UNIVERSITY OF CALIFORNIA
Los Angeles

Support for Scalable Analytics over Databases and Data-Streams

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Nikolay Pavlovich Laptev

2012

ABSTRACT OF THE DISSERTATION

Support for Scalable Analytics over Databases and Data-Streams

by

Nikolay Pavlovich Laptev

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2012

Professor Carlo Zaniolo, Chair

The world's information is doubling every two years, largely due to a tremendous growth of data from blogs, social medias and Internet searches. 'Big Data Analytics' is now recognized as an emerging technology area of great opportunities and technical challenges. Parallel systems, such as those inspired by MapReduce architectures, provide a key technology to cope with those challenges—however they often cannot keep up with the fast-growing size of data and application complexity, nor can they deliver the response times required by data stream applications. In this thesis, therefore, we show that many of said limitations can be overcome by building on classical approximation techniques from statistics to estimate (i) the sample quality and (ii) the required sample size given the user-prescribed accuracy.

To achieve (i) we look into the bootstrap theory. The bootstrap approach, based on resampling, provides a simple way of assessing the quality of an estimate. The bootstrap technique, however, is computationally expensive, thus our first contribution involves making the bootstrap estimation efficient. Following our initial results, we realized that in a distributed environment the cost of transferring the data to independent

processors as well as the cost of computing a single resample can be high for large samples. Furthermore the lack of a scalable support for the popular time-series data was also a problem. For these reasons, we provide an improved bootstrap approach that uses the Bag of Little Bootstraps (BLB) along with other recent advances in bootstrap and time-series theory to provide an effective Hadoop-based implementation for assessing a time-series sample quality.

To achieve (ii) we look into the data complexity and learning theory. Recently it has been shown that the performance of a classifier can be analyzed in terms of the data complexity. We start by analyzing how model complexity can be used to create a scalable pattern matching automaton. We then extend our findings to other algorithms where we explain how problem complexity affects the required sample size for a given machine-learning algorithm and accuracy requirement. We also use the learning theory to estimate the error convergence rate needed for sample size estimation. Our experimental results provide the motivation for further exploring these ideas.

A spectrum of classical data mining tasks and newly developed mining applications are used to validate the effectiveness of the proposed approaches. For example, extensive empirical results on a Twitter dataset show that the proposed techniques provide substantial improvements in processing speeds while placing the user in control of the result accuracy.

The dissertation of Nikolay Pavlovich Laptev is approved.

Junghoo Cho

Mark S. Handcock

Todd Millstein

Carlo Zaniolo, Committee Chair

University of California, Los Angeles

2012

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

To write a dissertation is a mighty undertaking, and I could not have reached the finish line without the influence, advice, and support of many colleagues, friends, and family. My thanks are due first and foremost to my advisor, Carlo Zaniolo, for being willing to take me on as a student, and for being unfailingly generous with his time and his support throughout my graduate career. Carlo provided consistently good advice, helped me situate my work in a broader context, and gave me tremendous freedom in pursuing my ideas, while always ensuring that I was making progress toward some fruitful outcome. I will be forever grateful for his help.

I am deeply obliged to the other members of my dissertation committee for their advice and encouragement. From Mark Handcock, I received positive feedback that helped me keep my motivation strong. Mark Handcock was of great help in working out various statistical issues. I am indebted to John Cho for his pioneering research in database management that showed me that simplicity and clarity are key in research. Finally Todd Millstein's research on programming language technology helped me understand the importance of language design in my project.

The faculty, students, and visitors involved with the Web Information Lab have provided a congenial and stimulating environment in which to pursue my studies. I am particularly indebted to my frequent collaboratorsincluding Kai Zeng, Alexander Shkapsky, Barzan Mozafari, Vincenzo Russo, Hetal Thakkar and many otherswho shared my struggles and helped shape my thoughts. I am also grateful to my other officemates Shi Gao, Hamid Mousavi and Mohan Yang for their warm companionship and many interesting discussions. Finally, I offer my deepest thanks to my siblingAnya-whose love and support has sustained me; to my parents, Olga and Pavel, who instilled in me the love of learning that has fueled all my efforts; and to Faith, who believed in me most when I believed in myself least.

VITA

2006	B.S. (Computer Science), UCSB.
2007	M.A. (Economics), UCSB.
2008	M.S. (Computer Science), UCLA.
Summer 2008	Software Engineer Intern, Citrix.
Summer 2009	Software Engineer Intern, Citrix.
Fall 2009	Teaching Assistant, Computer Science Department, UCLA.
Summer 2010	Research Software Engineer Intern, Teradata.
Summer 2011	Software Engineer Intern, Google.
Summer 2012	Software Engineer Intern, HRL.

CHAPTER 1

Introduction

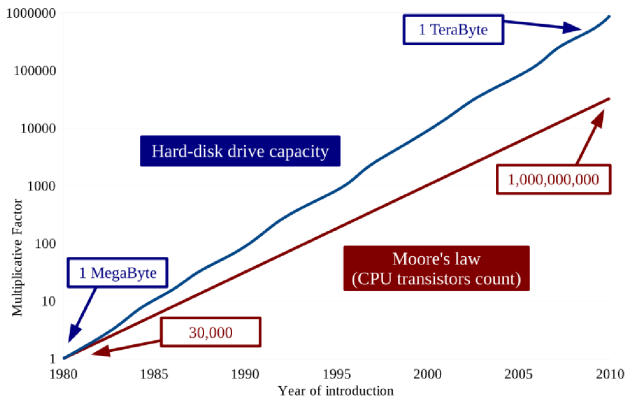
1.1 Problem Motivation

Along with the tremendous growth of data from a fast-expanding set of sources, that include blogs, social medias, Internet searches, and sensor networks[1],we witness a growing demand for more sophisticated analytics on big data, as needed to spot business trends, prevent diseases, and combat crime[2].

'Big Data Analytics' is now universally recognized as a high-tech area of business opportunities into which major companies and a throng of startups are moving aggressively, typically proposing very different systems and solutions to address the various facets of the problem. The difficult task of realizing and deploying advanced analytics in real life applications is complicated by the shear data volume that must be processed to complete various task. Many of these tasks, however, share a common software development cycle that typically begins with (i) writing and testing packages and applications on locally stored data and then (ii) extending them for parallel processing on massive distributed data sets and/or data streams. At each step, the user must work with a different sized set of the data to take advantage of the available

[1] According to the 2011 Digital Universe Study, the world's information is doubling every two years. In 2011 the world will create a staggering 1.8 zettabytes. By 2020 the world will generate 50 times the amount of information and 75 times the number of "information containers" while IT staff to manage it will grow less than 1.5 times (http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf).

[2] Kenneth Cukier: Data, data everywhere. The Economist, 25 February, 2010 (*http://www.economist.com/specialreports/displaystory.cfm?story_id=15557443).*



YouTube	> **45 Terabytes** of videos (early 2007)
Wikipedia	> **13 millions** articles (mid 2009)
Facebook	> **200 millions** active users (mid 2009)
Flickr	> **3 billions** photos (end 2008)
Google	> **1,000 billions** indexed pages (mid 2008)

Figure 1.1: Current data-set sizes are beginning to be impractical for today's machine learning tasks. We provide a way to deal with this problem using simple and robust approximation techniques.

resources while satisfying her time and accuracy constraints. To achieve the above requirement, scalable approximation techniques must be developed that achieve the desirable accuracy goals. As of today, however, no approaches yet exist that are able to fulfill this very ambition requirement.

In this dissertation, we claim that the goal of the scalable approximation support is within the reach of this research project. The approximation support is naturally provided by ideas from statistics and proceeds by: (i) taking a sample S of the data, (ii) running a machine learning algorithm M over S, (iii) estimating the error ξ of the result and (iv) if applicable, increasing the sample size until the required accuracy ϵ is reached. For this approach to work in practice, however, we must develop scalable accuracy and sample size estimation techniques.

Often both time and monetary constraints force the user to work on a small sample of the data for which quality assessment is required to assure a desired result accuracy.

For example a user, when debugging her machine learning algorithm locally, may prefer to work with a sample of the data while remaining confident in the expected accuracy of her final result. When the same user deploys her application on the cloud system such as EC2[3] money and time can also be saved by only working with a sample of the data and using the minimal resources[4]. Unfortunately, deriving fast and accurate estimator assessment is difficult as previously recognized in the literature [KTS12b, ELZ10a]. This obstacle is getting worse with processing power growth failing to match dataset growth thereby resulting in an ever-widening gap as shown in Figure 1.1. Time-series further exacerbates the problem since quality assessment must be carried out in a way that captures the dependence structure of the data generation process.

Early approximate results can also be used in seemingly unrelated domains such as fault tolerance. For instance, in a MapReduce scenario if some of the participants fail or their answers are not returned within a reasonable time, the system might still allow the computation to proceed when the resulting loss of accuracy falls within acceptable limits. In many situations this approach can be preferable to that of repeating the computation, or relying on redundant computations. For example, in situations requiring iterations, such as the computation of transitive closures or K-Means the inaccuracies in early iterations can be tolerated since they tend to be compensated for in later iterations.

In this dissertation we describe techniques that achieve a scalable accuracy assessment with a reliable sample size prediction. We provide a scalable accuracy assessment through an improved bootstrap technique. We also achieve a reliable sample size estimation by using the recent learning theory [HB02, Ho08, Ho04]. Furthermore, we show that the accuracy obtainable by a particular sample size depends on (i) the complexity of the algorithm and on (ii) the intrinsic complexity of the data. Using (i) and

[3] http://aws.amazon.com/ec2/

[4] The pricing of cloud services is based on (a) storage and (b) processing unit usage.

(ii) we can select the best mining algorithm, where 'best' refers to an algorithm requiring least sample size for a given accuracy. As we show in detail in this dissertation, the above methods enable the support for scalable analytics over massive datasets and data-streams.

1.2 Current Techniques and Drawbacks

A crucial step in statistical analysis is to use the given data to estimate the accuracy measure, such as the bias, of a given statistic. In a traditional approach, the accuracy measure is computed via an empirical analog of the explicit theoretical formula derived from a postulated model [ST95]. As is described in chapter 3.1, it is often infeasible to come up with an explicit theoretical formula for result variance of complex analytics, for which reason other nonparametric approaches are needed.

The bootstrap approach [ST95, Efr79] and its time-series variants [BK99, PR94] provide a simple way of assessing the quality of an estimate. The computational cost of the bootstrap, however, is prohibitively large. In the 'Big Data' environment, this can be the source of incessant problems in the natural life-cycle of advanced analytics development and a major stumbling block that prevents interactive result exploration that many users seek [LZZ12a, LK12]. Thus we need a scalable and efficient way of assessing the quality of an estimator for large independently and identically distributed (i.i.d.) and time-series data samples in a distributed environment.

The error convergence rate with respect to the size of the sample is required to estimate the needed sample size. A crucial result in this regard is the central limit theorem (CLT) which shows that for any random sample from any probability distribution with finite mean and finite variance the sample distribution converges to a normal distribution $\mathcal{N}(0, 1)$. The CLT, however, does not say anything about the convergence rate,

requiring us to look for other approaches in modeling accuracy prediction.

While the above represents a very narrow, yet largely sufficient overview of the state of the art, a full literature review on systems, sampling and approximation techniques can be found in chapter 2.

1.3 Quality Assessment

Our initial attempt to improve the original bootstrap and make it more scalable resulted in the development of the Early Accurate Result Library (EARL), which made the bootstrap approach faster and easier to use. The EARL system uses Hadoop and thus takes advantage of the MapReduce framework to compute the resamples in parallel.

EARL, however, also has scalability limits for very large data-sets and needs further extension to support time-series. Therefore, to provide a scalable approach for assessing the quality of large time-series data samples we build on recent advances in bootstrap [LZZ12a, KTS12b, KTS11] and in time-series theory [BK99, PR94]. We introduce the BLB-TS framework that takes advantage of the trend in computational resources which is shifting towards a multicore and distributed architecture with cloud services providing thousands of processing units[5]. The core of BLB-TS relies on (i) the Bag of Little Bootstraps (BLB) [KTS12b] for bootstrap scalability, (ii) on the Stationary Bootstrap (SB) [PR94] for time-series resampling support and (iii) on Hadoop[6] for computational scalability. BLB-TS works by first selecting in parallel s random blocks of the original time-series data sample of size n. Then in parallel r resamples are generated by applying SB to each block b to obtain a resample of size n. The errors computed on the resamples are then averaged across all nodes to produce the final

[5]The total number of processing units in a cluster is: number of nodes $\times$ the number of CPUs per node.

[6]hadoop.apache.org

accuracy estimate. Thus, in BLB-TS, the estimator is applied on only a single block b instead of on the entire time-series sample. In other words, the computational cost incurred only depends on the size of b and not on the size of the sample n. Furthermore, BLB-TS varies r independently on each node to achieve a faster convergence rate.

1.4 Sample Size Estimation

To estimate the required sample size, we provide a new framework called SS-TS, which works by iteratively constructing a learning curve (LC), where LC denotes a function of accuracy in terms of the sample size. For a wide range of machine learning (ML) algorithms, including classification and association, the shape of an LC follows a power law [MTH02]. Thus, we can estimate the parameters of the power law curve by computing only k points, where k is typically less than five. Therefore we execute our association finding algorithm on k different sample sizes and estimate the convergence rate which is then used to predict the sample size for the required accuracy.

In addition to the learning curve, we present techniques for leveraging data complexity theory to pick the appropriate association finding algorithm given the problem complexity. Recently it has been shown that the performance of a classifier can be analyzed in terms of the data complexity [HB02, Ho08, Ho04]. We apply this idea to the association complexity to explain how relationship complexity affects the required sample size needed to capture this relationship given a fixed association mining algorithm. We then also apply data complexity theory for pattern matching [LZ12] and dynamically vary the complexity of the pattern automaton to maximize performance. Our experimental results provide the motivation for further exploration of this idea.

1.5 Dissertation Contributions

To the best of our knowledge, this dissertation is the first to provide a scalable approach for sample size estimation over massive datasets. In particular this dissertation makes the following contributions:

1. A scalable, non-parametric accuracy estimation technique for massive datasets and data-streams is introduced. Our technique provides sample quality assessment for general machine learning algorithms.

2. A sample size estimation technique is presented which introduces a new methodology based on the learning curve which can provide a tighter prediction for the sample size needed to achieve the accuracy prescribed by the user.

3. The notion of problem complexity is introduced as a means for problem domain exploration. We show that the best ML algorithm, the algorithm that minimizes the required sample size, can be determined automatically by analyzing the complexity of the data.

4. A robust and extensible Hadoop-based implementation of our techniques is presented in detail, along with extensive experimental results over real-life data.

1.6 Overview of the Dissertation

In this dissertation, we explore a range of approaches to provide a scalable quality and sample size estimation for general ML algorithms.

After presenting a literature overview in chapter 2 we develop a baseline quality assessment algorithm in chapter 3.1. Despite its extreme simplicity, this initial attempt achieves surprisingly good results for estimating quality of i.i.d. samples. Its effective-

ness, however, is limited by its failure to support time-series and large sample sizes. To remedy these shortcomings, in section 3.2, we introduce the improved quality assessment technique that scales and supports time-series.

Next in chapter 4 we consider the problem of sample size estimation given the user prescribed accuracy target. Specifically, we examine the relation between accuracy and sample size also known as the learning curve. We model the accuracy convergence as an inverse power law and use the method of least squares for parameter estimation. Using the quality estimation framework from chapter 3.1 we can also naturally provide the confidence interval of the derived curve.

Having explored sample size estimation given a fixed algorithm, next we seek to deal with the case where multiple competing algorithms are applicable for a given task. Thus in chapter 5 we use recent data complexity theory results in order to select the appropriate ML algorithm for the given problem complexity to minimize the required sample size. Then, in section 5.3 we apply the complexity theory to the pattern matching problem.

In chapter 6.1, we describe the application of our approximation framework for quickly detecting nontrivial associations among millions of variables. Finally in chapter 7 we summarize the contributions of the dissertation, and offer exciting future directions.

CHAPTER 2

Literature Overview

2.1 Systems

MapReduce: The MapReduce (MR) model is becoming increasingly popular for tasks involving large data processing. The programming model adopted by MapReduce was originally inspired by functional programming. In the MR model two main stages, map and reduce, are defined with the following signatures:

$$map : (k_1, v_1) \rightarrow (k_2, list(v_2))$$

$$reduce : (k_2, list(v_2)) \rightarrow (k_3, v_3)$$

where the map function is applied on every tuple (k_1, v_1) and produces a list of intermediate (k_2, v_2) pairs. The reduce function is applied to all intermediate tuples with the same key producing (k_3, v_3) as output.

Hadoop, an open source implementation of the MapReduce framework, leverages Hadoop Distributed File System (HDFS) for distributed storage. HDFS stores file system metadata and application data separately. HDFS stores metadata on a dedicated node, termed the NameNode (other systems, such as the Google File System (GFS) [GGL03] do likewise). The application data is stored on servers termed DataNodes. All communication between the servers is done via TCP protocols. The replication, the file block-partitioning and the logical data-splitting provided by HDFS simplify the sampling techniques discussed in this dissertation.

Recent work has shown the efficacy of using the MapReduce paradigm for data mining and machine learning algorithms [CKL06, ZMH09a, PHB09]. Mahout [mah] is a machine learning library built on top of Hadoop that relies on algorithm-specific driver programs to perform and control iteration. HaLoop [BHB10a] is a modified version of Hadoop with native support for iteration. HaLoop efficiently handles data that is invariant across iterations through caching and efficient task scheduling. Similarly, Twister [ELZ10b] is also an iterative MapReduce system. The authors of [LMD11] proposed an optimized version of MapReduce suitable for incremental one-pass analytics that performs fast in-memory processing utilizing new hashing techniques. PrIter [ZGG11] is a modified version of Hadoop MapReduce framework designed around the notion of prioritized iterative computation that supports a large collection of iterative algorithms, including page-rank and shortest path. As part of our future work, we plan on taking advantage of the above systems to optimize portions of our framework that require iteration.

To alleviate the burden on the end-user who has to deal with the many low-level intricacies of programming against MapReduce systems' APIs, projects such as SCOPE [CJL08], Hive [hiv, TSJ09, TSJ10], Pig [GNC09], DryadLINQ [YIF08], and Jaql [BEG11] have been proposed to compile a higher-level programming language into jobs for the target system. For example in [LK12], the authors describe how acyclic machine learning workflows can be incorporated into Twitter's Hadoop and Pig [ORS08a]-based analytics ecosystem through the use of Pig scripts for tasks such as data sampling, feature generation, training, and testing. Hive, a data warehouse system, compiles queries written in HiveQL, a subset of SQL that also allows for the specification of map and reduce tasks within queries, into a DAG of MapReduce jobs which are executed on Hadoop. Pig compiles queries written in Pig Latin [ORS08b], which draws from both declarative and procedural programming, into MapReduce jobs which are also executed on Hadoop. With DryadLINQ, users can write programs composed of

sequential statements of an extended version of LINQ [LIN], which are compiled and executed over Dryad [IBY07]. Jaql provides a scripting language and a flexible data model that is inspired by JSON. Jaql scripts are also compiled into MapReduce jobs which can then be executed over Hadoop.

Parallel databases were first pioneered by Teradata [ter85] and the GAMMA [DGG86] project, which, in mid-1980s, introduced a new architecture based on a cluster of shared-nothing computers that communicate through high-speed interconnects. Recently, hybrid MapReduce/parallel-DBMS solutions seek to leverage the benefits of both platforms. For instance, HadoopDB [ABA09] is a hybrid MapReduce-parallel DBMS system that uses Hadoop as the communication layer and a PostgreSQL instance at each node for query answering. Queries are written in SQL, which are translated to MapReduce jobs through an extended version of Hive [hiv]. As an alternative, the SQL/MR framework [FPC09] presents an API that allows specification of map-like and reduce-like user defined functions which are executed in parallel over each node of the parallel DBMS. UDFs can be written in a variety of programming languages and packaged in libraries.

Approximation integration into the above efforts is unfortunately very primitive or almost non-existent.

Scalable Data Stream Processing: The data stream research community has proposed several distributed stream processing systems. Borealis [AAB05, ABc05] is a distributed data stream processor that provides fault tolerance and load-balancing. In Borealis, partitioning is used to parallelize expensive operators. Similarly, partitioned parallelism is used in the FLuX extension [SHB04] to TelegraphCQ and in the D-Cape system [LZJ05] for high-availability and load balancing, respectively. The SPADE declarative stream processing engine [GAW08] of IBM's System S, a large scale distributed data stream processing middleware, supports parallel and distributed

data flows which the user can express using the SPADE language constructs for looping and partitioning.

The S4 system [NRN10, S4] is a scalable stream processing platform that provides an API for users to implement processing elements using a high-level language such as Java. S4 is distributed with no single point of failure and throughput scales linearly as nodes are added.

The Hadoop Online Prototype [CCA10a] moves toward support for continuous query processing in a MapReduce-like framework, without system features such as load shedding and support for slides and windows. The work described in [LMD11] improves on the bottlenecks found in earlier MapReduce systems thus further advancing the MapReduce paradigm towards scalable data stream processing.

Again, however, no simple and general approximation techniques are provided in the streaming environment with the exception of the convoluted load-shedding approaches, which are still preferred to the alternative–system crash.

2.2 Approximation Techniques

Error Estimation: Hellerstein et al. [CCA10b] presented a framework for Online Aggregation which is able to return early approximate results when querying aggregates on large stored data sets [HHW97]. This work is based on relational database systems, and is limited to simple single aggregations, which restricts it to AVG, VAR, and $STDDEV$. Later, B. Li et al. [LMD11] and Condie et al. [CCA10b] built systems on top of MapReduce to support continuous query answering. However, none of the systems could demonstrate the accuracy of the approximate results, and no model for the resulting approximation error was provided.

The error of arbitrary analytical functions can be estimated via a technique from

statistics called the bootstrap [ST95]. This technique relies on resampling methods. The function of interest is then computed on each resample resulting in the sampling distribution used for assigning a measure of accuracy to sample estimates. Sampling in the bootstrapping technique is done with replacement, and therefore an element in the resample may appear more than once. There exist other resampling techniques, such as the jackknife [Efr79], which perform resampling without replacement. The difference between the various resampling methods is in (1) the number of resamples required to obtain a reliable error estimate and in (2) the sampling method used. In this dissertation we use the bootstrapping technique because of its generality and accuracy [Syr01].

The original bootstrap has several desirable features, including automation and applicability to a wide range of inferential problems. With the help of the bootstrap, one can estimate the bias and the quantity of uncertainty via a standard error or via a confidence interval. Bootstrap quantities are typically approximated via a Monte-Carlo approach which requires applying a given estimator to the resamples of the original dataset [ST95] to produce a result distribution from which the corresponding error is estimated. Even with a Monte-Carlo approximation, however, the realization of the bootstrap quantities incurs a substantial computational expense.

The m of n bootstrap approach [BZ97] was introduced to mitigate the high cost of the bootstrap by only performing the computation on the subsample of size m rather than the size of the sample n where $m << n$. The success of the m of n bootstrap, however, heavily depends on the size of the subsample m. Furthermore because the variability of the subsample differs from its variability of the full dataset, the output from the m of n bootstrap must be rescaled, which can only be performed if the explicit knowledge of the convergence rate of the estimator in question is known.

While the bootstrap technique in [Efr79] enjoys a great advantage of estimating

accuracy of arbitrary functions, the block-wise bootstrap [SPR91] extends it to give correct results for dependent stationary observations. The problem with the block bootstrap techniques is similar to that of the m of n bootstrap, namely that the user has to manually choose the block size which is a critical component in accuracy performance of the bootstrap. Thus, authors in [BK99] provide a method for automatic block size selection. Asymptotic theory tells us that the optimal block size should be $O(n^{1/3})$, however the constant in front of $n^{1/3}$ depends on the statistic and on the dependence among the observations both of which can be unknown to the user. Therefore authors in [BK99] provide an automatic block length selection to make the blockwise bootstrap more applicable for non-specialists. In this dissertation we show that our framework is less dependent on the block size selection because we pick the block size randomly.

The Stationary Bootstrap (SB) [PR94] is a popular time-series sampling technique. Unlike other methods, SB guarantees that the resampled pseudo-time series is stationary (conditional on the original data). The stationary procedure is based on resampling blocks of random length, where the length of each block has a geometric distribution. As shown in section 3.2.2 we use SB in our framework to support the time-series data.

Sampling: Sampling techniques for Hadoop were studied previously in [GC12] where authors introduce an approach of providing Hadoop job with an incrementally larger sample size. The authors propose an Input Provider which provides the JobClient with the initial subset of the input splits. The subset of splits used for input are chosen randomly. More splits are provided as input until the JobTracker declares that enough data had been processed as indicated by the required sample size. The claim that the resulting input represents a uniformly random sample of the original input, however, is not well validated. Furthermore the assumption that each of the initial splits represents a random sample of the data, which would validate the claim that the

overall resulting sample is a uniform sample of the original data, is not well justified. Finally the authors do not provide an error estimation framework which would make it useful for the case where only a small sample of the original data is used.

Random sampling over database files is closely related to random sampling over HDFS and the authors in [OR90] provide a very nice summary of various file sampling techniques. The technique discussed in [OR90] that closely resembles our sampling approach is known as a '2-file technique combined with an ARHASH method'. In the method, a set of blocks, F_1, are put into main memory, and the rest of the blocks, F_2, reside on disk. When seeking a random sample, a 2-stage sampling process is performed where F_1 or F_2 is first picked randomly, and then depending on the choice, a random sample is drawn from memory or from disk. The expected number of disk seeks under this approach is clearly much less than if only the disk was used for random sampling. The method described, however, is not directly applicable to our environment and therefore must be extended to support a distributed filesystem (e.g., HDFS).

Authors in [CDS04] explore another efficient sampling approach, termed 'block sampling'. Block-sampling suffers, however, from a problem that it no longer is a uniform sample of the data. The approximation error derived from a block-level sampling depends on the layout of the data on disk (i.e., the way that the tuples are arranged into blocks). When the layout is random, then the block-sample will be just as good as a uniform tuple sample, however if there is a statistical dependence between the values in a block (e.g., if the data is clustered on some attribute), the resulting statistic will be inaccurate compared to a statistic derived from a uniform-random sample. In practice most data layouts fall somewhere between the clustered and the random versions [CDS04]. Authors in [CDS04] present a solution to this problem where the number of blocks to include in a sample, are varied to achieve a uniformly random distribution.

The approach of taking larger and larger samples while the accuracy of a learning

model improves had been studied in [PJO99]. Because the distribution parameters are unknown apriori, authors were able to successfully apply progressive sampling thus adapting the sample size to specific data characteristics. In [PJO99] it was shown that in fact simple, geometric progressive sampling schedule is asymptotically optimal which underlines the efficiency of progressive sampling.

Unsatisfied by the potentially redundant iterations in progressive sampling, authors in [LB04] proposed a method that avoids several samples in the geometric progression to reach the accuracy plateau faster. Their idea is to predict the learning curve from a few initial set of points, however their approach relies on having a set of learning curves from other datasets which the authors use to predict the learning curve of the current dataset of interest.

Another approach was proposed by [MTH02] that uses progressive sampling together with EM algorithm to approximate model-based clustering. The authors in [MTH02] use benefit/cost approach to decide when increasing the sample size is no longer beneficial. The work in [MTH02], however, is again task-specific and does not do delta maintenance when sample size is increased.

A new sampling approach was proposed by [ND09] that combines the two sampling strategies for the specific task via the proposed optimization function that minimizes the distance between the sampled data and the actual dataset. A lot of the above work however is only applicable to specific mining tasks.

Besides sampling, distributed processing is another technique for performing analytics on large amounts of data. Authors in [CHB04] show that by combining sampling with distributed processing one can obtain good results for learning ensembles. Their approach works by partitioning the data into T nodes and performing importance sampling on each node, while training the model, until the error converges. The author's approach, however, is again task-specific.

2.3 Sample Size Prediction

The pioneering work of Vapnik et al. [Vap99] introduces a VC (Vapnik-Chervonenkis) dimension which is a measure of the capacity of a statistical classification algorithm. The VC dimension has utility in statistical learning theory, because it can predict a probabilistic upper bound on the test error of a classification model. Previous work however considered a worst case (pessimistic) learning strategy where an adversary can pick input that can result in a very high error. Because the VC dimension theory is general it must also deal with the pessimistic case which results in a very loose upper bound for error estimation making the VC dimension approach to error estimation unhelpful for the average case.

Motivated by the previous observations that the VC bounds can sometimes fail led authors in [HSK94] to provide a new theory where the concept of *error shells* is introduced that partition the learning curve into segments, thus capturing the potential "phase transitions" that learning curves sometimes exhibit. As a result the authors provide a theory that explains many nontrivial behavioral phenomena of learning curves. The theory proposed, however, is theoretical and in this work we take a more practical approach to predicting learning curves.

Recent works [Ho08, HB02] provide ways to analyze the behavior of a classifier by analyzing the geometric and topological data complexity. These results provide a deeper look into the classifier behavior that go beyond studying the error of a classifier. The authors provide three ways in which data complexity can be derived (1) class ambiguity, (2) boundary complexity and (3) sample sparsity / feature space dimensionality. The uses of their work include determining the existence of any learnable structure and a set of expectations on potential gains by automatic learning algorithms. The complexity measure can be used to compare different problem formulations, including alternative class definitions, noise conditions and sampling strategies. The

classifier complexity measure can guide the process of classifier selection, or control the process of classifier training. Motivated by this work, we relate the complexity of the inter-variable relationship in the data to the sample size.

2.4 Applications Background

As application examples, throughout the chapters of this dissertation, we use classification, pattern matching, and association discovery. All three of these applications share the common BigData problems of prohibitively large data size and time-constrained response requirement. Next we describe the background necessary to be familiar with these domains.

2.4.1 Classification

In the course of improving approximation we have also made improvements to the performance of classifiers. Specifically, motivated by poor classification speed of Support Vector Machines (SVM) as compared to neural networks authors in [BS96] improve the accuracy of SVM by incorporating the knowledge of invariances of the problem available. Furthermore, the authors increase the classification speed by reducing the decision function complexity. Overall, however, there is no practical approximation techniques for SVM which are critical for processing of 'Big Data'.

There have been numerous methods that focus on selecting a subset of the data [WNC05, LH07], specifically targeting SVM, however the proposed methods are applicable to only special cases. In [WNC05] authors propose a method that selects a subset of the data that are most likely to be support vectors. The authors' decision is based on the observation that SVM's classification decision depends on a small subset of the training data, called support vectors. In [LH07] authors choose a subset of the

training examples using uniform sampling which they show to be an optimal and a robust selection scheme in terms of several statistical measures. The above approaches however are special approaches to solving the problem with large data and the user still has no ability to control the training size based on the model accuracy which is at the core of this dissertation.

2.4.2 Association Discovery

Another important application area is that of time series, and we have used a time-series dataset to discover interesting associations between variables. The Pearson Correlation coefficient r is often used to measure the strength of association between a pair of variables. Given two signals x and y of equal length m, with respective averages μ_x and μ_y and standard deviations σ_x and σ_y their Pearson correlation coefficient is defined as:

$$corr(x, y) = \frac{1}{m} \sum_{y=0}^{m-1} \left(\frac{x_i - \mu_x}{\sigma_x} \right) \left(\frac{y_i - \mu_y}{\sigma_y} \right)$$

For example, the Pearson correlation between the height of a child and their parents is $r \approx 0.5$ [Spe] and that of the wheat yield and annual rainfall is $r \approx 0.75$. Pearson's r, however, only captures linear correlations, and it is not applicable to signals which have nonlinear associations.

In section 6.1 we present a sophisticated association discovery technique that works for nonlinear associations and show how one can apply the approximation framework presented to discover inter-variable relationships in a reasonable time.

2.4.3 Pattern Matching

Another important application area studied in the dissertation is pattern matching. There is a wide variety of work on pattern matching all of which assume a static

environment and thus static optimizations. To keep the static size of the automaton small, several limitations had to be imposed. For example, some of the limitations include disallowing the use of the Kleene Closure in underlying REs [WDR06], performing pattern matching without outputting complete matches [DGH06] and employing a spatial indexing structure for indexing which is not suitable for data streams [CGR03]. Instead in this dissertation we show how the data complexity theory can be used to relax the static environment assumption to improve the pattern matching performance.

Authors in [YCD06] proposed a technique for rewriting regular expressions to avoid exponential memory blow-up of a DFA due to Kleene Closure, however according to [MRV08] their approach only works for a subset of Snort REs [Roe99].

Another way to adhere to the system resource constraints is to construct a DFA lazily (on the fly) [GGM04] assuming that a completely expanded lazy DFA will be small and be able to fit in memory, however this is a strong assumption and will not always hold.

There are many other efficient techniques to do pattern matching [CGR03, LJL08, DF03], however none of the techniques address a resource constrained environment where system-stress level changes and dynamic adjustment of the underlying pattern representation is necessary.

The pattern matching portion of this dissertation uses similar motivation as Adaptive Query Processing (AQP) [IDR07] namely the fact that the static query does not provide optimal performance in an environment where the system resources and data characteristics constantly change. Instead of dynamically modifying the query plan as is done with AQP our approach modifies the underlying system, which in our case is a pattern automaton, for optimal pattern matching performance. The key idea of our pattern matching work is dynamic automaton reconfiguration, which is different from AQP, and has not yet been addressed in literature. Furthermore, our dynamic pattern

matching optimization allows for a more effective use of a multicore system.

