically in a streaming setting.

Minimal Description Length. Minimal Description Length (MDL) principle, which is described in paper [87], is used there to prune binary decision trees for classification task. The principle can also be used for the general purposes of estimating a data model. There are several difficulties in adapting this principle to online pruning. Applying the MDL principle involves estimating the costs of multiple trees and their ability to represent the data seen so far. As we would like to prune our trees in real time, this procedure has to be performed every time the space budget is reached. On the other hand, MDL is

an objective pruning strategy that has theoretical foundations, in contrast to heuristic approaches.

Distribution of children. Prior algorithms for (hierarchical) heavy hitters worked by expanding internal nodes that were "heavy", and collapsing nodes that were "light" into their parents. However, we might want to expand the nodes where the children seem to have "different" distributional behavior from their siblings, and collapse nodes where all siblings seem to behave similarly. We could also measure the entropy of the child distribution in order to chose which parents should be collapsed (the larger the entropy, the more uniform the distribution is).

Conditional entropy and entropy for the Markov processes are described in the work [14]. In general, in order to compute the entropy of the whole chain, we need to estimate the probabilities of all possible contexts. Then, we can compare how much entropy each sibling node produces and what is the distributional similarity between the sibling nodes. We can also estimate the amount of information we will gain/lose by splitting/deleting a node. Unfortunately, the computational cost of this approach is prohibitive in a streaming setting.

Relying on the uniformity of the VMM. If the VMM model is uniform (see Section 4.1.1), then, for any context that is considered meaningful, all longer contexts, containing this context as a suffix, can be pruned form the model.

In the next section we obtain and discuss a pruning strategy that is optimal in the sense of the minimal LogLoss of the generative model represented by VOCHH. LogLoss is a measure of the goodness of fit of generative models, discussed in Section 2.3.5.

4.2 Pruning Strategies for the LogLoss Function

In this section, we propose a new pruning strategy that is optimal with respect to minimizing the LogLoss of the model described by VOCHHs.

Lemma 7. *Suppose that the distribution of the streaming data does not change over time. The approximate VMM model with limited memory budget, where nodes are pruned based on the rule:*

$$x = \underset{x}{\operatorname{argmin}}\, Fr[y^k, x] \cdot \log \frac{Pr[x|y^k]}{Pr[x|y^{k-1}]}, \tag{4.1}$$

where y^{k-1} is a suffix of y^k, minimizes the LogLoss on the training data.

Proof. Approximate VMM model represents the variable order probabilistic model that defines the probability distribution $\hat{P}(\cdot|\cdot)$ Given the data x^m, the best approximate model should minimize the LogLoss for x^m:

$$l(\hat{P}, x^m) = -\frac{1}{m}\sum_{i=1}^{m} \log \hat{P}(x_i|x_1, x_2, ..., x_{i-1}) \rightarrow min. \tag{4.2}$$

As we consider the bounded order of the model and the markovian assumption, that after a certain order the longer contexts lead to the same probability as their sorter suffix contexts, the minimization problem is equivalent to:

$$\sum_{i=1}^{m} \log \hat{P}(x_i|x_{i-k}, ..., x_{i-1}) \rightarrow max, \tag{4.3}$$

where $k \leq D$. Expression 4.3 can be read as the maximization of the log-likelihood of the sequence given the model. Grouping the repeating subsequences together, we can transform the Expression 4.3 as follows:

$$\sum_{unique(x^k)} Fr(x_{i-k}, ..., x_{i-1}, x_i) \cdot \log \hat{P}(x_i|x_{i-k}, ..., x_{i-1}) \rightarrow max. \tag{4.4}$$

When pruning is performed, the probabilities of the terms under the sum in Equation 4.4 are substituted with conditional probabilities given the shorter parent $(x_{i-k+1}, ..., x_{i-1})$, which is a suffix of the corresponding parent $(x_{i-k}, ..., x_{i-1})$. Hence, pruning makes the model of the data more independent. When selecting the node to be pruned, in order to minimize the LogLoss (maximize the likelihood), we should pick the node, for which substituting its conditional probability with the conditional probability given the shorter parent results in the smallest change in the expression 4.4. In other words, the best candidate for pruning (x_{prune}) is defined as follows:

$$x_{prune} = \underset{x_i}{\operatorname{argmin}}\, Fr(x_{i-k}, ..., x_{i-1}, x_i)(\log \hat{P}(x_i|x_{i-k}, ..., x_{i-1}) - \log \hat{P}(x_i|x_{i-k+1}, ..., x_{i-1})).$$

or

$$x_{prune} = \underset{x_i}{\operatorname{argmin}}\, Fr(x_{i-k}, ..., x_{i-1}, x_i) \log \frac{\hat{P}(x_i|x_{i-k}, ..., x_{i-1})}{\hat{P}(x_i|x_{i-k+1}, ..., x_{i-1})}. \tag{4.5}$$

Equation 4.5 directly corresponds to the condition 4.1 of the lemma. □

Thus, the expression that defines the "usefulness" of a parent-child pair (y, x) is the logarithm of the ratio between conditional probabilities of the child given the longer parent $(P(x|y))$ and the shorter parent $(P(x|y'))$. In particular, this means that the parent-child pairs that do not increase the conditional probability compared to the shorter parent $(P(x|y) <= P(x|y'))$ will be pruned first.

Definition 8 (Merit of a parent-child pair)**.** *Given a parent-child pair (y, x), and the longest suffix y' of the parent y, the value of*

$$M(y, x) = Fr[y, x] \cdot \log \frac{Pr[x|y]}{Pr[x|y']} \tag{4.6}$$

is called the merit of the parent-child pair (y, x).

The higher the merit of the parent-child pair is, the more importance it has to the VMM modeling.

The optimal pruning strategy described by Lemma 7 is offline, because when any counts of a path w of the length $< D$ is updated, all the criteria values of the longer paths that have w as a suffix should be updated as well. This requires either traversing the whole trie during pruning, or keeping track, for every sequence w, of all the paths that have w as a suffix. Adapting the optimal criteria based on Equation 2.5 to online pruning is our future work. In this chapter we analyze several alternative, more lightweight, pruning strategies that can be used online.

The main challenge of the optimal pruning criterion is in the computation of the conditional probability of a child given the suffix of its parent. The problem is that in a prefix trie there is no direct link between these two elements and it is hard to find all the elements that correspond to the same suffix, when the count of the suffix is updated. As discussed in Section 4.1.2, criteria based on the frequency and/or conditional probability can be easily incrementally updated online. Below we consider which assumptions are made in case of more lightweight pruning strategies as compared to the optimal pruning.

Minimizing LogLoss: Online Pruning Strategies. Here we discuss variations of the optimal pruning that can be used in real time, and their limitations.

Frequency-based pruning.

Widespread and popular pruning of the VMM models based on the compression algorithms is the frequency-based pruning. This pruning is the easiest and the fastest. The assumption that it makes compared to the optimal merit-based pruning is that the ratio of the probabilities is constant for all the trie paths. This means that for this kind of pruning the values of the conditional probability and its meaningfulness are not taken into account. This approach favors the most frequent subsequences, regardless of the state transition knowledge they might or might not have.

Pruning based on conditional probability. This pruning strategy is used in our algorithm for detecting Conditional Heavy Hitters, which we described in Chapter 3. To the best of our knowledge, this type of pruning has not been used before for variable order modeling. This approach ignores the frequency of the candidate sequence or the conditional probability given its longest suffix, but it takes into account the strength of the state transition, which is important for estimating the highest transitional probabilities of the model. It is also quite lightweight, though it requires additional computation. Updating the count of a node changes not only the conditional probability of that node, but also that of its siblings, which means that the value of the pruning criterion has to be updated for them as well.

Entropy pruning. This kind of pruning where the criterion is $Fr(y^k, x) \log Pr[x|y^k]$ assumes that the probability given the corresponding suffix $Pr[x|y^{k-1}]$ is the highest and equals unity. This criterion is still lightweight and can be easily calculated incrementally.

Experimentally, we have noticed that for the limited budget case, the logarithm function leads to poorer VOCHH estimation, as small differences in probability transform into much larger differences in their logarithms. At the same time, high probability values become closer to each other, and, when multiplied by frequencies this leads to erroneous pruning of many elements. This phenomenon is illustrated in Section 4.5.4 by the plots with the LogLosses (Figure 4.9) of VOCHH algorithms for different pruning strategies and various space budget requirements.

Multiplication pruning. Because of the approximation issues of the entropy criterion $Fr(y^k, x) \log Pr[x|y^k]$ discussed above, we use the simpler criterion $Fr(y^k, x) \cdot Pr[x|y^k]$, as according to the Taylor series $log(z)$ is well approximated by $z - 1$ if z is close to 1 (Figure 4.1). According to our experimental results, VOCHH with this pruning criterion are well estimated in the limited memory settings (Section 4.5.4).

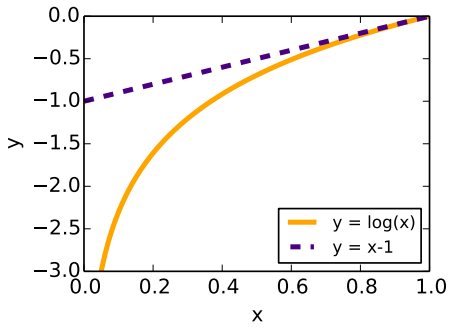


Figure 4.1: Approximating $\log(x)$ with $x - 1$ for x close to 1.

All the pruning strategies discussed in this Section are experimentally evaluated for real datasets in Section 4.5.4.

4.3 Problem Definition

Our goal is to detect VOCHH with the order up to D for sequential data streams in the case of the limited space budget. Having a data stream $X = (x_1, x_2, ...)$, where $x_i \in \Sigma$, Σ is a finite alphabet, we would like to detect those parent-child pairs (y_i^D, x_i) where the conditional probability $Pr[x_i|y_i^D]$ and $Fr[x_i|y_i^D]$ are the highest and the parent $y_i^D = (x_{i-1}, x_{i-2}, ..., x_{i-k})$ has a variable order k, $k \leq D$. In general case D might be infinite.

As VOCHH represent the generative model of a data stream, we are interested in both high conditional probability, which is associated with the transition probabilities of the model, and in high frequency, which is connected to the importance of the sequential pattern represented using VOCHH. Besides, in order to minimize LogLoss (Equation 2.5)

(which is the negation of the sum of conditional probabilities of the sequences in the test dataset) we need to extract those VOCHH elements that have both high frequency and high conditional probability.

If the size of the alphabet Σ is high, the exact solution to this problem is infeasible, as it requires estimating $|\Sigma| + |\Sigma|^2 + |\Sigma|^3 + ... + |\Sigma|^D$ frequencies. In case of bounded D it is a geometric series with a sum $\frac{1-|\Sigma|^{D+1}}{1-|\Sigma|}$. For example, if $|\Sigma| = 1000$ and $D = 5$, we need to keep track of 10^{15} frequencies. This demonstrates the need for the approximate solution.

The highest conditional probabilities and frequencies of parent-child pairs can be taken into account using a threshold semantics, for example, $Pr[x_i|y_i^D] \geq \phi$, or $top{-}\tau$ semantics, when only τ parent-child paris with the highest conditional probability and/or frequency should be extracted. We are interested in both definitions, as both can be useful for different application scenarios.

Highest conditional probabilities of the parent-child pairs alone are not sufficient, as they can correspond to the pairs with very low support or overall frequency. We call this phenomenon *(1,1)-frequency problem*. An example of this problem is when both the parent and the parent-child pair appear only once in the dataset, in which case their conditional probability is maximum and equals to 1. This case is not interesting for us as the highest conditional probability does not have enough statistical evidence and it could happen by chance. The (1,1)-frequency problem is very challenging for the variable order approach, as when the order increases we observe more and more sequences that appear only once. This problem also arises for fixed-order Conditional Heavy Hitters (Section 4.5.1) and it is solved there by the definition of popular CHH, which take into account the frequency of a parent-child pair as well.

4.3.1 Formalization with Statistical Significance

In order to formally define Conditional Heavy Hitters of variable order, we use the tools of statistical hypothesis testing. Specifically, we perform a statistical test measuring if the frequency of a parent-child pair is significantly high, compared to what could occur by chance in case of independent variables. Additionally, we check if there is a statistically significant connection between the parent and the child. The latter test shows whether the parent and the child co-occur more frequently than would happen by chance if they were independent.

Additionally, we would like to store only the meaningful parent-child pairs as per Definition 2. With a meaningful parent-child pair, the conditional probability of the child given the parent should be different from the conditional probability of the child given any suffix of the parent. Thus, the ratio between the probabilities of the longer and shorter contexts should be significantly different from 1. Using the notions of statistical

significance and meaningfulness we can define VOCHH as follows:

Definition 9 (VOCHH)**.** *Consider a data stream $X = (x_1, x_2, ...)$, where $x_i \in \Sigma$, Σ is a finite alphabet. Consider a set of parent-child pairs (y_i^D, x_i), where parents $y_i^D = (x_{i-1}, x_{i-2}, ..., x_{i-k})$ have a variable order k, $k \leq D$. The pairs (y_i^D, x_i) are called VOCHH if:*

1. *A merit of a parent-child pair (Definition 8) is high, which means that $M(y_i^D, x_i) \geq \phi$, or (y_i^D, x_i) is among top-τ elements sorted according to $M(y_i^D, x_i)$.*
2. *$Pr[x_i|y_i^D]$ should be meaningful.*
3. *$Fr[y_i^D, x_i]$ is statistically significantly higher than the expected frequency of (y_i^D, x_i) in case when all symbols were independent random variables.*
4. *$Fr[y_i^D, x_i]$ is statistically significantly higher that the expected frequency of (y_i^D, x_i) in case when the parent y_i^D and the child symbol x_i were independent random variables.*

The constraint 1 on the merit of the parent-child pairs is related to minimizing the LogLoss of the VOCHH model (lemma 7). This constraint also helps us avoid using two separate constraints on support and confidence that are popular for Association Rules, Frequent Itemsets, and is adopted by Probabilistic Suffix Trees as well. In our approach we used as few threshold parameters as possible.

The requirement of meaningfulness of the parent-child pair $((y^D, x))$ is inspired by the condition used in the Probabilistic Suffix Trees [104] and is checked with the following conditions:

$$\frac{Pr[x|y^D]}{Pr[x|y^{D-1}]} > 1 + \varepsilon \quad \text{or} \quad \frac{Pr[x|y^D]}{Pr[x|y^{D-1}]} < \frac{1}{1+\varepsilon}, \tag{4.7}$$

where $\varepsilon > 0$ and y^{D-1} is the largest suffix of a parent y^D.

In order to conduct the tests induced by the constraints 3 and 4 from Definition 9, we need to build a statistic with the known distribution. At the same time, the value of this statistic should depend on the properties of the data. We use statistical tests based on the binomial distribution of the random independent variables. The confidence level α is set to be 0.01. Thus, these probability of true negatives is 0.99.

We do not control the II type error (the false positives) as it is computationally prohibitive in streaming settings. Similar statistical tests together with more advanced control of the type II error are used to check the significance of frequent itemsets and association rules in the works [61, 73]. These methods described in these words require heavy computation. In case of VOCHH, the computations would have to be even more complex: if association rules represent combinations without repetitions from a set of items,

the VOCHH represent variations with repetitions, in which the order of the symbols is important.

The tests of the statistical significance for the parent-child frequency together with the Entropy-based constraint is a great alternative to the user-defined thresholds common for CHH of the fixed order (Chapter 3) and related notions, such as Heavy Hitters [34, 84], Association Rules [3], Frequent Itemsets [64], Correlated Heavy Hitters [77], because these thresholds are arbitrary. Setting the thresholds to a low value leads to a high number of false positives – sequences of independent elements that are formed by chance, and are declared VOCHH. On the other hand, if the thresholds are too high, many interesting VOCHH can be missed.

Our Definition of VOCHH brings novelty to variable order Markov modeling by injecting statistical significance instead of usual threshold-based constraints in the area of lossless compression algorithms.

4.3.2 Hypotheses Testing

In order to check the conditions 3 and 4 in the definition of VOCHH, we keep the exact frequencies of each symbol $Fr(\sigma_1)$, $Fr(\sigma_2)$, ..., $Fr(\sigma_{|\Sigma|})$, which can be easily converted to their probabilities by dividing them on the number of symbols N seen so far: $Pr[\sigma] \approx Fr(\sigma)/N$ These frequencies are then used for building the test statistic. Thus, the significance test depends on the actual frequencies of the individual symbols, the number of the observed parent-child pairs of particular length, and the estimated frequencies of the parent and the parent-child pair. The test statistic shows if the current frequency of a parent-child pair could be generated by chance.

Total Independence Testing

Estimation of the expected by-chance frequency of a parent-child pair is based on the model, in which every symbol in the sequence is drawn independently from a categorical distribution. The probabilities of the individual symbols can be estimated from their occurrences in the stream. The probability of seeing a sequences of symbols $X = (x_1, x_2, ..., x_k)$ in this case is equal to: $Pr[X] = Pr[x_1, x_2, ..., x_k] = Pr[x_1] \cdot Pr[x_2] \cdot ... \cdot Pr[x_k]$. The distribution of the frequency of a sequence X is binomial with the parameters N and $Pr(X)$: $Fr_X \sim B(N, Pr(X))$. Thus, the frequency Fr_X of X is a random variable with known distribution that is based on the properties of the dataset (frequencies of the symbols), exactly the way we needed it for significance testing. We formulate the hypothesis as follows:

- H_0: null hypothesis – the frequency Fr_X of a parent-child pair is not significant with respect to the seen part of a data stream.

- H_1: alternative hypothesis – the frequency Fr_X of a parent-child pair is significant. It is unlikely that the frequency comes form the random i.i.d case.

After setting the significance level α, which corresponds to the probability of rejecting H_0 when it is true, we test the hypothesis by checking the following p_{value}:

$$p_{value} = Pr[B(N, Pr(X)) \geq Fr_X].$$

In case if $p_{value} \leq \alpha$ the null hypothesis H_0 is rejected, and the frequency of the parent-child pair is considered to be statistically significant. Due to the existing lookup tables for binomial distribution, p_{value} can be obtained in constant (O(1)) time.

Parent-Child Independence Testing

Similar test can be used to check the significance of the parent-child co-occurrence frequency. Using this test, we simply check if there is a connection between a child x and a parent y^k. In the independent case $Pr[y^k, x] = Pr[y^k] \cdot Pr[x]$.Our goal is to verify weather the frequency of a parent-child pair $Fr_{(y^k,x)}$ could be generated if y^k and x were independent variables. In this case, we consider the distribution of the support (frequency) of a parent-child pair $Fr_{(y^k,x)}$, which in the random case is again distributed as the Binomial random variable. The difference with the previous test is that we consider the parent sequence as a single random variable drawn from the categorical distribution, and check whether this variable is independent from the child symbol.

Thus, after calculating the probabilities of the parent ($Pr[y^k] = Fr_{y^k}/N_k$) and the child ($Pr[x] = Fr_x/N$), we compute $Pr[x, y^k]$ as their multiplication, thus treating them as independent categorical variables. In this case, the components of the hypothesis are the following:

- H_0: null hypothesis – there is no significant dependence between the parent and the child, according to the observed part of the data stream: $Pr[y^k, x] \approx Pr[y^k] \cdot Pr[x]$.
- H_1 alternative hypothesis – there is a significant dependency between the parent and the child.

In order to reject H_0 we compare the following p_{value} with the significance level α.

$$p_{value} = Pr[B(N, Pr[y^k] \cdot Pr[x]) \geq Fr_{(y^k,x)}].$$

The effect of the hypothesis testing on the exact VOCHH candidates for three real datasets is discussed in Section 4.5.1.

4.4 Solution Approach

In the thesis to solve the problem of VOCHH discovery we adopt the best performing PPM algorithm with PPM-C back-off model to the settings with limited space budget.

We consider several pruning strategies that can be implemented online. The best strategy that can be used for pruning is the optimal pruning described in lemma 7, where pruning criterion is the merit of the VOCHH candidate. As we have not found a way to use this strategy in real time, we use other pruning criteria, discussed in Section 4.2. Although the strategies are heuristics, they correspond to different special cases of the optimal pruning strategy.

One of the natural strategies is the multiplication pruning, which prunes the nodes with the minimal value of the frequency multiplied by the conditional probability. This strategy is an extension used for conditional space saving (CSS) structure, which is designed in order to extract Conditional Heavy Hitters of the fixed length (see Chapter 3). It combines two widely used criteria, namely frequency and conditional probability, which correspond to support and confidence used in Association Rules literature.

We call the algorithm for VOCHH estimation VariableCHH algorithm. The variation of VariableCHH with the reintroduction of new subsequences as described in Section 4.4.1 is called VariableCHHr.

The PPM-like prefix trie used for both VariableCHH and VariableCHHr is filled in with all observed sequences up to the maximum predefined length D.

When the space budget is exhausted, the pruning of the VOCHH candidates is done according to one of the variations of the merit constraint 1 from Definition 9. Auxiliary structures, such as binary search trees, are used in order to perform the pruning. The auxiliary structure keeps the pointers to the leaves of the tree, sorted according to the value of the pruning criterion. The leaf node with the smallest value of the criterion value is pruned. Given the linear complexity of the PPM update and the sublinear complexity of the update of auxiliary structures, the overall running time of VariableCHH and VariableCHHr is $O(logM)$ per element, where M is the number of nodes in the VOCHH trie.

For practical reasons, we also use a threshold on the minimal frequency ψ_{min}, as for the high order sequences, if the symbols are rare, even very infrequent sequences can be treated as significant by the independence test. We use the threshold as follows:

$$Fr[y_i^D, x_i] \geq \psi_{min}. \tag{4.8}$$

The algorithm can use several enhancements when the speed of the data stream allows it. We call the first enhancement *deep digging*. Before excluding the element with the lowest value of the criterion, we perform the statistical tests 3 and 4 and check the condition and 2 (Definition 9), and exclude the element only in the following cases:

1. there is not enough evidence to reject H_0,
2. the corresponding conditional probability is not meaningful,
3. the minimum support condition from Equation 4.8 is not satisfied.

If the leaf node with the smallest value of the criterion is not pruned, we proceed to the element with the second lowest value, and so on until we have checked ν first elements or some element has been pruned. If none of the checked elements gets pruned, we delete the element with the minimum value of the criterion. Parameter ν defines how deep we can dig into the auxiliary structure of sorted leaves. The value of ν can be decided based on the speed of the stream and can be as low as 1 in case when the minimum running time is necessary.

The second enhancement is called *periodic pruning*. It is a global pruning procedure that scans through the whole VOCHH trie, deleting the elements based on the constraint 2 and Equation 4.8.

Before describing the algorithm, we would like to notice that the data stream of symbols is usually divided into trajectories or strings, which correspond to different initial states of the process. As the symbols of one trajectory do not depend on the previous trajectories, the trajectories are considered separately. VariableCHH algorithm is formalized in the pseudocode of Algorithm 6. P^{min} is the auxiliary structure that is used for pruning.

Thanks to the tests of statistical significance, VariableCHH algorithm is able to prune the paths where the elements are not dependent on each other, or where there is low correlation among the elements. This leads to smaller memory requirements because if the elements in the sequence $(x_{i-D}, ..., x_{i-1}, x_i)$ are not dependent, then, the corresponding parts of the sum in equation 4.3 are transformed to terms: $Fr(x_i) \cdot \log \hat{P}(x_i)$. These terms are estimated accurately by the model, as the frequencies of the single symbols are kept exactly, while the paths that correspond to the independent elements are pruned according to the conditions 3 and 4 of Definition 9.

The VariableCHHr algorithm described in the next section has the same variations as VariableCHH depending on the pruning criterion used.

