Scalable Machine Learning through Approximation and Distributed Computing

THEODOROS VASILOUDIS

Doctoral Thesis in Computer Science
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden 2019

School of Electrical Engineering
and Computer Science
KTH Royal Institute of Technology
SE-164 40 Kista
SWEDEN

TRITA-EECS-AVL-2019:43
ISBN: 978-91-7873-181-7

Akademisk avhandling som med tillstånd av Kungliga Tekniska Högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i datalogi på tisdagen den 28 maj 2019 kl. 14:00 i Sal B, Electrum, Kungliga Tekniska Högskolan, Kistagången 16, Kista.

Printed by Universitetsservice US-AB

To my parents

Abstract

Machine learning algorithms are now being deployed in practically all areas of our lives. Part of this success can be attributed to the ability to learn complex representations from massive datasets. However, computational speed increases have not kept up with the increase in the sizes of data we want to learn from, leading naturally to algorithms that need to be resource-efficient and parallel. As the proliferation of machine learning continues, the ability for algorithms to adapt to a changing environment and deal with uncertainty becomes increasingly important.

In this thesis we develop scalable machine learning algorithms, with a focus on efficient, online, and distributed computation. We make use of *approximations* to dramatically reduce the computational cost of exact algorithms, and develop *online learning* algorithms to deal with a constantly changing environment under a tight computational budget. We design *parallel and distributed* algorithms to ensure that our methods can scale to massive datasets.

We first propose a scalable algorithm for graph vertex similarity calculation and concept discovery. We demonstrate its applicability to multiple domains, including text, music, and images, and demonstrate its scalability by training on one of the largest text corpora available. Then, motivated by a real-world use case of predicting the session length in media streaming, we propose improvements to several aspects of learning with decision trees. We propose two algorithms to estimate the uncertainty in the predictions of online random forests. We show that our approach can achieve better accuracy than the state of the art while being an order of magnitude faster. We then propose a parallel and distributed online tree boosting algorithm that maintains the correctness guarantees of serial algorithms while providing an order of magnitude speedup on average. Finally, we propose an algorithm that allows for gradient boosted trees training to be distributed across both the data point and feature dimensions. We show that we can achieve communication savings of several orders of magnitude for sparse datasets, compared to existing approaches that can only distribute the computation across the data point dimension and use dense communication.

Sammanfattning

Algoritmer för maskininlärning används i allt högre grad i praktiskt taget alla delar av våra liv. En anledning till deras framgång är förmågan att lära sig komplexa representationer från enorma datamängder. Datorers beräkningshastighet ökar inte lika snabbt som de volymer av data vi vill lära oss från, vilket naturligt leder till algoritmer som måste vara resurseffektiva och parallella. I och med att användandet av maskininlärning fortsätter att breda ut sig blir förmågan att anpassa sig till en föränderlig värld och att hantera osäkerhet allt viktigare.

I denna avhandling utvecklar vi skalbara algoritmer för maskininlärning med ett fokus på effektiva distribuerade online-beräkningar. Vi använder oss av *approximationer* för att dramatiskt reducera beräkningskostnaden jämfört med exakta algoritmer, utvecklar algoritmer för *inkrementell inlärning* som hanterar föränderliga miljöer med begränsad beräkningsbudget, samt utvecklar parallella och distribuerade algoritmer som garanterar att våra algoritmer kan hantera massiva datamängder.

Avhandlingen börjar med att beskriva en skalbar algoritm för att beräkna likhet mellan noder i en graf och för att upptäcka begrepp. Vi visar dess användbarhet i flera olika domäner och dess skalbarhet genom att träna på en av de största tillgängliga textsamlingarna på ett fåtal minuter. Baserat på en praktisk tillämpning inom prediktion av längden på sessioner i media-strömning föreslår vi sedan ett flertal förbättringar på sätt att träna beslutsträd. Vi beskriver två algoritmer för att skatta osäkerheten för prediktioner gjorda av så kallade "online random forests". Vår metod uppvisar bättre träffsäkerhet än de bäst presterande metoderna inom forskningsfältet, samtidigt som den tar avsevärt mindre tid att köra.

Avhandlingen föreslår sedan en parallell och distribuerad algoritm för så kallad "online tree boosting" som ger samma garantier gällande korrekthet som seriella algoritmer samtidigt som den i genomsnitt är flera storleksordningar snabbare. Slutligen föreslår vi en algoritm som tillåter gradientbaserad träning av beslutsträd att distribueras över både datapunkts- och attributs-dimensioner. Vi visar att vi kan reducera mängden kommunikation avsevärt för glesa datamängder i jämförelse med existerande metoder som bara distribuerar beräkningarna längs datapunktsdimensionen och använder täta representationer av datamängder.

Acknowledgements

As is always the case in a dissertation, the results of this thesis is a culmination of work and support from multiple people to whom I am deeply grateful. My supervisors, colleagues, mentors and friends have all in their own way contributed to my journey to this point.

First, my academic supervisors, Anders Holst, Henrik Boström, and Seif Haridi. They have provided me with guidance and support throughout my thesis and for that I am grateful. My industrial supervisor, Daniel Gillblad, has been as good a lab manager and research colleague as one can hope. His door was always open and he's helped me grow as a researcher through our joint work and by letting me explore the industry through my internships. Olof Görnerup in many ways taught me how research is done when I was still starting out, with an eye for detail, creative ideas and an adherence to rigorous methods.

I want to make special mention of Gianmarco De Fransisci Morales, who guided me through, from a fledgling PhD student to an independent researcher. Gianmarco taught me how to insist on the highest standards, provided intuitive explanations for the most complicated topics, and showed me how one can start from an idea and transform it to a rigorous paper. This dissertation would not have been possible without his help.

I've been extremely lucky to work in the industry throughout my studies and work with people much smarter than me, that I've always learned from. Boxun Zhang at Spotify, Till Rohrmann at Data Artisans, Hossein Vahabi at Pandora Radio, and Thomas Drake and Hyunsu Cho at Amazon. I am deeply grateful for the opportunity to work at all these wonderful places and for the guidance everyone has provided. I've also had the chance make lasting friendships on my internships. Massimo, Sergio, Andreu, Charis, Nikos, Gianni, Kiki, Yannis, Martha, Kleopatra, thank you for making these times unforgettable.

Throughout my studies I've had the chance to talk and learn from some excellent colleagues here. Paris Carbone is always around for a coffee and discussions, be that streaming algorithms or the latest games. Vasia Kalavri who I'll always look up to for how to maintain a work-life balance while producing excellent work. Anis Nasir for excellent times at conferences and around Stockholm. Erik Ylipää for our many discussions about everything machine learning, couldn't have asked for a better office-mate. Bakr Karali for always finding humor about the life of a southerner in Sweden. Ian Marsh especially for all the pints, advice, and emotional support, always there when I needed someone to talk to. Martin Neumann without whom I would have never had this opportunity, I'm forever indebted and thankful for the good times over a board game.

Finally, I'm grateful for all my friends. Riccardo and Josh for PhD life chats and Liene for making sure we talk about other things too. Kehua, Rado, August, and Jacques, even though you guys left Stockholm I'm grateful for the good times

we had while we were all here. Angelina and Dimitris for unequivocal support of whatever I chose to do, can't tell you how important that is. Iraklis and Mitsos for always being there, and keeping me grounded to our roots.

Last but not least, my parents Vasilis and Erifyli, and my siblings Despina and Kostas. When everything else goes away, we still have each other.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER

1

Introduction

Machine learning (ML) is now being introduced into more and more areas in our lives. What once was mostly used in the realm of computer science for tasks such as digit recognition [152] is now being used in medicine [229, 80], energy [181, 219], agriculture [173, 160], climate science [172], structural engineering [83], and finance [64]. This explosive growth of machine learning has originated to a large part from the dramatic decrease in the cost of storing and processing data. The accuracy gains for machine learning models these days are made possible by this abundance of data in combination with sophisticated models.

However, "more" data means "more" computation which leads to the need for machine learning algorithms that can be trained *efficiently* on massive datasets. Efficiency can be interpreted in different ways. One aspect is the ability of an algorithm to take full advantage of modern hardware architectures. As CPU clock speed increases have stalled, developers and algorithm designers have turned to parallelism as a means of extracting more performance out of existing hardware. For gathering and processing the massive datasets available today, clusters of commodity machines have risen to be the dominant choice in the industry due to both locality in terms of taking the computation to an already distributed dataset, as well as cost-effectiveness. These distributed architectures introduce additional challenges on top of learning in parallel on a single machine.

Another efficiency factor is learning under a tight computational budget. Whether that is learning at the edge on internet of things devices, or federated learning with the intent of protecting user privacy, there has been an increasing demand for algorithms that can be trained with bounded computation and memory resources. These use-cases also highlight another challenge which is learning in a constantly evolving environment. Traditional batch models trained on historical data can perform sub-optimally as the relationships between the features and dependent evolve.

Many systems and algorithms have been developed in the past to tackle these issues. The popularity of deep learning [103] in particular has led to a deluge of research on efficient algorithms and systems. These include systems like Tensorflow [1], MXNet [56], and PyTorch [192] and have expanded to support heterogeneous hardware with recent projects like TVM [57] and Glow [208]. While these systems are optimized for the gradient descent optimization that neural networks are commonly trained with, this thesis is focused instead on two other models, *graphs and decision trees*.

The first problem we deal with originates from the data mining domain and tackles the problem of determining the similarity between nodes (vertices) in a graph, based on structural information, and uncovering semantic concepts from related nodes. For this purpose algorithms like SimRank [125] and its variants have been proposed, however due to their at least quadratic complexity in the number of nodes, their scalability suffers for graphs with billions of nodes. In our approach we instead approximate the problem by focusing our computation on local neighborhoods, dramatically reducing the computational cost to arrive at an approximate solution, and perform the secondary step to uncover semantic concepts from the similarity graphs.

The second model we focus on in this thesis are decision trees, both in the batch and online domain, which we motivate through our work on a real-world application for session length prediction in music streaming. In the batch domain we investigate gradient boosted trees, whose popularity has also led to the development of multiple scalable systems like XGBoost [55], LightGBM [135], and CatBoost [195]. What all these systems have in common is that they only allow scale-out across the data point dimension. This means that in order to scale learning for high-dimensional data we need to scale *up* to bigger, more expensive machines, a limitation which we address in our research. Apart from batch decision trees, online decision trees have been proposed to bring the accuracy and interpretability of decision trees to the online domain [75]. However, the concessions made to bring the learning algorithm to the online domain have led to decreased accuracy in practice and do not allow us to estimate the uncertainty in the predictions. In our research we tackle the accuracy problem for large-scale streaming data by developing efficient online boosted trees, and propose online algorithms to estimate the uncertainty in the predictions of tree ensembles.

These challenges set the stage for this thesis: We propose scalable algorithms for learning from massive data efficiently. We utilize different techniques to deal with the data size: 1) designing efficient parallel and distributed algorithms, 2) making use of approximations to reduce the computational cost, 3) using approximate data structures to bound the memory and computational cost of continuously updating the models. Throughout the development of this thesis the guiding principle has been to design algorithms that can scale regardless of data size. This is

realized through algorithms that exhibit linear scale-out characteristics, bounding the memory and computational costs through approximations, and designing algorithms that are optimized for distributed settings through communication-efficient learning.

Open Challenges in Scalable Machine Learning

The open challenges we tackle are:

- *Scalable vertex similarity and concept discovery*. All-to-all vertex similarity methods like SimRank [125] do not scale to massive graphs. Domain-specific alternatives, such as word2vec [178] for text do not provide a principled way to uncover concepts, relying on parametric clustering methods like k-means clustering to group related concepts. The similarity calculation for such embedding methods involves the calculation of a dot product for all pairs, making the process computationally costly.
- *Uncertainty estimation for online tree models*. As ML models are being deployed in areas where mistakes can be costly, e.g. in autonomous driving and finance, the ability to quantify the uncertainty in predictions becomes key. The constantly changing environment and real-time demands of such applications indicate that online learning methods can be of great use. Existing online learning methods that can provide uncertainty estimates, such as online quantile linear regression [142], do not provide the necessary accuracy and cannot deal with highly non-linear problems. Flexible models such as online decision trees do not currently provide a way to estimate the uncertainty in their predictions.
- *Scalable tree boosting in high dimensions.* Boosted decision trees are one of the most successful models in both industry and academia [55, 111, 159]. Their success can be attributed to their scalability, ease of use, and accuracy. However, the current state of the art in boosted trees has two open challenges in terms of their scalability. First, for online decision trees, all existing boosting algorithms are sequential, meaning that they cannot take advantage of modern parallel systems, severely limiting their applicability to large-scale streaming datasets. For batch decision trees on the other hand, current approaches can only parallelize their training along the data point dimension. For high-dimensional data scale-out is currently not possible, requiring bigger and more expensive hardware to speed up training per feature. In addition, current distributed approaches use dense communication that can be highly inefficient for sparse, high-dimensional data [55, 135].

Given these limitations of the state of the art, we formulate our research question and goals for this thesis.

1.1 Research Question & Objectives

The objective of this dissertation is to create scalable algorithms for machine learning, specifically for learning with graphs and decision trees. Our approach has been to use two aspects to ensure scalability: approximation from the algorithmic side, and parallelism from the system side. By approximation we mean either reducing an expensive problem to a more easily solved one that provides a useful answer, or by producing online models which can be trained on datasets of arbitrary size using bounded computation and memory. From the system side we develop algorithms that are parallel or are specifically optimized for the distributed setting. Our approach is to design solutions and provide implementations in popular open-source frameworks like Apache Spark [247], MOA [63], and XGBoost [55], ensuring the reproducibility of our work, and further contributing to the open-source community (see Section 1.3.1).

In order to create algorithms that scale to large-scale datasets and are able to do so efficiently in a distributed setting, we can identify three objectives that can lead to efficient solutions [131]:

1. **O1**: Reduce the amount of computation.
2. **O2**: Reduce the amount of communication.
3. **O3**: Bound the space cost.

Chapters 6, 8, and 9 are examples where approximation has been used to make a computationally expensive approach tractable, reducing the total amount of computation (**O1**). In Chapter 6 for example, we provide an approximate solution to the all-to-all graph node similarity problem by using the structure of the graph to limit the computational cost. While the produced result cannot give us a similarity score between any two arbitrary nodes in a graph, the result captures much of the relevant information and provides results that can be of use to practitioners.

Chapters 6 and 9 describe learning algorithms that are designed from the ground up with the distributed setting in mind. As network communication is a sparse resource in shared cluster environments [8], our algorithms minimize communication cost while providing valid solutions for the problems of graph node similarity calculation and boosted tree learning (**O2**).

Chapter 8 and Section 9.2 are examples of online learning models that were explicitly designed to use bounded compute and memory regardless of the dataset size (**O3**). For example, Chapter 8 provides online learning approaches for methods that previously only existed in a batch setting, where the data were assumed to be static and bounded, making it possible to train the algorithms on infinite streams of data. Approximations are made in this setting as well, as we replace exact data structures with approximate sketches, and trade-off consistency guarantees for the ability to train the models online.

1.1.1 Research Question

This thesis taken as a whole constitutes an effort of finding appropriate uses of approximation in order to make exact methods scale to datasets with up to billions of datapoints and millions of features. In all our work we take full advantage of current computing capabilities, whether that is providing parallel implementations of our algorithms, or by designing algorithms from the ground up to be efficient in a distributed setting.

The research question that summarizes this dissertation is:

> *How can we use approximations and parallelism to develop scalable learning algorithms?*

Based on this research question, we formulate our research statement as:

> *Approximations allow us to make otherwise computationally costly approaches scalable. By carefully designing our trade-offs, we can extract useful results using a fraction of the resources exact approaches require. These approximations combined with utilizing parallel and distributed computation lead to efficient, scalable solutions.*

1.2 Methodology

The research strategy for this dissertation is to perform quantitative, empirical research. Our approach has been to perform a literature review to identify the open challenges in the state-of-the-art, identify areas where approximation and parallelism can be applied to optimize performance, and proceed with the design and implementation of the algorithms.

The algorithm design is guided by the research objectives. Specifically, once we identify a problem to work on, e.g. uncertainty estimation in online decision trees, we use the three objectives (O1, O2, O3) as a guide for our optimizations. When exact solutions exist for a particular problem or for a different domain, e.g. a solution exists for batch learning but not for the online domain, we look to identify the approximations that will make the approach scale to massive data efficiently. In particular we identify different kinds of optimizations:

1. *Replacing exact data structures with approximate ones.* This approach can be taken to improve both the computational and memory cost of existing approaches. Depending on the level of accuracy necessary for a particular estimator, approximate data structures can often deliver similar accuracy to exact ones, at a fraction of the computational cost. One example in our research described in Chapter 8 is replacing a dense array of dependent values for the estimation

of quantiles at tree leaves with efficient quantile sketches [133], thereby bounding the memory cost of the algorithm.

2. *Using a proxy problem to tackle an intractable one.* Instead of trying to tackle an intractable problem directly, we can instead try to solve a tractable proxy problem that can inform us about the true solution, as we have done in Chapter 6. There we provide an approximate efficient algorithm for the node similarity problem by localizing the computation to two-hop neighborhoods, dramatically reducing the amount of computation necessary to produce an approximate but informative answer.

3. *Adapting the algorithm to the data to improve accuracy and efficiency.* In many problems we have prior information about the data properties. For example many real-world graphs exhibit power-law degree distribution [239, 4], many user interaction metrics are power-law distributed [216, 82, 17] or exhibit extreme sparsity [110]. These properties can be exploited to reduce the computational and memory cost of learning algorithms as well as improve their accuracy. In Chapter 6 we rely on the power-law degree distribution of many real-world graphs to provide approximations with measurable error, making an otherwise intractable computation efficient. In Chapter 7 we adapt the objective function of our predictive model to better fit the power-law distribution of the dependent. In Chapter 9 we make use of data sparsity to dramatically reduce the communication cost of distributed algorithms for gradient boosted trees.

1.2.1 Delimitations

The extent to which we try to answer the research question is of course limited by a number of factors. First, the two models we focus on are graphs and decision trees. We find graphs to be powerful data structures that can encode complex relationships and are therefore worthy of further study in terms of scalability. Decision trees have been shown to perform well for a wide variety of tasks [84]. They have distinct advantages like interpretability, although that is limited for large ensembles, and have the ability to deal with missing data in a principled manner, both important aspects in an industrial setting. Comparing decision trees with deep learning models, the space of architecture search in a deep network, i.e. designing a network that best fits a particular problem, is much larger compared to tuning hyperparameters for decision trees.

Second, although we develop online learning methods that can deal with concept drift, i.e. the relationship between the features and dependent can change over time, we do not propose any new concept drift adaptation mechanism. Instead we rely on existing concept drift adaptation mechanisms provided by the underlying online learning algorithms in our ensembles.

Third, for our decision tree algorithms, our comparisons focus on competing decision tree approaches. For example, for the online uncertainty estimation problem in Chapter 8, online linear models can also be used to determine prediction intervals. In our evaluation we focus on the state of the art decision tree algorithms as most real-world problems are highly non-linear.

Finally, as we mentioned previously, our evaluation is experimental and focuses on empirical results drawn from quantitative and qualitative experiments. For example in Chapter 8 we trade-off theoretical consistency guarantees for efficient computation and only demonstrate empirically that the algorithms are able to maintain validity.

1.2.2 Methodological Challenges

During the development of this thesis we have encountered several methodological challenges where a choice had to be made that delineates the extent to which our research tackles the original research question. We list these challenges here in order to make clear the limitations of this dissertation.

1. *Evaluation of unsupervised models.* In Chapter 6 we present our work on graph vertex similarity and uncovering concepts. The main methodological challenge for this work is the evaluation of the model's quality, as the vertex similarity task is unsupervised, because establishing the ground truth similarity would require extensive user studies, and it is impossible for the scale of the problems we are trying to tackle [125]. Instead, to provide a quantitative evaluation of the algorithm we need to rely on proxy tasks, such as determining the similarity of words using the Wordsim-353 dataset [85] and our model's agreement with other established similarity models like GloVe [194].

2. *Fair comparison of learning systems.* As our research has focused on optimizations, a major factor in evaluating the performance of an approach is the execution time. One challenge in assuring the fair comparison of our methods against existing algorithms and systems is to isolate the source of any potential gains in performance. For instance, in Chapter 8 we present a comparison of our approach with the current state-of-the-art algorithm, Mondrian Forests. Due to the complexity of Mondrian Forests we did not re-implement them from scratch in the same framework we used to develop our algorithm, introducing possible uncertainties in the runtime comparison. However, we make sure to demonstrate through the theoretical computational cost of the algorithm that our method scales much better as the number of instances grow. In subsequent work presented in Section 9.3 we implemented every algorithm from scratch to eliminate such discrepancies.

3. *Reliance on underlying frameworks.* To ease development and ensure our work can have a wider impact we have used established ML frameworks to develop our work. While using such frameworks can speed up development, it also introduces possible inefficiencies in our learning pipeline [174]. For example, our use of Apache Spark [247] in Chapter 6 required configuration and tuning of system parameters to achieve high performance. Similarly, the use of MOA for Chapter 8 and SAMOA for Section 9.2 introduces possible inefficiencies as these frameworks are meant to provide generic algorithm development platforms, and therefore lack optimizations for specific algorithms. Implementing our algorithms from the ground up in a low-level language would potentially bring further runtime optimizations, but would introduce many sources of error and slow down development, especially for error-prone distributed algorithms.

1.3 Contributions

The main contributions of this dissertation, along with the papers they appear in, are listed below:

- **Paper I**: *Knowing an object by the company it keeps: A domain-agnostic scheme for similarity discovery.* Olof Görnerup, Daniel Gillblad, and Theodore Vasiloudis. 2015. *Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM '15)*, pages 121-130, 2015.

  Paper II: *Domain-Agnostic Discovery of Similarities and Concepts at Scale.* Olof Görnerup, Daniel Gillblad, and Theodore Vasiloudis. In *Knowledge and Information Systems 51(2)*, pages 531–560, 2017.

  In these works we deal with the following research questions:
  How to create a scalable way to calculate the similarity between nodes in a graph? How can we uncover semantic concepts from the resulting graphs?

  All-to-all graph node similarity approaches, like the established SimRank algorithm [125], cannot scale to massive graphs due to the computational cost scaling with at least a quadratic factor in the number of nodes. Our approach is to approximate the all-to-all problem by limiting the similarity calculation to two-hop neighbors, following the notion of context similarity as stated by Firth in [86] as "You shall know a word by the company it keeps". By limiting the amount of computation we introduce approximations with controllable error allowing for similarity calculations on massive data. Our implementation is optimized for the distributed setting, with its most expensive operation being a *self-join* operation that has linear speed-up and constant scale-up, with limited communication.

Contribution: The author of this dissertation actively contributed in the design of this work, designed and implemented the scalable algorithm, and contributed in the paper writing and experiments.

- **Paper III**: *Predicting session length in media streaming*. Theodore Vasiloudis, Hossein Vahabi, Ross Kravitz, and Valery Rashkov. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*, pages 977–980, 2017.

  In this work we deal with the following research questions:
  What are the characteristics of session length distributions in media streaming?
  Can we use that information to effectively predict the amount of time a user will spend using a music streaming application at the moment they start it?

  While session length distribution has been investigated for search queries and post-ad click behavior, the behavior of users in a media streaming service is likely to differ greatly. In this work we first provide an analysis of the session length distribution of a major online music streaming service using tools from survival analysis. We then develop an appropriate model to predict session length from a number of features including user-based and contextual, session-based features. We demonstrate the differences in the way that user sessions evolve and end, and illustrate the importance of selecting an appropriate objective function for a non-negative, power-law distributed dependent value. This work acts as a use-case for our follow-up work, as it motivates the use of uncertainty estimation, online learning, and large-scale distributed learning with gradient boosted trees, topics we subsequently worked on in Papers IV, V, and VI respectively.

  Contribution: The author of this dissertation designed and implemented the work presented the paper, performed the experiments, and contributed most of the text plus all visualizations.

- **Paper IV**: *Quantifying Uncertainty in Online Regression Forests*. Theodore Vasiloudis, Gianmarco De Francisci Morales, and Henrik Boström. *Under Review*.

  In this work we deal with the following research question: *How can we estimate the uncertainty in the predictions of online tree ensemble methods?*

  Uncertainty estimation is of paramount importance when applying learning methods to domains where mistakes can be costly, like finance and autonomous vehicles. In addition, in these domains we have a constantly changing environment, and learning algorithms need to be able to constantly adapt. While ensembles of decision trees have been shown to be accurate estimators [84], their online counterparts lack any way to estimate the uncertainty in their predictions. In this paper we develop two general algorithms

for uncertainty estimation from online random forest ensembles, adapting batch methods to the online domain through approximate computation.

Contribution: The author of this dissertation designed and implemented the work presented the paper, performed the experiments, and contributed most of the text plus all visualizations.

- **Paper V**: *BoostVHT: Boosting Distributed Streaming Decision Trees.* Theodore Vasiloudis, Foteini Beligianni, and Gianmarco De Francisci Morales. 2017. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17)*, pages 899–908, 2017.

  In this work we deal with the following research question: *How can we make use of modern hardware to parallelize and distribute the computation of otherwise serial online boosting algorithms?*

  While online decision trees allow us to maintain an up-to-date scalable model using bounded memory and computation, the approximations they make can lead to decreased accuracy. Boosting is one of the most successful ensemble techniques to increase the accuracy of weak learners. However, existing online boosting approaches are strictly sequential, making parallelization challenging, while existing batch parallel boosting algorithms require the data points to be processed simultaneously, breaking the assumptions of the online algorithms. In this paper we bridge this disconnect between online and parallel boosting, by introducing online distributed boosting. We parallelize learning across the features while keeping the algorithm sequential, thereby maintaining the guarantees of online boosting algorithms, while at the same time providing significant speedups.

  Contribution: The author of this dissertation designed and implemented the work presented the paper, contributed to the experimentation, and contributed the majority of the text.

- **Paper VI**: *Block-distributed Gradient Boosted Trees*. Theodore Vasiloudis, Hyunsu Cho, and Henrik Boström. To appear in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19)*, 2019.

  In this work we deal with the following research questions: *How can we improve the scalability of distributed gradient boosted tree learning in high dimensions? Can we add another scale-out dimension while keeping the communication costs manageable?*

  Gradient boosted decision trees (GBT) are one of the most popular algorithms in use in industry and research alike. Their popularity stems from their ability to deal with massive datasets, and several systems for distributed learning of GBTs have been developed. However one common aspect of these systems is

Publication	O1 (Computation)	O2 (Communication)	O3 (Memory)
Paper I	✓	✓	-
Paper II	✓	✓	-
Paper III	-	-	-
Paper V	-	✓	✓
Paper IV	✓	-	✓
Paper VI	-	✓	✓

Table 1.1: Contributions by publication.

that they only enable scale-out across the data point dimension, meaning that in order to speed-up learning per-feature we need to scale up using bigger and more expensive machines. In this work we demonstrate how to enable scale-out across both the data points *and* feature dimensions, thereby allowing for scale-out across both, and by taking advantage of the structure of sparse datasets, enable faster and more cost-efficient training.

