AzureML: Anatomy of a machine learning service

AzureML Team, Microsoft *
1 Memorial Drive, Cambridge, MA 02142

Editor: Louis Dorard, Mark D. Reid and Francisco J. Martin

Abstract

In this paper, we describe AzureML, a web service that provides a model authoring environment where data scientists can create machine learning models and publish them easily (http://www.azure.com/ml). In addition, AzureML provides several distinguishing features. These include: (a) collaboration, (b) versioning, (c) visual workflows, (d) external language support, (e) push-button operationalization, (f) monetization and (g) service tiers. We outline the system overview, design principles and lessons learned in building such a system.

Keywords: Machine learning, predictive API

1. Introduction

With the rise of big-data, machine learning has moved from being an academic interest to providing competitive advantage to data driven companies. Previously, building such predictive systems required deep expertise in machine learning as well as scalable software engineering. Recent trends have democratized machine learning and made building predictive applications far easier. Firstly, the emergence of machine learning libraries such as scikit-learn Pedregosa et al. (2011), WEKA Holmes et al. (1994), Vowpal Wabbit Langford et al. (2007) etc. and open source languages like R R Development Core Team (2011) and Julia Bezanson et al. (2012) have made building machine learning models easier even without machine learning expertise. Secondly, scalable data frameworks Zaharia et al. (2010); Low et al. (2010) have provided necessary infrastructure for handling large amounts of data. Operationalizing models built this way still remains a challenge. Many web services (e.g. `http://dataiku.com`, `http://bigml.com`, `http://cloud.google.com/prediction`, `http://datarobot.com` etc.) have attempted to address this problem providing turnkey solutions that enable software developers and data scientists to build predictive applications without requiring deep machine learning or distributed computing expertise. In this paper, we describe AzureML, a web service that provides a model authoring environment where data scientists can create machine learning models and publish them easily (`http://www.azure.com/ml`). In addition, AzureML provides several distinguishing features. These include:

1. Collaboration: AzureML provides the ability to share a workspace of related modeling workflows (called experiments) with other data scientists. Experiments can also be shared with the wider community through an experiment gallery. Experiments in

* Corresponding authors: sudarshan.raghunathan@microsoft.com, sharat@alum.mit.edu (formerly at Microsoft)

the gallery are indexed and searchable. Discovery is enabled by ranking experiments organically based on popularity and community rating.

2. Versioning: AzureML provides the ability to save the state of individual experiments. Each experiment has a version history. This enables rapid iteration without the overhead of tracking and maintaining individual experiments and different parameter settings.

3. Visual workflows: Data workflows can be authored and manipulated in a graphical environment that is more natural and intuitive than scripts. An experiment graph can be composed by joining modules (functional blocks that provide a way to package scripts, algorithms and data manipulation routines).

4. External language support: AzureML supports packaging scripts and algorithms in external languages such as python and R as modules. These modules can be inserted into any experiment graph along with built-in modules. This allows data scientists to leverage existing code along with built-in modules provided by AzureML.

5. Push button operationalization: Experiments containing models and data manipulation workflow can be operationalized into a web service with minimal user intervention. In contrast to other web-services, the prediction service can consist of data pre and post-processing. Operationally, each prediction corresponds to an experiment graph run in memory on a single instance or a groups of instances.

6. Monetization: AzureML provides a market place where authors of models can monetize their published data or prediction services. This opens up new opportunities for modelers in the prediction economy.

7. Service tiers: AzureML currently exposes two service tiers, paid and free. Anyone with a liveID can create a workspace and begin to author and run experiments. These experiments are limited in the size of data that can be manipulated, the size of experiments that can be created, and the running time of these experiments. In addition, free users cannot operationalize models to a production level of service. Paid users have no such limits. In the rest of this paper, we outline the system overview, design principles and lessons learned in building such a system.

2. System Overview

AzureML consists of several loosely coupled services (see Figure 2) which are described in the following.