4.4.1 VariableCHHr Method

If some nodes in a trie are pruned, the information about their initial frequency is lost and cannot be restored when the nodes are reintroduced afterwards. In order not to miss the VOCHH candidates due to underestimating their frequency, we can reintroduce the new coming nodes using one plus the maximum frequency of a previously deleted node for a particular parent. When we reintroduce not a single symbol but a path that follows a particular context node with nonzero maximum frequency, all the nodes further in the path are reintroduced with the frequency equal to one plus the maximum frequency of a deleted node, and their corresponding maximum frequencies are assigned to be the maximum frequency of the parent node. VariableCHHr algorithm is formalized in the pseudocode of Algorithm 6. Lines 2, 20 and 24 represent the reintroduction.

This reintroduction is similar to reintroduction strategies proposed for the fixed order

Algorithm 6 VariableCHH for VOCHH discovery

Input: Data stream of symbols $X = \{x_i\}, i = 1, 2, ...$ that constitute the trajectories $t_j, j = 1, 2, ...$ D - the maximum length of the context M - number of elements in the trie (memory budget) ν - digging depth for the deep digging enhancement

Output: T - VOCHH trie of maximum size M

1: P^{min} – a binary search tree, which keeps the pointers to all leaves of T for paths of particular length $\leq D$, where the elements are sorted according to the value of the pruning criterion
2: **for** each element $(y_i^k, x_i) \in t_j$, $i = 1, ..., |t_j|$, $k \leq D$, $j = 1, 2, ...$ **do**
3: **if** $(y_i^k, x_i) \in T$ **then**
4: **for** each symbol s in path (y_i^k, x_i) **do**
5: $T[s \in (y_i^k, x_i)] = T[s \in (y_i^k, x_i)] + 1$
6: **end for**
7: **else**
8: **if** $|T| \geq M$ **then**
9: deepdigging_finished = false
10: **for** First ν elements $s \in P^{min}[pruningLength]$ and deepdigging_finished == false **do**
11: **if** $s.goodForPruning()$ **then**
12: $candidate = s$
13: deepdigging_finished = true
14: **end if**
15: **end for**
16: **if** deepdigging_finished == false **then**
17: $candidate = P^{min}[pruningLength].first$
18: **end if**
19: $T.delete(candidate)$
20: $P^{min}.delete(candidate)$
21: **end if**
22: $T.insert((y_i^k, x_i))$
23: **end if**
24: **end for**

Conditional Heavy Hitters discussed in Section 3.4. The fact that the true frequency of the parent-child pair is always smaller or equal to the reintroduced frequency kept in the VOCHH trie guarantees that the true frequency is never underestimated. In Lemma 8 we show that, with this reintroduction strategy, the corresponding conditional probabilities are not underestimated either.

Lemma 8. *For every context y of order $k \leq D$, and every child x of y, such that both x and y are in the VOCHH trie, VariableCHHr algorithm produces an estimated conditional probability $P_k^*[x|y]$ that overestimates the probability $\hat{P}_k[x|y]$ produced by the exact lossless PPM-C algorithm:*

$$P_k^*[x|y] \geq \hat{P}_k[x|y]. \tag{4.9}$$

Algorithm 7 VariableCHHr for VOCHH discovery

Input: Data stream of symbols $X = \{x_i\}, i = 1, 2, ...$ that constitute the trajectories $t_j, j = 1, 2, ...$ D - the maximum length of the context M - number of elements in the trie (memory budget) ν - digging depth for the deep digging enhancement

Output: T - VOCHH trie of maximum size M,

1: P^{min} – a binary search tree, which keeps the pointers to all leaves of T for paths of particular length $\leq D$, where the elements are sorted according to the value of the pruning criterion
2: $R(t)$ – reintroduction value for children of a path t, $R(t)$ equals to the maximum frequency of a deleted child from t, if no children have been deleted or pruned $R(t) = 0$. $R(t)$ is kept within the trie T for each node.
3: **for** each element $(y_i^k, x_i) \in t_j$, $i = 1, ..., |t_j|$, $k \leq D$, $j = 1, 2, ...$ **do**
4: **if** $(y_i^k, x_i) \in T$ **then**
5: **for** each symbol s in path (y_i^k, x_i) **do**
6: $T[s \in (y_i^k, x_i)] = T[s \in (y_i^k, x_i)] + 1$
7: **end for**
8: **else**
9: **if** $|T| \geq M$ **then**
10: deepdigging_finished = false
11: **for** First ν elements $s \in P^{min}[pruningLength]$ and deepdigging_finished == false **do**
12: **if** $s.goodForPruning()$ **then**
13: $candidate = s$
14: deepdigging_finished = true
15: **end if**
16: **end for**
17: **if** deepdigging_finished == false **then**
18: $candidate = P^{min}[pruningLength].first$
19: **end if**
20: $R(parent(candidate)) = T(candidate)$
21: $T.delete(candidate)$
22: $P^{min}.delete(candidate)$
23: **end if**
24: $T.insert((y_i^k, x_i), R(y_i^k) + 1)$
25: **end if**
26: **end for**

Proof. Consider Equation 2.3 for a context y and a symbol x that is a child of y, such that both x and y are in the trie by the end of the algorithm. The equation can be rewritten as follows:

$$\hat{P}_k[x|y] = \frac{Fr[y, x]}{|\Sigma_y| + \sum_{x' \in \Sigma_y} Fr[y, x']}. \tag{4.10}$$

The maximal frequency y_{max} of any deleted child x' that has been seen after y is kept in the node corresponding to last symbol of y. When the new child comes, it is reintroduced with the count $Fr[y, x] = y_{max} + 1$. Note that all the frequencies of the observed children x kept

in the approximate trie are greater or equal to the maximal frequency y_{max}, as otherwise the child would have been pruned according to frequency-based or conditional probability-based pruning strategies. Consider a child a of context y and different possibilities for how its conditional probability changes after possible deletions and reintroductions:

1. If a was deleted and then reintroduced: $Fr^*[y, a] = Fr[y, a]$, and the denominator of Equation 4.10 also remains the same. In this case Inequality 4.9 holds true.

2. If another child b was deleted – the numerator remains the same, while the denominator contains a lower count of outgoing edges $|\Sigma_y|$ and a smaller sum of seen frequencies. Thus the new ratio $P_k^*[a|y]$ with the lower denominator is strictly greater than the exact ratio $\hat{P}_k[a|y]$ in the trie and Inequality 4.9 holds true again.

3. If b was deleted and then b or a previously seen child c was reintroduced – the numerator of $\hat{P}_k[a|y]$ remains the same, while the denominator contains the same or lower count of the outgoing edges $|\Sigma_y|$ and the same or smaller sum of the observed frequencies. Thus, the new ratio $P_k^*[a|y]$ with the equal or smaller denominator will still be larger than the exact ratio $\hat{P}_k[a|y]$ in the trie, and Inequality 4.9 holds true.

4. If another child b was deleted and then a new child c was reintroduced – the numerator remains the same, while the denominator contains the same count of the outgoing edges $|\Sigma_y|$ and the same sum of the observed frequencies. Thus, the new ratio is exactly the same as the corresponding ratio in the lossless trie, and Inequality 4.9 holds true.

The same analysis holds when b is a sibling path of a rather than a sibling symbol. This is because in equation 4.10 the probability of x depends only on its frequency, and on the number and frequencies of its direct siblings.

The difference between the conditional probabilities in the approximate and the exact tries depends on the possible number of children per each node. Thus, if the number of children is bounded for a particular domain, than tighter guarantees could be provided. □

Lemma 8 guarantees that the conditional probability of the reintroduced node, estimated by VariableCHHr, is never lower than the true conditional probability. As both frequency and conditional probability are overestimated, with the pruning strategies based on their combination, VOCHH should not be missed, though the might be false positives due to the limited space. Thus, the recall of this method is higher than the the recall of VariableCHH if there is enough memory to keep certain amount of highly overestimated nodes which are not VOCHH, but the precision can suffer if multiple reintroductions of rare nodes are made.

4.4.2 Approach Validation

We have two types of validation of our approach. First, it is the validation of the ability of the VariableCHH and VariableCHHr algorithms to retrieve VOCHH in the streaming settings with limited memory budget. Second, we need to validate the ability of VOCHH to restore the generative model of the data.

For the VOCHH extraction we use (a) standard measures of precision and recall; (b) top-τ precision[2]; and (c) average until τ precision for VOCHH, where the average is taken over all top-k sets of VOCHH, where $k = 1, 2..., \tau$ relative to the "true" set; We use these measures for each order separately, as well as overall. (d) We also estimate the mean absolute error of the estimated conditional probability of VOCHH compared to its true value.

For the second validation goal, similarly to [13], we train the VOCHH model on a part of data (training dataset) and then evaluate it on another part of data (test dataset) using the LogLoss function, introduced by Equation 2.5.

4.5 Experimental Evaluation

4.5.1 Datasets and Preliminary Analysis

We consider three real datasets in this study: WorldCup'98 dataset, Taxicab dataset, and BUWeb dataset.

The WorldCup data[3] contains information about the requests made to the World Cup website during the 1998 tournament. Each request contains a ClientID (a unique integer identifier for the client that issued the request), time label and an ObjectID (a unique integer identifier for the requested URL). Given this information, we extract usage sessions that are accessed by the same ClientIDs, and are sorted by ObjectIDs. We consider a new session to start if there is a gap of more than 30 minutes between successive observations.

We used data from day 41 to day 46 of the competition. The total number of records in this period is more than 104 million; the number of distinct ClientIDs is 540K; the number of distinct ObjectIDs, corresponding to the number of symbols in the alphabet, is 21606; the number of extracted sessions is 957K and the average number of observations per session is 109.

The Taxicab data consists of about 20 million GPS points for a fleet of taxis, collected over the course of a month, obtained from cabspotting.org. To go from the fine-grained GPS locations to streams of values, we performed pre-processing to clip the data to a

[2]Note that when restricting output to size exactly τ, precision and recall are identical, so we do not duplicate this measurement.

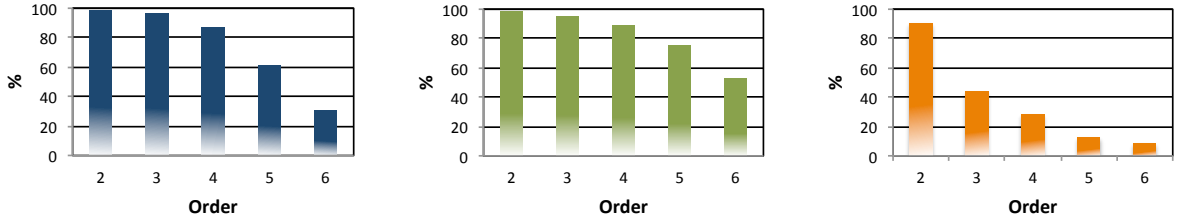
[3]`http://ita.ee.lbl.gov/html/contrib/WorldCup.html`

bounded region and coarsen it to a grid. The measurements are restricted to a rectangular region in the area of San Francisco, with latitude in the range [37.6...37.835], which covers 26km and longitude in the range $[-122.52... - 122.35]$, which covers 15km. The clipping was performed in order to remove a few incorrect readings which were far outside this region.

The space was partitioned into 10,000 rectangles using a 100×100 grid. Given the readings within this grid, we proceeded to define trajectories from the data as a sequence of grid cells occupied by the same cab. We considered a new trajectory to start if there was a gap of more than 30 minutes between successive observations. Following this definition, we extracted 54,308 trajectories.

The BUWeb data[4] contains traces of Internet network traffic made of HTTP requests from the Boston University Computer Science Department, made in the timeframe from November 21, 1994 till May 8, 1995. A trace log file contains a sequence of WWW requests that were made by one user during one session. The data contains more than 1 million requests. Based on the traces we encoded the URLs into symbols and built the "trajectories" of usage behavior, which contain the sequence of symbols from the sessions. The alphabet size in this case is 5222, the number of sessions is 10K, and the number of distinct users is 762.

Preliminary analysis. For each dataset we considered the following properties of potential VOCHH up to the 6th order. First, we calculated the number of the meaningful conditional probabilities out of the set of all possible probabilities for each order. The percentage of the meaningful VOCHH candidates among all seen VOCHH candidates for each dataset separately can be seen in Figure 4.2.



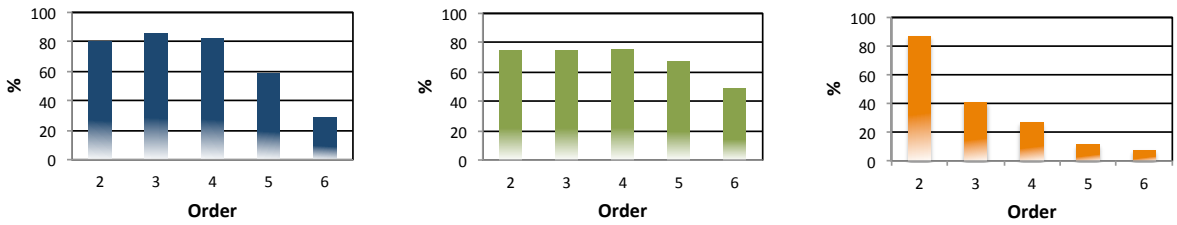
(a) WorldCup'98 dataset. (b) Taxicab dataset. (c) BUWeb dataset.

Figure 4.2: Percentage of meaningful VOCHH candidates among all observed VOCHH candidates of a particular order.

In order to minimize the LogLoss, longer contexts should be kept only if they increase the probability compared to the shorter contexts. Because of this, in the definition of meaningfulness (Equation 4.7), only the constraint that the ratio of the probabilities is

[4] `http://ita.ee.lbl.gov/html/contrib/BU-Web-Client.html`

greater than $1+\varepsilon$ should be taken into account. Thus, further in the experimental section, we call VOCHH meaningful if its conditional probability is larger than the probability conditioned on the shorter contexts. The percentage of the meaningful VOCHH of this kind for each dataset is shown in Figure 4.3.



(a) WorldCup'98 dataset. (b) Taxicab dataset. (c) BUWeb dataset.

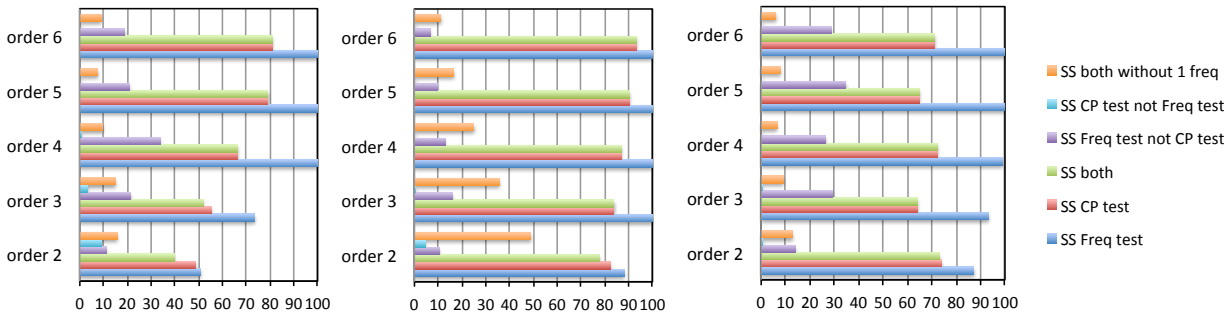
Figure 4.3: Percentage of meaningful (in the sense of LogLoss) VOCHH candidates among all observed VOCHH candidates of a particular order.

The results demonstrate that all the datasets have meaningful VOCHH candidates up to the sixth order, though their number is decreasing for higher orders. This property demonstrates the variable order nature of the data.

Our next experiment of preliminary analysis checks the statistical significance of the VOCHH candidates. We run two kinds of tests described in Section 4.3.1 in order to see the effect of significance testing on pruning the meaningful VOCHH candidates. Percentage of statistically significant VOCHH candidates (labeled as SS) among all meaningful VOCHH for different combinations of significant tests is shown in Figure 4.4, where "SS Freq test" stands for the percentage of statistically significant candidates that successfully passed the test on the frequency support; "SS CP test" stands for the percentage of statistically significant candidates that successfully passed the test on the parent-child dependence (so called child-parent (CP) test); "SS both" stands for the candidates that passed both tests; "SS Freq test not CP test" stands for the percentage of those SS that passed the frequency test but did not pass the CP test; "SS CP test not Freq test" – vice versa. The last item in the legend "SS both without 1 freq" – shows the percentage of those candidates that passed both tests and have frequency larger than 1, in this case $\psi = 1$. The orange bar shows the percentage of actual VOCHH we would like to find in those datasets according to the Definition 9.

As we can see from the analysis of WorldCup dataset illustrated by Figure 4.4a, frequency test prunes some VOCHH candidates only for small orders, thus, preferring longer VOCHH to the sorter ones. CP test also has the same tendency, but it is able to prune more candidates even for larger orders. Similar result was achieved for the BUWeb dataset.

Although "Freq" test has difficulty in pruning VOCHH candidates, when both tests



(a) WorldCup'98 dataset. (b) Taxicab dataset. (c) BUWeb dataset.

Figure 4.4: Percentage of statistically significant VOCHH among all meaningful VOCHH of a particular order. Two statistical tests are used: "Freq test" – test of independence of the symbols of the full VOCHH sequence, and "CP test" – test of independence between the parent sequences and the child symbol.

are used, at lest 10% of meaningful VOCHH candidates are pruned even for larger orders. At the same time, if CP test is rejected, almost in all cases the Freq test is also rejected. But there are some cases with small % numbers, in which the sequence is statistically significant according to the Freq test, while the last symbol is not significantly dependent on the parent.

With the increasing order it can be seen that the results of pruned VOCHH candidates are almost the same as the result of just CP test, and there are no VOCHH candidates that passed the CP test but did not pass the Frequency test.

The most interesting result is the orange bar, which contains the VOCHH candidates that passed both tests and have frequency > 1. We can see that both tests do not filter the candidates of any order with such a small frequency, meaning that the threshold ψ for the minimal frequency is important.

We focus more on the distribution of frequencies and conditional probabilities for the VOCHH candidates inside the orange bar. The distribution plots for the Taxicab data can be seen in Figures 4.5–4.6. Other detests show similar behaviour of the distributions.

According to the plots, the the distribution of the frequencies and conditional probabilities is quite reasonable for the preferred VOCHH candidates. In Section 4.5.5 we calculate the accuracy based on the top-τ VOCHH sorted by the combined conditional probability and frequency criterion, for each order separately, as well as overall. We also measure the precision and the recall of the retrieval of VOCHH for $\phi = 0$, that is, the VOCHH that satisfy the constraints of meaningfulness, statistical significance, and minimum frequency threshold.

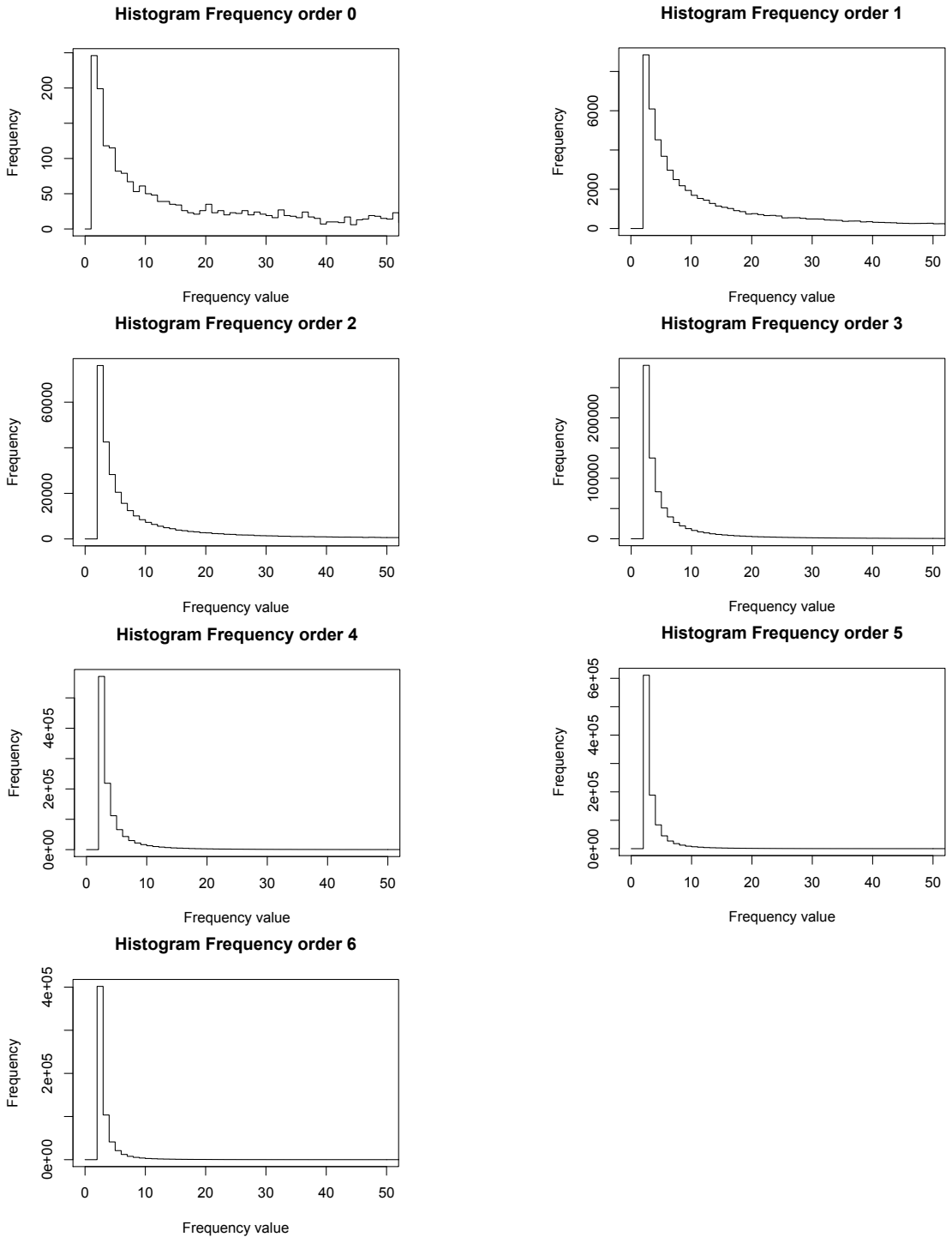


Figure 4.5: Distribution of frequencies of the parent-child pairs for meaningful statistically significant VOCHH candidates with the frequency > 1 for the Taxicab dataset.