CHAPTER 3

Quality Assessment

3.1 Quality Assessment Using the Standard Bootstrap: the EARL System

Approximate results based on samples often provide the only way in which advanced analytical applications on very massive data sets can satisfy their time and resource constraints. Unfortunately, methods and tools for the computation of accurate early results are currently not supported in MapReduce-oriented systems although these are intended for 'big data'. Therefore, in this chapter we devise and implement a non-parametric extension of Hadoop which allows the incremental computation of early results for arbitrary work-flows, along with reliable on-line estimates of the degree of accuracy achieved so far in the computation. These estimates are based on a technique called bootstrapping that has been widely employed in statistics and can be applied to arbitrary functions and data distributions. In this chapter, we describe the Early Accurate Result Library (EARL) for Hadoop that was designed to minimize the changes required to the MapReduce framework. Various tests of EARL of Hadoop are presented to characterize the required situations where EARL can provide major speed-ups over the current version of Hadoop.

3.1.1 Introduction

In today's fast-paced business environment, obtaining results quickly represents a key desideratum for 'Big Data Analytics' [HLL11]. In analyzing large data sets, performing careful sampling on the data and computing early results from such samples provide a fast and effective way to obtain approximate results within the prescribed level of accuracy for most applications. Although the need for approximation techniques obviously grow with the size of the data sets, general methods and techniques for handling complex tasks are still lacking in both MapReduce systems and parallel databases although these claim 'big data' as their forte. Therefore in this chapter, we focus on providing this much needed functionality. To achieve our goal, we explore and apply powerful methods and models developed in statistics to estimate results and the accuracy obtained from sampled data [ST95, Efr79]. We propose a method and a system that optimize the work-flow computation on massive data-sets to achieve the desired accuracy while minimizing the time and the resources required. Our approach is effective for analytical applications of arbitrary complexity and is supported by an Early Accurate Result Library (*EARL*) that we developed for Hadoop, which will be released for experimentation and non-commercial usage [rel]. The early approximation techniques presented are also important for fault-tolerance, where only a portion of the data is available and the error estimation is required to determine if node recovery is necessary.

The importance of EARL follows from the fact that real-life applications often generate a tremendous amount of data. Performing analytics and delivering exact query results on such large volumes of stored data can be a lengthy process, which can be entirely unsatisfactory to a user. In general, overloaded systems and high delays are incompatible with a good user experience; moreover approximate answers that are accurate enough and can be generated quicker are often of much greater value to users

than tardy exact results. The first line of research work pursuing similar objectives is that of Hellerstein et al. [HHW97], where early results for simple aggregates are returned. In EARL however, we seek an approach that is applicable to complex analytics and dovetails with a MapReduce framework.

When computing some analytical function in EARL, a uniform sample, s, of stored data is taken, and the resulting error is estimated using the sample. If the error is too high, then another iteration is invoked where the sample size is expanded and the error is recomputed. This process is repeated until the computed error is below the user-defined threshold. The error for arbitrary analytical functions can be estimated via the bootstrapping technique described in [ST95]. This technique relies on resampling methods, where a number of samples are drawn from s. The function of interest is then computed on each sample resulting in the sampling distribution used for assigning measure of accuracy to sample estimates. Sampling in the bootstrapping technique is done *with replacement*, and therefore an element in the resample may appear more than once. There exist other resampling techniques, such as the jackknife [Efr79], which perform resampling without replacement. The difference between the various resampling methods is in (1) the number of resamples required to obtain a reliable error estimate and in (2) the sampling method used. In this chapter we use the bootstrapping technique because of its generality and accuracy [Syr01]. While incorporating other resampling methods provides an exciting research direction for future work, it is beyond the limited scope of this chapter.

Hadoop is a natural candidate for implementing EARL. In fact, while our error approximation approach is general, it benefits from the fundamental Hadoop infrastructure. Hadoop employs a data *re-balancer* which spreads HDFS [had] data uniformly across the DataNodes in the cluster. Furthermore, in a MapReduce framework there are a set of (key, value) pairs which map to a particular reducer. This set of pairs can

be distributed uniformly using random hashing and by choosing a subset of the keys at random, a uniform sample can be generated quickly. These two features make Hadoop a desirable foundation for *EARL*, while Hadoop's popularity maximizes the potential for practical applications of this new technology.

Because *EARL* can deliver approximate results, it is also able to provide fault-tolerance in situations where there are node failures. Fault-tolerance is addressed in Hadoop via data-replication and task-restarts upon node failures, however with *EARL* it is possible to provide a result and an approximation guarantee despite node failures without task restarts.

Our approach, therefore, addresses the most pressing problem with Hadoop and with MapReduce framework in general: a high latency when processing large data-sets. Moreover, the problem of reserving too many resources to ensure fault-tolerance can also be mitigated by our approach and is discussed in Section 3.1.7.

Contributions and Organization This chapter makes the following three contributions:

1. A general early-approximation method is introduced to compute accurate approximate results with reliable error-bound estimation for arbitrary functions. The method can be used for processing large data-sets on many systems including Hadoop, Teradata, and others. An Early Accurate Result Library (EARL) was implemented for Hadood and used for the experimental validation of the method.

2. An improved resampling technique was introduced for error estimation; the new technique uses delta maintenance to achieve much better performance.

3. A new sampling strategy is introduced that assures a more efficient drawing of random samples from a distributed file system.

Organization In section 3.1.2 we describe the architecture of our library as it is implemented on Hadoop. Section 3.1.3 describes the statistical techniques used for early result approximation. Section 3.1.9 presents the resampling optimizations. In Section 3.1.10 performance evaluation of EARL is provided.

3.1.2 EARL System Architecture

This section describes the overall EARL architecture and gives a background on the underlying system. For a list of all symbols used refer to Table 3.1. EARL consists of (1) a sampling stage, (2) a user's task, and (3) an accuracy estimation stage which are presented in Figure 3.1. The sampling stage draws a uniform sample s of size n from the original data set S of size N where $n << N$. In Section 3.1.6 we discuss how this sampling is implemented using tuple-based and key-based sampling for MapReduce. After the initial sample s is drawn from the original data-set, B samples (i.e. resamples) with replacement are taken from s. These resamples are used in the work phase (user's task) to generate B results, which are then used to derive a result distribution [DM01] in the accuracy estimation phase. The sample result distribution is used for estimating the accuracy. If the accuracy of a result is unsatisfactory, the above process is repeated by drawing another sample Δs which is aggregated with the previous sample s to make a larger sample s' for higher accuracy. The final result is returned when a desired accuracy is reached.

Extending MapReduce

For implementing the underlying execution engine, we evaluated three alternatives, (1) Hadoop, (2) HaLoop [BHB10b] and (3) Hadoop online [CCA09]. Although HaLoop would allow us to easily expand the sample size on each iteration, it would be slow for non-iterative MR jobs due to the extra overhead introduced by HaLoop. With Hadoop online, we would get the benefit of pipelining, however further modifications

Symbol	Description
B	Number of bootstraps
b	A particular bootstrap sample
n	Sample size
s	Array containing the sample
p	Percentage of the data contained in a sample
N	Total data size
S	Original data-set
F_i	File split i
c_v	Coefficient of variation
f	Statistic of interest
σ	User desired error bound
τ	Error accuracy
X_i	A particular data-item i
θ	Parameter of interest
D	Total amount of data processed

Table 3.1: EARL System: Symbols used

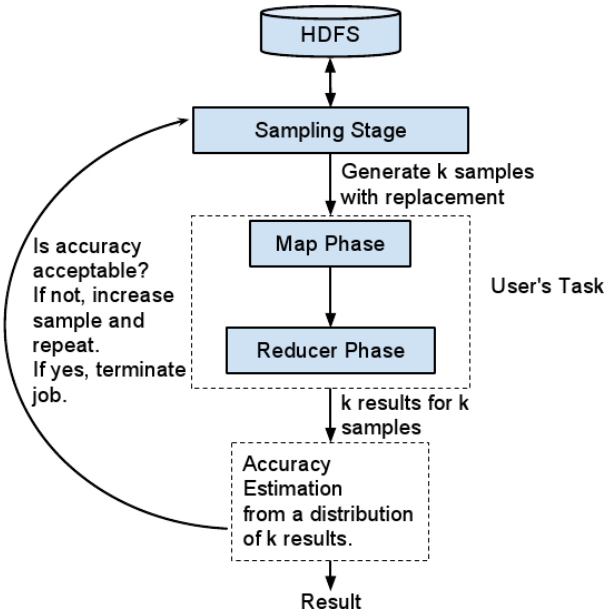


Figure 3.1: A simplified EARL architecture

would be needed to allow the mapper to adjust the current sample size. Since both Hadoop Online and HaLoop do not exactly fit our requirements, we therefore decided to make a relatively simple change to Hadoop that would allow dynamic input size expansion required by out approach. Thus *EARL* adds a simple extension to Hadoop to support dynamic input and efficient resampling. An interesting future direction is to combine *EARL*'s extensions with those of HaLoop and HOP to make a comprehensive data-mining platform for analyzing massive data-sets. In summary, with the goals of seeking *EARL* fast and requiring the least amount of changes to the core Hadoop implementation we decided to use the default version of Hadoop instead of using Hadoop extensions such as Hadoop online or HaLoop .

To achieve dynamic input expansion we modify Hadoop in three ways (1) to allow the reducers to process input before mappers finish (2) to keep mappers active until explicitly terminated and (3) to provide a communication layer between the mappers and reducers for checking the termination condition. While the first goal is similar to that of pipelining implemented in Hadoop Online Prototype (HOP) [CCA09], *EARL*

is different from HOP in that in *EARL* the mapper is actively, rather than a passively, transfers the input to the reducer. In other words, the mapper actively monitors the sample error and actively expands the current sample size. The second goal is to minimize the overall execution time, thus instead of restarting a mapper every time sample size expands, we reuse an already active mapper. Finally, each mapper monitors the current approximation error and is terminated when the required accuracy is reached.

We also modify the reduce phase in Hadoop to support efficient incremental computation of the user's job. We extend the MapReduce framework with a finer-grained reduce function, to implement incremental processing via four methods: (i) *initialize()*, (ii) *update()*, (iii) *finalize()* and (iv) *correct()*. The $initialize()$ function reduces a set of data values into a *state*, i.e. $< k, v_1 >, < k, v_2 >, ..., < k, v_k > \rightarrow < k, state >$. A state is a representation of a user's function f after processing s on f. Each resample will produce a state. Saving states instead of the original data requires much less memory as needed for fast in-memory processing. The $update()$ function updates the state with a new input which can be another state or a <key, value> pair. The $finalize()$ function computes the current error and outputs the final result. The $correct()$ function takes the output of the $finalize()$ function, and corrects the final result. When computed from a subset of the original data, some user's tasks need to be corrected in order to get the right answer. For example, consider a SUM query which sums all the input values. If we only use p of the input data, we need to scale the result by $1/p$. As the system is unaware of the internal semantics of user's MR task, we allow our users to specify their own correction logic in $correct()$ with a system provided parameter p which is the percentage of the data used in computation.

Hadoop's limited two stage model makes it difficult to design advanced data-mining applications for which reason high level languages such as PIG [ORS08a] were introduced. *EARL* does not change the logic of the user's MapReduce programs and

achieves the early result approximation functionality with minimal modifications to the user's MR job (see Figure 3.5).

Next the accuracy estimation stage is described.

3.1.3 EARL Details

In EARL, error estimation of an arbitrary function can be done via resampling. By re-computing a function of interest many times, a result distribution is derived from which both the approximate answer and the corresponding error are retrieved. EARL uses a clever delta maintenance strategy that dramatically decreases the overhead of computation. As a measurement of error, in our experiments, we use a coefficient of variation (c_v) which is a ratio between the standard deviation and the mean. Our approach is independent of the error measure and is applicable to other errors (e.g., bias, variance). Next a traditional approach to error estimation is presented, after which our technique is discussed.

A crucial step in statistical analysis is to use the given data to estimate the accuracy measure, such as the bias, of a given statistic. In a traditional approach, the accuracy measure is computed via an empirical analog of the explicit theoretical formula derived from a postulated model [ST95]. Using variance as an illustration let $X_1, ..., X_n$ denote the data set of n independent and identically distributed (i.i.d) data-items and let $f_n(X_1, ..., X_n)$ be the function of interest we want to compute. The variance of f_n is then:

$$var(f_n) = \int \left[f_n(x) - \int f_n(y) d \prod_{i=1}^{n} F(y_i) \right]^2 d \prod_{i=1}^{n} F(x_i) \tag{3.1}$$

where $x = (x_1, ..., x_n)$ and $y = (y_1, ..., y_n)$. Given a simple f_n we can obtain an equation of $var(f_n)$ as a function of some unknown quantities and then substitute the estimates of the unknown quantities to estimate the $var(f_n)$. In the case of the

sample mean, where $f_n = \bar{X}_n = n^{-1}\sum_{i=1}^{n} X_i$, $var(\bar{X}_n) = n^{-1}var(X_1)$. We can therefore estimate $var(\bar{X}_n)$ by estimating $var(X_1)$ which is usually estimated by the sample variance $(n-1)^{-1}\sum_{i=1}^{n}\left(X_i - \bar{X}_n\right)^2$. The use of Equation 3.1 to estimate the variance is computationally feasible only for simple functions, such as the mean. Next we discuss a resampling method used to estimate the variance of arbitrary functions.

Resampling approaches, such as *bootstrap* and *jackknife* [Tho00], provide an accuracy estimation for general functions. Both of these resampling methods do not require a theoretical formula to produce the error estimate of a function. In fact these techniques allow for estimation of the sampling distribution of almost any statistic using only very simple methods [Efr87]. The estimate of the variance can then be determined from the sampling distribution. Both techniques however require repeated computation of the function of interest on different resamples. The estimate of the variance of the result, σ, produced by this repeated computation is $\sigma^2(F) = E_F(\hat{\theta} - E_F(\hat{\theta}))^2$, where θ is the parameter of interest. The jackknife has a fixed requirement for the number of resamples, n, that is often relatively low. The number of samples required by the *bootstrap* approach, however, is not fixed and can be much lower than that of the *jackknife*. Moreover, it is known that jackknife does not work for many functions such as the median [Efr79]. Thus, in this first version of EARL we concentrate on bootstrapping and leave the study of other resampling methods for future work.

To compute an *exact* bootstrap variance estimate $\binom{2n-1}{n-1}$ resamples are required, which for $n = 15$ is already equal to 77×10^6, therefore an approximation is necessary to make the bootstrap technique feasible. The *Monte-Carlo* [ST95] is the standard approximation technique used for resampling methods including the bootstrap that requires less than n resamples. It works by taking B resamples resulting in variance estimate of $\hat{\sigma}_B^2 = \frac{1}{B}\sum_{n=1}^{B}(\hat{\theta}_n^* - \hat{\theta}^*)^2$ where $\hat{\theta}^*$ is the average of $\hat{\theta}_n^*$'s. The theory suggests that B should be set to $\frac{1}{2}\epsilon_0^{-2}$ [Efr87], where ϵ_0 corresponds to the desired error

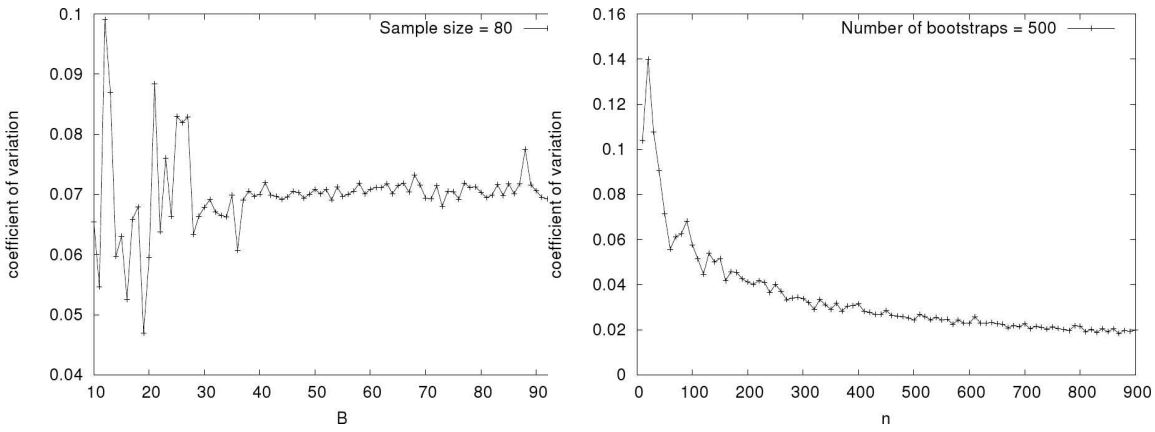


Figure 3.2: Bootstrap in EARL: (left) Effect of B on c_v, (right) Effect of n on c_v

of the Monte Carlo approximation with respect to the the original bootstrap estimator. Experiments, however, show that a much better value of B can be used in practical applications, therefore in section 3.1.5 we develop an algorithm to empirically determine a good value of B.

3.1.4 Accuracy Estimation Stage

The accuracy estimation stage (*AES*) uses the bootstrap resampling technique [DM01] outlined in the previous subsection to estimate the standard error c_v of the statistic f computed from sample s.

In many applications, the number of bootstrap samples required to estimate c_v to within a desired accuracy τ can be substantial. τ is defined as $\tau = (c_{v_i} - c_{v_{i+1}})$ which measures the stability of the error. Before performing the approximation, we estimate the required B and n to compute f with $c_v \leq \sigma$. If $B \times n \geq N$, then EARL informs the user that an early estimation with the specified accuracy is not faster than computing f over N and instead the computation over the entire data-set is performed. AES allows for error estimation of general MR-Jobs (mining algorithms, complex functions etc). Furthermore, EARL works on categorical, as well as on inter-dependent data as discussed in Appendix 3.1.9.3.

For completeness, we will first discuss how B and n impact the error individually, and then in Section 3.1.5 we present an algorithm to pick B and n that empirically minimizes the product $B \times n$. Figure 3.2 (left) shows how B affects c_v experimentally. Normally roughly 30 bootstraps are required to provide a confident estimate of the error. The sample size, n, given a fixed B has a similar effect on c_v as shown in Figure 3.2 (right). A larger n results in a lower error. Depending on the desired accuracy, n can be chosen appropriately as described next.

3.1.5 Determining the Sample Size and Number of Bootstraps

To perform resampling efficiently (i.e. without processing more data than is required) we need to minimize the sample size (n) and the number of resamples performed (B). A straightforward sample size adjustment might work as follows: pick an initial sample size s of size n which theoretically achieves the desired error σ and compute f on s. If the resulting error $\hat{\sigma}$ is greater than σ then the sample size is increased (e.g., doubled). A similar naïve strategy may be applicable when estimating the minimum B. This naïve solution however may result in an overestimate of the sample size and the number of resamples. Instead, following [CDS04] we propose a two phase algorithm to estimate the final early approximate result satisfying the desired error bound while empirically minimizing $B \times n$. As shown later, our algorithm requires only a single iteration.

Sample Size And Bootstrap Estimation (SSABE) algorithm we propose, performs the following operations: (1) In the first phase, it estimates the minimum B and n and then (2) in the second phase, it evaluates the function of interest, f, B times on s of size n. To estimate the required B, the first phase an initially small n, a fraction p of N, is picked. In practice we found that $p = 0.01$ gives robust results. Given a fixed n, a sample s is picked. The function f is then computed for different candidate values

of B ($\{2, ..., \frac{1}{\tau}\}$). The execution terminates when the difference $|c_{v_i} - c_{v_{i-1}}| < \tau$. The B value so determined is used as the estimated number of bootstraps. In practice the value of B so calculated is much smaller than the theoretically predicted $\frac{1}{2}\epsilon_0^{-2}$.

To estimate the required sample size n, first the initial sample size $\frac{1}{\tau}$ is picked. The initial sample is split into l smaller subsamples s_i each of size n_i where $n_i = \frac{n}{2^{l-i}}$ and $1 \leq i \leq l$. In our experiments we found it to be sufficient to set $l = 5$. For each s_i we compute the c_v using B resamples. When computing f on s_i we perform delta maintenance discussed in Section 3.1.9. The result will be a set of points $A[s_i] = c_v$. For these set of points, the best fitting curve is constructed. The curve fitting is done using the standard method of least squares. The best fitted curve yields an s_i that satisfies the given σ. Finally, once the estimate for B and n is complete, the second phase is invoked where the actual user job is executed using s of size n and B.

The initial n is picked to be small, therefore the sample size and the number of bootstraps estimation can be performed on a single machine prior to MR job start-up. Thus, when performing the estimation for n and B we run the user's MR job in a local mode without launching a separate JVM. Using the local-mode we avoid running the mapper and the reducer as separate JVM tasks and instead a single JVM is used which allows for a fast estimation of the required parameters needed to start the job.

3.1.6 I.I.D. Sampling in a Distributed Environment

In order to provide a uniformly random subset of the original data-set, *EARL* requires sampling. While sampling over memory-resident, and even disk resident, data had been studied extensively, sampling over a distributed file system, such as HDFS, has not been addressed [OR90]. Therefore, we provide two sampling techniques: (1) pre-map sampling and (2) post-map sampling. Each of the techniques has its own strengths and weaknesses as discussed next.

In HDFS, a file is divided into a set of blocks, each block is typically 64MB. When running an MR job, these blocks can be further subdivided into "Input Splits" which are used as input to the mappers. Given such an architecture, a naïve sampling solution is to pick a set of blocks B_i at random, possibly splitting B_i into smaller splits, to satisfy the required sample size. This strategy however will not produce a uniformly random sample because each of the B_i and each of the splits can contain dependencies (e.g., consider the case where data is clustered on a particular attribute resulting in clustered items to be placed next to each other on disk due to spatial locality). Another naïve solution is to use a reservoir smapling algorithm to select n random items from the original data-set. This approach produces a uniformly random sample, but it suffers from slow loading times because the entire dataset needs to be read, and possibly re-read when further samples are required. We thus seek a sampling algorithm that avoids such problems.

In a MapReduce environment, sampling can be done before or while sending the input to the Mapper (*pre-map* and *post-map* sampling respectively). Pre-map sampling significantly reduces the load times, however the sample produced may be an inaccurate representation of the total $< k, v >$ pairs present in the input. Post-map sampling first reads the data and then outputs a uniformly random sample of desired size. Post-map sampling also avoids the problem of inaccurate $< k, v >$ counts.

Post-map sampling works by reading and parsing the data before sending the selected $< k, v >$ pairs to the reducer. Each $< k, v >$ pair is stored by using random hashing that generates a pre-determined set of keys, of size proportional to the required sample size. We store all *key*, *value* pairs on the mapper locally, and when all data had been received, we randomly pick p *key*, *value* pairs that satisfies the sample size and send it to the reducer. Because sampling is done without replacement, the key, value pairs already sent are removed from the hashmap. Post-map sampling is shown in

Algorithm 1.

Algorithm 1: Post-map sampling

$hash \leftarrow$ initialize the hash;

$timestamp \leftarrow$ initialize the timestamp;

while *input* ! = *null* **do**

$key \leftarrow$ get random key for input;

$value \leftarrow$ get value for input;

$hash[key] \leftarrow value$;

end

sendSample(hash(rand()%hash_size);

while *true* **do**

if *get_new_error_average (timestamp)* $>$ *required* **then**

sendSample(hash(rand()%hash_size));

end

else

return

end

end

Unlike *post-map* sampling, which first reads the entire dataset and then randomly chooses the required subset to process, *pre-map* sampling works by sampling a portion p of the initial dataset before it gets passed into the mapper. Therefore, because sampling is done prior to data loading stage, the response time is greatly improved, with a potential downside of a slightly more inaccurate result. The reason for this is because when sampling from HDFS directly, we can efficiently only do so by sampling lines[1]. Each line however may contain a variable number of $< k, v >$ pairs so that when producing a 1% sample of the *key*,*value* pairs, we may produce a larger or

[1] A default file format in Hadoop is a line delimitted by a new-line character. Another format can be specified via the *RecordReader* class in Hadoop.

a lesser sample. Therefore, for f which needs correction, we would be unable to do so accurately without additional information from the user. For majority of the cases however correcting the final result is not necessary, and even for cases when correction is required, the estimate of the number of the *key, value* pairs produced by the *pre-map* sampling approach is good enough in practice. Nevertheless the user has the flexibility to use *post-map* sampling if an accurate correction to the final result is desired.

We assume, w.l.o.g., that the input is delimited by *new-lines*, as opposed to commas or other delimiters. A set of logical splits is first generated from the original file which will be used for sampling. For each split F_i, we maintain a bit-vector representing the *start* byte locations of the lines we had already included in our sample. Therefore until the required sample size is met, we continue picking a random F_i and a random *start* location which will be used to include a line from a file. To avoid the problem of picking a file *start* location which is not a beginning of a line, we use Hadoop's *LineRecordReader* to backtrack to the beginning of a line. Using *pre-map* sampling we avoid sending an overly large amount of data to the mapper which improves response time as seen in experiment in Section 3.1.10.1. In rare cases where a larger sample size is required for an in-progress task, a new split is generated and the corresponding map task is restarted in the *TaskInProgress* Hadoop class. Algorithm 2 presents the HDFS sampling algorithm used in pre-map sampling.

Algorithm 2: HDFS sampling algorithm used in pre-map sampling

```
start ← split.getStart();
end ← start + split.getLength();
sample ← ∅;
while |sample| < n do
    start ← pick a random start position;
    if start ! = beginning of a line then
        skipFirstLine ← true;
        fileIn.seek(start);
    end
    in = new LineReader(fileIn, job);
    if skipFirstLine then
        start += in.readLine(new Text(), ;
        0, (int)Math.min((long)Integer.MAX_VALUE,
        end - start));
    end
    sample ← includeLineInSample();
    skipFirstLine ← false
end
```

Therefore, while pre-map is fast and works well for most cases, post-map is still very useful for applications where a correction function relies on an accurate estimate of the total *key, value* pairs. Experiments highlighting the difference between the two sampling methods are presented in experiment of Section 3.1.10.5.

In both the post-map and the pre-map sampling, every reducer writes its computed error together with a time-stamp onto HDFS. These files are then read by the mappers to compute the overall average error. Because both the mappers and the reducers share the same JobID, it is straight forward to list all files generated by the reducers within

the current job. The mapper stores a time-stamp that corresponds to the last successful read attempt of the reducer output. The mapper collects all errors, and computes the average error. The average error, incurred by all the reducers, is used to decide if sample size expansion is required. Lines 9-15 in Algorithm 1 demonstrate this for pos-map sampling.

Note that in a MapReduce framework independence is assumed between *key, value* pairs. In addition to being natural in a MapReduce environment, the independence assumption also makes sampling applicable to algorithms relying on capturing data-structure such as correlation analysis.

3.1.7 Fault-Tolerance

Most clusters that use Hadoop and the MapReduce frameworks utilize commodity hardware and therefore node failure are a part of every-day cluster maintenance. Node failure is handled in the Hadoop framework with the help of data-replication and task-restarts upon failures. Such practices however can be avoided if the user is only interested in an approximate result. Authors in [SG07] show that in the real world, over 3% of hard-disks fail per year, which means that in a server farm with 1,000,000 storage devices, over 83 will fail every day. Currently, the failed nodes have to be manually replaced, and the failed tasks have to be restarted. Given a user specified approximation bound however, even when most of the nodes have been lost, a reasonable result can still be provided. Using the ideas from *AES* stage the error bound of the result can still be computed with a reasonable confidence. Using our simple framework, a system can therefore be made more robust against node failures by delivering results with an estimated accuracy despite node failures.

3.1.8 Early Approximation for K-Means

Results for the initial experiments in the realm of classical data mining applications are encouraging. The experiment below provides a performance study that uses *EARL* to approximate *K-Means*.

It is well known that *K-Means* algorithm converges to a local optima and is also sensitive to the initial centroids. For these reasons the algorithm is typically restarted from many initial positions. There are various techniques used to speed up K-Means, including parallelization [ZMH09b]. Our proposed approach, compliments previous techniques by speeding up K-Means without changing the underlying algorithm.

Figure 3.8 shows the results of running *K-Means* with EARL and stock Hadoop. Our approach leads to a speed up due to two reasons: (1) K-Means is executed over a small sample of the original data and (2) K-Means converges more quickly for smaller data-sets.

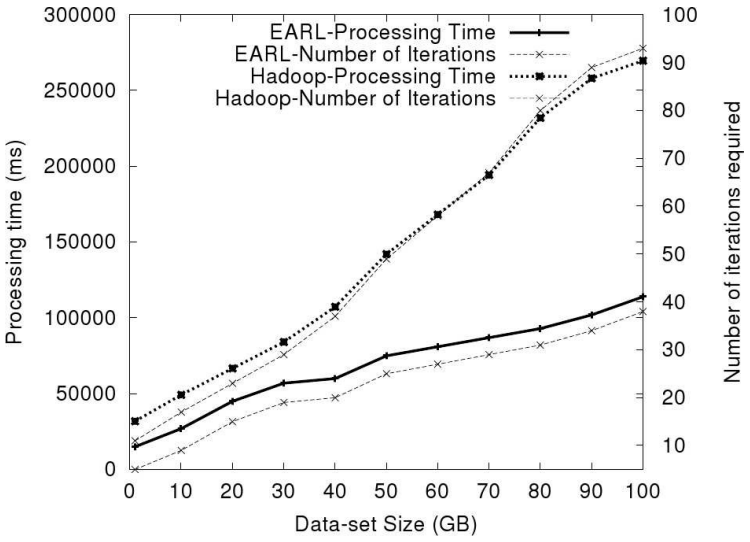


Figure 3.3: Computation of K-Means using EARL and stock Hadoop

3.1.9 EARL Optimizations

The most computationally intensive part of *EARL*, aside from the user's job j, is the re-execution of j on an increasingly larger sample sizes, during both the main job execution and during initial sample size estimation. One important observation is that this intensive computation can reuse its results from the previous iterations. By utilizing this incremental processing, performing large-scale computations can be dramatically improved. We first take a more detailed look at the processing of two consecutive bootstrap iterations and then we discuss the optimization of the bootstrapping (resampling) procedure so that when recomputing f on a new resample s' we can perform delta maintenance using a previous resample s.

3.1.9.1 Inter-Iteration Optimization

Let s denote the sample of size n used in the i-th iteration, and let $\{b_i, 1 \leq i \leq B\}$ denote the B bootstrap resamples drawn from s. The user's job j is repeated on all b_i's. In the $(i+1)$-th iteration, we enlarge sample s with another sample Δs. s and Δs are combined to get a new sample s' of size n'. B bootstrapping resamples $\{b'_i, 1 \leq i \leq B\}$ are drawn from s', and the user's job j is repeated on all b'_i's. Each resample b'_i can be decomposed into two parts: (1) the set of data-items randomly sampled from s, denoted by $b'_{i,s}$, and (2) the set of data-items randomly sampled from Δs, denoted by $b'_{i,\Delta s}$.

Therefore, in the $(i+1)$-th iteration, instead of drawing a completely new $\{b'_i\}$ from s', we can reuse the resamples $\{b_i\}$ generated in the i-th iteration. The idea is to generate $b'_{i,s}$ by updating b_i, and to generate $b'_{i,\Delta s}$ by randomly sampling from Δs. This incremental technique has two benefits, in that we can save a part of: (1) the cost of bootstrapping resampling $\{b'_i\}$, and (2) the computation cost of repeating the user's

job j on $\{b'_i\}$.

The process of generating $b'_{i,s}$ from b_i is not trivial, due to the following observation. Each data item in b'_i is drawn from b_i with probability $\frac{n}{n'}$, and from Δs with probability $1 - \frac{n}{n'}$. We have the following equation modeling the size of $b'_{i,s}$ by a binomial distribution.

$$P(|b'_{i,s}| = k) = \binom{n'}{k} \left(\frac{n}{n'}\right)^k \left(1 - \frac{n}{n'}\right)^{n'-k} \tag{3.2}$$

This means that we may need to randomly delete data-items from b_i, or add data-items randomly drawn from s to b_i. We first present a naive algorithm which maintains a resample b'_i from s' by updating the resample b_i form s in three steps: (1) randomly generate $|b'_{i,s}|$ according to Equation 4.4. (2) if $|b'_{i,s}| < n$, then randomly delete $(n - |b'_{i,s}|)$ data-items from b_i; if $|b'_{i,s}| > n$, then randomly sample $(|b'_{i,s}| - n)$ data-items from s and combine them with b_i. (3) generate $(n' - |b'_{i,s}|)$ random sample from Δs and combine them with b_i.

The above process requires us to record all the data-items of s and b_i, which is a huge amount of data that cannot reside in memory. Therefore, s and b_i must be stored on the HDFS file system. Because this data will be accessed frequently, the disk I/O cost can be a major performance bottleneck.

Next, we present our optimization algorithm with a cache mechanism that supports fast incremental maintenance. Our approach is based on an interesting observation from Equation 4.4. With n' very large and n/n' fixed, which is usually the case in massive MapReduce tasks, Equation 4.4 can be approximated by the Gaussian distribution

$$N\left(n, n\left(1 - \frac{n}{n'}\right)\right) \tag{3.3}$$

For a Gaussian distribution, by the famous *3-sigma rule*, most data concentrate around the mean value, to be specific, within 3 standard deviations of the mean.

As an example, for the distribution 4.5 with its standard deviation denoted by $\sigma_0 = \sqrt{n\left(1-\frac{n}{n'}\right)}$, over 99.7% data lie within the range $(n-3\sigma_0, n+3\sigma_0)$; over 99.9999% data lie within the range $(n-5\sigma_0, n+5\sigma_0)$. Note that $\sigma_0 < \sqrt{n}$.