2.1. Studio (UX)

This is the primary user interface layer that provides tools for authoring experiments Studio provides a palette of available modules, saved user assets (models, datasets) and provides a way to define and manipulate experiment graphs. Studio also provides UX necessary for uploading assets, sharing experiments and converting experiments to published web services. The primary document type within AzureML is an experiment. Operationally, an

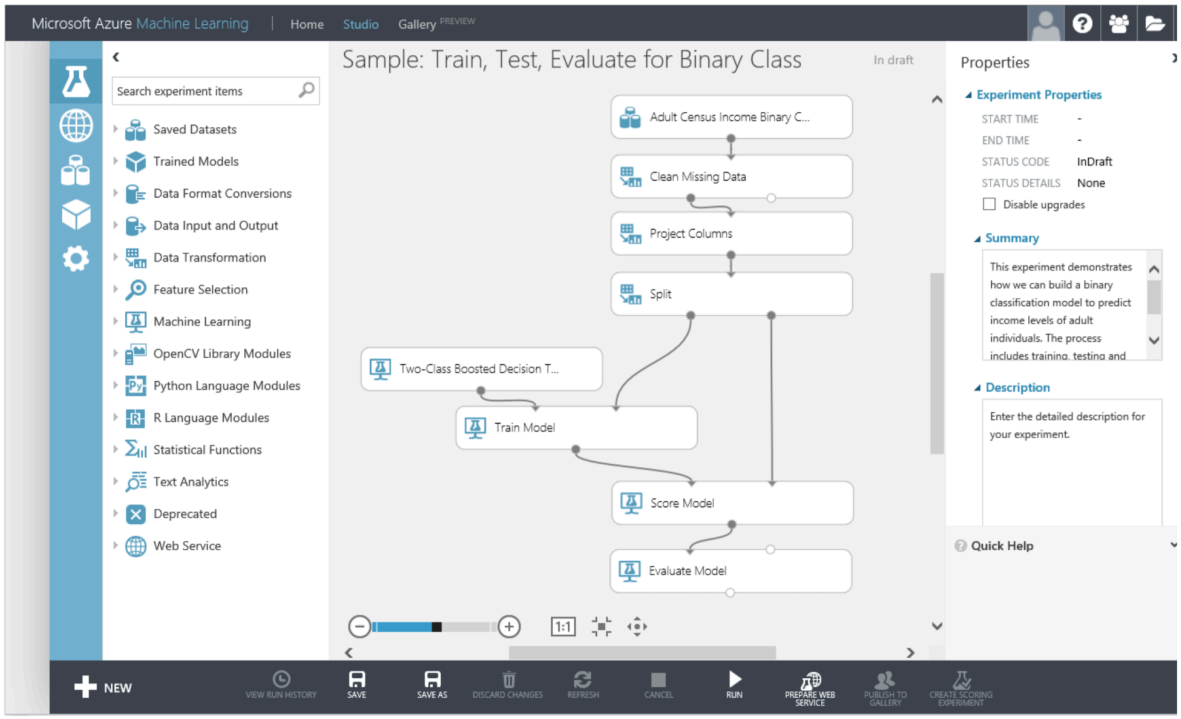

Figure 1: AzureML UX showing an example experimental graph and the module palette

experiment consists of a directed acyclic graph connecting several modules. Modules encapsulate data, machine learning algorithms, data transformation routines, saved models and user-defined code. Modules are divided into many categories such as machine learning, data manipulation, text analytics etc. Such modules can be referenced from multiple experiments and web services.

2.2. Experimentation Service (ES)

ES is the primary backend service that orchestrates interaction among all component services. ES is responsible for handling and dispatching UX events from Studio and communicating the results of experiments back to the user. Interaction between Studio and ES are defined by strong contracts allowing Studio and ES to be developed iteratively and independently. In addition, ES is the repository for all assets (datasets, models, transforms) and information about users and workspaces. Encapsulating all experiment related activities in a service that exposes an API allows programmatic creation, migration and manipulation of experiments. This allows other services to be built on top of AzureML. Internally, we use

this API for end to end tests, runners, and validation of individual modules. The API also enables AzureML datasets to be manipulated by external environments. A Python SDK is available where datasets can be imported (with an auto-generated code snippet), explored, and saved back to AzureML workspaces.