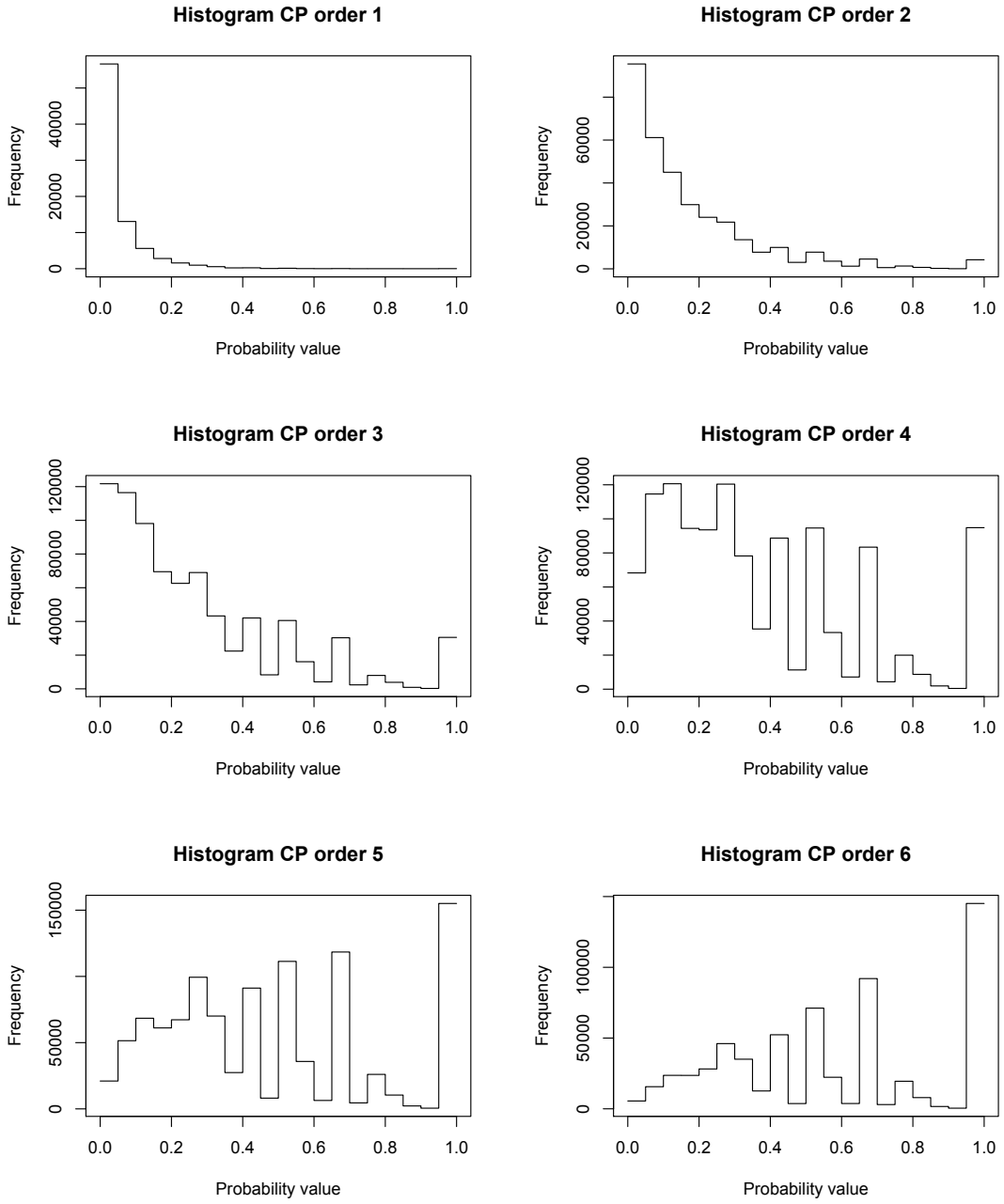


Figure 4.6: Distribution of conditional probabilities of the parent-child pairs for meaningful statistically significant VOCHH candidates with the frequency > 1 for the Taxicab dataset.

4.5.2 Uniform and Nonuniform Markov Models

In this section we check experimentally whether the Taxicab dataset is better described by uniform or the nonuniform VMM. First, we check how many meaningful VOCHH we

Table 4.2: Comparison between the number of nodes in the full VMM trie and the trie with only meaningful paths for different orders of VMM.

	order 2	order 3	order 4	order 5
fullSize	1104021	4118160	10396713	19811715
meaningfulSize	1103385	4026873	9780618	17493952
difference	636	91287	616095	2317763

have out of the whole set of exact VOCHH. The results for different maximal orders of VMM are shown in Table 4.2.

The results illustrate that the number of meaningful transition probabilities decreases with increasing order of the contexts. At the same time, as high orders still have essential number of meaningful probabilities, we can safely assume that the generative model of the dataset is of variable order.

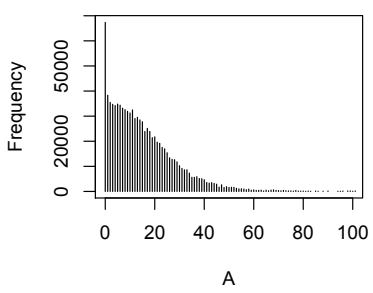
Second, for each meaningful transition probability we calculate how many sequences there are with the same parent but a different child (denoted by A), how many meaningful transition probabilities have longer parents (denoted by B), and then compare the histograms of As and Bs. When we compute A and find a path with the parent that has already produced a meaningful transition probability, we omit it from further consideration in order not to count it twice.

The plots for As and B for different orders of VOCHH can be seen in Figure 4.7. Let us remind that the order defines the length of a parent. We plot B only for the second order, because the values of Bs for larger orders are all zero, meaning that there are no longer contexts for meaningful transition probabilities that also produce meaningful transition probabilities with a different child. Similarly for As, if we look at higher orders, the distribution is more skewed, which means that there are more transition probabilities that are specific to the particular parent-child pair, where the parent does not produce any other meaningful VOCHH with other children.

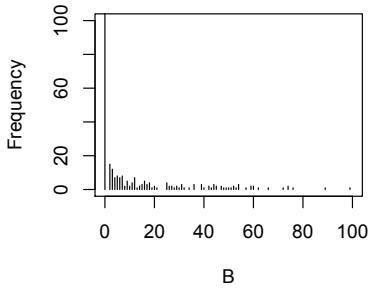
With the respect to the results, we can conclude that the Taxicab data is rather nonuniform, as there are parents of length 2 that constitute meaningful transition probabilities for some children, though for other children they are the suffixes of the longer parents with meaningful transition probabilities. At the same time, for larger orders the model of the data tends to be more uniform, because parents of the meaningful VOCHH of length 3 and 4 do not participate as suffixes of longer parents for other meaningful VOCHH.

4.5.3 Fixed Versus Variable Order

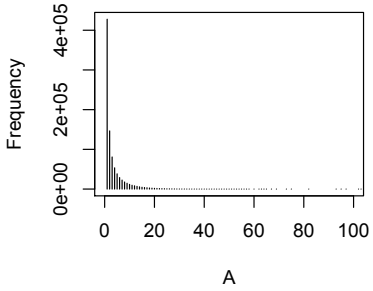
In this section we experimentally show that VMM can fit the Taxicab trajectories better than Markov Chain of the fixed order. The trajectories where randomly divided into the training (60%) and testing (40%) datasets. The estimated models were evaluated using



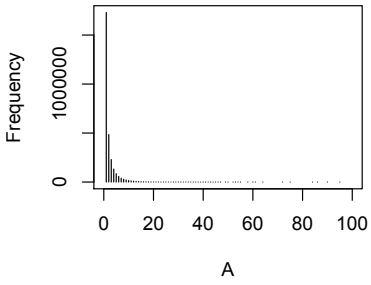
(a) Histogram of A for order 2.



(b) Histogram of B for order 2.



(c) Histogram of A for order 3.



(d) Histogram of A for order 4.

Figure 4.7: Distribution of the parameters A and B, which characterize the uniformity of VMM for Taxicab dataset.

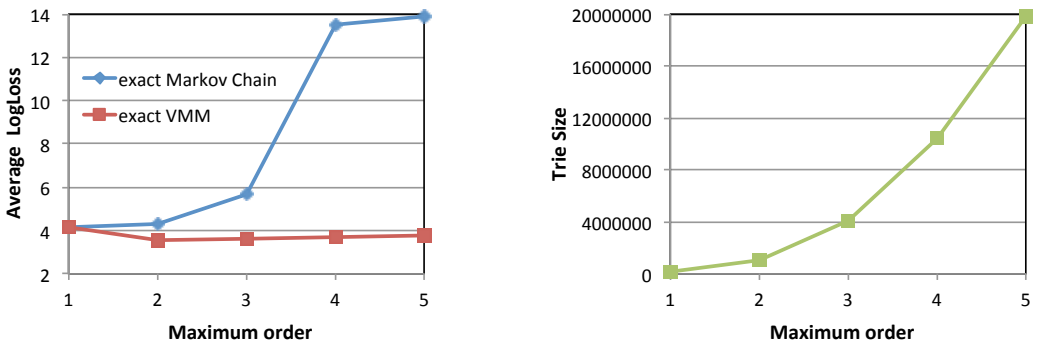
Table 4.3: Average LogLoss of the fixed and variable order Markov Models.

max order	exact Markov Chain	exact VMM
1	4.15	4.14
2	4.30	3.56
3	5.71	3.59
4	13.52	3.67
5	13.90	3.76

the LogLoss on the test dataset. We use the repeated random subsampling validation with 5 splits and report the averaged LogLoss values. The smaller the LogLoss, the better the model. In these experiments we use the exact models of particular order without any pruning. The results can be seen in Table 4.3 and Figure 4.8a.

VMM was estimated using PPM-C algorithm implemented in Java by the authors of the work [13]. We modified the implementation in order to keep the counts (frequencies) of all the intermediate nodes, as this information is essential for further pruning. Other VMM algorithms mentioned in [13] produce less accurate results.

The best fixed order result is at the first order, while the best VMM result is at the second order. We guess that the main difference is that when we have the first order estimates for VMM we can more accurately predict the first points of the trajectories. In



(a) Average LogLoss.

(b) Trie size.

Figure 4.8: Average LogLoss of the fixed and variable order Markov Models and trie size of PPM-C for different maximum orders.

the fixed order model if there is not enough context (the first few states of the trajectory) we assign a symbol to be equiprobable. The same happens when we meet a new (parent, child) pair we have not seen before.

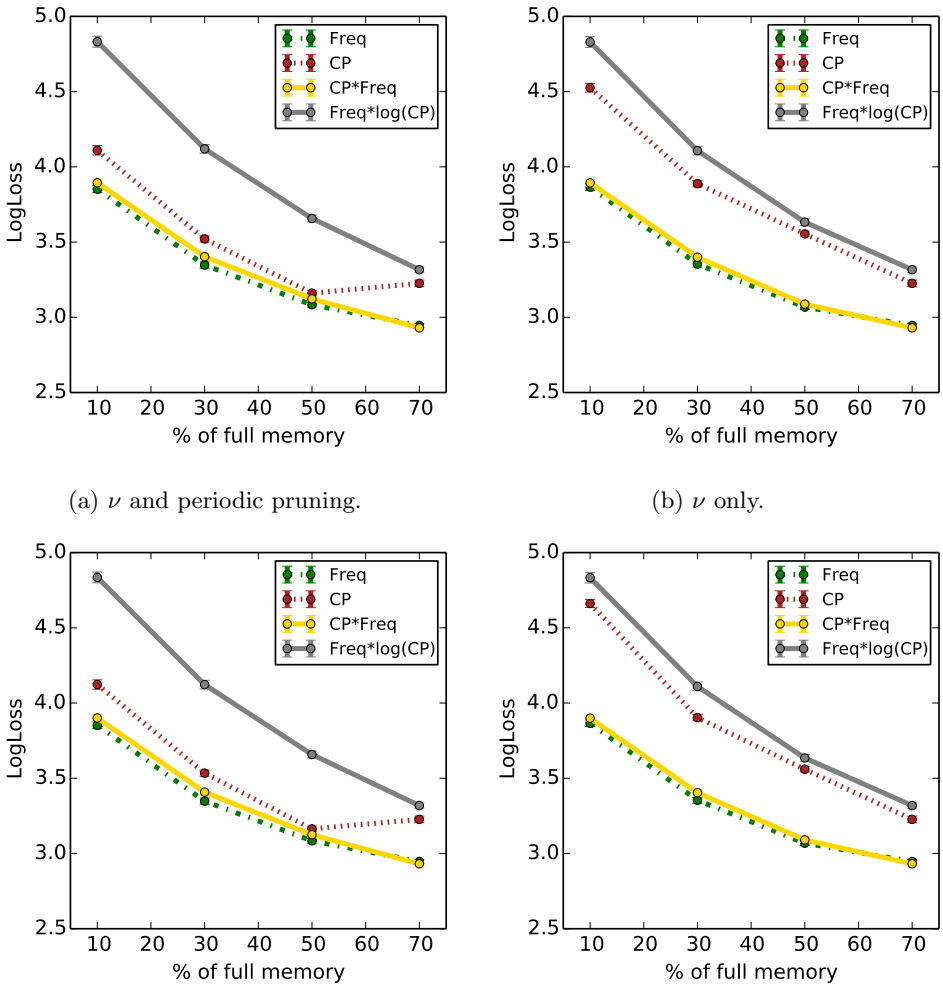
For the full PPM-C model, the size of the trie (number of nodes) for different maximal orders is shown in Figure 4.8b.

According to the results, the best model among all is VMM with the maximal order 2. VMMs of higher orders produce slightly worse results, while still being much more accurate than the best result of the Markov chain, which is reached at the second order as well. It seems that the best maximum order for this dataset is 2 or 3. The LogLoss might not be representative of the results for the fixed order, as it is possible that there is not enough training data to estimate the large number of high order probabilities. In our case of alphabet size = 10000, the LogLoss of the trivial scheme is 13.28, which is $\log_2(10000)$, meaning that the algorithms outperform the trivial solution. Hence, the first part of the sequence contains predictive information about its second part. The differences in accuracies of the exact Markov model and PPM-C is also due to the back-off mechanism used in PPM-C for treating the frequencies that were not seen before (Section 2.3).

This LogLoss experiment shows that there is not enough training data in order to accurately estimate all necessary high order transition probabilities for the Taxicab dataset. This is why the LogLoss drastically increases for the high orders of the fixed order model, while the LogLoss of VMM shows much more accurate results. This shows that, in addition to other benefits, VMM is easier to train on smaller data.

4.5.4 LogLoss as a Measure of Goodness of Fit

In this experiment we experimentally compare the results of the VOCHH estimation using different pruning variations of the optimal pruning strategy discussed in Section 4.2. So far, we do not know if it is possible to use the optimal pruning criterion in real time. From the results of the previous section, we have noticed that, when the space is limited, it is harder to estimate the VOCHH model using the entropy pruning than the multiplication pruning. The average LogLoss and its 95% confidence interval for varying memory budget experiments and different pruning strategies is shown in Figures 4.9. The average LogLoss and the confidence intervals are computed over the repeated random sub-sampling validations with 10 splits, where 60% of the data is used for training and 40% for testing.



(a) ν and periodic pruning.

(b) ν only.

(c) Periodic pruning only.

(d) Without any enhancements.

Figure 4.9: Average LogLoss for different algorithm variations for varying budget for the BUWeb data.

The results show that for the BUWeb data two pruning criteria of the VariableCHH algorithm – the frequency, and the multiplication criteria – perform the best. At the same time, the entropy criterion, which is the closest among the online implementations to the optimal pruning criterion, does not perform well in the settings of limited memory, though it tends to improve when the memory budget grows. The pruning criterion based on the conditional probability is not the best performer, though it provides better results that the entropy-based criterion.

For the BUWeb data 4.9, VariableCHH algorithm versions with pruning based on frequency, multiplication of frequency and conditional probability, and entropy are only slightly affected by the usage of enhancing mechanisms, such as deep digging, using ν, and/or periodic pruning (Figures 4.10a–4.10c). In contrast, conditional probability based pruning strategy benefits considerably from the periodic pruning enhancement. As we used a rather small deep digging parameter $\nu = 20$ for this experiments, we did not observe any significant effect of the deep digging on the results. As future work we plan to study the influence of different values of ν on the LogLoss.

The LogLoss values for the Taxicab dataset can be seen in Figure 4.10.

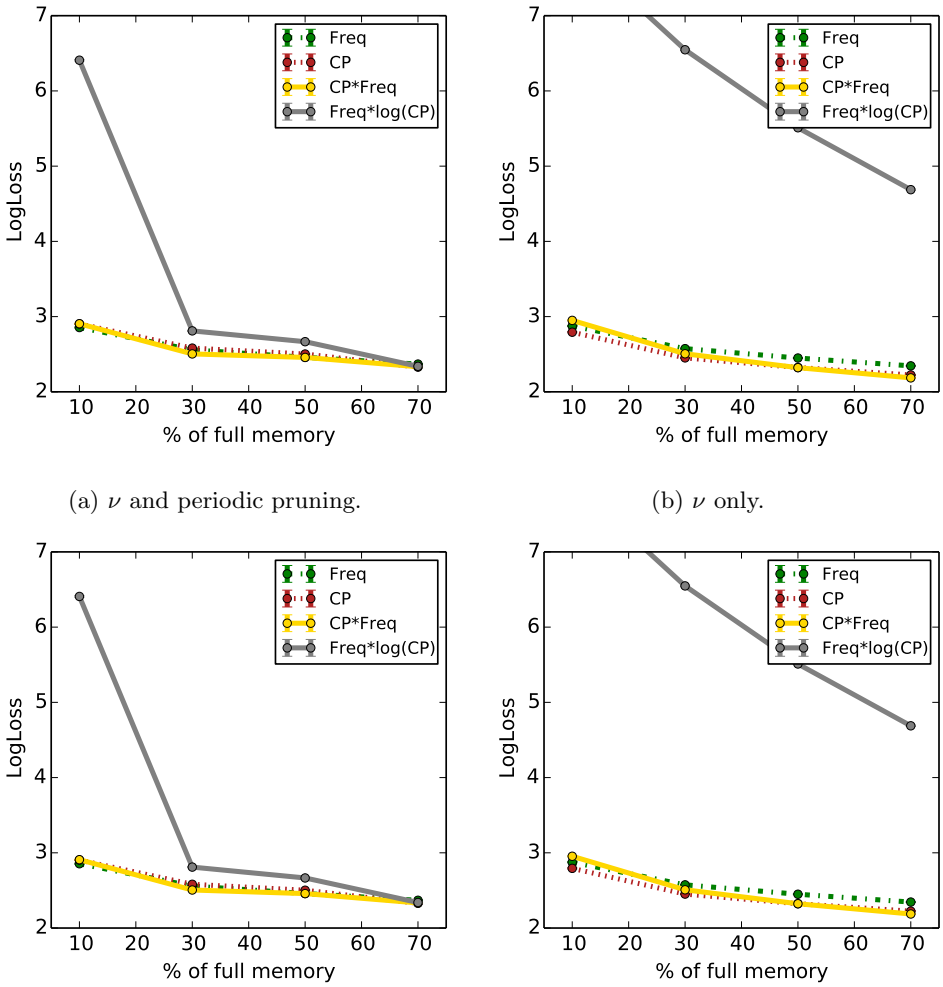
According to the results, pruning based on conditional probability, frequency and their multiplication show the best accuracy for different variations of the VOCHH algorithm, while entropy criteria lead to less accurate models. It can also be seen that the VariableCHH algorithm variations that use deep digging or none of the enhancements show minimum possible LogLoss, which is achieved together with multiplication-based pruning (Figures 4.10b, 4.10d). At the same time, periodic pruning slightly increases the LogLoss of best performing pruning strategies, while the entropy based strategy clearly benefits from it (Figures 4.10a, 4.10c).

Given the good performance of the multiplication-based pruning, in the next section we report the accuracy of VOCHH discovery mainly for this strategy.

4.5.5 Accuracy of VOCHH Estimation

In this section we discuss the experiments of VOCHH mining for three real datasets. In order to validate the results of the VariableCHH and VariableCHHr algorithms, we compare the approximate VOCHH with the exact VOCHH. For the experiments, we used pruning based on the multiplication of the conditional probability and frequency and the entropy-based pruning. Approximate algorithms for VOCHH estimation used a fraction of memory needed for the exact computation, with this fraction marked on the x-axis of the plots. We used 6 as the maximum order for almost all of the experiments. For all the results presented here, we have used deep digging and periodic pruning enhancements.

The accuracy results for the BUWeb dataset, obtained using VariableCHH and VariableCHHr algorithms with multiplication pruning are shown on Figure 4.11.



(a) ν and periodic pruning.

(b) ν only.

(c) Periodic pruning only.

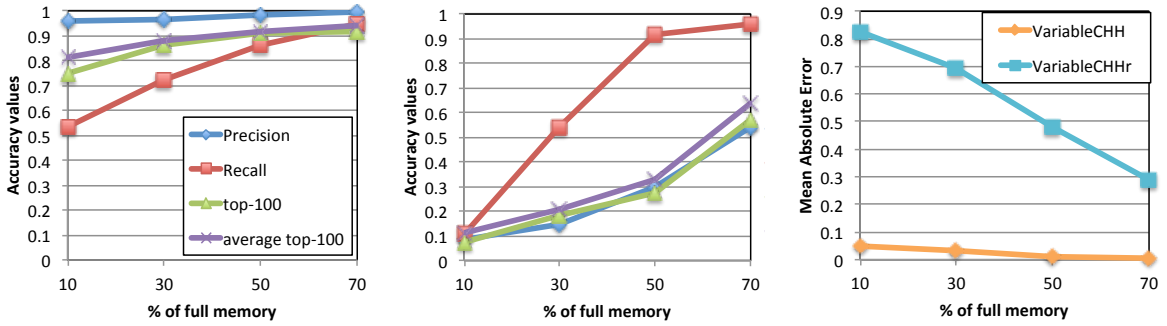
(d) Without any enhancements.

Figure 4.10: Average LogLoss for different algorithm variations for varying budget for the Taxicab data.

As can be seen from the results for the BUWeb dataset, the usage of the reintroduction strategy helps to increase the recall of VOCHH estimation if there is enough memory to additionally keep the overestimated VOCHH candidates (Figure 4.11b). The accuracy of the estimation of conditional probabilities is much better for the algorithms where no reintroduction is used (Figure 4.11c). The experiments with the multiplication pruning and no reintroduction show the best behavior for all the accuracy measures (Figure 4.11a), with precision being close to one even if only 10% of memory is exploited.

The plots for the Taxicab datasets for the models with maximum order of 6 can be seen in Figure 4.12.

Similarly to the BUWeb case, reintroduction leads to lower precision and top-τ precision, though recall is higher for all the memory budgets, except when it equals 10%

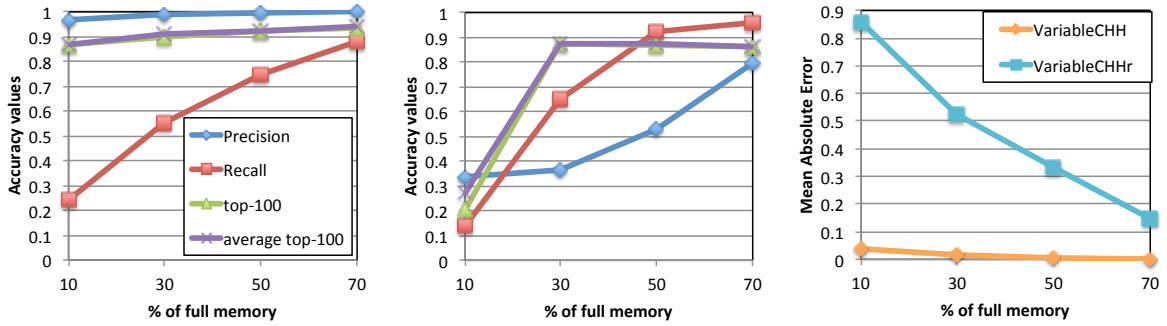


(a) Accuracy results for VariableCHH algorithm (no reintroduction).

(b) Accuracy results for VariableCHHr algorithm (with reintroduction).

(c) Mean Absolute Error of estimation of the VOCHH conditional probabilities.

Figure 4.11: Validation of the approximate VOCHH algorithm for BUWeb data. Multiplication pruning.



(a) Accuracy results for VariableCHH algorithm.

(b) Accuracy results for VariableCHHr.

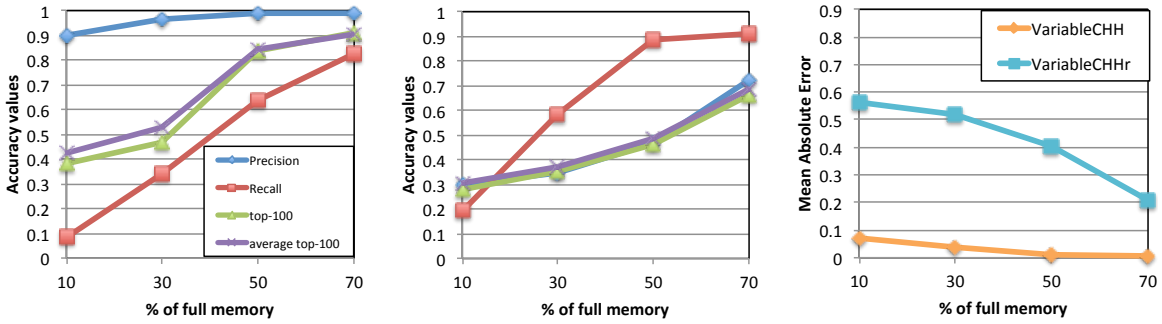
(c) Mean Absolute Error of estimation of the VOCHH conditional probabilities.