Contribution: The author of this dissertation designed and implemented the work presented the paper, performed all the experiments, and contributed most of the text and all visualizations.

An illustration of how each of the papers listed above covers the objectives listed in Section 1.1 is given in Table 1.1. We note that Paper III is an exploratory study that acts as motivation to our follow-up work.

1.3.1 Software Contributions

As part of this dissertation we have contributed to the following open-source software:

- XGBoost [55] is the most popular gradient boosted trees library, used across academia and industry. As part of our work we have used XGBoost as the learning tool for Paper III, and have worked directly to extend XGBoost to support block-distribution for Paper VI. As both of these works were performed during internships in private companies (Pandora Media and Amazon AWS respectively), we have not been able to release the code as open-source at the time of this writing. However, as a result of this work we have become actively involved in the wider development efforts of XGBoost and plan to integrate our block-distributed work in the future.

- MOA [24] is one of the most popular online learning library and includes support for training, evaluation and data generation for online/streaming

learning. We have used MOA as a baseline comparison for Paper V and extended it with support for uncertainty estimation among other improvements in Paper IV, which we plan to contribute back to the project.

- Apache SAMOA [63] is a library and framework that enables the development of distributed online learning algorithms, that can then be run on top of multiple stream processing engines like Apache Flink [49] and Apache Storm [230]. Our work on Paper V has been developed on top of SAMOA and has been contributed back to the project.

Other software developed during our PhD studies that is not included in this thesis, is the distributed machine learning library for Apache Flink, FlinkML, which is the official ML library for Apache Flink since the 0.9 release[1].

1.4 Thesis Organization

The rest of this dissertation is organized as follows: In Part I of the dissertation we provide background information and related work relevant to the topics covered. Chapter 2 introduces some basic building blocks for parallel and distributed machine learning, Chapter 3 describes the problem of graph node similarity and existing approaches, Chapter 4 acts as a brief introduction in decision tree learning in the context of this dissertation, and we end the first Part with Chapter 5 that presents an overview of online learning.

Part II presents the results of the work developed for this dissertation. We have tried to make each chapter self-sufficient, providing brief explanations of the approaches along with the most significant results. Our work on graph node similarity from Papers I and II is presented in Chapter 6, the motivating example on media streaming session length prediction from Paper III is presented in Chapter 7. The final two chapters of this Part focus on decision tree learning, with Paper IV on uncertainty estimation for online tree ensembles described in Chapter 8 and Papers V and VI being presented in Chapter 9, describing our work on distributed online boosted trees and block-distributed gradient boosted trees respectively.

We conclude in Part III with a conclusions and discussion chapter.

[1] https://flink.apache.org/news/2015/06/24/announcing-apache-flink-0.9.0-release.html

Part I

Background

CHAPTER

2

Parallel and Distributed Machine Learning

As the size of the datasets to learn from have grown, so has the need for methods that can scale and learn from massive data, with billions of data points and millions of features. With the slowdown of CPU frequency growth, the main avenue for better utilizing modern hardware is to make use of parallelism, whether that is in a single-worker, or distributed, making use of a cluster of computers to train models. In this chapter we will provide an introduction to the systems that make parallel learning possible and efficient. We start by presenting the different parallelization paradigms available in Section 2.1, continue by presenting the different methods of communicating in distributed machine learning in Section 2.2, and finish the chapter by highlighting some challenges in distributed learning in Section 2.3. For an in depth look into parallel and distributed Machine Learning (ML) system design we refer the reader to [243].

2.1 Parallelization Paradigms

In an abstract representation of any machine learning problem, we can separate two functional items: the data, and the model. The data refers to the set, or stream, of data from which we try to learn, be that a labeled dataset for supervised learning, or an unlabeled dataset from which we try to extract structure in unsupervised learning. The model refers to the approximate representation of the real-world process that we use to encode the information about the world, through the available data.

These two items provide us with a clean separation of how to perform parallel learning: we can either try to parallelize our learning over the set of data, known as *data-parallel* learning, or over different parts of our model, known as *model-*

parallel learning. Certain kinds of models may be more amenable to data or model parallelism. For example, training linear models using stochastic gradient descent lends itself well to data-parallel approaches [69], while random forest (RF) models allow for easy model-parallel learning, since every tree in an RF can be grown independently.

Finally, it is also possible to employ data and model parallelism simultaneously, often referred to as block-parallelism, as done by the Latent Dirichlet Allocation model of [242].

In our work we employ data-parallel learning for Papers I–III, model-parallel learning for Papers IV and V, and block-parallel learning for Paper VI. In the following sections we describe and provide examples of each learning approach.

Data-parallel Learning

Data-parallel learning is the most commonly applied form of parallelism in machine learning models due to the base assumption of many ML models that the training data are independent and identically distributed (i.i.d.) [243]. When performing data-parallel training, we distribute the data over a set of workers, each of which trains the model on its own partition of the data. At the end of each iteration there is commonly one communication step to synchronize the models of each worker to avoid diverging models, although this requirement can be relaxed, for example for asynchronous gradient descent algorithms like Hogwild [200] or through the use of stale-synchronous communication [114].

As Xing *et al.* [243] note, practically any ML algorithm that makes the i.i.d. data assumption and has an objective function that sums over the data indices, can be made into a data-parallel algorithm. As a result, almost every model available in ML now has multiple data-parallel versions available. Data-parallel optimization algorithms include stochastic and batch gradient descent [69, 255], L-BFGS [58, 3], ADMM [37, 249], and coordinate descent [202, 204]. Data-parallel models exist for deep learning [65], Latent Dirichlet Allocation (LDA) [221], gradient boosted trees [55, 135], collaborative filtering [143, 252], Support Vector Machines [105, 253, 227], and Gaussian processes [112, 68] among others, see [12] for a collection of approaches.

Model-parallel Learning

While developing data-parallel versions of existing algorithms is relatively straightforward due to the i.i.d. assumption, the same cannot be said for model-parallel algorithms. In this setting, we parallelize the problem by working on different parts of the *model* in parallel, rather than different partitions of data. The motivation is often high-dimensional data [38], or memory limitations. In order to learn complex representations, some models, like LDA and deep networks, require models with

up to trillions of parameters [246, 65] that cannot fit into the memory of one machine or a single Graphics Processing Unit (GPU).

The main challenge becomes that, unlike i.i.d. data, the parameters of most models are dependent on each other, often through an explicit structure like a neural network or Bayesian network, or a sequential dependence, for example in additive models like gradient boosting [110]. Care needs to be taken therefore to ensure that models are kept consistent and do not diverge due to the limited view of the model each worker possesses.

Commonly, the approach taken is to determine parts of the model that are indeed independent and can be optimized in parallel. For example, Bradley *et al.* [38] present a parallel Coordinate Descent algorithm that updates at each iteration a random number of coordinates in parallel. The authors prove that the number of coordinate updates that can occur in parallel without divergence can be derived from the data and is therefore highly problem specific. This approach of making use of the *structure* of the problem to avoid updates that interfere with each other, can also be applied to other models, see Lee *et al.* [154] for a principled approach to the problem. Modeling this structure explicitly, Graphlab [163, 164] presented a learning framework where models and data are represented in a graph-like structure that can reside in different workers and updated in parallel, requiring communication only as the graph structure indicates. This approach has the limitation however that the models need to be possible to be represented in a graph structure, something which is not possible for many categories of algorithms.

Finally, we mention approaches that are both data and model parallel. These are cases where models with trillions of parameters are needed to be trained over datasets with trillions of examples, for example in LDA or deep learning models [65, 246]. This kind of optimization requires a flexible communication paradigm that allows for efficient updates of parts of the model that may reside in different workers, from data that are also distributed. The Parameter Server architecture [158] allows for such flexible interactions and we describe it in the next section that focuses on the different communication patterns used in large-scale ML.

2.2 Distributed Communication in Machine Learning

Initial distributed ML approaches like parallel Stochastic Gradient Descent [255] eschewed communication until the very last step of the algorithm, however as Dekel *et al.* [69] show, this comes at a cost of the optimality of the solution. Dekel *et al.* instead suggest a method that communicates the gradients for each batch and uses their average to update the weights, leading to an asymptotically optimal algorithm for smooth convex loss functions. What this example demonstrates is the need for synchronization in ML models, which in a distributed setting means communicating model updates over the network. In this section we will examine

some of the approaches for distributed communication that have been employed in machine learning.

Efficient distributed communication has been a focus in the HPC and distributed systems communities since their inception, we refer the reader to the monographs [234, 225, 47, 231] for thorough introductions. In the context of machine learning, the focus has been on ways to synchronize sets of parameters, commonly represented as vectors or matrices, over a network of computers. The dominant paradigms have been the use of collective communications like *Allreduce* [228], the MapReduce programming paradigm [66], and the Parameter Server [158].

Allreduce is a high-level communication primitive provided by distributed communication systems like the Message Passing Interface (MPI) protocol [236] and the GPU-targeted NCCL[1]. At the highest level, an allreduce operation on a vector of objects will apply an aggregation function on the copy of the vector on each worker in a cluster and communicate the results between workers. At the end of the operation all workers end up with the same, aggregated, copy of the vector. Allreduce is a high level operation and as a result there have been many different implementations proposed, like the binary tree reduction algorithm, ring reduce, and butterfly reduce [228]. One of the main advantages of the allreduce operation is that it offers developers a simple "interface", as the developers only need to provide an aggregation function, and all workers share the same role. This simplifies the programming burden significantly, however limits the expressivity of the algorithm. For algorithms where the computational load cannot be easily divided between workers, for example the existence of high-frequency words in LDA [246], allreduce can result in imbalances between the workers, leading to increased runtime [109, 114].

Another important drawback of allreduce is that its implementations (like MPI) make use of dense communication. That is, it is currently not possible to utilize the sparsity of data when communicating, because the implementations require that the maximum byte size of the object to be known in advance, which is typically `number of elements` $\times$ `size of single element`. For sparse data where the number of non-zero values can be very small, this can lead to orders of magnitude higher communication cost than is necessary. We demonstrate this issue in the context of distributed Gradient Boosted Tree learning and provide a solution in Paper VI. We note that multiple research efforts exist to make sparse allreduce possible [203, 250, 116] but none that can be considered ready for production.

The MapReduce programming paradigm [66] offered some flexibility in terms of the programming model, in which the base primitives are a map operation that is applied locally at each worker on the input, and a reduce operation that aggregates the results from workers that correspond to the same key. Developers can assign custom partitioners to have more flexible communication, however the aggregation

[1] `https://github.com/NVIDIA/nccl`

pattern is still rigid. The use of disk-based aggregations for MapReduce are not a good fit for the iterative nature of most ML algorithms and often leads to increased runtime [254]. In-memory data processing systems based on MapReduce-like computation, like Apache Spark [247] suffer from similar issues [174], although systems like Apache Flink have proposed native iterations in a distributed stream processing system [81].

A more flexible paradigm is the Parameter Server (PS) [158, 157]. In the PS programming paradigm, machines assume the role of *workers* or *servers*. Workers are responsible for the computation of parameter changes, for example by iterating through their local partition of the data to collect gradient updates. Servers are responsible for the storage and update of the parameters. Similarly to a distributed key-value store [67], the workers in the PS programming model have two operations available, *push* that allows them to push a set of parameters to the servers, and *pull*, that retrieves a set of parameters. These push and pull operations take as input a *key* that transparently determines the server that is responsible for that set of parameters, thereby allowing one-to-one communication between servers and workers. However, worker to worker communication is not allowed in this paradigm. By allowing the parameters to be distributed over a cluster of computers, PS allows for intuitive implementation of model-parallel and block-parallel solutions as shown by Li *et al.* [158], Yuan *et al.* [246].

The flexibility of the PS comes at cost of more complex programming on the developer side, compared to an allreduce operation, as the developer needs to keep track of the keys that correspond to the parameters of interest they would like to update or retrieve, and do additional housekeeping for the aggregation functions, or broadcasting updates to all workers. The PS architecture lends itself very well to cases where updates to the model are local, i.e. they only touch small parts of the model. This ensures that the communication necessary for the model updates remains small.

2.3 Distributed Learning Challenges

The challenges in distributed learning stem from the need to synchronize the solutions of different workers after each iteration, as mentioned in Section 2.1. This model is commonly referred to as Bulk Synchronous Parallel (BSP) [114]. In a shared cluster environment a common issue arising is the existence of "stragglers", that is, workers that fall behind in their computation of a particular iteration of an algorithm, causing the whole system to stall as other workers have to wait for them to finish. Depending on the reliance of the model on a synchronized solution, this can have more or less adverse effects, see Harlap *et al.* [109] for background on the problem and proposed solutions.

The Hogwild optimization algorithm [200] and to a larger degree the stale-

synchronous parameter server (SSP) [114] were designed to counter-act the effect of stragglers. SSP allows for some slack between the iterations at which different workers can be. In an iterative learning algorithm, this would mean that as workers work through their partition of the data and move on to the next iteration, they are allowed to fall behind, with the constraint that the fastest worker is at most $\mathcal{S}$ iterations ahead of the slowest worker, where $\mathcal{S}$ is called the *staleness threshold*. Compared to asynchronous solutions like Hogwild, Ho *et al.* [114] are able to give stronger convergence guarantees for SSP and show how asynchronous algorithms can lead to slow convergence in the presence of stragglers.

Another consideration for the design of distributed learning systems is the limited network communication available, a problem attenuated in shared clusters, where many applications are competing for resources, and in cloud environments. Since network communication is necessary to synchronize the models, creating communication-efficient algorithms has attracted significant research. We refer the reader to Arjevani and Shamir [8] for an in-depth look at the communication costs of distributed optimization algorithms. Systems like the parameter server have been developed with the aim of reducing communication [158] at the systems level, while specialized algorithms like CoCoA [124] focus on local computation in order to reduce the amount of communication necessary. Other approaches include making use of information theory and codes to speed up learning [153] and gradient compression [161]. In our work in Papers V and VI, we take advantage of data sparsity to create efficient representations and reduce the communication cost of distributed boosted tree training by several orders of magnitude.

CHAPTER

3

Graph Vertex Similarity

The problem of computing the similarity between nodes in a graph has been well-studied under the context of link prediction [167], anomaly detection [190], recommendation [88], and information retrieval [180]. Therefore a number of algorithms have been proposed to tackle the issue. In this chapter we provide an introduction to the problem and highlight some of the established approaches as well as recent research focused on scalability. We divide the approaches into local and global based on their ability to determine the similarity of nodes based on local information or by looking into the entire graph, as proposed by Lü and Zhou [167]. In this chapter we focus on measures that use the structure of the graph to determine the similarity, however for specific applications there exist similarity measures based on content or curated databases such as utilizing the graph structure of the Wordnet databaset [193, 44, 85].

As the problem is closely related to link prediction, we point the interested reader to the survey by Lü and Zhou [167] or the more recent survey by Martínez *et al.* [170] on the subject, that include vertex similarity algorithms.

Preliminaries

We denote the graph $G(V, E)$ where V is the set of vertices and E the set of edges. When discussing computational cost, we denote the average degree of the graph as d. To ease notation we assume undirected graphs, however most of the methods mentioned here can apply to directed graphs as well. For a node $v \in V$, $N(v)$ denotes the set of nodes in v's *neighborhood*, that is, all the nodes that are reachable from v within a single hop. We denote the adjacency matrix of the graph by $\mathbf{A}$.

3.1 Local Similarity Measures

Local similarity measures generally determine the similarity between nodes by examining structural properties between nodes that are two hops apart in the graph. While this limits their ability to model long-range similarities, they have been shown to have accurate results, and are computationally efficient [167]. In the following section we present some of the most popular local similarity measures.

Common Neighbors. This method makes the assumption that nodes that share many common neighbors will be similar themselves. The complexity of the method is $O(|V|d^2)$. For two nodes u, v it is simply denoted as:

$$S_{CN} = |N(u) \cap N(v)| \tag{3.1}$$

Jaccard Coefficient. The Jaccard coefficient [123] measures the similarity between sets, defined as the size of the intersection of the two sets, divided by the size of their union. Its definition is the following, and has the same complexity as Common Neighbors:

$$S_{Jac} = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|} \tag{3.2}$$

Sørensen-Dice coefficient. The Sørensen-Dice measure [72, 223] is analogous to the Jaccard index through a monotonic transform and is defined as:

$$S_{SD} = \frac{2|N(u) \cap N(v)|}{d_u + d_v} \tag{3.3}$$

where d_u, d_v denote the degrees of nodes u and v respectively.

Adamic/Adar index. This measure relies on shared neighbors to determine the similarity of nodes, and was originally proposed for the task of link prediction [2]. It is defined as:

$$S_{AA} = \sum_{y \in N(u) \cap N(v)} \frac{1}{\log |N(y)|} \tag{3.4}$$

Resource Allocation. This measure was also developed for the link prediction task [251], and has a definition similar to the Adamic/Adar measure:

$$S_{RA} = \sum_{y \in N(u) \cap N(v)} \frac{1}{|N(y)|} \tag{3.5}$$

3.2 Global Similarity Measures

Global methods use information from the complete graph to determine the similarity between nodes that can be more than two hops apart. These methods are commonly

iterative and have a much higher computational cost than local measures, as they calculate all-to-all rather than local similarities. The general approach is to calculate the similarity in a recursive manner: Two nodes are considered similar if their neighbors are also similar. The algorithms listed below all use some variation of this principle.

Katz index. The Katz index [155] is based out the Katz centrality measure [134] that determines the centrality of a node based on the number of paths between two nodes. The measure can be defined as:

$$S_{Katz} = [I - \beta \mathbf{A}]^{-1} \tag{3.6}$$

where β is an attenuation (damping) factor, and $0 < \beta < 1$.

SimRank. In SimRank [125] vertices are considered to be similar if they are related to similar objects. It is defined in a recursive manner as:

$$S_{SR}(u, v) = C \cdot \frac{\sum_{y \in N(u)} \sum_{y' \in N(v)} S_{SR}(y, y')}{d_u \cdot d_v} \tag{3.7}$$

where C is a decay factor. Compared to the Katz measure described above, SimRank only includes paths of even length, which can affect the similarity calculation, and can result to invalid results for bipartite graphs [155]. SimRank is also very computationally intensive with cubic runtime cost with respect to the number of nodes. Optimized versions like [245] have been proposed to alleviate this drawback. The measure proposed in Blondel *et al.* [34] is also closely related to SimRank and has the same drawbacks.

Average Commute Time. This metric makes use of random walks to determine the average number of steps required to reach node v from node u and return. The assumption this method makes is that two nodes are more similar if they have a smaller average commute distance, and is defined in terms of elements l of the pseudoinverse of the graph Laplacian, L as defined in [167].

$$S_{ACT}(u, v) = \frac{1}{l^+_{u,u} + l^+_{v,v} - 2l^+_{uv}} \tag{3.8}$$

3.3 Applications

As a general graph problem the applications of node similarity are as varied as the problems that we can model using a graph representation. In this section we will describe a few applications that exemplify the importance of the problem.

One of the areas where node similarity has been most successful is link prediction. In the link prediction task, we try to predict missing edges in a graph, which might represent real connections that are encoded in the graph, or connections which are yet to happen in the real world. In the taxonomy of link prediction methods

proposed by Martínez *et al.* [170], node similarity is one of the four major categories of algorithms, along with probabilistic and statistical methods, algorithmic methods and pre-processing methods. Node similarity methods are used for link prediction by ranking node pairs by the similarity score given by the algorithm and the top-ranked pairs of nodes that are currently not connected are considered the most likely to form a connection. In the paper describing the Resource Allocation measure, Zhou *et al.* [251] consider the probability of a true missing link being given a higher score than a randomly chosen missing link to evaluate the accuracy of their method.

Node similarity has also been explored in the context of recommendation. The Average Commute Distance [89] metric mentioned above, uses the distance required by a random walker to move between two nodes as an indication of the similarity between nodes. The authors make use of this score to provide recommendations, either using a direct approach, where the similarity between users and items is calculated on the same graph, or indirectly, by first measuring the similarity between users in a graph, and then determining the preference of said user for an item given the preferences of their top-k neighbors for that particular item. Hang and Singh [108] use vertex similarity to recommend trustworthy nodes to a query node that belongs in the same trust network. The similarity is calculated this time between nodes in two graphs, the structure graph and the trust network, using again structural information between the graphs.

Vertex similarity methods have also been used in natural language processing, to determine the similarity between words and documents. Kandola *et al.* [132] make use of a diffusion process on a lexicon graph where nodes are terms in the text and edges represent co-occurrence. This allows them to discover semantic relationships between nodes that are not directly connected in the graph, but can however be semantically related. Ramage *et al.* [198] on the other hand calculate similarities using random walks between words in the Wordnet [179] graph to estimate their semantic similarity. A similar approach is taken by Alvarez and Lim [6], where the authors use the distance between the common ancestors of words in WordNet. More recently, Recski *et al.* [201] use a combination of the WordNet graph, embedding methods like GloVe [194] and a concept dictionary to determine the semantic similarity between words.

CHAPTER

4

Decision Trees

Decision trees is one of the most important and successful algorithms in machine learning. They have seen widespread adoption in practically every area of machine learning, from their original domain in classification and regression, to ranking [45], semi-supervised learning [136], survival analysis [122], quantile regression [175], and unsupervised learning [33]. In this chapter we will provide a brief review of the literature on decision trees in areas that are relevant to our work, starting with a short introduction in Section 4.1, and focusing on ensembles of decision trees in Section 4.2. For a comprehensive review of decision tree literature we refer the reader to the monograph by Rokach and Maimon [206] or the extended survey by Criminisi *et al.* [62].

4.1 Learning Algorithms

The two original tree-learning algorithms, the Classification and Regression Trees (CART) [42] algorithm and the ID3 algorithm [196], were developed in parallel. In our description we will focus on the CART regression algorithm, following the description from Hastie *et al.* [110].

Decision trees create recursive binary partitions of the input space, with axis parallel cuts. Let $R_1, ..., R_M$ denote the M regions created by the model. The model predicts the same value $\hat{c}_m$ for all data points that fall under the same region in the partitioned space. For regression, using the sum of squared errors as the objective function, the best possible $\hat{c}_m$ is the average of the label values in that region. Finding the optimal partitioning for decision trees is known to be NP-complete [119]. Consequently a common choice is to use a greedy algorithm to select the splits [147]. In the regression case, CART achieves that by selecting splits that minimize the squared error of the created binary split, selecting for each region the average value of the labels. In the classification case, the splitting

criterion is an information-theoretic node impurity measure like the Gini index or the cross-entropy, defined as:

$$\begin{aligned} \text{Gini Index} &= \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}), \\ \text{Cross-entropy} &= -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}, \end{aligned} \tag{4.1}$$

where k is the class, m the node and $\hat{p}_{mk}$ the proportion of the observations for class k in the node m.

While one could keep growing a decision tree until it perfectly fits the data, that will lead to overfitting and bad generalization. Therefore it is common to employ some form of *pruning* strategy to limit the complexity of the model. This can be done by introducing some form of complexity penalty to the tree, limiting the number of terminal nodes, or having a minimum number of examples in each terminal node [42]. This process is also important in limited memory settings where deleting and merging nodes can lead to models that take up less space in memory, especially in ensemble approaches mentioned in the next section. A creative approach to reducing model size are the so called *decision jungles* [220] that change the data structure being learned from a tree to a Directed Acyclical Graph (DAG), allowing nodes in the DAG to have more than one parent, providing the same learning capability as a tree, while using fewer nodes.

4.2 Ensembles of Decision Trees

The success and popularity of decision trees comes in large part from their use in model ensembles. The two dominant paradigms in ensemble learning are *bagging* [39] and *boosting* [214, 92]. Both methods can be interpreted using a resampling procedure. Bagging is based on the bootstrap resampling method [78] and re-uses samples to train the members of the ensemble, and average their predictions. Boosting methods on the other hand assign increased weights to the hardest instances in the data. In the "AdaBoost.M1" version of the algorithm, a new weak learner is trained at each iteration, and a weighted sum of the decisions of each learner is taken to produce the final prediction. Further work by Breiman [40] established a statistical framework for boosting and formalized *gradient boosting* as gradient descent in function space.

Decision trees have been used as parts of either bagging or boosting ensembles with great success [84, 55]. Their characteristics as a fast non-linear learner with high flexibility fit well both the bagging and boosting paradigms. The *random forest* algorithm uses bagging to train an ensemble of randomized trees and combine their

predictions, while gradient boosted trees uses a combination of trees and gradient boosting to produce accurate learners. In the following we give a brief overview for each method, and refer the reader to [20, 62] and [18] for recent surveys on random forests and gradient boosting respectively, or the introductory monograph by Hastie *et al.* [110].

4.2.1 Random Forests

Random Forests (RF), originally proposed by Breiman [41] based on the work of Amit and Geman [7], have been shown to be one of the most versatile algorithms in machine learning [84]. They are an ensemble method the combines multiple randomized trees, each trained with a sample of the original dataset, and averages their predictions to produce the final outcome. Random forests can deal with classification or regression tasks and provide estimates of the importance of variables, aiding in their interpretability.

Following the notation in Biau and Scornet [20], the algorithm works by growing M randomized trees, each on a sample a_n from the original dataset, with p features. The sample a_n can be taken with or without replacement, although usually it is a boostrap sample [78] with a size equal to the original dataset. In the original algorithm by Breiman, the trees are grown using the CART criteria, with the difference that they limit the number of features being considered for the splits, with Breiman selecting a either single feature or $\log_2 p + 1$ features, while Biau and Scornet recommend using $\lceil p/3 \rceil$; see [71] for an investigation of the RF parameter settings. For the regression task, the predictions of the forests are calculated by taking the average prediction of all trees, while for classification they take a majority vote among all trees.

The random sampling and independent growing of each tree provides two distinct advantages to the RF algorithm. First, because the trees are grown independently, unlike the boosted trees discussed in the next section, parallelizing their learning is trivial. In the parallel setting where we have access to the complete dataset, or if the complete dataset fits in memory, parallelization is as simple as training a different tree on each bootstrap sample. The situation changes however for distributed data because of the nature of the bootstrap: depending on the sample selection there might be the need for data points to be communicated over the network, with a resulting high communication cost. Chawla *et al.* [52] propose instead training the trees independently and merging their votes, or one can use of the "bag of little bootstraps" [141] to train the models, to avoid shuffling the complete dataset. Alternatively, we can create split histograms on data partitions and communicate those over the cluster to determine the best split for each leaf, similar to how trees are grown for distributed gradient boosted tree methods [55, 135], which we expand up in Section 4.2.2.