Next we explain our optimized algorithm in more detail. For the i-th iteration, we define the delta sample added to the previous sample as Δs_i. For the first iteration, we can treat the initial sample as a delta sample added to an empty set. Therefore we can denote it by Δs_1. The size of Δs_i is n_i. After the i-th iteration, a bootstrapping resample b can be partitioned into $\{b_{\Delta s_k}, k < i\}$, where $b_{\Delta s_k}$ represents the data-items in b drawn from Δs_k. We build a two-layer memory-disk structure of b. Instead of simply storing b on a hard-disk, we build two pieces of information of it: (i) memory-layer information (a sketch structure) and (ii) disk-layer information (the whole data set). A *sketch* of data set of size n is $c\sqrt{n}$ data items randomly drawn without replacement from it where c is a chosen constant. Determining an appropriate c is a trade-off between memory space and the computation time. A larger c will cost more memory space but will introduce less randomized update latency. The sketch structure contains $\{sketch(b_{\Delta s_k})\}$ and $\{sketch(\Delta s_k)\}$.

During updating, instead of accessing s and b directly, we always access the sketches first. Specifically, for step 2 in our algorithm, if we need to randomly delete data-items from $b_{\Delta s_k}$, we sequentially pick the data-items from $sketch(b_{\Delta s_k})$ for deletion; if we need to add data-items randomly drawn from Δs_k, we sequentially pick the data-items from $sketch(\Delta s_k)$ for addition. For already picked data-items, we mark them as *used*. At the end of each iteration, we will randomly substitute some of the unused data items in $sketch(b_{\Delta s_k})$ with the used data items in $sketch(\Delta s_k)$ by following a reservoir sampling approach, in order to maintain $sketch(b_{\Delta s_k})$ as a random sketch of $b_{\Delta s_k}$. If we use up all the data-items in a sketch, we access the copy stored in HDFS, applying two operations: (1) committing the changes on the sketch, and (2) resampling a new sketch

from the data.

3.1.9.2 Intra-Iteration Optimization

When performing a resample, at times a large portion of the new sample is identical to the previous sample in which case effective delta maintenance can be performed. Our main observation is shown in Equation 3.4. The equation represents the probability that a fraction y of a resample is identical to that of another resample. Therefore, for example if $n = 29$ and $y = 0.3$, that means that 35% of the time, resamples will contain 30% of identical data. In other words, for roughly 1 in 3 resamples, 30% of each resample will be identical to one-another. Because of the relatively high level of similarity among samples, an intra iteration delta maintenance can be performed where the part that is shared between the resamples is reused.

$$P(X = y) = \frac{n!}{(n - y * n)! \times n^{y*n}} \tag{3.4}$$

Using Equation 3.4 we can find the optimal y, given n, that minimizes the overall work performed by the bootstrapping stage. The overall work saved can be stated as $P(X = y) * y$. Figure 3.4 shows how the overall work saved varies with n for different y. The optimal y for given n can be found using a simple binary search. Overall we found that on average we save over 20% of work using our Intra Iteration Optimization procedure when compared to the standard bootstrapping method.

While the optimization techniques presented in this section greatly increase the performance of the standard bootstrap procedure there is still more research to be done with regards to delta maintenance and sampling techniques. Our optimization techniques are best suited for small sample sizes, which is reasonable for a distributed system where both response time and data-movement must be minimized.

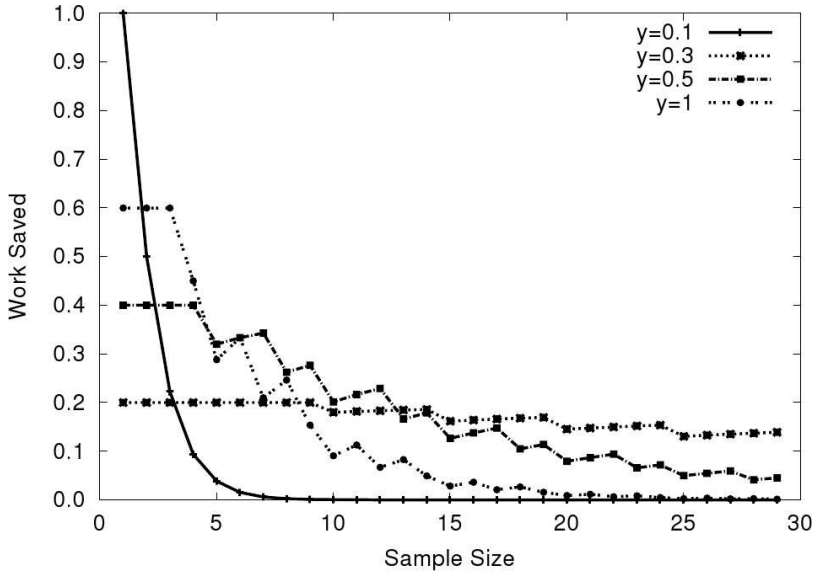


Figure 3.4: Work saved using EARL's Intra Iteration Optimization

3.1.9.3 Categorical and Weakly-dependent Data

In this section w.l.o.g., we assumed that N consists of numerical data and that f also returns numerically ordered results. Our approach, however, is also applicable to categorical data with a small modification discussed next.

The analysis of categorical data will involve the proportion of "successes" in a given population. The success can be defined as an estimate of the parameter of interest. Therefore, given a random sample of size n the number of successes X divided by the sample size gives an estimate for the population proportion p. This proportion follows a binomial distribution with mean p and variance $\frac{p(1-p)}{n}$. Because the binomial distribution is approximately normal, for large sample sizes, a z-test can be used for testing confidence interval and significance. This approach allows *EARL* to be applied even to categorical data.

We have also assumed that all samples contain i.i.d. data, however the bootstrap technique can be modified to support non-iid (dependent) data when performing resampling [ST95, Lah03, PW04, LL09]. The approach used to deal with b-dependent

data is usually called block-sampling. A data-set that is b-dependent contains $\frac{N}{b}$ blocks where each block $X_i, ..., X_{i+b}$ represents b inter-dependent tuples. Such dependency is usually present in time-series data. In the block based sampling instead of a single observation, blocks of consecutive observations are selected. Such a sampling method insures that dependencies are preserved amongst data-items. In section 3.2 we address the inter-dependent data problem in detail.

3.1.10 EARL Performance Evaluation

We have used Hadoop version 0.20.2-dev, to implement our extension and run the experiments on a small cluster of machines of size 5 containing Intel Core duo (CPU E8400 @ 3.00GHz), 320MB of RAM and Ubuntu 11.0 32bit. Each of the parts shown in Figure 3.1 are implemented as separate modules which can seamlessly integrate with user's Map-Reduce jobs. The sampler is implemented by modifying the *RecordReader* class to implement the *Pre-Map sampling* and extending the *map* class to implement the *Post-Map sampling*. The resampling and update strategies are implemented by extending the *Reduce* class. The results generated from resamples are used for result and accuracy estimation in the *AES* phase. The *AES* phase computes the coefficient of variation (c_v) and outputs the result to HDFS which is read by the main Map-Reduce job where the termination condition is checked. Because the number of required resamples and the required sample size are estimated via regression, a single iteration is usually required. Figure 3.5 shows an example of an MR program written using EARL's API. As can be observed from the figure, the implementation allows for a generic *user_job* to take advantage of our early approximation library.

The biggest implementation challenge with *EARL* was reducing the overhead of the *AES* phase and of the sample generation phase. If implemented naively, (i.e. making both the sampler and the *AES* phase its separate job) then the execution time would

```
public static void main(String[] args)
  throws Exception {
// Initialization of local variables
// ...
// ...
Sampler s = new Sampler();

while (error > sigma) {
 // path_string is the initial DataSet
 s.Init(path_string);
 // num_resamples of resamples of size
 // sample_size is generated. Both of these
 // variables are determined empirically.
 s.GenerateSamples(sample_size,
                   num_resamples);
 JobConf aes_job = new JobConf(AES.class);

 JobConf user_job;
 // For each sample we execute user_job
 for (int i = 0; i < num_resamples; i++) {
  user_job = new JobConf(
    MeanBootstrap.class);
  // Init of the user_job
  //...
  JobClient.runJob(user_job);

 // AES uses the input from user_job to
 // compute the
 // approximation error.
 // Init of the aes_job
 // ...
 // ...
 // The aes job also updates the err.
 JobClient.runJob(aes_job);
 // In cases where early approximation
 // is not possible, sample_size and
 // num_resamples will be set to N and 1
 // respectively.
 UpdateSampleSizeAndNumResamples();
 }
}
```

Figure 3.5: An example of how a job definition in EARL.

be inferior to that of the standard Hadoop especially for small data-sets and light aggregates where *EARL*'s early approximation framework has little impact to begin with. We wanted to make *EARL* light-weight so that even for light tasks, *EARL* would not add additional overhead and the execution time in the worst case would be comparable to that of the standard Hadoop.

The potential overhead of our system is due to three factors: (1) creating a new MR job for each iteration used for sample size expansion (2) generating a sample of the original dataset and (3) creating numerous resamples to compute a result distribution that will be used for error estimation. The first overhead factor is addressed with the help of pipelining, similar to that of Hadoop Online, however in our case the mappers also communicate with reducers to receive events that signal sample size expansion or termination. With the help of pipelining and efficient inter task communication, we are able to reuse Hadoop tasks while refining the sample size. The second challenge is addressed via the added feature of the mappers to directly ask for more splits to be assigned, in the case of pre-map sampling, when a larger sample size is required. Alternatively a sample can be generated using post-map sampling as discussed in Section 3.1.6. Post and pre-map sampling work flawlessly with the Hadoop infrastructure to deliver high quality random sample of the input data. Finally the last challenge is addressed via a resampling algorithm and its optimizations presented in Section 3.1.6. Re-sampling is actually implemented within a reduce phase, to minimize any overhead due to job restarts. Due to delta maintenance, introduced in Section 3.1.9, resampling becomes efficient and its overhead is tremendously decreased making our approach not only feasible but to deliver an impressive speed-up over standard Hadoop. Next key experiments are presented which showcase the performance of *EARL*.

A set of experiments measuring the efficiency of the accuracy estimation and sampling stages are presented in the following sections. To measure the asymptotic behav-

ior of our approach a synthetically generated data-set is used. The synthetic dataset allows us to easily validate the accuracy measure produced by EARL. The normalized error used for the approximate early answer delivery is 0.05 (i.e. our results are accurate to within 5% of the true answer). The experiments were executed on simple, single phase MR tasks to give concrete evidence of applicability of EARL, and more elaborate experiments on a wider range of mining algorithms is part of our future work.

3.1.10.1 A strong case for EARL

In this experiment, we implemented a simple MR task computing the mean, and tested it on both standard Hadoop and EARL. Figure 3.6 shows the performance comparison between these two. It shows that when the data-set size is relatively large ($> 100GB$), our solution provides an impressive performance gain (4x speed-up) over standard Hadoop even for a simple function such as the mean. In the worst case scenario, where our framework cannot provide early approximate results ($< 1GB$), our platform intelligently switch back to the original work flow which runs on the entire data-set without incurring a big overhead. It demonstrates clearly that EARL greatly outperforms the standard Hadoop implementation even for light-weight functions. Figure 3.6 also shows that a standard Hadoop data loading approach is much less efficient than the proposed pre-map sampling technique.

3.1.10.2 Approximate Median Computation

In this experiment, we did a break-down study, to measure how much a user defined MR task can benefit from resampling techniques and our optimization techniques. We used the computation of a median as an example, and tested it using three different implementations: (1) standard Hadoop, (2) original resampling algorithm, and (3) optimized resampling algorithm. Figure 3.7 shows that: (1) With a naïve Monte Carlo

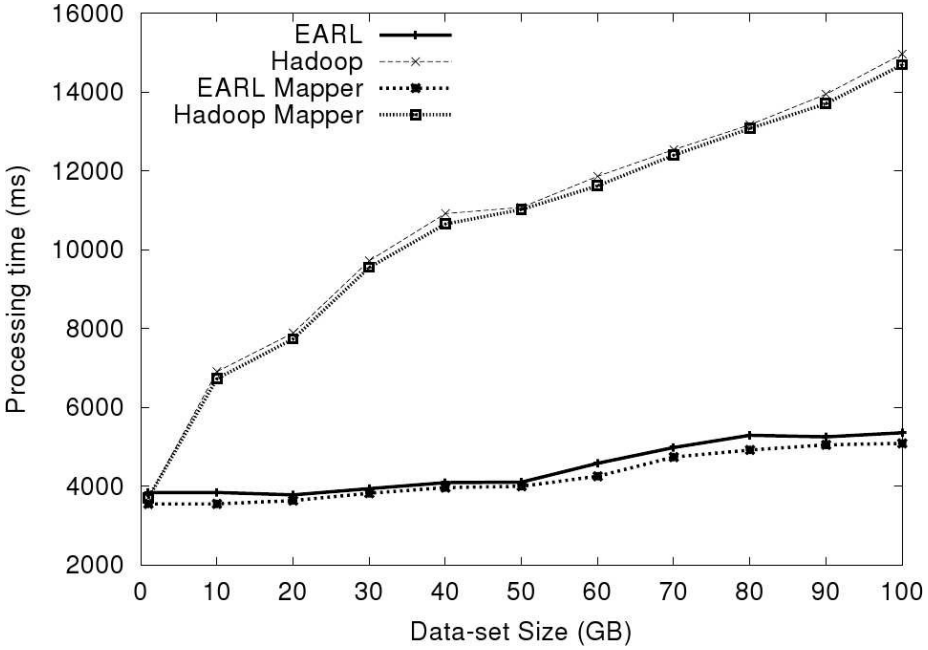


Figure 3.6: Computation of average using EARL and Hadoop

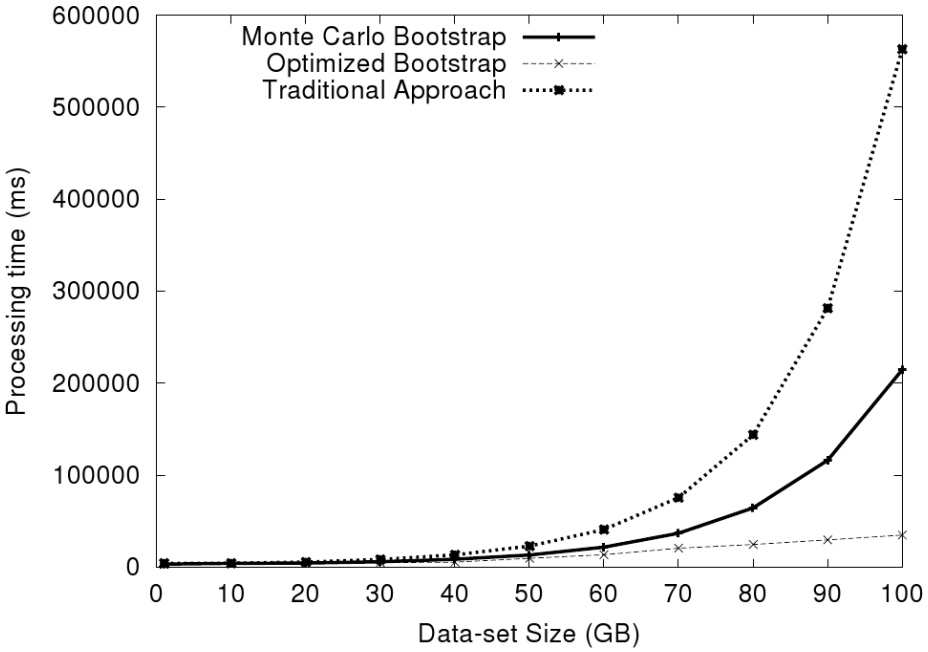


Figure 3.7: Computation of median using EARL and Hadoop

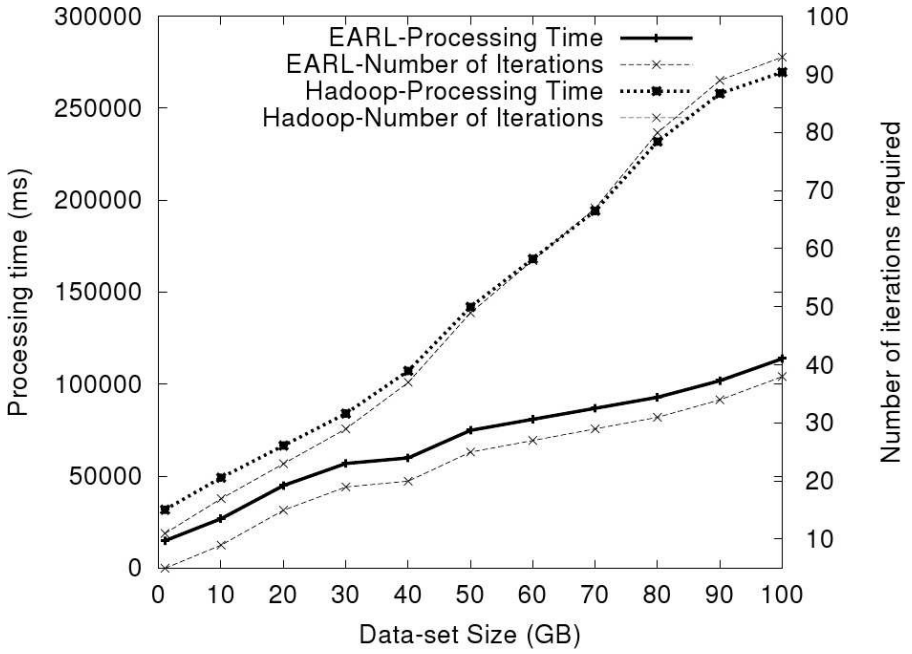


Figure 3.8: Computation of K-Means using EARL and Hadoop

bootstrap, we can provide a reliable estimate for median with a 3-fold speed-up, compared to the standard Hadoop, due to a much smaller sample size requirement. (2) Our optimized algorithm provides another 4x speed-up over the original resampling algorithm.

3.1.10.3 EARL and Advanced Mining Algorithms

EARL can be used to provide early approximation for advanced mining algorithms, and this experiment provides a performance study when using *EARL* to approximate *K-Means*.

It is well known that *K-Means* algorithm converges to a local optima and is also sensitive to the initial centroids. For these reasons the algorithm is typically restarted from many initial positions. There are various techniques used to speed up K-Means, including parallelization [ZMH09b]. Our approach, compliments previous techniques by speeding up K-Means without changing the underlying algorithm.

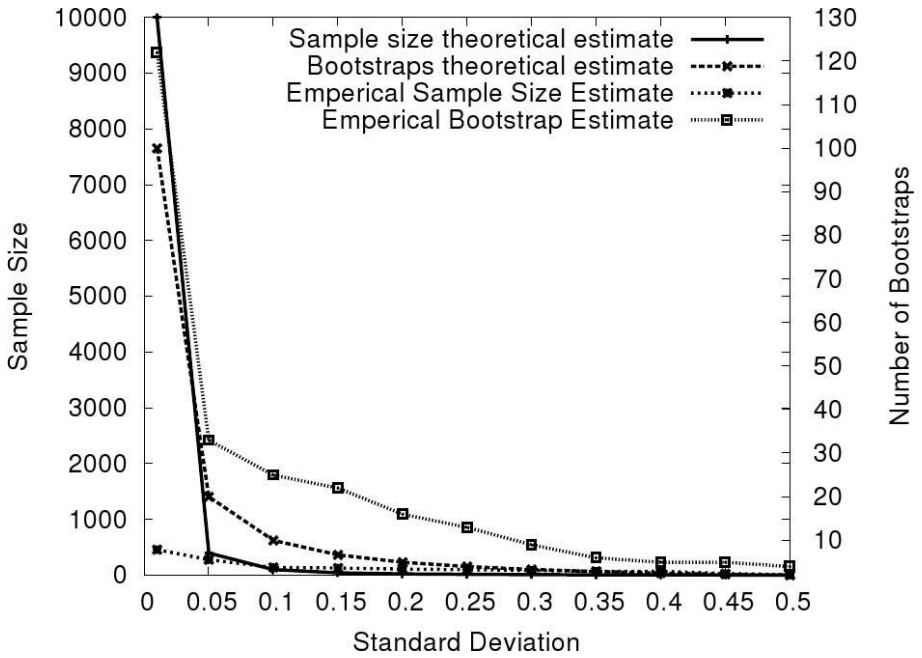


Figure 3.9: EARL's empirical sample size and number of bootstraps estimates compared to a theoretical prediction

Figure 3.8 shows the results of running *K-Means* with EARL and stock Hadoop. Our approach leads to a speed up due to two reasons: (1) K-Means is executed over a small sample of the original data and (2) K-Means converges more quickly for smaller data-sets. Because of a synthetic data-set, we were also able to validate that EARL finds centroids that are within 5% of the optimal.

3.1.10.4 Determining Sample size and Number of Bootstraps

In this experiment we measure how the theoretical sample size and the theoretical number of bootstraps prediction compare to our empirical technique of estimating the sample size and the number of bootstraps. We use a sample mean as the function of interest. Frequently, theoretical prediction for sample size is over estimated given a low error tolerance and is under-estimated for a relatively high error tolerance. Furthermore, theoretical bootstrap prediction frequently under-estimates the required number

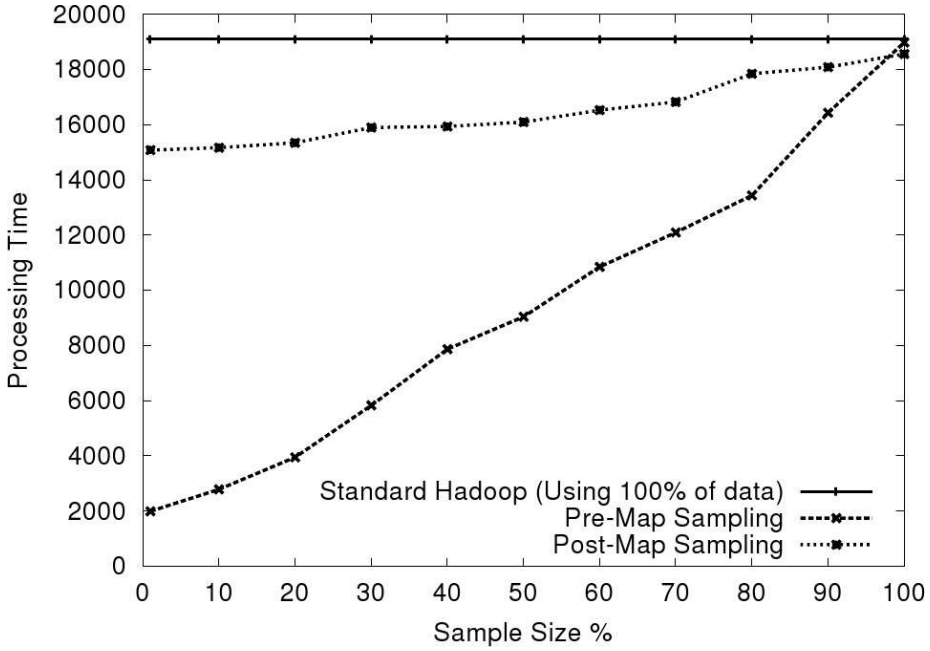


Figure 3.10: Processing times of pre-map and post-map sampling in EARL

of bootstraps. In other empirical tests we have observed cases where theoretical bootstrap prediction is much higher than the practical requirement. This makes a clear case for the necessity of an empirical way to determine the required sample size and the number of bootstraps to deliver the user-desired error bound. In the case of the sample mean, we found that for a 5% error threshold, a 1% uniform sample and 30 bootstraps are required.

3.1.10.5 Pre-map and Post-map sampling

In this experiment we determine the efficiency of pre-map and post-map sampling as described in Section 3.1.6 when applied to computation of the mean. Recall that pre-map sampling is done before sending any input to the mapper thus significantly decreasing the load-times and improving response time. The down-side of pre-map sampler is a potential decrease in accuracy of estimating the number of *key, value* pairs which may be required for correcting the final output. In post-map sampling,

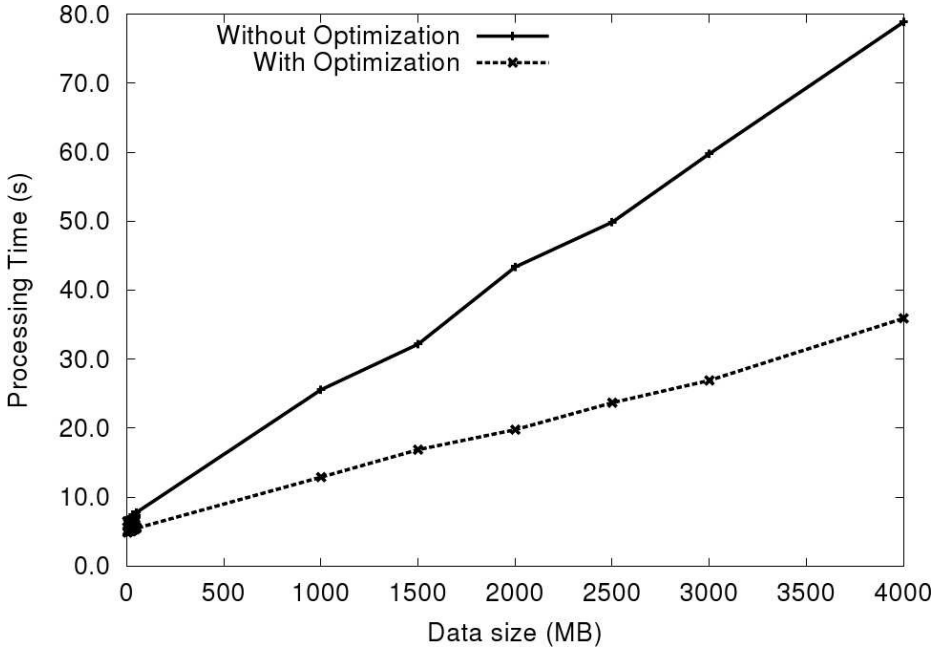


Figure 3.11: Processing times using EARL's delta maintainance procedure.

the sampling is done per-key, which increases the load-times but potentially improves accuracy of estimating the number of *key, value* pairs. As presented in Figure 3.10 the pre-map sampling is faster than post-map sampling in terms of total processing time. Furthermore, our empirical evidence suggests that for a large sample size, pre-map sampler is as accurate in terms of the number of *key, value* prediction as the post-map sampler. Therefore, to decrease the load-times, and to produce a reasonable estimate for functions that require result correction, the *pre-map* sampler should be used. The *post-map* sampler should be used when load-times are of low concern and a fast as well as accurate estimates of a function on a relatively small sample size are required.

3.1.10.6 Update Overhead

This experiment measures the benefit that our inter-iteration and intra-iteration (incremental processing) strategies achieve. Recall that in order to produce samples of larger sizes and perform resampling efficiently, we rely on delta maintenance as described in

section 3.1.9. Figure 3.11 shows the total processing time of computing the mean function with and without the delta maintenance optimization. The data-size represents the total data that the function was to process. The without optimization strategy refers to executing the function of interest on the entire dataset and with optimization strategy refers to execution the function on half of the data and merging the results with the previously saved state as described in section 3.11. The optimized strategy clearly outperforms the non-optimized version. The optimized strategy introduced achieved a speedup of close to 300% for processing a 4GB data-set as compared to the standard method.

3.2 Quality Assessment Using the Improved Bootstrap

In this chapter we propose a scalable method for assessing the quality of machine learning algorithms over sampled time-series data. While bootstrap provides a simple and powerful means of estimating accuracy, its application to large time-series data still suffers from scalability issues. As an alternative we introduce BLB-TS, a scalable extension of bootstrap for time-series which utilizes the recent advances in bootstrap and time-series theory to provide a practical implementation for assessing a time-series sample quality using Hadoop. For instance, our new procedure yields a robust and computationally efficient means of assessing the quality of our Twitter analytics workflow over large, real-world, time-series data.

3.2.1 Introduction

A large portion of today's data is in a time-series format (e.g., Twitter stream, system logs, blog posts), and time constraints as well as monetary constraints force the user to work on a sample of the data, for which the quality assessment is required. Further-

more the trend of dataset growth is accelerating, with 'big data' becoming increasingly prevalent. The original bootstrap approach [ST95, Efr79] and its time-series variants [BK99, PR94] provide a simple way of assessing the quality of an estimate, however in the distributed environment the cost of transferring the data to independent processors as well as the cost of computing a single resample can be high. This is the source of incessant problems in the natural life-cycle of advanced analytics development and a major stumbling block that prevents interactive result exploration that many users desire [LZZ12a, LK12]. Thus we need a scalable and efficient way of assessing the quality of an estimator for large time-series data samples in a distributed environment.

In this section we propose a simple, accurate and scalable technique for assessing the quality of results computed from samples of the real-world time-series data. This technique allows the average user or a small business to derive analytics on the data of massive sizes by using sampling. Therefore our method will go towards allowing everyone to gain access to and benefit from knowledge discovery.

Reliable estimation of the quality obtained from sampled data often leads to sample sizes that are orders of magnitude smaller than the original dataset thus providing major savings in terms of time and money [LZZ12a]. The average user, when debugging her machine learning algorithm locally can work with a sample of the data while remaining confident in the expected accuracy of her final result. When the same user deploys her application on the cloud system such as EC2[2] money and time can also be saved by only working with a sample of the data and using only the needed resources[3].

This section addresses the scalability issue of error estimation for sampled time-series data. We build on previous work of bootstrap [LZZ12a, KTS12b, KTS11] and time-series theory [BK99, PR94] to provide a scalable approach for assessing the quality of large time-series data samples. We take advantage of the trend in computational

[2]http://aws.amazon.com/ec2/

[3]The pricing of cloud services is based on (a) storage and (b) processing unit usage.

(a) A 40,000 ft. view of Twitter hashtag association computation.

(b) Details on ②. Bias, s.d. and variance are among the many statistical error measures that can be used.

Figure 3.12: An example of an analytics workflows over time-series data.

resources which is shifting towards a multicore and distributed architecture with cloud services providing thousands processing units[4]. These new processing units support fast execution of 'big data' applications. Motivated by this trend, we introduce the BLB-TS system that integrates recent research in both bootstrap and time-series theory. The core of our approach relies on the Bag of Little Bootstraps (BLB) [KTS12b] for bootstrap scalability, on the Stationary Bootstrap (SB) [PR94] for time-series resampling support and on Hadoop[5] for computational scalability. BLB-TS works by first selecting in parallel a random block b of the original time-series data sample of size n. Then in parallel r resamples are generated by applying SB to each block b to obtain a resample of size n. In our implementation, r is not fixed, and each node can converge independently which was a limitation in [KTS12b]. The errors computed on the resamples are then averaged across all nodes to produce the final accuracy estimate. Thus, in BLB-TS, the estimator is applied on only a single block b instead of on the entire time-series sample. In other words, the computational cost incurred only depends on the size of b and not on the size of the resample n.

The difficulty in supporting fast and accurate estimator assessment based on a sample has long been recognized as one of the major obstacles in interactive big data processing [KTS12b, ELZ10a]. The problem is more complicated when the data are a time-series because bootstrap must be carried out in a way that captures the depen-

[4]The total number of processing units in a cluster is: number of nodes $\times$ the number of CPUs per node.

[5]hadoop.apache.org

dence structure of the data generation process. As a concrete application of BLB-TS we use one of our own Twitter analytics workflows which is composed of four stages: ① Get latest Twitter data of size $W \rightarrow$ ② Estimate error based on a sample n of $W \rightarrow$ ③ Compute the similarity between Twitter hashtags[6] from $|s| \rightarrow$ ④ Update internal tag similarity model. Note that we compute similarity across more than 2 Million Twitter tags which further motivates this research. Step ② is computed infrequently, only to estimate the sample size required to achieve a user defined accuracy. The goal of this workflow is to compute a similarity score of all hashtag pairs based on a state of the art algorithm called MIC [RRF11]. Chapter 6.1 will provide more details on this workflow. While all stages of this workflow provide interesting challenges, this section is specifically on step ②. Figure 3.12 further elucidates this workflow and emphasizes the focus of this section.

To the best of our knowledge, this work is the first to provide scalable bootstrap with favorable statistical guarantees. Our preliminary experiments show a promising trend when applying BLB-TS to the analytics workflow shown in Figure 3.12 and to other ML algorithms such as classification and regression.

Notation: Let $\{X_i\}_{i=-\infty}^{\infty}$ be an $\mathbb{R}^d$-valued stationary process observed from some underlying distribution P with $\mathbb{P}_n = n^{-1} \sum_{i=1}^{n} \delta_{X_i}$ denoting the corresponding empirical distribution. Let $\mathcal{X}_n = \{X_1, ..., X_n\}$ denote the available observations (usually drawn i.i.d. from $\{X_i\}$). Based on $\mathcal{X}_n$ we compute an estimator $\hat{\theta}_n$ of the parameter of interest θ. For example $\hat{\theta}_n$ may estimate the accuracy of a classifier. Borrowing notation from [KTS12b] we define $Q_n\{P\} \in \mathbb{Q}$ as the true distribution of $\hat{\theta}_n$. Thus our final goal is to estimate $\xi(Q_n\{P\}, P)$. In our case ξ denotes the coefficient of variation but it can also denote bias, standard error, confidence region or a quantile. Sample size estimation is driven by the user's threshold on ξ. Using bootstrap with Monte

[6]The # symbol, called a hashtag, is used to mark keywords or topics in a Tweet.

Carlo approximation [Efr79] one can repeatedly resample n points i.i.d. from $\mathbb{P}_n$ from which an empirical distribution $\mathbb{Q}_n$ is formed and from which $\xi(Q_n\{P\}, P) \approx \xi(\mathbb{Q}_n)$ is approximated.

To preserve the inter-tuple dependency, we employ a block resampling technique called the Stationary Block bootstrap (SB) [PR94]. Letting the expected size of the block be l which is an integer satisfying $1 < l < n$, SB functions as follows: we let $L_{ni} \equiv L_i, i \geq 1$ be conditionally i.i.d. random variables with parameter $p = l^{-1} \in (0, 1)$. Also let $\mathcal{I}_1, ..., \mathcal{I}_n$ be conditionally i.i.d. random variables with the discrete uniform distribution on $\{1, ..., n\}$. Then, the SB resample $X_1^*, ..., X_n^*$ is produced from the first n elements in the array $\mathbb{B}(\mathcal{I}, L_1), ..., \mathbb{B}(\mathcal{I}, L_{\mathbb{K}})$ where $\mathbb{K} \equiv inf\{\kappa \geq 1 : L_1 + ... + L_\kappa \geq n\}$ and where the block $\mathbb{B}(i, \kappa) = (X_i, ..., X_{i+\kappa-1}), i \geq 1, \kappa \geq 1$.