Figure 4.12: Validation of VOCHH approximate algorithm for Taxicab data. Multiplication pruning.

(Figure 4.12b). Accuracy of the estimation of conditional probabilities is still better when no reintroduction is used (Figure 4.12c). The performance of the VariableCHH with multiplication criterion and no reintroduction is the best, with precision being close to one even for the 10% of memory usage, and recall being over 50% (Figure 4.12a).

As we discussed before, entropy pruning leads to less accurate results, as logarithm is harder to estimate in case of the limited memory budget. Figure 4.13 illustrates this phenomenon on the Taxicab dataset.

The results with the entropy pruning are worse than the results of the multiplication pruning regardless of the reintroduction. While the precision of the entropy pruning without reintroduction is still relatively high, the other measures are much lower. The recall of the method with the reintroduction is still higher, as expected due to the reintroduction quality guarantees (Figures 4.13a, 4.13b). The only benefit of the entropy pruning is that is keeps fairly accurate conditional probabilities for the reintroduction



(a) Accuracy results for VariableCHH algorithm.

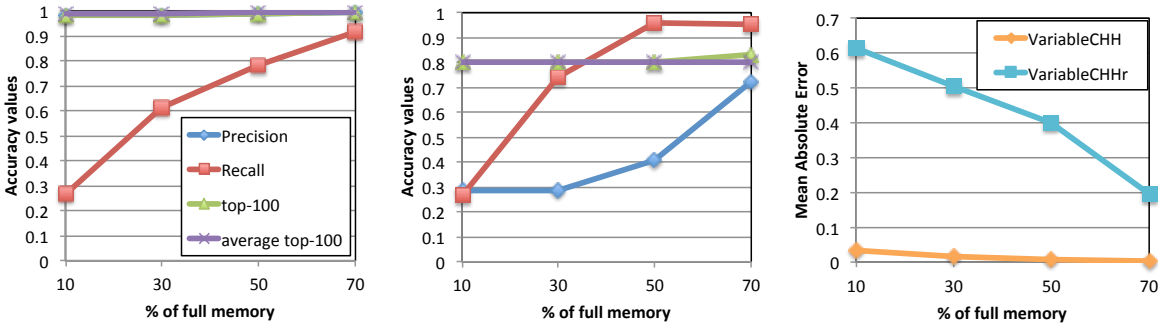
(b) Accuracy results for VariableCHHr algorithm.

(c) Mean Absolute Error of estimation of the VOCHH conditional probabilities.

Figure 4.13: Validation of VOCHH approximate algorithm for TaxiCab data. Entropy pruning.

(Figure 4.13c). Similar behavior of the entropy pruning extends to the BUWeb and the WorldCup datasets, where multiplication pruning also performs the best.

The Results for WorldCup data for the models with maximum order equal to 5 are shown on Figure 4.14.



(a) Accuracy results for VariableCHH algorithm.

(b) Accuracy results for VariableCHHr algorithm.

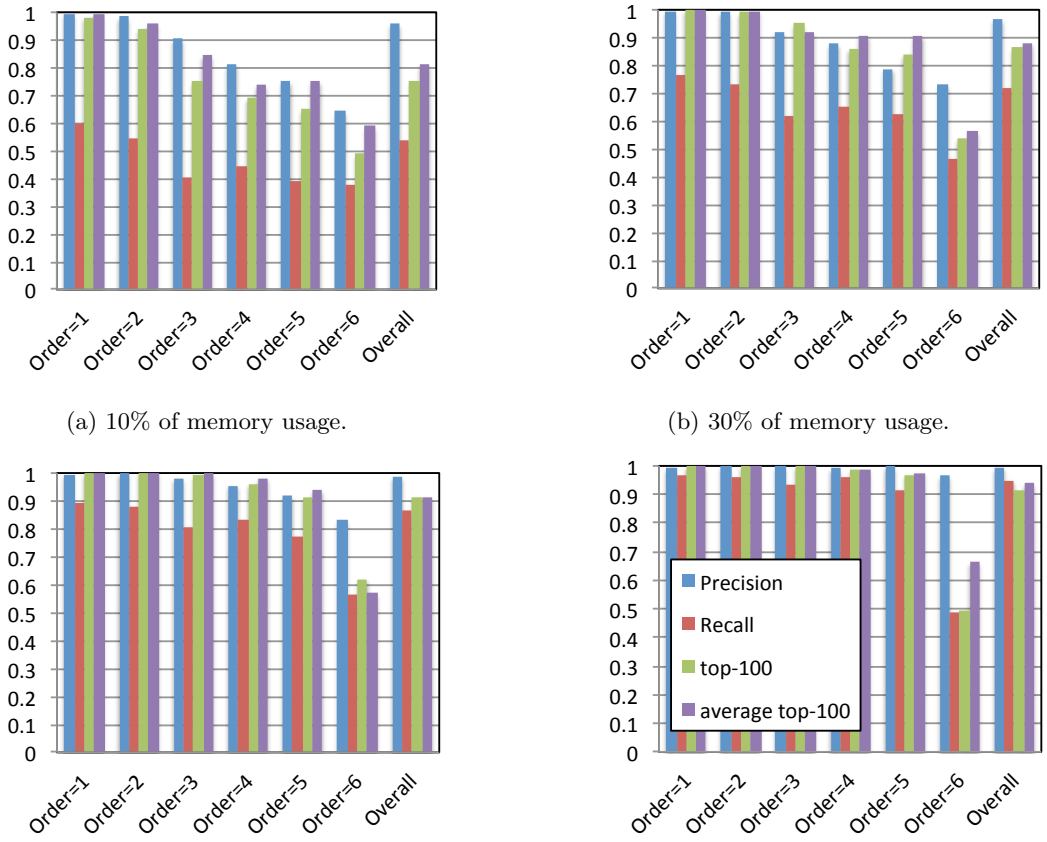
(c) Mean Absolute Error of estimation of the VOCHH conditional probabilities.

Figure 4.14: Validation of VOCHH approximate algorithm for the WorldCup dataset. Multiplication criterion. Maximum order 5.

Precision and top-τ precision of the results is almost perfect for the Taxicab dataset when the reintroduction is not used (Figure 4.14a). Similarly to other methods, recall is higher for the method with reintroduction, as we can see in Figure 4.14b. The accuracy of the estimation of conditional probabilities is higher for the method without reintroduction (Figure 4.14c), which is also the case for the rest of the methods and for the entropy criterion as well.

We would also like to consider precision, recall, and top-τ measures to see how well we are able to extract VOCHH of each separate order. Estimation accuracies of VOCHH for

BUWeb dataset for all the orders separately for 10, 30, 50 and 70 % memory usage are shown in Figure 4.15. The plots correspond to the methods with multiplication pruning without reintroduction.



(a) 10% of memory usage.

(b) 30% of memory usage.

(c) 50% of memory usage.

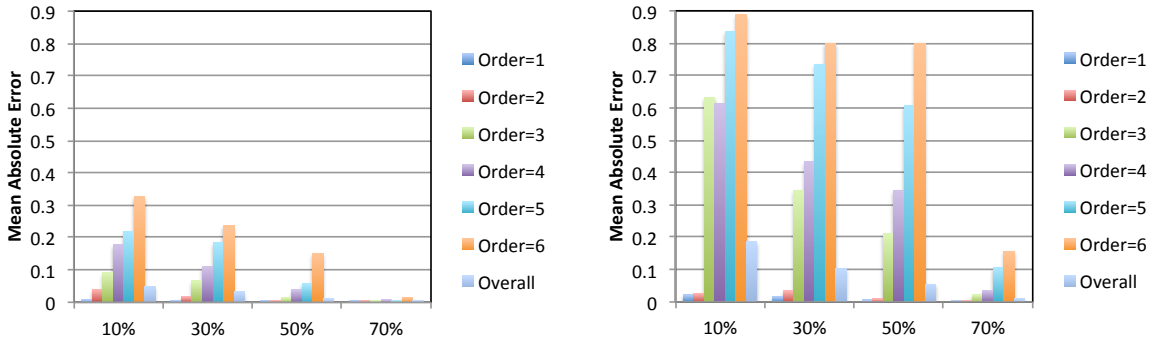
(d) 70% of memory usage.

Figure 4.15: Accuracies of VOCHH estimation for the BUWeb datasets for each order separately.

As we can see from the plots, VOCHH of higher orders are harder to extract, and the accuracy grows when more memory is used. At the same time, precision and top-τ measures are fairly high for all the orders even when only 10% of memory is used, while recall values are satisfactory. This means that multiplication pruning exploits the space quite efficiently.

In order to see the accuracy of estimation of the conditional probabilities of VOCHH for each order separately, we show the Mean Absolute Error of the estimation in Figure 4.16. The results are obtained with multiplication (Figure 4.16a) and entropy pruning (Figure 4.16b), without reintroduction.

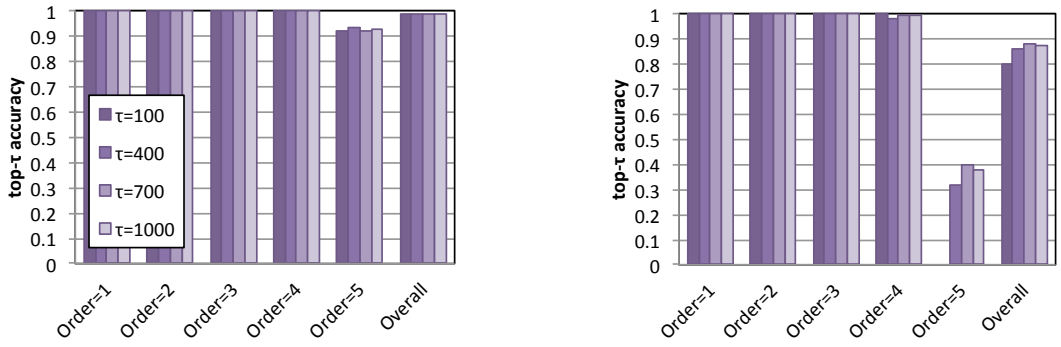
The results show that both pruning strategies estimate the conditional probabilities of the higher orders less accurately, although multiplication pruning provides much better results (Figure 4.16a). For example, the accuracy of the estimation of the conditional



(a) Multiplication pruning.

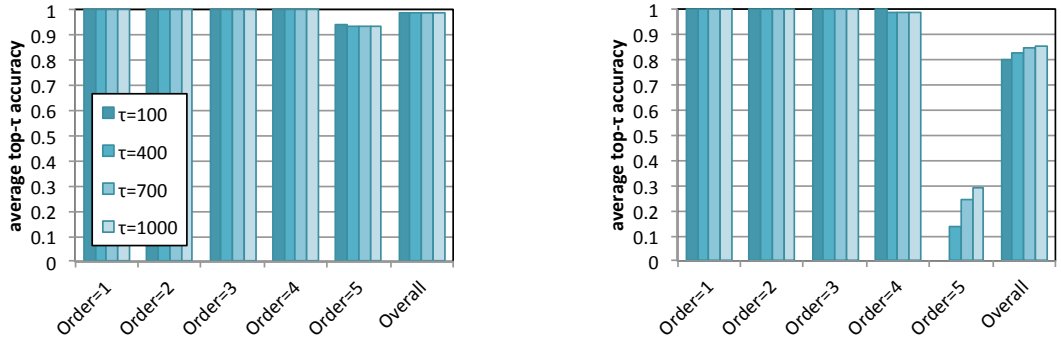
(b) Entropy pruning.

Figure 4.16: The Mean Absolute Error of the conditional probabilities of VOCHH for BUWeb dataset.



(a) Top-τ for 10% memory usage, no reintroduction.

(b) Top-τ for 30% memory usage, with reintroduction.



(c) Average top-τ for 10% memory usage, no reintroduction..

(d) Average top-τ for 30% memory usage, with reintroduction.

Figure 4.17: Top-τ accuracies of VOCHH estimation for WorldCup dataset for varying values of τ.

probabilities of the sixth order is three times higher with multiplication pruning than with the entropy pruning when only 10% of the full memory is used.

For the WorldCup dataset we also considered accuracy results for different τ values.

The results can be seen on Figure 4.17.

As we can see from the results, no reintroduction leads to better top-τ accuracies, even if less memory is used, though it leads to higher recall values. Apparently, when reintroduction is used, longer sequences struggle to beat the overestimated shorter sequences, which appear much more frequently (Figures 4.17b, 4.17d). When no reintroduction is used, we have almost perfect top-τ and average top-τ results for all the orders and for different values of τ (Figures 4.17a, 4.17c).

4.6 Summary

VOCHH are useful for the modeling of various real-life processes, such as moving trajectories, networking data, and others. For several data sources, VMM describes the generative model of the data better than the fixed order models.

In this chapter, we describe VariableCHH and VariableCHHr algorithms for real-time estimation of VOCHH for the data sources with large finite alphabets. We provide the quality guarantees for VariableCHHr algorithm. The algorithms use several pruning strategies based on the minimization of LogLoss. Based on the experimental evaluation on three real datasets, we demonstrate that VariableCHH algorithm that uses multiplication pruning has high precision and top-τ precision of VOCHH estimation with fairly high recall. VariableCHHr algorithm leads to higher recall, though the accuracy of the estimated conditional probabilities is lower due to the overestimation. We also demonstrate that VariableCHH with multiplication pruning is efficient in VMM modeling in the settings of limited memory.

As future work, we would like to adapt the optimal pruning for the LogLoss minimization to the streaming setting, and conduct a more thorough experimental evaluation of the deep digging and periodic pruning enhancements.

Chapter 5

Sigmoid Rule Framework: Classifier Behavior in the Presence of Noise

In this Chapter we analyze the behavior of the machine learning classifiers in the presence of label noise. We propose the Sigmoid Rule Framework, which defines formal criteria for selecting the best learning algorithm depending on the characteristics of data. The related work for this problem has been surveyed in Section 2.4.

In Section 5.1 we study the sigmoid function, and define the dimensions of the Sigmoid Rule Framework. This set of dimensions provides both quantitative and qualitative support for selecting the learner in different settings. We test our framework on multiple datasets (Section 5.2), by statistically analyzing the connection between the properties of the dataset and the SRF dimensions (Section 5.2.4) for both sequential and non-sequential classification tasks. Section 5.3 contains our concluding remarks.

The work presented in this chapter is published in the conference paper [92] and is submitted as a journal publication [91].

5.1 The Sigmoid Rule Framework

In order to describe the performance of a classifier, the "sigmoid rule", introduced in the work [53], uses a function that relates signal-to-noise ratio of the training set to the expected performance. This function is called the *characteristic transfer function* (CTF) of the learning algorithm. In this work we also refer to it as the *sigmoid function* of the algorithm, and use the terms CTF and sigmoid function interchangeably. The function is of the form:

$$f(Z) = m + (M - m)\frac{1}{1 + b \cdot \exp(-c(Z - d))},$$

where $m \leq M$, $b, c > 0$ are the parameters of the sigmoid function, $Z = log(1 + S) - log(1 + N)$ is the signal-to-noise ratio; S is the amount of "signal" (true data), while N

is the amount of "noisy", distorted data.

The behavior of different machine learning algorithms in the presence of noise can be compared along several *axes of comparison*, based on the parameters of the sigmoid function. Parameters related to *performance* include:

(a) the minimal performance m,

(b) the maximal performance M,

(c) the width of the performance range $r_{alg} = M - m$.

With regard to the *sensitivity* of performance to the change in the signal-to-noise ratio, we consider:

(a) within which range of noise levels, change in the noise leads to significant change in performance;

(b) how we can tell apart algorithms that improve their performance even when the signal-to-noise levels are low, from those that only improve in high ranges of the signal-to-noise ratio;

(c) how we can measure the stability of performance of an algorithm against varying noise;

(d) at what noise level an algorithm reaches its average performance;

(e) whether reducing the noise in a dataset is expected to have a significant impact on the performance.

To answer these questions we perform an analytic study of the sigmoid function of a particular algorithm. This analysis helps to devise measurable dimensions that can answer our questions.

We investigate the properties of the sigmoid in order to determine how each of the parameters m, M, b, c, and d affects the shape of the sigmoid, and how this translates to the expected performance of the learning algorithm. We start by a direct analysis of the sigmoid function and its parameters.

The domain of the sigmoid is, in the general case, $Z \in (-\infty, +\infty)$. The range of values is (m, M). The first order derivative is

$$f'(Z) = \frac{bc \cdot \exp(-c(Z-d)(M-m))}{(1 + b \cdot \exp(-c(Z-d)))^2}.$$

As $f'(Z) > 0$ for $\forall Z \in (-\infty, +\infty)$, the function $f(Z)$ is monotonically increasing. The second order derivative is

$$f''(Z) = \frac{(M-m)bc^2 \cdot \exp(-c(x-d))}{(1 + b \cdot \exp(-c(Z-d)))^2} \times$$

$$\left(\frac{2b \cdot \exp(-c(Z-d))^2}{1+b \cdot \exp(-c(Z-d))} - 1\right).$$

Thus, $f''(Z) = 0$ if $Z = d + \frac{1}{c}\log b$, and the point $Z_{inf} = d + \frac{1}{c}\log b$ is *the point of inflection*, which shows when the curvature of the function changes its sign. In the case of sigmoid, the point of inflection is the point of symmetry. Furthermore, the point of inflection, Z_{inf} (the middle point in Figure 5.2) indicates the shift of the sigmoid with respect to the origin, which is equal to $d + \frac{1}{c}\log b$. So, the shift of the sigmoid depends on three parameters: b, c, and d.

The slope of the sigmoid reflects the improvement of an expected performance of an algorithm per change in the signal-to-noise ratio. To estimate the slope, we use the distance d_s between the point of inflection Z_{inf} (zero of the second derivative) and the zeros of the third derivative (both zeros of the third derivative are at the same distance from the point of inflection). The zeros of the third order derivative are $Z^{(3)}_{1,2} = d - \frac{1}{c}\log\frac{2 \pm \sqrt{3}}{b}$. In this case d_s looks like:

$$d_s = Z^{(3)}_1 - Z_{inf} = Z_{inf} - Z^{(3)}_2 = \frac{1}{c}\log\frac{1}{2-\sqrt{3}}.$$

The larger this distance is, the more expanded the sigmoid curve is. Since $\log\frac{1}{2-\sqrt{3}} \approx 1.32$, d_s is in inverse proportion to parameter c: $d_s = \frac{a}{c}$, where $a \approx 1.32$. We find that the parameter c directly influences the slope of the sigmoid curve. We term c as the *slope indicator* of the CTF. Figure 5.1 depicts sigmoids with different c values, illustrating gradual and sharp slopes of the sigmoid function.

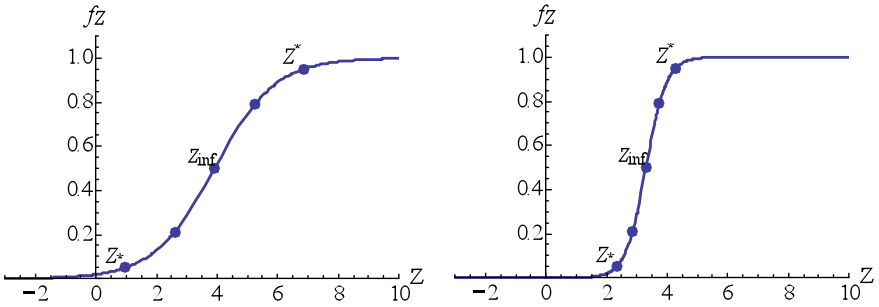


Figure 5.1: Varying c parameter. $c = 1$ (left), $c = 3$ (right). Other parameters are the same: $m = 0$, $M = 1$, $b = 2.5$, $d = 3$.

Figure 5.2 shows the sigmoid curve along with its points of interest.

In the following section, we formulate and discuss dimensions that describe the behavior of algorithms, based on the axes of comparison discussed above.

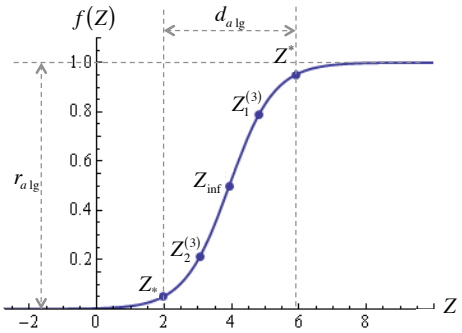


Figure 5.2: Sigmoid function and points of interest.

5.1.1 SRF Dimensions

In this section we determine several SRF dimensions based on the sigmoid properties, in addition to m, M, r_{alg}, and c, discussed in Section 5.1. We define *active noise range* as the range $[Z_*, Z^*]$ in which the change in noise induces a measurable change in the performance. In order to calculate $[Z_*, Z^*]$ we assume that there is a good-enough performance on a given task, approaching M for a given algorithm. We know that $f(Z) \in (m, M)$, and we say that the performance is "good enough" if $f(Z) = M - (M - m) * p, p = 0.05$[1]. We define the *learning improvement* of the algorithm as the size of the signal-to-noise interval in which $f(Z) \in [m + (M - m) * p, M - (M - m) * p]$. Then, using the inverse

$$f^{-1}(y) = d - \frac{1}{c} \log\left(\frac{1}{b}\left(\frac{M-m}{y-m} - 1\right)\right),$$

we calculate the points Z^* and Z_*, which respectively are the bottom and the top points in Figure 5.2 for a given p. We term the distance $d_{alg} = Z^* - Z_*$ as the *width of the active area* of the machine learning classifier (see Figure 5.2). Then we define a ratio between the width of the active area and the width of the performance range $\frac{r_{alg}}{d_{alg}}$, which describes (and is termed as) the *learning performance improvement over signal-to-noise ratio change*.

In the following paragraphs we describe how the analysis of the CTF allows us to compare the performance of learning algorithms in the presence of noise.

5.1.2 Comparing Algorithms

Given the performance dimensions described above, we can compare algorithms as follows. For *performance*-related comparison we can use minimal performance m, maximal performance M, and the width of performance range r_{alg}. Algorithms not affected by the

[1]Instead of 0.05, one can use any value close to 0, describing a normalized measure of distance from the optimal performance. In the case of $p = 0.05$, the distance from the optimal performance is 5%.

presence or absence of noise will have a minimal r_{alg} value. In a setting with a randomly changing level of noise, this parameter is related to the possible variance in performance. Related to the *sensitivity* of performance to the change of the signal-to-noise ratio, we can use the following dimensions.

1. The *active noise range* $[Z_*, Z^*]$, which shows how early the algorithm starts operating (measured by Z_*) and how early it reaches good-enough performance (measured by Z^*). We say that the algorithm *operates* when the level of noise in the data is within the active noise range of the algorithm.

2. The width of the active area $d_{alg} = Z^* - Z_*$ of the algorithm, which is related to the speed of changing performance for a given r_{alg} in the domain of noise. A high d_{alg} value indicates that the algorithm varies its performance in a broad range of signal-to-noise ratios, implying less performance stability in an environment with heavily varying degrees of noise.

3. The parameter c of the sigmoid function (*slope indicator*), which is related to the distance $d_s = 1.32/c$. The distance has similar properties as the inversion of r_{alg}/d_{alg}, but it is independent of the predefined parameter p, which shows the percentage of performance considered as good-enough for a given task and is not directly connected to the active noise range. d_s reflects the change in the slope of the function. If c is large, then d_s is small and indicates higher stability of the algorithm in the presence of noise, and vice versa.

4. The point of inflection Z_{inf}, which shows the signal-to-noise ratio for which an algorithm gives the average performance. The parameter can be used to choose the algorithm that reaches its average performance earlier in a noisy environment, or the algorithm whose speed of improvement changes earlier.

A parameter related to both *performance* and *sensibility* is the learning performance improvement over signal-to-noise ratio change, r_{alg}/d_{alg}. It can be used to determine whether reducing the noise in a dataset is expected to have a significant impact on the performance. An algorithm with a high value of r_{alg}/d_{alg} would imply that it makes sense to put more effort into reducing noise. Furthermore, using this dimension a decision maker can choose more stable algorithm, when the variance of noise is known. In this case, the algorithm with the lowest value of r_{alg}/d_{alg} should be chosen in order to limit the corresponding variance in performance.