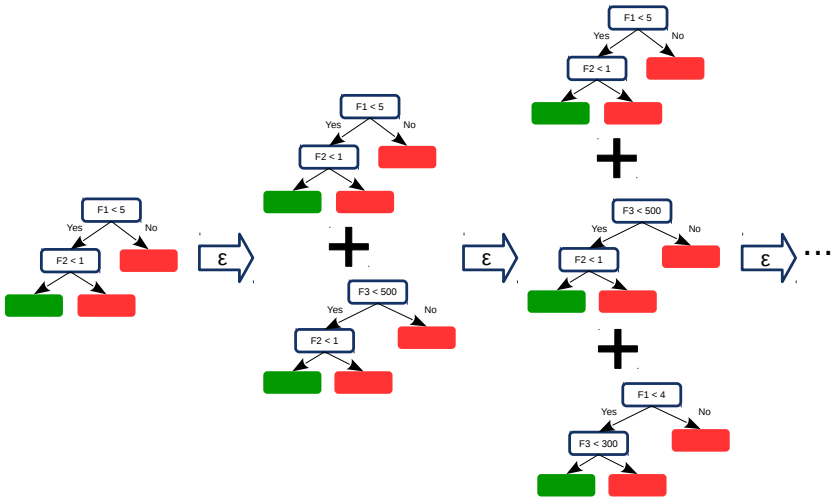


Figure 4.1: Example of growing a boosted tree ensemble. At each iteration we add a new tree, that is trained using the errors (gradients) of the previous ensemble.

Second, the fact that for each tree some samples from the dataset are not used to train it, known as *out-of-bag* (OoB) samples, creates opportunities to use those data points to extract useful information about the performance of the tree. We can use the OoB samples to adjust the hyper-parameters of the algorithm without the need to use a validation dataset. We can also use the out-of-bag sample to estimate *variable importance*. Breiman [41] proposed the Mean Decrease Accuracy measure of variable importance, where using the OoB instances for a tree, we randomly permute the values a feature, and make predictions for the permuted and original data. We take the average of the difference of the OoB error between the permuted and original data and use that as an estimate of the importance of the variable. Biau and Scornet [20] note that this approach has issues with correlated variables, because the algorithm tests variables in isolation rather than conditional to each other. Finally, Johansson *et al.* [128] make use of the OoB samples to provide uncertainty estimates in inductive conformal prediction, without the need for a validation set, a property we exploit in Paper IV as well.

4.2.2 Gradient Boosted Trees

Gradient boosted trees have risen to be one of the most popular algorithms in machine learning due to their speed, scalability, and accuracy. As a result, they are being used in mission-critical production use-cases such as Web search ranking

[159, 195] and click-through rate prediction [111]. Efficient implementations like XGBoost [55] and LightGBM [135] are now widely deployed and further emphasis is being given to efficient training [126] and inference [168, 226, 9]. In this section we will describe the training and prediction process of GBTs, with a focus on the scalability aspect of the algorithm. We provide an in-depth description of the histogram construction algorithm which we have found to be missing from most relevant literature. We refer the interested reader to a recent overview of the optimization process of GBTs by Biau and Cadre [18] for details on GBT training.

Model

The gradient boosted tree model consists of an ensemble of decision trees, which are trained in an additive manner. At each iteration a new tree is added to the ensemble, chosen so that it minimizes an objective function. Figure 4.1 shows the ensemble growing process at a high level. At each iteration we use the errors (gradients) of the previous step to train a new tree, and add it to the ensemble. This process is iterated for a pre-determined number of steps.

We use the regularized objective formulation given by Chen and Guestrin [55], and follow their notation throughout our description of the algorithm. The objective function is shown in Eq. 4.2:

$$\begin{aligned} \mathcal{L}(\phi) &= \sum_i l\left(\hat{y}_i, y_i\right) + \sum_k \Omega\left(f_k\right) \\ \text{where } \Omega\left(f_k\right) &= \gamma T + \frac{1}{2}\lambda \|w\|^2 \end{aligned} \tag{4.2}$$

where $l(\cdot,\cdot)$ is a differentiable loss function and Ω is a regularization term that penalizes complex models, with λ and γ being regularization factors. The regularization term takes into consideration the number of leaves T and the norm of the weights at the leaves w. This helps smooth the weights to avoid overfitting. This objective defined in XGBoost has then been used in follow-up GBT learning systems like LightGBM and DimBoost [135, 126]. A note here is that XGBoost assumes that the loss function is twice differentiable in order to calculate the Hessian analytically for each loss function, in order to reach the optimum quickly. However, as shown recently in [19, 165] it is possible to use Nesterov's first-order accelerated gradient descent method [184] to produce gradient boosting models of similar accuracy, utilizing far fewer trees (iterations), and hence speeding up training and inference. In distributed training this has the added benefit of reducing the amount of data being communicated by half, as we would only need to communicate first-order gradients and not Hessians, as we explain in the following section.

Index	Feature 1	Feature 2	Feature 3	Gradient
1	13	0	488	1.5
2	7	1	667	2.5
3	3.5	0	122	2
4	1.6	2	366	2.5

Table 4.1: Example dataset. Each color-coded data point has its feature values, along with a gradient value, for a squared error objective function, this would be the ensemble's residual for the data point.

Training

The training process for a single gradient boosted tree can be roughly divided into three stages: First, we use the existing ensemble to make predictions for every data point in the training set. Second, we use the approximated objective function to determine the first and second order gradients for each data point. Finally, we use the calculated gradients to greedily determine the optimal split for every leaf in the tree. When we mention gradients here we are referring to both the first order and second order gradients of the loss function for x_i, denoted as g_i and h_i respectively.

We will focus on the third part of the training stage in our description, as that entails the most computationally intensive part of GBT learning, the construction of the *gradient histograms*. Throughout our examples we'll be using the dataset from Table 4.1, which we have color-coded to correspond to the figures used later.

As a tree grows, we partition the complete training set into smaller parts, with each data point ending up in one leaf as we drop it down the tree. In order to determine the best split for every leaf, we need to iterate through the data partition of each leaf and create a gradient histogram for each feature for that leaf.

In its simplest form, a gradient histogram gives us the sum of gradients at a leaf, given a feature value, that is, $\sum_j (G_i | x_i(f) = f_j)$, where x_i is a data point that belongs to the current leaf, G_i is the gradient value of x_i and j is an indicator over the unique values of the feature f. In other words, it gives us the sum of the gradients of all data points in the leaf for a specific value of a given feature f.

Enumerating every gradient-feature value combination requires that we calculate a gradient sum for each unique value of a feature. This can quickly become computationally impractical for real-valued features, as we might have millions or billions of unique feature values. Systems like XGBoost and LightGBM instead use *quantized histograms*, where we accumulate gradients for ranges of values in histogram buckets for every feature.

To determine the bucket ranges we can use *quantile sketches* [106] that approximate the Cumulative Distribution Function (CDF) of each feature. We select a number of equi-weight buckets, that is, the ranges are selected such that every bucket

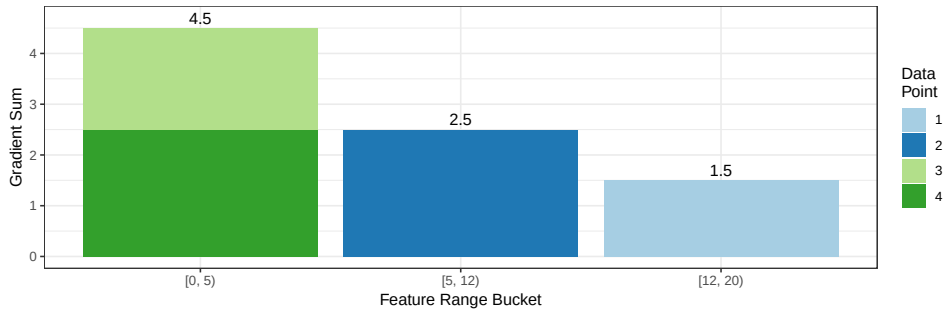


Figure 4.2: Gradient histogram for Feature 1 of the Table 4.1 data. The first bucket is the sum of the data points with index 3 and 4, since both their Feature 1 values fall in the $[0, 5)$ bucket.

has approximately the same weight-sum of data points. In the simplest case all instances have the same weight, and regular quantile sketch algorithms can be used, otherwise specialized, weighted sketches like the ones used by XGBoost have to be employed. The number of buckets to use is a parameter of the quantile sketch algorithm and depends on the type of sketch. There exist sketches where we select the number of buckets that will be used to represent the CDF, like the ones proposed by Ben-Haim and Yom-Tov [13]. Alternatively, the number of buckets is derived from the approximation error we choose for the sketches, for example for the sketches used by XGBoost or by the state-of-the-art KLL sketch [133] which we used in our own work in Papers IV and VI.

The selection of bucket ranges can be done either at the beginning of training, taking the overall feature CDF, or at every leaf taking the CDF of each feature using only the partition for that leaf. Chen and Guestrin [55] show that by selecting the buckets for every leaf separately, we can achieve the same level of accuracy, while using histograms with a higher approximation error, and hence reducing the memory footprint of the algorithm.

Once the bucket ranges have been determined for each feature, we use them to create the gradient histograms. After the prediction step, we have a gradient value for each data point, so we go through each feature and add the data point's gradient value to the gradient histogram bucket that corresponds to the feature value. In our example data in Table 4.1, for Feature 1 we use the buckets $[0, 5), (5, 12], (12, 20)$. Then, for the data point x_1 with a feature value $f_1 : 13$, we would add its gradient value to the the bucket $(12, 20]$ of the gradient histogram for f_1, and do this for every data point, adding the point's gradient to the corresponding bucket given the feature value. The gradient histogram for Feature 1 is shown in Figure 4.2. In the Figure we use color to identify the contribution of each data point to each bucket. When we are done iterating, each feature will have a corresponding gradient

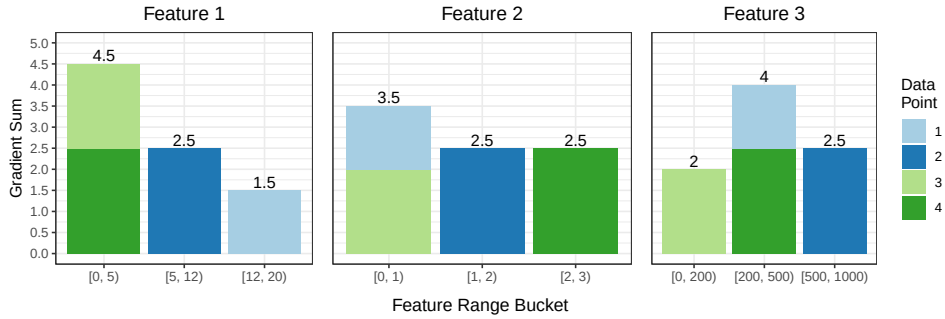


Figure 4.3: Gradient histograms for all features of the Table 4.1 data.

	Splitpoint 1	Splitpoint 2
Feature 1	**36**	21
Feature 2	35	30
Feature 3	26	30

Table 4.2: Potential gains for each possible split point, given the gradient histograms of Figure 4.3.

histogram, whose sum is equal to the sum of gradients for the leaf. For the data of Table 4.1, the complete gradient histograms are shown in Figure 4.3.

Once we have all the gradient histograms, we can use them to determine the optimal split point for the leaf, by calculating the potential gain of each feature and split point combination using the following equation [55]:

$$\mathcal{G}_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{4.3}$$

where I_L, I_R determine the data points that end up to the left and right side of the split respectively, I is the complete set of data for the leaf partition, and γ, λ are regularization parameters. At this point, to determine the best split we need to simply iterate through all the feature-splitpoint combinations and rank every candidate according to their loss reduction (or *gain*) and select the best one. In our example we have three buckets, and hence two possible split points for each feature. The result for our example data is shown in Table 4.2, where we have simplified the problem to only include the first order gradients. From that example we can see that the best feature-split combination would be choosing the first split point for the first feature (i.e. Feature $1 < 5$) as the split condition.

In the case of millions of features and a large bucket count B this can also be computationally heavy operation, as we need to evaluate $|F| \times B$ split candidates.

To mitigate this, we can apply efficient boosting approaches that try to reduce the number of features being examined. Examples include LazyBoost [79] that uniformly subsamples the set of features, bandit methods [46] that use information from previous iterations, or other adaptive methods like Laminating [76] that determines the best feature in multiple rounds, by incrementally halving the number of features being considered and doubling the number of examples being included in the gradient calculations.

CHAPTER

5

Online Learning

In this chapter we will provide an introduction to online learning, which is the learning setting for Papers IV and V. We start by providing a brief overview of the kinds of models available for online learning in Section 5.1. Since our work focuses on decision trees and ensemble methods we present an overview of the state of the art in Sections 5.2 and 5.3 respectively. We identify issues and solutions for the evaluation of online models in Section 5.4, and conclude the chapter with a look at the available open-source software for online learning in Section 5.5.

5.1 Models

As with batch learning, a wide range of models exists in the online learning domain. Commonly, these are models that were first developed for the batch setting and subsequently adapted to online learning. In this section we focus on supervised learning methods for classification and regression, and explore different learning representations to demonstrate the variety of models available for online learning, following the classification in [14]. For a comprehensive introduction to the various models available in streaming learning we refer the reader to [24]. An important aspect of online learning that we do not cover in this chapter is learning under concept drift, that is, when the underlying distribution of the data changes during learning. We refer the reader to the surveys [256, 95] for in-depth looks at the topic.

Instance Based Learning

One of the simplest and most intuitive methods in machine learning is Instance Based Learning (IBL), which usually takes the form of a nearest neighbor model. In Instance Based Learning, we make predictions based on the instances themselves, unlike model-based approaches that try to extract a model (function) from the

data. IBL exemplifies many characteristics that make it a good candidate for online learning, and the IBLStreams work of Shaker and Hüllermeier [217] demonstrates how these can be put to use to create an efficient online learning model. IBL algorithms are inherently incremental, and can be faster to update compared to a model-based approach, but they suffer from two distinct disadvantages: First, they need to store the data points themselves, which in a massive streaming data scenario can be infeasible. To deal with this issue IBLStreams will evict the oldest examples once an upper limit on the number of data points stored is reached. Second, inference can be much slower as it requires finding the k nearest neighbors to the incoming dataset, which even when using efficient indexing structures (like Locality Sensitive Hashing [121, 99]) can be very costly compared to a model-based prediction, e.g. from a linear model. Shaker and Hüllermeier suggest that IBL makes more sense in scenarios where updates are often but queries infrequent.

In terms of drift adaptation, IBLs can have the advantage that removing the effect of older data points is easy compared to say an ANN model, but it is still highly model-dependent. The authors suggest different ways to deal with concept drift in IBLStreams. They mention the limitations of a non-adaptive, window-based approach and suggest that IBL methods are good candidates for learning under drift because of their locality property, that is, that introducing a new example will only affect the predictions made around that region. In theory, an example should only be included in the model if it improves the accuracy, but that is impossible to know ahead of time. However, one can use heuristics to determine when to discard examples. These include the temporal and/or spatial relevance, or consistency of the examples to the current concept. In IBLStreams, the contents and size of the "case base", i.e. the examples that form the nearest neighbor representation, is updated automatically. IBLStreams tries to maintain a balance between maintaining few examples in the case base in order to be able to deal easier with concept drift, while keeping the set large enough so that predictions under a single concept are accurate. To that end, they use statistical change detection mechanisms and also provide ways to update the parameters of the algorithm, e.g. the number of nearest neighbors to examine, based on the observed error.

Linear Models

In terms of model-based approaches, linear models are popular for online learning due to their simplicity, efficiency and interpretability. For regression these models have the general form [110]:

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j \tag{5.1}$$

where p is the number of features in the dataset, where a linear relationship is assumed between the dependent and independent predictors.

One of the earliest examples of an online linear classification model is the seminal Perceptron algorithm [207] that tries to find a separating hyperplane between two classes based on the distance of the misclassified points to the decision boundary [110]. In the online setting, linear models are typically trained using a form of stochastic gradient descent, that moves the coefficients of the model according to the gradient of a single data point and a step size parameter. The Perceptron is a simple algorithm that however has many drawbacks noted by Hastie *et al.* [110], regarding the stability of the algorithm for perfectly separable data, its dependence on correctly setting the step size, and the fact that the algorithm does not converge when the data are not separable. Another important family of linear algorithms designed for the online domain are Passive-Aggressive (PA) algorithms [60]. These are margin-based algorithms for classification, regression and sequence prediction that solve a constrained optimization problem. For every incoming example, PA algorithms try to achieve a margin between the predicted value and true label by either not changing the model (passive) when the margin is satisfied, or by applying the necessary correction to the model coefficients to enforce a zero loss when the margin is not satisfied (aggressive). While being a linear model, PA algorithms can make use of the kernel trick to employ non linear predictors, as done by the seminal work on Support Vector Machines (SVM) by Vapnik [233] described next.

Support Vector Machines

Support Vector Machines, similar to the Perceptron, are algorithms for determining optimal separating hyperplanes, but unlike the Perceptron, they can deal with cases where the data are not linearly separable. They achieve that by using the "kernel trick" to transform the data and determining a linear boundary in the transformed space [110]. SVMs have also been extended to work in the online domain [139] where the main challenge is the need to maintain a set of support vectors in memory, as they grow linearly with the number of prediction errors [61]. The work of Crammer *et al.* [61] deals with this issue by providing an online learning algorithm that enforces sparsity through an insertion and deletion phase. Once an example with an erroneous prediction is inserted, the algorithm will look for past examples that are made redundant by the new point, and remove those to save memory. Another approach is taken by the Forgetron [70] which, for every mistake made by the algorithm, runs the standard Perceptron update, shrinks the support vector coefficients, and removes the support vectors with the smallest coefficients. The Pegasos algorithm [218] trains SVM models using Stochastic Gradient Descent, employing subgradients to deal with the non-linearity of the hinge loss. The Passive-Aggressive algorithm has also been modified to perform kernel learning

on a budget [238], i.e. with a bounded model size, by introducing an additional constraint to the original problem that removes one old support vector for every new one once the budget is met. One key drawback of these methods is the need to determine a budget for the model size beforehand. More recently, new methods have been proposed to scale up online kernel learning to massive datasets while maintaining a bounded model size, by transforming the feature space, and then performing linear learning [166] or by approximating new instances by "core points" scattered across the input domain [151].

Rule-based Learning

Decision rules [205] and decision trees [42] are two closely related learning methods that make predictions in terms of a set of *if-then* rules. Since online decision trees play a central role in our research we dedicate a separate section to review them (Section 5.2) and focus on online decision rules here. Decision rules use conjunctions of conditions on the attribute values to make predictions of the form `if <conditions> then <prediction>`. This structure makes them some of the more easily interpretable models available. In the classification task these are typically learned by maximizing the information gain of introducing a rule for a given outcome. Gama and Kosina [93] provided one of the first decision list learning algorithms aimed at the streaming domain, which updates its rules by maintaining an up-to-date set of sufficient statistics for each rule. Rules are expanded by selecting the condition, for example a threshold on a numerical feature, that minimizes the entropy of the class labels of the example that are covered by that rule [93]. The method uses the Hoeffding bound [115] to determine when it is time to update a rule, either by expanding an existing one or introducing a new rule. The process was later expanded to handle cases of concept drift [144] and regression [5].

5.2 Online Decision Trees

As we mentioned in Section 4.1, decision trees are trained by recursively splitting the complete training set as we introduce more leafs. This process is incompatible with the online learning scenario where data points arrive sequentially and we never have access to the complete dataset. To tackle such a scenario, a number of online decision tree alternatives have been proposed.

Hoeffding Trees

By far the most popular online decision tree algorithm for classification is the Hoeffding Tree (HT) [75], which has served as the basis for most of the follow up work on online decision trees. The aim of the HT is to create an online decision tree algorithm that will converge to its batch equivalent given enough samples. The

learning in an HT, or as Domingos and Hulten [75] name it, the Very Fast Decision Tree (VFDT), happens only at the leaves, and every data point is only utilized once, i.e. the HT is a single-pass algorithm. The leaves accumulate statistics with the purpose of probabilistically determining the best split. The Hoeffding tree got its name from the use of the Hoeffding bound to determine if we have accumulated enough information at a leaf to trigger a split with high confidence.

Following the description from Domingos and Hulten [75], the Hoeffding bound [115] is a probability inequality that allows us to probabilistically bound the true mean of a random variable r for which we have made n independent observations and computed their sample mean, $\bar{r}$. Given the range of the variable, R, the Hoeffding bound states that with probability $1-\delta$, the true mean of the variable r will be at least $\bar{r}-\epsilon$, where ϵ is defined as:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}. \tag{5.2}$$

In VFDT, the Hoeffding bound is used to determine with a high degree of certainty when it is time to split a leaf. Let $G(X_i)$ be the measure used to choose the feature to split on. As we mentioned in Section 4.1, this can be information theoretic measures like the cross-entropy or the Gini index. Ideally we want the feature we select given a limited sample of n examples, to be the same as would be chosen given infinite examples. The Hoeffding bound allows us to achieve that with high probability by applying it to determine the maximum possible difference between the best and second best features to split on. Specifically, let $\bar{G}(X_a)$ be the heuristic value for the current best feature, after having observed n instances, and $\bar{G}(X_b)$ the second best. The difference between these two is $\Delta\bar{G} = \bar{G}(X_a) - \bar{G}(X_b)$. We can then use the Hoeffding bound to determine that $\bar{G}(X_a)$ is indeed the best feature to split on with probability $1-\delta$, if $\Delta\bar{G} > \epsilon$.

The process of learning at each leaf is then the following: For every incoming sample that ends up in a leaf, we update the sufficient statistics for that leaf, which we use to calculate the heuristic for the split, e.g. the Gini index. Theoretically, one could check if it has gathered enough information to split a leaf after each incoming sample, but that would introduce a large computational overhead. What is often done instead, for example in the implementation of the algorithm in the MOA library [26], is to only check if the Hoeffding bound is satisfied periodically, for example every 200 samples. Another parameter of the algorithm is the probability of an error, δ. This parameter can have a large effect on the final tree, as setting it too high can lead to early splits being taken that end up being sub-optimal, and as a result, values in the order of 10^{-7} are often used [30].

HT was designed for the classification setting, where handling discrete attributes is straightforward: we can just keep a table of frequencies for classes and feature values. The situation is different for continuous attributes however, where we need to maintain per-class information about the values. In the worst case we would need

to maintain the class frequencies for every unique value in a continuous feature, something that is impractical for online learning, where datasets are potentially unbounded and the memory footprint of the algorithms should remain as small as possible. For features with many repeated values it is possible to use a binary tree structure with counters that allows for fast storage of the values, however this again has a large memory cost. Alternatives include using online histograms or quantile sketches [106] to maintain approximations of the Cumulative Distribution Function of each feature, or approximating the distribution of each continuous feature using a Gaussian. We refer the interested reader to Chapter 4 of Bifet and Kirkby [30] for details on these methods.

The Hoeffding Tree has served as the starting point for most of the follow up work on online decision trees. Jin and Agrawal [127] use a "Normal" test to improve upon the statistical efficiency of the Hoeffding bound. The proposed method achieves the same probabilistic bound with a reduced sample size, by taking advantage of properties of the Gini index and entropy. More recently, Manapragada *et al.* [169] propose an algorithm that uses the Hoeffding bound but will re-visit nodes that have already been split and evaluates if their split decision should be updated. This leads to a large improvement in accuracy compared to the base HT, at the cost of added computation and memory necessary to maintain the sufficient statistics for internal nodes and re-evaluate split decisions.

In a contrasting view, Rutkowski *et al.* [212] claim that the assumptions made by the Hoeffding bound-based algorithms are commonly violated as the bound assumes real-valued data, and the fact that measures like the Gini index and information gain cannot be expressed as sums of elements. As an alternative they propose using McDiarmid's inequality of which Hoeffding's inequality is a special case. Their use of the McDiarmid bound is however computationally intensive. This drawback is mitigated in their follow-up work [211] which follows in part the work from Jin and Agrawal [127] and provides a bound on the information gain difference between two potential split attributes based on the Gaussian distribution.

One of the few online decision tree algorithms that is aimed at regression is the Fast Incremental Model Tree (FIMT) algorithm by Ikonomovska *et al.* [120], also based on the HT. FIMT is a model tree, i.e. instead of simply using the average of the labels in its leaves to make predictions, it maintains a linear model, which is fit on the samples arriving at the leaf, and is then used for prediction. The merit of a split is calculated based on the Standard Deviation Reduction (SDR), similarly to batch regression trees like the M5 model [197], defined as:

$$\begin{aligned} \mathrm{SDR}(h_A) &= \mathrm{sd}(S) - \frac{|S_L|}{|S|}\mathrm{sd}(S_L) - \frac{|S_R|}{|S|}\mathrm{sd}(S_R), \\ \mathrm{sd}(S) &= \sqrt{\frac{1}{|S|}\left(\sum_{i=1}^{N} y_i^2 - \frac{1}{|S|}\left(\sum_{i=1}^{N} y_i\right)^2\right)} \end{aligned} \tag{5.3}$$

where h_A is the proposed split on feature A, and S_L, S_R the resulting data partitions on the left and right side of the split.

SDR is possible to be calculated online, by maintaining only three statistics per leaf, making the learning of these trees efficient. However, the base FIMT uses a binary tree to store feature values, leading to a potentially high memory cost, for which the authors provide solutions like disabling non-promising split points and dropping parts of the tree. To determine the probability of a split being optimal, the authors again use the Hoeffding bound on the SDR ratio of the two best splits which will be in $[0, 1]$ range. The linear models at the leaves are perceptron models, updated using stochastic gradient descent. Finally, FIMT includes a change detection system in order to adapt to concept drift based on the Page-Hinckley change detection test [188, 183].

Finally, we mention online tree ensemble methods, like the online random forests developed by Saffari *et al.* [213] and Gomes *et al.* [101], and the online boosting tree developed by Beygelzimer *et al.* [15] and Son *et al.* [222]. We focus on online ensemble methods in Section 5.3.

Mondrian Forests

All the methods we mentioned so far in this section have made use of the same base tree building algorithm that was first proposed for VFDT: each data point is only evaluated once, the statistics are maintained only at the leafs, with the exception of [169], and heuristics that take into consideration the conditional distribution of the features given a class are used to determine the splits. Lakshminarayanan *et al.* [149] proposed a new class of algorithm based on Mondrian processes [209] that provides a new way to train decision trees online.

Mondrian processes are a continuous-time Markov process that form hierarchical partitions of the feature space $\mathbb{R}^D$. The partitions are nested and each subsequent partition refines its parent. While these processes are non-parametric and define infinite partitions, Mondrian trees restrict them using a *lifetime* parameter λ. This parameter is however hard to tune, so the the authors choose instead to stop splitting nodes when the data points within them all have the same class value. Compared to regular decision trees, Mondrian trees have two main differences: The split decisions are always within the range of observed data, i.e. the split decisions create "boxes"

in feature space and not axis parallel cuts, and similar to the Extremely Randomized Tree algorithm [98], the splits positions are chosen uniformly at random. The main property that makes Mondrian trees possible to train online is *projectivity*. We can grow a new tree by sampling from a restricted distribution of Mondrian trees that have already been trained, extending the tree to include the new data. The Mondrian tree distribution is in this sense "self-consistent" [149] which allows us to grow them from the previously sampled tree in an online manner. The original Mondrian trees are aimed at classification and follow up work extends them to handle regression as well [148].