3.2.2 BLB-TS: The Scalable Time-Series Resampling Algorithm

In this section, we first discuss the limitations of a naïve approach for quality assessment of a time-series sample and then we present the new BLB-TS approach.

3.2.2.1 The Naïve Approach

Predicting the quality of an estimate over time-series can be done in an embarrassingly parallel way using bootstrap variations for time-series [SPR91, BK99, PR94]. The time-series data is sent to all nodes on which block sampling is then applied. Since only a certain number of resamples is needed, the computation time scales down as the number of nodes is increased. This naïve approach is inefficient because it sends the entire dataset to all machines, which can still be impractical for large datasets. Authors in [KTS12b] address this concern for i.i.d. data by splitting the sample into chunks and processing each chunk in parallel thus avoiding sending the whole dataset to all nodes. The limitations of the solution proposed in [KTS12b] are: (i) it is only applicable to

Variable	Definition
t_i	The tuple count for a mapper m_i
m_i	Mapper i.
p_i	The split provided to mapper i.
b	The sample that each reducer receives.
s	The number of reducers such that $s \times \lvert b \rvert = n$
k	A reducer chosen randomly s.t. $s > k \geq 0$
<blk_start, blk_end>	The start and end of the block picked from split p_i
n	Sample Size
S	Sample
S'	New Sample

Table 3.2: BLB-TS: Variables used in Algorithm 3 and Figures 3.13a and 3.13b.

i.i.d. data and (ii) it treats all block samples as equal. Treating all block samples as equal can be inefficient in practice because some nodes may get a bad block sample and require more resamples, r, to converge. The block quality is measured using the Kendell Test described in Section 3.2.2.3. Thus setting a fixed r, as is done in [KTS12b], may lead to inefficiencies. To address (i) and (ii) we propose our BLB-TS approach next.

3.2.2.2 BLB-TS Algorithm

Algorithm 3 presents the BLB-TS algorithm. The algorithm starts by selecting s random blocks from a set of possible blocks $\mathbb{B}(\mathcal{I}, L_1), ..., \mathbb{B}(\mathcal{I}, L_{\mathbb{K}})$ observed in the time-series sample. The stationary bootstrap is then applied to each block b by appending the next point (wrapping around if necessary) in the series to the current block with probability $1 - p$, and appending a random point from b with probability p. The act

Algorithm 3: BLB-TS

Input: Time-series data sample $X_1, \ldots, X_n$

$\hat{\theta}$: Estimator of interest

ξ: Estimator quality assessment
b: Selected subsample

s: Number of sampled blocks

r: Max. number of Monte Carlo iterations
Output: An estimate of $\xi(Q_n(P))$

for $j \leftarrow 1$ **to** s **do**

Randomly select a block b from $\mathbb{B}(\mathcal{I}, L_1), ..., \mathbb{B}(\mathcal{I}, L_{\mathbb{K}})$

for $k \leftarrow 1$ **to** r **do**

while *Size of sample* $\leq n$ **do**

if $rgeom(1, p) == 0$ **then**

currentIndex $\leftarrow$ A random index in b

pick the next item as $b[currentIndex]$ and continue

end

pick the next item in our sample as $b[currentIndex$++$]$

end

$\mathbb{P}^*_{n,k} \leftarrow n^{-1} \sum_{a=1}^{b} n_a \delta_{X_{i_a}}$

$\hat{\theta}^*_{n,k} \leftarrow \hat{\theta}(\mathbb{P}^*_{n,k})$

check if $\hat{\theta}^*_{n,k}$ converged and if so, append $(r - k)$ estimates

to $\hat{\theta}^*_{n,k}$ drawn i.i.d. from $\hat{\theta}^*_{n,k}$ and break

end

$\mathbb{Q}^*_{n,j} \leftarrow r^{-1} \sum_{k=1}^{r} \delta_{\hat{\theta}^*_{n,k}}$

$\xi^*_{n,j} \leftarrow \xi(\mathbb{Q}^*_{n,j})$

end

return $s^{-1} \sum_{j=1}^{s} \xi^*_{n,j}$

of selecting a 0 with probability p is captured by the *rgeom* method in Algorithm 3. Note that although other time-series approaches are applicable, we focus on SB due

to its popularity and its robustness with respect to the block size [PR94]. The error $\xi(Q_n(\mathbb{P}_{n,b}^{(j)}))$ for each block b is estimated by applying the Monte Carlo approximation via repeatedly resampling the blocks using the SB method and computing the estimator of interest r on each resample. The number of resamples r is locally determined by each node with $O(r) \approx 100$ in our experiments. Local convergence is important because two nodes, having subsamples of different quality, should not have the same r. Thus by locally determining r for each s, BLB-TS turns out to be less sensitive to the quality and to the size of b than SB. More details on local convergence are given in section 3.2.2.3. Note that if a local node converges after m resamples, BLB-TS must append $r - m$ resample estimates to $\hat{\theta}_n^*$; this is done by drawing $r - m$ resample estimates i.i.d. from $\hat{\theta}_m^*$.

Observe that r is dependent on $|b|$ and at convergence can be as low as 1-5 for $|b| = n^{0.9}$ or 10-20 for $|b| = n^{0.5}$ which is consistent with results for i.i.d. data found in [KTS12b]. The r resample estimates then form an empirical distribution which is used to approximate $\xi(Q_n(\mathbb{P}_{n,b}^{(j)})) \approx \xi(\mathbb{Q}_{n,j})$. The quality estimates are then averaged across all the nodes and final error is returned as $s^{-1} \sum_{j=1}^{s} \xi_{n,j}^*$.

3.2.2.3 BLB-TS Optimizations

Varying the number of resamples r: To compute the exact bootstrap variance estimate, $\binom{2n-1}{n-1}$ resamples are required, which for $n = 15$ is already equal to 77×10^6. Therefore, an approximation is necessary to make the bootstrap technique feasible. The Monte-Carlo [ST95] is the standard approximation technique used for resampling methods including the bootstrap that requires less than $\binom{2n-1}{n-1}$ resamples. It works by taking r resamples resulting in variance estimate of $\hat{\sigma}_r^2 = \frac{1}{r} \sum_{i=1}^{r} (\hat{\theta}_r^* - \hat{\theta}^*)^2$ where $\hat{\theta}^*$ is the average of $\hat{\theta}_r^*$'s. The theory suggests that r should be set to $\frac{1}{2}\epsilon_0^{-2}$ [Efr87], where ϵ_0 corresponds to the desired error of the Monte Carlo approximation with respect to

the original bootstrap estimator. Our experiments in [LZZ12a] for i.i.d. data show that a much lower value of r can be used in a great majority of practical applications. As a rule of thumb, for i.i.d. data we found that 30 bootstraps are enough for most applications [LZZ12a] (see section 3.1).

For time-series data, however, setting a fixed r for all nodes is inefficient because not all nodes receive the same quality block sample. In our experiments we control the quality of a time-series sample by using the Kendell Test which is commonly used to determine the probability p that a sampled distribution X came from the original distribution Y [SS07]. Therefore a 'bad' time-series sample would correspond to a relatively low p. This test is non-parametric, as it does not rely on any assumptions on the distributions of X or Y or the distribution (X, Y).

Thus we propose an adaptive algorithm to select r. Similar to [KTS12b], we select r by continuously processing more resamples and updating $\hat{\theta}^*_{n,k}$ until it ceases to change significantly. Thus, r in Algorithm 3 is used as an upper-bound, and each node uses a threshold ϵ for local convergence checking. Despite some nodes converging quicker than others, BLB-TS forces all nodes to produce the same number of error estimates by enlarging $\hat{\theta}^*_n$ to r via i.i.d. resampling from $\hat{\theta}^*_{n,k}$ if necessary. Authors in [KTS12b] suggest a similar approach, however no experiments are provided. We present experimental results of our approach in section 3.2.3.3.

3.2.2.4 I.I.D. and Time-Series Sampling in BLB-TS

Recall that BLB-TS needs to generate s random samples of size $|b|$ such that $|b| \times s = n$. To solve this problem we draw from past work on memory-resident and disk-resident sampling [OR90].

Consider the HDFS architecture where a file is divided into several blocks and where each block is typically 64MB. A naïve single-sampling solution is to pick a

set of blocks B_i at random, possibly splitting B_i into smaller splits, to satisfy the required sample size. This strategy however will not produce a uniformly random sample because each of the selected B_i can contain dependencies (e.g., consider the case where data is clustered on a particular attribute resulting in clustered items to be placed next to each other on disk due to spatial locality). Another naïve single-sample generating solution is to use a reservoir sampling algorithm to select k random items from the original data-set. This approach produces a uniformly random sample, but it suffers from slow loading times because the entire dataset needs to be read, and possibly re-read when further samples are required. Below we present the i.i.d. and time-series sampling algorithms that run over HDFS, generate s random samples and avoid the problems of the naïve approaches presented above. For convenience, the variables used in the algorithms below are explained in Table 3.2.

In the case of i.i.d. data, as an input, each mapper m_i will receive a split p_i which will have a set of tuples t_i. For each tuple $t \in t_i$, mapper m_i will output $< k, t >$, where k is a random number such that $s > k \geq 0$ and where s is the total number of reducers. The mapper function has the complexity of $O(t_i)$. The reducer for i.i.d. sampling simply evaluates the estimator of interest $\hat{\theta}$ at most r times and has the complexity of $O(\hat{\theta}(b)r)$.

When computing a random sample of time-series data the dependency between tuples must be captured. Thus, for each split the mappers perform block sampling and output a $< k, < blk\_start, blk\_end >>$ where k is defined as before and blk_start, blk_end denote respectively the start and end of the selected block from split p_i. When performing block sampling, b may span multiple splits which are on different mappers. Therefore, each mapper m_i for each split p_i must send the first b tuples to each s reducers. The mapper and reducer are shown in Figures 3.13a and 3.13b respectively. The complexity of the mapper is $O(|b|s + t_i)$ and the complexity of the reducer is

Input: A split p_i

Output: Random Sample

while *Size of sample* $\leq |b|$ **do**

  if $rgeom(1, p) == 0$ **then**

    currentIndex $\leftarrow$ A random index in b

    pick the next item as $b[currentIndex]$ and continue

    write($< k$,current sample$>$)

  end

  pick the next item in our sample as $b[currentIndex$++$]$

end

for $i \leftarrow 1$ **to** s **do**

  send_first_b_tuples_to(i)

end

(a) Mapper for time-series sampling

Input: A set of random blocks

Output: r models

block $\leftarrow$ empty()

for *For each received block* b_i **do**

  `// append() joins blocks together taking care of inter-map spanning blocks.`

  block $\leftarrow$ append(block, b_i)

end

while ξ *not converged* **do**

  bnew = Resample block

  M.add($\hat{\theta}$(bnew))

  update ξ

end

write($<$output_key, $M >$

(b) Reducer for time-series sampling

Figure 3.13: Mapper and Reducer algorithms for time-series sampling $O(\hat{\theta}(b)r)$.

3.2.3 BLB-TS Performance Evaluation

3.2.3.1 Implementation

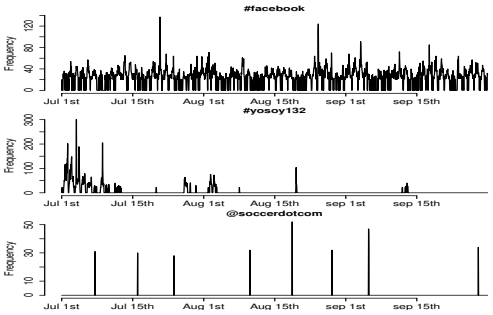
The performance of the current implementation of BLB-TS relies on the two design decisions of adopting: (1) Model-error computation separation and (2) Convergence aware computation. The first decision proved invaluable for achieving a simple API design that is easy to understand and test. In model-error computation separation the

error computation is performed in three stages: (a) Model computation (b) Model Validation and (c) Error derivation from empirical distribution of (b). It is important to realize that a lot of machine learning algorithms fall under this three-stage model, and the quality assessment of many classic machine learning algorithms (SVM, regression) can easily be expressed using it. Decision (2) is also important because in providing block samples, not every block will be of the same 'quality', error computation on some nodes may actually converge faster; therefore in our implementation r in Algorithm 3 is actually used as an upper-bound, and each node instead uses a threshold for local convergence checking. Note however that all nodes produce the same number of error estimates because we enlarge $|\hat{\theta}_n^*|$ to r via iid resampling if necessary.

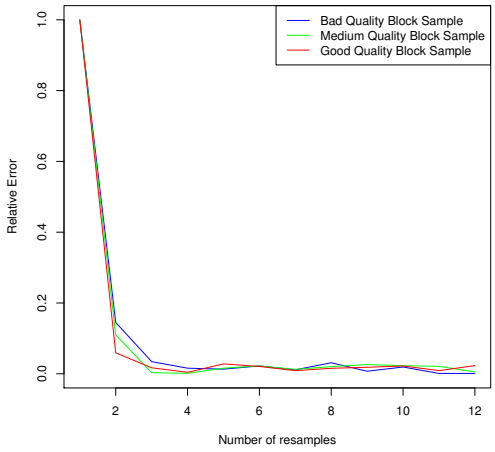
3.2.3.2 Experiments

All experiments were performed on a 7-node cluster with 140 map-slots on Hadoop 1.0.4. Each node is a 16-core AMD Opteron Processor 6134 @ 2.3GhZ with 132GB RAM. The Total Twitter Dataset is of size 4.3TB.

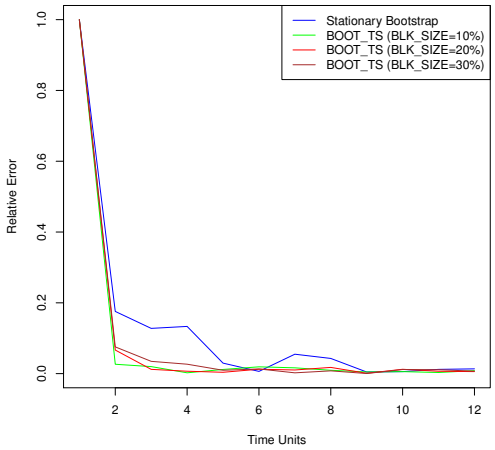
In Figure 6.3a we show the type of time-series BLB-TS was designed to deal with. Figure 6.3a shows only three time-series for three tags over a period of three months. Our experiments in production, however, were run on two million tags, with the goal of determining similarity measure across all $\frac{(n-1)(n)}{2}$ hashtag pairs. Note that some tags may have many random spikes (e.g., #riot), some tags may be fairly stable (e.g., #youtube) and some may show very little level of activity (e.g., #mycatsnameisboris). In Figure 3.14c we show the convergence rate of BLB-TS and of Stationary Bootstrap. BLB-TS succeeds in converging to a low relative error significantly faster than the Stationary Bootstrap. Although here we only show convergence results for similarity computation, we observed similar results for other machine learning tasks such as SVM and linear regression as discussion in the application chapter of the dissertation.



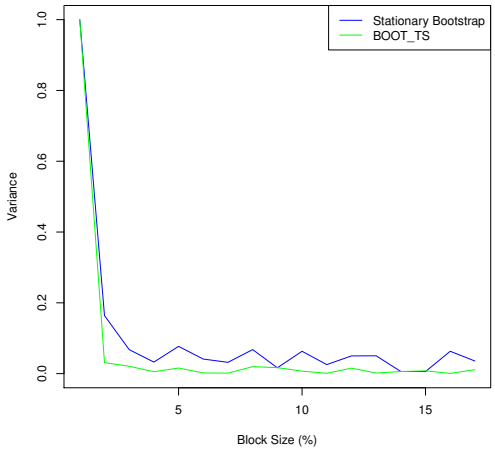
(a) Sample time-series of several hashtags.



(b) Different quality samples require different time to converge.



(c) BLB-TS succeeds at converging faster than the Stationary Bootstrap for a fixed $|b|$.



(d) BLB-TS is less dependent on the size of b than the Stationary Bootstrap.

Figure 3.14: BLB-TS Experiments against the Stationary Boostrap

In Figure 3.14d we show the block size dependence on the estimated variance of the Stationary Bootstrap and of the BLB-TS. Notice that the variance of the BLB-TS is much more stable than that of the Stationary Bootstrap and therefore in practice the user should be much less concerned with picking the perfect block size.

3.2.3.3 Convergence Rate

Varying r: This experiment measures the convergence rate when the quality of the block is varied. Our goal was to test our hypothesis that more resamples are required given a bad vs. a good time-series sample. Recall that the quality of the block sample is controlled by the Kendall test which is widely used by others [SS07]. The Kendall test assigns the probability p that the sample came from the original dataset. The results are shown in Figure 3.14b. Although the results may seem insignificant, in practice we found that for a relatively poor sample with $p < 0.2$ the required r can be as much as 10 times greater than for a relatively good sample with $p > 0.9$.

Fixed Block Size: Next we measure the convergence rate of BLB-TS when the size of the block is kept constant. From Figure 3.14c we observe that BLB-TS succeeds in converging to a low relative error significantly faster than SB, which means BLB-TS requires a smaller value of r than SB. The reason for this is because SB uses b whereas BLB-TS resamples b n times resulting in lower variance.

Fixed Number of Resamples: Motivated by our results in Figure 3.14c we suspected that BLB-TS will also be less dependent on the value of b. Figure 3.14d varies the block size and estimates the variance for both BLB-TS and SB. Notice that the variance of the BLB-TS is much more stable than that of SB and therefore in practice the data analyst should be much less concerned with picking the perfect block size.

3.3 Conclusion

A key part of big data analytics is the need to collect, maintain and analyze enormous amounts of data efficiently. With such application needs, frameworks like MapReduce are used for processing large data-sets using a cluster of machines. Current systems however are not able to provide accurate estimate of incremental results. In this chap-

ter we presented EARL which is a non-parametric extension of Hadoop that provides a way to deliver early results with a reliable accuracy. EARL can be applied to improve efficiency of fault-tolerance techniques by avoiding the restart of failed nodes if the desired accuracy is reached. Our approach relies on resampling techniques from statistics. To further improve EARL, we introduced optimizations to the resampling methods. to make our framework even more attractive. We also introduced sampling techniques that are suitable for a distributed file-system. Our empirical results suggest an impressive improvement in processing times for the various scenarios. Based on our experimental results, we are confident that this is a promising method for providing the interactive support that many users seek when working with large data-sets. In future work we will investigate the use and the range of applications of other resampling methods (e.g., jackknife) that although are not as general and as robust as bootstrapping can provide better performance in specific situations.

Also in this chapter we improve EARL by making it scalable and adaptable to time-series. We have discussed BLB-TS and shown that it is a promising approach to scalable time-series sample quality assessment. We have shown preliminary results of its superiority to the Stationary Bootstrap and found that using BLB-TS we were able to discover some interesting patterns in our dataset, that would otherwise be impossible due to an impractically large dataset. Our work on BLB-TS will continue in the direction of providing more rigorous statistical guarantees of BLB-TS' quality assessment.

CHAPTER 4

Classifier Sample Size Prediction using the Learning Curve

In this chapter we rely on a nonparametric accuracy estimation techniques developed in sections 3.1 and 3.2 to speed-up Support Vector Machine (SVM) classifier by accurately predicting the learning curve and choosing the appropriate training set size required for achieving the relative accuracy specified by the user. While SVM is a popular classification tool, its applicability is limited by the fact that the training is based on solving a quadratic programming (QP) problem which is slow for large datasets. We provide the necessary theoretical backing that extends our approach to other learning algorithms. This section describes the library called the Fast Accurate Classifier Training (FACT) library used in local and distributed environments. In both environments, FACT provides orders of magnitude classification speed improvement whereby a very accurate prediction on a 4.3TB Twitter dataset is achieved by only using 1% of the data.

4.1 Chapter Introduction

Motivation: Here, we try to answer the following question: What training sample size yields a given classifier accuracy? Better yet we try to describe the classification error as a function of the desired confidence, the variants of the classifier, the complexity of

the task to be learned and the number of training examples. This question is important because analytics today run over an increasingly large amount of data, and in particular for classifiers, training a model on a large dataset containing thousands of features currently takes days even on a powerful cluster[1]. In this chapter we show that often only a fraction of a dataset is required to construct an accurate classifier.

We are motivated by the growing data-size/processing-speed gap, previously shown in Figure 1.1. This gap is a major problem today when performing big scale analytics, for instance, over Twitter and Google trends data. Furthermore, ambitious learning tasks such as major event prediction (e.g., stock market crashes, large social uprisings etc) require even more data aggregated from multiple sources. The complexity of such learning tasks will only increase, therefore new tools and methods are required that bridge the gap by stopping the classifier learning process early when the user prescribed accuracy had been reached.

Thus, we propose a method and a system that optimizes the classifier training on a massive dataset. We picked a classification task, as opposed to an unsupervised learning task, as a target for optimization because there is a well agreed measure of classification accuracy, whereas in unsupervised learning the measure of accuracy is not well defined and therefore optimization can be difficult. The optimization minimizes the computation time while taking the user prescribed model accuracy into account. To estimate the accuracy we explore and apply powerful methods and models developed in statistics to compute the classification accuracy obtained from training data [ST95, Efr79]. Our approach is supported by the Fast Accuracy Classifier Training (FACT) library that we developed for local and distributed environments. As an underlying distributed system we use Hadoop [had]. To the best of our knowledge our system is the first complete implementation for early classifier training.

[1] In our lab, an 8 machine state of the art cluster takes a day to train an SVM model, that classifiers the embedded emotion in a *tweet*, on a medium sized (1GB) Twitter dataset.

FACT consists of three components that help answer the underlying question of this chapter: *How does one make a classifier run faster on a large data-set given a user-prescribed accuracy requirement?* The three components of FACT are: (i) the classifier accuracy estimator (ii) the optimizer of the accuracy estimator and (iii) the learning curve predictor. The first and second components provide a quick classification accuracy estimation for a specific data size and these components were discussed in chapter 3. The third component extrapolates the classification accuracy for data-sizes that might be impractical to deal with directly. Together the three components of FACT allow for fast classifier training over bigdata.

Background: The first line of research pursuing similar objectives is that of Lee et al. [LH07], where uniform random sampling is used to train a classifier model. The approach in [LH07] however is application specific and does not take the user-specified accuracy requirement into account. Our approach, however, seeks a technique that is nonparametric (i.e., parameter-less) and supports early termination based on the user accuracy requirement.

FACT is effective for a wide range of classifiers but for practical purposes we focus on a Support Vector Machines (SVM) classifier, and provide a theoretical backing for FACT's applicability to other classifiers. The SVM concept is used to analyze and recognize patterns for classification and regression analysis. For example the input to the standard SVM can be a dataset for which a binary class can be predicted as an output. Therefore, given a set of training examples together with the corresponding categories the SVM training algorithm builds a model which predicts the class of testing data. An SVM model represents training examples as points in space such that a clear line can be drawn between the points belonging to different classes. The optimization approaches presented in this chapter take advantage of SVM's incremental computation support. For other types of classifiers, such as Decision Trees, incremental compu-

tation represents a more difficult task, and thus we leave the detailed study of those classifiers for future work. Building an SVM model for a large set of examples is time consuming and therefore we seek to find the minimum training size to satisfy the user-specified model accuracy requirement. Next, we introduce key ideas of FACT and we recollect the SVN algorithm in Section 4.2.1

FACT is based on our previous work [LZZ12b], which is discussed in chapter 3 where a resampling approach, called the bootstrap [ST95], is used to support approximation for arbitrary functions with no assumption on the underlying dataset distribution. The fundamental idea that resampling can be used for error estimation is carried over from [LZZ12b]. In short, using resampling one can estimate a sampling distribution of a sample statistic from a single sample s by repeatedly executing a function of interest on s. Our work differs from [LZZ12b] in the following ways: we provide (i) an optimized extension of [LZZ12b] to classification, (ii) techniques to speedup the SVM error estimation and (iii) classifier error prediction (i.e., learning curve prediction) in addition to point-based error estimation. More details on resampling can be found in Section 2.

We implement our approach in Hadoop. Hadoop is a natural candidate for implementing FACT. That is to say, while our fast accurate classifier approximation is platform independent, it benefits from the fundamental Hadoop infrastructure. Hadoop employs a data *re-balancer* which distributes HDFS [had] data uniformly across the DataNodes in the cluster. Furthermore, in a MapReduce framework there are a set of $<$*key*, *value*$>$ pairs which map to a particular reducer. This set of pairs can be distributed uniformly using random hashing and by choosing a subset of the keys at random, a uniform sample can be generated quickly. These two features make Hadoop a desirable foundation for *FACT*, while Hadoop's popularity maximizes the potential for practical applications of this new technology.

Challenges: Designing a reliable classifier accuracy estimator was our primary concern. Previous work for modeling accuracy prediction in both theoretical and practical sense are ill-suited for today's fast-paced business environment. The pioneering work of Vapnik et. al. [Vap99] introduces a VC (Vapnik-Chervonenkis) dimension which is a measure of the capacity of a statistical classification algorithm. The VC dimension has utility in statistical learning theory, because it can predict a probabilistic upper bound on the test error of a classification model. However, because the VC dimension theory is general it must also deal with the pessimistic case which results in a very loose upper bound for error estimation making the VC dimension approach to error estimation unhelpful for the average case. The empirical accuracy measure of classifier is usually estimated using a cross-validation technique [HTF09]. Cross-validation, however, is parametric and does not provide a reliable confidence interval of the error estimation. FACT addresses the above challenges by providing an empirically tight, nonparametric, classifier accuracy estimation for the given training size.

As described in chapter 3 computing all possible resamples of a sample is infeasible for accuracy estimation. In principle computing all $\binom{2n-1}{n}$ values of a statistic, where n is the sample size, to obtain the "ideal" bootstrap is possible, but is computationally difficult. Even for $n = 15$, there are already 77558760 distinct samples. The workaround for this problem involves using the Monte Carlo simulation to approximate the bootstrap by only drawing B number of resamples. This however is still not enough and to make bootstrap practical, in section 4.5 we present resampling optimization techniques which take advantage of the incremental computation in the context of classification.

Lastly it was necessary to preform classifier accuracy prediction (i.e. constructing the learning curve) which is a key challenge for large data classification. Accuracy as a function of training size is referred to as a *learning curve* (LC). The shape of LC often

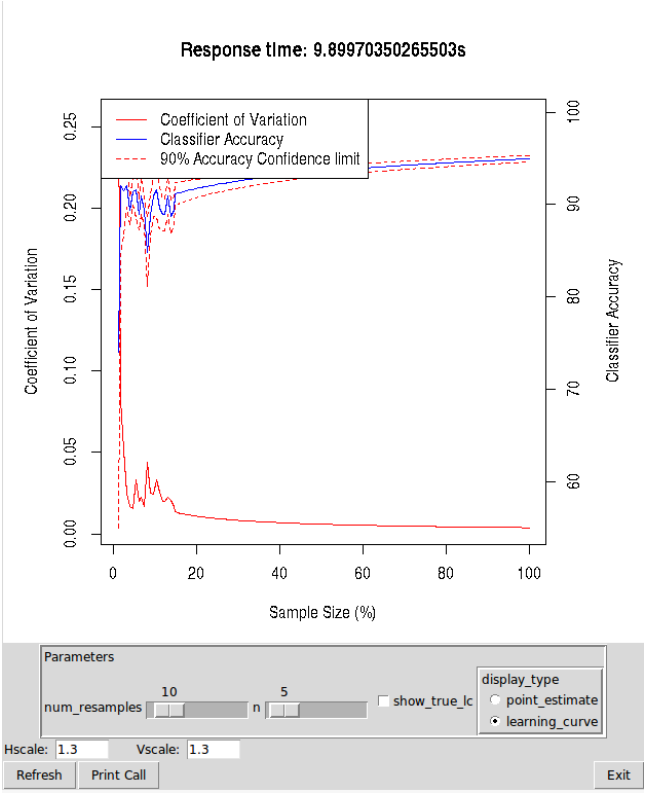


Figure 4.1: The interface of our system *FACT*. This screenshot shows FACT's prediction of the learning curve of the Twitter dataset.

follows a power law. Thus, we wish to estimate the parameters of the power law curve by computing only k points. The challenge was to find how k related to prediction accuracy and if the power law assumption works on real-world datasets. In other words, before this project it was unclear how good our prediction of the convergence rate of the power law curve will be by only computing k points. A crucial result in this field is the *central limit theorem* (CLT) which shows that for any random sample from any probability distribution with finite mean and finite variance the sample distribution converges to a normal distribution $N(0, 1)$. The CLT, however, does not say anything about the convergence rate. Nevertheless, we will show that using simple methods we are able to provide a powerful accuracy prediction tool to the user. The interface of our tool is shown in Figure 4.1.

Contributions: Therefore our approach addresses a very pressing problem in the real-world 'big data' analytics: *How to minimize the time and sample size required to achieve a given user prescribed accuracy?* In answering this challenge we make the following three contributions:

1. A general early classifier approximation method is introduced with a specific application for providing early SVM model training with a reliable classification error estimation. The method is platform independent and can be used for processing large data-sets on many systems including Hadoop, Teradata, R, and others. A Fast Accurate Classifier Training (FACT) library is implemented for Hadood and used for the experimental validation.

2. A new methodology is introduced that can provide a tighter empirical estimate of the learning curve prediction to determine the required training size for the prescribed user accuracy.

4.2 SVM Algorithm and More Background

4.2.1 SVN Algorithm

Let the element of the training data be $x_i \in \zeta$, $\zeta = \Re^d$, $i = 1, ..., l$, and the class labels $y_i \in \{\pm 1\}$. A mapping Φ is performed by SVM into a high, potentially infinite, dimensional Hilbert Space $\mathfrak{H}$ where $\Phi : \zeta \to \mathfrak{H}, x \to \bar{x}$. In $\mathfrak{H}$ the decision rule of SVM is a separating hyperplane with normal $\bar{\Psi} \in \mathfrak{H}$ which separates the x_i into two classes:

$$\bar{\Psi} \cdot \bar{x_i} + b \geq k_0 - \xi_i, y_i = +1 \tag{4.1a}$$

$$\bar{\Psi} \cdot \bar{x_i} + b \geq k_1 - \xi_i, y_i = -1 \tag{4.1b}$$

where k_0 and k_1 are usually defined to be $+1$ and -1 respectively, and where ξ_i are the slack variables to handle the non-separable case. The $\bar{\Psi}$ is computed by minimizing the objective function,

$$\frac{\bar{\Psi}\bar{\Phi}}{2} + C\left(\sum_{i=1}^{l}\xi_i\right)^p \text{ s.t. 4.1a and 4.1b} \tag{4.2}$$

where C is a constant and p is chosen to be 2. In a separable case, SVM constructs the hyperplane $\mathfrak{H}$ such that the margin between the positive and the negative class label examples is maximized. Given a test vector $x \in \zeta$, a class label $\{+1, -1\}$ is assigned to x depending on whether $\bar{\Psi} \cdot \Phi(x) + b$ is less than or greater than $(k_0 + k_1)/2$. To distinguish them from the rest of the training examples, support vectors $s_j \in \zeta$ are defined as training samples for which one of the Equations 4.1a 4.1b is an equality. Therefore, the solution $\bar{\Psi} \in \mathfrak{H}$ can be written as:

$$\bar{\Psi} = \sum_{j=1}^{|s|}\alpha_j y_j \Phi(s_j) \tag{4.3}$$

where $|s|$ is the number of support vectors, $y_j \in \{\pm 1\}$ are the class labels of s_j and $\alpha_j \geq 0$ are weights determined during training. Therefore, in order to classify a test point x we must multiply $\bar{\Psi}$, from Equation 4.3, by x. One of the key ideas in SVM is the use of *kernels* to efficiently compute the dot products in a possibly infinite dimension in $\mathfrak{H}$ without having to explicitly compute the mapping Φ.

4.3 Learning Curve for Sample Size Prediction

Correctly estimating the parameters of a learning curve (LC) can help us estimate n or the sample size to achieve the desired accuracy. Recall that the LC is a function that relates the accuracy and the sample size. Vapnik-Chervonenkis (VC) theory [VC71]

states that a random sample of n examples leads to a generalization error $O(\frac{d}{n})$ of a function class F where d is a measure of the complexity of F. VC theory presents universal bounds that are distribution independent. In our case, through empirical support, we are able to provide a tighter empirical estimate of the accuracy error than those given by the VC theory.

To estimate the required sample size, we provide a new framework called SS-TS, which works by iteratively constructing a learning curve (LC), where LC denotes a function of accuracy in terms of the sample size. For a wide range of machine learning (ML) algorithms, including association, the shape of an LC follows a power law [MTH02]. Thus, we can estimate the parameters of the power law curve by computing only k points, where k is typically less than five. Therefore we execute our association finding algorithm on k different sample sizes and estimate the convergence rate which is then used to predict the sample size for the required accuracy.

For a wide range of machine learning algorithms, including classification, the learning curve follows a power law [MTR03] and thus we estimate the parameters of a learning curve by subsampling and extrapolating. Our method parametrizes the learning curve as an inverse function of the power law: $\xi(n) = a + n^{-\alpha}$, where α is the power law index or 'scaling' parameter and a is a constant. The unknown parameters of this expression, $a \in R$ and $\alpha \geq 0$, are estimated using nonlinear least squares, which estimates the parameters via minimization. The estimation of the parameters is done by evaluating $\xi(n_i)$ for various i exactly. Figure 4.1 shows SS-TS's interface for the learning curve prediction. Empirically we observe that the learning curve can be estimated with a useful accuracy for real-world datasets.