2.3. Job Execution Service (JES)

When an experiment is submitted for execution by the Studio, ES constructs a job consisting of individual module executions and sends it to JES. JES is the scheduler that dispatches these module executions to our execution cluster. Within a workspace, multiple experiments can be scheduled in parallel. Within each experiment, the experiment graph establishes dependencies among individual module invocations. Modules whose dependencies have been satisfied can be executed in parallel. The JES only schedules individual modules and does not execute the task itself. The individual tasks are queued and eventually gets executed on a worker node (SNR). The JES is responsible for tracking the execution of tasks and scheduling downstream modules when results of individual modules are available. In addition, JES enforces resource limits on executions. Each workspace can only run a certain number of jobs in parallel and each job has a limit on the number of tasks that can be executed concurrently. These limits are established to maintain fairness among the different users and also to differentiate among our different tiers of service.

2.4. Singe Node Runtime (SNR)

Tasks from JES are pushed on to a task queue from where SNRs pick them up for execution. Each module task can potentially get scheduled on a separate machine. Therefore the task and resources required for execution of a module are copied over locally to each SNR. Their stateless nature allows us to scale the number of SNRs based on the experiment workload. However, this design also adds some overhead associated with transporting data, dependent and assemblies to the SNR prior to task execution. This overhead can be amortized by careful JES scheduling where adjacent tasks on the graph get executed on the same SNR.

2.5. Request Response Service (RRS)

RRS is responsible for answering prediction requests for single input. It implements web services that are published based on experiments authored in Studio. A distinguishing feature of AzureML is that prediction/scoring workflow can be represented and manipulated as an experiment graph. The presence of web input/output entry points are the only differentiating factors between training and scoring workflows. This implies that that prediction may involve a subset or complete experiment graph including all data transformation modules. A training graph can be transformed to a scoring/prediction graph based using simple rules (which are in-fact automated). Some examples are: (a) replacing model training with a pre-trained and serialized model (b) bypassing data splits in the module (c) replacing input data sets with a web-input port etc. Another difference between training and scoring experiments is that within RRS, modules are executed in memory and on the same machine. Predictions can be made per instance or on a batch basis. When experiments are published to the RRS, test endpoints are created along with an autogenerated help page for the API. This help page describes the format of the requests that are accepted by the endpoint as well as