One main characteristic of Mondrian trees is that they are able to determine the full predictive posterior distribution of the dependent. This means that they are able to produce a distribution of the form $p_T(y|\mathbf{x}, \mathcal{D}_{1:N})$, where $y \in 1, ..., K$ for the multi-class classification scenario, or $y \in \mathbb{R}$ for regression. For classification the posterior is modeled as a hierarchy normalized stable processes [240], while for regression a hierarchical Gaussian is used. The algorithm uses an ensemble of Mondrian trees to create a Mondrian Forest and combine their predictions for a final output.

The main disadvantage of Mondrian forests is their computational cost. The model requires that learning happens at all levels of the tree, unlike most of the HT algorithms where learning only occurs at the leaf level. Because a complicated model of the posterior needs to be updated, the computational cost of the algorithm increases with each incoming data point and is $\mathcal{O}(\log n)$ for the n'th data point, or in other words it has a cost of $\mathcal{O}(\log N!)$ to train N data points. In addition, the online version of the algorithm needs to maintain all data points at the leafs in order to be able to update the distributions. This makes its memory cost prohibitive for a streaming setting with limited resources or unbounded data.

In our work we use Mondrian Forests as the state-of-the-art comparison in Paper IV and show that we are able to achieve similar accuracy with an order of magnitude reduction in runtime and bounded computational cost.

5.3 Online Ensemble Methods

Due to their approximate nature, online learning models often demonstrate limited accuracy compared to their batch counterparts. Ensemble methods have been shown to vastly improve the bias & variance characteristics of a wide range of models [73] and have therefore also been a focus in the online learning literature. In this section we will provide an introduction to some of the more established online ensemble methods that we have also used in our work, along with related recent work. We refer the interested to the survey by Gomes *et al.* [102] for a more in-depth look at online ensemble methods.

The two main paradigms in ensemble algorithms are bagging [39] and boosting

[90, 214]. In this section we will review how each of these algorithms has been adapted to the online setting. We will focus our descriptions on the first, and more established, work that deals with both online bagging and boosting proposed by Oza [186].

5.3.1 Online Bagging

Briefly, bagging works by training an ensemble of learners in parallel, each trained on a bootstrap sample [78] of the original dataset. The predictions of each learner are averaged to produce the final outcome.

The online bagging model proposed by Oza [186] and further explored in [187] is an intuitive algorithm, listed in Algorithm 1. We make use of this algorithm in Paper IV.

Algorithm 1: OzaBag($\mathbf{h}$, L_o, (x, y))

input : $\mathbf{h}$, the ensemble, a set of s hypotheses h_t; L_o, an online learning algorithm; (x, y), a labeled training instance.
output : Prediction $\hat{y}$.
foreach $h_t \in \mathbf{h}$ **do** // in order $t \in [1, s]$
  $k \leftarrow$ Poisson($\lambda = 1$)
  Assign weight k to (x, y)
  $h_t \leftarrow L_o(h_t, (x, y))$
return $\hat{y}$

The main insight of the algorithm, commonly referred to as *OzaBag*, is that for large N, we can approximate the binomial distribution used to determine whether a sample will be included in the bootstrap sample using a Poisson distribution. The distribution of bagging sample sizes, K, tends to a Poisson($\lambda = 1$) as the number of samples $N \to \infty$. To use bagging online then, for each incoming example and for each member of the ensemble $\mathbf{h}$, we draw a sample from a Poisson($\lambda = 1$). We use the drawn scalar k to modify the weight of the incoming instance, and train the algorithm using the updated weight. Oza proves that as the number of samples N grows to infinity, the distribution of the online training set will converge to that of the batch algorithm.

This algorithm has been adapted and extended in multiple online bagging methods. Wang *et al.* [237] make use of this strategy to deal with the class imbalance problem in online manner. Their strategy is to adjust the λ parameter of the Poisson distribution so that data points with underrepresented classes have a larger effect on the training. Bifet *et al.* [29] provide two alternative bagging methods with the purpose of dealing with concept drift, one that uses trees of different sizes, and one that makes use of the ADWIN [25] change detection method on top Oza's bagging algorithm to detect when a concept drift has occurred. Leveraging Bagging

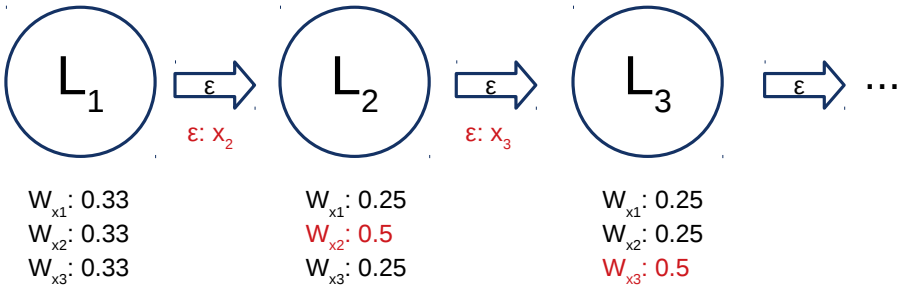


Figure 5.1: Example of batch boosting. After the ensemble makes an error for an instance, we increment the weight for that instance, and train the new learner using the updated weights.

[27] is an improvement to the previous model, where the authors increase the λ parameter of the Poisson distribution to "increase the diversity of the weights" and use the method of Dietterich and Bakiri [74] to handle multi-class cases as binary classification using error correcting codes.

5.3.2 Online Boosting

Boosting works by combining the predictions of many weak learners to produce a strong learner. The process adjusts the weight distribution of the data points at each iteration, assigning more weight to "difficult" instances that the current ensemble mis-predicts, forcing subsequent iterations to focus on those examples. An example is given in Figure 5.1. There, after L_1 mispredicts the class of x_1, we increase its weight in the following iteration, and train the new learner using the updated weights. The difficulty in online boosting is that instances arrive sequentially, making it challenging to maintain a distribution of weights.

Online boosting has found more widespread use compared to online bagging ensembles, as it has been used successfully in many computer vision applications, with a focus on object tracking [244, 10, 248, 137, 156, 104, 222]. As a result many different boosting algorithms have been proposed, some aimed at a specific application like object tracking, and others that are general learning algorithms.

We will focus on the original algorithm proposed by Oza, which we also make use of in Paper V, and the more recent general online boosting algorithms that improve upon the theoretical guarantees and performance of the original.

The original online boosting algorithm by Oza, referred to as OzaBoost, is listed in Algorithm 2. The learning process is similar to that of OzaBag (Algorithm 1), but now the algorithm is strictly sequential, and the weight the example takes

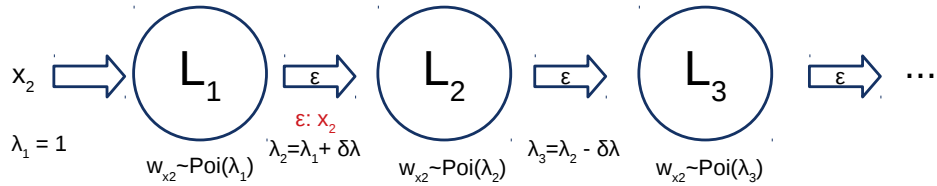


Figure 5.2: Example of online OzaBoost. The instance x_2 passes through each learner in sequence. Its weight is drawn from a Poisson distribution, whose λ parameter is increased when a learner makes an error, and decreased otherwise. Here L_1 makes an error, while L_2 correctly classifies the instance.

depends on whether the previous member of the ensemble was able to correctly classify the example. Specifically, the algorithm tries to assign half the total weight to the misclassified examples on the next stage, and the other half to the correctly classified ones. This is done by keeping track of the λ parameter sums for the two cases, correct and incorrect classification. We update the weight of an example before it passes on to the next member of the ensemble accordingly, increasing the weight every time is incorrectly classified, and decreasing it every time it is correctly classified. One thing to note about OzaBoost is that it requires that we set the number of boosting rounds from the beginning, unlike the batch AdaBoost algorithm. A simplified illustration of the algorithm is given in Figure 5.2

Despite its popularity, OzaBoost lacks rigorous theoretical guarantees. The first attempt to formalize online boosting was made by Chen *et al.* [54]. They re-visit the assumption made about the performance of the weak learners, namely that any weak learner will be able to do better than random guessing, as it is not a realistic assumption for the online setting where learner accuracy is more limited. They also provide a way to not have to set the number of learners in the ensemble beforehand, by dynamically assigning voting weights to learners. However, doing so requires the setting of another parameter γ, for which it is hard to determine good values. Their algorithm, OSBoost, extends the batch SmoothBoost [215] algorithm, which was designed as a boosting algorithm robust to noise, to the online setting. They use the weighting scheme from that algorithm to assign larger weights to incorrectly predicted examples, and prove that their ensemble can use the set of weak learners to achieve a small error.

The work of Beygelzimer *et al.* [16] improves upon OSBoost, by providing an optimal algorithm in terms of "the number of weak learners and the sample

complexity needed to achieve a specified accuracy", and also provides a parameter-free algorithm that does away with the need to set the γ parameter of OSBoost. The optimal algorithm, Online.BBM, is based on the Boost-by-majority batch algorithm [91] and relaxes the assumptions made by OSBoost, while the parameter-free algorithm, AdaBoost.OL, makes use of online loss minimization where Beygelzimer *et al.* choose to minimize logistic loss in order to avoid large weights which could adversely affect the error rate of the algorithm. We should note that all the above algorithms are strictly sequential and are therefore hard to parallelize. Our work in Paper V is able to make use of any online boosting algorithm to perform parallel online boosting, while maintaining their guarantees.

Algorithm 2: OzaBoost($\mathbf{h}$, L_o, (x, y))

init : $\lambda_t^c \leftarrow \lambda_t^w \leftarrow 0, \quad \forall t \in [1, s]$ `// cumulative weight of instances with correct and wrong predictions`

input : $\mathbf{h}$, the ensemble, a set of s hypotheses h_t; L_o, an online learning algorithm; (x, y), a labeled training instance.

output : prediction $\hat{y}$.

`// prequential evaluation: first test...`

$\hat{y} = \arg\max_{\dot{y} \in Y} \sum_{t=1}^{s} \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right) I(h_t(x) = \dot{y})$

`// ...then train`

$\lambda \leftarrow 1$

foreach $h_t \in \mathbf{h}$ **do** `// in order` $t \in [1, s]$

  $k \leftarrow$ Poisson(λ)

  while $k > 0$ **do** `// give weight` k `to the instance`

    $h_t \leftarrow L_o(h_t, (x, y))$

    $k \leftarrow k - 1$

  if $y = h_t(x)$ **then** `// correct prediction`

    $\lambda_t^c \leftarrow \lambda_t^c + \lambda$

    $\epsilon_t \leftarrow \frac{\lambda_t^w}{\lambda_t^c + \lambda_t^w}$

    $\lambda \leftarrow \lambda\left(\frac{1}{2(1-\epsilon_t)}\right)$

  else `// wrong prediction`

    $\lambda_t^w \leftarrow \lambda_t^w + \lambda$

    $\epsilon_t \leftarrow \frac{\lambda_t^w}{\lambda_t^c + \lambda_t^w}$

    $\lambda \leftarrow \lambda\left(\frac{1}{2\epsilon_t}\right)$

return $\hat{y}$

5.4 Evaluation

The evaluation of online learning algorithms differs from the batch case, not only because of the potential for a concept drift occurring, but also due to considerations about the computational cost of the methods themselves. The research on valid evaluation methods for streaming methods is somewhat limited however, with the notable exception of the work by Gama *et al.* [94], further expanded in [97]. In this section we provide a brief overview of the challenges and proposed solutions for the evaluation of online learning algorithms and refer the reader to [97] for an in-depth look into these issues.

Gama *et al.* [97] identify three areas that become important in the evaluation of streaming learning algorithms: *Space* in terms of the memory requirement of the algorithm, *learning time*, and *generalization power*. Each of these aspects is important in the online learning scenario, and we will discuss here the space and generalization power aspects, since learning time is a more intuitive concept.

Online learning algorithms are commonly designed for limited resource environments, and as such, the space requirements of the algorithm are an important consideration. Apart from the theoretical analysis of the space cost of the algorithms, their empirical costs should also be analyzed, as the empirical differences between algorithms with the same theoretical memory cost can be significant. Bifet *et al.* [28] introduced the concept of *RAM-hours* for this purpose, with the aim of estimating the cost-efficiency of an algorithm in a cloud environment where instances are charged according to the time taken to use them, and instances with higher memory capacities are usually more expensive. RAM-hours measure the empirical memory consumption of an algorithm, with one GB of RAM deployed for an hour equaling one RAM-hour. Domingos and Hulten [75] make specific mention of the memory requirements of the online decision tree algorithm they develop, and provide solutions that make the algorithm memory bounded. However as Gama *et al.* [97] mention, this is an aspect often overlooked in the evaluation of online learning algorithms.

In terms of evaluating the generalization power of online learning algorithms, one aspect that requires special attention is the fact that the learner evolves as we train it with more samples, so its performance in the early stages of learning can be very different to later stages. The two strategies proposed as viable evaluation strategies for the online setting in [97] are *holdout testing* and *prequential evaluation*. Holdout testing refers to the established evaluation method from batch learning, where we train our algorithm on a subset of the data, the *training set*, and evaluate its performance on a set of unseen data, the *test set* or *holdout set*. This requires us to set a specific interval at which we check the performance of the algorithm on the test set, although theoretically we could also perform this test after each example, introducing however a large computational overhead if the test set is large.

An alternative is to use predictive sequential (prequential) evaluation where

we present an example to the algorithm, make a prediction, update our metric, and finally reveal the example's label and proceed to train the algorithm with it. This method can also be used in situations where the label is available for only few of the points in the data set. However, because the model will exhibit worse performance in the early learning stages, it is recommended to apply a forgetting factor to the metric, so that large errors made in the start do not severely affect the overall evaluation of the algorithm. This can be done using sliding windows, i.e. evaluating the error using overlapping subsets of the incoming dataset, or using fading factors which discount the error for older examples [94]. These have the added advantage of being faster to compute and memory-less compared to using sliding windows.

One common issue in classification, in the batch as well as the online setting, is class imbalance. For example, on a dataset where 90% of the instances belong to a single class, a simple majority classifier will achieve 90% accuracy, although no real learning is taking place. To deal with this problem in the online setting, Bifet and Frank [23] proposed the Kappa statistic, and its extension κ_m [21],which is a robust estimator that takes into consideration the probability that a classifier will produce the correct prediction by chance. The metric has also been expanded to deal with temporal dependence in the labels of subsequent examples [257]. More recently, Brzezinski and Stefanowski [43] proposed an online adaptation of the area under the ROC curve metric (AUC) [87], called *prequential AUC* which can deal with the class-imbalance problem also in the presence of concept drift, and performed an evaluation against metrics like the Kappa statistic to show that each metric captures a different aspect of the algorithm's performance.

5.5 Software

Compared to the multitude of options available for batch learning, the availability of open-source software for online learning is relatively limited. In this section we mention the most popular libraries for online learning, and point out the ones we have have used and extended as part of this dissertation.

Perhaps the first online learning library released was the Very Fast Machine Learning (VFML) framework which was developed by Domingos and Hulten [75] as part of their work on the Very Fast Decision Tree (VFDT) algorithm (which we describe in Chapter 5.2). It includes the VFDT and its variations, as well as data pre-processing tools.

One of the most established open-source frameworks for online learning is MOA, which stands for Massive Online Analysis [26]. MOA includes a collection of learning algorithms, evaluation methods and metrics, data generators as well as a graphical interface to perform repeatable experiments. MOA is designed in the vein of WEKA [107] and includes interfaces that allow inter-operability between

the algorithms available in WEKA and MOA. Like WEKA, MOA has an extensible design that allows developers to re-use parts of the existing algorithms to develop new methods, and easily evaluate new methods using the evaluation strategies mentioned in Section 5.4, by adhering to a simple API. In our work, we have used MOA to implement the algorithms of Paper IV and as a baseline comparison in Paper V. Other single-worker online learning libraries include LIBOL [118] and more recently scikit-multiflow [182].

One drawback of these libraries is that they are designed to run on a single machine and therefore can have issues with massive, distributed streams. Apache SAMOA [63] is an effort to bring the design principles of MOA into the distributed setting, utilizing a platform-independent design that allows it to run on top of many distributed stream processing engines like Apache Flink [49], Apache Storm [230] and Apache Samza [185]. Similarly to MOA, it provides learning algorithms and evaluation methods, and its design makes it easy to extend with new algorithms. We have used SAMOA as the development framework for Paper V.

Another library aimed at distributed stream learning is the StreamDM library [31]. StreamDM also inherits some of the design aspects of MOA, providing similar interfaces to access learning algorithms and run evaluations, but unlike SAMOA is built to run on top of only the Apache Spark [247] streaming engine. While running on top of a distributed stream engine can make the development of distributed algorithms easier, it can have a negative effect on the performance of the library, due to the overhead introduced by the engine. To tackle this is issue, an optimized version of StreamDM was proposed in [32]. An earlier attempt at developing a distributed streaming learning framework was Jubatus [113].

Finally, one of the most successful frameworks that relies on online learning, albeit to deal with batch problems, is Vowpal Wabbit (VW) [3]. VW makes use of parallel and distributed Stochastic Gradient Descent [36] as its main learning algorithm and has been extended to provide a number of online learning models, including online boosting [16], online Latent Dirichlet Allocation [117], and contextual bandits [77].

Part II

Results

CHAPTER

6

Graph Vertex Similarity And Concept Discovery

In this chapter we provide a summary of our work on scalable graph vertex similarity calculation and concept discovery, described in Papers I and II. We describe the problem and some related work in Section 6.1 and present our approach in Section 6.2. We give our findings in Section 6.3 where we also present an experiment that was not part of our original work, that bridges deep learning with our work to extract visual concepts from images. We close the chapter with a discussion relating this work to our overall research question in Section 6.4.

6.1 Background

This work tackles the general problem of determining the similarity between *objects* in a data set, which can take the forms of cosine similarity between items embedded in a low-dimensional representation [178], or, as done in this work, by modeling the problem as graph vertex similarity calculation. We use the generic term *object* to emphasize the generality of our approach, made possible by the graph representation. In the papers we present examples from the linguistic, music, and molecular biology domains, and in this dissertation provide an additional example using generated image tags as input, in Section 6.3.3.

In our approach we model the objects as nodes in graph and their interactions as edges, initially weighted by a simple measure of correlation between the objects. In text for example, the nodes would be words and the edges could be weighted by their pointwise mutual information [59].

The computation of all-to-all similarities in a graph can quickly become intractable for large graphs. The popular SimRank algorithm developed by Jeh and Widom [125] for this purpose, has a cubic time complexity with respect to the

number of nodes in the graph. Clearly then for large-scale graphs some form of approximation needs to be used to make the computation feasible.

6.2 Scalable Vertex Similarity and Concept Discovery

We deal with the scalability problem by using a two-step process to calculate similarities. We first create a *correlation graph* that models simple correlations between objects. These correlations are easy to extract and do not necessarily hold semantic information. For example, given a corpus of text we can use a co-occurence measure, like pointwise mutual information, between any two pairs of words and use that as the edge weight between the two in the correlation graph. We then compute the similarities between nodes by applying a transformation on this graph, to produce a graph which surfaces deeper, semantic interactions which we term the *similarity graph*. In order to determine the similarity between objects in a graph, we examine their agreement in correlations to other objects: if two nodes are highly correlated to many common nodes, they are more likely to be similar themselves.

However, computing these all-to-all similarities can be challenging for correlation graphs with billions of nodes. As in other works in this thesis, we apply a combination of using approximations to limit the computational cost of the method, and develop a distributed algorithm that allows us to scale-out learning and take advantage of modern computing clusters.

The first approximation we employ makes use of a characteristic that many real-world datasets exhibit, that is that most objects are uncorrelated and the graph is therefore sparse. These type of graphs, appear in areas like gene co-expression [130], semantic networks [224], word co-occurrences [48] and social networks (see [4] for further examples). This allows us to prune the input correlation graph significantly before transforming it, while maintaining a controllable and small approximation error.

The second approximation has to do with the locality of the computation. Instead of trying to solve the computationally intractable all-to-all similarity problem, we instead only look at nodes that are two hops apart, keeping the computation local and dramatically reducing the cost per node in the graph. This approximation is what makes the distributed implementation of the algorithm efficient. In order to compute, for each pair of nodes in a neighborhood, their common sets of correlated nodes, we need only examine the common sets of incoming edges per node. By distributing our graph using the id of the incoming nodes as a key we can perform this computation using a self-join operation, which involves no communication of data in the cluster. As a result we are able to scale the computation to massive datasets, as we demonstrate by training on the Google Books n-gram dataset, which corresponds to approximately 4% of all the books ever printed [177] (24.5 billion rows), in a few minutes.

Once we have built the similarity graph, we can go one step further in order to extract groups of inter-similar objects that we call *concepts*. These correspond to groups of objects that belong to the same semantic group, based on their appearances in the context of other objects. In the language example these could be words that describe the same concept such as "cities", or words with the same syntactic and semantic purpose, like groups of adverbs. To create these groups we apply a community detection method on top of the similarity graph. Because the similarity graph itself can be large, we develop a scalable approach which is based on the speaker-listener label propagation algorithm (SLPA) [241].

6.3 Main Findings

In the paper we provide examples of the output produced by the algorithm, and perform a quantitative evaluation of the similarities produced using a gold standard dataset, WordSim-353 [85]. Here we provide a summary of some of the experiments presented in the paper, and a new set of results using images as input.

6.3.1 Billion words corpus

In the paper we perform an evaluation on the Billion word corpus [53] which contains approximately one billion tokens and was scraped from online news sources. We create the correlation graph by using the relative co-occurrence frequency of word pairs, that is the (directed) correlation between the words i and j, $\rho_{i,j}$, will be $c_{i,j}/c_i$ where $c_{i,j}$ the number of times the words co-occur within a window, and c_i the frequency of word i in the corpus. For this experiment we select a window size of two, meaning we only take into account bigrams. We can see some example concepts uncovered by the algorithm in Figure 6.1.

6.3.2 Music

We also perform experiments on a music listening dataset created by Celma from the Last.fm music service [51]. The dataset contains 19 million track plays from 992 users, which we use to create a correlation graph between artists. The correlation measure $\rho_{i,j}$ is the number of times artist i was followed by artist j, divided by the total number of plays for artist i. Some example artist concepts which correspond to music genres can be seen in Figure 6.2.

6.3.3 Uncovering Visual Concepts

In this experiment we make use of a combination of deep neural networks and our algorithm to uncover visual concepts from images. Deep neural networks can be trained on labeled datasets of images to recognize multiple objects in an image

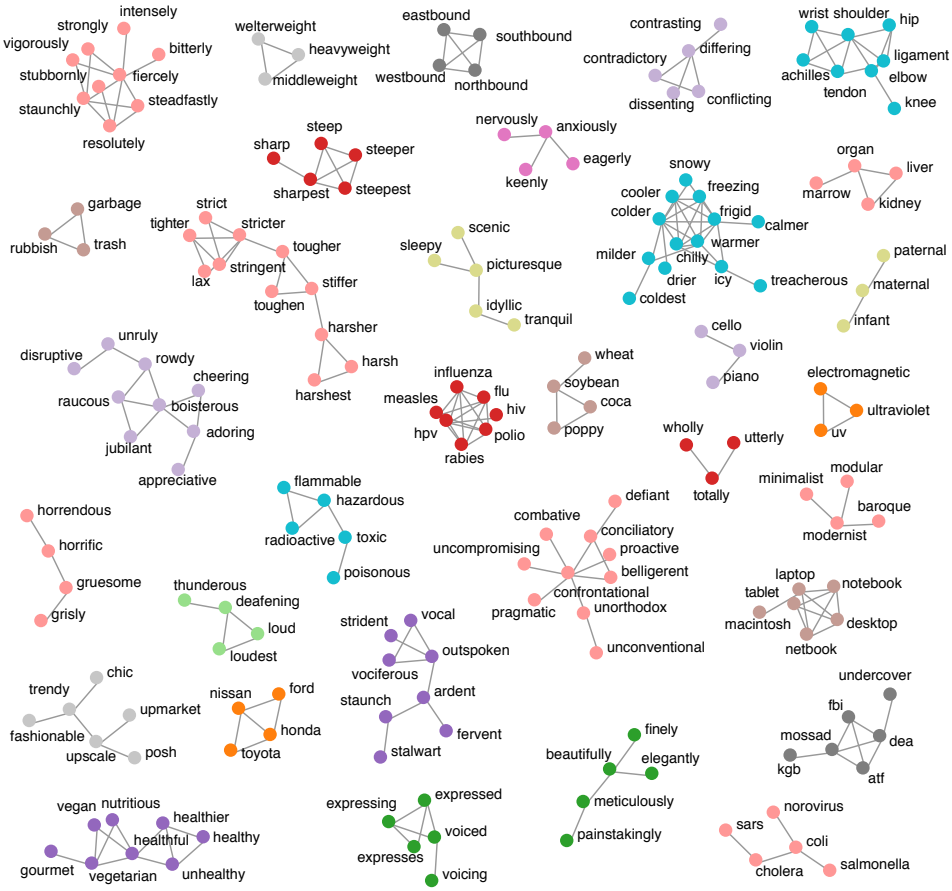


Figure 6.1: Example concepts extracted from the Billion word corpus

[103]. The networks' accuracy creates a unique opportunity to generate visual concepts out of raw images: A network trained on labeled dataset like ImageNet [210] can be used to generate labels from unlabeled raw images thus generating a "silver-standard" dataset which we can use as input for our algorithm.

In our experiments we have used the OpenImages dataset [146] which uses multiple image classifiers to annotate images with labels from 19,794 different classes. The complete training dataset contains 9,011,219 images scraped from the Flickr service[1], of complex scenes annotated with 8.4 objects per image on average.

We use the OpenImages data to create a correlation graph of objects: We form a clique for every set of objects that exists together in one image, creating a graph that

[1] www.flickr.com

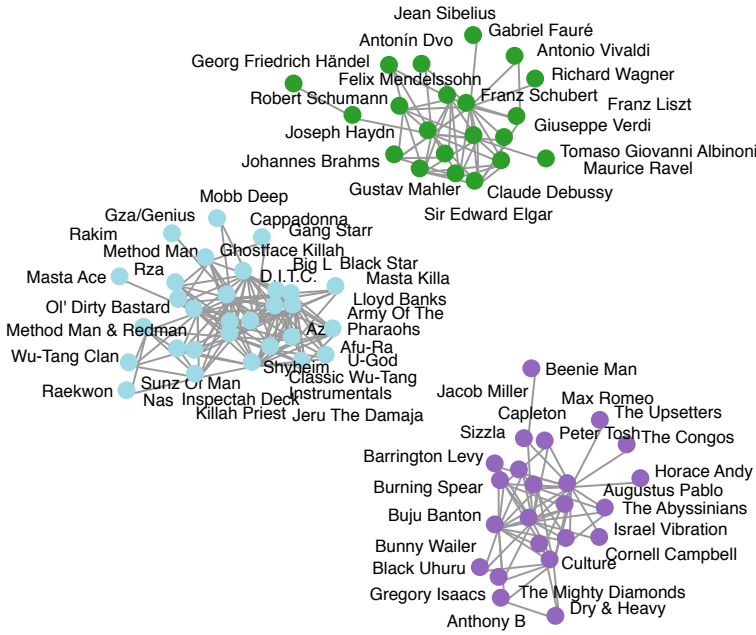


Figure 6.2: Example concepts extracted from the music listening dataset.

represents how objects commonly appear together in real images. Take for example the annotated image shown in Figure 6.3a. The image is annotated with both the *horse* and *cowboy hat* labels. All the labels in the image will form a clique in our graph. Looking at another image, 6.3b we can now extend this graph, by connecting the *cowboy hat* label with the *guitar* label, creating a second degree link between the *horse* and *guitar* labels. By incorporating all such images, our correlation graph will have a representation of objects that tend to appear together in the real world. Using this graph as input, we can apply our similarity transformation to group together objects that belong to semantically similar classes, based on the similarity between the contexts in which they appear in, in the real world. Finally we can apply our community detection algorithm to group together items to form visual concepts in the similarity graph.