4.4 Classifier Error Estimation

This section develops a technique for the error estimation of a classification rule constructed from a training set of data. The training set $s = (s_1, s_2, ..., s_n)$ consists of n observations $s_i = (t_i, y_i)$, with t_i representing the features and y_i the class. Thus, on the basis of x we wish to construct a classifier $c_s(t)$ and wish to estimate its accuracy error. Using the foundation developed in this section the user can get a quick estimate of the accuracy for a given dataset size.

Recall that FACT uses resampling to perform error estimation of classifier accuracy. By re-computing a function of interest many times, a result distribution is derived from which both the approximate answer and the corresponding error are retrieved. FACT uses a delta maintenance strategy, to be discussed in the next section, that dramatically decreases the overhead of computation. As a measurement of the accuracy error, in our experiments, we use a coefficient of variation (c_v) which is a ratio between the standard deviation and the mean. Our approach is independent of the error measure. Other error measures (e.g., bias, variance) are applicable to our approach.

Our classifier error estimation works in three steps: (i) select a sample of the training set (ii) execute classifier validation estimator B times on the training data and (iii) estimate the accuracy and confidence intervals from the resulting accuracy distribution. For step (ii) the classifier validation is performed by training the model on half of the sample and using the other half for testing. Other classifier validation techniques (e.g. CV) are also acceptable. Next we present more details on the above steps.

The traditional way of estimating classifier accuracy uses cross-validation [ET97]. Cross-validation, although a nearly unbiased estimate of the error rate, has high variability. Our proposed classifier error estimator, as shown in the experiments, can substantially reduce the variance of the estimated error. Specifically, in the case of clas-

Table 4.1: FACT can significantly reduce the variation in prediction accuracy compared to the standard cross-validation.

Method	Estimated Accuracy	True Accuracy	Error (c_v)
CV1	0.76	0.85	NA
FACT	0.83	0.85	0.012

sification, our generalized accuracy model can be thought of as a smoothed version of the cross-validation. As shown in Table 4.1 our proposed method produces a closer estimate to the true error together with a reliable accuracy error estimation.

We needed to also find the empirically correct value for B in step (ii) above that would provide a reliable accuracy measure and not introduce too much processing overhead due to resampling. To compute an *exact* bootstrap variance estimate $\binom{2n-1}{n-1}$ resamples are required, which for $n = 15$ is already equal to 77×10^6, therefore an approximation is necessary to make the bootstrap technique feasible. The *Monte-Carlo* [ST95] is the standard approximation technique used for resampling methods including the bootstrap that requires less than n resamples. It works by taking B resamples resulting in variance estimate of $\hat{\sigma}^2_B = \frac{1}{B}\sum_{n=1}^{B}(\hat{\theta}^*_n - \hat{\theta}^*)^2$ where $\hat{\theta}^*$ is the average of $\hat{\theta}^*_n$'s. The theory suggests that B should be set to $\frac{1}{2}\epsilon_0^{-2}$ [Efr87], where ϵ_0 corresponds to the desired error of the Monte Carlo approximation with respect to the the original bootstrap estimator. Experiments, however, over and over again show that a much lower value of B can be used in practical applications. In practice as discussed in chapter 3 30 bootstraps are enough to provide a confident estimate of the error.

Thus, using resampling we can provide a reliable estimate of the classifier accuracy error. The computational efficiency of the approach presented, however, can still be improved by using the techniques from section 3.1.9 as we describe next.

4.5 Incremental Classifier Error Estimation

The most computationally intensive part of *FACT*, aside from the classifier training job j, is the re-execution of j on an increasingly larger sample sizes. One important observation is that this intensive computation can reuse its results from the previous iterations. By utilizing this incremental processing, performing large-scale computation can be dramatically improved. Thus in this section we describe how classifiers from multiple bootstrap samples can be combined.

Given m_1 and m_2 SVM models that were trained on samples s_1 and s_2 respectively, we want to merge m_1 and m_2 such that the resulting SVM m_3 is just as accurate as an SVM model trained on $s_1 \cup s_2$. Because an SVM model consists of a set of support vectors, merging two models implies combining the support vectors of both models together. We combine and retrain the support vectors from two models only if their separating hyperplanes are sufficiently different. By combining the two models we avoid re-computation. Recall that we build the model on half of the sample and test it on the other half. Therefore, when expanding a sample, we in fact have to combine the test data *and* the support vectors. Thus we first discuss how the combining of support vectors is done and then describe how to combine the testing data to ensure that our test examples come from a uniformly random distribution.

4.5.1 Merging Support Vectors from Training Data

Training SVM is costly because it requires solving a quadratic programming (QP) problem in a number of coefficients equal to the number of training examples. Thus, standard numeric techniques for QP become infeasible for very large datasets [CP00]. In our case, SVM models are trained on the training sample data that expands in size. Thus the goal of this subsection is to show how results from different samples can be

combined.

We combine support vectors produced from uniform samples of the training data. This is an important difference from previous approaches [Rup01, CP00], where SVM is trained via a techniqued called 'chunking' where subsets of the training data are optimized iteratively until the global optimal is reached [Rup01]. The key difference here is that each chunk may not be uniformly random. Thus, eliminating non-support vectors from each subset becomes important. Like the above approach, most of the previous work on incremental SVM processing deals with a non-uniformly random subsets of the data [WNC05, Rup01]. This is a problem due to the lack of knowledge about the population distribution from such a sample. Because our training samples are i.i.d. we avoid the issues of incremental processing of previous work.

Due to the i.i.d sample property we can combine support vectors v_1 from s_1 with support vectors v_2 of s_2 directly. In other words, to get a new set of support vectors v_3 from $s_3 \subseteq s_1 \cup s_2$ SVM is re-trained on $v_1 \cup v_2$ which is a much smaller set than the original number of training examples. As an additional improvement we make an optimization where $v_1 \cup v_2$ is retrained only if the distance between the separating hyper-planes of s_1 and s_2 is greater than ϵ.

An intuitive picture of this approach is presented in Figure 4.2. If s_1 is chosen randomly from a training set, its support vectors are very likely to contain support vectors of the whole training data. Therefore, non-support vectors of s_1 have a high chance of being non-support vectors of the whole dataset and thus can be safely excluded.

4.5.2 Merging Test Data

This subsection explains how to ensure that the combined test data represents a uniformly random sample. Let s denote the test data sample of size n used in the i-th iteration, and $\{b_i, 1 \leq i \leq B\}$ denote the B bootstrap resamples drawn from s. The

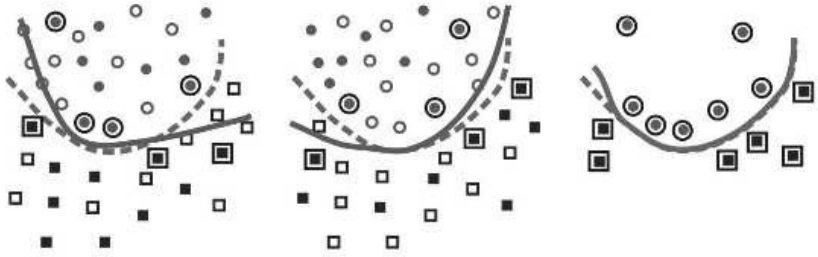


Figure 4.2: Two random subsets are selected from the training data and each is trained individually. The support vectors in each of the subsets are marked with frames. They are merged for the final optimization (right), resulting in a classification boundary (solid curve) close to the one obtained on the entire training data (dashed curve).

classifier testing job j is repeated on all b_i's. In the $(i+1)$-th iteration, we enlarge sample s with another sample Δs. s and Δs are combined to get a new sample s' of size n'. B bootstrapping resamples $\{b'_i, 1 \leq i \leq B\}$ are drawn from s', and the user's classifier testing job j is repeated on all b'_i's. Each resample b'_i can be decomposed into two parts: (1) the set of test data-items randomly sampled from s, denoted by $b'_{i,s}$, and (2) the set of test data-items randomly sampled from Δs, denoted by $b'_{i,\Delta s}$.

Therefore, in the $(i+1)$-th iteration, instead of drawing a completely new $\{b'_i\}$ from s', we can reuse the resamples $\{b_i\}$ generated in the i-th iteration. The idea is to generate $b'_{i,s}$ by updating b_i, and to generate $b'_{i,\Delta s}$ by randomly sampling from Δs. This incremental technique has the benefit of producing a uniformly random sample of the data from multiple subsamples.

The process of generating $b'_{i,s}$ from b_i is not trivial, due to the following observation. Each data item in b'_i is drawn from b_i with probability $\frac{n}{n'}$, and from Δs with probability $1 - \frac{n}{n'}$. We have the following equation modeling the size of $b'_{i,s}$ by a binomial distribution.

$$P(|b'_{i,s}| = k) = \binom{n'}{k} \left(\frac{n}{n'}\right)^k \left(1 - \frac{n}{n'}\right)^{n'-k} \tag{4.4}$$

This means that we may need to randomly delete data-items from b_i, or add data-items randomly drawn from s to b_i. We first present a naive algorithm which maintains a resample b'_i from s' by updating the resample b_i form s in three steps: (1) randomly generate $|b'_{i,s}|$ according to Equation 4.4. (2) if $|b'_{i,s}| < n$, then randomly delete $(n - |b'_{i,s}|)$ data-items from b_i; if $|b'_{i,s}| > n$, then randomly sample $(|b'_{i,s}| - n)$ data-items from s and combine them with b_i. (3) generate $(n' - |b'_{i,s}|)$ random sample from Δs and combine them with b_i.

The above process requires us to record all the data-items of s and b_i, which is a huge amount of data that cannot reside in memory. Therefore, s and b_i must be stored on the HDFS file system. Because this data will be accessed frequently, the disk I/O cost can be a major performance bottleneck.

Next, we present our optimization algorithm with a cache mechanism that supports fast incremental maintenance. Our approach is based on an interesting observation from Equation 4.4. With n' very large and n/n' fixed, which is usually the case in massive MapReduce tasks, Equation 4.4 can be approximated by the Gaussian distribution

$$N\left(n, n\left(1 - \frac{n}{n'}\right)\right) \tag{4.5}$$

For a Gaussian distribution, by the famous *3-sigma rule*, most data concentrate around the mean value, to be specific, within 3 standard deviations of the mean. As an example, for the distribution 4.5 with its standard deviation denoted by $\sigma_0 = \sqrt{n\left(1 - \frac{n}{n'}\right)}$, over 99.7% data lie within the range $(n - 3\sigma_0, n + 3\sigma_0)$; over 99.9999% data lie within the range $(n - 5\sigma_0, n + 5\sigma_0)$. Note that $\sigma_0 < \sqrt{n}$.

Next we explain our optimized algorithm in more detail. For the i-th iteration, we define the delta sample added to the previous sample as Δs_i. For the first iteration, we can treat the initial sample as a delta sample added to an empty set. Therefore we can denote it by Δs_1. The size of Δs_i is n_i. After the i-th iteration, a bootstrapping

resample b can be partitioned into $\{b_{\Delta s_k}, k < i\}$, where $b_{\Delta s_k}$ represents the data-items in b drawn from Δs_k. We build a two-layer memory-disk structure of b. Instead of simply storing b on a hard-disk, we build two pieces of information of it: (i) memory-layer information (a sketch structure) and (ii) disk-layer information (the whole data set). A *sketch* of data set of size n is $c\sqrt{n}$ data items randomly drawn without replacement from it where c is a chosen constant. Determining an appropriate c is a trade-off between memory space and the computation time. A larger c will cost more memory space but will introduce less randomized update latency. The sketch structure contains $\{sketch(b_{\Delta s_k})\}$ and $\{sketch(\Delta s_k)\}$.

During updating, instead of accessing s and b directly, we always access the sketches first. Specifically, for step 2 in our algorithm, if we need to randomly delete data-items from $b_{\Delta s_k}$, we sequentially pick the data-items from $sketch(b_{\Delta s_k})$ for deletion; if we need to add data-items randomly drawn from Δs_k, we sequentially pick the data-items from $sketch(\Delta s_k)$ for addition. For already picked data-items, we mark them as *used*. At the end of each iteration, we will randomly substitute some of the unused data items in $sketch(b_{\Delta s_k})$ with the used data items in $sketch(\Delta s_k)$ by following a reservoir sampling approach, in order to maintain $sketch(b_{\Delta s_k})$ as a random sketch of $b_{\Delta s_k}$. If we use up all the data-items in a sketch, we access the copy stored in HDFS, applying two operations: (1) committing the changes on the sketch, and (2) resampling a new sketch from the data.

4.6 Sample Size Prediction

In this section, a learning-curve based technique for sample size prediction is presented. We also discuss our work towards using recent data complexity theory to identify the best ML algorithm so a minimal sample size can be used.

One of the exciting parts of our work is predicting the learning curve, or the relationship between the size of the training data and the classifier accuracy. Thus, in this section we describe how a learning curve can be estimated naturally using techniques presented so far. The study of learning curves is important to us because learning curves can be used to estimate how large n must be before the probability of a misclassification drops below a user specified level τ. Thus, we are interested in the asymptotic complexity as n becomes large. Vapnik-Chervonenkis (VC) theory [VC71] states that a random sample of n examples leads to a generalization error $O(\frac{d}{n})$ of a function class F where d is a measure of the complexity of F. VC theory presents universal bounds that are distribution independent. In our case, through empirical support, we are able to provide a much tighter empirical estimates of the accuracy error than those given by the VC theory.

The learning curve expresses the error rate of a predictive modeling procedure as a function of the sample size of the training dataset. The learning curve typically follows the power law. By predicting the learning curve one can gauge if more examples will improve the accuracy of a classifier. Empirically we observe that the learning curve can be estimated with a useful accuracy for real-world datasets. Although this is a relatively new research area we show that, for data-sets examined, our prediction has a low error compared to the actual pre-computed learning curves.

Predicting learning curves can be useful in many areas including the interpretation of modeling results as well as dealing with large datasets. For example if training data obtained is of a modest size one might naturally ask if the trained model is a useful predictor given the small training size. If we learn that the expected error can be substantially improved when using a larger sample size, then one would be encouraged to retrieve a larger sample [LS12]. Other, more pressing issues, such as the scalability of a learning algorithm given a growing training size are also critical and can be

addressed by FACT.

The problem we consider is to *estimate* the learning curve rather than to bound it. Thus, we assume that the data from which the estimation can be computed is available, and in evaluating our estimation we are focused on statistical criteria such as the variance, bias etc. Thus to achieve this estimation, we must capture the general form of the function as well as the constants and lower order terms.

The learning curve follows a power-law [MTR03] and thus we estimate the parameters of a learning curve by subsampling and extrapolating. Our method parametrizes the learning curve as an inverse function of the power law: $\tau(n) = a + m^{-\alpha}$. The unknown parameters of this expression, $a \in R, b, \alpha \geq 0$, are estimated using nonlinear least squares, which estimates the parameters via minimization. The estimation of the parameters is done by evaluating $\tau(n_i)$ for various i exactly. The estimation of the learning curve, using our optimization techniques, gives the user the ultimate power of making many data-gathering/model training decisions quickly.

4.7 Learning Curve Experiments

4.7.1 Methodology

We chose datasets from the UC Irvine repository [A 07] that contained more than 500 instances. We found 25 such datasets. While we felt that 500 instances were enough for the initial testing, larger datasets are needed to draw any meaningful conclusions. Thus, we also used a private Twitter dataset that is many TB in size. We also validated our theoretical claims using a synthetic dataset, which is a no information random dataset with 20 boolean features and a boolean random label. While the true accuracy cannot be determined because the target concept is not known, we can approximate the true accuracy using the *holdout method*. Using the holdout method, a sample of a

Table 4.2: Datasets used

Name	# Features	Examples	True Accuracy	% data needed to achieve 90% accuracy (relative)
Adult	14	48,842	86%	2%
Breast Cancer (W)	32	569	93.2%	5%
Vehicle	18	946	61%	3%
Mushroom	22	8,124	96.5%	3%
Rand	20	100,000	49%	$<$1%
Twitter	1,000	3B	85%	$<$1%

given size is taken and a model is trained on that sample. The rest of the data is used to compute the accuracy. This process is repeated 30 times. Table 4.2 summarizes the datasets used in our experiments. Table 4.2 also provides the training size actually needed to achieve a 90% relative accuracy. By relative accuracy we mean the accuracy obtained relative to a model trained on the entire dataset. In our experiments, for comparison, we use a 10-fold CV method.

4.7.2 Classifier Accuracy Estimation Experiments

We now show the experimental results for our proposed classifier accuracy estimation using resampling. We begin with the discussion of the bias for our proposed estimation method and the standard cross-validation method. We conclude with the discussion on variance of these techniques.

The bias of the method that estimates a parameter θ is defined as the expected value minus the estimated value. An unbiased method is that which has a zero bias. Figure

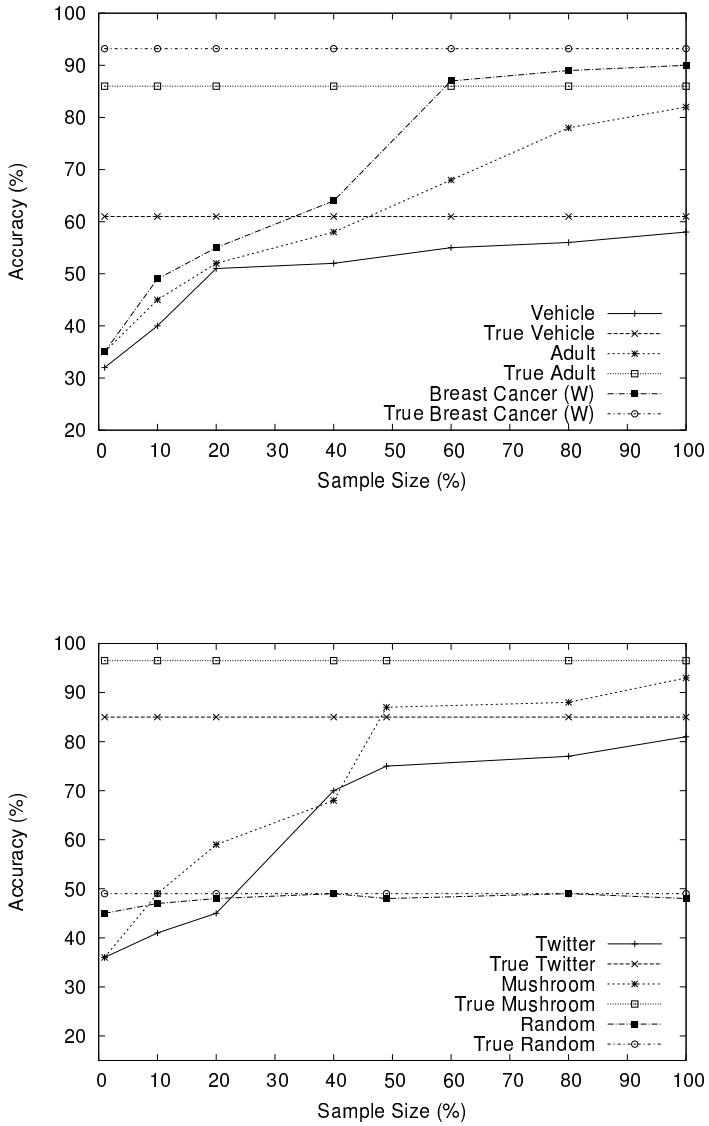


Figure 4.3: The bias of cross-validation with varying sample sizes.

4.3 shows that the 10-fold cross validation is pessimistically biased especially for low sample sizes. Figure 4.4 on the other hand shows that our approach is much less biased at lower sample sizes.

While a given method may have low bias, it can be at the expense of high variance.

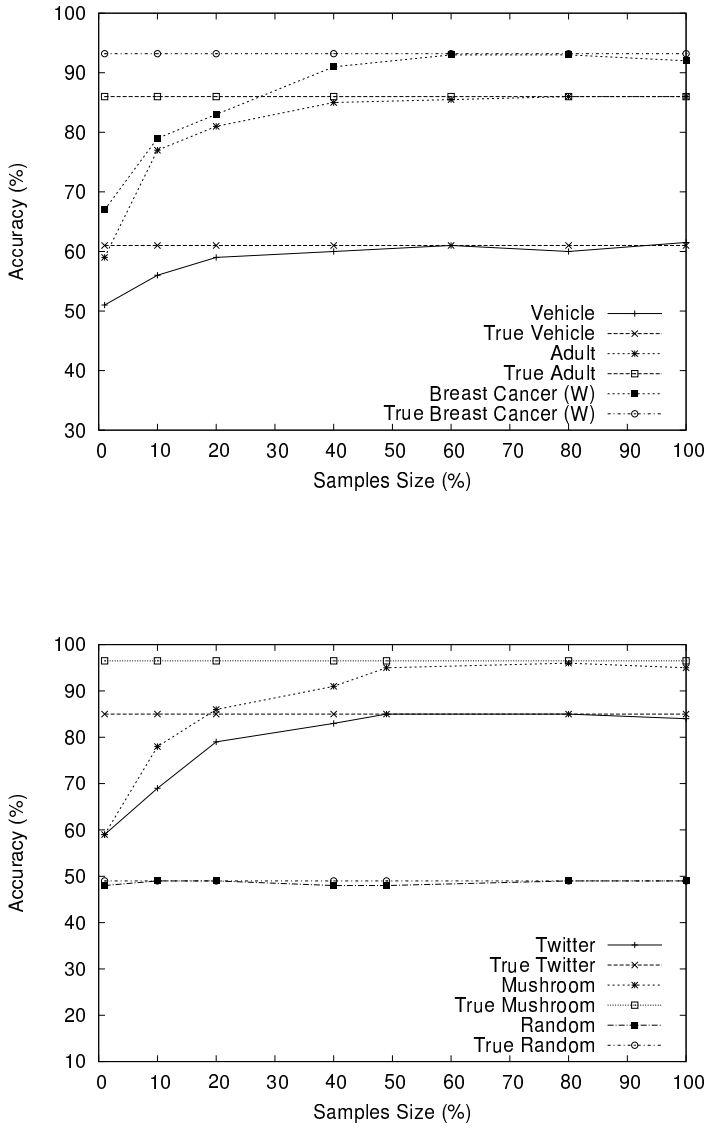


Figure 4.4: The bias of FACT with varying sample sizes.

Thus in this experiment we have also computed the coefficient of variation. Figure 4.6 shows the c_v for cross-validation and Figure 4.5 shows that for our approach. Cross-validation has higher c_v than that of our approach.

Thus, FACT performs better in terms of the c_v and the bias, which proves our

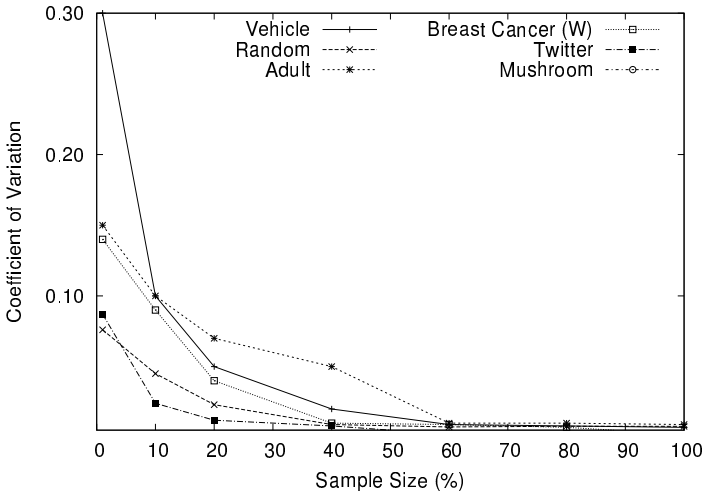


Figure 4.5: The c_v of accuracy of FACT.

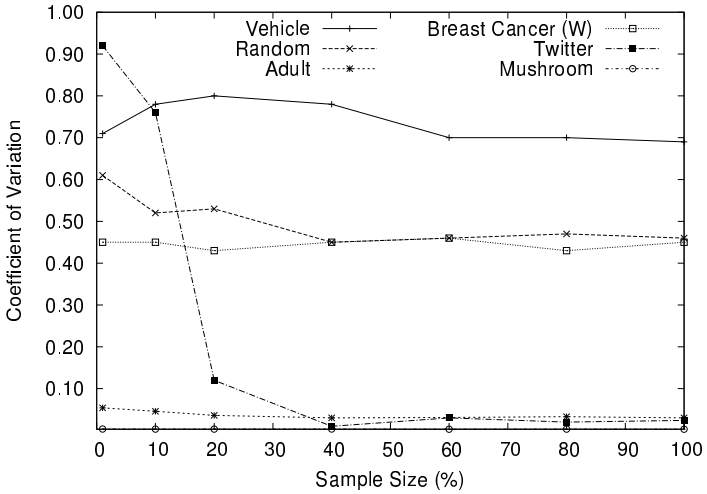


Figure 4.6: The c_v of cross-validation

hypothesis that FACT is a *smoother* accuracy estimator compared to the standard cross validation.

Table 4.3: Classification using the merge optimization.

Dataset	Accuracy w/o merge	Accuracy w/ merge	Training Time w/o merge	Training Time w/ merge
Adult	85.21	85.50	$\approx$ 11.23min	$\approx$ 1.22min
Breast Cancer (W)	93.2	93.23	$\approx$ 32sec	$\approx$ 3.43sec
Vehicle	62	62.21	$\approx$ 34.2sec	$\approx$ 3.67sec
Mushroom	96.3	96.35	$\approx$ 2.2min	$\approx$ 12.7sec
Rand	49.3	49.23	$\approx$ 23min	$\approx$ 2.1min
Twitter	85.2	85.7	$\approx$ 20hrs	$\approx$ 1.7hr

4.7.3 SVM Merge Optimization Experiments

To check the validity of our merging algorithm we compared it against the results produced by a 10-fold cross-validation on the entire dataset. Merging was done via partitioning the entire dataset into 10 parts and performing merging on each part as described in Section 4.5. Table 4.3 shows that our merging algorithm, while performing significantly faster, does not introduce major accuracy degradation. Recall that while merging was introduced specifically for SVM in this paper, theoretical work was presented that can be used to extend FACT to other classifiers.

4.7.4 Learning Curve Prediction

In this experiment we use the initial training set of $n = 50$ to estimate the $\sigma(n)$ for $n = 75, 100, 150$ and 200. Therefore our task of extrapolating beyond the original sample is ambitious. Figure 4.7 shows a learning curve prediction for various datasets

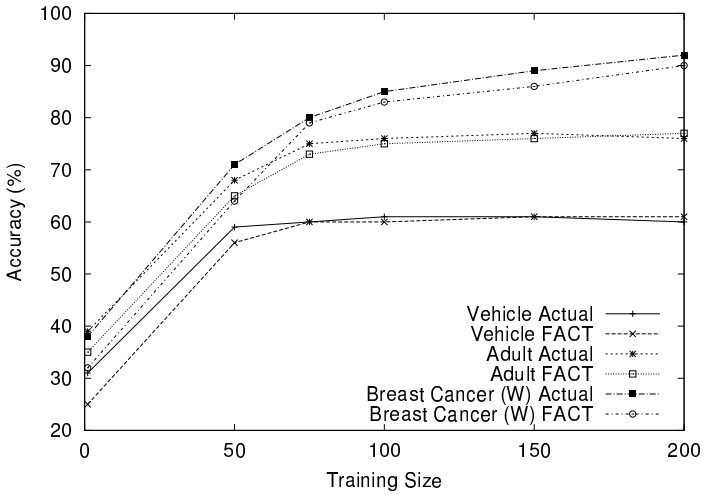


Figure 4.7: Learning curves predicted by FACT, for various datasets, are similar to actual learning curves.

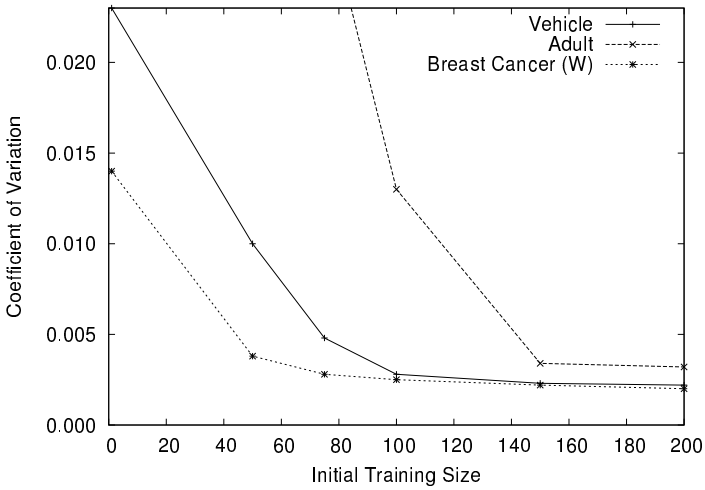


Figure 4.8: We can extrapolate the learning curve quite far without incurring too much error, and using only a small dataset.

given a fixed initial training size of $n = 50$.

Figure 4.8 shows the learning curve prediction error for various initial training set sizes. This experiment aimed to accomplish two objectives (i) show that our estimated predicted curve is close to the actual learning curve for various datasets and (ii) show that the extrapolated the learning curve does not incur too much error when only a small initial training sample is used. As can be observed, we successfully accomplish both of these objectives. Note however that our LC approach is not perfect, and there are still some discrepancies in Figure 4.7.

4.8 Future work for Extension to Other Classifiers

Many machine learning algorithms use gradient descent or exception maximization, for which online and stochastic versions exist, e.g. stochastic gradient descent and online exception maximization [CM07, BB07]. These online and stochastic algorithms allow training points to be considered in a streaming manner. That is, when the learning model is trained on new data, we do not need to re-consider the old training data we have already seen. Using this observation, it is promising to develop light weight delta maintenance techniques for large range of machine learning algorithms using incremental bootstrap computation. However, this would be our future work, and is not discussed in this dissertation.

Furthermore, recent bootstrap variants, such as the Bag of Little Bootstraps [KTS12a], are developed in order to take advantage of distributed computing setting. These methods can be plugged into our framework to achieve faster bootstrap computation and less network overhead.

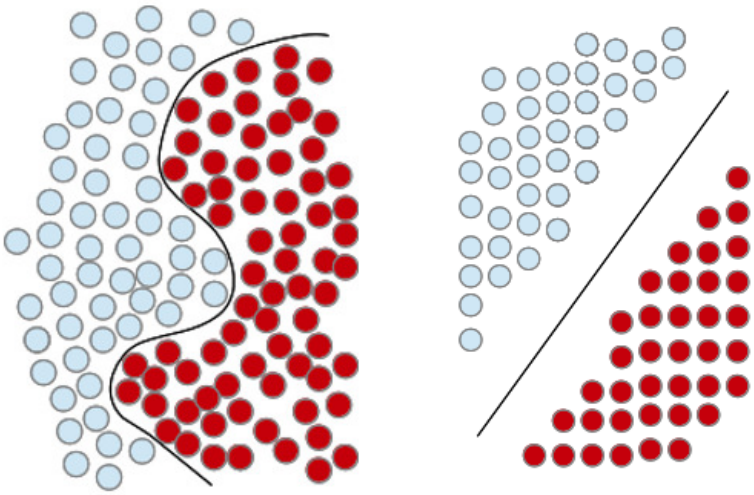
4.9 Conclusion

In this chapter we presented *FACT*, a non-parametric classifier approximation system that relies on techniques presented in chapter 3. We showed how FACT can be applied to the SVM classifier for quick model training. We found that our approach provides an accurate estimation method for SVM. We also presented an initial insight for extending our approach to other classifiers. Using the resampling techniques from chapter 3 we found that it is rarely necessary to use more than 1% of the training data to achieve identical accuracy levels compared to the classifier trained on the entire dataset. In our internal tasks we found that the learning curve estimation was quite helpful in classifying large datasets. In general the learning curve proved effective in planning the classification task by determining with confidence the point after which no noticeable improvement is seen. We plan to extend the presented approximation approach to other classification algorithms and other data mining tasks, which suggests a future direction of our research.

CHAPTER 5

Data Complexity

Up to this point in the dissertation we dealt with quality assessment and sample size selection while keeping the underlying machine learning algorithm fixed. In this chapter we study the relationship between the three factors: (i) sample size (ii) data complexity and (iii) algorithm complexity. The goal is to determine the required sample size given the algorithm complexity and data complexity. The study of data complexity is a complicated issue, and we do not expect to obtain immediate success. For this reason, we restrict our analysis to very simple data models and two types of algorithms: (i) association finding and (ii) pattern matching.



(a) A complex dataset. (b) A simple dataset.

Figure 5.1: Different types of problem complexity in the context of classification.

5.1 About Data Complexity

Data complexity theory can be used to complement LC parameter estimation. This approach selects the most competent algorithm for a given problem complexity to minimize the required sample size. The data complexity part of this dissertation is very novel, and we present its results only as independent experiments in sections 5.2.2 and 5.3.3. We believe, and confirm through experiments, that data complexity represents a strong potential for future work in the area of ML algorithm selection. Our belief is also justified by the recent notable works in the area [HB02, Ho08, Ho04, KB06].

The intuition behind data complexity is best presented from the classification point of view. Consider Figure 5.1 which shows the different levels of classification complexity. The complexity of the data for the classification problem is often measured in terms of the percentage of points on boundary (estimated by the minimum spanning tree) and linear separability of the classes [KB06]. Based on these metrics, Figure 5.1a gives an example of a complex problem due to a nonlinearly separable classes and Figure 5.1b shows an example of a less complex problem. Authors in [KB06] identify classifiers which are most adept at dealing with problems of various complexities [1].