Based on the above discussion, we favor the classifiers with the following properties:

- higher maximal performance M,
- larger width of performance range r_{alg},
- higher learning performance improvement over signal-to-noise ratio change r_{alg}/d_{alg},
- shorter width of the active area of the algorithm d_{alg}, and

- larger slope indicator c.

We expect to get high performance from an algorithm if the level of noise in the dataset is very low, and low performance if the level of noise in the dataset is very high. Decision makers can easily formulate different criteria, based on the proposed dimensions.

5.2 Experimental Evaluation

In the following paragraphs we describe the experimental setup, the datasets used, and the results of our experiments. We first consider non-sequential and then sequential classifiers.

5.2.1 Experimental setup for non-sequential classifiers

In our study we applied the following machine learning algorithms, implemented in Weka 3.6.3 [60]:
(a) IBk – K-nearest neighbor classifier;
(b) Naïve Bayes classifier;
(c) SMO – support vector classifier (cf. [71]);
(d) NbTree – a decision tree with Naïve Bayes classifiers at the leaves;
(e) JRip – a RIPPER [29] rule learner implementation.

We have chosen representative algorithms from different families of classification approaches, covering very popular classification schemes. The experiments were performed on a 2.4GHz machine with 4GB RAM.

We used a total of 24 datasets for our experiments. Most of the real datasets come from the UCI machine learning repository [50], and one from the study [53]. Fourteen of the dataset are real, while ten are synthetic. All the datasets are divided into groups according to the number of classes, attributes (features) and instances in the dataset, as shown in Figure 5.3.

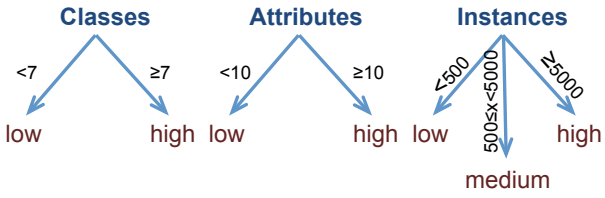


Figure 5.3: Dataset grouping labels.

There are 12 possible groups that include all combinations of the parameters. Two datasets from each group were employed for the experiments. We created artificial datasets in the cases were real datasets with a certain set of characteristics were not available. The distribution of the dataset characteristics is illustrated in Figure 5.4. The

traits of the datasets illustrated are the number of classes, the number of attributes, the number of instances, and the estimated intrinsic (fractal) dimension.

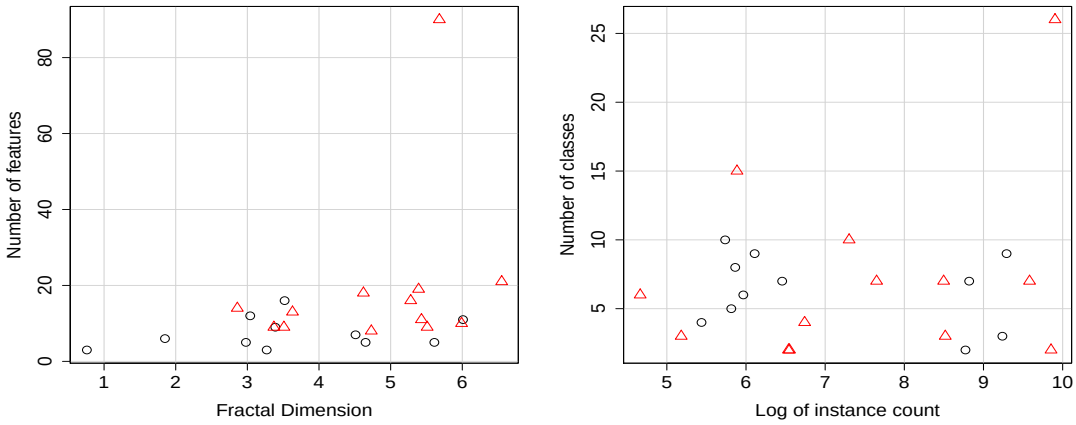


Figure 5.4: Distribution of the dataset characteristics. Real data: triangles, Artificial: circles.

Table 5.1 contains the description of the datasets and their sources.

We build ten artificial datasets using the following procedure. Having randomly sampled the number of classes, features and instances, we sample the parameters of each feature distribution. We assume that the features follow the Gaussian distribution with mean value (μ) in the interval $[-100, 100]$ and standard deviation(σ) in the interval $[0.1, 30]$. The μ and σ intervals allow overlapping features across classes. The description of the parameters of the generated datasets is shown in Table 5.1, where artificial datasets are prefixed with the letter "a".

The noise was induced as follows. We created the stratified training sets, equal in size to the stratified test sets. To induce noise, we created noisy versions of the training sets by mislabeling the instances. Using different levels l_n of noise, $l_n = 0, 0.05, ..., 0.95$[2], a training set with l_n noise is a set with fraction l_n of mislabeled instances. Hence we

[3]See http://archive.ics.uci.edu/ml/datasets/Wine
[4]See http://archive.ics.uci.edu/ml/datasets/Statlog%28Australian+Credit+Approval%29
[5]See http://archive.ics.uci.edu/ml/datasets/Yeast
[6]See http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Prognostic%29
[7]See http://archive.ics.uci.edu/ml/datasets/Breast+Tissue
[8]See http://archive.ics.uci.edu/ml/datasets/Letter+Recognition
[9]See http://archive.ics.uci.edu/ml/datasets/Statlog+%28Vehicle+Silhouettes %29
[10]See http://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope
[11]See http://archive.ics.uci.edu/ml/machine-learning-databases/waveform/
[12]See http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/ shuttle/
[13]See http://archive.ics.uci.edu/ml/datasets/Libras+Movement
[14]See http://archive.ics.uci.edu/ml/machine-learning-databases/image/
[15]See http://archive.ics.uci.edu/ml/datasets/Wine+Quality
[16]from [53]
[2]We note that high levels of noise such as 95% are often observed in the presence of *concept drift*, e.g., when learning computer-user browsing habits in a network environment with a single IP, and several different users sharing it.

Table 5.1: The description of the datasets used. x_1 is a number of classes, x_2 is the number of attributes (features), x_3 is the number of instances, and x_4 is the intrinsic dimensionality of datasets. The sources of the datasets are given in the last column.

Dataset	**Parameters**				**Reference**
	# of classes	# of attr	# of inst	fract dim	
	x_1	x_2	x_3	x_4	
r1	3	13	178	3.63	real[3]
r2	2	14	690	2.86	real[4]
r3	4	6	230	1.85	real from[16]
r4	10	8	1484	4.73	real[5]
a5	5	12	335	3.04	artificial
a6	9	16	450	3.52	artificial
a7	10	5	310	2.98	artificial
a8	7	9	637	3.39	artificial
a9	8	3	352	0.76	artificial
r10	2	9	699	3.51	real[6]
r11	6	9	106	3.37	real[7]
r12	26	16	20000	5.28	real[8]
a13	6	5	390	4.65	artificial
a14	3	7	10287	4.51	artificial
a15	2	3	6452	3.27	artificial
r16	4	18	846	4.62	real[9]
r17	2	10	19020	5.99	real[10]
r18	3	21	5000	6.55	real[11]
r19	7	9	14500	5.51	real[12]
a20	9	5	10845	5.61	artificial
r21	15	90	360	5.68	real[13]
r22	7	19	2100	5.39	real[14]
r23	7	11	4898	5.43	real[15]
a24	7	11	6762	6.01	artificial

obtained 20 versions of the datasets with varying noise levels.

The equal percentages of instances from different classes (for stratification) were put into test and training sets (as in the original dataset with 0% of noise), using the following procedure:

1. calculate the number N of observations in the original dataset; the number of observations in the test or training set should be the integer part of $N/2$;

2. calculate the percentages of each class in the original dataset;

3. calculate the number of observations from each class that should be in the training and test sets based on the percentages from the previous step (rounding the numbers

if necessary);

4. divide the observations according to the classes to obtain "class datasets", i.e., sets of instances with only one class per set;

5. from each "class dataset", sample observations (based on the results of step 3) for the training set, and place the remaining observations of each "class dataset" into the test set;

6. make "noisy" training datasets from the original training dataset with different levels l_n of noise.

7. get the performance of the model trained on each "noisy" dataset $l_n = 0, 0.05, ..., 0.95$ and tested on a noise-free stratified test set.

Given the performance of a classifier for a particular dataset, we estimate the parameters of the sigmoid function for each algorithm-dataset pair using the genetic algorithm described in Appendix B. Having the sets of estimated sigmoids describing the performance for particular algorithms and datasets with certain characteristics, we reason about the properties of the algorithms along the SRF dimensions. The applicability of SRF for non-sequential classifiers is shown our paper [92].

5.2.2 Experimental setup for sequential classifiers

We now consider sequential datasets, where the order of instances is important. Using these datasets, we study whether sequential classifiers adhere to the sigmoid rule framework. The main difference to the non-sequential case is that for the classification task we also use the information about the previous instances in addition to the features of the current instance. For sequential data, we apply three classification algorithms based on Hidden Markov Models (HMM) [99], Conditional Random Fields (CRF) [76], and Recurrent Neural Networks (RNN) [46].

For the experiments, we consider the whole sequence of observations as an instance. For each algorithm, we define the order of dependency, and consider sequences of the length corresponding to that order. For HMM-based classification, we perform the experiments in two settings by using HMM of the third and the forth order. For CRF-based classification, we consider linear and quadratic CRFs. For RNN-based classification, we also use two settings with the second and the third order dependencies. All datasets used in the experiments contain time series where the instances appear in chronological order. The label of the last observation is assigned to the sequence to imply causality.

For the implementation of these sequential classifiers, we use the jahmm [68], Mallet [86], and Weka [60] libraries for HMM, CRF, and RNN respectively. The experiments and the libraries are implemented in Java, and the classification algorithms are called

as library functions. The experiments were performed on a 2.67GHz machine with 4GB RAM. We used 18 datasets for our sequential experiments. All of these datasets are real datasets that mostly come from the UCI machine learning repository [9].

The distribution of the characteristics of these datasets is illustrated in Figure 5.5. These characteristics are: the number of classes, the number of attributes, the number of instances, and the largest lag value that corresponds to significant autocorrelation of the labels. Autocorrelation is a linear dependence of a variable with itself at two points in time [16]. We use the autocorrelation function (ACF) implemented in R[3]to calculate autocorrelation with maximum lag equal to 50 for all datasets. We choose the first lag that gives the smallest autocorrelation falling out of the significance band, as shown, for example, in Figure 5.6 for the Pioneer gripper dataset, where we chose lag equal to 37.

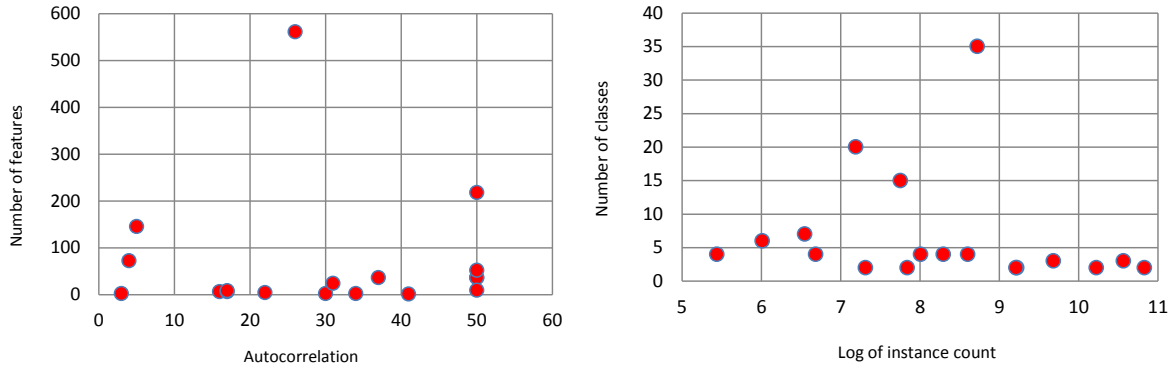


Figure 5.5: Distributions of the characteristics of sequential datasets.

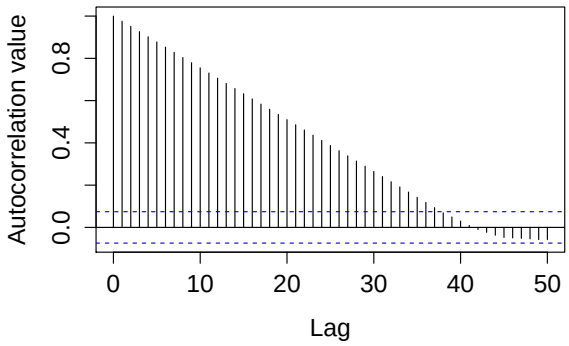


Figure 5.6: Autocorrelation plot of the Pioneer gripper dataset.

Description of the datasets are shown in Table 5.2.

[3]http://stat.ethz.ch/R-manual/R-patched/library/nlme/html/ACF.html

Table 5.2: Dataset descriptions. x_1 is the number of classes, x_2 is the number of instances, x_3 is the autocorrelation lag, and x_4 is the number of attributes (features). The sources of the datasets are given in the last column.

Dataset	**Parameters**				**Reference**
	# of classes	# of inst	autocorrelation lag	# of attr	
	x_1	x_2	x_3	x_4	
r1	4	230	16	6	Climate [16]
r2	6	408	26	561	Human Activity [17]
r3	7	697	37	36	Pioneer gripper [18]
r4	4	800	22	4	Robot walk 4 [19]
r5	20	1327	3	2	Diabetes [20]
r6	2	1500	4	72	Ozone Level 1 hour [21]
r7	15	2324	50	36	Pioneer turn [18]
r8	2	2534	5	145	Ozone Level 8 hour [21]
r9	4	3000	50	218	Opportunity [22]
r10	4	4000	30	2	Robot walk 2 [19]
r11	4	5434	31	24	Robot walk 24 [19]
r12	35	6129	50	36	Pioneer move [18]
r13	2	10082	41	1	Callt2 [23]
r14	2	10000	17	6	Electric 2 [16]
r15	3	16000	50	52	PAMAP [24]
r16	2	27552	17	8	Electric 3 [16]
r17	3	38774	50	9	Daphnet Freezing [25]
r18	2	50400	34	2	Dodger [26]

We induce noise into sequential data in a way similar to the independent data. We randomly split all data into equally sized training and test parts. Mislabeled instances are in the training data only. We induce varying levels of noise from 0% to 100% as follows. The noisy version of a class C_i of a training set is created by selecting the level of noise l_n, and replacing some instance with an instance of another class with probability l_n. Then sequential classifiers are trained using the training data with various levels of noise, and are used to predict the test data. Given the classification of all the test instances, we

[17] See http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones
[18] See http://archive.ics.uci.edu/ml/datasets/Pioneer-1+Mobile+Robot+Data
[19] See http://archive.ics.uci.edu/ml/datasets/Wall-Following+Robot+Navigation+Data
[20] See http://archive.ics.uci.edu/ml/datasets/Diabetes
[21] See http://archive.ics.uci.edu/ml/datasets/Ozone+Level+Detection
[22] See http://archive.ics.uci.edu/ml/datasets/OPPORTUNITY+Activity+Recognition
[23] See http://archive.ics.uci.edu/ml/datasets/CalIt2+Building+People+Counts
[24] See http://archive.ics.uci.edu/ml/datasets/PAMAP2+Physical+Activity+Monitoring
[25] See http://archive.ics.uci.edu/ml/datasets/Daphnet+Freezing+of+Gait
[26] See http://archive.ics.uci.edu/ml/datasets/Dodgers+Loop+Sensor
[16] from [53]

compute the overall performance of the algorithm using classification accuracy:

$$P = \frac{\#\text{CorrectClassifications}}{\#\text{TotalClassifications}}$$

We perform repeated random sub-sampling validation with 20 splits by randomly choosing half of the instances as training data and the remaining instances as test data, and calculate the average performance of the classifier. For the sequential datasets the correct order of the instances is kept for both training and test phases. The parameters of the corresponding sigmoid are assessed using the values of signal-to-noise ratio and corresponding average performance levels.

In the following sections we consider sequential classifiers in more detail.

Classifier Based On Hidden Markov Models

The first sequential classifier we use is based on Hidden Markov Models (HMMs). HMM describes the generative model of a time series.

More formally, HMM is a double stochastic process, with an underlying stochastic process that is not observed, and a different set of stochastic processes that produce the sequence of observed symbols [99]. An HMM models a sequence of observations $Y = \{y_t\}_{t=1}^{T}$ from a finite state set C, assuming that there is an underlying sequence of hidden states $X = \{X_t\}_{t=1}^{T}$ drawn from a finite set S, as shown in Figure 5.7.

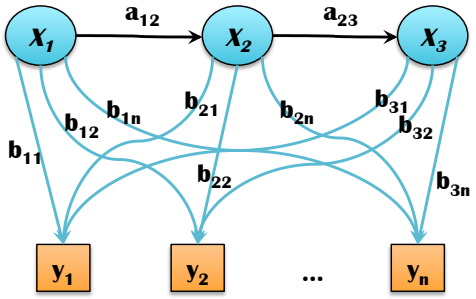


Figure 5.7: Hidden Markov Model. X define the hidden states, while y are possible observations, a – state transition probabilities, b – output probabilities.

An HMM is described by a state transition probability distribution $A = \{a_{ij}\}$, $a_{ij} = \Pr(X_{t+1} = s_j | X_t = s_i)$, an emission probability distribution in a state $B = \{b_j(k)\}, b_j(k) = \Pr(y_t = c_k | X_t = s_j)$, and an initial state distribution $\Pi = \{\pi_i\}, \pi_i = \Pr(X_t = s_j)$. The model parameters A, B, Π can be labeled as a triple $\lambda = (\pi, A, B)$. Given a sequence of observations Y and a state sequence X, the parameters λ are chosen to maximize $\Pr(Y|\lambda)$.

We use the implementation of the HMM sequential classifier provided by the jahmm library [68]. This library contains an implementation of the Segmental K-Means algorithm for estimating the parameters of HMM [69]. The algorithm uses the state-optimized

joint likelihood of the observed data and the underlying Markovian state sequence as the objective function:

$$\max f(x, y|\lambda) = \max \pi_{s_0} \prod_{t=1}^{T} a_{s_{t-1}s_t} b_{s_t}(x_t).$$

Compared to the previous state of the art method used to estimate the parameters of HMM, based the forward-backward algorithm (often called the Baum-Welch algorithm [11]), the Segmental K-Means algorithm requires less computations, and has been successfully used for both modeling and decoding task in speech recognition [69].

For the inference task – finding the most likely state sequence that matches a given observation sequence – we use the Viterbi algorithm [115] implemented in jahmm. In our case, the observation sequence is the sequence of class labels in a sequential dataset. For the experimental evaluation of the HMM-based classifier, we use only the labels of the datasets, since the attributes cannot be utilized due to the limitations of the model.

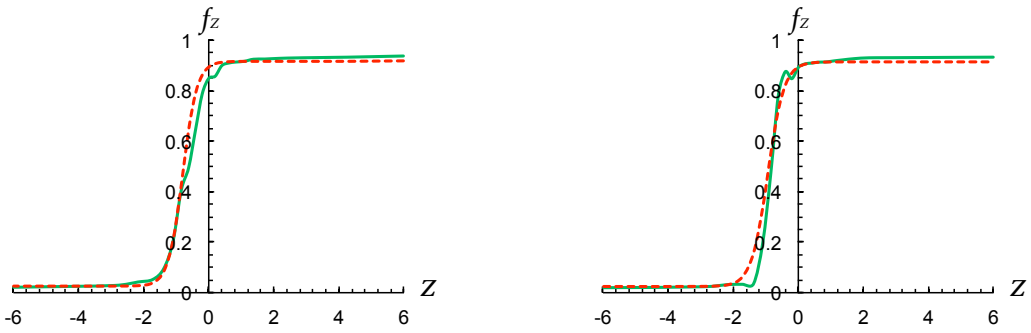
We study the HMM based classifier in two settings, according to the order of the HMM. In the first setting, we consider an HMM of the third order, in which the current label depends on the three previous labels. The second setting corresponds to an HMM of the fourth order, in which a label depends on the four previous labels. Figure 5.7 illustrates the structure of a first-order HMM.

In Figure 5.8, we demonstrate the sigmoids of the HMM algorithm for the "Robot walk 4" dataset. The green solid line corresponds to the true measurements of the performance of the HMM-based classifier for different values of signal-to-noise ratio Z, while the dashed red line corresponds to the estimated sigmoid, the parameters being assessed with a generic algorithm (Appendix B).

The Pearson's correlation test between the obtained performance and the estimated performance based on the sigmoid function shows statistically significant correlation values, with $coef > 0.99$ for significance level $\alpha = 0.01$. This holds for both the first and the second setting (the third and the fourth order models, correspondingly). Similar results were obtained for the rest of the datasets and sequential learners. These results confirm that the sequential learners adhere to the "Sigmoid Rule" framework.

Classifier Based On Conditional Random Fields

Classification algorithms based on HMMs use only the class labels of the data. However, we would like to consider a sequential classifier that takes into account the data attributes as well. To this purpose, we employ a classifier based on Conditional Random Fields (CRF). Given a sequence of observations, a conditional random field calculates the probabilities of possible label of an instance based not only on the labels of the previous instances, but also on the arbitrary, non-independent features of the observation sequence.



(a) Sigmoid CTF of HMM 1st setting.

(b) Sigmoid CTF of HMM 2nd setting.

Figure 5.8: Sigmoid CTF of the two settings of the HMM-based classification algorithm for the "Robot walk 4" dataset. Green solid line: true measurements, dashed red line: estimated sigmoid.

Moreover, the probability of transition between labels is specified based on past and future observations besides the current observation, whenever available [76].

Let Y, X be random vectors, $\Lambda = \lambda_k \in R^K$ be a parameter vector, and $f_k(y, y', x_t)_{k=1}^K$ be a set of real-valued feature functions. A *linear-chain conditional random field* is defined by the distribution $p(y|x)$ that takes the form

$$p(y|x) = \frac{1}{Z(x)} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right\},$$

where $Z(x)$ is an instance-specific normalization function

$$Z(x) = \sum_{y} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right\}.$$

In order to estimate the parameter $\theta = \lambda_k$ of a linear-chain CRF, we consider the training data $D = \{x^{(i)}, y^{(i)}\}_{i=1}^N$, where each $x^{(i)} = \{x_1^{(i)}, x_2^{(i)}, ...x_T^{(i)}\}$ is a sequence of inputs, and each $y^{(i)} = \{y_1^{(i)}, y_2^{(i)}, ...y_T^{(i)}\}$ is a sequence of the desired predictions, distinct sequences being independent. To model the conditional distribution, the following log likelihood, sometimes called the conditional log likelihood, is appropriate:

$$l(\theta) = \sum_{i=1}^{N} \log p(y^{(i)}|x^{(i)}).$$

As an implementation of CRF, we use the Mallet library [86]. The library relies on the conjugate gradient optimization in order to estimate the parameters of the model.