To illustrate the uncovered concepts, we take the 30,000 most similar pair-wise object similarities and present the output similarity graph in Figure 6.4, where the colors indicate the uncovered concepts. To ease presentation we provide two zoomed-in parts in Figure 6.5, where we can see objects that can roughly be described as "uniformed people" are grouped together in Figure 6.5a, and objects that relate to cameras grouped together in Figure 6.5b.

Using our algorithm in combination with deep learning models, we can use the

wealth of unlabeled images that exist in services like Flickr to create visual concepts.

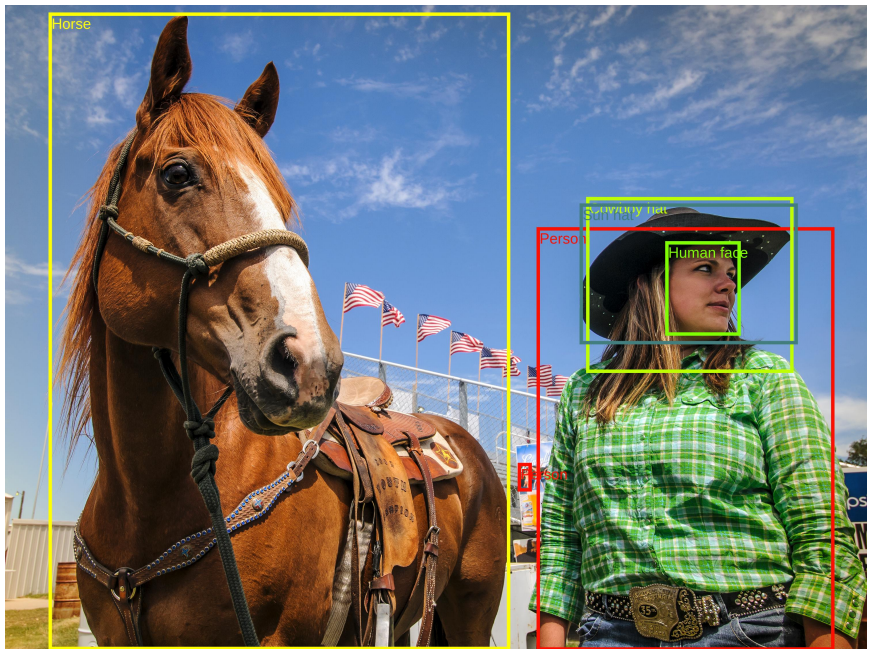
6.3.4 Quantitative evaluation

To provide a quantitative evaluation of the method we made use of the WordSim-353 (WS-353) dataset. This dataset includes 353 word pairs that are rated by humans for their similarity. This dataset includes unrelated words which presents a problem for our approach that by design does not calculate similarities for words that are unrelated, as using the graph structure is one of the main ways we can make the computation scale. For that reason we use the word pairs that appear both in the evaluation dataset *and* exist in our similarity graph. We use the Google n-grams data with a corpus of 361 billion tokens, which in under 10 minutes produces a similarity graph that contains 60% of the word pairs present in WS-353.

The produced similarities have a Spearman rank correlation of 0.76, which is exactly what the inter-annotator agreement is for this dataset. Finally we compare the WS-353 and our produced similarities with the cosine similarities of the GloVe [194] embedding vectors, which was at the time of publication the state of the art language embedding method. We train our method on a smaller corpus and smaller window than the GloVe vectors, but are still able to generate similarities comparable to GloVe (0.71 Spearman rank correlation).

6.4 Discussion

In this chapter we presented our work on scalable vertex similarity and concept discovery. We made use of approximations to dramatically reduce the amount of computation, first by using locality in our computations, and then by taking advantage of the structure of the graphs to dramatically reduce the number of edges that are involved in our similarity calculations, further decreasing the computational cost. We designed the algorithm for a distributed setting, and through careful design of the similarity calculation step, we are able to minimize the amount of network communication necessary. Thus, in this work we cover two of the three objectives defined in Section 1.1 and tackle our original research question in the context of vertex similarity.



(a) An image that contains a horse and a cowboy hat.

(b) An image that contains a cowboy hat and a guitar.

Figure 6.3: Examples of annotated images from the OpenImages dataset.

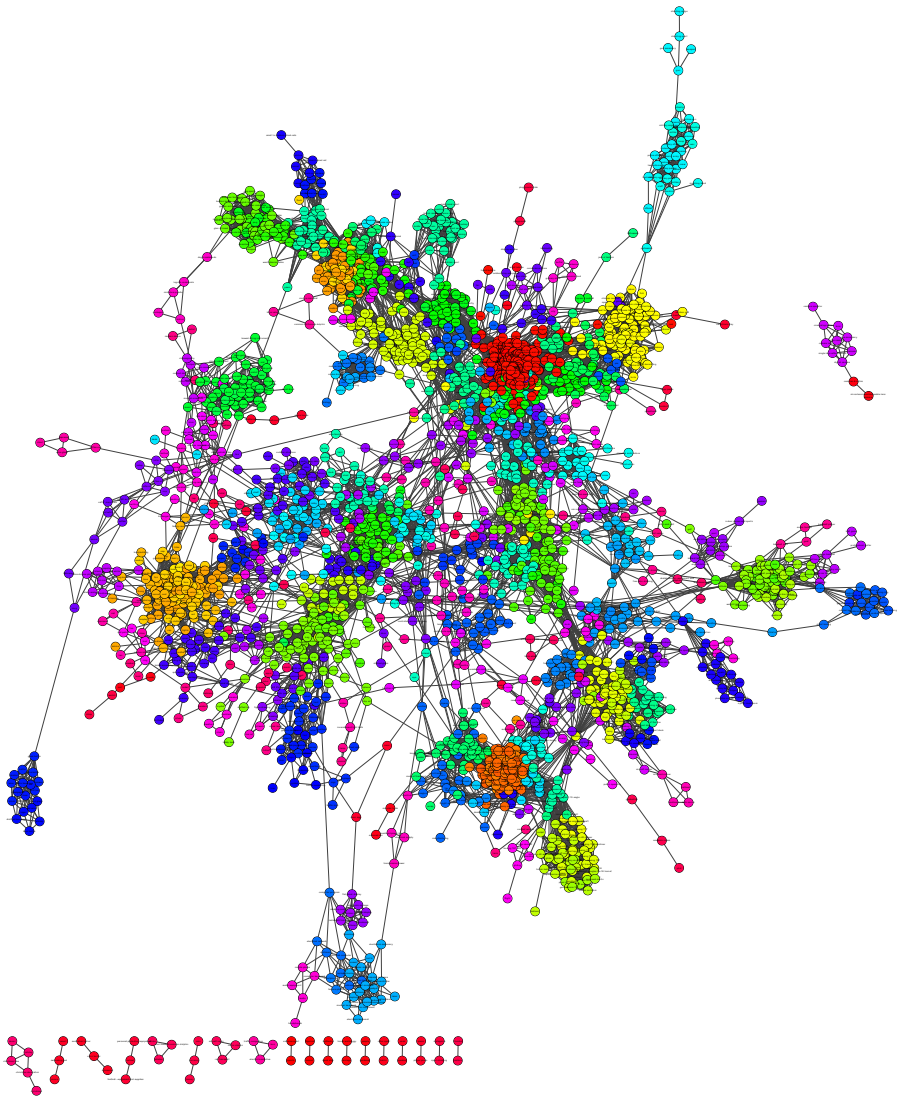
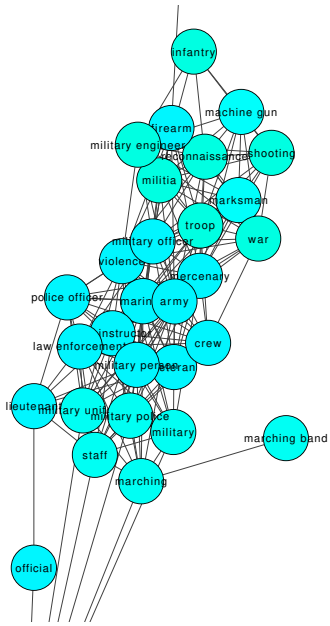
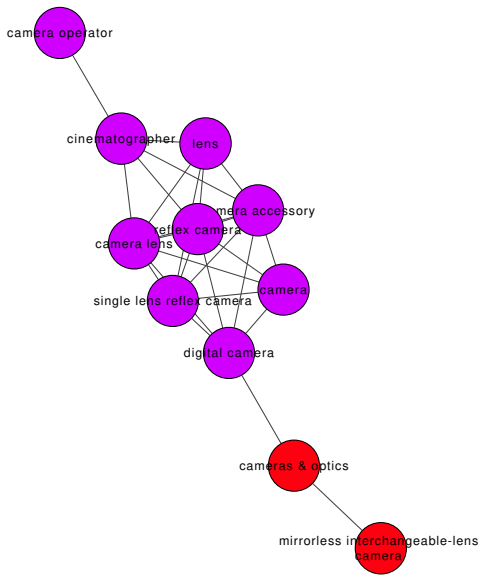


Figure 6.4: The concepts uncovered by using the OpenImages data as input to our algorithm. The labels are legible through zooming in the electronic version. See Figure 6.5 for zoomed-in crops of the graph.



(a) A concept that brings together uniformed professions and military objects.



(b) A concept that groups together camera equipment.

Figure 6.5: Two visual concepts from the top-right corner of Figure 6.4.

CHAPTER

7

Use-case: Session Length Prediction

In this chapter we provide a real-world use-case that serves as motivation for the rest of the work described in this dissertation. After giving an introduction to the problem and related work in Section 7.1, we describe the methods we used to analyze the data and make predictions in Section 7.2 and summarize our findings in Section 7.3. Finally, in Section 7.4 we provide desiderata for a real-world system and identify gaps in the current state-of-the-art research. We provide solutions to each of the specific problems identified here in the following chapters.

7.1 Background

As media streaming services have proliferated it has become important for streaming companies to analyze their users' behavior to optimize their offering and meet their business goals. One important factor in user satisfaction is the length of users' sessions [138], which we define as the the complete interaction of a user starting up the service, consuming a number of items, and ending their interaction after some elapsed time. This kind of interaction has been studied in the web search [138, 35] and ad click [150, 11] domains, but no study had investigated the media streaming domain before.

Streaming services can use the length of user sessions to optimize their recommendations, for example providing exploratory and more "risky" recommendations for long sessions, versus exploitation and more "safe" recommendations for short sessions. In ad-supported services, having an indication of the session length can also help with scheduling ads, allowing the provider to meet their revenue target, while minimizing the annoyance to the user [100].

Predicting the length of user sessions in a mobile streaming service can be challenging, because user interaction lengths typically exhibit long-tail distributions [11, 162, 216], and the external factors that can influence the length of the session

can be hard to model, like users commuting, taking phone calls, or connectivity issues. Media streaming sessions also differ from dwell time after ad clicks and web search. In music specifically, which is the domain we are examining, users typically consume multiple items in one session and have different behaviors depending on the type of session as we show in our results (Section 7.3).

In our paper we provide an analysis of the session length distribution using tools from survival analysis and build a predictive model using gradient boosted trees with specialized loss functions to place the probability mass correctly in the presence of a skewed dependent distribution.

7.2 Analysis and Prediction of Media Streaming Session Length

In this section we describe the methods we have used to analyze the data and create a predictive model for session length.

7.2.1 Weibull Analysis of Session Length

To analyze the user length behavior of the users we use tools from survival analysis [140], and specifically the Weibull distribution [50]. The Weibull distribution is a flexible parametric distribution, commonly used in survival analysis because it allows to model different kinds of failure rates, where the probability of a unit failing changes over time. Its probability density function is:

$$f(t) = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1} e^{-(t/\lambda)^k}, t \geq 0 \tag{7.1}$$

and provides two parameters: the shape k and the scale λ. The shape k determines the evolution of the failure rate over time, while the scale, λ, determines the spread of the distribution. The effect of k is best shown through an illustration of the hazard rate for the Weibull function, that gives us the failure rate for an item that has survived until time t, shown in Figure 7.1.

The failure rate increases with time for $k > 1.0$, which is called the "positive-aging" effect, and decreases with time for $k < 1.0$, an effect called "negative-aging". Positive aging is what we commonly associate with web sessions and is indeed observed in 98.5% of post-click behavior [162]: as time goes on users become more likely to quit the session at any point. On the other hand, negative aging means that sessions become less likely to end as time goes on. This behavior is also described as "infant mortality" where defective units may fail early on, but as time goes on they become less likely to fail. For $k = 1$ the failure rate is constant and the distribution becomes equivalent to the exponential distribution.

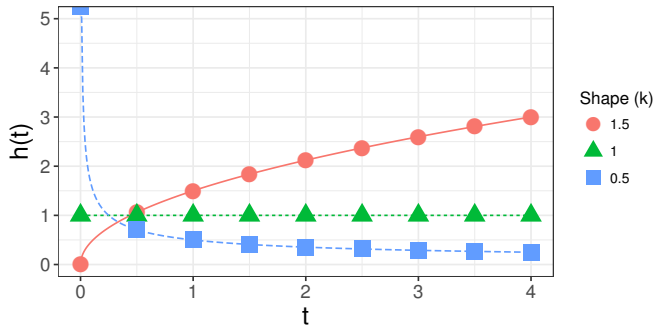


Figure 7.1: The failure rate of the Weibull distribution for different values of the shape parameter, k. We set $\lambda = 1$.

7.2.2 Prediction

For prediction, in order to tackle the issue of the long-tail distribution of the sessions and to extract as much power as possible from the predictive features, we choose the gradient boosted tree (GBT) algorithm. GBTs allow us to customize the loss function to better fit the long-tail distribution of the dependent, and are flexible enough to discover patterns in the large data we have available, and can readily handle missing data which are common in industrial settings and our dataset in particular.

We extracted a set of features for each user, like their gender, age, subscription status (paid or free user), and a set of contextual features for each session, like the device type the session is on, the type of network (mobile, WiFi), or the duration of the last session.

Our dependent is non-negative and long-tail distributed. One common solution in these cases is to log-transform the dependent and then fit using a squared loss, but we often want to use a loss function that is better suited to the task. To that end we try fitting the GBT both on the log-transformed data with a mean squared error objective, as well as fitting on a log-likelihood objective of a Gamma function with a log link function, which can explicitly model non-negative data with a positive skew. We also test two versions of each model: One where all data are aggregated and we build a single model, and one where we build a personalized model for each user.

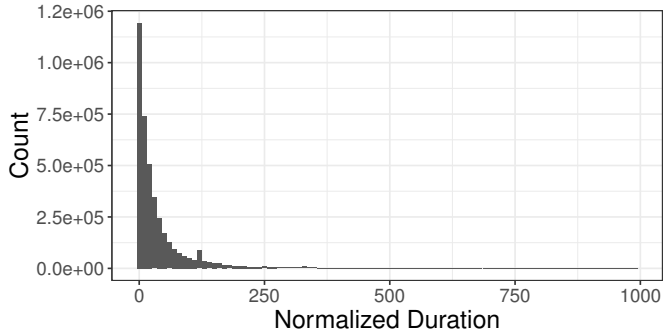


Figure 7.2: Histogram plot of session length. The x-axis has been normalized to the 1-1000 range.

7.3 Main Findings

7.3.1 User session distribution characteristics

We use a dataset of user interaction from the Pandora music streaming service, which is a major US-based music streaming service, and mainly ad supported. We define sessions as periods of listening activity that are demarcated by breaks or pauses of 30 minutes or more. We gather data from a random subset of users for the months of February to April 2016, resulting in 4,030,755 sessions. We plot the normalized duration data in Figure 7.2.

We analyze the session length distribution by fitting a Weibull distribution to each user's data, and plot the resulting Empirical Cumulative Distribution Function for the shape parameter in Figure 7.3. We can see that unlike the web site visits after a search, that had a negative aging effect ($k < 1$) for 98.5% of the users, only 44% of the users exhibit this behavior for music streaming. The behavior of the users is then split roughly down the middle, with some users having sessions that become more likely to end as time goes on, and others whose sessions become less likely to end as they grow longer.

7.3.2 Predictive model performance

In our analysis of the performance of the predictive model we use two metrics. We use the normalized Root Mean Squared Error which is a common choice for regression problems, but also employ the normalized mean absolute error, a metric that is robust to outliers, to account for the long-tail distribution of the dependent. For a baseline predictor we use the mean session length per user, a heuristic that is simple, but personalized. We present the results in Table 7.1.

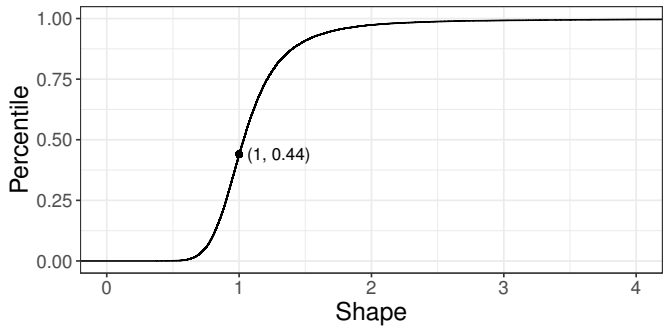


Figure 7.3: The empirical cumulative distribution for the shape parameter per user. The x axis has been truncated at $x = 4$ for readability (99.5 % of data points shown).

Table 7.1: Performance metrics for length prediction task. We report the mean value across the 10 CV folds, and the standard deviation in parentheses.

Method (*Objective*)	Normalized MAE	nRMSE
Baseline	1 *(0.001)*	1.16 *(0.005)*
Aggregated (*MSE*)	**0.71** *(0.008)*	1.23 *(0.008)*
Aggregated (*Gamma*)	0.93 *(0.007)*	**1.10** *(0.005)*
Per-user (*MSE*)	0.83 *(0.002)*	1.29 *(0.004)*
Per-user (*Gamma*)	0.86 *(0.001)*	1.31 *(0.003)*

From the results we can see that the aggregated models perform better overall, something that can be explained by the fact that the per-user models are trained on a few data points for most users, leading to over-fitting. The models using the MSE objective place most of their probability mass closer to the origin and as a result perform better in terms of MAE, but miss many of the longer sessions and as a result perform worse in terms of RMSE.

7.4 Discussion

In this chapter we presented our work on the analysis of session length distribution in media streaming, and presented a specialized predictive model. Through this work we are able to identify several limitations in the current state-of-the-art in decision tree learning.

First, this work demonstrated the need for algorithms that are able to learn online and adapt to a constantly changing environment. When providing estimates of users' session length, we want the predictions of the algorithms to adapt to the patterns exhibited throughout the day, such as day/night and work/home cycles,

and be able to adjust to drifting distributions. In order to be able to continuously adapt our models, we have focused on online learning methods in Papers IV and V that are both scalable and can continuously update tree models with new information about the world.

Second, we were able to identify the importance of quantifying the uncertainty in predictions. For a quantity such as session length, exact predictions can be of little value. The ability to quantify the uncertainty in a prediction can be more useful as it allows us to make decisions with confidence and avoid giving users a bad experience. For example, if we know with a high degree of certainty that the true value of the session length will be between 30 and 45 minutes we can optimize recommendations and ad scheduling for a long running session. While there exist methods that are able to provide uncertainty estimates from decision trees for static datasets, no methods exist for online decision trees. Paper IV fills this gap in research and provides online decision trees models with the ability to produce uncertainty estimates.

Finally, during this study we were able to get first hand experience with the accuracy, flexibility and scalability that boosted decision trees provide. The success of boosted trees has been well-documented, from winning data mining competitions [55], to being the model of choice for mission-critical applications such as ad targeting at major enterprises [159, 111]. What's common in these applications is the need to deal with potentially very high-dimensional data at scale. In our follow-up work, we have focused on expanding the area of scalable boosted trees for high-dimensional data in Paper V in the online domain, and Paper VI in the batch domain.

CHAPTER

8

Uncertainty Quantification In Online Decision Trees

In this chapter we present our work on estimating the uncertainty in the predictions of online decision trees. We define the problem and present some related work in Section 8.1, describe the two algorithms developed for this purpose in Section 8.2 and present our empirical evaluation results in Section 8.3. We close the chapter with a discussion placing the work in the context of our original research question in Section 8.4.

8.1 Background

As machine learning methods mature and become increasingly popular, they are more often being deployed in critical scenarios where mistakes can be costly. Such domains include autonomous vehicles, and the medical and financial sector. In order for critical decision-making to be done by learning algorithms in these areas, we need to have a notion of uncertainty in the algorithms' predictions. Some of these critical domains also include the requirement that critical decisions be made in real-time in a constantly evolving environment, and under a tight computational budget. Examples include the financial domain [96] and autonomous vehicles [171].

These examples demonstrate the need for highly accurate algorithms that can be trained online, adapt to an evolving environment, and quantify the uncertainty in their predictions. Most of the online learning methods available today do not offer quantification in their predictions, while most methods that provide uncertainty estimates are designed for the batch domain where the data are static and bounded. In this work we aim to provide highly accurate algorithms that can be trained online, on a sequentially arriving and potentially unbounded dataset, that also provide uncertainty estimates for their predictions.

We develop two algorithms, one based on conformal prediction [235] and one based on quantile regression forests [175]. The two algorithms are general random forest learners that enable high accuracy [84], are computationally bounded and adapt to concept drift through the use of online base learners [120], and provide uncertainty estimates at an accuracy level equal to or better than the state-of-the-art [149], while being an order of magnitude faster to run.

8.2 Uncertainty Estimation for Ensembles of Online Decision Trees

In this section we provide an overview of the methods developed for this work. We provide a brief description of the batch algorithms that we base our algorithms on, and proceed to explain how we make use of approximations to transfer them to the online domain. We target regression problems, and our task is to develop algorithms that exhibit what is called *conservative validity* in conformal prediction literature. This refers to algorithms that produce predictive intervals such that the probability of the true value not being part of the interval is *less than or equal to* α, which is referred to as the *significance level*. This task is different from quantile regression [142] where the objective is for the error level to be *exactly* α.

8.2.1 Inductive Conformal Prediction

Conformal prediction (CP) is a meta-algorithm that can add the ability to produce *prediction regions*, that is, sets of classes for classification or intervals for regression, to any point-prediction algorithm. It works by quantifying how surprising or *non-conforming* an instance is compared to what the learner has observed and producing a region that is guaranteed to contain the true value at a requested degree of certainty. CP is rigorously proven to be *valid* [235], that is, the probability of error is bounded by the pre-selected *significance level*.

While CP was originally proposed as an online learning framework, its computational requirements made it impractical for large scale data, as it required to re-train the model from scratch with the complete dataset for each incoming example. Papadopoulos *et al.* [191] proposed *Inductive Conformal Prediction* (ICP) as a way to bound the computational cost of the method in the batch setting. Instead of constantly re-training the learner, it sets aside a subset of the training examples referred to as the *calibration set*, and uses this to determine the non-conformity scores of the examples and determine the intervals.

However, ICP is an offline method designed for the batch setting and assumes that the complete dataset is available at training time, in addition to requiring a external validation set to use as the calibration set. As such it cannot be applied to the online domain.

8.2.2 Quantile Regression Forests

Quantile Regression Forests (QRF) [175] are an extension to the random forest [41] algorithm that enables estimation of the full conditional cumulative distribution function (CDF) instead of only the conditional mean. Using the conditional CDF we can estimate prediction intervals for a given significance level by computing the quantiles at the edges of that interval. For example, for a significance level of 0.1, we can take the 0.05 and 0.95 conditional CDF values to produce the interval.

QRF works by maintaining at the leafs not just the mean label value of all the data points that get routed to that leaf, but maintaining a mapping from every instance to its label in its leafs. Otherwise, it grows trees like a regular random forest model. QRF uses a weighted sum of all the label values to determine the CDF estimates.

However because QRF requires access to the complete dataset and needs to maintain a mapping from each instance to its label value, it can cannot be used in an online setting where instances arrive in sequence, and memory is bounded.

8.2.3 Conformal Prediction with Online Regression Forests

Our algorithm adapts ICP to the online setting by using online tree learners to always maintain an up-to-date model, and uses a bounded FIFO queue of instances as its calibration set to keep the memory requirements of the algorithm bounded. We make use of out-of-bag examples from the bagging process in order to remove the need for a separate calibration set, as done in [129]. We make use of the online bagging algorithm by Oza [186] and the online regression trees by Ikonomovska *et al.* [120] to develop online regression forests that produce confidence intervals.

The algorithm works by maintaining a bounded FIFO queue of examples that are used as the calibration set for the CP algorithm. This set gets populated whenever an example is out-of-bag for at least one tree in the ensemble. In the online bagging algorithm we are using, a draw from a Poisson distribution is made for a each incoming instance, for each learner in the ensemble. If the drawn sample is not larger than zero, that example is out-of-bag for the learner and we add it to the calibration set. We maintain a sorted list of predictions for the examples that are out-of-bag which give us the *non-conformity* scores for the set.

At prediction time, we use these non-conformity scores to produce the intervals: After a regular ensemble prediction is made for the incoming instance, we determine the interval boundaries by moving through the sorted list of non-conformity scores, and pick the value $\phi = \mathbf{C}[\lfloor(1-\alpha)\cdot|\mathbf{C}|\rfloor]$ to be half the length of the interval, where $\mathbf{C}$ the sorted set of non-conformity scores and α the requested significance level. We call this meta-algorithm CPExact.

Approximations

The map of predictors to out-of-bag predictions needs to be up-to-date with the latest version of the predictors. That is, whenever an online learner is modified, we need to update all its predictions in the calibration set. In the worst case, it is possible that we will need to make $|\mathbf{C}| \cdot |\mathrm{L}|$ predictions for a single incoming instance, where L is the set of learners. For a calibration set with thousands of instances and hundreds of trees in the ensemble, this can quickly become computationally intractable. For this purpose we propose an approximate version of the algorithm that only updates the predictions of the new data points that have entered the calibration set since the last prediction. This allows for significant computation savings, while still providing an up to date representation of the data points' non-conformity, by constantly adding new instances to the set and removing older ones. We call this meta-algorithm CPApproximate.