5.2 Data Complexity and Association Detection

In this section we show how the recent data complexity theory can be used to provide sample size estimation given the relationship complexity and algorithm complexity. The data complexity results presented can be seamlessly integrated with the quality estimation, BLB-TS, and with the sample size estimation, SS-TS, approaches presented in Chapters 3 and 4 respectively. The findings in this section can be used as an

[1] In [KB06] authors find that nearest neighbor (nn) and linear classifier (lc) have opposite domains of competence. Surprisingly (lc) performs best for complex problems and (nn) performs best for easy problems.

initial stage for algorithm selection instead of the often-used trial-and-error strategy.

5.2.1 Introduction to Data Complexity for Association Discovery

Here we present techniques for leveraging data complexity[2] theory to pick the appropriate association finding algorithm given the problem complexity. Recently it has been shown that the performance of a classifier can be analyzed in terms of the data complexity [HB02, Ho08, Ho04]. We apply this idea to the association complexity to explain how relationship complexity affects the required sample size needed to capture this relationship given a fixed association mining algorithm. Our experimental results provide the motivation for further exploring this idea.

Definition of Association Complexity: In this work, we study how the complexity of data dependence affects the sample size required when we hold constant both the association finding algorithm and the user prescribed accuracy. Section 6.1 further describes the association finding method used and below we only present the required pieces of the algorithm to understand complexity computation. To compute the complexity of a relationship between two variables x and y we use the approach proposed by [RRF11] which describes complexity as:

$$MCN(D, \epsilon) = \max_{xy<B} \{log(xy) : M(D)_{x,y} \geq (1 - \epsilon)MIC(D)\}$$

where MIC is defined as

$$MIC(D)_{x,y} = \frac{I^*(D, x, y)}{logmin\{x, y\}} \tag{5.1}$$

and where D is a finite set of ordered pairs, and x, y correspond to an x-by-y grid into which x and y values of D can be partitioned into. I^* corresponds to the maximum

[2]Here data complexity refers to the complexity of the data distribution and is different from the complexity of evaluating a query on a database instance which is expressed as a function of the size of the instance when the query is fixed.

mutual information of D when it is partitioned into a specific x-by-y grid G. Note that $0 \leq I^*(D,x,y) \leq logmin\{x,y\}$ therefore dividing by $logmin\{x,y\}$ bounds $M(D)_{x,y}$ to a $[0,1]$ range. The MCN statistic measures the complexity of the relationship in terms of the number of cells required to reach the MIC score. The MIC score is the maximum MI score over all possible x-by-y grids that aim at 'encapsulating' the inter-variable relationship (see Section 2.2). Using the MCN metric we can determine that a function like $f(x) = x$ represents a simple relationship because very few cells are required (e.g., four) to encapsulate the dependency whereas a function like $f(x) = sin(18\pi(x))$ requires many (thirty-six) cells. Notice that $MCN(D,\epsilon)$ depends on $MIC(D)$, but because $MIC(D)$ roughly equals R^2 for most practical purposes it is a good approximation to the actual inter-variable dependence.

5.2.2 Experiments

Experimental Setup: We use a real-world dataset of size 4.3TB from Twitter that we have privately collected. This dataset consists of roughly 2 Million hashtags and their frequency in a time-series format. We use hourly granularity and at a maximum, each hashtag has 4,000 observations[3]. Empirical evidence suggests that computing an MIC-association matrix for the above dataset is very expensive and without approximation can take months to compute. We use a private Hadoop cluster of size 7 where each node is a 16-core AMD Opteron(tm) Processor 6134 @ 2.3GhZ with 132GB RAM. We use Hadoop 1.0.4 with a total of 140 map slots.

Experimental Results: We vary the data complexity and a machine learning algorithm and observe how the sample size is impacted. Table 5.1 presents the ML algorithms used. The association finding algorithms presented in Table 5.1 are ranked based on their ability to find nonlinear associations as measured from the set of func-

[3]Not every hashtag contains the same number of observations

Relationship Type	Description	Required Sample Size
Linear	$y = x$	20
Parabolic	$y = 4(x - \frac{1}{2})^2$	20
Cubic	$y = 128(x - \frac{1}{3})^3 - 48(x - \frac{1}{3})^2 - 12(x - \frac{1}{3}) + 2$	20
Exponential	$y = 10^{10x} - 1$	20
Linear/Periodic	$y = sin(10\pi x) + x$	20
Sinusoidal (Fourier Frequency)	$y = sin(16\pi x)$	50
Sinusoidal (non-Fourier Frequency)	$y = sin(13\pi x)$	160
Sinusoidal (Varying Frequency)	$y = sin(7\pi x(1 + x))$	310
Categorical	64 points chosen from the following set: {(1,0.297), (2, 0.796), (3,0.290),(4,0.924), (5,0.717)}	580
Random	random number generator	660

(a) Different levels of data complexity and the required sample size needed to capture the relationship using MIC.

Figure 5.2: Problem complexity and required sample size.

Correlation Method	Type
Pearson	Linear
Spearman	Linear
MI (KDE)	Nonlinear
MI (Kroskov)	Nonlinear
CovGC	Nonlinear
Maximal Correlation	Nonlinear
MIC	Nonlinear

Table 5.1: Types of machine-learning algorithms used.

tional Relationship Types shown in Table 5.2a. Thus we compare the score produced by algorithm A with the score given by R^2 and rank A accordingly. Figure 5.3 shows the results indicating that to achieve a minimal sample size, one should apply the simplest association finding algorithm to find the simplest complexity and one should apply a more sophisticated algorithm to a problem with more complexity. Note that there are several spikes which are due to Spearman and Pearson correlation being more efficient for monotonic and linear relationships respectively. Table 5.2a further elucidates the relationship between the problem complexity and the required sample size when using MIC. We observe that as relationship complexity increases, the required sample size also increases. Note that the required sample size in Table 5.2a is measured in the number of tuples, ignoring the total size of the dataset. The types of relationships we examined only include those that can be represented by a function and the study of other relationship types are part of our future work.

Next, application of data complexity to pattern matching is presented.

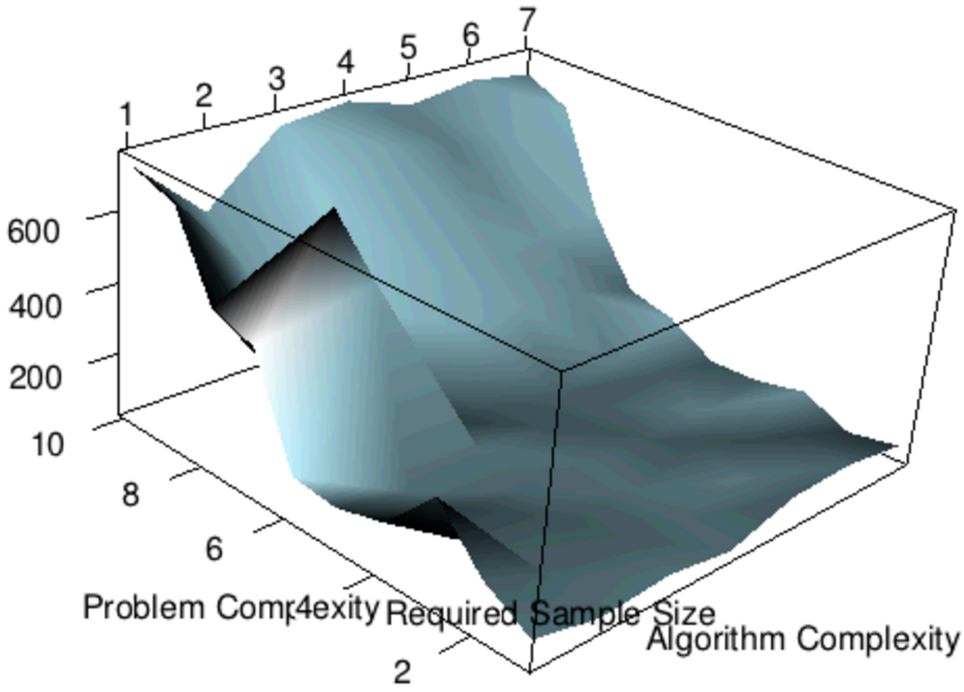


Figure 5.3: Data complexity results.

5.3 Data Complexity and Pattern Matching

The previous section introduced data complexity for association finding algorithm selection. In this chapter we extend this idea to pattern matching. In particular we dynamically select the type of automaton loaded in memory based on the available system resources. Thus, we define automaton complexity as the amount of nondeterminism present in the automaton and we maximize throughput while adhering to the available system resources. We term the above problem as an optimization problem and solve it using linear programming techniques.

5.3.1 Introduction to Pattern Matching

In this section, we improve the performance of pattern matching by changing the pattern representation dynamically according to available resources. Previously, pattern matching was done using a static automaton, however because ‘one size does not fit

all', we propose a dynamic model that adopts to its resource environment thus dramatically improving the performance of pattern queries. Indeed, the optimization of pattern queries represents a problem of great research interest and practical importance due the many application of such queries.

In the financial sector, pattern queries (PQs), that run continuously over the data streams, are used to detect fleeting opportunities by monitoring trends. Informally we define a pattern query as a regular expression (RE) with optional predicates assigned to each symbol. Note that if a pattern query q only contains equality predicates, then q is exactly equivalent to a regular expression [PC03]. In network management continuous pattern queries can be utilized to monitor online traffic and detect anomalies (e.g., link congestion) and their cause (e.g., hardware failure, denial-of-service attack). Because a set of patterns is usually large (a set of HTTP patterns alone takes a few GB [Roe99]) and because we look at applications where processing of data needs to be done in realtime, dynamically adjusting the pattern matching system to take advantage of the entire system resources without sacrificing the overall performance is of great importance. These are just two of the many application examples where pattern matching plays an important role. The importance of pattern matching is underscored by the large series of increasingly powerful and sophisticated query languages proposed for pattern matching. A very incomplete list includes SQL-TS supporting regular expressions and backtrack optimization [SZZ01], SASE+ [DIG07] supporting powerful Kleene-closure queries, and K*SQL [MZZ10] supporting the nested word model and queries on XML software logs, and genomics.

In these pattern languages, regular Kleene-star expressions are used to characterize patterns which are then implemented using an FSA. The type of FSA constructed and thus the resulting query performance much depend on the available system resources. Thus in this section, we address the problem of dynamically changing the underly-

Figure 5.4: Bounded FSA example for a regular expression

ing automata in response to changing system resources, in order to optimize pattern matching performance.

High level of performance is, for instance, required by long running pattern queries that perform decision support and data mining against massive databases or bursty data streams that put a strain on the already limited resources such as processing power, network bandwidth and battery capacitance. The problem is exacerbated when pattern query evaluation is performed using a multi-core or a cloud environment where frequent changes in the availability of workstation resources and the overall workstation load are common. Thus in this work we make the first steps towards designing a dynamic pattern matching system, called *Morpheus* that alleviates the above issues. In fact, Morpheus automatically reconfigures its pattern representation, based on the finite state automata, according to the available workstation resources.

Further evidence for the need of dynamic systems is provided by the introduction of Adaptive Query Processing (AQP)[IDR07]. With AQP run-time statistics are recorded and optimizations on the query, including selection and projection optimizations, are performed [FHA10] based on the current run-time statistics. AQP is especially popular in data streams processing where the data and system resources change frequently. Our work leaves the query unchanged and instead modifies the *system*, which in our case is the pattern automaton, for efficient pattern matching execution. It is common for a multicore system to experience a mixture of predictable and sporadic changes in the workload thereby creating a necessity for a dynamic system optimization. To address these conditions, "resource-aware" systems are beginning to emerge [LWZ01, NSN97] that provide a varying quality of service depending on the available resources. The closest work to ours relies on convoluted tactics to cache frequently accessed program instructions [LCH00] to help with the dynamic workload. These approaches, however,

are not effective when workloads are unpredictable. Other previous works mainly focused on tuning the query representation given a static environment. For example authors in [DF03] employ a non-deterministic finite state automaton (NFA) to represent a set of XPath patterns to filter XML documents. Other methods [KCT07] explore the trade-off between a deterministic finite automaton (DFA) pattern representation and an NFA to find the middle ground that achieves desired performance while minimizing memory usage. These works, however, do not consider changing workloads. This is a serious limitation given the rapid, bursty nature of data streams, and the real-time response requirement of many applications.

Dynamic reconfiguration methods present several problems in terms of reconfiguration overhead (time and space). Because reconfiguration is done during run-time, the overhead is part of the run-time cost and needs to be minimized. The dynamic reconfiguration problem is further complicated by the search requirement that must select the best system configuration. This is often difficult because resource and performance estimations of a potential configuration are required. One way to address the issues with dynamic reconfiguration is by using a naive method. The obvious, but deficient solution, is to let the user reconfigure manually the representation of pattern queries. Unfortunately, the user may know *what* she wants but not know *how* to achieve it. The assumption that the user understands the low level system details including the available resources and is able to adapt a query representation to environment changes is clearly unreasonable. Thus in our approach the above objectives are instead achieved through fast automatic approximation methods that are very effective in practice and significantly improve the state of the art in dynamic pattern matching.

For completeness we briefly define a pattern matching problem. Informally a pattern matching problem seeks to detect all sub-sequences of input that match a given set of patterns. More formally, given a potentially infinite stream I consisting of sym-

bols drawn from the alphabet Σ and given $P = \{\rho_1, ..., \rho_m\}$ which is a set of patterns over Σ then the pattern matching problem involves detecting all subsequences of I matching $\rho_i \in P$.

Traditional pattern matching approaches generally construct one finite state machine (a deterministic or a nondeterministic finite automaton (DFA or NFA respectively)) for all patterns to be matched. A DFA is a quintuple $M(S, s_0, \sigma, \Sigma, F)$ where S is a finite set of states, s_0 is an initial state, $\sigma : S \times \Sigma \rightarrow S$ is a transition function, Σ is the alphabet and $F \subseteq S$ is a set of accepting states. NFA is defined similarly except that the transition function σ may return multiple states, thus multiple states may be active for input i. As compared to a DFA, an NFA can represent a pattern using storage that is on the same order as the number of characters present in the pattern, however the processing cost of the NFA is expensive due to a potentially large number of concurrently active states. While DFA are largely immune from the processing cost issues plaguing NFA, representing a patterns with a DFA may lead to a state explosion and thus to a prohibitively large automata. Therefore, the number of concurrently active states is the primary criterion that determines both the processing bandwidth and the space requirements of the automaton. Thus, by using a middle-ground between an NFA and a DFA to represent a set of patterns in a dynamic environment we could potentially achieve the speed of a DFA and the space efficiency of NFA during run-time.

In order to dynamically adjust the automaton configuration our system prototype needs to be able to efficiently switch between a non-deterministic finite automaton (NFA), a determinism finite automaton (DFA) and an automaton that has k active states. There are standard techniques to convert a pattern (RE) into an NFA or a DFA [HMU06]. However no published research has so far considered the important problem of dynamically adjusting an automaton with a bounded number of concurrently

active states. Thus we define a *K-Finite Automaton* (K-FA) to be a finite automaton where there is at most k concurrently active states at any time.

Figure 5.4 shows a motivating example of using an automaton with a bounded number of concurrently active states. Figure 5.4 (a) shows an NFA that accepts pattern p and has up to three concurrently active states. Figure 5.4 (b) shows a DFA that accepts p and has only one active state at any time. Figure 5.4 (c) presents a *K-FA* that accepts p where there is only a bounded number of $k = 2$ concurrently active states. Now, if $D(i, j)$ is the difference in the number of states between automaton i and j, we have that $D((b), (c)) < D((a), (c))$. Therefore if resources change such that the automaton in Figure 5.4 (c) was the desired configuration, template in (b) would be morphed into an automaton in (c) as this morphing would require least amount of operations. This type of analysis will be used when deciding how dynamic reconfiguration should be done.

Morpheus uses dynamic system reconfiguration to reconfigure the underlying pattern matching automaton based on the current system resource availability to improve pattern matching performance. Morpheus pre-computes *critical* query configurations that will be used as *templates* which will be morphed into the desired configurations during run time. By using templates the overhead of dynamic reconfiguration is significantly reduced as shown in the experimental section. The *critical* templates to precompute are selected such that the average expected reconfiguration time is minimized. Empirical evidence also shows that the templates virtually do not increase the amortized memory cost while greatly increasing the throughput when compared to static solutions. Throughout this section we refer to a *configuration* as a representation of an automaton m at time t given resources R_{it} where $i \in$ *Resources (R)*.

Anomaly Detection With the help of pattern matching and statistical methods we also will employ our system to detect anomalies, Anomaly Detection, which is defined

as detecting events that deviate from a normal behavior, has recently been attracting a lot of attention. The administrator of a server may use anomaly detection to monitor real-time system statistics and to raise alerts if an abnormal usage pattern is detected. Anomaly detection can also be used for detecting suspicious behavior in a network traffic. Most research on anomaly detection, however, is application-specific and works based on predefined rules. Furthermore, current anomaly detection algorithms are not scalable and cannot adapt to a changing behavior in the input. Our project takes advantage of Stream Mill, a Data Stream Management System designed at UCLA, to construct a light-weight statistical model of the stream that can be used to make predictions of a parameter of interest based on which anomalous behavior can be detected. The model we use is derived from a classical auto regressive model, which is a one-dimension time-series model. Since a large portion of data-streams corresponding to the applications (e.g., real-time system statistics, web-traffic) of interest are one-dimensional time-series, our approach is attractive. We conduct experiments that demonstrate the scalability and efficiency of our approach.

Morpheus performs efficient dynamic reconfiguration in three stages: (i) template computation, (ii) run-time optimization and (iii) template adjustment. In (i) the templates are computed such that the expected dynamic reconfiguration (morphing) time is minimized. The optimal number of templates is determined based on the estimated amortized memory cost and the template update cost. In (ii) resource estimation is performed when searching for a potential configuration. Furthermore the run-time stage identifies the time when it is feasible and desirable to perform the automaton reconfiguration by analyzing the cost/benefit of the reconfiguration. Steps necessary to perform the reconfiguration are also carried out in (ii). Stage (iii) updates the precomputed templates based on the template usage statistics. All three of the dynamic reconfiguration stages are seamlessly integrated into Morpheus.

Figure 5.5: Architecture of Morpheus

This section makes the following contributions:

1. A system prototype, termed Morpheus, is presented that uses a novel way of performing dynamic reconfiguration of pattern automata via morphing of precomputed templates to speed up pattern matching in a dynamic environment.

2. Linear optimization techniques are discussed that are used for dynamic reconfiguration of pattern automaton that increase/decrease automaton parallelism thus adjusting to the current system resources.

3. An Empirical study that evaluates Morpheus over various workloads is presented.

The rest of the section is organized as follows: in section 5.3.2 we present an overview of the architecture of Morpheus. Section 5.3.2.1 is devoted to discussion of the details of each of the components of our system. Section 4.7 presents a performance evaluation of our system.

5.3.2 Pattern Matching System Overview

The main components of Morpheus are briefly described next. The focus of Morpheus is on improving the performance of pattern matching and the main way this is accomplished is by reconfiguring the pattern representation based on the available system resources. Dynamic reconfiguration can negatively impact matching performance, however templates help mitigate the problem by precomputing several configurations in advance that can be used as starting points for achieving the final configuration. Precomputed templates are nondeterministic and have a small memory foot-print as shown by empirical evidence in Figure 5.3.3.4.

Morpheus as an input takes an *NFA* representing a collection of patterns. The inputted NFA is then used by the reconfiguration engine to precompute templates. The key components presented in Figure 5.5 can be summarized as follows:

1. *Reconfiguration Engine:* The reconfiguration engine is responsible for precomputing and adjusting the automata on the basis of the current workload. Specifically, Morpheus uses linear optimization techniques to find (i) the optimal number of precomputed configurations, (ii) the optimal set of operations needed to transition between configurations and (iii) the optimal time when to update the precomputed configurations. The automaton access probabilities are also taken into account during reconfiguration. Furthermore the reconfiguration engine evolves over time because both the resources and the access probabilities change over time.

2. *Resource Monitor:* Morpheus includes an automated statistics collection tool that, with a negligible overhead, captures data from the OS (CPU and RAM usage) and from the currently active automaton (state access probabilities). The CPU usage reported by the Linux kernel is expressed as a percentage of one CPU core. Initially we assume a uniform random distribution of the input symbols (i.e., an equal access probability of all states). During run-time, however, we consider a trace driven probability distribution of various input symbols. With these traces we can more accurately determine the nondeterminism of a particular configuration. The collected statistics are utilized to decide *when* to reconfigure and into *what* reconfiguration.

3. *Resource Usage Estimator:* When choosing an automaton M from the search space we must estimate its resource usage to insure that the available resources are properly utilized. Based on our experiments the average CPU load is linearly proportional to the average non-determinism of approximately the first 5% of

the states of the automaton. Therefore doubling non-determinism of the first 5% of the automaton states doubles the CPU usage requirement. The intuitive explanation for the 5% figure is that on average only the first 5% of the pattern is matched by the input. This crude estimate produces good results as discussed in the performance evaluation section. The memory usage of M is again estimated by a linear approximation and as seen in Figure 4.7 this produces good results.

A brief overview of the resource monitor and the resource usage estimator will be given in Section 5.3.2.5; next we describe the details of the reconfiguration engine.

5.3.2.1 Reconfiguration Engine Details

The reconfiguration engine can be decomposed into three stages: *pre-configuration*, *execution* and *update*. In the first phase several automaton configurations are pre-computed to minimize the dynamic re-configuration time when system resources or workload varies. Then in the execution phase the pre-computed configurations are used as *templates* to *morph* the pre-configuration into a desired state. Finally during the last phase the pre-computed configurations are updated based on the historical statistics gathered. The above phases are discussed in detail next.

5.3.2.2 Pre-Configuration

When pre-computing templates, the assumption is made that all operations are done on a single FSA, (i.e. no clustering is performed). Breaking up the FSA into disjoint clusters may improve memory usage however for the sake of simplicity we only deal with a single automaton. Nevertheless clustering is important especially when dynamically adding patterns into the automaton. While Morphesus also supports simple pattern clustering method using distance metrics such as MDL [CGR03], these are orthogonal

to the focus of this chapter. Thus we will assume that a single NFA **N** representing a set of patterns is given as input. Given N and a set of resources R representing system constraints (e.g. memory, cpu) the pre-configuration problem consists of computing a set of automata *derived* from N such that the average *conversion* time between any two automata is minimized and all resource constraints are satisfied. Therefore, we will derive from N a new automaton M that accepts the same language but has a different level of non-determinism, k, where k denotes the average number of states that are concurrently active per input. A DFA has $k = 1$ and an NFA has no upper bound on k. Thus we define *K-FA* (K-Finite Automaton) as automaton that has a bounded number of concurrently active states equal to k.

$$
\begin{aligned}
&minimize: \\
&\qquad max \sum d_i \\
&\qquad s.t. \sum m_{ij} x_i \leq R_j \\
&\qquad x_i \in \{0, 1\}
\end{aligned}
\tag{5.2}
$$

More formally, the pre-configuration problem can be formulated as shown in Equation 5.2. Variable d_i is defined as the minimum *distance* to automaton i from any other automaton and overall we want to minimize the maximum *distance* to any automaton. The *distance* refers to the difference in the number of states between the original and the derived automaton. Variable d_i is directly proportional to the number of operations (*merge* or *split*) needed to be performed to derive automaton i. The formalization of these operations is discussed in Section 5.3.2.3. The estimation of the size of various automata is presented in Section 5.3.2.2. m_{ij} refers to the amount of resource j that automaton i uses, where i varies from k to 1 and where k is initially set to be the average number of the concurrently active states in N.

The constraints in Equation 5.2 guarantee the feasibility of the solution. Because each $x_{i,j}$ can only vary between 0 and 1, and because the sum of all $m_{i,j}$ for a given i

must be less than or equal to R_j, we guarantee that the combined load imposed on the system will not exceed what is available; this avoids saturation and overcommitment of the system resources. The *resource constraints* include the CPU and the RAM because these were the most constrained resources in the real-world scenarios we explored. Extending a set of constraints, however, to include other resources (e.g., network, disk space) is straight forward. Furthermore note that the goal function in Equation 5.2 also depends on the time t as discussed in Section 5.3.2.3. The feasible solution imposed by the constraints may be fractional and must be rounded as discussed in Section 5.3.2.5.

The number of pre-computed templates also affects the overall dynamic reconfiguration performance. Recall that templates are used as a way to speed up reconfiguration by picking a starting point to reconfigure from that will result in the desired configuration in the least amount of time. Figure 5.3.3.4 shows the number of precomputed template configurations, n, should be such that the benefit (average time to perform dynamic reconfiguration) is equal to the cost (the sum of the average search and the update times of the templates). Automatically determining n is important in decreasing the overhead of Morpheus.

Estimating the number of operations: In Equation 5.2, the number of operations (d_i) is measured in terms of the difference in the number of states between two automata. To estimate the number states (size) of the *K-FA* for some k we use linear fitting between the size of an NFA N and the size of the corresponding DFA D. Thus given a desired k we can quickly estimate the size of the *K-FA* without explicitly deriving it. The number of states of N can be accurately computed, because N is given as the input. The number of states in D can be estimated based on the average nondeterminism per state of the automaton and on the underlying pattern characteristics. The summary of the memory and processing complexities of different types of automata, including *K-FA* is presented in Tables 5.3 and 5.2 [YCD06] where n is the length of

the regular expression and m is the number of regular expressions. Given the initial pattern set we can determine the class of N from Table 5.2, based on which the size of D is computed. To estimate the sizes of subsequent automata a linear model is constructed given two points $|D|$ and $|N|$. When the exact sizes of different $K - FA$ automaton are computed subsequently, Lagrange polynomial is used to adjust our size estimation model into a more accurate, polynomial, version.

Pattern Type	**Example**	**# of states**
Strings of length *k*	^abcd	$k+1$
Wildcards	ab.*cd	$k+1$
Patterns with ^, a wildcard of length j	^ab.{j+}cd	$O(kj)$
Patterns with ^, overlapping with prefix of length j	^a+[a-z]—{j}d	$O(k+j^2)$
Patterns with length j, where prefix overlaps with a wildcard or a set of characters	.*ab.{j}cd	$O(k+2^j)$

Table 5.2: Size estimates of different types of patterns

	Processing complexity	**Storage cost**
NFA	$O(n^2m)$	$O(nm)$
DFA	O(1)	$O(m2^n)$
K-FA	O(k)	$O(m2^n - \sum \binom{n}{k})$

Table 5.3: Classification of automata where n represents the average length of a pattern and m represents the number of patterns

Estimating non-determinism: It was shown how to estimate the memory resource requirements of K-FA given k. Next we demonstrate how non-determinism of the K-FA is estimated. Estimating the non-determinism of the K-FA is done by computing the average nondeterminism per state of the automaton as is shown in Equation

5.3.

$$\frac{\sum_{states} P(i) \times a}{|s_n|} \tag{5.3}$$

$P(i)$ is the probability of executing a transition with input i, and a is the number of states that is activated when input i is processed. $|s_n|$ is the total size of the input. In our experiments $|s_n|$ is equal to the length of the string that is accepted by the current automaton. Note that initially all symbols in alphabet σ occur with equal probability, however throughout execution $P(i)$s can be updated to $P(i)'$. This presents a problem because when $P(i)$ is updated, the nondeterminism of K-FA can increase/decrease thus violating the guarantee of our automaton that at most k states are simultaneously active. When this occurs a set of merge/split operations needs to be performed to decrease/increase the non-determinism. In our prototype the threshold α determines the delta such that if $\alpha < |P(i) - P(i)'|$ then the update is performed.

5.3.2.3 Run-Time

During run-time three main steps are performed: (i) decide whether a reconfiguration is needed (ii) when a reconfiguration is needed, find the most applicable configuration from the search space and (iii) perform the necessary merge and split operations to achieve the desired configuration. In this section we give a detailed overview of these three steps.

A *reconfiguration* is the process of applying a set of *operations* to an automaton to make it more or less deterministic depending on the available system resources. More determinism is beneficial for systems with high available memory but low processing availability. Less determinism is attractive for systems with low memory resources and high processing availability. Valid *operations* that can be applied to an automaton include a *merge* operation and a *split* operation. The *merge(S_i, S_j)* operation is performed by merging two simultaneously active states S_i and S_j (for input a) thus

decreasing k. The *split(S_i)* operation is performed by splitting state S_i into two states S_{im} and S_{in}. Both the *split* and the *merge* operations are performed with a goal of minimizing the overall size of the resulting automaton.

To change the automaton m (into a more deterministic or into a more non-deterministic system m') we search for a template automaton that requires the least number of operations to derive m'. Once the automaton m_i, which will be used to derive m'', is determined, the reconfiguration process can start by applying the *merge* or the *split* operations required to achieve the level of non-determinism of m'. The operations are applied continuously until the desired level of non-determinism (k) is obtained. The merge and split operations are *safe*, meaning that after reconfiguration the automaton is left in the same state as prior to reconfiguring and the matching process can continue.

When to reconfigure: The reconfiguration process can be required or optional:
1) *Reconfiguration is required:* Occurs when the available system RAM does not meet the memory requirements of the current automaton configuration.
2) *Reconfiguration is optional:* Occurs for two reasons: (i) system RAM is increased or (ii) the percentage of free cores is increased. In (i) we can decrease the non-determinism (via the *merge* operation) of the current configuration thus increasing system performance if the current nondeterminism of the system is greater than the number of available cores. In (ii) nondeterminism can be increased (via the *split* operation) to match the number of free cores. If both (i) and (ii) occur simultaneously, Morpheus considers (ii) first when searching for a feasible configuration because applying the *split* operation will not increase memory usage and therefore is considered a *resource safe* operation.

Search Space: When searching for a solution we consider the available resources R to find the target configuration T that maximizes the throughput. T is determined by first considering the available cores k and estimating if $|T_k| \leq R_m$ where R_m is

the available memory resource and T_k is the target automaton with k level of nondeterminism. Once T_k is determined we search for a template M which will be morphed into T_k. M is picked such that the potential number of operations ($|(|T_k| - |M|)|$) required to derive T_k is minimal. To find both T_k and M in our implementation we perform a binary search. This is an applicable solution because in our case $max(k)$ (maximum level of nondeterminism) is about 16 (number of concurrent threads in a system) which is acceptable for today's multi-core systems.

Next we discuss the formulation of the split and merge operations as an optimization problem.

Split Operation: Suppose we have a DFA automaton M and the available cpu or free memory resources change. Suppose we are given the available parallelism of the system k. Assuming that $k > 1$, we want to increase the parallelism (non-determinism) of M while bounding the number of concurrently active states at k resulting in automaton M'. The nondeterminism of automaton M is increased via a *split* operation. Intuitively the split operation takes state S_i and splits it into two states S_{i1} and S_{i2}. All transitions belonging to S_i now are part of S_{i1} and S_{i2}. The split operation is repeated until nondeterminism of M is equal to k. Furthermore S_i is selected in such a way that $|M'|$ is minimized. The above problem is referred to as the *State Split Problem (SSP)* [XJS10], and if we let S be the set of all NFA states, N be the set of states in M (i.e. all NFA state combinations), S_i $(i = 1, ..., N)$ be i-th combination of NFA active states, $S_{i,j}$ be the j-th subset split from S_i and Q be the union of $S_{i,j}$ (i=1,...,N; j = 1,...,k) the SSP can be formulated as the following equation:

$$
\begin{aligned}
& min|Q| \\
& \qquad s.t. \\
& \cup_j S_{i,j} = S_i; (i = 1, ..., N; j = 1, ..., k) \\
& Q = \{S_{i,j} | i = 1, ..., N; j = 1, ..., k\} - \{\emptyset\}
\end{aligned}
\tag{5.4}
$$

Equation 5.4 minimizes the number of distinct $S_{i,j}$ which are non-empty and the union of which will translate into a smaller automaton. Authors in [XJS10] show that the *SSP* problem is NP-complete for any $k > 1$ and therefore we rely on a heuristic to solve it. The heuristic used by Morpheus depends on u which specifies the minimum size of each $S_{i,j}$, which restricts the number of resulting sets. With this heuristic, when splitting S, at least one of the splits must be of size u. Once the $S_{i,j}$ is obtained, the problem reduces to a *minimum subset cover* problem where the minimum number of sets has to be picked to minimize $|Q|$. We implement other heuristics, such as limiting the amount of non-parallelism to consider k, requiring that there is no overlap between the subset split from the same NFA active state combination and requiring that at least one of the subset splits $S_{i,j}$ for some NFA active state combination y must satisfy $S_{i,j} = y$ where $i \neq y$. For more details on heuristics and optimizations used see Section 5.3.2.6. The SSP problem can be formulated as a Linear Program (LP) which is presented in Equation 5.5.

$$
\begin{aligned}
min \sum_{T} X_T & \\
& \textstyle\sum_{D \in S} X_D \geq 1 \; \forall \; States \; S \\
& X_T \geq X_D \forall \; New \; States \; T \\
& 0 \leq X_T, X_D \leq 1 \forall \; decompositions \; D
\end{aligned}
\tag{5.5}
$$

The LP formulation presented in equation 5.5 can be interpreted as minimizing the number of new states, with the following conditions: (i) for each decomposition of states X_D, we must pick at least one such decomposition and (ii) the number of new states must be at least as great as the number of decompositions, which guarantees that all original states can be derived. X_D is generated using the heuristics discussed. Note that during run time we collect access statistics p_i for each state i, thus all subsets are weighted by p_i. Initially the weight of all subsets is the same.