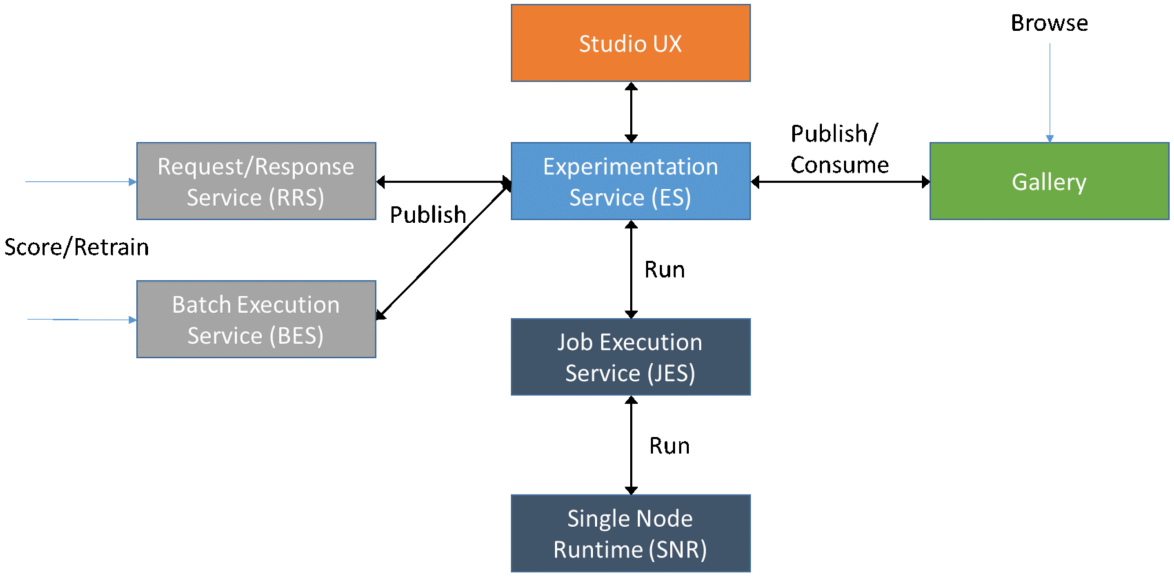

Figure 2: Component services of AzureML

sample code in C#, Python, and R for consuming the endpoints. Each RRS call is stateless and therefore can be replicated on multiple machines to scale with demand. We recently introduced a retraining API where customers can programmatically re-run their training experiments on new data, evaluate the resulting models, and update their web services with the new models if they are satisfactory.

2.6. Batch Execution Service (BES)

BES provides prediction services on larger datasets in contrast to RRS that provides prediction for single inputs. The execution of the graph within BES resembles that of a training workflow.

2.7. Community/Gallery

Authors of experiments can share their workspaces with individual users who can either edit or only view the experiments. In addition, AzureML also provides a gallery where

users can share experiments with all AzureML users. Gallery effectively provides a set of templates that can serve as starting point for experiments. Gallery also supports discovery of experiments through organic ranking as well as through a searchable index. Gallery is available to browse for free on the web and experiments saved there can be imported into any AzureML workspace.

2.8. Marketplace

Marketplace (http://datamarket.azure.com) allows users to monetize their published experiments or data. Currently, the marketplace supports web services that were generated using AzureML experiments as well as standalone web services authored externally. These end points can be integrated into regular experiments allowing composition of functionalities.

3. Life of an experiment

In the following, we describe the typical (classification/regression) workflow of an AzureML experiment.

3.1. Ingress

Data can be imported into AzureML in several different ways uploading CSV/TSV/ARFF files, referencing a URL, Azure blob storage, SQL server etc. During ingress, schema from these various formats are reconciled into an internal format (.dataset that will be discussed later). In cases where input schema does not specify the data type of individual columns (e.g. CSV), the types are guessed by looking at the composition of the data. AzureML also provides mechanism to explicitly change data types after ingress.

3.2. Data manipulation

AzureML provides several mechanisms for manipulating data prior to training/predictions.

- Data transformation modules: AzureMLs palette provides modules such as Clean Missing Values Join Columns, Project Columns, SQL transformation etc to allow manipulation of data.
- R/Python scripts: Arbitrary R and python scripts can be packaged as modules to transform the data. AzureML handles marshalling of data and schema.

3.3. Modeling

AzureML exposes both open-source as well algorithms developed at Microsoft to model data. Table 1 provides a list of algorithms that is continuously expanding.

3.4. Parameter tuning and evaluation

Practical machine learning involves some art, those of choosing the appropriate model, algorithm and finally their parameters. Some recent advances Bergstra et al. (2011); Snoek et al. (2012) have automated the task of model selection and parameter optimization. AzureML

Table 1: List of algorithms supported in AzureML

Binary classification	Average perceptron	Freund and Schapire (1999)
	Bayes point machine	Herbrich et al. (2001)
	Boosted decision tree	Burges (2010)
	Decision jungle	Shotton et al. (2013)
	Locally deep SVM	Jose and Goyal (2013)
	Logistic regression	Duda et al. (2000)
	Neural network	Bishop (1995)
	Online SVM	Sha (2011)
	Vowpal wabbit	Langford et al. (2007)
Multiclass classification	Decision Forest	Criminisi (2011)
	Decision jungle	Shotton et al. (2013)
	Multinomial regression	Andrew and Gao (2007)
	Neural network	Bishop (1995)
	One-vs-all	Rifkin (2002)
	Vowpal wabbit	Langford et al. (2007)
Regression	Bayesian linear regression	Herbrich et al. (2001)
	Boosted decision tree regression	Burges (2010)
	Linear regression (batch and online)	Bottou (2010)
	Decision forest regression	Criminisi (2011)
	Random forest based quantile regression	Criminisi (2011)
	Neural network	Bishop (1995)
	Ordinal regression	McCullagh (1988)
	Poisson regression	Nelder and Wedderburn (1972)
Recommendation	Matchbox recommender	Stern et al. (2009)
Clustering	K-means clustering	Jain (2010)
Anomaly detection	One class SVM	Schölkopf and Williamsonx
	PCA-based anomaly detection	Duda et al. (2000)
Feature selection	Filter based feature selection	Guyon et al. (2003)
	Permutation feature importance	Breiman (2001)
Topic modeling	Online LDA using Vowpal Wabbit	Hoffman et al. (2010)