We experiment with CRF in two different settings, according to the order of the connection between the instances. To label an unseen instance, the most likely Viterbi labeling $y^* = \arg\max_y p(y^{(i)}|x^{(i)})$ is computed. In the first setting we consider a linear chain CRF,

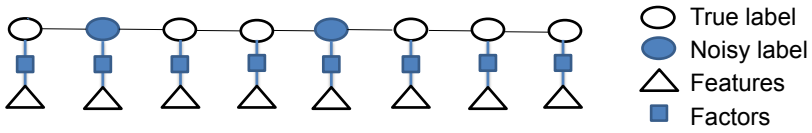
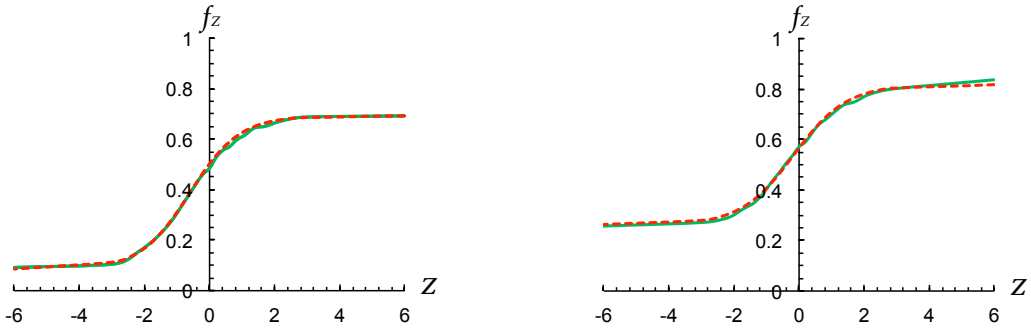


Figure 5.9: Linear chain CRF architecture.

in which an instance depends only on the previous one, as depicted in Figure 5.9. Since we consider only label noise, we induce noise only in the labels of the instances. In the second setting, we consider the model of the second order, where an instance depends on the two previous instances.

As an example, the sigmoids of the CRF algorithm for the "Robot walk 4" dataset are shown in Figure 5.10. Like in the case with HMMs, the sigmoids fit the performance curves of the CRF classifier well for all the datasets.



(a) Sigmoid CTF of CRF 1st setting.

(b) Sigmoid CTF of CRF 2nd setting.

Figure 5.10: Sigmoid CTF of the two settings of CRF-based classification algorithm for "Robot walk 4" dataset. Green solid line: true measurements, dashed red line: estimated sigmoid.

Classifier Based On Recurrent Neural Networks

The next sequential classification algorithm we study in this thesis is based on Recurrent Neural Networks (RNN). This algorithm applies a distinct view on the information flow, using a different graphical model for sequential data than the two previous algorithms. Whereas HMM takes into account only the data labels of the sequence, and CRF uses both class labels and the attributes of the previous instances in the sequence in order to classify the current instance, RNN uses the attributes of the current instance and only the label of the previous instances. Therefore, recurrent networks are sensitive to the past inputs and can adapt to them [15], but use different structure and less information than CRF. Here we describe how RNN is different from a feed forward (usual) neural network. Consider a two-layered network excluding the input layer, each layer has its own index variable k for output nodes, j for hidden states, h for hidden weight, and i for input nodes.

In a feed forward network, the input vector x is propagated through a weight layer V as follows:

$$y_j(t) = f(net_j(t)),$$
$$net_j(t) = \sum_i^n x_i(t)v_{ji} + \theta_j,$$

where n is the number of input, θ_j is a bias, and f is an output function (of any differentiable type).

In a simple recurrent network, the input vector is similarly propagated through a weight layer, but also combined with the previous state activation through an additional recurrent weight layer U

$$y_j(t) = f(net_j(t)),$$
$$net_j(t) = \sum_i^n x_i(t)v_{ji} + \sum_h^m y_h(t-1)u_{jh} + \theta_j,$$

where m is the number of "state" nodes.

The output of the network is in both cases determined by the state and the set of output weights, W,

$$y_k(t) = g(net_k(t)),$$
$$net_k(t) = \sum_j^m y_j(t)w_{kj} + \theta_k,$$

where g is an output function (possibly the same as f).

We use the implementation of Multilayer Perceptron from the Weka library [51] for our experiments with the RNN-based classifier. In this experimental setting, we provide the previous label as input, to calculate the output of the current instances (see Figure 5.11), as in the work [15]. We examine the two settings of the classifiers: in the first setting, the two previous labels are used, while in the second setting the three previous labels are used.

The sigmoids of the RNN algorithms for the "Robot walk 4" dataset are shown in Figure 5.12. According to Pearson's correlation test for setting 1 ($coef = 0.9947$, $p - value = 0.0053$) and setting 2 ($coef = 0.9962$, $p - value = 0.0038$) between the obtained performance and estimated sigmoid function, we can say that the classifier based on RNN also follows the "Sigmoid Rule". As in the previous cases of HMMs and CRFs, the sigmoids fit well the performance curves for all the datasets for the RNN classifier. Thus, the "Sigmoid Rule" fits RNN-based classifiers as well.

5.2.3 Using Sigmoid Rule Framework

We perform experiments of "noisy" classification using repeated random sub-sampling validation with 10 splits per algorithm per dataset per noise level, and calculate the

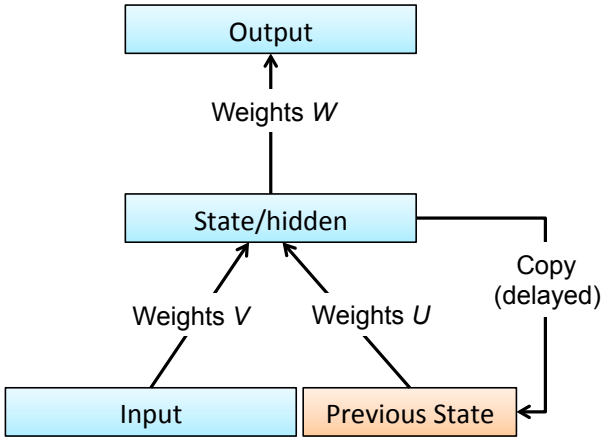
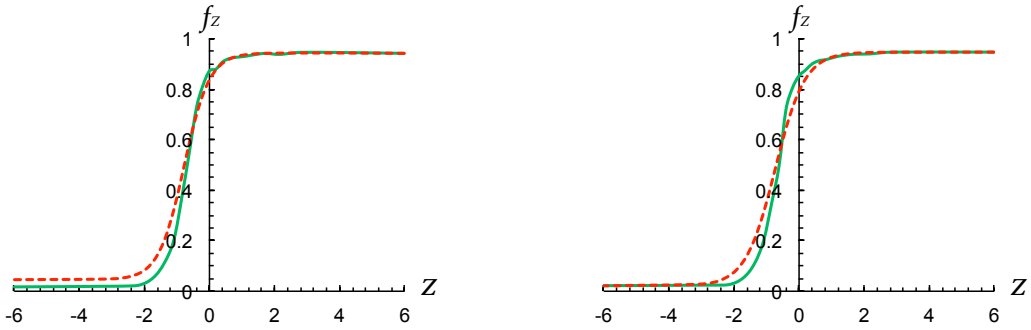


Figure 5.11: The architecture of RNN.



(a) Sigmoid CTF of RNN, 1st setting.

(b) Sigmoid CTF of RNN, 2nd setting.

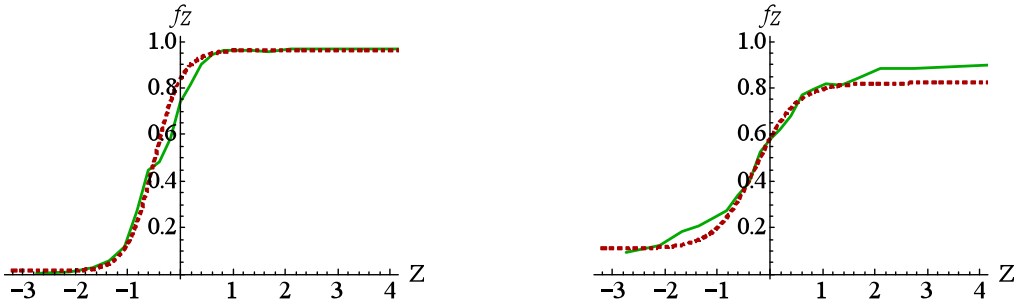
Figure 5.12: Sigmoid CTF of the two settings of the RNN-based classification algorithm for the "Robot walk 4" dataset. Green solid line: true measurements, dashed red line: estimated sigmoid.

average performance for each setting. Given the 20 levels of the signal-to-noise ratio and the corresponding algorithm performance, (i.e., classification accuracy) we estimated the parameters of the sigmoid. The search in the space of sigmoid parameters is performed by a genetic algorithm[4], as was proposed in [53].

A sample of true and the sigmoid-estimated performance graphs for varying levels of noise for a non-sequential classifier can be seen in Figure 5.13 and for sequential classifiers can be seen on Figures 5.8, 5.10 and 5.12. Although in our experiments the parameters of the sigmoid were estimated offline, the Sigmoid Rule Framework can be applied in an online scenario, after a certain training period.

Figure 5.14 illustrates the values of the mean and the standard deviation of the SRF parameters per algorithm, over all 24 datasets for the non-sequential classifiers. As an example of an interpretation of the figure using SRF, the plots indicate that (for the range

[4]The settings of the genetic algorithm can be found in the Appendix B.



(a) Sigmoid CTF of the SMO algorithm.

(b) Sigmoid CTF of the IBk algorithm.

Figure 5.13: Sigmoid CTF of SMO and IBk algrorithms for "Wine" dataset. Green solid line: True Measurements, dashed red line: estimated sigmoid.

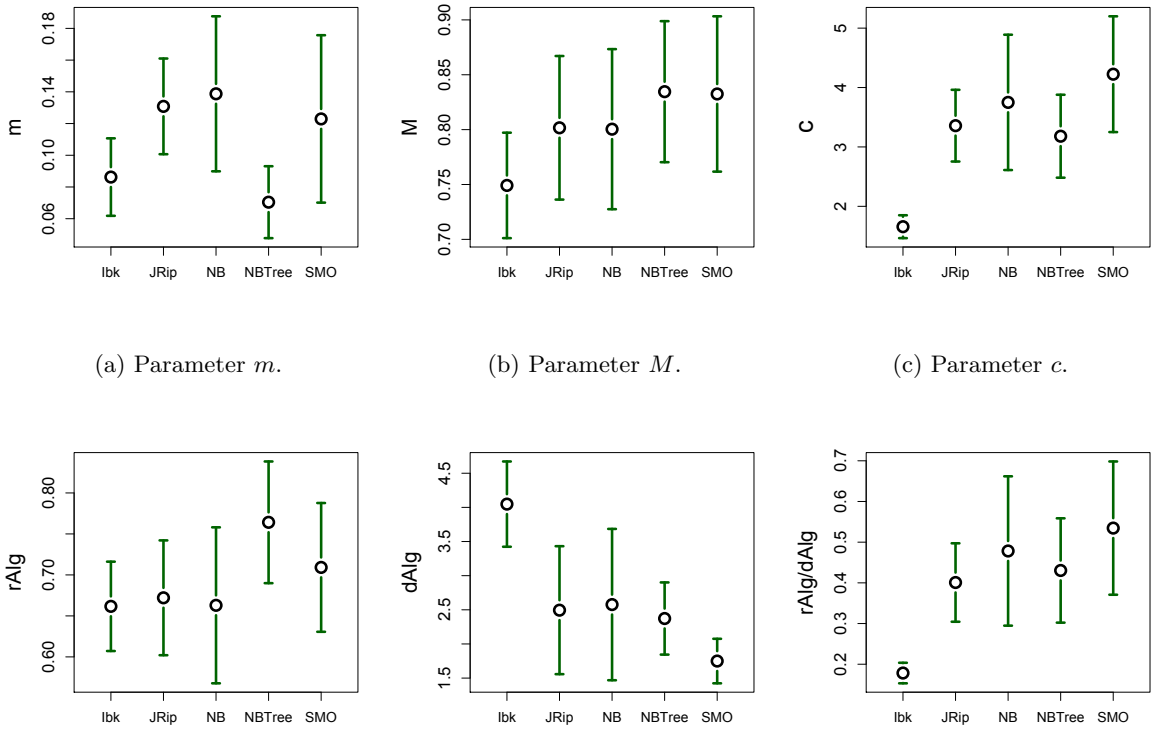
of datasets studied) SMO is expected to improve its performance faster on average than all other algorithms, when the signal-to-noise ratio increases. This conclusion is based on $\frac{r_{alg}}{d_{alg}}$ dimension and values of the slope indicator c (Figures 5.14f, 5.14c). At the same time, the confidence interval for $\frac{r_{alg}}{d_{alg}}$ and c of the Naïve Bayes algorithm overlaps with that of the SMO, so these algorithms can also be recommended if the same criteria are optimized. Nevertheless, SMO and Naïve Bayes algorithms can be well differentiated by the d_{alg} criterion (Figure 5.14e).

IBk has a smaller potential for improvement of performance (but also smaller potential for loss) than SMO, when noise levels change, given that the width of the performance range r_{alg} is higher for SMO (Figure 5.14d). This difference can also be seen in Figure 5.13, where the distance between the minimum and the maximum performance values is larger for the SMO case (see Figure 5.13a).

Figure 5.15 illustrates the values of the mean and the standard deviation for the SRF parameters per algorithm, over all 18 datasets for the sequential case. The plots indicate that HMM is expected to improve its performance faster than other algorithms, when the signal-to-noise ratio increases (Figures 5.15c and 5.15f). Though the two settings of RNN perform similarly along these parameters, they are clearly distinct from the rest of the algorithms. On the other hand, CRF has smaller potential for improvement of its performance, but also smaller risk for low performance than HMM, when noise levels change (Figures 5.15f and 5.15d).

As can be seen from Figure 5.15, all the families of the sequential classifiers have their specific behavior along the devised SRF dimensions, with certain variations depending on the order of the model.

We stress that the parameter estimation does not require previous knowledge of the noise levels, but is dataset dependent. In the special case of a classifier selection process, having an estimate of the noise level in the dataset helps to reach a decision through the



(a) Parameter m. (b) Parameter M. (c) Parameter c.

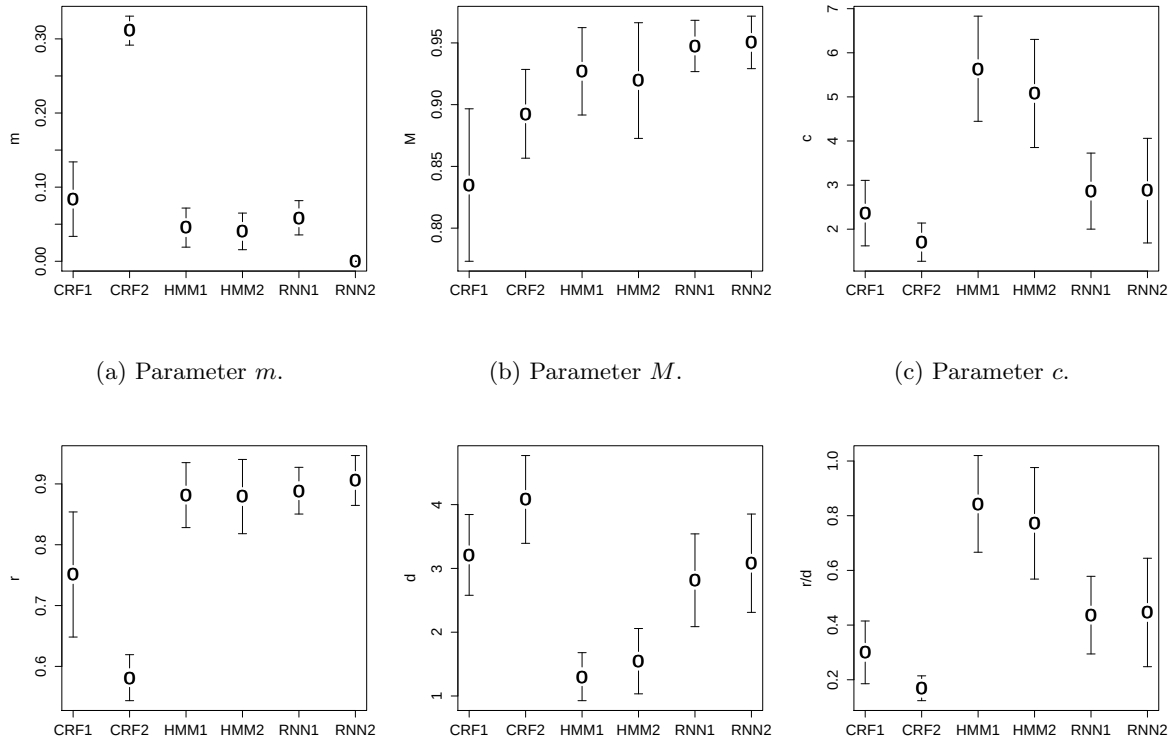
(d) Parameter r_{alg}. (e) Parameter d_{alg}. (f) Parameter r_{alg}/d_{alg}.

Figure 5.14: Estimated SRF parameters per algorithm. X axis labels (left-to-right): IBk, JRip, NB, NBTree, SMO.

use of SRF. In the following, we demonstrate the existence of this connection between the parameters of the sigmoid curve and the characteristics of the datasets. Therefore, SRF can help in choosing the better learner, provided that we know the characteristic of the dataset.

5.2.4 Statistical Analysis

We now study the connection between the dataset characteristics and the sigmoid parameters, irrespective of the choice of the algorithm. We consider the results obtained from all the algorithms as different samples of the SRF parameters for a particular dataset. We use regression analysis to observe the cumulative effect of the dataset characteristics on a single parameter, and we use correlation analysis to detect any connection between each pair of the dataset characteristic and the sigmoid parameter. We examine the connections between the dataset characteristics and the sigmoid parameters both individually, and all together, in order to draw the complete picture.



(a) Parameter m. (b) Parameter M. (c) Parameter c.

(d) Parameter r_{alg}. (e) Parameter d_{alg}. (f) Parameter r_{alg}/d_{alg}.

Figure 5.15: SRF parameters per algorithm. X axis labels (left-to-right): CRF 1st setting and 2nd setting, HMM 1st setting and 2nd setting, RNN 1st setting and 2nd setting.

Linear regression

We use the "lm" method (Fitting Linear Models) implemented in the package "stats" in R 2.11.1 to estimate the values of the parameters of the linear regression model for each chosen parameter y:

$$y = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4.$$

For choosing the best model, we used a modification of the *bestlm* function from *nsRFA* package in R. This function has been obtained using the function *leaps* of the R package *leaps*. The criterion for the best model was the adjusted R^2 statistic (the closer the adjusted R^2 is to one, the better is the estimation). R^2 defines the proportion of variation in the dependent variable (y) that can be explained by the predictors (X variables) in the regression model.

As some variation of y can be predicted by chance, the adjusted value of R_a^2 is used. Adjusted R^2 takes into account the number of observations (N) and the number of pre-

dictors (k), and is computed using the formula:

$$R_a^2 = 1 - \frac{(1 - R^2)(N - 1)}{N - k - 1}.$$

After applying the linear model, we exclude the non-significant parameters, and search for the model that maximizes the adjusted R^2 statistic (i.e., the best linear model).

In addition to R_a^2, we applied a statistical test with significance level $\alpha = 0.05$ for each chosen regression model in order to check if the model fits well. If the model describes the distribution of the values statistically significantly, then, after the hypothesis testing, the p-value should be within the significance level.

We applied a leave-one-out validation process, where one dataset is left out from training and used for testing on every run. We performed this analysis separately with each of the variables m, M, r_{alg}, d_{alg}, $\frac{r_{alg}}{d_{alg}}$, and c as a dependent variable.

Non-sequential case.

For the non-sequential classifiers, the following dataset parameters were chosen: a number of classes (x_1), a number of features (x_2), a number of instances (x_3), and the intrinsic dimensionality (x_4) of a dataset[5]as its fractal correlation dimension [23].

The results of model fitting and prediction of the SRF dimensions are reported in Table 5.3, where average errors between the observed and predicted SRF dimensions are shown. For each SRF dimension chosen, we have observed 5 values (since 5 machine learning algorithms were used), and having estimated them for 24 datasets, we ended up with 120 predictions for a single SRF dimension. We calculated four types of errors: (1) MSE – mean square error; (2) MAE – mean absolute error; (3) RMSE – relative mean square error, and (4) RMAE – relative mean absolute error. The last column of Table 5.3 shows the average of the adjusted R^2 statistic for models that where estimated for all the SRF dimensions (average on the 24 datasets).

Table 5.3: Prediction error of linear regression models for non-sequential classifiers.

Parameters	**Error measures**				**average**(R_a^2)
	MSE	MAE	RMSE	RMAE	
m	0.11	0.09	148.92	29.17	0.54
M	0.35	0.30	0.51	0.41	0.88
r_{alg}	0.32	0.27	0.71	0.46	0.85
d_{alg}	1.98	1.41	0.97	0.68	0.67
$\frac{r_{alg}}{d_{alg}}$	0.37	0.27	4.83	1.46	0.55
c	2.40	1.81	1.41	0.78	0.65

Figure 5.16 illustrates how our models fit the test data, showing that in most cases the true values of the sigmoid parameters for each dataset (illustrated by circles that

[5]We would like to thank Christos Faloutsos for kindly providing the code for the fractal dimensionality estimation.

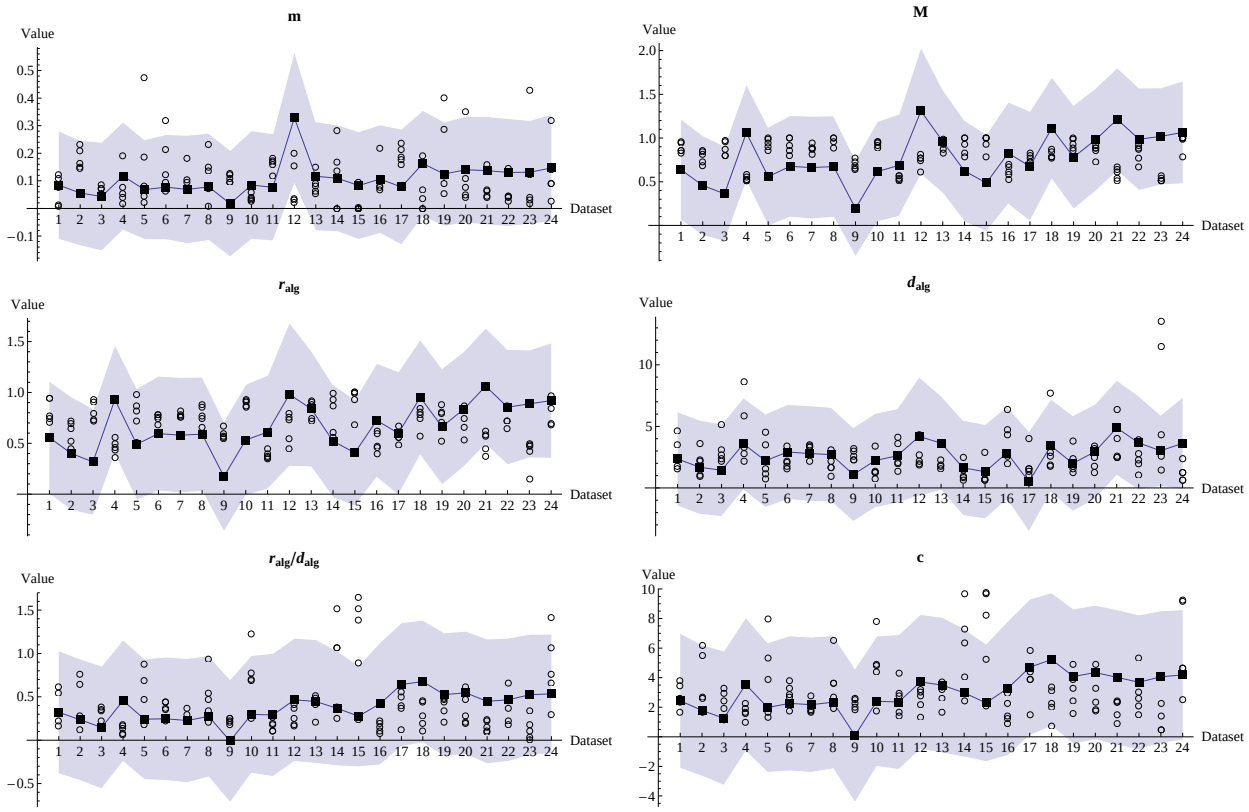


Figure 5.16: Real and estimated values of the sigmoid parameters for non-sequential classifiers. Real values: Black rectangles, Estimated values: circles, Gray zone: 95% prediction confidence interval.

correspond to 5 algorithms for each test dataset i, $i = 1, 2, ..., 24$) are within the 95% confidence level zone around the estimated values. This finding further supports the connection between the dataset parameters and the SRF dimensions. According to the results, the chosen parameters of the datasets can be used to reason about the parameters of the sigmoid. This allows us to recommend algorithms for a new dataset with known characteristics, if we have estimated the SRF dimensions for a set of algorithms on the datasets with similar characteristics.