Merge Operation: Similarly to the split operation above we can define a linear

program that solves the merge optimization problem. In the merge optimization problem a set S of states has to be derived that is a combination of a subset of the current K-FA states the union of which minimizes the overall size of the resulting automaton. The relaxed LP is depicted in Equation 5.6. The LP in equation 5.6 minimizes the number of unique sets as a result of merging two states from the possible K-FA 2-state combinations. The variable X_T represents the number of new states as a result of picking X_M.

$$
\begin{aligned}
min \sum_{T} X_T & \\
& \textstyle\sum_{M \in S} X_M \geq 1 \forall\ States\ S \\
& X_T \geq X_M \forall\ New\ States\ T \\
& 0 \leq X_T, X_M \leq 1 \forall\ decompositions\ D
\end{aligned}
\tag{5.6}
$$

5.3.2.4 Update-Time

Based on the history of reconfigurations performed, an *update* of pre-computed configurations (*templates*) can be carried out. In other words, we may *change* a pre-computed configuration if a *better* pre-configuration can be generated which will result in a lesser dynamic configuration overhead. A *better* pre-configuration p' may arise if an existing pre-configuration p is infrequently used thus utilizing resources without having a significant impact on the reconfiguration time. *Changing* a precomputed reconfiguration p involves either removing p or *deriving* p' via *split* or *merge* operations of p. Additionally, when resources change, we may need to remove or add a reconfiguration. Therefore the formulation in 5.2 must be rephrased to encompass the dynamic nature of the problem in terms of time t.

At time t two scenarios might occur:

- Resources may change.

- By keeping run-time statistics, we may notice that some pre-configurations p are less frequently used compared to others and the available resources can be utilized better by pre-computing other configurations instead of p'.

Thus, at time t, R_{tj}, specifies the capacity of resource j at time t, $w_{it}d_i$, specifies the minimum number of operations needed to *derive* automaton (template) i at time t, w_{it} specifies the weight of automaton i. w_{it} increases with the increase of the frequency of usage of i. By changing the maximum distance d_i via the weight w_{it} as a function of time t our pre-configuration algorithm becomes dynamic with respect to both the time and the available resources. The above formulation can be expressed using the LP in Equation 5.7.

$$
\begin{aligned}
minimize: & \\
& max(\sum w_{it}d_i) \\
& s.t. \sum m_{ij}x_{ti} \leq R_{tj} \\
& x_{it} \in \{0,1\} \forall i \in m, t \in T
\end{aligned}
\tag{5.7}
$$

Thus, based on the template usage statistics we can remove a template p, modify it to p' or add a new template p''. The weight w_{it} is adjusted by a constant c each time template i is used. In our prototype version of Morpheus c was specified manually, and a fully automated approach is part of the future work.

When solving the optimization problem in Equation 5.7, we have to estimate if updating template i to j is 'worthwhile'. When template i needs updating, we term this event as template i *expiring*. We can estimate the average throughput of i and j by calculating the average nondeterminism and the available processing power present. Specifically, the overall throughput can be estimated as

$$
T = \alpha \sum_j \frac{\sum_i (d(s_j) \times (p_j + (1 - p_j))}{|s|} \tag{5.8}
$$

where $|s|$ is the number of states of the current FSA, α is a system maximum throughput, $d_s j$ is the determinism measure of state j (see Equation 5.3. Notice that if s_j has low throughput (e.g ≤ 0.1) and its activation probability p_j is also low (close to 0), the overall throughput will be close to 1 because the slow s_j will not be active frequently hence the term $(d(s_j \times (p_j) + (1 - p_j))$. Thus, T (the overall throughput of the system) is equivalent to the average throughput of all states j of the current automaton.

We can also track the average time i_t before template i *expires*. Thus to check if updating i is 'worthwhile' we compare the cost with the benefit of updating. The cost of updating is the reconfiguration (update) time and the benefit is the time it takes for i to process the same data as j ($\frac{T_i \times i_t}{T_j}$) where T_i is the throughput of automaton i. Thus reconfiguration is performed if the benefit is greater than the cost. Because of the benefit analysis before reconfiguration, Morpheus does not suffer from too much oscillation between different reconfigurations. Furthermore the response frequency to the changing system environment can be easily tuned in Morpheus by adjusting the minimum gap required for reconfiguration between the benefit and the cost.

5.3.2.5 Implementation

In this section we briefly discuss the implementation details. The components shown in Figure 5.5 are implemented in Java. We use *lp_solve* [lps] to solve the Linear Program optimizations defined. To solve the fractional optimization LP problems presented, we use the technique of *rounding* [RT85], which in *expectation* will give an optimal integer solution. Thus suppose that a solution to an instance of the LP in Equation 5.5 is as follows: X_{D_1} = 0.2 X_{D_2} = 0.5 and X_{D_3} = 0.3. Via *rounding* we can get a good approximation to the above solution by rounding the largest X_D to 1 and others to 0, therefore X_{D_2} would be rounded to 1 and others to 0. Our experiments clearly show that using a rounding technique in expectation produces a good solution.

During run-time the access probabilities of states have to be updated and automaton reconfiguration has to be performed without losing the currently active states. The original NFA states are stored in a hash-table, which allows for a fast update of access probabilities of states. During reconfiguration, the necessary state transitions are copied to the currently active automaton. Furthermore, upon reconfiguration, the state of the previous automaton M_{old} should be restored in the new automaton M_{new} for the pattern matching process to continue from the same point. This is achieved by recording the active state labels in M_{old} and setting the same states to be active in M_{new}. By keeping the original NFA states separate from the states in the currently active automaton, the update of access probabilities and the reconfiguration of the current automaton can be performed efficiently.

5.3.2.6 Optimizations

In this section more details about the optimization techniques used for solving the split problem are presented. In [XJS10] it is shown that the upper bound on $|Q|$ (the size of unique states after a split) using the heuristic presented in Section 5.3.2.3 is given by Equation 5.9.

$$|Q| \leq 2u' + N - \frac{\sqrt{2\alpha}}{m^2}t'^2 \tag{5.9}$$

The run time of the SSP algorithm [XJS10], that is used to determine the split, is too expensive for the online environment thus we use an approach based on Fractional Linear Programming that provides a fast approximation (see Figure 5.3.3.5). Furthermore the heuristic in [XJS10] generates a search space that can still be very large. We make this simple observation: a subset split must be such that: $\exists i S_i \in S_{i,j}$. In other words to reduce the size of the overall automaton, and thus the memory usage, a subset split must contain at least one of the nodes. If no such subset exists then the split is not performed. This observation in practice further reduces the search space. Figure 5.6 il-

lustrates this heuristic, where the arrow from S_i to S_j indicates that S_i is a subset of S_j. Thus intuitively our LP solver would first pick the largest-degree subset whose sizes are larger than the threshold, and the degrees are larger than 1. Thus, first {A,D,O} is picked, and {A,D,O,G} is split into {A,D,O} and {G}. The sets {A,D,O,G} and {A,D,O} are then removed from the graph as well as the edges between these two vertices. Then we pick {G,O} and {O}, and split {G,O} into {G} and {O}. Removing all edges and nodes associated with {G,O} and {O} leaves an empty graph, therefore the final graph set of nodes resulting from the split are {A,D,O}, {O}, {G}.

Figure 5.6: Example of subset split generation

Other optimizations used in Morpheus include the restriction on the amount of non-determinism to consider (k) and the *isolation constraint*. The bound on the amount of non-determinism is usually determined by the number of cores or by the number of concurrently supported threads. The bound on k reduces the search space during *runtime*. Furthermore similar to the heuristic in [XJS10], we add the following isolation constraint:

$$S_{i,j}\hat{S}_{i,k} = \emptyset(\forall j \neq k, i = 1, ..., N) \tag{5.10}$$

The isolation constraint requires that there is no overlap between the subset split from the same NFA active state combination. Even with the isolation constraint, as evident from the experimental results (see Figure 5.3.3.5), the approximation obtained by Morpheus for the split and merge operations is reasonable.

5.3.3 Pattern Matching Experiments

In this section we begin by demonstrating the accuracy of our resource and throughput estimation models of the K-FA automaton. The main motivation of our research is to improve the performance of pattern matching in environments where system resources constantly change. Because we are the first to propose a dynamic pattern matching

system, we compare our approach against the fastest known static pattern matching system, YFilter [DF03], to validate our claim that dynamic pattern reconfiguration can dramatically improve pattern matching performance. YFilter is a query processing engine for XML, and as will be shown XML queries will benefit from the techniques discussed, but so will many other query languages (e.g., K*SQL [MZZ10]), for which FSA provide a powerful execution model as it has been widely recognized in previous literature [DF03, YCD06, PC03, GGM04] We then study how the number of precomputed configurations affect the dynamic reconfiguration overhead and the amortized memory usage. The goodness of the split/merge approximation algorithms presented in Section 5.3.2.3 is also measured with respect to the optimal. The effect of our dynamic reconfiguration strategy is then measured on resource utilization and on the overall performance given workloads of different severity levels. As is explained in more detail in the following section, a *severity level* refers to the current input speed measured in characters per second (*cps*) and to the current memory and CPU load due to non-pattern matching related processes.

5.3.3.1 Experimental Setup

Workloads: There are two types of workloads that we use to test our system: (i) Server workloads and (ii) Input (transaction) workloads. For (i) we use our *synthetic micro-benchmark* (SMB) and the real work-load datasets. Using SMB we derive five independent workloads. In order to specify these workloads we use a *BurnInTest* [bur11] utility with which we specify the amount of *CPU* and *RAM* to utilize. By precisely controlling these parameters we obtain five different workloads of varying time-patterns (sinusoidal, sawtooth, flat with different amplitude and period etc). The goal of these workloads is to validate that Morpheus can dynamically adjust the underlying automaton under various system loads. Furthermore these workloads test the

ability of our system to automatically recognize an opportunity for dynamic reconfiguration. Furthermore we obtained real-world load-statistics from *Wikipedia* and *Nasdaq* servers. Pattern matching is heavily used in these services to respectively detect any malicious activity and investment opportunities in the current stream of stock prices.

To simulate the transaction load we use the MIT Lincoln Lab intrusion detection traces [MIT]. These traces provide a way to test a real system against the network intrusion attacks. To process these traces we used a set of patterns found in Snort network intrusion detection system [Roe99]. The advantage of using the real MIT Lincoln Lab traces is that it tests the nature of real attack traffic that includes periods of spurious activity.

A high severity level refers to the system load of 90% for both CPU and RAM (10% of resources available) and the input speed of 100K *cps*. A low severity level refers to 10% system load and 1K *cps*.

Data Sets: We use a mixture of real pattern queries and synthetically designed queries. Real pattern queries are retrieved from Snort [Roe99] Network Intrusion Detection system. To generate synthetic queries, we design a query generation tool to synthesize pattern queries with predictable properties.

Our query generator tool responds to a number of parameters presented in Table 5.4.

System Information and Implementation: All experiments were run on an eight core 2.0GhZ with hyper-threading server with 16GB of RAM running Java 1.6. We implement the monitoring tools in Java and utilize SSH to collect OS-level statistics. Morpheus utilizes an open source LP solver *lp_solve* [lps] to solve our global linear approximation program. We have used the standard automaton library for java to generate NFAs and DFAs [aut].

Synthetic Query Parameter List		
Parameter	Range	Description
Q	1000 to 500000	Number of queries
W	0 to 1	Probability of a wild-card "*" occurring in a qeury
Distinct	0-100%	Percentage of unique predicates
P	0 to 20	Number of predicates per query

Table 5.4: Parameters for synthetic query generation

5.3.3.2 Resource Estimation

Given an environment with limited resources, the resource consumption by a particular configuration c must be estimated without explicitly computing c. Based on the observations made in Table 5.2 we used a simple linear approximation for the size of the automaton. This experiment measures the estimated size of our automaton versus its actual size, for a given level of non-determinism K. The results indicate a fairly small overestimate of the memory usage of *K-FA* (Figure 5.7). This is expected because frequently the memory is reduced due merging opportunities of duplicate states. The CPU usage is slightly over-estimated (Figure 5.8), which can be explained by an irregular work-load. The memory overestimation problem is mitigated by a reasonable over-estimation of the *RAM* resource is acceptable because it is considered to be a limiting constraint.

5.3.3.3 Comparison Against YFilter

We compared Morpheus against the state of the art XML index YFilter. The results can be seen in Figure 5.9. We have varied the input speed *cps* given an automata consisting of 500 HTTP pattern queries found in Snort. The available system resources are also varied gradually with the increased *cps* starting at 90% and are decreased to

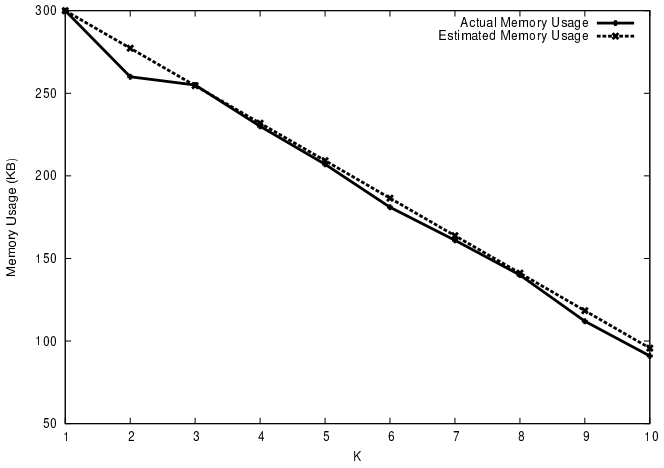


Figure 5.7: Actual vs estimated resource usage (memory)

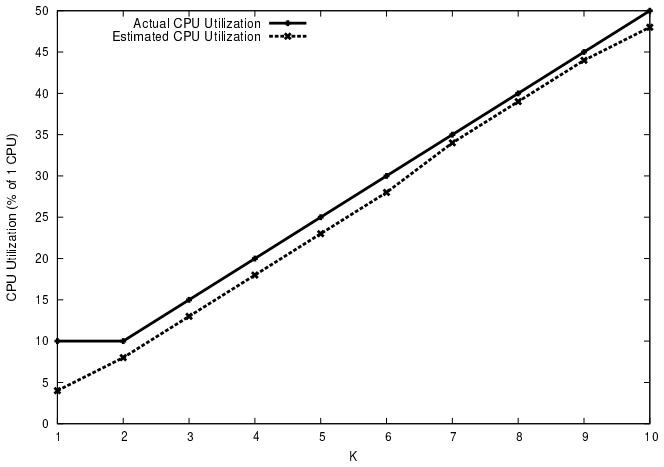


Figure 5.8: Actual vs estimated resource usage (CPU)

10%. The results confirm that due its inherently static NFA implementation, YFilter is unable to adapt to a changing system environment. We have varied the input speed, measured in characters per second (*cps*), between 500 *cps* and 128K*cps* and recorded the average time for both YFilter and Morpheus to perform a match. The match time can be interpreted as latency for the overall system. Thus, even though YFilter has a fast and memory-light approach to represent patterns, its inability to adjust to the changing system environment causes it to suffer from poor performance when a large *cps* is used. Our approach is able to quickly adjust to the current system resources and thus shows a five-fold performance gain over YFilter. Morpheus experiences slight

increase in system resources during reconfiguration (roughly 3%), however the overall matching time of Morpheus is still significanly less than that of YFilter. Given the input speed of 1K *cps* Morpheus outperforms YFilter in matching speed 1100230ns to 50000000ns, and for 128K *cps* the performance gap increases with Morpheus and YFilter having match times of 1100230ns to 68000000ns. It should be noted however that in the case where the input speed and system load is overly high ($> 128K$ *cps* and 90% respectively), our approach defaults to that of the YFilter, namely utilizing a full NFA for pattern matching.

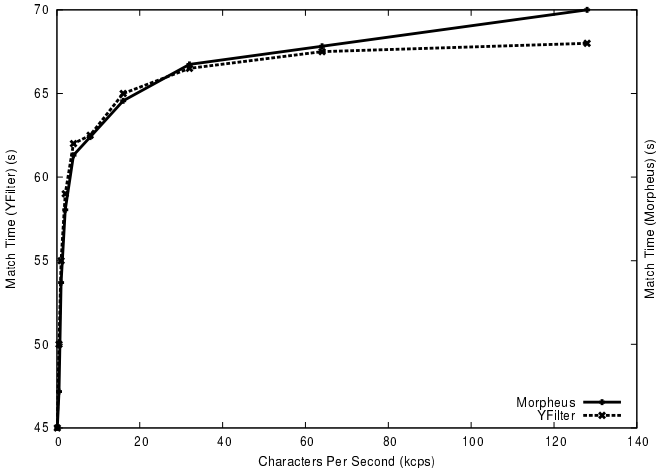


Figure 5.9: Matching speed

5.3.3.4 Optimal Number of Precomputed Configurations

The experiment in Figure 5.10 shows the effect of the number of precomputed configurations on the dynamic reconfiguration time and on the memory usage. Given a current system C, a target system T and a set of precomputed configurations $S = \{S_1, S_2, ..., S_n\}$, the dynamic reconfiguration time includes the time to search for a solution in S given C and T, morph the found solution S_k into T and update S depending on the reconfigurations performed so far (see Section 5.3.2.4). The *amortized memory* usage includes the memory used for the buffer to hold the tuples from the

input stream while reconfiguring and the size of S. Note that the longer it takes to perform a reconfiguration, the larger the input buffer has to be. Figure 5.10 shows that in our case, after precomputing more than five configurations, the search and update time of S starts to dominate and the performance deteriorates. Furthermore the amortized memory usage starts to increase after five precomputed configurations because an additional precomputed configuration consumes more memory than the decrease in the size of the temporary input memory buffer needed. From this experiment we can conclude that given our system set-up and the workload from SMB the optimal number of precomputed configurations to achieve the highest throughput should be five.

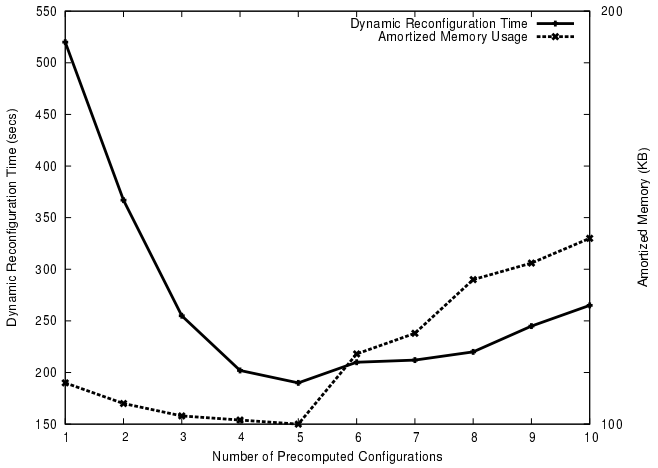


Figure 5.10: The effect of increasing the number of preconfigurations on the dynamic reconfiguration time and on the amortized memory usage

5.3.3.5 Optimal Split vs Approximate Split

In Section 5.3.2.3 we have presented an approximation *LP rounding* formulation for the split and merge problems. In this experiment we measure how the approximation factor ϵ (error) affects the resulting size of the automaton. The error in Figure 5.11 measures the fraction of the set split generated. Thus error of 'n' implies that $\frac{1}{n}$ subset splits out of possible $S_{i,j}$ splits are generated. Therefore when ϵ=1, we get the best split

because our search space includes a collection of all possible splits. By decreasing ϵ we spend more time on deciding how to perform a split but in return lower the size of the resulting automaton. We observe that the benefit of a lower ϵ decreases rapidly (in terms of the memory usage). Furthermore the less time we spend on choosing the right split, the less of an input buffer we need, thus the point p of where the buffer size and the automaton size intersect is the optimal ϵ necessary to minimize the overall memory usage for our SMB. The optimal value of p can be determined by a simple experimental process that is being incorporated into Morpheus.

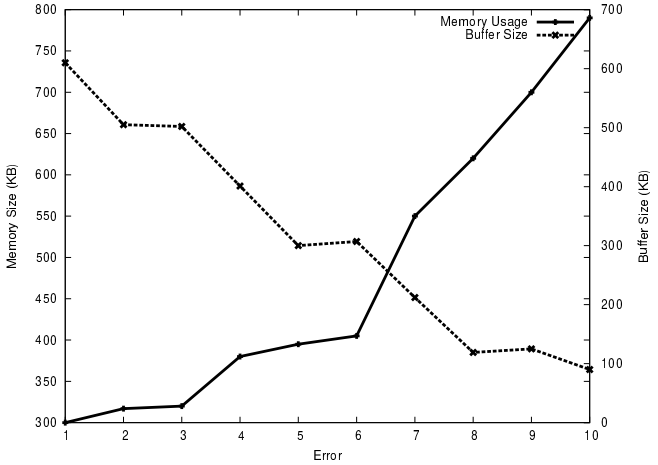


Figure 5.11: Optimal vs approximated split operation

5.3.3.6 Effects on Throughput

In this experiment we evaluate the effect of dynamic reconfiguration on the overall throughput. The evaluation is done by comparing the memory usage increase and the overall throughput increase with respect to a situation where there is no dynamic reconfiguration. Figure 5.12 shows that for a fraction of the cost in memory increase we get a great increase in throughput. The x-axis in Figure 8 represents the severity of the underlying system changes (i.e. severity of 1 implies that only memory slightly increased/decreased and severity of 10 indicates that the CPU usage, memory and

access patterns have changed dramatically). To simulate severity we used the *Burn In Test* utility to specify precisely how much *CPU* and *RAM* resources to consume.

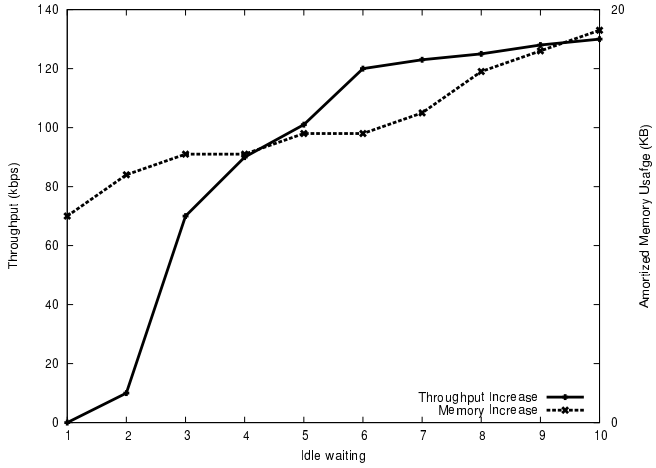


Figure 5.12: Affects of the dynamic update on throughput

5.4 Conclusion

In this chapter we presented two domains where data complexity is useful. In association discovery we provided the first set of experiments showing the relationship between problem complexity, association finding algorithms and sample size. For a fixed algorithm we validated a natural hypothesis that as relationship complexity increases, the required sample size also increases. For other association finding algorithms we observed that as relationship complexity increases the sample size also increases in proportion to the complexity of the algorithm. Using the problems of known levels of difficulty we were able to validate the chosen measure of association complexity.

In section 5.3 we described how the model complexity can be adjusted automatically with changing resources to optimize the overall throughput in the context of pattern matching. The resulting system, called Morpheus, dramatically improves the performance of pattern matching with the help of dynamic reconfiguration of the underlying pattern automaton given a changing system stress load. Morpheus precom-

putes several key pattern configurations (templates) that are then morphed into a required form during run-time depending on the available system resources. Our extensive experiments show that the approach proposed here is attractive due to its low amortized memory overhead and an order of magnitude decrease in dynamic reconfiguration time when compared to standard approaches that do not use templates. In the future we hope to add load-shedding and dynamic addition and removal of patterns support into Morpheus.

The rapid growth and availability of massive time-series datasets and a throng of algorithm proposals to deal with this data are creating a demand for ways to sort through all the possible tools that can be applied over this data. We believe that the presented descriptors of data complexity are useful for identifying and comparing the different algorithm types. Our work with data complexity will continue in the direction of generalizing these descriptors towards their applicability to other problem domains.

CHAPTER 6

Applications

In this chapter we provide an important application of quality assessment and sample size prediction that deals with computing nonlinear associations and demonstrates the applicability of our approach on the static 'Big Data' problem. Although it is the application we were directly involved in, the approximation approaches presented in this dissertation are general enough to be applicable to a variety of other fields as is discussed in sections 7.2, 1.1 and 6.2.

6.1 Association Finding

In this section we focus on applying our approximation approach to the problem of association computation. We propose a scalable end-to-end system, called Associate-TS, for finding approximate nonlinear associations in time-series (TS) data. We use the two critical parts for approximation support presented in chapters 3, 4 and 5: (i) sample quality assessment and (ii) sample size estimation. Recall that to achieve (i) we use the Bag of Little Bootstraps (BLB) along with other recent advances in bootstrap and time-series theory to provide an effective Hadoop-based implementation for assessing a time-series sample quality. To achieve (ii), in sections 4 and 5 we proposed a technique called SS-TS that uses the recent learning and data complexity theory to estimate the required sample size (SS). As we show in this section, the application of the above framework to the Twitter dataset demonstrates that Associate-TS provides

substantial improvements in processing speeds while placing the user in control of the result accuracy.

6.1.1 About Association Discovery

Problem and Motivation: Consider a time-series dataset with hundreds, or even millions of variables such as those common in domains as varied as genomics, physics, political science, economics and social media. A natural challenge is to discover the many unknown relationships in this massive dataset. If you do not already know the relationships to search for, how do you quickly identify the important ones? There are billions of variable pairs to consider; far too many to provide the user with the result in a reasonable time, making this question of growing importance [RRF11].

One way to approach this challenging problem is to use approximation techniques. For that, (i) one could take a sample S of the data, (ii) compute a measure of the inter-variable dependence M over S using existing approaches [RRF11], (iii) estimate the accuracy ξ of M and (iv) increase the sample size until the required accuracy ϵ is reached. For this approach to work in practice, however, we must develop scalable accuracy and sample size estimation techniques.

Recall our quality assessment technique described in chapter 3. To provide a scalable approach for assessing the quality of large time-series data samples we built on previous work of bootstrap [LZZ12a, KTS12b, KTS11] and time-series theory [BK99, PR94] We introduced the BLB-TS framework that takes advantage of the trend in computational resources which is shifting towards a multicore and distributed architecture with cloud services providing thousands of processing units. Here we use the framework presented to support the use-case described below.

Use-Case: As an input to our system, the user specifies a machine learning (ML) algorithm A, its input I and desired accuracy ϵ. Without the loss of generality of

our approach, we restrict ourselves to M = Maximal Information Coefficient (MIC) [RRF11], an overview of which is given in subsection 6.1.2, and $I = M(i,t)$ where M is a matrix with i representing a variable, t representing the time and $M(i,t)$ representing a value of i at time t. Our job is to compute all pairs association between all i variables using MIC. We choose MIC because it can find nonlinear associations as is described in more detail in section 6.1.2. While we restrict our discussion to association computation, our approximation techniques can be applied to other machine learning algorithms such as SVM and linear regression as discussed in subsection 6.1.3.

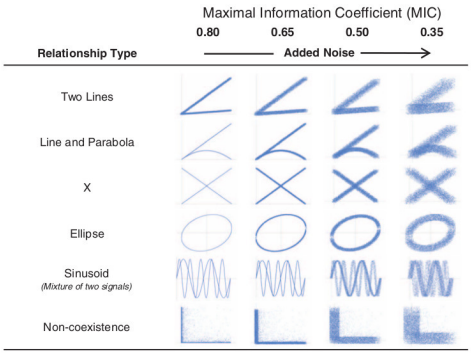
As a concrete application of Associate-TS consider the Twitter analytics workflow shown in Figure 3.12. The goal of this workflow is to compute a similarity score of all hashtag pairs using MIC. The workflow consists of four stages: ① Get latest Twitter data of size $W \rightarrow$ ② Estimate error based on a sample S of $W \rightarrow$ ③ Compute the similarity between Twitter hashtags[1] from $S \rightarrow$ and ④ Update internal tag similarity model. Step ② consists of BLB-TS and of SS-TS described in sections 3.2 and 4 which are respectively used for accuracy estimation and sample size prediction. Step ② creates s subsamples from the original sample. To compute the error ξ_i for sample size n, s subsamples of size b are taken. Then, each subsample b_i is resampled r times and for each resample the association matrix M_i is computed. The variance of b_i is determined by the variance of the r association matrices computed. Cumulative error is then obtained by averaging $s^{-1}\sum_{i=1}^{s}\xi_i^*$. The $s^{-1}\sum_{i=1}^{s}\xi_i^*$ value would correspond to a point on the learning curve for a given sample size n. The value of n is then adjusted based on the convergence rate of the learning curve and the required user error.

As we show in this section, our preliminary experiments show the effectiveness of Associate-TS for applications such as the analytics workflow shown in Figure 3.12,

[1]The # symbol, called a hashtag, is used to mark keywords or topics in a Tweet.

Relationship Type	MIC	Pearson	Spearman	Mutual Information (KDE)	Mutual Information (Kraskov)	CorGC (Principal Curve-Based)	Maximal Correlation
Random	0.18	-0.02	-0.02	0.01	0.03	0.19	0.01
Linear	1.00	1.00	1.00	5.03	3.89	1.00	1.00
Cubic	1.00	0.61	0.69	3.09	3.12	0.98	1.00
Exponential	1.00	0.70	1.00	2.09	3.62	0.94	1.00
Sinusoidal (Fourier frequency)	1.00	-0.09	-0.09	0.01	-0.11	0.36	0.64
Categorical	1.00	0.53	0.49	2.22	1.65	1.00	1.00
Periodic/Linear	1.00	0.33	0.31	0.69	0.45	0.49	0.91
Parabolic	1.00	-0.01	-0.01	3.33	3.15	1.00	1.00
Sinusoidal (non-Fourier frequency)	1.00	0.00	0.00	0.01	0.20	0.40	0.80
Sinusoidal (varying frequency)	1.00	-0.11	-0.11	0.02	0.06	0.38	0.76

(a) Scores given to various noiseless functional relationships by several different statistics [RRF11].



(b) Several relationship types and their MIC scores with added noise [RRF11].

Figure 6.1: Different relationships and corresponding scores assigned by various associding finding algorithms.

as well as for other ML workflows including classification and regression (see section 6.2). Finding associations is a computationally expensive task, and in our use case a 2,000,000 by 4,000 matrix takes over a month to process without approximation. In this section we show how using the techniques presented in this dissertation we can dramatically decrease the required time while only marginally sacrificing the result quality.

6.1.2 Dependency Measure

As presented in section 2.4.2, the Pearson Correlation coefficient r is often used to measure the strength of association between a pair of variables. Given two signals x and y of equal length m, with respective averages μ_x and μ_y and standard deviations σ_x and σ_y their Pearson correlation coefficient is defined as:

$$corr(x, y) = \frac{1}{m} \sum_{y=0}^{m-1} \left(\frac{x_i - \mu_x}{\sigma_x} \right) \left(\frac{y_i - \mu_y}{\sigma_y} \right)$$

For example, the Pearson correlation between the height of a child and their parents is $r \approx 0.5$ [Spe] and that of the wheat yield and annual rainfall is $r \approx 0.75$ [Spe]. Pear-

son's r, however, only captures linear correlations, and it is not applicable to signals which have nonlinear associations.

In this section, we use the Maximal Information Coefficient (MIC) [RRF11] algorithm which provides a normalized association score for nonlinear relationships and reduces to Pearson's in the linear case. Intuitively, MIC is based on the idea that if a relationship exists between two variables, then a grid can be drawn on the scatterplot of these variables that partitions the data to encapsulate that relationship [RRF11]. Therefore, to compute the MIC score of two signals, all grids up to a maximal grid resolution, which depends on the sample size, are explored, computing for every pair of integers the largest possible mutual information (MI) achievable by any x-by-y grid applied to the data. These MI values are then normalized to obtain modified values between 0 and 1 and to ensure a fair comparison between grids of different sizes. Specifically, the MIC statistic is the maximum value in the characteristic matrix $M(D)$ defined as:

$$M(D)_{x,y} = \frac{I^*(D,x,y)}{logmin\{x,y\}} \tag{6.1}$$

where D is a finite set of ordered pairs, and x, y correspond to an x-by-y grid into which x and y values of D can be partitioned into. I^* corresponds to the maximum mutual information of D when it is partitioned into a specific x-by-y grid G. Note that $0 \leq I^*(D,x,y) \leq logmin\{x,y\}$ therefore dividing by $logmin\{x,y\}$ bounds $M(D)_{x,y}$ to a $[0,1]$ range.

The authors in [RRF11] show that with probability approaching 1 as sample size grows, (i) MIC scores approach 1 for all never-constant noiseless functional relationships; (ii) MIC scores approach 1 for a large class of noiseless relationships and (iii) MIC scores approach 0 for statistically independent variables. Simulations in Figure 6.1a show that for noiseless functional relationships with $R^2 = 1.0$ receive MIC scores approaching 1. For various other functions and various noise levels, MIC also roughly

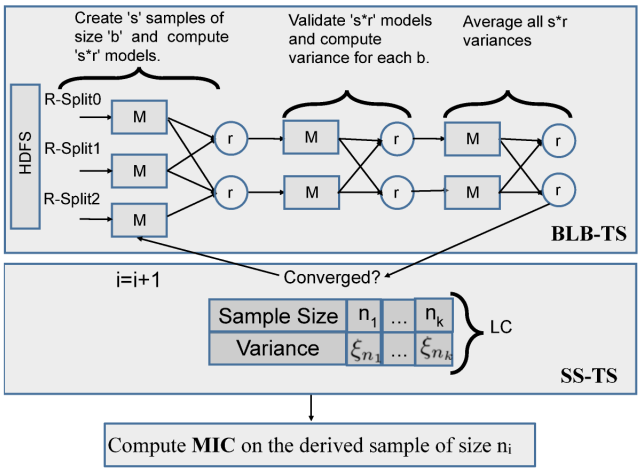


Figure 6.2: Associate-TS workflow architecture.

corresponds to the coefficient of determination R^2 relative to each respective noiseless function. For a wide range of associations that are not well modeled by a function, authors also show that MIC scores degrade in an intuitive manner as more noise is added (see Figure 6.1b).

6.1.3 Implementation

Although the techniques presented in this section are platform independent, we have chosen to implement Associate-TS in Hadoop to take advantage of the Hadoop infrastructure. Hadoop's MapReduce framework makes it easy to send data to random nodes via the widely-adopted <*key*, *value*> constructs that we use to provide an i.i.d. as well as a time-based sample described in this section. We hope that Hadoop's popularity maximizes the potential for practical applications of our new technology.