provides a sampling based approach to optimization where a grid of parameters can be fully explored or sampled randomly to optimize metrics. Random sampling is typically more cost efficient than grid search in practice.

3.5. Operationalization

In AzureML, operationalization refers to the procedure of taking a trained model along with associated data transformations and converting it to a web service than can be queried for individual or bulk predictions. The primary challenge in operationalization is optimizing for latency and ability to handle large throughput. Some services export the learned model to a popular format (e.g. PMML Guazzelli et al. (2009)) and yield the choice of operationalization to the model owner. In contrast, AzureML manages the model for the user. The models are replicated based on the volume of prediction requests. Further, AzureML supports machine learning models that cannot be fully expressed in standards like PMML.

3.6. Modules

In AzureML, modules are the individual components in an experiment graph. All machine learning algorithms, data processing routines and custom code are packaged as modules. Each module may consist of one or more input and output ports. Each port has an assigned type (example dataset, ilearner, itransform etc.) which determines how modules are connected.

3.7. Graph Execution

The experiment graph is a directed acyclic graph that also defines the data dependencies between the modules. Modules that do not depend on each other can be executed in parallel. The job execution service (JES) is responsible for this functionality.

3.8. Schema validation

Datasets in AzureML are tagged with meta-data. In addition to column names and column types as in R and python, meta-data in AzureML attaches semantic meaning to individual columns. For instance, the meta-data information keeps track of columns that are used for training vs. those generated by predictions. This schema is propagated across the experiment graph even before the results are computed. This is essential in validating and early reporting of data compatibility. A module missing certain columns can report an error even before the upstream computation can take place.

3.9. External language support

AzureML support mixing of built-in modules as well as custom modules. Custom modules are user defined code that is packaged as a module. Currently, both R and Python language modules are supported. Data is marshalled in an out at the module boundary along with the associated metadata. Security is a primary consideration when allowing user code within a managed environment. Within AzureML, these modules are executed within a sandbox environment that limits system level operations.

3.10. Experiments as documents

AzureML in addition to being a modeling environment is foremost an authoring environment. Experiments are treated as assets of a data science process that need to be tracked, versioned and shared. Each run of an experiment is recorded and saved. Results from previous experiments are cached. If a section of the graph consists of deterministic modules, the modules are re-run only if upstream outputs are changed. Otherwise, cached results are returned immediately. Previous runs of an experiments are frozen can be accessed and copied over as new experiments.

4. Implementation details

4.1. Data types

AzureML experiments can be viewed as transformations on a restricted set of data types. Modules take as input and produce as output objects of the following types:

- DataSets: These are typically CSV, TSV, ARFF, and files in our own proprietary format
- Models: Trained and untrained models such as SVMs, Decision Trees, and Neural Networks.
- Transforms: Saved transformations that can be applied to data sets. Examples include feature normalization, PCA transforms etc.

Because the inputs and outputs to modules are typed, the UX can quickly determine the legality of connections between modules and disallow incompatible ones at experiment authoring time.

4.2. Data representation

After ingress, data is serialized in a proprietary .dataset format. The file is organized as schema, an index, and a linked list of 2D tiles, each tile containing a rectangular subset of data. This flexible representation allows us to decompose at both row and column level granularity. Wide slices are more suitable for streaming where as thin-and-long slices are better suited for compression and column manipulation. The tiled representation allows us to deliver good I/O performance across different access patterns streaming, batch loading into memory, and random access. Conversion modules to other formats (ARFF/CSV/TSV) are provided so that data can be egressed from our system in readable formats.

