

University of Nebraska - Lincoln

DigitalCommons@University of Nebraska - Lincoln

Computer Science and Engineering: Theses,
Dissertations, and Student Research

Computer Science and Engineering, Department of

8-2010

Managing Large Data Sets Using Support Vector Machines

Ranjini Srinivas

University of Nebraska at Lincoln, ranjinis@cse.unl.edu

Follow this and additional works at: <http://digitalcommons.unl.edu/computerscidiss>



Part of the [Computer Engineering Commons](#), and the [Computer Sciences Commons](#)

Srinivas, Ranjini, "Managing Large Data Sets Using Support Vector Machines" (2010). *Computer Science and Engineering: Theses, Dissertations, and Student Research*. 7.

<http://digitalcommons.unl.edu/computerscidiss/7>

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in Computer Science and Engineering: Theses, Dissertations, and Student Research by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

MANAGING LARGE DATA SETS USING SUPPORT VECTOR MACHINES

by

Ranjini Srinivas

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor David R. Swanson

Lincoln, Nebraska

August, 2010

MANAGING LARGE DATA SETS USING SUPPORT VECTOR MACHINES

Ranjini Srinivas, M.S.

University of Nebraska, 2010

Adviser: David R. Swanson

Hundreds of Terabytes of CMS (Compact Muon Solenoid) data are being accumulated for storage day by day at the University of Nebraska-Lincoln, which is one of the eight US CMS Tier-2 sites. Managing this data includes retaining useful CMS data sets and clearing storage space for newly arriving data by deleting less useful data sets. This is an important task that is currently being done manually and it requires a large amount of time. The overall objective of this study was to develop a methodology to help identify the data sets to be deleted when there is a requirement for storage space. CMS data is stored using HDFS (Hadoop Distributed File System). HDFS logs give information regarding file access operations. Hadoop MapReduce was used to feed information in these logs to Support Vector Machines (SVMs), a machine learning algorithm applicable to classification and regression which is used in this Thesis to develop a classifier. Time elapsed in data set classification by this method is dependent on the size of the input HDFS log file since the algorithmic complexities of Hadoop MapReduce algorithms here are $O(n)$. The SVM methodology produces a list of data sets for deletion along with their respective sizes. This methodology was also compared with a heuristic called Retention Cost R_C which was calculated using size of the data set and the time since its last access to help decide how useful a data set is. Accuracies of both were compared by calculating the percentage of data sets predicted for deletion which were accessed at a later instance of time. Our methodology using SVMs proved to be more accurate than using the Retention Cost

heuristic. This methodology could be used to solve similar problems involving other large data sets.

DEDICATION

For Lord Venkateshwara

ACKNOWLEDGMENTS

I would like to thank my advisor and committee chair, Dr. David R. Swanson for his guidance and encouragement throughout the course of my Master's program. Besides his expertise, I really appreciate his patience and thank him for his editorial advice which was essential for the completion of this thesis. I would also like to thank Dr. Lisong Xu and Dr. Byrav Ramamurthy for their advice and serving on my examining committee.

I thank the members of the Holland Computing Center for their time and research resources. I would like to thank all my friends in the Computer Science department for their support and encouragement throughout my stay at UNL.

Finally, I would like to thank my parents, who so enthusiastically supported and encouraged me to make it through my Master's program.

Contents

Contents	vi
List of Figures	ix
List of Tables	xii
1 Introduction	1
2 Background	5
2.1 Related Work	5
2.1.1 Hierarchical Storage Management	6
2.1.2 Caching Algorithms	7
2.1.3 Clustered File Systems	9
2.1.4 Prominent Users of Hadoop	13
2.2 HDFS logs used in this study	15
2.3 File Classification Methods via Support Vector Machines	17
2.4 Hadoop MapReduce for feature extraction	24
2.4.1 Hadoop MapReduce	24
2.4.1.1 Map Function	25
2.4.1.2 Reduce Function	26

2.4.2	Hadoop Streaming	26
2.4.3	Hadoop Distributed File System: HDFS	27
2.4.4	Hadoop On Demand (HOD)	29
2.5	Python	32
2.6	Weka: Machine Learning Software	32
3	Materials and Methods	34
3.1	Data Collection	34
3.1.1	Data Source	34
3.1.2	Data Pre-processing	34
3.1.2.1	Data Cleaning	35
3.1.2.2	Discretization	35
3.1.3	Training Data	35
3.1.4	Test Data	42
3.1.4.1	Testing against training data belonging to the same time frame	42
3.1.4.2	Testing against training data belonging to a different time frame	42
3.1.5	Running MapReduce using Hadoop On Demand (HOD)	43
3.1.6	Creating an ARFF file for WEKA	44
3.2	File Classification Method - Support Vector Machines	46
3.3	Performance Analysis	48
3.3.1	Cross-Validation	48
3.3.2	Confusion Matrix and Accuracy Rate	49
3.3.3	Receiver Operating Characteristic	52
3.4	CMS Data Sets to be Deleted	53