Sequential case.

In this set of experiments, not all the sequential classification algorithms use the features of the data instances. Therefore, we examine the influence of the dataset on the CTF parameters by studying the number of classes (x_1), the number of instances (x_2), and the maximal autocorrelation lag of a dataset (x_3), which is described in more detail in Section 5.2.2.

We also applied a leave-one-out method for six dependent variables, as in the non-sequential dataset case. The results of model fitting and predictions of SRF dimensions are reported in Table 5.4.

For each SRF dimension chosen, we have observed 6 values, since 3 sequential learning algorithms were used, and each algorithm is applied in 2 different settings of dependency order. These 6 SRF dimensions are estimated for 18 datasets, thus 108 predictions were done for each SRF dimension. We also calculated four types of errors, as with the non-sequential datasets (Table 5.4).

Table 5.4: Prediction error of linear regression models for sequential data.

Parameters	**Error measures**				**average**(R_a^2)
	MSE	MAE	RMSE	RMAE	
m	0.13	0.10	46.80	8.67	0.25
M	0.41	0.33	0.45	0.37	0.81
r_{alg}	0.37	0.30	0.48	0.39	0.81
d_{alg}	2.01	1.57	0.98	0.69	0.58
$\frac{r_{alg}}{d_{alg}}$	0.36	0.29	1.24	0.88	0.67
c	2.38	1.88	0.85	0.68	0.67

Just like the non-sequential data, for sequential data classification, we can see the relationship between the dataset parameters and the SRF dimensions. Figure 5.17 illustrates how our models fit the test data.

The results show that most of the true parameter values are within the 95% prediction confidence levels. Thus, we can use the dataset parameters in order to reason about the parameters of the sigmoid of the algorithms. This enables us to recommend the algorithm of choice in advance for a new dataset with known characteristics. We can additionally recommend the order of the sequential classifier, if we have computed the SRF dimensions for a set of algorithms for datasets with similar characteristics.

Logistic regression

We also employed logistic regression, by applying the same leave-one-out process, for the purpose of predicting the SRF dimensions. While applying logistics regression, we use the Akaike Information Criterion (AIC) instead of the adjusted R^2_{adj} in order to choose the best model.

Being a measure of the relative quality of a statistical model for given data, AIC provides the means for model selection. In the general case, the AIC is calculated as follows

$$AIC = 2k - 2\ln(L),$$

where k is a number of parameters in the statistical model, and L is the maximized value of the likelihood function for the estimated model [4]. We choose the best prediction model by minimizing the AIC.

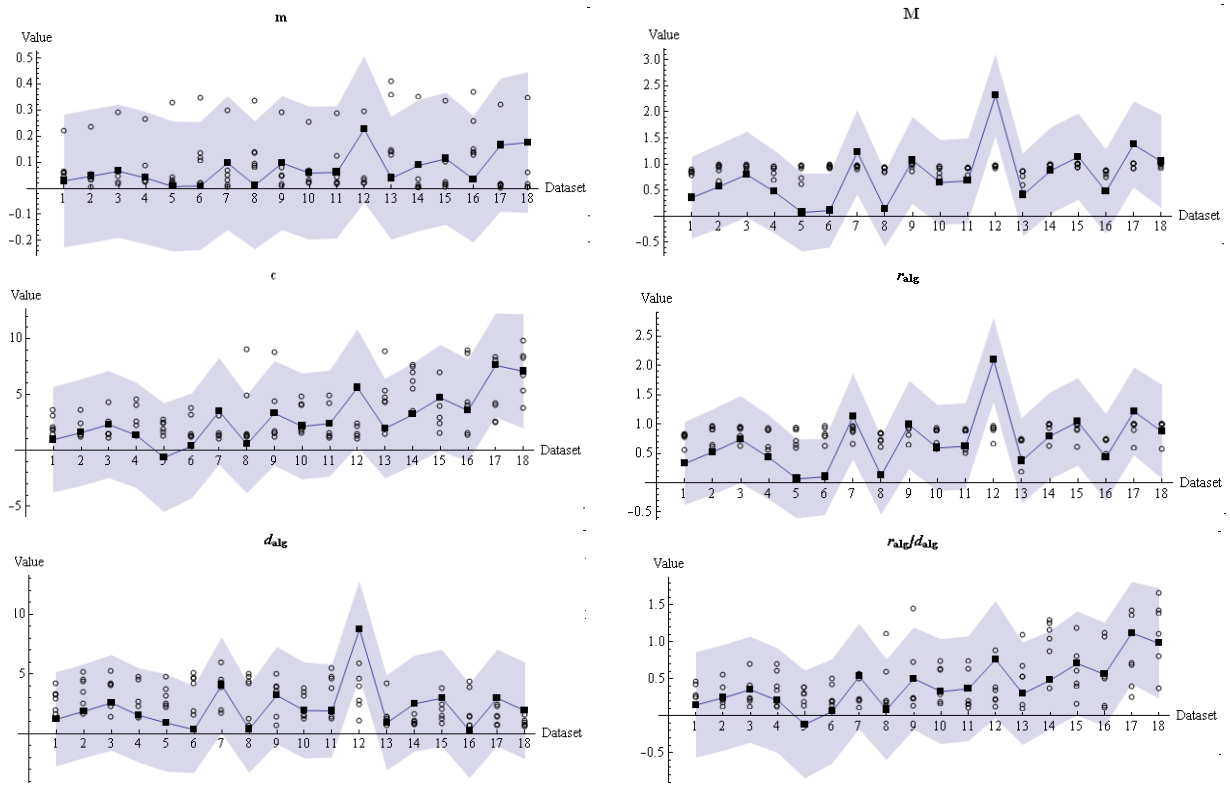


Figure 5.17: Real and estimated values of the sigmoid parameters for sequential classifiers. Real values: black rectangles, Estimated values: circles, Gray zone: 95% prediction confidence interval.

We also calculate four types of errors, as in linear regression. The results of model fitting and prediction of SRF dimension for non-sequential data sets are shown in Table 5.5, while the results for sequential data sets are shown in Table 5.6.

According to the experimental evaluation, the logistic regression model gives better prediction for some SRF dimensions, producing relatively smaller errors compared to the linear regression results. For both the non-sequential (Tables 5.3 and 5.5) and sequential (Tables 5.4 and 5.6) cases, the parameters M, r_{alg}, and c are better predicted with the logistic regression model. The accurate prediction of the CTF parameters supports the connection between the dataset characteristics and the SRF dimensions. Thus, given the characteristics of a dataset, we can reason about the parameters of the sigmoid of the algorithms, which enables us to recommend the algorithms of choice in advance for the dataset at hand.

Correlation analysis

We used three different correlation coefficients – the Pearson's correlation coefficient for linear correlation, the Spearman's rho, and the Kendall's tau coefficients for monotonic

Table 5.5: Prediction error of logistic regression models for non-sequential data.

Parameters	**Error measures**				**average(AIC)**
	MSE	MAE	RMSE	RMAE	
m	0.14	0.10	300.26	51.73	37.26
M	0.19	0.16	0.27	0.22	54.30
r_{alg}	0.20	0.17	0.51	0.30	120.91
d_{alg}	2.07	1.27	0.70	0.51	-220.05
$\frac{r_{alg}}{d_{alg}}$	0.38	0.29	3.90	1.47	-289.60
c	2.15	1.42	0.76	0.47	-246.88

Table 5.6: Prediction error of logistic regression models for sequential data.

Parameters	**Error measures**				**average(AIC)**
	MSE	MAE	RMSE	RMAE	
m	0.15	0.11	15.98	5.40	38.11
M	0.13	0.09	0.15	0.10	36.19
r_{alg}	0.17	0.14	0.35	0.20	68.19
d_{alg}	3.03	2.40	1.15	0.96	131.51
$\frac{r_{alg}}{d_{alg}}$	0.33	0.26	1.30	0.87	-255.32
c	2.29	1.63	0.53	0.48	-223.99

correlation – to analyze the connection between the parameters of the datasets and the SRF dimensions. Each coefficient varies from -1 (total negative correlation) to 1 (total positive correlation). A value of 0 implies that there is no correlation between the variables.

We qualitatively interpret the strength of the absolute values of correlation as follows: $[0.0; 0.1)$ →No Correlation, $[0.1; 0.3)$ →Low Correlation, $[0.3; 0.5)$ →Medium Correlation, $[0.5; 1]$ →Strong Correlation. This interpretation is widely accepted [30], [47], [117], though the borders may vary slightly depending on the domain (for example, in medicine, higher thresholds for strong correlation are usually required [110]).

Non-sequantial case.

In the non-sequential case, the parameters of a dataset are the following: x_1 – the number of classes, x_2 – the number of features, x_3 – the number of instances, and x_4 – the intrinsic dimensionality. The results of the correlation tests are shown in Table 5.7. Colored cells: statistically significant correlation ($p - value < 0.05$ **underlined bold** cells, $p - value < 0.1$ ***italics-bold*** cells). Green (dark) cells mark the pairs that have medium correlation, gray (light) indicate low correlation.

Summarizing the results from all the correlation coefficients (refer to Table 5.7), some interesting conclusions can be drawn. First, the number of classes (x_1) is inversely correlated to $\frac{r_{alg}}{d_{alg}}$, c, r_{alg} and M. Thus, the higher the number of classes is, the lower the

Table 5.7: The values of three correlation coefficients between parameters of the dataset and the parameters of the sigmoid. Non-sequential case.

Pearson's correlation coefficient							
Parameter		m	M	r_{alg}	d_{alg}	$\frac{r_{alg}}{d_{alg}}$	c
x1	$coef$	-0.03	-0.26	-0.21	0.13	-0.29	-0.28
	$p-val$	0.736	**0.005**	**0.025**	0.179	**0.001**	**0.002**
x2	$coef$	-0.07	-0.31	- 0.23	0.13	-0.21	-0.18
	$p-val$	0.463	**0.001**	**0.012**	0.173	**0.025**	**0.049**
x3	$coef$	0.14	0.08	-0.01	-0.12	0.12	0.16
	$p-val$	0.130	0.402	0.950	0.209	0.193	***0.093***
x4	$coef$	0.04	-0.16	-0.16	0.13	-0.09	-0.06
	$p-val$	0.690	***0.086***	***0.092***	0.168	0.364	0.549
Spearman's rank correlation coefficient							
Parameter		m	M	r_{alg}	d_{alg}	$\frac{r_{alg}}{d_{alg}}$	c
x1	$coef$	0.02	-0.26	-0.25	0.31	-0.34	-0.34
	$p-val$	0.810	**0.005**	**0.008**	**0.001**	**0.000**	**0.000**
x2	$coef$	0.03	-0.26	-0.24	0.14	-0.20	-0.15
	$p-val$	0.718	**0.006**	**0.011**	0.124	**0.030**	0.110
x3	$coef$	-0.05	0.03	0.02	-0.21	0.18	0.18
	$p-val$	0.579	0.752	0.863	**0.024**	***0.053***	***0.054***
x4	$coef$	-0.03	-0.20	-0.18	0.06	-0.11	-0.07
	$p-val$	0.712	**0.036**	***0.058***	0.559	0.241	0.426
Kendall's τ rank correlation coefficient							
Parameter		m	M	r_{alg}	d_{alg}	$\frac{r_{alg}}{d_{alg}}$	c
x1	$coef$	0.01	-0.17	-0.18	0.22	-0.24	-0.24
	$p-val$	0.909	**0.009**	**0.007**	**0.001**	**0.000**	**0.000**
x2	$coef$	0.02	-0.18	-0.16	0.10	-0.14	-0.11
	$p-val$	0.715	**0.007**	**0.013**	0.111	**0.032**	***0.094***
x3	$coef$	-0.05	0.01	0.01	-0.14	0.12	0.12
	$p-val$	0.400	0.818	0.896	**0.032**	***0.063***	***0.071***
x4	$coef$	-0.03	- 0.12	-0.11	0.04	-0.07	-0.06
	$p-val$	0.681	***0.062***	***0.081***	0.507	0.267	0.378

sensitivity to noise variation (check on $\frac{r_{alg}}{d_{alg}}$); the lower the number of classes, the higher the impact of reducing noise on performance (check r_{alg} and M). These conclusions are also supported by the direct correlation between the number of classes and the width of the active area of the algorithm d_{alg}. We also note the complete lack of significant correlation between the minimum performance m and all of the SRF dimensions: given enough noise an algorithm always performs badly. Thus, the number of classes significantly influences the behavior of an algorithm, regardless of the family of the algorithm.

Second, the number of features (x_2) provides a minor reduction of sensitivity to noise variation (resulting from low correlation to d_{alg} and c). This conclusion is also supported by the negative influence on $\frac{r_{alg}}{d_{alg}}$, r_{alg}. We also note that the number of features affects the maximal performance M, which shows (rather contrary to the intuition) that more features may negatively affect performance in a noise-free scenario. This is most probably related to features that are not essentially related to the labeling process, thus inducing feature noise. Third, there is a correlation between the number of instances (x_3), $\frac{r_{alg}}{d_{alg}}$, and the slope indicator c. This shows that larger datasets (providing more instances) increase the sensitivity to noise variation. Furthermore, in such highly populated datasets, reducing the label noise is expected to have a significant impact on performance. Last, the fractal dimensionality (x_4) of a dataset has low, but statistically significant, negative influence on M and on r_{alg}. Fractal dimensionality is indicative of the "complexity" of the dataset. Thus, if a dataset is complex (high x_4), machine learning is difficult even at low noise levels. We note that low r_{alg} may be preferable in cases where the algorithm should be stable even for low signal-to-noise ratios.

The correlation analysis demonstrates the connection between dataset characteristics and SRF dimensions. Consequently, the SRF can be used to reveal a priori the properties of an algorithm with respect to the dataset with certain characteristics. This allows an expert to select a good algorithm for a given setting, based on the requirements of that setting. Such requirements may, for example, relate to the stability of an algorithm to varying levels of noise, and the expected maximum performance in non-noisy datasets.

Both regression and correlation analysis showed that classifiers behave similarly for datasets with similar characteristics. We can therefore divide the datasets into groups with similar characteristics. When we need to choose a classifier for a new dataset, we can just find the group of datasets that is the closest to the given dataset, and recommend the best classifier for that group.

Sequential case.

For the sequential case, we study the connection between SRF dimensions and the following dataset characteristics: the number of classes (x_1), the number of instances (x_2), and the maximal autocorrelation lag of a dataset (x_3) (Table 5.8). Colored cells: statistically significant correlation ($p - value < 0.05$ **underlined bold** cells, $p - value < 0.1$ ***italics-bold*** cells). Orange (dark) cells mark the pairs that have strong correlation, green (dark) cells mark the pairs that have medium correlation, gray (light) indicate low correlation.

The results demonstrate that the number of classes x_1 is inversely correlated with the slope indicator c, and positively correlated with the active performance range d_{alg}, as in the non-sequential case. This leads to the following conclusion: the higher the number of classes is, the lower the sensitivity to noise variation is, regardless of the learning algorithm. The number of instances $x2$ is positively correlated with the slope indicators

Table 5.8: Three correlation coefficients between parameters of the dataset and parameters of the sigmoid. Sequential case.

Pearson's correlation coefficient							
Parameter		m	M	r_{alg}	d_{alg}	$\frac{r_{alg}}{d_{alg}}$	c
x1	$coef$	-0.117	0.063	0.113	0.238	-0.220	-0.265
	$p-val$	0.228	0.520	0.246	**0.013**	**0.022**	**0.006**
x2	$coef$	0.019	0.191	0.088	-0.446	0.523	-0.526
	$p-val$	0.842	**0.048**	0.362	**0.000**	**0.000**	**0.000**
x3	$coef$	-0.199	0.421	0.359	-0.117	0.226	0.125
	$p-val$	**0.039**	**0.000**	**0.000**	0.227	**0.018**	0.196
Spearman's rank correlation coefficient							
Parameter		m	M	r_{alg}	d_{alg}	$\frac{r_{alg}}{d_{alg}}$	c
x1	$coef$	-0.174	-0.004	0.082	0.393	-0.290	-0.393
	$p-val$	***0.072***	0.964	0.396	**0.000**	**0.002**	**0.000**
x2	$coef$	-0.099	-0.264	0.165	-0.483	0.409	0.483
	$p-val$	0.309	**0.006**	***0.088***	**0.000**	**0.000**	**0.000**
x3	$coef$	-0.335	0.511	0.434	-0.127	0.221	0.127
	$p-val$	**0.000**	**0.000**	**0.000**	0.190	**0.021**	0.190
Kendall's τ rank correlation coefficient							
Parameter		m	M	r_{alg}	d_{alg}	$\frac{r_{alg}}{d_{alg}}$	c
x1	$coef$	-0.124	0.001	0.067	0.302	-0.219	-0.302
	$p-val$	***0.080***	0.991	0.347	**0.000**	**0.002**	**0.000**
x2	$coef$	-0.068	0.176	-0.16	-0.328	0.280	0.328
	$p-val$	0.306	**0.008**	***0.081***	**0.000**	**0.000**	**0.000**
x3	$coef$	-0.234	-0.354	0.295	-0.089	0.160	0.089
	$p-val$	**0.000**	**0.000**	**0.000**	0.191	**0.019**	0.191

r_alg/d_{alg} and c, and is negatively correlated with the active area of the algorithm. Thus, the number of instances has the opposite effect of the number of classes, and in datasets with a large number of instances, reducing the label noise is expected to have a significant impact on performance. The same effect is observed for non-sequential datasets. The autocorrelation lag x_3 has an influence on the performance range: if the dependency history is deep (high $x3$), then the maximal performance grows (M), and the minimal performance drops (m). Thus, the expected accuracy of sequential classification task is higher if the dataset has deeper history, which is an intuitive result.

Compared to the correlation analysis of non-sequential data, sequential data overall shows stronger correlations. Therefore, the characteristics of sequential data exhibit a connections to the SRF dimensions, which implies a better potential for predicting the best suitable algorithm for a given dataset.

5.3 Summary

Machine learning algorithms are often used in a noisy environment. Therefore, it is important to know a priori the properties of an algorithm depending on the characteristics of the dataset. In this work, we investigated whether some simple dataset properties (namely, the number of classes, the number of features, the number of instances, the fractal dimensionality for the non-sequential case, and the maximal autocorrelation lag for the sequential case) can help in the above direction.

We proposed the Sigmoid Rule Framework, describing a set of dimensions that may be used by a decision maker to choose a good classifier based on a range of the dataset characteristics. Our approach is applicable to user modeling tasks, when the user changes behavior over time, and to any concept drift problems for data series mining.

We showed that the parameters related to the behavior of learners correlate with the dataset characteristics, and the range of their variation may be predicted using linear or logistic regression models. Therefore, SRF is a useful meta-learning framework, applicable to a wide range of settings that include noise. The SRF models for parameter prediction, however, do not provide enough precision to predict the performance of the algorithms with high accuracy.

Chapter 6

Conclusion

Large amounts of streaming data are originating from observing natural phenomena connected to human activities, sensor readings, moving trajectories, networking and others. To process data streams in real time, we need effective algorithms for handling various tasks, such as detecting sequential patterns, data stream summarization, classification, and prediction of the future states. These online algorithms need to maintain a compact representation of the potentially infinite data stream. As the observations of the data streams are often strongly correlated, the representations should take into account the dependencies among the observations.

In this thesis we proposed two notions that capture important correlations in streaming data: Conditional Heavy Hitters and Variable Order Conditional Heavy Hitters. These notions describe sequential patterns of fixed and variable order, respectively. The definition of fixed order Conditional Heavy Hitters is based on high conditional and joint probability, while Variable Order Conditional Heavy Hitters additionally requires the statistical significance of the pattern occurrence. Facing the tradeoff between memory usage and accuracy of the extracted patterns, we proposed streaming algorithms that efficiently estimate Conditional Heavy Hitters of the fixed and variable order. We provided analytical quality guarantees for the proposed algorithms, and also analyzed their applicability for data with different characteristics. The experimental evaluation on synthetic and real datasets showed that the algorithms are able to accurately estimate Conditional Heavy Hitters for various types of data. With our experiments on the moving trajectories dataset we demonstrated that the extracted Conditional Heavy Hitters can efficiently estimate the generative model of the data.

Data streams often contain noisy observations. For the task of classification, it is important to understand the behavior of machine learning algorithms in noisy environment. Moreover, it is essential to know a priori how the properties of the algorithm depend on the characteristics of the dataset. In this thesis, we proposed the "Sigmoid Rule" Framework, which describes a set of dimensions that are useful for choosing the appro-

priate classifier for a particular user need. Our approach is applicable to user modeling tasks, when the user changes behavior over time, and to various concept drift problems for data series mining. We discovered that the simple dataset properties (namely, number of classes, number of features, number of instances, fractal dimensionality for the non-sequential case, and maximal autocorrelation lag for the sequential case) influence the performance of the classifiers. We showed experimentally that the parameters related to the behavior of learners correlate with dataset characteristics, and the range of their variation may be predicted using linear or logistic regression models. The experimental results demonstrated that SRF is a useful meta-learning framework, applicable to a wide range of settings that include noise.

6.1 Future work

6.1.1 Conditional Heavy Hitters

Further Theoretical Analysis and Empirical Evaluation. Our study of the Conditional Heavy Hitters of variable order is our most recent work and it requires more thorough analysis and experimental evaluation. First, we would like to investigate the existence of analytical quality guarantees for all the proposed online pruning strategies, and consider the applicability of the optimal pruning criterion to the streaming setting. We plan to evaluate, both experimentally and analytically, how the parameters of the procedures, such as deep digging and periodical pruning, influence the accuracy of VOCHH estimation. For the evaluation of the algorithms, we are planing to use more datasets from different domains, especially text, which is known to be accurately modeled with offline VMMs. Nowadays, the large sources of text streams are news, such as online and regular reports, comments on news, online traffic reports. Another source is social media, such as discussion forums (e.g., Twitter, Facebook), blogs and others. All the text streams have strong temporal dimension, and Conditional Heavy Hitters can be used to find significant sequential patterns in text, build text models, summarize text in order to capture the evolution of topics over time.

Advanced Probabilistic Models. In this thesis, we applied Conditional Heavy Hitters to estimate the parameters of Markov chains. We envision that Conditional Heavy Hitters and VOCHH can be used as a building block for estimating more complex probabilistic models. The models contain more parameters and can also take into account the attributes (features) of the data stream elements, not only the states as considered in this thesis. In future work, we would like to investigate the potential of Conditional Heavy Hitters and VOCHH for estimating more complex models with temporal correlations such as Hidden Markov Models of high orders and other types of Bayesian models. This might require the adaptation of the proposed algorithms to the structures of the models.

Distributed Setting. In many real life systems, data streams from multiple sources should be processed simultaneously. The computational power required to process multiple streams in real time may exceed the capabilities of a single machine. As a future work, we would like to parallelize the presented algorithms for discovering Conditional Heavy Hitters of fixed and variable order and apply them in the distributed settings, where the maintained summaries of the individual streams should be combined to give a summary of the union of all the input data. Conditional Heavy Hitters rely greatly on the notion of conditional probability. For computing the top-τ query, even estimating the conditional probabilities on the union of the data streams, given the data structures described in this thesis, is not an obvious task, even if the counts are maintained exactly. A fruitful direction for further research is investigating what quality guarantees for the union of data the separately maintained data structures of the individual streams can allow.

6.1.2 Learning in The Presence of Noise

In this thesis we presented the Sigmoid Rule Framework (SRF) that helps choosing the learning algorithm depending on the characteristics of the dataset. The next step in this research direction is building an SRF-based system that could provides recommendations to the users depending on their preferences and peculiarities of the data that needs to be processed. For this purpose, the study we presented in the thesis should be extended with both new learning algorithms and the datasets with diverse characteristics.