4.3. Modules

Modules are described by a formal interface what data types they accept and produce, and their parameters. Parameters also have types and are organized into parameter trees where certain parameters are only active if others are set to particular values. These descriptions are loaded into the Studio UX in order to present a rich authoring and viewing experience. Because we desire to support a multiplicity of data set formats (CSV/TSV/ARFF/.dataset and others in the future) we decided that module writers should not be responsible for I/O,

but rather should code against a data table interface. This interface, which includes access to both data and schema takes the place of datasets in our code. When modules are executed, the system converts the input file to an object implementing the data table interface and passes this object to the module code. When module is execution is finished, the system performs the reverse process it saves the object to a .dataset file and in addition creates separate schema and visualization files for consumption by the Studio UX.

4.4. External languages

AzureML supports a growing list of external languages including R and Python. We provide Execute R and Execute Python modules where users can enter scripts for manipulating data. Data internal to AzureML is marshalled as R and Pandas data-frames respectively. AzureML thus provides the ability to compose a graph consisting of built-in as well as user defined modules. In addition to supporting generic Execute Script modules for various external languages, users also have the ability to define their own module implementations in external languages. While the interface of, say, the Execute R module is static, a custom module allows for a writer to fully define an interface, generating a black-box implementation of routines in the external language of their choice. These custom modules interoperate with production modules and can be dragged in to experiments from the module palette like any other firstclass module.

4.5. Testing

All aspects of the product are extensively tested; ranging from usability sessions, mock security attacks and integration testing to an extensive automated test framework. The automated framework includes correctness testing for algorithms, fuzzing techniques to detect boundary cases, endto-end tests that ensure that the various system parts integrate well together, etc. These tests both gate source code changes as well as act to detect regressions post acceptance.

5. Lessons learned

Implementating a general machine learning web service required several design/performance tradeoffs with a bias towards usability. The subsequent use of the system provided us certain lessons outlined below.

5.1. Data wrangling support is useful

One of the key challenges in data science is data manipulation prior to building models. In fact, it seems to take more effort than building models (http://nyti.ms/1t8IzfE). This is reflected in our internal metrics that show that data-manipulation modules form majority of experiment graphs. AzureML provides a rich library of data manipulation modules including specific functionality (e.g. split, join etc.) and also very general modules such as (e.g. SQL transform).

5.2. Make big data possible and small data efficient

It is very tempting to adopt distributed frameworks prior to evaluating its tradeoffs. It has been shown that mapreduce style computing is inefficient for machine learning leading to in-memory solutions such as Spark Zaharia et al. (2010). A typical data science workflow involves rapid exploration and experimentation using a small data set in a development environment followed by productizing the module using large amounts of data. AzureML spans both ends of the size dimension. It makes big-data possible but small data efficient. This is a fluid position that will evolve over time.

5.3. Reproducibility is important, but expensive

AzureML provides the notion of deterministic and nondeterministic modules. Workflows containing only deterministic components is guaranteed to be reproducible across multiple runs. This immutability of module output allows us to cache intermediate results between experiment runs. Further, when executing an experiment graph, only modules downstream of a modified module need to be re-executed. Other results are available immediately due to guarantees of deterministic behavior. Caching allows a data-scientists to rapidly iteration on an experiment. From an implementation perspective this implies that intermediate results have to be serialized during graph execution. Further downstream modules cannot be scheduled until upstream results are serialized. This adds a large I/O overhead to the run-time of an experiment compared to a pipeline or deferred evaluation style execution (e.g. Apache Spark). The design assumes that the amortized cost over many variants will be minimized with checkpoints.

5.4. Feature gaps are inevitable but can be mitigated by user code

AzureML provides a rich set of algorithms and data transformation. The available functionality is expanded on a continuous basis. Users may be drawn to AzureML in order to utilize the authoring environment and versioning capabilities but may not find specific algorithms available in the palette. Because AzureML allows user to package custom code as modules, required functionality can be easily added while feature gaps are eventually closed by the AzureML. This allows users to leverage the rich set of libraries provided by R and Python along with the authoring functionalities provided by AzureML.