3.5	Retention Cost Heuristic	54
4	Results and Discussion	55
4.1	Identification of Data sets to be deleted	55
4.1.1	Accuracy rates	55
4.1.2	Confusion Matrix Analysis	59
4.1.3	Receiver Operating Characteristic (ROC) Analysis	59
4.2	Testing	60
4.2.1	Unit Testing - Testing Python Scripts	60
4.2.2	Performance and Scalability Testing	61
4.2.3	Integration Testing	61
4.3	Tool Usage and Result Interpretation	62
4.3.1	Setting up the Environment for Tool Deployment	63
4.4	Comparing SVM based Methodology with Retention Cost Heuristic	65
4.5	Classifier A's accuracy against test set J	68
5	Conclusions and Future Work	69
5.1	Conclusion	69
5.2	Future Work	70
	Bibliography	71

List of Figures

1.1	Flowchart depicting how Large Data Sets are managed using Support Vector Machines in this study.	3
2.1	CMS Computing Distributed Hierarchy.	15
2.2	In this figure, a hyperplane classifies two classes of training data which is two dimensional. The two classes of data are represented by squares and circles. The bold line, which is separated from the closest training vectors by distance γ is the hyperplane. The classification of triangle, which is an unknown sample, is done by determining which side of the hyperplane it falls. In this example, the prediction for the unknown sample would be square.	18
2.3	In this figure, a hyperplane classifies two classes of training data which is two dimensional by allowing error ξ_i in classification. The two classes of data are represented by squares and circles. The bold line, which is separated from the closest training vectors by distance γ is the hyperplane.	20
2.4	The figure shows the mapping $\Phi(\cdot)$ of attributes of the input space to the feature space by a Kernel Function.	21
2.5	Map Function creates a new output list i.e (key, value) pairs from an input list. Courtesy [1]	25

2.6	Reduce Function processes the output from the Map Function. It sorts the (key, value) pairs and groups them by the key values [2]. Courtesy [1]	26
2.7	HDFS Architecture. Courtesy [1]	28
2.8	Allocation of a virtual Hadoop cluster and running jobs on it.	31
3.1	The graph shows the pattern of file access requests received from some IPs.	39
3.2	Figure shows a script used for submitting Hadoop Streaming jobs using SGE on Prairiefire.	43
3.3	ARFF file for the log data.	45
3.4	A basic ROC graph showing five discrete classifiers. A discrete classifier is one that outputs only a class label. Each discrete classifier produces an (FP rate,TP rate) pair, which corresponds to a single point in ROC space. (courtesy [3]).	52
4.1	Accuracy rates of cross-validation test of all the nine SVM classifiers trained on different data sets.	56
4.2	Accuracy rates of all the nine SVM classifiers tested against different test data sets.	58
4.3	Accuracy rates of top two classifiers “A” and “B” tested against different test data sets.	58
4.4	ROC of all the nine SVM classifiers.	60
4.5	This figure shows the performance of the MapReduce programs used in this study and also shows that they are scalable	61
4.6	Flowchart depicting how Large Data Sets are managed using Support Vector Machines in this study.	62
4.7	Contents of Input HDFS Log File.	64
4.8	Contents of Input Data set Dump.	64

4.9	Contents of Final Deletion Report.	65
4.10	Performance of different classifiers against test data J	68

List of Tables

2.1	An extended taxonomy of some existing and proposed caching algorithms.	10
3.1	Percentage of files being accessed by a single IP.	39
3.2	The training sets used in this study. A, B and C belong to February 2009; D, E and F belong to March 2009; G, H and I belong to April 2009. . . .	40
3.3	Confusion Matrix	49
4.1	Cross Validation Accuracy rates of the nine classifiers	56
4.2	Accuracy rates of the top two classifiers against real data.	57
4.3	Confusion Matrix of Classifier A	59
4.4	Performance Metrics calculated using Confusion Matrix as discussed in 3.3.2.	59
4.5	Retention Cost Metric calculated for some randomly picked data sets. . .	66
4.6	Comparing SVM methodology with Retention Cost Heuristic.	67

List of Algorithms

1	SVM Algorithm	19
2	SVM Algorithm with a Kernel Function $K()$	22
3	To create Delete-List - Mapper	37
4	To create Save-List - Mapper	37
5	To compute the number of times a file was accessed - Mapper	38
6	To compute the number of times a file was accessed - Reducer	38
7	To extract the IP from which a file was accessed - Mapper	40
8	To extract the IP from which a file was accessed - Reducer	40
9	To calculate the age of a file - Mapper	41
10	To calculate the age of a file - Reducer	41
11	To create ARFF file for the log data	46

Chapter 1

Introduction

This thesis proposes and demonstrates a methodology that could be used to solve management problems involving large data sets using Support Vector Machines. Nebraska, being a US CMS Tier-2 site [4], stores terabytes of data and is a good candidate for the “Big Data” problem. Disk space is required for new data sets constantly arriving from Fermilab and for older data sets frequently used by physicists for their analysis [4]. This manageability issue is addressed in this thesis. We present a policy which aides in cleaning up a specified amount of disk space intelligently and conveniently in a very short time. All this data is being archived at FermiLab, Chicago and can be retrieved at a later point of time, if need be.