8.2.4 Online Quantile Regression Forests

The batch QRF algorithm cannot be applied to the online setting as it requires a mapping from the the complete set of instances to their labels to be stored in memory at the leafs. The main idea for our algorithm is to use an approximate data structure to store a sketch of the labels cumulative distribution at each leaf, making it possible to estimate the quantiles from them.

To that end we use the state-of-the-art KLL quantile sketch [133], that gives a memory and computation-efficient way to approximate the CDF from a stream of data. An important characteristic of the KLL sketch is that it is *mergeable*, that is, we can apply the sketch to different partitions of a stream, and subsequently merge them, and the result should be the same as if we had applied the sketch to the complete stream. We use this property to make the predictions in our ensemble which are trained with samples of the data.

We maintain one of these sketches at every leaf in the trees of the ensemble. For every incoming instance that gets filtered to a leaf, we update its sketch with the label, thereby maintaining an up-to-date sketch of the CDF for every leaf. At prediction time, we drop the instance down every tree in the ensemble, and retrieve every sketch. We can then merge those sketches together to extract the overall estimate of the CDF for the ensemble.

8.3 Main Findings

We evaluate our algorithms against a simple baseline and a state-of-the-art algorithm, Mondrian Forests (MF) [149]. We use 20 small-scale datasets gathered from the OpenML repository [232], as well as 10 large scale datasets that exhibit concept drift. We note that while the approaches we developed are model-parallel, i.e. every tree

Dataset	MF	OnlineQRF	CPApprox	CPExact
Small-scale data	41.6	5.4	5.7	102.3
Flight delays	6340	836	899	27863
Friedman	2380	114	213	2011

Table 8.1: Average running times for all algorithms (seconds).

can be trained independently in parallel, the results listed are single-threaded to ensure fair comparison with MF.

Evaluating interval prediction algorithms entails two aspects: The error rate and the width of the intervals. The algorithms should maintain an error rate that is below the significance level set by the user. However, this can be easily achieved by producing very wide intervals that are uninformative. For this reason we also need to take into account the width of the intervals when evaluating the quality of the predictions. We measure the actual error rate using Mean Error Rate (MER) which is the mean of the 0-1 loss of the algorithm, and for interval size we compute the Relative Interval Size (RIS) that is the normalized mean size of the intervals produced.

To ease presentation we use two different metrics that combine both aspects: the quantile loss [142] and Utility, based on the time-utility functions used in real-time systems [199]. Unlike quantile loss, Utility will only penalize methods that go above the requested error rate, and as such is a better fit for our conservative validity task.

We can see the results in terms of Utility for the small-scale data in Figure 8.1 and for the large-scale drifting datasets in Figure 8.2. We can see that our methods are able to outperform the Mondrian Forest baseline in terms of Utility, while at the same time being computationally more efficient. The runtimes for each category of dataset are listed in Table 8.1, where we can see that our algorithms are up to an order magnitude faster than Mondrian Forest.

We also provide an evaluation for different significance levels. Depending on the cost of making a mistake, we might require different levels of confidence in our predictions. Thereby it's important to measure the performance of the algorithms over a range of different significance levels. The results are listed in Table 8.2 where we can see that Mondrian Forests cannot maintain the error guarantees above 80% confidence, and that OnlineQRF provides the best compromise between maintaining the error rate below the requested level, while keeping the produced intervals informative.

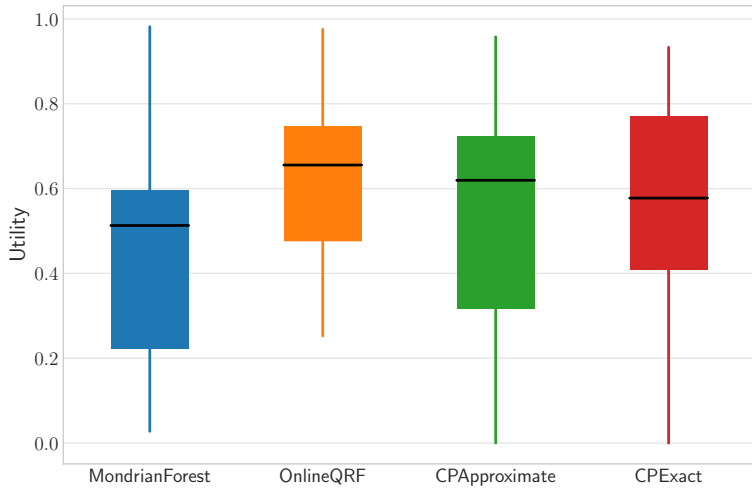


Figure 8.1: Utility for the small-scale data. Higher is better.

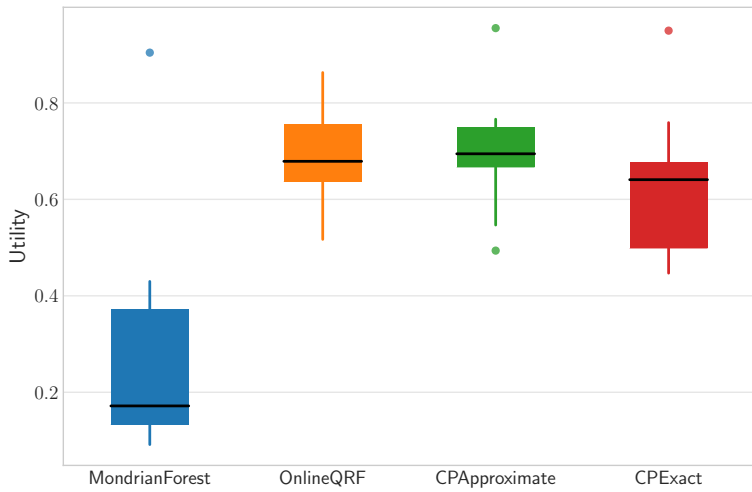


Figure 8.2: Utility for the concept drift data.

Method	Metric	α				
		0.3	0.2	0.1	0.05	0.01
MondrianForest	MER	0.28	0.20	0.13	0.094	0.062
	RIS	0.19	0.23	0.29	0.349	0.460
OnlineQRF	MER	0.23	0.14	0.07	0.036	0.015
	RIS	0.20	0.23	0.31	0.345	0.510
CPApprox	MER	0.22	0.13	0.06	0.032	0.006
	RIS	0.21	0.31	0.48	0.785	1.130
CPExact	MER	0.25	0.16	0.08	0.039	0.009
	RIS	0.28	0.37	0.57	0.622	0.944

Table 8.2: MER and RIS for different significance levels. The MER should be at most α where α the significance level.

8.4 Discussion

In this chapter we have presented our work on uncertainty estimation for online decision trees. We developed two algorithms for this purpose, and demonstrated their accuracy and efficiency compared to a state-of-the-art algorithm.

Connecting this work to our original goals in Section 1.1, we have dramatically reduced the computation necessary for online conformal prediction by maintaining up-to-date online models. For both OnlineCP and OnlineQRF we have bounded the memory use of the algorithms by using approximate and bounded data structures, that are able to provide the estimates we need at a fraction of the memory cost, compared to exact data structures. The approximations made however create a tradeoff, as we can no longer guarantee the consistency and validity of the algorithms. In terms of communication cost, as the trees in a random forest are trained independently there is no need to communicate model updates when running the algorithms in parallel. However, for OnlineQRF in a distributed setting we would need to merge the histograms of each tree to provide a quantile prediction. Through our use of the near-optimal KLL sketch we ensure that the communication cost is minimal. The optimizations listed cover two of the three objectives listed in Section 1.1 and this work investigates an aspect of our research question in the context of uncertainty estimation.

CHAPTER

9

Scalable Boosted Trees for High-dimensional Data

In this chapter we provide an overview of our work on scaling boosted tree learning. We tackle the problem in two different domains: online boosting (Paper V) and batch gradient boosting (Paper VI). We develop algorithms that are optimized for the distributed setting by being communication efficient and exhibiting good scaling characteristics.

9.1 Background

Boosting algorithms are some of the most successful and widely used algorithms in machine learning, due to their simplicity and excellent accuracy. As a result, a wide array of research has been proposed with the aim of improving the scalability of boosting, either through parallelizing the base algorithm, or by developing algorithms that can perform boosting online.

However, when it comes to high dimensional data, current approaches fall short in terms of scalability. In the online domain, due to the sequential nature of online boosting algorithms, current online methods do not make use of parallelism for training [15, 186, 54]. On the other hand, existing methods for the batch domain provide parallelism along only the data point dimension, limiting their scale-out capabilities. As a result, for high-dimensional data with millions of features, we are left with few choices: In the online domain there exist no parallel algorithms, while in the batch domain the existing feature-parallel algorithms make the assumption that the entire dataset can fit in the memory of each worker, severely limiting the size of the problems we can tackle [135, 55].

In our work we tackle each domain separately: In Paper V, we enable parallelism in the online learning setting, while maintaining the guarantees of existing online

boosting algorithms. To achieve that, we use a sequential ensemble of model-parallel learners that train on different parts of a single data point in a distributed environment. In the batch domain we enable scalability across *both* the data point and feature dimensions by developing a block-distributed gradient boosted tree algorithm. We show how to make block-distributed training feasible and communication-efficient.

9.2 Online and Distributed Boosted Trees

In Paper V we make parallel online boosting possible, while maintaining the correctness of the online boosting algorithms, by using model-parallel learning instead of the data-parallel approaches used in the batch setting. We make use of the Vertical Hoeffding Tree (VHT) algorithm [145] which is a model-parallel extension of the online Hoeffding tree algorithm [75], often referred to as Very Fast Decision Tree (VFDT). We call our algorithm BoostVHT.

We provided a description of the VFDT algorithm in Section 5.2. Briefly, the VFDT algorithm will sort instances down the tree and update the statistics for the leaf the instance ends up in. This separation of duties allows for the scalable design of VHT: the algorithm uses two components, one called the *model aggregator* that is responsible for sorting instances to leafs and maintaining the model, and one called the *local statistics*, which are potentially many workers responsible for maintaining the statistics for the leaves of the tree. The model aggregator is responsible for breaking up each instance into its constituent features, and sending them, along with the label, to the local statistics workers. The model aggregator will periodically query the local statistics to check if we have collected enough information to split each leaf with high confidence, as done in the original VFDT.

Optimizations

Our method proposes optimizations that make the learning process efficient in a distributed setting. We use a design that is similar to the original VHT in that we maintain two main components, one model aggregator and local statistics. However, our model aggregator hosts a sequential chain of VHT models, through which we pass each example. This design allows us to maintain the sequential nature of the boosting algorithm, and as a result the guarantees of any online boosting algorithm, while performing the training in parallel and asynchronously in a set of local statistics workers.

We propose three optimizations over the original VHT algorithm to improve its characteristics for distributed boosting. First, we design our local statistics so that the same worker can host statistics from multiple members of the ensemble. This enables to decouple the level of parallelism from the number of learners in the ensemble, separating the algorithm's functional (speedup) and non-functional

(accuracy) aspects. Second, this enables us to host the statistics for a range of features on the same worker, across the ensemble. This makes our communication pattern efficient: We only need to send p messages per instance, where p is the chosen parallelism level, compared to sending m messages, where m the number of features. This will dramatically reduce the number of messages sent, since $m \gg p$ in the high-dimensional settings we are focusing on. In addition, compared to existing parallel boosting methods like AdaBoost.PL [189] that needs to communicate the complete models between each update, we only need to communicate the split decisions which are much more compact (3 floating point values instead of a complete tree data structure). Finally, our approach allows us to send each attribute slice only once to each worker, as it suffices for each subsequent learner after the first to only send their updated weight. This brings a communication reduction of a factor s, where s is the ensemble size, which can often be hundreds of trees [84].

9.3 Block-distributed Gradient Boosted Trees

In Paper VI we extend distributed gradient boosted trees, by parallelizing their training across both the data point and feature dimensions, making use of block-distribution. Here we provide a summary of each part of the training algorithm that we had to adapt to the block-distributed setting. We provided an explanation of the training process for GBTs in Section 4.2.2, and we briefly re-iterate here: The training process can be roughly divided into three stages: First comes the prediction part, in which we use the existing ensemble to make a prediction for each point in the dataset. We use these predictions to get the gradient value for each data point. Second comes the gradient histogram creation. In this stage we use the computed gradients to create gradient histograms for every feature, one for each leaf in the tree. A gradient histogram for a feature contains in each bucket the sum of the gradients that corresponds to that feature range. Finally, given the gradient histograms for each feature, we have a final step of split selection, where we use these histograms to greedily select the optimal split for each leaf.

9.3.1 Block-distributed prediction

In the row-distributed setting, each worker has access to all the features for the horizontal slice of the data they are responsible for. This allows us to determine the exit node for each local data point without any communication. That is not the case however for block-distributed data where each worker only has access to parts of each data point. In this case, when the model contains internal nodes that correspond to features that are not present in one worker, they will need to communicate with the other workers responsible for the same horizontal slice to determine the exit nodes. What we want to avoid in this scenario is shuffling the data between workers, as we could have millions of features leading to impractical communication costs. In our

Index	Feature 1	Feature 2	Feature 3	Gradient
1	13	0	488	1.5
2	7	1	667	2.5
3	3.5	0	122	2
4	1.6	2	366	2.5

Table 9.1: Example dataset.

approach we develop a novel use of the Quickscorer [168] algorithm that gives us provably correct and efficient block-distributed predictions.

Briefly, in Quickscorer the exit node for a data point dropped down a decision tree can be determined by applying the bit-wise `AND` operation between specific bit-strings for every internal node. These bit-strings are of length $|L|$, where L the set of leaves, and every zero indicates that a leaf node becomes impossible to reach if the node's condition evaluates to false, and all other elements are one. When determining the exit node for a particular example, we need to identify all the nodes in the tree that evaluate to false for that example, and aggregate their bit-strings into one overall bit-string $\mathbf{v}$. Lucchese *et al.* [168] prove that the left-most bit set to one in $\mathbf{v}$ will indicate the exit node for the example.

We provide an illustrative example, re-using the data from Section 4.2.2, which we reproduce in Table 9.1. An example decision tree is given in Figure 9.1, along with the corresponding bit-strings for each internal node. Each bit in the bit-string corresponds to one of the leafs of the tree, with the left-most bit corresponding to the left-most leaf. For example, the bit-string for the root is 0011, because if the root evaluates to false, the two left-most leaves become inaccessible (we move right when a node evaluates to false), but either of the two right-most leaves could still be reached.

Let's take data point 2, which has the feature values $f1 : 7, f2 : 1, f3 : 667$. Using our oracle we can tell that the nodes that evaluate to false are the root, as $7 > 5$, and its right child, as $667 > 500$. Their bit-strings are 0011 and 1101 respectively. To determine the exit node we simply need to perform an `AND` operation between these two bit-strings: $0011 \wedge 1101 = 0001$. The left-most bit set to one is bit 4 (or 3 if we are using 0-indexing), hence the exit node corresponds to the fourth leaf from the left. We can verify that is true by following the tree conditions.

We adapt the above algorithm to work in the block-distributed setting. Each worker now only has access to particular range of features for a subset of the data points. So each worker can only evaluate the conditions of the tree that correspond to its slice of the features. We create local bit-strings at each worker, one for each example. These bit-strings have the same properties as in the original Quikscorer algorithm, however can only eliminate leafs that are children of internal nodes that

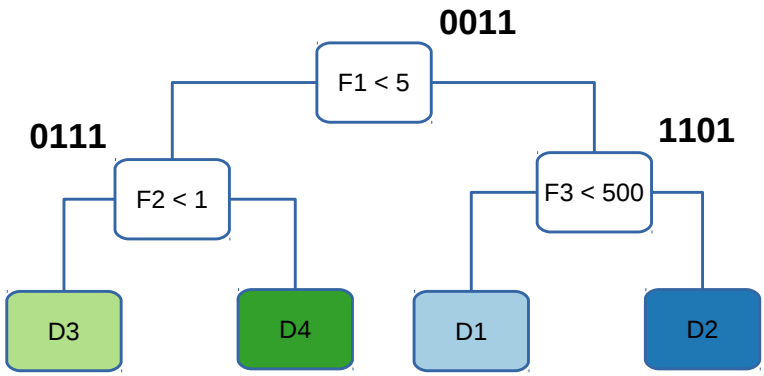


Figure 9.1: Example decision tree, and the end positions of the data points in Table 9.1. When a node condition evaluates to false, we move to the right child of the node. The bitstring zeros indicate the leafs that become unreachable when a node evaluates to false.

correspond to the features available at that worker. To fully determine the output for each data point, the workers that hold data for the same horizontal slice will need to aggregate their bit-strings for each data point. We do this by utilizing the parameter server [158] architecture: Each server is responsible for a one horizontal slice of the data. The workers that hold the different parts of a horizontal slice, all send their local bit-strings to the same server, for each example in their data block. The server then aggregates the partial bit-strings using a simple bit-wise `AND` to determine the exit node for each data point.

In our example, say data point 2 was split between two workers, Worker 1 and Worker 2, with Worker 1 responsible for Features 1 & 2, and Worker 2 responsible for Feature 3. Worker 1 would be able to only evaluate the condition of the root as false and produce the local bistring 0011 and Worker 2 would evaluate the condition involving Feature 3, producing the local bistring 1101. Each worker sends their local bit-strings to the same server, where they are aggregated to produce the final bit-string 0001.

Because the `AND` operator is both commutative and associative, the order in which these aggregations happen does not affect the result, and the overall aggregation will be equal to the aggregation of the partial aggregations. This ensures that our predictions will be correct. By using this method we can determine the exit nodes for each data point, and from that, the corresponding gradients that we need for the next step of histogram aggregation. By only communicating bit-strings and

performing bitwise operations that are hardware optimized we ensure that the communication and computation cost of this step remains low.

9.3.2 Histogram Aggregation

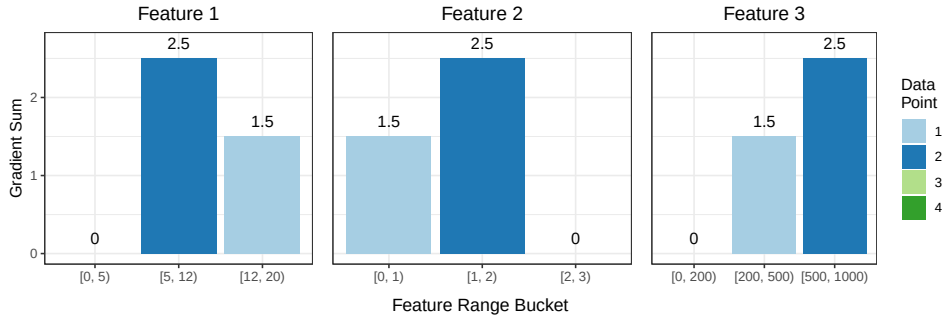
With centralized data the histogram aggregation step requires no communication, and can be performed by running a simple local aggregation. In the example given in Section 4.2.2, we determined the full gradient histograms for all features and data points by aggregating the gradient values for each feature range bucket. In that example we had three buckets per feature, determined by the empirical CDF of each feature, which we populated by iterating through each feature value in the current leaf partition. The resulting full histogram was shown in Figure 4.3. We again use the data from Table 9.1.

Now assume that we have a row-distributed dataset, where we assign data points 1 and 2 to Worker 1, and data points 3 and 4 to Worker 2. To get a complete picture of the gradient histograms as shown in Figure 4.3, we need to first create local histograms at each worker, and then aggregate them between the workers, meaning that the the algorithm involves one communication step.

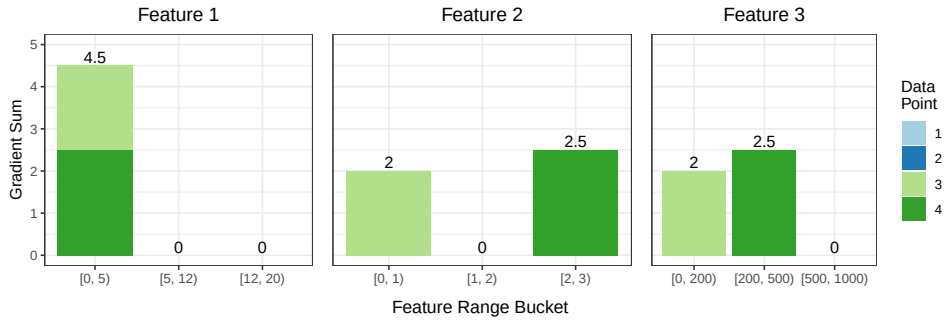
One drawback of all current row-distributed methods is that they use dense communication for the aggregation we just described [55, 135, 195, 126]. Dense communication requires us to communicate $|\mathsf{F}| \cdot \mathsf{B}$ values for each worker, where F the set of features, and B the number of histogram buckets. However, many of the histogram values can be zero as can be seen in Figure 9.2, where we show the local gradient histograms for each worker. This results in redundant communication when a dense communication pattern is used. This problem is exacerbated when we are dealing with sparse datasets with millions of features.

Block-distribution introduces another level of complexity into this operation, as in that case, no worker has a complete view of any data point. Each worker gets one block, that is, a horizontal *and* vertical slice of the data. If we think of the complete gradient histograms as a tensor of dimensions $|\mathsf{L}| \times |\mathsf{F}| \times \mathsf{B}$, each worker will populate a part of this tensor, across the L (leafs) and B (buckets) dimensions, but limited to a specific subset of F (features). In Paper VI, we use a sparse representation of this tensor at each worker which we then send to the servers, which allows us to eliminate the extraneous communication described above. Our experiments show that for sparse datasets this brings the communication cost down by multiple orders of magnitude.

Once each worker iterates through their block of data, they send their partial tensor to one server. Each server is now responsible for a range of features, so workers that belong to the same *vertical* slice of data will send their tensors to the same server. Each server ends up having a full view of the gradient histograms for a subset of features. The split finding step can then be performed using an algorithm similar to the one proposed in Jiang *et al.* [126].



(a) Local gradient histogram for Worker 1.



(b) Local gradient histogram for Worker 2.

Figure 9.2: Local gradient histograms for row-distributed data. Note the existence of multiple zero values that would nonetheless need to be communicated using a dense communication pattern like MPI allreduce.

9.4 Main Findings

In this section we present the evaluation of the online distributed tree boosting algorithm from Paper V and the block-distributed GBT algorithm from Paper VI.

Online and Distributed Boosted Trees

For our online learning experiments of BoostVHT we use 10 artificial and 4 real-world datasets. The artificial datasets are generated with different numbers of attributes using one generator from a tree-like structure, one that generates random tweet text for a sentiment analysis task, and one rotating hyperplane. The baselines we test against are a single-threaded online boosting algorithm from the MOA [26] library using the Hoeffding tree as a base learner, and the base VHT tree. We measure the speedup over the single-threaded algorithm, and use the robust Kappa [22] statistic to measure accuracy, that considers the probability of agreement

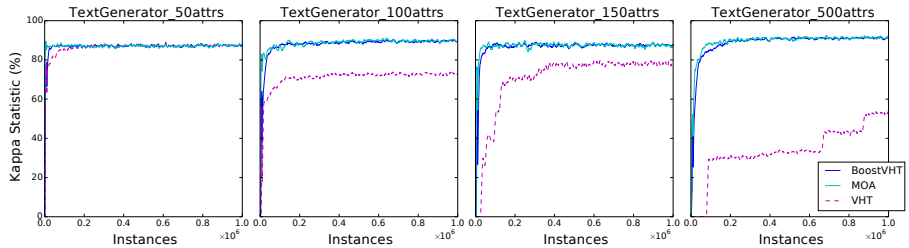


Figure 9.3: Kappa statistic (accuracy) as a function of arriving instances over time for text generator datasets with an increasing number of attributes.

by chance in the evaluation.

We start by showing that the boosted version of the algorithm performs better than the base VHT. Figure 9.3 shows an example experiment using different levels of dimensionality for the text-generator data. We can see that as we increase the number of features and make the problem more difficult, the accuracy of the VHT algorithm deteriorates, while BoostVHT is consistently accurate, and is practically equivalent to the single-threaded boosting algorithm. At the same time, the concurrent and parallel design of the algorithm provides us with a speedup of $\times$**45** on average, compared to the single-threaded implementation of MOA (shown in Table 2 of Paper V).

Finally we examine the scalability of the algorithm in terms of strong and weak scaling. Ideal strong scaling provides *linear speed-up*: Doubling the amount of resources while keeping the problem size constant should halve the execution time. Ideal weak scaling should have *linear scale-out*: doubling both the problem size and resources at the same time should not affect the training time significantly. We illustrate the scaling characteristics of the algorithm in Figures 9.4 and 9.5 for weak and strong scaling respectively. As we can see in the figures, the algorithm scales almost ideally both in terms of weak scaling, keeping its training time close to constant as we double resources and the problem size, and provides near-linear speed-up for our strong scaling experiments.

Block-distributed Gradient Boosted Trees

To determine the performance gains and limitations of our block-distributed approach and the sparse communication we employ, over the dense row-distributed approach, we use 4 datasets with different sparsity levels. First we have two highly sparse datasets, URL and avazu with roughly 3,2 million and 1 million features respectively. Second, we use the RCV1 dataset with approximately 47 thousand features and the dense Bosch dataset with 968 features. We implement both approaches from scratch in C++ to ensure a fair comparison. We use 12 workers for

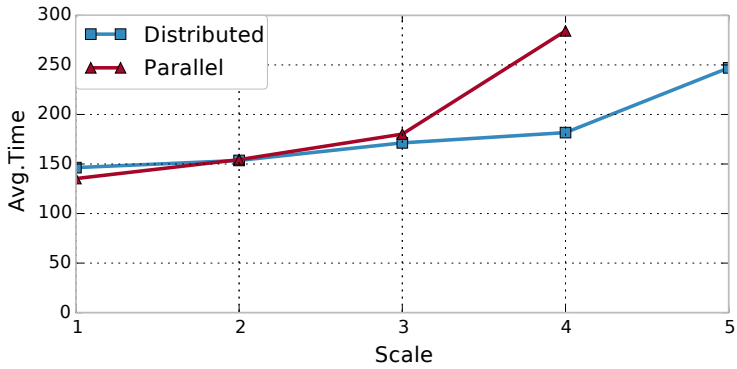
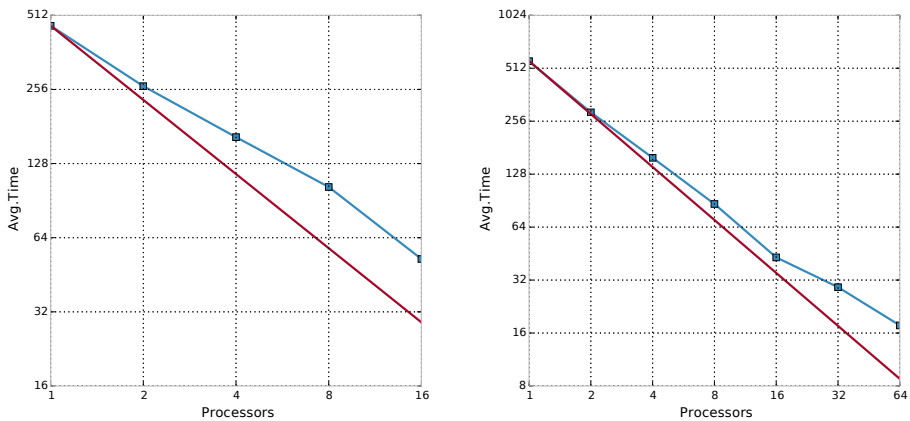


Figure 9.4: Weak scaling experiments, time in milliseconds. Scale 1x on the x-axis refers to 500 attributes with 2 Processors, and we double both Processors and attributes for each scale increment (up to 8,000 attributes with 32 Processors).