According to the results of the current work, different dependencies between dataset characteristics and SRF dimensions are best fitted by different statistical models. We would like to continue our research in this direction and examine if the dependencies can be better described by other models, or learned using more complex systems, for example, neural networks.

Bibliography

[1] Charu Aggarwal, editor. *Data Streams – Models and Algorithms*. Springer, 2007.

[2] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. On demand classification of data streams. In *KDD*, pages 503–508, 2004.

[3] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.

[4] H. Akaike. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723, Dec 1974.

[5] S. Ali and K.A. Smith. On learning algorithm selection for classification. *Applied Soft Computing*, 6(2):119–138, 2006.

[6] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *ACM Symposium on Theory of Computing*, pages 20–29, 1996.

[7] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *ACM Principles of Database Systems*, 2004.

[8] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.

[9] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[10] Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. Efficient lz78 factorization of grammar compressed text. In *String Processing and Information Retrieval*, volume 7608, pages 86–98. Springer, 2012.

[11] LE Baum, T Petrie, G Soules, and N Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, 1970.

[12] Leonard E. Baum and Ted Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.

[13] Ron Begleiter, Ran El-Yaniv, and Golan Yona. On prediction using variable order markov models. *CoRR*, abs/1107.0051, 2011.

[14] Fabrizio Biondi, Axel Legay, BoFriis Nielsen, and Andrzej Wsowski. Maximizing entropy over markov processes. In *Language and Automata Theory and Applications*, volume 7810 of *Lecture Notes in Computer Science*, pages 128–140. Springer Berlin Heidelberg, 2013.

[15] M Bodén. A guide to recurrent neural networks and backpropagation. *The Dallas project, SICS technical report*, pages 1–10, 2002.

[16] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[17] B. Boyer and J. Moore. A fast majority vote algorithm. Technical Report ICSCA-CMP-32, Institute for Computer Science, University of Texas, February 1981.

[18] A.P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[19] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2005.

[20] Ceren Budak, Theodore Georgiou, Divyakant Agrawal, and Amr El Abbadi. Geoscope: Online detection of geo-correlated information trends in social networks. *PVLDB*, 7(4):229–240, 2013.

[21] S. Budhaditya, Duc-Son Pham, M. Lazarescu, and S. Venkatesh. Effective anomaly detection in sensor networks data streams. In *Data Mining, 2009. ICDM '09. Ninth IEEE International Conference on*, pages 722–727, 2009.

[22] Suzanne Bunton. On-line stochastic processes in data compression, 1996.

[23] F. Camastra and A. Vinciarelli. Estimating the intrinsic dimension of data with a fractal-based method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(10):14041407, 2002.

[24] Joong Hyuk Chang and Won Suk Lee. Finding recent frequent itemsets adaptively over online data streams. In *KDD*, pages 487–492, 2003.

[25] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Procedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2002.

[26] Yann Chevaleyre and Jean-Daniel Zucker. Noise-tolerant rule induction from multi-instance data. In *In Proceedings of the ICML-2000 workshop on Attribute-Value and Relational Learning: Crossing the Boundaries, L. De Raedt*, 2000.

[27] J. Cleary and I. Witten. Data compression using adaptive coding and partial string matching. *Communications, IEEE Transactions on*, 32(4):396 – 402, apr 1984.

[28] John G. Cleary, W.J. Teahan, and Ian H. Witten. Unbounded length contexts for ppm. In *Data Compression Conference, 1995. DCC '95. Proceedings*, pages 52–61, Mar 1995.

[29] William W. Cohen. Fast effective rule induction. In *ICML*, 1995.

[30] Gregory W Corder and Dale I Foreman. *Nonparametric statistics for non-statisticians: a step-by-step approach*. John Wiley & Sons, 2009.

[31] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding hierarchical heavy hitters in data streams. In *International Conference on Very Large Data Bases*, pages 464–475, 2003.

[32] G. Cormode, F. Korn, and S. Tirthapura. Time decaying aggregates in out-of-order streams. In *ACM Principles of Database Systems*, 2008.

[33] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[34] Graham Cormode and Mario Hadjieleftheriou. Finding frequent items in data streams. In *VLDB*, 2008.

[35] Michele Dallachiesa, Besmira Nushi, Katsiaryna Mirylenka, and Themis Palpanas. Similarity matching for uncertain time series: analytical and experimental comparison. In *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Querying and Mining Uncertain Spatio-Temporal Data, QUeST 2011, Chicago, IL, USA, November 1, 2011*, pages 8–15, 2011.

[36] Michele Dallachiesa, Besmira Nushi, Katsiaryna Mirylenka, and Themis Palpanas. Uncertain time-series similarity: Return to the basics. *PVLDB*, 5(11):1662–1673, 2012.

[37] Michele Dallachiesa, Besmira Nushi, Katsiaryna Mirylenka, and Themis Palpanas. Uncertain time-series similarity: Return to the basics. *Proc. VLDB Endow.*, 5(11):1662–1673, July 2012.

[38] Michele Dallachiesa and Themis Palpanas. Identifying streaming frequent items in ad hoc time windows. *Data Knowl. Eng.*, 87:66–90, 2013.

[39] E.P.M. de Sousa, A.J.M. Traina, C. Traina Jr, and C. Faloutsos. Evaluating the intrinsic dimension of evolving data streams. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 643–648. ACM, 2006.

[40] E. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *European Symposium on Algorithms (ESA)*, 2002.

[41] Mukund Deshpande and George Karypis. Selective markov models for predicting web page accesses. *ACM Trans. Internet Technol.*, 4(2):163–184, 2004.

[42] Gideon Dror, Marc Boullé, Isabelle Guyon, Vincent Lemaire, and David Vogel, editors. *Proceedings of KDD-Cup 2009 competition, Paris, France, June 28, 2009*, volume 7 of *JMLR Proceedings*. JMLR.org, 2009.

[43] Tarn Duong, Bruno Goud, and Kristine Schauer. Closed-form density-based framework for automatic detection of cellular morphology changes. *Proceedings of the National Academy of Sciences*, 109(22):8382–8387, 2012.

[44] Pierre Dupont. Noisy sequence classification with smoothed markov chains. *Conférence francophone sur l'apprentissage ...*, 2006.

[45] Benjamin Van Durme and Ashwin Lall. Streaming pointwise mutual information. In *NIPS*, pages 1892–1900, 2009.

[46] JL Elman. Finding structure in time. *Cognitive science*, 211:179–211, 1990.

[47] Sean B Eom, Michael A Ketcherside, Hu-Hyuk Lee, Michael L Rodgers, and David Starrett. The determinants of web-based instructional systems' outcome and satisfaction: An empirical investigation. *Instructional technologies: Cognitive aspects of online programs*, pages 96–139, 2004.

[48] M. Feder and N. Merhav. Relations between entropy and error probability. *Information Theory, IEEE Transactions on*, 40(1):259–266, Jan 1994.

[49] C. Feregrino. High performance ppmc compression algorithm. In *Computer Science, 2003. ENC 2003. Proceedings of the Fourth Mexican International Conference on*, pages 135–142, 2003.

[50] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[51] Stephen R. Garner. Weka: The waikato environment for knowledge analysis. In *In Proc. of the New Zealand Computer Science Research Students Conference*, pages 57–64, 1995.

[52] Johannes Gehrke, Flip Korn, and Divesh Srivastava. On computing correlated aggregates over continual data streams. In *ACM SIGMOD International Conference on Management of Data*, pages 13–24, 2001.

[53] George Giannakopoulos and Themis Palpanas. The effect of history on modeling systems' performance: The problem of the demanding lord. In *ICDM*, 2010.

[54] George Giannakopoulos and Themis Palpanas. Revisiting the effect of history on learning performance: the problem of the demanding lord. *Knowledge and Information Systems*, 36(3):653–691, 2013.

[55] Chris Giannella, Jiawei Han, Jian Pei, X. Yan, and Philip S. Yu. Mining frequent patterns in data streams at multiple time granularities. In *Data Mining: Next Generation Challenges and Future Directions*, 2004.

[56] C. Giraud-Carrier, R. Vilalta, and P. Brazdil. Introduction to the special issue on meta-learning. *Machine Learning*, 54(3):187–193, 2004.

[57] Wei guang Teng, Ming syan Chen, and Philip S. Yu. A regression-based temporal pattern mining scheme for data streams. In *In VLDB*, pages 93–104, 2003.

[58] Sudipto Guha. On the space-time of optimal, approximate and streaming algorithms for synopsis construction problems. *VLDB J.*, 17(6):1509–1535, 2008.

[59] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. Knowl. Data Eng.*, 15(3):515–528, 2003.

[60] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[61] W. Hamalainen and M. Nykanen. Efficient discovery of statistically significant association rules. In *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, Dec 2008.

[62] J. Han and M. Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.

[63] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[64] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *SIGMOD Conference*, pages 1–12, 2000.

[65] SherriK. Harms, Jitender Deogun, and Tsegaye Tadesse. Discovering sequential association rules with constraints and time lags in multiple sequences. In Mohand-Sad Hacid, ZbigniewW. Ra, DjamelA. Zighed, and Yves Kodratoff, editors, *Foundations of Intelligent Systems*, volume 2366 of *Lecture Notes in Computer Science*, pages 432–441. Springer Berlin Heidelberg, 2002.

[66] H. V. Jagadish, Olga Kapitskaia, Raymond T. Ng, and Divesh Srivastava. One-dimensional and multi-dimensional substring selectivity estimation. *The VLDB Journal*, 9(3):214–230, 2000.

[67] Nan Jiang and Le Gruenwald. Research issues in data stream association rule mining. *SIGMOD Rec.*, 35(1):14–19, March 2006.

[68] jm.francois. a java implementation of hidden markov model (hmm), 2013.

[69] BH Juang and L Rabiner. The segmental K-means algorithm for estimating parameters of hidden Markov models. *Acoustics, Speech and Signal ...*, 3(9):1639–1641, 1990.

[70] Elias Kalapanidas, Nikolaos Avouris, Marian Craciun, and Daniel Neagu. Machine learning algorithms: a study on noise. Technical report, in 1st Balcan Conference in Informatics, 2003.

[71] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13(3):637–649, 2001.

[72] Christopher Kermorvant and Pierre Dupont. Improved smoothing for probabilistic suffix trees seen as variable order markov chains. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Machine Learning: ECML 2002*, volume 2430 of *Lecture Notes in Computer Science*, pages 185–194. Springer Berlin Heidelberg, 2002.

[73] Adam Kirsch, Michael Mitzenmacher, Andrea Pietracaprina, Geppino Pucci, Eli Upfal, and Fabio Vandin. An efficient rigorous approach for identifying statistically significant frequent itemsets. In *Proceedings of the Twenty-eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '09, pages 117–126, 2009.

[74] R. Kneser. Statistical language modeling using a variable context length. In *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, volume 1, pages 494–497 vol.1, Oct 1996.

[75] Anthony Kuh, Thomas Petsche, and Ronald L. Rivest. Learning time-varying concepts. In *NIPS*, pages 183–189, 1990.

[76] John Lafferty, A McCallum, and FCN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceeding ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning*, 2001(Icml):282–289, 2001.

[77] Bibudh Lahiri and Srikanta Tirthapura. Finding correlated heavy-hitters over data streams. In *IPCCC*, pages 307–314, 2009.

[78] L.K. Lee and H.F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *ACM Principles of Database Systems*, 2006.

[79] L.K. Lee and H.F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *ACM Principles of Database Systems*, 2006.

[80] Julie Letchner, Christopher Re, Magdalena Balazinska, and Matthai Philipose. Approximation trade-offs in markovian stream processing: An empirical study. In *ICDE*, pages 936–939, 2010.

[81] Q. Li, T. Li, S. Zhu, and C. Kambhamettu. Improving medical/biological data classification performance by wavelet preprocessing. *Proceedings ICDM Conference*, 2002.

[82] C. Low-Kam, A. Laurent, and M. Teisseire. Detection of sequential outliers using a variable length markov model. In *Machine Learning and Applications, 2008. ICMLA '08. Seventh International Conference on*, pages 571–576, Dec 2008.

[83] Nizar R. Mabroukeh and C. I. Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Comput. Surv.*, 43(1):3:1–3:41, December 2010.

[84] Nishad Manerikar and Themis Palpanas. Frequent items in streaming data: An experimental evaluation of the state-of-the-art. *Data Knowl. Eng.*, 68(4):415–430, 2009.

[85] G.S. Manku and R. Motwani. Approximate frequency counts over data streams. In *International Conference on Very Large Data Bases*, pages 346–357, 2002.

[86] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[87] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. Sliq: A fast scalable classifier for data mining. In *Advances in Database Technology EDBT '96*, volume 1057 of *Lecture Notes in Computer Science*, pages 18–32. Springer Berlin Heidelberg, 1996.

[88] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*, 2005.

[89] K. Mirylenka, T. Palpanas, G. Cormode, and D. Srivastava. Conditional heavy hitters: Detecting interesting correlations in data streams. *The International Journal on Very Large Data Bases (VLDBJ)*, page 120, 2015.

[90] Katsiaryna Mirylenka, Graham Cormode, Themis Palpanas, and Divesh Srivastava. Finding interesting correlations with conditional heavy hitters. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 1069–1080, April 2013.

[91] Katsiaryna Mirylenka, Le Minh Do, Themis Palpanas, and George Giannakopoulos. On classifier behavior in the presence of label noise. *Under submission*, 2015.

[92] Katsiaryna Mirylenka, George Giannakopoulos, and Themis Palpanas. Srf: A framework for the study of classifier behavior under training set mislabeling noise. In *Advances in Knowledge Discovery and Data Mining (PAKDD2012)*, volume 7301 of *Lecture Notes in Computer Science*, pages 109–121. Springer Berlin Heidelberg, 2012.

[93] J. Misra and D. Gries. Finding repeated elements. *Science of Computer Programming*, 2:143–152, 1982.

[94] Alistair Moffat. Implementing the ppm data compression scheme. *Communications, IEEE Transactions on*, 38(11):1917–1921, 1990.

[95] David F. Nettleton, Albert Orriols-Puig, and Albert Fornells. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33(4):275–306, January 2010.

[96] Mordechai Nisenson, Ido Yariv, Ran El-Yaniv, and Ron Meir. Towards behaviometric security systems: Learning to identify a typist. In *Knowledge Discovery in Databases: PKDD 2003*, volume 2838 of *Lecture Notes in Computer Science*, pages 363–374. Springer Berlin Heidelberg, 2003.

[97] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., 1988.

[98] Mark Pendrith and Claude Sammut. On reinforcement learning of control actions in noisy and non-markovian domains. Technical report, School of Computer Science and Engineering, the University of New South Wales, Sydney, Australia, 1994.

[99] L Rabiner and BH Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 1986.

[100] Michael Rabinovich and Oliver Spatschek. *Web caching and replication*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[101] Adrian E. Raftery. A model of high-order markov chains. *Journal of the Royal Statistical Society (Series B Methodological)*, 47(3):528–539, 1985.

[102] J. Rissanen. Universal coding, information, prediction, and estimation. *Information Theory, IEEE Transactions on*, 30(4):629–636, 1984.

[103] J. Rissanen and G. G. Langdon. Arithmetic coding. *IBM J. Res. Dev.*, 23(2):149–162, 1979.

[104] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3):117–149, 1996.

[105] Yossi Rubner, Carlo Tomasi, and LeonidasJ. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[106] Cynthia Rudin, Benjamin Letham, Ansaf Salleb-Aouissi, Eugene Kogan, and David Madigan. Sequential event prediction with association rules. In *COLT 2011 - The 24th Annual Conference on Learning Theory, June 9-11, 2011, Budapest, Hungary*, pages 615–634, 2011.

[107] DavidB. Skalak, Alexandru Niculescu-Mizil, and Rich Caruana. Classifier loss under metric uncertainty. In *Machine Learning: ECML 2007*, volume 4701 of *Lecture Notes in Computer Science*, pages 310–322. Springer Berlin Heidelberg, 2007.

[108] Charles Sutton and A McCallum. An introduction to conditional random fields. *arXiv preprint arXiv:1011.4088*, 2010.

[109] Ferry Irawan Tantono, Nishad Manerikar, and Themis Palpanas. Efficiently discovering recent frequent items in data streams. In *SSDBM*, pages 222–239, 2008.

[110] Richard Taylor. Interpretation of the correlation coefficient: a basic review. *Journal of diagnostic medical sonography*, 6(1):35–39, 1990.

[111] O. Teytaud. Learning with noise. Extension to regression. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 3, pages 1787–1792. IEEE, 2002.

[112] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Academic Press, 2003.

[113] E Ukkonen. Online construction of suffix trees. *Algorithmica*, 14(2):249–260, 1995.

[114] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[115] AJ Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, pages 260–269, 1967.

[116] David S. Vogel, Eric Gottschalk, and Morgan C. Wang. Anti-matter detection: Particle physics model for kdd cup 2004. *SIGKDD Explor. Newsl.*, 6(2):109–112, December 2004.

[117] Liska Waluyan, Sirikpudee Sasipan, Stephanie Noguera, and Tatsuo Asai. Analysis of potential problems in people management concerning information security in cross-cultural environment -in the case of malaysia-. In *HAISA*, pages 13–24, 2009.

[118] Peng Wang, Haixun Wang, and Wei Wang. Finding semantics in time series. In *SIGMOD Conference*, pages 385–396, 2011.

[119] Peng Wang, Haixun Wang, and Wei Wang. Finding semantics in time series. In *ACM SIGMOD International Conference on Management of Data*, pages 385–396, 2011.

[120] B. L. Welch. The Generalization of 'Student's' Problem when Several Different Population Variances are Involved. *Biometrika*, 34(1/2):28–35, 1947.

[121] I. H. Witten, T. C. Bell, A. Moffat, C. G. Nevill-Manning, T. C. Smith, and H. Thimbleby. Semantic and generative models for lossy text compression. *The Computer Journal*, 37(2):83–87, 1994.

[122] Ian H. Witten and T. Bell. The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. *Information Theory, IEEE Transactions on*, 37(4):1085–1094, 1991.

[123] David Wolpert. The existence of a priori distinctions between learning algorithms. *Neural Computation*, 8:1391–1421, 1996.

[124] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8:1341–1390, October 1996.

[125] D.H. Wolpert. The supervised learning no-free-lunch theorems. In *Proc. 6th Online World Conference on Soft Computing in Industrial Applications*. Citeseer, 2001.

[126] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. A brief survey on sequence classification. *ACM SIGKDD Explorations Newsletter*, 12(1):40, November 2010.

[127] Philip S. Yu and Yun Chi. Association rule mining on streams. In *Encyclopedia of Database Systems*, pages 136–139. Springer-Verlag, 2009.

[128] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *In Proceedings of the Eighteenth International Conference on Machine Learning*, pages 609–616. Morgan Kaufmann, 2001.

[129] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, 24(5):530–536, 1978.

Appendix A

Summary of Notations and Abbreviations

A.1 Conditional Heavy Hitters

We use the following the notations related to Conditional Heavy Hitters:

- $(\mathbf{p}, c)$ – a parent-child pair
- f_x – the frequency of an item x
- $Pr[\cdot]$ – the probability of an event
- s - space budget

We use the following the abbreviations related to Conditional Heavy Hitters:

- HH – Heavy Hitters
- SS structure – Space Saving structure
- CSS structure – Conditional Space Saving structure
- EMD – the Earth Movers distance
- MED – Mean Euclidean Distance
- ME – Misclassification Error

A.2 Variable Order Conditional Heavy Hitters

We use the following VOCHH-related notations:

- Training dataset consists of n strings (or trajectories) of total length N;

- D – the maximum order of model, which shows the maximum length of dependency among the neighboring observations. The order of the model corresponds to the length of the context or a parent that is used for the modeling of time correlations;
- m – the length of a query string;
- z – the number of occurrences of this string in a tree,
- r – is the number of resulting LZ78 factors for the corresponding algorithm
- Σ – a finite alphabet, the letters from which (σ) are used as the values of symbols for the strings
- y - a parent or a context sequence
- y^k – is a parent or a context of a parent-child pair, that have a length k, thus, consists of k symbols
- (y, x) – is a parent-child pair, where a parent may consist of several symbols, while a child is usually one symbol
- $Fr(y, x)$ – frequency of the context y followed by a symbol x;
- $Pr[\cdot]$ – probability of an event.
- y' – the longest suffix of y: if $y = (x_1, x_2, ..., x_k)$, then, its longest suffix $y' = (x_2, x_3..., x_k)$
- ϵ, δ – the parameters of PAC learning guarantees for Probabilistic Suffix Trees algorithm
- ε a small value close to 1, used for the meaningfulness definition
- p_{min} – minimal probability that is defined to cover zero-frequency problem in Probabilistic Suffix Trees
- T – a trie of the PPM method

We use the following VOCHH-related abbreviations:

- VOCHH – Variable Order Conditional Heavy Hitters
- VMM – Variable order Markov Model
- PST – Probabilistic Suffix Trees
- PPM – Prediction by Partial Match
- PAC – Probably Approximately Correct
- HMM – Hidden Markov Model
- CRF – Conditional Random Field

A.3 Sigmoid Rule Framework

We use the following SRF-related notations:

- $f(Z)$ – characteristic transfer function or sigmoid function with the parameters m, M, b, c, d
- Z – signal to noise ratio, $Z = log(1 + S) - log(1 + N)$
  - S – the amount of signal or true data
  - N – amount of noisy or distorted data
- SRF directions:
  - m – minimal performance
  - M – maximal performance
  - r_{alg} – the width of the performance range
  - c – slope indicator
  - d_{alg} – the width of the active area of a classifier
  - $\frac{r_{alg}}{d_{alg}}$ – learning performance improvement over signal-to-noise ratio change
- Z_{inf} – the point of inflection
- $Z^{(3)}_{1,2}$ – zeros of the third order derivative
- $[Z_*, Z^*]$ – active noise range
- l_n - noise level
- Dataset characteristics for the non-sequential case:
  - x_1 – number of classes
  - x_2 – number of attributes (features)
  - x_3 – number of instances
  - x_4 – intrinsic dimensionality
- Dataset characteristics for the sequential case:
  - x_1 – number of classes
  - x_2 – number of instances
  - x_3 – the autocorrelation lag
  - x_4 – number of attributes (features)

- α – significance level for hypothesis testing

We use the following SRF-related abbreviations:

- SRF – Sigmoid Rule Framework
- PAC – Probably Approximately Correct
- NFL theorems – No-Free-Lunch theorems
- HMM – Hidden Markov Model
- CRF – Conditional Random Field
- RNN – Recursive Neural Network
- CTF – Characteristic Transfer Function
- IBk K-nearest neighbor classifier
- SMO Sequential Minimal Optimization, the approximate variation of support vector machines
- NbTree a decision tree with Naïve Bayes classifiers at the leaves
- JRip a RIPPER rule learner
- ACF – autocorrelation function

Appendix B

The Genetic Algorithm Settings

We used the JGAP[1] genetic algorithms package for the search in the sigmoid parameter space and Java Statistical Classes library for the statistical measurement the Kolmogorov-Smirnov test[2]. Default operators for double numbers were used. The alleles were five parameters, one per parameter:

- m with allowed values in $[0.0, 0.5]$.
- M with allowed values in $[0.5, 1.0]$.
- b with allowed values in $[0.0, 50.0]$.
- c with allowed values in $[0.0, 50.0]$.
- d with allowed values in $[-5.0, 5.0]$.

Essentially, employing the genetic algorithm, we try to maximize the following quantity:

$$fitness(i) = 100 * \left(1 + \frac{1}{D+1}\right),$$

where i is a candidate individual in the genetic algorithm, corresponding to a given set of parameter values and $fitness(i)$ the value of the fitness function for that individual. The population per iteration is 10000 individuals. Our search ends when there is no significant (that is $> 10^{-5}$) improvement after 20 consecutive iterations of the genetic algorithms, or when 1000 iterations have been completed.

[1] See http://jgap.sourceforge.net/.
[2] See http://www.jsc.nildram.co.uk/.