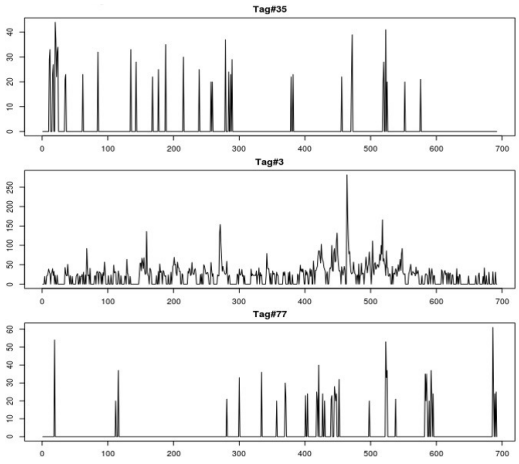
Next we describe the Associate-TS architecture shown in Figure 6.2 and key implementation decisions.

System Architecture: Figure 6.2 presents an overview of the Associate-TS architecture. Stage 1, the sample-size estimation stage (SS-TS), determines the required sample size and initializes the learning curve. Given the current sample size estimate

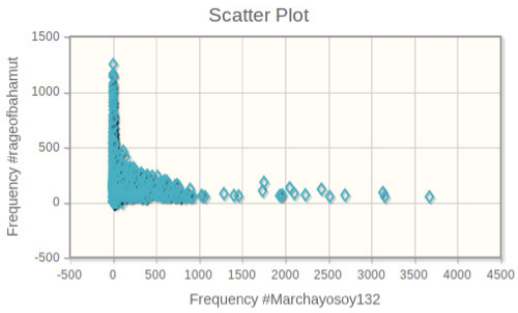
n_i, stage 2, the accuracy estimation stage (BLB-TS), is executed. If the error for a given sample size is too high, the workflow returns to stage 1, until the accuracy requirement is satisfied. Otherwise, the computation of the association matrix over n_i using MIC is performed which returns the final result to the user.

The key entities of Associate-TS are (i) the Learning Curve, (ii) the Model, (iii) the Model Validator, (iv) the Data Path and (v) the Error Calculator. The Model class holds model parameters and is a super class for entities such as ClassifierModel and AssociationModel representing Classifiers and Association models respectively. The Validator class tests the given model, and hence it is a super class of ClassifierValidator and AssociationValidator. As an input, the Validator class takes a model M and outputs a ValidatedModel object, which for a classifier is an accuracy score and for Association Mining it is an identity operation. The Error Calculator takes in a Set of ValidatedModel objects and outputs a single Variance value v_i. The Learning Curve class stores the output from the Error Calculator as $< n_i, v_i >$, and depending on the accuracy score, is responsible for picking the next set of inputs from the Data Path object or terminating.

Next, we map the above description to a MapReduce environment. The main application is initialized with an empty learning curve, an appropriate model along with a model validator and error calculator. The Data Path is initialized with a set of available splits. The iteration starts by picking 1% of the splits in the Data Path and creating s random subsamples (described below). The reducers then create the appropriate Model M_i for a given subsample b and use the Model Validator to output $< M_i, ValidatedModel >$ which is used as an input to the next MapReduce job. Then, for each key M_i the next MR Job computes the variance with Error Calculator. Finally by sending all v_i to a single reducer the variances are averaged and the learning curve is updated checking for convergence.



(a) Example of the hashtag time-series from the Twitter dataset.



(b) An interesting relationship between hashtags in the Twitter dataset.

Figure 6.3: Associate-TS Twitter results.

Incremental MIC Computation: In the SS-TS component, when increasing the sample size to improve the accuracy, we use a delta maintenance strategy that avoids expensive recomputation of the MIC algorithm on the new sample S' and instead uses the result from the old sample S. To compute MIC incrementally, upon arrival of a tuple t, we update the tuple count in cell i to which t maps in the grid $M_{x,y}$. Note also that due to its small size, we store $M_{x,y}$ in memory. Therefore, by incrementally updating $M_{x,y}$ we incur only the cost of S' instead of $S' + S$.

6.1.4 Results

The goal of the Twitter Associate workflow is to determine the similarity measure across all $\frac{(n-1)(n)}{2}$ hashtag pairs. In particular we have used a real-world dataset of size 4.3TB from Twitter that we have privately collected. This dataset consists of roughly 2 Million hashtags and their frequency in a time-series format. We use hourly granularity and at a maximum, each hashtag has 4,000 observations[2]. Empirical evidence suggests

[2]Not every hashtag contains the same number of observations

that computing an MIC-association matrix for the above dataset is very expensive and without approximation can take over a month to compute on our Hadoop cluster. We use a private Hadoop cluster of size 7 where each node is a 16-core AMD Opteron(tm) Processor 6134 @ 2.3GhZ with 132GB RAM. We use Hadoop 1.0.4 with a total of 140 map slots. Figures 6.3a and 6.3b respectively show the type of time-series and the type of relationships that Associate-TS was designed to deal with and find. Observe that some tags may have many random spikes (e.g., #riot), some tags may be fairly stable (e.g., #youtube) and some may show very little level of activity (e.g., #mypersonal-statement). The association between these tags may result in interesting[3] relationships such as the one shown in Figure 6.3b.

6.2 Extension to other ML Algorithms

The performance of the current implementation of BLB-TS relies on the model computation and error estimation separation. This insight proved invaluable for achieving a simple API design that is easy to understand and test. In the model-error computation separation the error computation is performed in three stages: (a) Model computation (b) Model Validation and (c) Error estimation. These steps are depicted in the BLB-TS box in Figure 6.2. As we explain in this section, the accuracy of a wide range of machine learning algorithms can be estimated using this three-stage workflow.

First, let's apply the three stage accuracy estimation workflow to the problem of estimating the sample error of the inter-variable relationship task. Step (a) would correspond to computing the different association matrices M based on the s samples each of which are of size b (recall Section 3.2.2). Step (b) would be an identity[4] stage

[3]The relationship in Figure 6.3b explains that when people are rioting (#MarchaYoSoy132) they do not play the said video game (#rageofbahamut) and vice versa.

[4]Identity stage refers to the stage that has no effect on its input.

because the association computation task is an unsupervised machine learning problem. Finally, in step (c) we compute the error by averaging the s errors computed for each b. The variance of b corresponds to the variance of the r MIC score matrices computed from resamples of b.

Let's now try to see how the BLB-TS can work for other ML algorithms such as SVM [Bur98] and linear regression (LR). When training the SVM classifier, step (a) would build the s models, and step (b) would validate these models using cross validation. Step (c) would collect the model accuracies from step (b) from which the error will be derived based on the variance of the accuracy scores. Similar to our use-case, step (a) for LR would compute s regression estimates with step (b) being an identity step. Step (c) would compute the variance from the empirical distribution of the r regression estimates for each b.

6.3 Conclusion

The rapid growth and availability of massive time-series datasets is creating demand for scalable and robust techniques for time-series data analytics. To support nonlinear association analytics over massive time-series data we have presented Associate-TS. Using Associate-TS we were able to process the 2,000,000 by 4,000 matrix of Twitter hashtag time-series data in 2 hours compared to the original one month requirement; a greater than 99% execution time speedup while only sacrificing 10% accuracy.

CHAPTER 7

Conclusion

7.1 Summary of Contributions

Our work on scalable approximation has two high-level goals. First we provide the user with the quality assessment score so that she is confident in the results computed from a sample of the data. Second, we provide the support for sample size prediction and algorithm selection so that the user can pick the right sample size and the right algorithm to minimize the overall resource usage. We summarize the key contributions as follows:

- We propose the scalable accuracy assessment framework [LZZ12b] that is based on the resampling method called the bootstrap. Given a sample and the underlying machine learning (ML) algorithm, our framework resamples the input r times and for each resample re-executes the ML algorithm. From the r estimates, the result variance is derived. We also make the bootstrap method scalable through parallelism and applicable to time-series data [LLZ12].

- We then address the problem of (i) sample size prediction [LZZ13] and (ii) algorithm selection [LZL12, LZ12]. Therefore, we achieve two main objectives: (i) the user's need for accurate sample size prediction given the prescribed user accuracy and (ii) the important aspect of algorithm selection to minimize resource utilization. We achieve the sample size prediction using the learning curve and

we achieve the algorithm selection using the recent data complexity theory.

- We evaluate the performance of our approaches using the real world Twitter data and classification datasets from UCI's machine learning repository [Rep10]. We validate that our system is efficient and scalable in the face of massive real-world data. Our experimental results show that BLB-TS is superior to the Stationary Bootstrap for sampling over massive time-series datasets and that SS-TS is an accurate sample size estimator. Another very novel aspect of our work is that we established the relationship between problem complexity, algorithm complexity and sample size and provided experiments to elucidate this relationship. Using Associate-TS we were able to process the 2,000,000 by 4,000 matrix of Twitter hashtag time-series data in 2 hours compared to the original one month requirement; a greater than 99% execution time speedup while only sacrificing only 10% of accuracy.

7.2 Research Directions

We realize that a comprehensive set of methods and systems that support scalable approximation is challenging and it involves many research issues. Only the first steps towards a general solution have been taken in this dissertation, but they have opened up the door to many research opportunities of great significance as described next.

We relied on the bootstrap resampling technique for quality assessment. As discussed in section 2.2, however, other resampling methods exist and analyzing them presents another exciting research direction.

The sample size prediction approaches discussed rely on iterative computation which can make use of the existing optimization in iterative computation. MapReduce-based systems such as HaLoop [BHB10b] and PrIter [ZGG11] provide elaborate iter-

ative support including static data caching and improved convergence performance. Therefore another interesting future direction would be integrating the existing state of the art systems into our approximation workflow for a more efficient learning curve construction.

The recent explosion of query languages shows the growing interest in this field but also leaves plenty of room for approximation support. For example notable languages including SQL-TS supporting regular expressions and backtrack optimization [SZZ01], SASE+ [DIG07] supporting powerful Kleene-closure queries, and K*SQL [MZZ10] provide powerful pattern matching capabilities but have little or no approximation semantics. The recent 10 year VLDB best paper award [MM12] on approximate item-set counting over streams provides a SQL-like language suggesting the interest in approximation support at the language level and leveraging this interest is an ongoing part of our current and future work.

Often in a streaming environment it is difficult to estimate the required window size necessary to achieve a result of a particular quality. Using the approximation framework presented in this dissertation, however, it now becomes feasible to dynamically determine the necessary window size. Currently, in dealing with extreme system load, streaming systems rely on load-shedding techniques [TZ06, TcZ03, CWY05] whereby the data stream management system sheds a fraction of its input to meet the quality of service requirement. Providing accuracy estimates for load-shedding systems is very difficult especially for complex analytics, however it is still better then the alternative–system crash. As an ongoing part of the future work we will extend our data-stream management system, called Stream-Mill Miner (SMM) [TLM11], to dynamically determine the required window size to meet the prescribed user accuracy requirement.

Furthermore, our work on Associate-TS will continue in a direction of providing

more rigorous statistical guarantees of BLB-TS' quality assessment as well as making further strides in the area of data complexity.

7.3 Closing

We live in the world where the amount of data continues to grow. With an unbounded data growth comes the increasing need to make sense of all the data and hence approximation support becomes critical. In this dissertation, we have addressed the problem of quality assessment and sample size prediction that when combined help the user minimize the needed resources while ensuring the desired accuracy. Our approach is scalable and is applicable to independent data or data with weak dependency. Our experience with the systems developed suggests that the bootstrap and data complexity-based techniques are effective for quality and sample size estimation. Finally, our results open doors for many future research directions in the approximation domain.

REFERENCES

[A 07] D.J. Newman A. Asuncion. "UCI Machine Learning Repository.", 2007.

[AAB05] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. "The Design of the Borealis Stream Processing Engine." In *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, January 2005.

[ABA09] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, and Avi Silberschatz. "HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads." *PVLDB*, **2**(1):922–933, 2009.

[ABc05] Yanif Ahmad, Bradley Berg, Ugur Çetintemel, Mark Humphrey, Jeong-Hyon Hwang, Anjali Jhingran, Anurag Maskey, Olga Papaemmanouil, Alex Rasin, Nesime Tatbul, Wenjuan Xing, Ying Xing, and Stanley B. Zdonik. "Distributed operation in the Borealis stream processing engine." In *SIGMOD Conference*, pp. 882–884, 2005.

[aut] "Automaton implementation." `http://www.brics.dk/automaton/`.

[BB07] Léon Bottou and Olivier Bousquet. "The Tradeoffs of Large Scale Learning." In *NIPS*, 2007.

[BEG11] Kevin S. Beyer, Vuk Ercegovac, Rainer Gemulla, Andrey Balmin, Mohamed Y. Eltabakh, Carl-Christian Kanne, Fatma Özcan, and Eugene J. Shekita. "Jaql: A Scripting Language for Large Scale Semistructured Data Analysis." *PVLDB*, 2011.

[BHB10a] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. "HaLoop: Efficient Iterative Data Processing on Large Clusters." *PVLDB*, pp. 285–296, 2010.

[BHB10b] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. "HaLoop: Efficient Iterative Data Processing on Large Clusters." In *36th International Conference on Very Large Data Bases*, pp. 285–296, Singapore, September 14–16, 2010.

[BK99] Peter Bhlmann and Hans R Knsch. "Block length selection in the bootstrap for time series." *Computational Statistics And Data Analysis*, **31**(3):295 – 310, 1999.

[BS96] Christopher J. C. Burges and Bernhard Schölkopf. "Improving the Accuracy and Speed of Support Vector Machines." In *NIPS*, pp. 375–381, 1996.

[Bur98] Christopher J. C. Burges. "A tutorial on support vector machines for pattern recognition." *Data Mining and Knowledge Discovery*, **2**:121–167, 1998.

[bur11] "Burn-in Test.", 2011. `http://www.passmark.com/products/bit.htm`.

[BZ97] P. J. Bickel, F. tze, and W. R. Van Zwet. "Resampling fewer than n observations: gains, losses, and remedies for losses." *STATIST. SINICA*, **7**:1–32, 1997.

[CCA09] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. "MapReduce Online." Technical Report UCB/EECS-2009-136, EECS Department, University of California, Berkeley, 2009.

[CCA10a] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. "MapReduce Online." In *NSDI*, pp. 313–328, 2010.

[CCA10b] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, John Gerth, Justin Talbot, Khaled Elmeleegy, and Russell Sears. "Online aggregation and continuous query support in MapReduce." In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pp. 1115–1118, New York, NY, USA, 2010. ACM.

[CDS04] Surajit Chaudhuri, Gautam Das, and Utkarsh Srivastava. "Effective use of block-level sampling in statistics estimation." In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pp. 287–298, New York, NY, USA, 2004. ACM.

[CGR03] Chee Yong Chan, Minos N. Garofalakis, and Rajeev Rastogi. "RE-tree: an efficient index structure for regular expressions." *VLDB J.*, **12**(2):102–119, 2003.

[CHB04] Nitesh V. Chawla, Lawrence O. Hall, Kevin W. Bowyer, and W. Philip Kegelmeyer. "Learning Ensembles from Bites: A Scalable and Accurate Approach." *J. Mach. Learn. Res.*, **5**:421–451, December 2004.

[CJL08] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. "SCOPE: easy and efficient parallel processing of massive data sets." *PVLDB*, **1**(2):1265–1276, 2008.

[CKL06] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun. "Map-Reduce for Machine Learning on Multicore." In *NIPS*, pp. 281–288, 2006.

[CM07] Olivier Cappé and Eric Moulines. "Online EM Algorithm for Latent Data Models." *CoRR*, **abs/0712.4273**, 2007.

[CP00] Gert Cauwenberghs and Tomaso Poggio. "Incremental and Decremental Support Vector Machine Learning." In *NIPS*, pp. 409–415, 2000.

[CWY05] Yun Chi, Haixun Wang, and Philip S. Yu. "Loadstar: load shedding in data stream mining." In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pp. 1302–1305. VLDB Endowment, 2005.

[DF03] Yanlei Diao and Michael J. Franklin. "High-Performance XML Filtering: An Overview of YFilter." *IEEE Data Engineering Bulletin*, **26**:41–48, 2003.

[DGG86] David J. DeWitt, Robert H. Gerber, Goetz Graefe, Michael L. Heytens, Krishna B. Kumar, and M. Muralikrishna. "GAMMA - A High Performance Dataflow Database Machine." In *Proceedings of the 12th International Conference on Very Large Data Bases*, VLDB '86, pp. 228–237, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.

[DGH06] Alan J. Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker M. White. "Towards Expressive Publish/Subscribe Systems." In *EDBT*, pp. 627–644, 2006.

[DIG07] Yanlei Diao, Neil Immerman, and Daniel Gyllstrom. "Sase+: An agile language for kleene closure over event streams." Technical report, 2007.

[DM01] Russell Davidson and James G. MacKinnon. "Bootstrap Tests: How Many Bootstraps?" Working Papers 1036, Queen's University, Department of Economics, 2001.

[Efr79] B Efron. "Bootstrap Methods: Another Look at the Jackknife." *Annals of Statistics*, **7**(1):1–26, 1979.

[Efr87] Bradley Efron. "Better Bootstrap Confidence Intervals." *Journal of the American Statistical Association*, **82**(397):171–185, 1987.

[ELZ10a] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. "Twister: a runtime for iterative MapReduce." HPDC, pp. 810–818, 2010.

[ELZ10b] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. "Twister: a runtime for iterative MapReduce." In *HPDC*, pp. 810–818, 2010.

[ET97] Bradley Efron and Robert Tibshirani. "Improvements on Cross-Validation: The .632+ Bootstrap Method." *Journal of the American Statistical Association*, **92**(438):548–560, June 1997.

[FHA10] Fatima Farag, Moustafa A. Hammad, and Reda Alhajj. "Adaptive query processing in data stream management systems under limited memory resources." In *PIKM*, pp. 9–16, 2010.

[FPC09] Eric Friedman, Peter M. Pawlowski, and John Cieslewicz. "SQL/MapReduce: A practical approach to self-describing, polymorphic, and parallelizable user-defined functions." *PVLDB*, **2**(2):1402–1413, 2009.

[GAW08] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. "SPADE: the system s declarative stream processing engine." In *SIGMOD Conference*, pp. 1123–1134, 2008.

[GC12] Raman Grover and Michael Carey. "Extending Map-Reduce for Efficient Predicate-Based Sampling." ICDE '12, 2012.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pp. 29–43, New York, NY, USA, 2003. ACM.

[GGM04] Todd J. Green, Ashish Gupta, Gerome Miklau, Makoto Onizuka, and Dan Suciu. "Processing XML streams with deterministic automata and stream indexes." *ACM Trans. Database Syst.*, **29**(4):752–788, 2004.

[GNC09] Alan Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan Narayanam, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava. "Building a HighLevel Dataflow System on top of MapReduce: The Pig Experience." *PVLDB*, **2**(2):1414–1425, 2009.

[had] "Hadoop: Open-source implementation of mapreduce: http://hadoop.apache.org.".

[HB02] Tin Kam Ho and Mitra Basu. "Complexity Measures of Supervised Classification Problems." *IEEE Trans. Pattern Anal. Mach. Intell.*, **24**(3):289–300, 2002.

[HHW97] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. "Online Aggregation." In Joan Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pp. 171–182. ACM Press, 1997.

[hiv] "Hive. http://hive.apache.org.".

[HLL11] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. "Starfish: A Self-tuning System for Big Data Analytics." In *CIDR*, pp. 261–272, 2011.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[Ho04] Tin Kam Ho. "Geometrical Complexity of Classification Problems." *CoRR*, **cs.CV/0402020**, 2004.

[Ho08] Tin Kam Ho. "Data Complexity Analysis: Linkage between Context and Solution in Classification." In *SSPR/SPR*, p. 1, 2008.

[HSK94] David Haussler, H. Sebastian Seung, Michael J. Kearns, and Naftali Tishby. "Rigorous Learning Curve Bounds from Statistical Mechanics." In *COLT'94*, pp. 76–87, 1994.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning, Second Edition: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 2nd ed. 2009. corr. 3rd printing 5th printing. edition, February 2009.

[IBY07] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. "Dryad: distributed data-parallel programs from sequential building blocks." In *EuroSys*, pp. 59–72, 2007.

[IDR07] Zachary G. Ives, Amol Deshpande, and Vijayshankar Raman. "Adaptive query processing: Why, How, When, and What Next?" In *VLDB*, pp. 1426–1427, 2007.

[KB06] Tin Kam Ho and Ester Bernad-Mansilla. "Classifier Domains of Competence in Data Complexity Space." In Mitra Basu and TinKam Ho, editors, *Data Complexity in Pattern Recognition*, Advanced Information and Knowledge Processing, pp. 135–152. Springer London, 2006.

[KCT07] Sailesh Kumar, Balakrishnan Chandrasekaran, Jonathan S. Turner, and George Varghese. "Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia." In *ANCS*, pp. 155–164, 2007.

[KTS11] Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael Jordan. "Bootstrapping Big Data." NIPS, 2011.

[KTS12a] Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael Jordan. "The Big Data Bootstrap." 2012.

[KTS12b] Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael I. Jordan. "A Scalable Bootstrap for Massive Data." Technical report, UC Berkeley, 2012.

[Lah03] S.N. Lahiri. *Resampling methods for dependent data*. Springer series in statistics. Springer-Verlag, 2003.

[LB04] Rui Leite and Pavel Brazdil. "Improving Progressive Sampling via Meta-learning on Learning Curves." In *ECML*, pp. 250–261, 2004.

[LCH00] Zhiyuan Li, Katherine Compton, and Scott Hauck. "onfiguration Caching Techniques for FPGA." IEEE Symposium on FPGAs for Custom Computing Machines, 2000.

[LH07] Yuh-Jye Lee and Su-Yun Huang. "Reduced Support Vector Machines: A Statistical Theory." *Neural Networks, IEEE Transactions on*, **18**(1):1 –13, jan. 2007.

[LIN] "LINQ. http://msdn.microsoft.com/en-us/library/bb397926.aspx.".

[LJL08] Dong Gyu Lee, Young Jin Jung, Young Wook Lee, and Keun Ho Ryu. "Hashed Multiple Lists: A Stream Filter for Processing Continuous Query with Multiple Attributes in Geosensor Networks." pp. 104–109, July 2008.

[LK12] Jimmy Lin and Alek Kolcz. "Large-scale machine learning at twitter." SIGMOD, pp. 793–804, 2012.

[LL09] Stephen M. S. Lee and P. Y. Lai. "Double block bootstrap confidence intervals for dependent data." *Biometrika*, **96**(2):427–443, 2009.

[LLZ12] Nikolay Laptev, Tsai-Ching Lu, and Carlo Zaniolo. "BOOT-TS: Scalable Bootstrap over Massive Time-Series data." In *NIPS*, 2012.

[LMD11] Boduo Li, Edward Mazur, Yanlei Diao, Andrew McGregor, and Prashant J. Shenoy. "A platform for scalable one-pass analytics using MapReduce." In *SIGMOD Conference*, pp. 985–996, 2011.

[lps] "lp_solve, a Mixed Integer Linear Programming (MILP) solver." Website.

[LS12] Eric Laber and Kerby Shedden. "An imputation method for estimating the learning curve in classication problems." *Phys.Rev.*, **1**:4043–4061, 2012.

[LWZ01] Eyal de Lara, Dan S. Wallach, and Willy Zwaenepoel. "Puppeteer: Component-based Adaptation for Mobile Computing." In *USITS*, pp. 159–170, 2001.

[LZ12] Nikolay Laptev and Carlo Zaniolo. "Optimization of Massive Pattern Queries by Dynamic Configuration Morphing." In *ICDE*, pp. 917–928, 2012.

[LZJ05] Bin Liu, Yali Zhu, Mariana Jbantova, Bradley Momberger, and Elke A. Rundensteiner. "A Dynamically Adaptive Distributed System for Processing Complex Continuous Queries." In *VLDB*, pp. 1338–1341, 2005.

[LZL12] Nikolay Laptev, Carlo Zaniolo, Tsai-Ching Lu, and Alexander Shkapsky. "Associate-TS: Scalable Association over Massive Time-Series Data." Technical report, Department of Computer Science, University of California, Los Angeles, 2012.

[LZZ12a] Nikolay Laptev, Kai Zeng, and Carlo Zaniolo. "Early Accurate Results for Advanced Analytics on MapReduce." *PVLDB*, **5**(10):1028–1039, 2012.

[LZZ12b] Nikolay Laptev, Kai Zeng, and Carlo Zaniolo. "Early Accurate Results for Advanced Analytics on MapReduce." *PVLDB*, **5**(10):1028–1039, 2012.

[LZZ13] Nikolay Laptev, Kai Zeng, and Carlo Zaniolo. "Very Fast Estimation for Result and Accuracy of Big Data Analytics: the EARL System." In *ICDE*, 2013.

[mah] "Mahout. http://mahout.apache.org/.".

[MIT] MIT. "DARPA INTRUSION DETECTION EVALUATION.".

[MM12] Gurmeet Singh Manku and Rajeev Motwani. "Approximate Frequency Counts over Data Streams." *PVLDB*, **5**(12):1699, 2012.

[MRV08] Anirban Majumder, Rajeev Rastogi, and Sriram Vanama. "Scalable regular expression matching on data streams." In *SIGMOD Conference*, pp. 161–172, 2008.

[MTH02] Christopher Meek, Bo Thiesson, and David Heckerman. "The learning-curve sampling method applied to model-based clustering." *J. Mach. Learn. Res.*, **2**:397–418, March 2002.

[MTR03] Sayan Mukherjee, Pablo Tamayo, Simon Rogers, Ryan M. Rifkin, Anna Engle, Colin Campbell, Todd R. Golub, and Jill P. Mesirov. "Estimating Dataset Size Requirements for Classifying DNA Microarray Data." *Journal of Computational Biology*, pp. 119–142, 2003.

[MZZ10] Barzan Mozafari, Kai Zeng, and Carlo Zaniolo. "K*SQL: a unifying engine for sequence patterns and XML." In *SIGMOD Conference*, pp. 1143–1146, 2010.

[ND09] Willie Ng and Manoranjan Dash. "Which Is Better for Frequent Pattern Mining: Approximate Counting or Sampling?" DaWaK '09, pp. 151–162. Springer-Verlag, 2009.

[NRN10] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. "S4: Distributed Stream Computing Platform." In *ICDM Workshops*, pp. 170–177, 2010.

[NSN97] Brian Noble, Mahadev Satyanarayanan, Dushyanth Narayanan, J. Eric Tilton, Jason Flinn, and Kevin R. Walker. "Agile Application-Aware Adaptation for Mobility." In *SOSP*, pp. 276–287, 1997.

[OR90] Frank Olken and Doron Rotem. "Random Sampling from Database Files: A Survey." In *SSDBM*, pp. 92–111, 1990.

[ORS08a] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. "Pig latin: a not-so-foreign language for data processing." In *SIGMOD*, pp. 1099–1110. ACM, 2008.

[ORS08b] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. "Pig latin: a not-so-foreign language for data processing." In *SIGMOD Conference*, pp. 1099–1110, 2008.

[PC03] Feng Peng and Sudarshan S. Chawathe. "XPath Queries on Streaming Data." In *SIGMOD Conference*, pp. 431–442, 2003.

[PHB09] Biswanath Panda, Joshua Herbach, Sugato Basu, and Roberto J. Bayardo. "PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce." *PVLDB*, **2**(2):1426–1437, 2009.

[PJO99] Foster Provost, David Jensen, and Tim Oates. "Efficient progressive sampling." KDD '99, pp. 23–32, New York, NY, USA, 1999. ACM.

[PR94] Dimitris N. Politis and Joseph P. Romano. "The Stationary Bootstrap." *Journal of the American Statistical Association*, **89**(428):1303+, December 1994.

[PW04] Dimitris N. Politis and Halbert White. "Automatic Block-Length Selection for the Dependent Bootstrap." *Econometric Reviews*, **23**:53–70, 2004.

[rel] "EARL Release Website: http://yellowstone.cs.ucla.edu/wis/.".

[Rep10] UCI Machine Learning Repository. "http://archive.ics.uci.edu/ml/datasets.html.", December 2010.

[Roe99] Martin Roesch. "Snort: Lightweight Intrusion Detection for Networks." In *LISA*, pp. 229–238, 1999.

[RRF11] David N. Reshef, Yakir A. Reshef, Hilary K. Finucane, Sharon R. Grossman, Gilean McVean, Peter J. Turnbaugh, Eric S. Lander, Michael Mitzenmacher, and Pardis C. Sabeti. "Detecting Novel Associations in Large Data Sets." *Science*, **334**(6062):1518–1524, 2011.

[RT85] Prabhakar Raghavan and Clark D. Thompson. "Randomized Rounding: A Technique for Provably Good Algorithms and Algorithmic Proofs." Technical Report UCB/CSD-85-242, EECS Department, University of California, Berkeley, May 1985.

[Rup01] Stefan Ruping. "Incremental Learning with Support Vector Machines." *Data Mining, IEEE International Conference on*, **0**:641, 2001.

[S4] "S4. http://incubator.apache.org/s4/.".

[SG07] Bianca Schroeder and Garth A. Gibson. "Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you?" In *Proceedings of the 5th USENIX conference on File and Storage Technologies*, Berkeley, CA, USA, 2007. USENIX Association.

[SHB04] Mehul A. Shah, Joseph M. Hellerstein, and Eric A. Brewer. "Highly-Available, Fault-Tolerant, Parallel Dataflows." In *SIGMOD Conference*, pp. 827–838, 2004.

[Spe] Terry Speed.

[SPR91] Stanford University. Dept. of Statistics, D.N. Politis, J.P. Romano, and National Science Foundation (U.S.). *A Circular Block-resampling Procedure for Stationary Data*. 1991.

[SS07] Mark Sanderson and Ian Soboroff. "Problems with Kendall's tau." In *SIGIR*, pp. 839–840, 2007.

[ST95] Jun Shao and D. Tu. *The jackknife and bootstrap*. Springer series in statistics. Springer Verlag, 1995.

[Syr01] Stephen E. Syrjala. "A bootstrap Approach to making sample-size calculations for resource surveys." In *SSC*, pp. 53–60, 2001.

[SZZ01] Reza Sadri, Carlo Zaniolo, Amir M. Zarkesh, and Jafar Adibi. "Optimization of Sequence Queries in Database Systems." In *PODS*, 2001.

[TcZ03] Nesime Tatbul, Uğur Çetintemel, Stan Zdonik, Mitch Cherniack, and Michael Stonebraker. "Load shedding in a data stream manager." In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pp. 309–320. VLDB Endowment, 2003.

[ter85] *teradata Corp. Database Computer System Manual*, Feb 1985.

[Tho00] Alun Thomas. "Bootstrapping, jackknifing and cross validation, reusing your data." Utah University Lecture, 2000.

[TLM11] Hetal Thakkar, Nikolay Laptev, Hamid Mousavi, Barzan Mozafari, Vincenzo Russo, and Carlo Zaniolo. "SMM: a Data Stream Management System for Knowledge Discovery." In *ICDE*, 2011.

[TSJ09] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. "Hive - A Warehousing Solution Over a Map-Reduce Framework." *PVLDB*, **2**(2):1626–1629, 2009.

[TSJ10] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Anthony, Hao Liu, and Raghotham Murthy. "Hive - a petabyte scale data warehouse using Hadoop." In *ICDE*, pp. 996–1005, 2010.

[TZ06] Nesime Tatbul and Stan Zdonik. "Window-aware load shedding for aggregation queries over data streams." In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pp. 799–810. VLDB Endowment, 2006.

[Vap99] Vladimir Vapnik. "An overview of statistical learning theory." *IEEE Transactions on Neural Networks*, **10**(5):988–999, 1999.

[VC71] V N Vapnik and A Y Chervonenkis. "On the uniform convergence of relative frequencies of events to their probabilities." *Theory of Probability and Its Applications*, **16**(2):264–280, 1971.

[WDR06] Eugene Wu, Yanlei Diao, and Shariq Rizvi. "High-performance complex event processing over streams." In *SIGMOD Conference*, pp. 407–418, 2006.

[WNC05] Jigang Wang, Predrag Neskovic, and LeonN. Cooper. "Training Data Selection for Support Vector Machines." In Lipo Wang, Ke Chen, and Yew-Soon Ong, editors, *Advances in Natural Computation*, volume 3610 of *Lecture Notes in Computer Science*, pp. 554–564. Springer Berlin Heidelberg, 2005.

[XJS10] Yang Xu, Junchen Jiang, Yang Song, Tang Jiang, and H. Jonothan Chao. "i-Dfa: A novel deterministic FInite Automaton without state explosion." Technical report, Polytechnic Institute of New York University, Brooklyn, NY, 2010.

[YCD06] Fang Yu, Zhifeng Chen, Yanlei Diao, T. V. Lakshman, and Randy H. Katz. "Fast and memory-efficient regular expression matching for deep packet inspection." In *ANCS*, pp. 93–102, 2006.

[YIF08] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. "DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language." In *OSDI*, pp. 1–14, 2008.

[ZGG11] Yanfeng Zhang, Qixin Gao, Lixin Gao, and Cuirong Wang. "PrIter: A Distributed Framework for Prioritized Iterative Computations." In *SOCC*, 2011.

[ZMH09a] Weizhong Zhao, Huifang Ma, and Qing He. "Parallel K-Means Clustering Based on MapReduce." In *Proceedings of the 1st International Conference on Cloud Computing*, pp. 674–679. Springer-Verlag, 2009.

[ZMH09b] Weizhong Zhao, Huifang Ma, and Qing He. "Parallel K-Means Clustering Based on MapReduce." *Cloud Computing*, **5931**:674–679, 2009.