(a) Parallel experiments.

(b) Distributed experiments

Figure 9.5: Strong scaling in the parallel (left) and distributed (right) setting. The time reported is the average time to train 1,000 instances, each with 1,000 attributes, in milliseconds. The red straight line indicates ideal linear scaling.

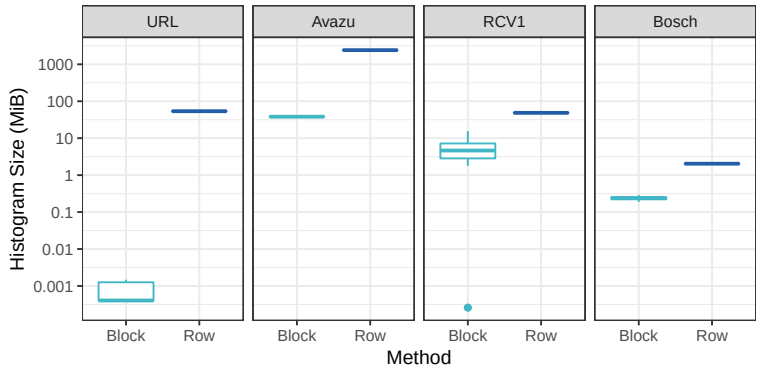


Figure 9.6: The byte size of the gradient histograms being communicated for the various datasets.

the row-distributed approach, and 9 workers and 3 servers for the block-distributed approach.

We measure the communication cost in MiB for the dense histograms of the row-distributed approach and our sparse block-distributed histograms. We also evaluate the end-to-end runtime of the histogram aggregation step, divided into communication and computation stages, to measure the real-world runtime. This step is the most computationally intensive part of GBT learning [176] and in our experiments constitutes at least 90% of the total runtime of the training.

In Figure 9.6 we can see the benefits of sparse communication: For the highly sparse URL and avazu datasets the reduction in communication is five and two orders of magnitude respectively. The sparse histograms however introduce additional computational cost: unlike dense data structures that are contiguous in memory, sparse structures use indirect addressing which makes building and accessing these histograms cache-unfriendly. This, in combination with the overhead introduced by the use of the parameter server, leads to increased computation time. While in the sparse datasets the significant gains made by minimizing the communication time allows us to compensate for the increased computation, that is not the case for the dense data, where computation time dominates, as shown in Figure 9.7.

Finally we demonstrate the communication savings possible in terms of the feature sketches. As we mentioned in Section 4.2.2, in order to determine the possible split points for each feature, we need to have an estimate of the CDF for each feature. These can be calculated once at the beginning of learning, or once per leaf, to get a more accurate representation of the CDF in every leaf. Chen and Guestrin [55] show that generating per-leaf (local) split points leads to the same overall accuracy as using the overall sketches (global) while using much less accurate sketches, thereby creating potential communication savings: The size of

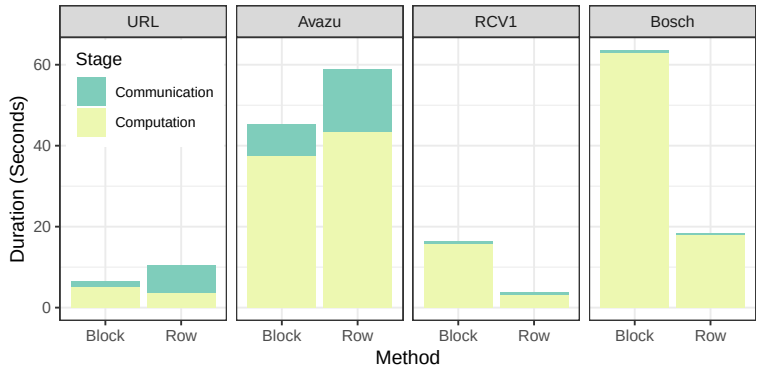


Figure 9.7: The communication and computation times for the various datasets, in seconds.

a quantile sketch is determined by the level of error in the approximation we can accept [133], and the values we populate it with.

However, when using dense communication we need to know the size of the sketch being communicated in advance. In a probabilistic sketch, this is only possible if we use the maximum theoretical size of a sketch, to ensure the sketches do not overlap in memory. This of course is a major source of inefficiency, as the actual size of the sketches can be orders of magnitude smaller. For this reason current distributed approaches all use the global sketch approach, in order to only communicate the sketches once at the beginning of learning instead of having to communicate once for every leaf.

Using our sparse approach however, allows us to only communicate the true size of the sketches, thereby providing massive savings in communication, as shown in Figure 9.8. This would allow the sketches to be communicated for every leaf, leading to even further savings by using lower accuracy sketches or increased accuracy in the CDF representation.

9.5 Discussion

In this chapter we have presented our work on scalable boosted trees. We first developed a model-parallel online algorithm with linear scaling characteristics and demonstrated its accuracy advantages over single trees and significant runtime gains compared to serial tree ensembles. We have used efficient communication to dramatically reduce the number of messages sent.

Second, we developed a block-distributed gradient boosted tree algorithm. Apart from enabling scale-out across both the feature and data point dimensions, we took advantage of data sparsity to reduce the communication necessary for

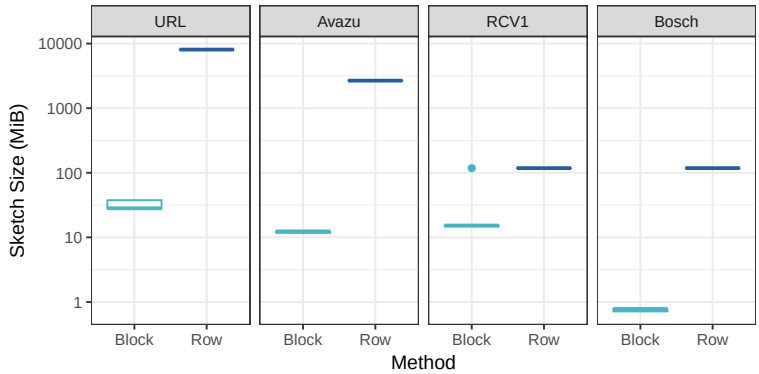


Figure 9.8: The byte size of the feature sketches being communicated for the various datasets.

training by several orders of magnitude.

Thus, these studies demonstrated how algorithmic co-design with distributed systems can lead to optimized runtimes and communication savings, aligning this work with our objectives and research question defined in Section 1.1. We note however that the benefit of both approaches is limited when the dimensionality of the data is low.

Part III

Concluding Remarks

CHAPTER

10

Conclusion

In this dissertation we have proposed efficient algorithms for graph node similarity and decision tree learning. Our goal was to create algorithms that scale out regardless of input size and to do so we set three design goals: reduce the amount of computation necessary to produce a result, reduce the amount of communication during distributed learning, and bound the memory use of the algorithms, making it possible to train them on unbounded datasets.

In this chapter we conclude this dissertation, by first providing a summary and critical view of the results, and putting our research in context with the state-of-the-art. We close the chapter with potential future work directions and open problems.

10.1 Summary of Results

We first proposed an approximate way to calculate similarities between nodes in a graph, and discover concepts in the derived similarity graph. Our method reduces the computational cost by localizing computation in the nodes' neighborhoods, and taking advantage of the sparse graph structure to further optimize the computation with controllable error. We demonstrated the generality of the approach in several domains and presented qualitative as well as quantitative evidence of its accuracy. In addition, we demonstrate the scalability of the algorithm by training on one of the biggest text datasets available in minutes. Compared to other global approaches like SimRank [125] the computational cost is linear in the number of nodes in the graph compared to cubic complexity for SimRank. Compared to local approaches like simply taking the Jaccard coefficient between nodes, our approach has controllable error, is designed from the ground up with the distributed setting in mind, and we propose a concept discovery step on top of the similarity calculation.

We then presented a real-world use case of music streaming session length analysis and prediction, which demonstrated the unique characteristics of music streaming listeners, and developed an appropriate predictive model. This work acts as a motivation for our follow-up work on decision tree learning.

For our first contributions to decision tree learning, we presented two algorithms to extract uncertainty estimates for the predictions of ensembles of online regression trees. We use approximate data structures and online learners to adapt two batch algorithms to the online domain, bounding their computational and memory use and making it possible to train the algorithms on unbounded streaming data. We demonstrate the favorable performance of the algorithm against the state of the art in terms of accuracy, and an order of magnitude improvement in the runtime.

Our next contribution again deals with online decision tree learning, where we develop an algorithm to learn boosted online decision trees in parallel, achieving significant learning speedups while maintaining the correctness guarantees of the underlying online boosting algorithms. Our approach scales almost linearly, for both weak and strong scalability and we clearly demonstrate the gains in terms of accuracy over single trees and speed compared to single-threaded approaches.

Finally we presented a new algorithm for distributed gradient boosted tree learning that enables a new dimension of scalability, allowing for scale-out across both the data point and feature dimensions. We make use of the Quickscorer algorithm to achieve efficient block-distributed prediction, and make use of the parameter server's flexibility to exploit data sparsity and achieve order of magnitude improvements in the communication cost of the training process.

10.2 Discussion

We discuss here the extent to which this work answers the research question posited for our thesis which we reproduce here:

> *How can we use approximations and parallelism to develop scalable learning algorithms?*

This dissertation has tackled this question by:

- Using approximate solutions and bounded error subsampling of edges to produce a vertex similarity algorithm with a scalable distributed implementation.
- Using approximate data structures and online learning methods to enable the estimation of online random forest uncertainty. The approximations made trade off convergence guarantees for efficient computation.
- Using distributed computation to tackle high-dimensional problems for boosted decision trees, maintaining the correctness of the algorithms while

providing significant speedups. We make clear the limitations of the approaches for low dimensional and dense data.

10.3 Future Directions

We identify several aspects of our work that can be expanded upon and improved, targeting specific limitations of the model. For our work on graph node similarity our next targets are to create hierarchical structures of concepts, thereby creating composable representations in an unsupervised manner. The main challenge for this task is again the computational cost, as we would need to maintain information about the role of each node at each hierarchy level as we propagate the relationships. Another interesting problem is developing a generalized algorithm for evaluating graph node similarity from a stream of edges, that form the graph in real time. Finally the calculation of similarities between nodes in the graph that are more than two hops away is currently, by design, not included in the model. However, there could be a way to calculate these by following paths in the resulting similarity graph.

For our work on the estimation of uncertainty in online decision trees, our foremost priority would be the theoretical justification of the validity of the produced intervals, i.e. to prove that, like the batch QRF and conformal prediction, the produced intervals are consistent estimators given the requested significance level. The ability to deal with concept drift at the meta-algorithm level would also be welcome, instead of relying on the underlying weak learner to provide this capability.

Our work on parallel boosted online trees has the drawback that datasets with a small number of features will limit the potential speedup of the algorithm. An interesting direction to explore is to create efficient data-parallel or block-parallel algorithms that might sacrifice the correctness guarantees of our feature-parallel algorithm for more available parallelism.

Finally for our work on block-distributed gradient boosted trees one priority is to improve the algorithm's performance for datasets with low sparsity. The overhead introduced by the sparse representations and distributed communication systems we use could be mitigated by, for example, choosing the representation to use (sparse or dense) based on the characteristics of the data, which can be determined in the initial pass over the data during the creation of the feature value histograms. In addition we would like to integrate the approach in an end-to-end learner like XGBoost to be able to compare its performance with other state-of-the-art systems like LightGBM.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=3026877.3026899`.

[2] Lada A. Adamic and Eytan Adar. 2001. Friends and neighbors on the web. *Social Networks*, 25:211–230.

[3] Alekh Agarwal, Oliveier Chapelle, Miroslav Dudík, and John Langford. 2014. A reliable effective terascale linear learning system. *Journal of Machine Learning Research*, 15:1111–1133. URL `http://jmlr.org/papers/v15/agarwal14a.html`.

[4] Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97.

[5] Ezilda Almeida, Carlos Ferreira, and João Gama. 2013. Adaptive model rules from data streams. In *Machine Learning and Knowledge Discovery in Databases*, pages 480–492. Springer Berlin Heidelberg.

[6] M. A. Alvarez and S. Lim. 2007. A graph modeling of semantic similarity between words. In *International Conference on Semantic Computing (ICSC 2007)*, pages 355–362.

[7] Yali Amit and Donald Geman. 1997. Shape quantization and recognition with randomized trees. *Neural Comput.*, 9(7):1545–1588. ISSN 0899-7667. URL `http://dx.doi.org/10.1162/neco.1997.9.7.1545`.

[8] Yossi Arjevani and Ohad Shamir. 2015. Communication complexity of distributed convex learning and optimization. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1756–1764. Curran Associates, Inc. URL `http://papers.nips.cc/paper/5731-communication-complexity-of-distributed-convex-learning-and-optimization.pdf`.

[9] N. Asadi, J. Lin, and A. P. de Vries. 2014. Runtime optimizations for tree-based machine learning models. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2281–2292. ISSN 1041-4347.

[10] B. Babenko, M. Yang, and S. Belongie. 2009. Visual tracking with online multiple instance learning. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 983–990.

[11] Nicola Barbieri, Fabrizio Silvestri, and Mounia Lalmas. 2016. Improving post-click user engagement on native ads via survival analysis. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 761–770, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4143-1. URL `http://dx.doi.org/10.1145/2872427.2883092`.

[12] Ron Bekkerman, Mikhail Bilenko, and John Langford. 2011. *Scaling up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press.

[13] Yael Ben-Haim and Elad Yom-Tov. 2010. A streaming parallel decision tree algorithm. *Journal of Machine Learning Research*, 11:849–872.

[14] András A. Benczúr, Levente Kocsis, and Róbert Pálovics. 2018. Online machine learning in big data streams. *CoRR*, abs/1802.05872.

[15] Alina Beygelzimer, Elad Hazan, Satyen Kale, and Haipeng Luo. 2015. Online gradient boosting. In *Advances in Neural Information Processing Systems 28*, pages 2458–2466. Curran Associates, Inc.

[16] Alina Beygelzimer, Satyen Kale, and Haipeng Luo. 2015. Optimal and adaptive algorithms for online boosting. In *ICML*, volume 37, pages 2323–2331.

[17] Zhiqiang Bi, Christos Faloutsos, and Flip Korn. 2001. The "dgx" distribution for mining massive, skewed data. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 17–26, New York, NY, USA. ACM. ISBN 1-58113-391-X. URL `http://doi.acm.org/10.1145/502512.502521`.

[18] Gérard Biau and Benoît Cadre. 2017. Optimization by gradient boosting. *arXiv e-prints*, page arXiv:1707.05023.

[19] Gérard Biau, Benoît Cadre, and Laurent Rouvière. 2018. Accelerated Gradient Boosting. *arXiv e-prints*, page arXiv:1803.02042.

[20] Gérard Biau and Erwan Scornet. 2015. A Random Forest Guided Tour. *arXiv e-prints*, page arXiv:1511.05741.

[21] Albert Bifet, Gianmarco de Francisci Morales, Jesse Read, Geoff Holmes, and Bernhard Pfahringer. 2015. Efficient online evaluation of big data stream classifiers. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 59–68, New York, NY, USA. ACM. ISBN 978-1-4503-3664-2. URL http://doi.acm.org/10.1145/2783258.2783372.

[22] Albert Bifet, Gianmarco De Francisci Morales, Jesse Read, Geoff Holmes, and Bernhard Pfahringer. 2015. Efficient Online Evaluation of Big Data Stream Classifiers. In *KDD*, pages 59–68.

[23] Albert Bifet and Eibe Frank. 2010. Sentiment knowledge discovery in twitter streaming data. In *Proceedings of 13th International Conference On Discovery Science*, pages 1–15. Springer.

[24] Albert Bifet, Ricard Gavaldà, Geoff Holmes, and Bernhard Pfahringer. 2018. *Machine Learning for Data Streams with Practical Examples in MOA*. MIT Press. https://moa.cms.waikato.ac.nz/book/.

[25] Albert Bifet and Ricard Gavaldà. 2007. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 443–448. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611972771.42.

[26] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. MOA: Massive online analysis. *JMLR*, 11:1601–1604.

[27] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. 2010. Leveraging bagging for evolving data streams. In José Luis Balcázar, Francesco Bonchi, Aristides Gionis, and Michèle Sebag, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 135–150, Berlin, Heidelberg. Springer Berlin Heidelberg.

[28] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Eibe Frank. 2010. Fast perceptron decision tree learning from evolving data streams. In *Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*, PAKDD'10, pages 299–310, Berlin, Heidelberg. Springer-Verlag.

[29] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. 2009. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 139–148, New York, NY, USA. ACM. ISBN 978-1-60558-495-9. URL http://doi.acm.org/10.1145/1557019.1557041.

[30] Albert Bifet and Richard Kirkby. 2011. Data Stream Mining: A Practical Approach. Technical report, The University of Waikato.

[31] Albert Bifet, Silviu Maniu, Jianfeng Qian, Guangjian Tian, Cheng He, and Wei Fan. 2015. Streamdm: Advanced data mining in spark streaming. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*, pages 1608–1611. IEEE.

[32] Albert Bifet, Jiajin Zhang, Wei Fan, Cheng He, Jianfeng Zhang, Jianfeng Qian, Geoff Holmes, and Bernhard Pfahringer. 2017. Extremely fast decision tree mining for evolving data streams. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 1733–1742.

[33] Hendrik Blockeel, Luc De Raedt, and Jan Ramon. 1998. Top-down induction of clustering trees. In *Proceedings of the 15th International Conference on Machine Learning*, pages 55–63. Morgan Kaufmann.

[34] Vincent D. Blondel, Anahí Gajardo, Maureen Heymans, Pierre Senellart, and Paul Van Dooren. 2004. A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM Rev.*, 46(4):647–666. ISSN 0036-1445. URL `http://dx.doi.org/10.1137/S0036144502415960`.

[35] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. 2016. A context-aware time model for web search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 205–214, New York, NY, USA. ACM. ISBN 978-1-4503-4069-4. URL `http://doi.acm.org/10.1145/2911451.2911504`.

[36] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

[37] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3 (1):1–122. ISSN 1935-8237.

[38] Joseph K. Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. 2011. Parallel coordinate descent for l1-regularized loss minimization. In *Proceedings of the 28th International Conference on Machine Learning*, ICML'11, pages 321–328, USA. Omnipress. ISBN 978-1-4503-0619-5. URL `http://dl.acm.org/citation.cfm?id=3104482.3104523`.

[39] Leo Breiman. 1996. Bagging predictors. *Machine Learning*, 24(2):123–140.

[40] Leo Breiman. 1998. Arcing classifier (with discussion and a rejoinder by the author). *Ann. Statist.*, 26(3):801–849.

[41] Leo Breiman. 2001. Random forests. *Machine learning*, 45(1):5–32.

[42] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. 1984. *Classification and Regression Trees*. CRC press.

[43] Dariusz Brzezinski and Jerzy Stefanowski. 2017. Prequential auc: properties of the area under the roc curve for data streams with concept drift. *Knowledge and Information Systems*, 52(2):531–562.

[44] Alexander Budanitsky and Graeme Hirst. 2006. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1): 13–47.

[45] Christopher J. Burges, Robert Ragno, and Quoc V. Le. 2007. Learning to rank with nonsmooth cost functions. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 193–200. MIT Press. URL `http://papers.nips.cc/paper/2971-learning-to-rank-with-nonsmooth-cost-functions.pdf`.

[46] Róbert Busa-Fekete and Balázs Kégl. 2010. Fast boosting using adversarial bandits. In *Proceedings of the 27th International Conference on Machine Learning*, ICML'10, pages 143–150, USA. Omnipress.

[47] Christian Cachin, Rachid Guerraoui, and Luis Rodrigues. 2011. *Reliable and Secure Distributed Programming*. Springer.

[48] Ramon F. Cancho and Richard V. Solé. 2001. The small world of human language. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 268(1482):2261–2265.

[49] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4).

[50] Kevin J. Carroll. 2003. On the use and utility of the Weibull model in the analysis of survival data. *Controlled Clinical Trials*, 24(6):682–701. ISSN 0197-2456.

[51] Òscar Celma. 2010. *Music Recommendation and Discovery in the Long Tail*. Springer.

[52] Nitesh V. Chawla, Lawrence O. Hall, Kevin W. Bowyer, and W. Philip Kegelmeyer. 2004. Learning Ensembles from Bites: A Scalable and Accurate Approach. *Journal of Machine Learning Research*, 5:421–451. ISSN 1532-4435.

[53] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. 2013. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005.

[54] Shang-Tse Chen, Hsuan-Tien Lin, and Chi-Jen Lu. 2012. An online boosting algorithm with theoretical justifications. In *ICML*, pages 1873–1880.

[55] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794.

[56] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*.

[57] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, Carlsbad, CA. USENIX Association. ISBN 978-1-931971-47-8. URL `https://www.usenix.org/conference/osdi18/presentation/chen`.

[58] Weizhu Chen, Zhenghao Wang, and Jingren Zhou. 2014. Large-scale l-bfgs using mapreduce. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1332–1340. Curran Associates, Inc.

[59] Kenneth Ward Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29.

[60] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585. ISSN 1532-4435.

[61] Koby Crammer, Jaz Kandola, and Yoram Singer. 2004. Online classification on a budget. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 225–232. MIT Press. URL `http://papers.nips.cc/paper/2385-online-classification-on-a-budget.pdf`.

[62] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. 2012. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends® in Computer Graphics and Vision*, 7(2–3):81–227. ISSN 1572-2740. URL http://dx.doi.org/10.1561/0600000035.

[63] Gianmarco De Francisci Morales and Albert Bifet. 2015. SAMOA: Scalable Advanced Massive Online Analysis. *JMLR*, 16:149–153.

[64] Marcos Lopez De Prado. 2018. *Advances in financial machine learning*. John Wiley & Sons.

[65] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. 2012. Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1223–1231. Curran Associates, Inc. URL http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf.

[66] Jeffrey Dean and Sanjay Ghemawat. 2008. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[67] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 205–220, New York, NY, USA. ACM. ISBN 978-1-59593-591-5. URL http://doi.acm.org/10.1145/1294261.1294281.

[68] Marc Peter Deisenroth and Jun Wei Ng. 2015. Distributed gaussian processes. In *Proceedings of the 32nd International Conference on Machine Learning - Volume 37*, ICML'15, pages 1481–1490. JMLR.org. URL http://dl.acm.org/citation.cfm?id=3045118.3045276.

[69] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. 2012. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13:165–202. ISSN 1532-4435.

[70] Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. 2005. The forgetron: A kernel-based perceptron on a fixed budget. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*, NIPS'05, pages 259–266. MIT Press.

[71] Ramón Díaz-Uriarte and Sara Alvarez de Andrés. 2006. Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 7 (1).

[72] L. R. Dice. 1945. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302.

[73] Thomas G. Dietterich. 2000. Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg. Springer Berlin Heidelberg.

[74] Thomas G. Dietterich and Ghulum Bakiri. 1995. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2(1):263–286. ISSN 1076-9757.

[75] Pedro Domingos and Geoff Hulten. 2000. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80.

[76] Charles Dubout and Francois Fleuret. 2014. Adaptive sampling for large scale boosting. *Journal of Machine Learning Research*, 15:1431–1453. URL `http://jmlr.org/papers/v15/dubout14a.html`.

[77] Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly robust policy evaluation and learning. In *Proceedings of the 28th International Conference on Machine Learning*, ICML'11, pages 1097–1104, USA. Omnipress.

[78] Bradley Efron. 1979. Bootstrap methods: Another look at the jackknife. *Ann. Statist.*, 7(1):1–26.

[79] Gerard Escudero, Lluís Màrquez, and German Rigau. 2000. Boosting applied to word sense disambiguation. In *Proceedings of the 11th European Conference on Machine Learning*, ECML'00, pages 129–141. Springer-Verlag. ISBN 3-540-67602-3, 978-3-540-67602-7.

[80] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. 2019. A guide to deep learning in healthcare. *Nature Medicine*, 25(1):24–29. ISSN 1546-170X. URL `https://doi.org/10.1038/s41591-018-0316-z`.

[81] Stephan Ewen, Sebastian Schelter, Kostas Tzoumas, Daniel Warneke, and Volker Markl. 2013. Iterative parallel data processing with stratosphere: An inside look. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1053–1056, New York, NY,

USA. ACM. ISBN 978-1-4503-2037-5. URL http://doi.acm.org/10.1145/2463676.2463693.

[82] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. 1999. On power-law relationships of the internet topology. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '99, pages 251–262, New York, NY, USA. ACM.

[83] Charles R Farrar and Keith Worden. 2012. *Structural health monitoring: a machine learning perspective*. John Wiley & Sons.

[84] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181. URL http://jmlr.org/papers/v15/delgado14a.html.

[85] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 406–414, New York, NY, USA. ACM.

[86] J. R. Firth. 1957. A synopsis of linguistic theory 1930-55. In *Studies in Linguistic Analysis (special volume of the Philological Society)*, volume 1952-59, pages 1–32. The Philological Society, Oxford.

[87] Peter A. Flach. 2010. *ROC Analysis*, pages 869–875. Springer US, Boston, MA.

[88] F. Fouss, A. Pirotte, J. Renders, and M. Saerens. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3): 355–369. ISSN 1041-4347.

[89] F. Fouss, A. Pirotte, J. Renders, and M. Saerens. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3): 355–369. ISSN 1041-4347.

[90] Yoav Freund. 1990. Boosting a weak learning algorithm by majority. In *COLT*, volume 90, pages 202–216.

[91] Yoav Freund. 1995. Boosting a weak learning algorithm by majority. *Inf. Comput.*, 121(2):256–285.

[92] Yoav Freund and Robert E Schapire. 1995. A desicion-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT*, pages 23–37.

[93] João Gama and Petr Kosina. 2011. Learning decision rules from data streams. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1255–1260. AAAI Press. ISBN 978-1-57735-514-4. URL http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-213.

[94] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. 2009. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 329–338. ACM.

[95] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37. ISSN 0360-0300. URL http://doi.acm.org/10.1145/2523813.

[96] Joao Gama. 2017. Evolving Data, Evolving Models in Economy and Finance. In *MIDAS '17: Second Workshop on Mining Data for Financial Applications*. Keynote.

[97] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. 2013. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346.

[98] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine Learning*, 63(1):3–42.

[99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. ISBN 1-55860-615-7. URL http://dl.acm.org/citation.cfm?id=645925.671516.