6. Conclusion

In this paper, we described AzureML, a web service that allows easy development of predictive models and APIs. Compared to other machine learning web services, AzureML provides several distinguishing features such as versioning, collaboration, easy operationalization and integration of user-code. The paper describes the design of the overall system as well as trade-offs that were made to implement this functionality. Finally, the paper describes some lessons learned that could prove useful for other providers of machine learning services.

References

Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127 (1):3–30, 2011.

Galen Andrew and Jianfeng Gao. Scalable training of L1 -regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning - ICML '07*, pages 33–40, 2007.

James Bergstra, Rémi Bardenet, Y Bengio, and B Kégl. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman. Julia: A Fast Dynamic Language for Technical Computing. *arXiv preprint*, pages 1–27, 2012. ISSN <null>.

C M Bishop. *Neural Networks for Pattern Recognition*, volume 92. 1995.

León Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. *Proceedings of COMPSTAT'2010*, pages 177–186, 2010.

L Breiman. Random forests. *Machine learning*, 2001. doi: 10.1023/A:1010933404324.

Christopher J C Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11:23–581, 2010. doi: 10.1111/j.1467-8535.2010.01085.x.

Antonio Criminisi. Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning, 2011.

R O Duda, P E Hart, and D G Stork. *Pattern Classification*. 2000. doi: 10.1038/npp.2011.9.

Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

Alex Guazzelli, Michael Zeller, Wc Lin, and G Williams. PMML: An open standard for sharing models. *The R Journal*, 2009.

I Guyon, I Guyon, A Elisseeff, and A Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

Ralf Herbrich, Thore Graepel, and Colin Campbell. Bayes Point Machines. *Journal of Machine Learning Research*, 1:245–279, 2001. doi: 10.1162/153244301753683717.

Matthew D Hoffman, David M Blei, and Francis Bach. Online Learning for Latent Dirichlet Allocation. *Advances in Neural Information Processing Systems*, 23:1–9, 2010.

Geoffrey Holmes, Andrew Donkin, and Ian H Witten. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994*, pages 357–361. IEEE, 1994.

Anil K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31 (8):651–666, 2010.

C Jose and P Goyal. Local deep kernel learning for efficient non-linear SVM prediction. *Proceedings of the 30th International Conference on Machine Learning*, 2013.

J Langford, L Li, and A Strehl. Vowpal wabbit online learning project, 2007.

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. GraphLab: A New Framework for Parallel Machine Learning. *The 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)*, pages 8–11, 2010. doi: 10.1.1.167.6156.

Peter McCullagh. Regression models for ordinal data. *Journal of the royal statistical society. Series B (Methodological)*, pages 109–142, 1988.

J A Nelder and R W M Wedderburn. Generalized Linear Models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3):370–384, 1972.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, and Others. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

R R Development Core Team. R: A Language and Environment for Statistical Computing, 2011.

Ryan Michael Rifkin. Everything Old Is New Again: A Fresh Look At Historical Approaches in Machine Learning. pages 1–221, 2002.

B Schölkopf and R Williamsonx. Support Vector Method for Novelty Detection. *alex.smola.org*.

Jamie Shotton, Toby Sharp, and Pushmeet Kohli. Decision Jungles: Compact and Rich Models for Classification. *Advances in Neural Information Processing Systems*, pages 1–9, 2013.

Jasper Snoek, Hugo Larochelle, and Ryan P. Rp Adams. Practical Bayesian Optimization of Machine Learning Algorithms. *arXiv preprint arXiv:1206.2944*, pages 1–12, 2012.

David Stern, Ralf Herbrich, and Thore Graepel. Matchbox : Large Scale Online Bayesian Recommendations. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*, pages 111–120, 2009.

Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark : Cluster Computing with Working Sets. In *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, page 10, 2010. doi: 10.1007/s00256-009-0861-0.