[100] Daniel G. Goldstein, R. Preston McAfee, and Siddharth Suri. 2013. The cost of annoying ads. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, pages 459–470, New York, NY, USA. ACM. ISBN 978-1-4503-2035-1. URL http://doi.acm.org/10.1145/2488388.2488429.

[101] Heitor M. Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. 2017. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106:1469–1495.

[102] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. 2017. A survey on ensemble learning for data stream classification. *ACM Computing Survey*, 50(2):23:1–23:36.

[103] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[104] Helmut Grabner and Horst Bischof. 2006. On-line boosting and vision. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, CVPR '06, pages 260–267, Washington, DC, USA. IEEE Computer Society. ISBN 0-7695-2597-0. URL http://dx.doi.org/10.1109/CVPR.2006.215.

[105] Hans Peter Graf, Eric Cosatto, Leon Bottou, Igor Durdanovic, and Vladimir Vapnik. 2004. Parallel support vector machines: The cascade svm. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, NIPS'04, pages 521–528, Cambridge, MA, USA. MIT Press. URL http://dl.acm.org/citation.cfm?id=2976040.2976106.

[106] Michael B. Greenwald and Sanjeev Khanna. 2016. *Quantiles and Equi-depth Histograms over Streams*, pages 45–86. Springer.

[107] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The weka data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18.

[108] Chung-Wei Hang and Munindar P Singh. 2010. Trust-based recommendation based on graph similarity. In *Proceedings of the 13th International Workshop on Trust in Agent Societies (TRUST). Toronto, Canada*, volume 82.

[109] Aaron Harlap, Henggang Cui, Wei Dai, Jinliang Wei, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson, and Eric P. Xing. 2016. Addressing the straggler problem for iterative convergent parallel ml. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, pages 98–111, New York, NY, USA. ACM. ISBN 978-1-4503-4525-5. URL http://doi.acm.org/10.1145/2987550.2987554.

[110] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer series in statistics. Springer. ISBN 9780387848570.

[111] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, ADKDD'14, pages 5:1–5:9.

[112] James Hensman, Nicolò Fusi, and Neil D. Lawrence. 2013. Gaussian processes for big data. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in*

Artificial Intelligence, UAI'13, pages 282–290, Arlington, Virginia, United States. AUAI Press. URL http://dl.acm.org/citation.cfm?id=3023638.3023667.

[113] Shohei Hido, Seiya Tokui, and Satoshi Oda. 2013. Jubatus: An open source platform for distributed online machine learning. In *NIPS 2013 Workshop on Big Learning, Lake Tahoe*.

[114] Qirong Ho, James Cipar, Henggang Cui, Jin Kyu Kim, Seunghak Lee, Phillip B. Gibbons, Garth A. Gibson, Gregory R. Ganger, and Eric P. Xing. 2013. More effective distributed ml via a stale synchronous parallel parameter server. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, pages 1223–1231, USA. Curran Associates Inc.

[115] Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

[116] T. Hoefler and J. L. Traff. 2009. Sparse collective operations for mpi. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–8.

[117] Matthew Hoffman, Francis R. Bach, and David M. Blei. 2010. Online learning for latent dirichlet allocation. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 856–864. Curran Associates, Inc.

[118] Steven C.H. Hoi, Jialei Wang, and Peilin Zhao. 2014. Libol: A library for online learning algorithms. *Journal of Machine Learning Research*, 15:495–499. URL http://jmlr.org/papers/v15/hoi14a.html.

[119] Laurent Hyafil and Ronald L. Rivest. 1976. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5:15–17.

[120] Elena Ikonomovska, João Gama, and Sašo Džeroski. 2011. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23: 128–168.

[121] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA. ACM. ISBN 0-89791-962-9. URL http://doi.acm.org/10.1145/276698.276876.

[122] Hemant Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone, and Michael S. Lauer. 2008. Random survival forests. *Ann. Appl. Stat.*, 2(3):841–860. URL https://doi.org/10.1214/08-AOAS169.

[123] Paul Jaccard. 1912. The distribution of the flora in the alpine zone. *New Phytologist*, 11(2):37–50.

[124] Martin Jaggi, Virginia Smith, Martin Takáč, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I. Jordan. 2014. Communication-efficient distributed dual coordinate ascent. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3068–3076, Cambridge, MA, USA. MIT Press. URL `http://dl.acm.org/citation.cfm?id=2969033.2969169`.

[125] Glen Jeh and Jennifer Widom. 2002. Simrank: A measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 538–543, New York, NY, USA. ACM.

[126] Jiawei Jiang, Bin Cui, Ce Zhang, and Fangcheng Fu. 2018. Dimboost: Boosting gradient boosting decision tree to higher dimensions. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 1363–1376.

[127] Ruoming Jin and Gagan Agrawal. 2003. Efficient decision tree construction on streaming data. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 571–576. ACM.

[128] Ulf Johansson, Henrik Bostrom, and Tuve Lofstrom. 2013. Conformal prediction using decision trees. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 330–339. IEEE.

[129] Ulf Johansson, Henrik Boström, Tuve Löfström, and Henrik Linusson. 2014. Regression conformal prediction with random forests. *Machine Learning*, 97: 155–176.

[130] I. King Jordan, Leonardo Mariño Ramírez, Yuri I. Wolf, and Eugene V. Koonin. 2004. Conservation and Coevolution in the Scale-Free Human Gene Coexpression Network. *Molecular Biology and Evolution*, 21(11):2058–2070.

[131] Vasiliki Kalavri. 2016. *Performance optimization techniques and tools for distributed graph processing*. PhD thesis, KTH Royal Institute of Technology.

[132] Jaz Kandola, Nello Cristianini, and John S. Shawe-taylor. 2003. Learning semantic similarity. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 673–680. MIT Press. URL `http://papers.nips.cc/paper/2316-learning-semantic-similarity.pdf`.

[133] Zohar Karnin, Kevin Lang, and Edo Liberty. 2016. Optimal quantile approximation in streams. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 71–78.

[134] Leo Katz. 1953. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43. URL https://doi.org/10.1007/BF02289026.

[135] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc.

[136] Charles Kemp, Thomas L. Griffiths, Sean Stromsten, and Joshua B. Tenenbaum. 2004. Semi-supervised learning with trees. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 257–264. MIT Press. URL http://papers.nips.cc/paper/2464-semi-supervised-learning-with-trees.pdf.

[137] T. Kim, T. Woodley, B. Stenger, B. Stenger, and R. Cipolla. 2010. Online multiple classifier boosting for object tracking. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, pages 1–6.

[138] Youngho Kim, Ahmed Hassan, Ryen W. White, and Imed Zitouni. 2014. Modeling dwell time to predict click-level satisfaction. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 193–202, New York, NY, USA. ACM. ISBN 978-1-4503-2351-2. URL http://doi.acm.org/10.1145/2556195.2556220.

[139] Jyrki Kivinen, Alex J. Smola, and Robert C. Williamson. 2001. Online learning with kernels. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 785–792.

[140] David G. Kleinbaum and Mitchel Klein. 2012. *Survival Analysis: A self-learning approach*. Statistics for Biology and Health. Springer. ISBN 978-0-387-23918-7.

[141] Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael I. Jordan. 2012. The big data bootstrap. In *Proceedings of the 29th International Conference on Machine Learning*, ICML'12, pages 1787–1794.

[142] Roger Koenker. 1996. *Quantile Regression*. Cambridge University Press.

[143] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37. ISSN 0018-9162.

[144] Petr Kosina and João Gama. 2012. Handling time changing data with adaptive very fast decision rules. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 827–842. Springer Berlin Heidelberg.

[145] N. Kourtellis, G. De Francisci Morales, A. Bifet, and A. Murdopo. 2016. VHT: Vertical Hoeffding Tree. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 915–922.

[146] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. 2018. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv:1811.00982*.

[147] Eduardo Sany Laber and Loana Tito Nogueira. 2004. On the hardness of the minimum height decision tree problem. *Discrete Applied Mathematics*, 144: 209–212.

[148] Balaji Lakshminarayanan, Daniel M Roy, and Yee Whye Teh. 2014. Mondrian Forests: Efficient Online Random Forests. In *Advances in Neural Information Processing Systems 27*, pages 3140–3148.

[149] Balaji Lakshminarayanan, Daniel M. Roy, and Yee Whye Teh. 2016. Mondrian Forests for Large-Scale Regression when Uncertainty Matters. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51, pages 1478–1487.

[150] Mounia Lalmas, Janette Lehmann, Guy Shaked, Fabrizio Silvestri, and Gabriele Tolomei. 2015. Promoting positive post-click experience for in-stream Yahoo Gemini users. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1929–1938, New York, NY, USA. ACM. ISBN 978-1-4503-3664-2. URL `http://doi.acm.org/10.1145/2783258.2788581`.

[151] Trung Le, Tu Dinh Nguyen, Vu Nguyen, and Dinh Q. Phung. 2017. Approximation vector machines for large-scale online learning. *Journal of Machine Learning Research*, 18:111:1–111:55.

[152] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. ISSN 0018-9219.

[153] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris S. Papailiopoulos, and Kannan Ramchandran. 2015. Speeding up distributed machine

learning using codes. *CoRR*, abs/1512.02673. URL http://arxiv.org/abs/1512.02673.

[154] Seunghak Lee, Jin Kyu Kim, Xun Zheng, Qirong Ho, Garth A Gibson, and Eric P Xing. 2014. On model parallelization and scheduling strategies for distributed machine learning. In *Advances in Neural Information Processing Systems 27*, pages 2834–2842. Curran Associates, Inc.

[155] E. A. Leicht, Petter Holme, and M. E. J. Newman. 2006. Vertex similarity in networks. *Phys. Rev. E*, 73:026120. URL https://link.aps.org/doi/10.1103/PhysRevE.73.026120.

[156] C. Leistner, A. Saffari, P. M. Roth, and H. Bischof. 2009. On robustness of on-line boosting - a competitive study. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1362–1369.

[157] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 583–598, Broomfield, CO. USENIX Association. ISBN 978-1-931971-16-4. URL https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu.

[158] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. 2014. Communication efficient distributed machine learning with the parameter server. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 19–27. Curran Associates, Inc.

[159] Ping Li, Christopher J. C. Burges, and Qiang Wu. 2007. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, pages 897–904.

[160] Konstantinos Liakos, Patrizia Busato, Dimitrios Moshou, Simon Pearson, and Dionysis Bochtis. 2018. Machine learning in agriculture: A review. *Sensors*, 18(8):2674.

[161] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. 2017. Deep gradient compression: Reducing the communication bandwidth for distributed training. *CoRR*, abs/1712.01887. URL http://arxiv.org/abs/1712.01887.

[162] Chao Liu, Ryen W. White, and Susan Dumais. 2010. Understanding web browsing behaviors through Weibull analysis of dwell time. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 379–386, New York, NY, USA. ACM. ISBN 978-1-4503-0153-4. URL http://doi.acm.org/10.1145/1835449.1835513.

[163] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph Hellerstein. 2010. Graphlab: A new framework for parallel machine learning. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, UAI'10, pages 340–349. AUAI Press.

[164] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2012. Distributed graphlab: A framework for machine learning in the cloud. *PVLDB*, 5:716–727.

[165] Haihao Lu, Sai Praneeth Karimireddy, Natalia Ponomareva, and Vahab S. Mirrokni. 2019. Accelerating gradient boosting machine. *CoRR*, abs/1903.08708. URL http://arxiv.org/abs/1903.08708.

[166] Jing Lu, Steven C.H. Hoi, Jialei Wang, Peilin Zhao, and Zhi-Yong Liu. 2016. Large scale online kernel learning. *Journal of Machine Learning Research*, 17 (47):1–43.

[167] Linyuan Lü and Tao Zhou. 2011. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170.

[168] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. 2015. Quickscorer: A fast algorithm to rank documents with additive ensembles of regression trees. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 73–82.

[169] Chaitanya Manapragada, Geoffrey I. Webb, and Mahsa Salehi. 2018. Extremely fast decision tree. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 1953–1962. ACM.

[170] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. 2016. A survey of link prediction in complex networks. *ACM Comput. Surv.*, 49(4):69:1–69:33. ISSN 0360-0300. URL http://doi.acm.org/10.1145/3012704.

[171] Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Weller. 2017. Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*.

[172] Amy McGovern, Kimberly L. Elmore, David John Gagne, Sue Ellen Haupt, Christopher D. Karstens, Ryan Lagerquist, Travis Smith, and John K. Williams. 2017. Using artificial intelligence to improve real-time decision-making for high-impact weather. *Bulletin of the American Meteorological Society*, 98(10): 2073–2090. URL https://doi.org/10.1175/BAMS-D-16-0123.1.

[173] Robert J McQueen, Stephen R Garner, Craig G Nevill-Manning, and Ian H Witten. 1995. Applying machine learning to agricultural data. *Computers and electronics in agriculture*, 12(4):275–293.

[174] Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! but at what cost? In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems*, HOTOS'15, pages 14–14, Berkeley, CA, USA. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2831090.2831104.

[175] Nicolai Meinshausen. 2006. Quantile Regression Forests. *Journal of Machine Learning Research*, 7:983–999.

[176] Qi Meng, Guolin Ke, Taifeng Wang, Wei Chen, Qiwei Ye, Zhi-Ming Ma, and Tie-Yan Liu. 2016. A communication-efficient parallel algorithm for decision tree. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 1279–1287.

[177] Jean-Baptiste Michel, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K. Gray, The Google Books Team, Joseph P. Pickett, Dale Holberg, Dan Clancy, Peter Norvig, Jon Orwant, Steven Pinker, Martin A. Nowak, and Erez Lieberman Aiden. 2010. Quantitative analysis of culture using millions of digitized books. *Science*.

[178] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

[179] George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

[180] Einat Minkov, William W Cohen, and Andrew Y Ng. 2006. Contextual search and name disambiguation in email using graphs. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 27–34. ACM.

[181] Miguel Molina-Solana, María Ros, M Dolores Ruiz, Juan Gómez-Romero, and María J Martín-Bautista. 2017. Data science for building energy management: A review. *Renewable and Sustainable Energy Reviews*, 70:598–609.

[182] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. 2018. Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, 19(72):1–5. URL http://jmlr.org/papers/v19/18-251.html.

[183] H. Mouss, D. Mouss, N. Mouss, and L. Sefouhi. 2004. Test of page-hinckley, an approach for fault detection in an agro-alimentary production system. In *2004 5th Asian Control Conference*, volume 2, pages 815–818.

[184] Yuri Nesterov. 2004. *Introductory Lectures on Convex Optimization*. Springer.

[185] Shadi A Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H Campbell. 2017. Samza: stateful scalable stream processing at linkedin. *Proceedings of the VLDB Endowment*, 10 (12):1634–1645.

[186] Nikunj C. Oza. 2005. Online bagging and boosting. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2340–2345.

[187] Nikunj C. Oza and Stuart Russell. 2001. Experimental comparisons of online and batch versions of bagging and boosting. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 359–364.

[188] E. S. Page. 1954. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115.

[189] Indranil Palit and Chandan K. Reddy. 2012. Scalable and Parallel Boosting with MapReduce. *TKDE*, 24(10):1904–1916.

[190] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. 2010. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, 1(1):19–30. URL https://doi.org/10.1007/s13174-010-0003-x.

[191] Harris Papadopoulos, Kostas Proedrou, Volodya Vovk, and Alex Gammerman. 2002. Inductive confidence machines for regression. In *Proceedings of the 13th European Conference in Machine Learning*, pages 345–356.

[192] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS 2017 Autodiff Workshop*.

[193] Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. 2004. Wordnet::similarity: Measuring the relatedness of concepts. In *Demonstration Papers at HLT-NAACL 2004*, HLT-NAACL–Demonstrations '04, pages 38–41, Stroudsburg, PA, USA. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1614025.1614037.

[194] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics.

[195] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. Catboost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6638–6648. Curran Associates, Inc. URL `http://papers.nips.cc/paper/7898-catboost-unbiased-boosting-with-categorical-features.pdf`.

[196] J. R. Quinlan. 1986. Induction of decision trees. *Machine Learning*, 1(1):81–106.

[197] Ross Quinlan. 1992. Learning with continuous classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348. World Scientific.

[198] Daniel Ramage, Anna N. Rafferty, and Christopher D. Manning. 2009. Random walks for text semantic similarity. In *Proceedings of the 2009 Workshop on Graph-based Methods for Natural Language Processing*, TextGraphs-4, pages 23–31, Stroudsburg, PA, USA. Association for Computational Linguistics. ISBN 978-1-932432-54-1. URL `http://dl.acm.org/citation.cfm?id=1708124.1708131`.

[199] Binoy Ravindran, E. Douglas Jensen, and Peng Li. 2005. On recent advances in time/utility function real-time scheduling and resource management. In *8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 55–60.

[200] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 693–701. Curran Associates, Inc.

[201] Gábor Recski, Eszter Iklódi, Katalin Pajkossy, and Andras Kornai. 2016. Measuring semantic similarity of words using concept networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 193–200. Association for Computational Linguistics. URL `http://aclweb.org/anthology/W16-1622`.

[202] Steffen Rendle, Dennis Fetterly, Eugene J. Shekita, and Bor-yiing Su. 2016. Robust large-scale machine learning in the cloud. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*,

KDD '16, pages 1125–1134, New York, NY, USA. ACM. ISBN 978-1-4503-4232-2. URL http://doi.acm.org/10.1145/2939672.2939790.

[203] Cédric Renggli, Dan Alistarh, and Torsten Hoefler. 2018. SparCML: High-Performance Sparse Communication for Machine Learning. *CoRR*, abs/1802.08021. URL http://arxiv.org/abs/1802.08021.

[204] Peter Richtárik and Martin Takáč. 2016. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1):433–484.

[205] Ronald L. Rivest. 1987. Learning decision lists. *Machine Learning*, 2(3):229–246. ISSN 0885-6125.

[206] Lior Rokach and Oded Maimon. 2014. *Data Mining With Decision Trees: Theory and Applications*, 2nd edition. World Scientific Publishing Co., Inc., River Edge, NJ, USA. ISBN 9789814590075, 981459007X.

[207] F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386.

[208] Nadav Rotem, Jordan Fix, Saleem Abdulrasool, Summer Deng, Roman Dzhabarov, James Hegeman, Roman Levenstein, Bert Maher, Nadathur Satish, Jakob Olesen, Jongsoo Park, Artem Rakhov, and Misha Smelyanskiy. 2018. Glow: Graph lowering compiler techniques for neural networks. *CoRR*, abs/1805.00907. URL http://arxiv.org/abs/1805.00907.

[209] Daniel M Roy and Yee W. Teh. 2009. The Mondrian Process. In *Advances in Neural Information Processing Systems 21*, pages 1377–1384.

[210] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252.

[211] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda. 2014. Decision Trees for Mining Data Streams Based on the Gaussian Approximation. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):108–119.

[212] Leszek Rutkowski, Lena Pietruczuk, Piotr Duda, and Maciej Jaworski. 2013. Decision Trees for Mining Data Streams Based on the McDiarmid's Bound. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1272–1279.

[213] Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof. 2009. On-line random forests. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1393–1400.

[214] Robert E Schapire. 1990. The strength of weak learnability. *Machine learning*, 5(2):197–227.

[215] Rocco A. Servedio. 2001. Smooth boosting and learning with malicious noise. In *Proceedings of the 14th Annual Conference on Computational Learning Theory and and 5th European Conference on Computational Learning Theory*, COLT '01/EuroCOLT '01, pages 473–489. ISBN 3-540-42343-5.

[216] Mukund Seshadri, Sridhar Machiraju, Ashwin Sridharan, Jean Bolot, Christos Faloutsos, and Jure Leskovec. 2008. Mobile call graphs: Beyond power-law and lognormal distributions. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 596–604, New York, NY, USA. ACM. ISBN 978-1-60558-193-4. URL http://doi.acm.org/10.1145/1401890.1401963.

[217] Ammar Shaker and Eyke Hüllermeier. 2012. Iblstreams: a system for instance-based classification and regression on data streams. *Evolving Systems*, 3(4): 235–249.

[218] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. 2007. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 807–814, New York, NY, USA. ACM. ISBN 978-1-59593-793-3. URL http://doi.acm.org/10.1145/1273496.1273598.

[219] Navin Sharma, Pranshu Sharma, David Irwin, and Prashant Shenoy. 2011. Predicting solar generation from weather forecasts using machine learning. In *2011 IEEE international conference on smart grid communications (SmartGridComm)*, pages 528–533. IEEE.

[220] Jamie Shotton, Toby Sharp, Pushmeet Kohli, Sebastian Nowozin, John Winn, and Antonio Criminisi. 2013. Decision jungles: Compact and rich models for classification. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 234–242. Curran Associates, Inc. URL http://papers.nips.cc/paper/5199-decision-jungles-compact-and-rich-models-for-classification.pdf.

[221] Alexander Smola and Shravan Narayanamurthy. 2010. An architecture for parallel topic models. *Proceedings of the VLDB Endowment*, 3(1-2):703–710. ISSN 2150-8097. URL http://dx.doi.org/10.14778/1920841.1920931.

[222] Jeany Son, Ilchae Jung, Kayoung Park, and Bohyung Han. 2015. Tracking-by-segmentation with online gradient boosting decision tree. In *2015 IEEE*

International Conference on Computer Vision, ICCV 2015, pages 3056–3064. URL https://doi.org/10.1109/ICCV.2015.350.

[223] T. Sørensen. 1948. A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. *Biol. Skr.*, 5:1–34.

[224] Mark Steyvers and Joshua B Tenenbaum. 2005. The large-scale structure of semantic networks: statistical analyses and a model of semantic growth. *Cognitive science*, 29(1):41–78.

[225] Andrew S Tanenbaum and Maarten Van Steen. 2007. *Distributed systems: principles and paradigms*. Prentice-Hall.

[226] Xun Tang, Xin Jin, and Tao Yang. 2014. Cache-conscious runtime optimization for ranking ensembles. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 1123–1126, New York, NY, USA. ACM. ISBN 978-1-4503-2257-7. URL http://doi.acm.org/10.1145/2600428.2609525.

[227] Shirin Tavara. 2019. Parallel computing of support vector machines: A survey. *ACM Computing Surveys*, 51(6):123:1–123:38. ISSN 0360-0300. URL http://doi.acm.org/10.1145/3280989.

[228] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of collective communication operations in mpich. *Int. J. High Perform. Comput. Appl.*, 19(1):49–66. ISSN 1094-3420.

[229] Eric J. Topol. 2019. High-performance medicine: the convergence of human and artificial intelligence. *Nature Medicine*, 25(1):44–56. ISSN 1546-170X. URL https://doi.org/10.1038/s41591-018-0300-7.

[230] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, *et al.* 2014. Storm at Twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM.

[231] John N. Tsikiklis and Dimitri P. Bertsekas. 1997. *Parallel and distributed computation*. Athena Scientific.

[232] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations*, 15: 49–60.

[233] Vladimir Vapnik. 1998. *Statistical Learning Theory*. John Wiley & Sons Inc.

[234] Victor Eijkhout with Robert van de Geijn and Edmond Chow. 2011. *Introduction to High Performance Scientific Computing*. lulu.com.

[235] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. 2005. *Algorithmic learning in a random world*. Springer Science & Business Media.

[236] David W Walker and Jack J Dongarra. 1996. Mpi: A standard message passing interface. *Supercomputer*, 12:56–68.

[237] Shuo Wang, Leandro L. Minku, and Xin Yao. 2016. Dealing with multiple classes in online class imbalance learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 2118–2124. AAAI Press. ISBN 978-1-57735-770-4. URL `http://dl.acm.org/citation.cfm?id=3060832.3060917`.

[238] Zhuang Wang and Slobodan Vucetic. 2010. Online passive-aggressive algorithms on a budget. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 908–915.

[239] D. J. Watts and S. H. Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):409–10.

[240] Frank Wood, Cédric Archambeau, Jan Gasthaus, Lancelot James, and Yee Whye Teh. 2009. A stochastic memoizer for sequence data. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1129–1136. ACM.

[241] Jierui Xie, B. K. Szymanski, and Xiaoming Liu. 2011. SLPA: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In *ICDM 2011 Workshop on DMCCI*.

[242] Eric P. Xing, Qirong Ho, Wei Dai, Jin-Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. 2015. Petuum: A new platform for distributed machine learning on big data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1335–1344. ACM. ISBN 978-1-4503-3664-2.

[243] Eric P. Xing, Qirong Ho, Pengtao Xie, and Wei Dai. 2015. Strategies and Principles of Distributed Machine Learning on Big Data. *arXiv e-prints*.

[244] Hanxuan Yang, Ling Shao, Feng Zheng, Liang Wang, and Zhan Song. 2011. Recent advances and trends in visual tracking: A review. *Neurocomputing*, 74 (18):3823–3831. ISSN 0925-2312.

[245] Weiren Yu, Wenjie Zhang, Xuemin Lin, Qing Zhang, and Jiajin Le. 2012. A space and time efficient algorithm for simrank computation. *World Wide Web*, 15(3):327–353. ISSN 1386-145X.

[246] Jinhui Yuan, Fei Gao, Qirong Ho, Wei Dai, Jinliang Wei, Xun Zheng, Eric Po Xing, Tie-Yan Liu, and Wei-Ying Ma. 2015. Lightlda: Big topic models on modest computer clusters. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1351–1361. International World Wide Web Conferences Steering Committee.

[247] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, *et al.* 2016. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65.

[248] B. Zeisl, C. Leistner, A. Saffari, and H. Bischof. 2010. On-line semi-supervised multiple-instance boosting. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1879–1879.

[249] Ruiliang Zhang and James T. Kwok. 2014. Asynchronous distributed admm for consensus optimization. In *Proceedings of the 31st International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1701–II–1709. JMLR.org. URL http://dl.acm.org/citation.cfm?id=3044805.3045082.

[250] H. Zhao and J. Canny. 2014. Kylix: A sparse allreduce for commodity clusters. In *2014 43rd International Conference on Parallel Processing*, pages 273–282.

[251] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. 2009. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630.

[252] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-scale parallel collaborative filtering for the netflix prize. In *International Conference on Algorithmic Applications in Management*, pages 337–348. Springer.

[253] Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, Hang Cui, and Edward Y. Chang. 2008. Parallelizing support vector machines on distributed computers. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 257–264. Curran Associates, Inc. URL http://papers.nips.cc/paper/3202-parallelizing-support-vector-machines-on-distributed-computers.pdf.

[254] Martin Zinkevich, John Langford, and Alex J. Smola. 2009. Slow learners are fast. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 2331–2339. Curran Associates, Inc. URL http://papers.nips.cc/paper/3888-slow-learners-are-fast.pdf.

[255] Martin A. Zinkevich, Markus Weimer, Alex Smola, and Lihong Li. 2010. Parallelized stochastic gradient descent. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume*

2, NIPS'10, pages 2595–2603, USA. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2997046.2997185.

[256] Indre Zliobaite. 2010. Learning under concept drift: an overview. *CoRR*, abs/1010.4784. URL http://arxiv.org/abs/1010.4784.

[257] Indrė Žliobaitė, Albert Bifet, Jesse Read, Bernhard Pfahringer, and Geoff Holmes. 2015. Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Machine Learning*, 98(3):455–482.