CMS Physics data consists of data sets in turn comprised of a number of files. Metadata maintained by the Dataset Bookkeeping Service (DBS) contains the mapping between these files and the data sets they belong to. In order to identify the data sets to be deleted, we first identify the files that could be deleted. The more files belonging to a data set that are identified as less useful, the better are the chances of it to be considered for deletion. Currently, files and data sets are being deleted manually. This approach is very time consuming and is quite error prone. It is hard

to identify and consider many contributing factors which are important in making a decision whether a file or a data set needs to be deleted. Since the user community is big and consists of multiple researchers there exists competing interests with regard to retentivity of data sets. There is absence of a clear criteria. This makes it imperative to find a different methodology for cleaning the disk space other than just using the latest access time or random deletion. The methodology developed here studies the pattern of deletion which has occurred in the past. A machine learning algorithm is used to learn this pattern and helps classify the data sets by classifying the files belonging to these data sets. The tool which is presented here not only automates this process using this methodology but also is more accurate and objective in predicting a list of candidate data sets to be deleted.

Apache Hadoop is a powerful tool used in the analysis of Big Data [5]. Hadoop provides the power of scanning and analyzing terabytes of log data using a collection of commodity servers working in parallel. SVM is a Kernel based technique used for classifying files into those which need to be retained and those which could be deleted [6]. The MapReduce programming model is used in conjunction with SVM (Support Vector Machines) to arrive at a list of files or data sets to be deleted.

Nine different training sets (A to I), three from each month of February, March and April of the year 2009 were created from the HDFS logs. There was also a training set created by randomly sampling all these nine training sets. Classifiers were developed from each of these training sets. These classifiers were tested against test sets belonging to the same time frame and against test sets belonging to different time frames, since the objective was to choose the classifier that could be useful in classifying files which are created and accessed in the future months after the classifier was created. Their classification performance was analyzed using various statistics like accuracy rates and receiver operating characteristic graphs as discussed

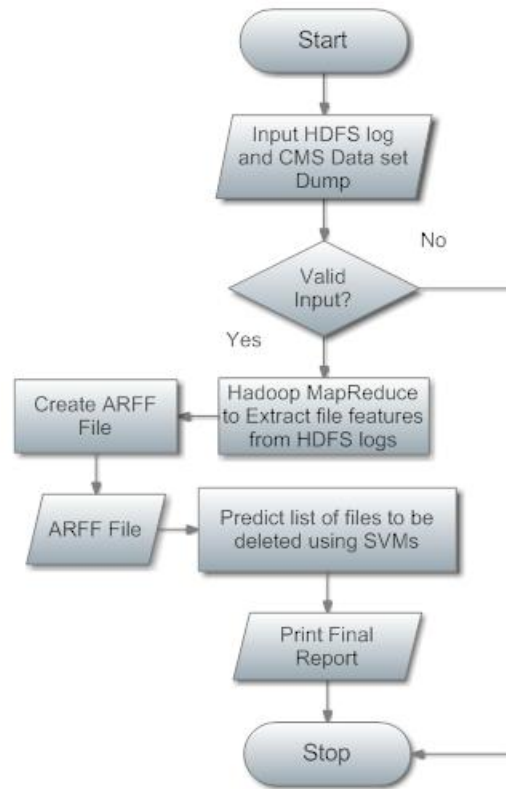


Figure 1.1: Flowchart depicting how Large Data Sets are managed using Support Vector Machines in this study.

in Chapter 4. The classifier with best classification performance was chosen. Weka, an open source machine learning suite developed at the University of Waikato in New Zealand was used in this study. It is a suite with an interface to various machine learning algorithms. For training a support vector classifier, it implements John Platt's sequential minimal optimization (SMO) algorithm [7]. Figure 1.1 shows the way in which this classifier is used to classify data sets.

A heuristic, namely Retention Cost was also used to classify the data sets. Here, data sets were predicted for deletion based on their Retention Cost values and the prediction accuracy was compared to that of the SVM based methodology developed in this thesis. The SVM based methodology outperformed the Retention Cost heuristic

and thus could be used to solve similar problems involving large data sets.

The remainder of this thesis is organized as follows. Related work and backgrounds of Hadoop MapReduce Framework, Support Vector Machines and Weka, a Data Mining Software in Java have been discussed in Chapter 2. Chapter 3 explains data pre-processing and data collection methods, using SVMs for developing a classifier, and performance analysis used in this study. It also discusses computation of Retention Cost heuristic. Results and discussions are given in Chapter 4. Finally, Chapter 5 concludes this thesis with overall discussion and future work.

Chapter 2

Background

2.1 Related Work

Various approaches to managing large data sets in different areas of research were studied. A method of functionally classifying genes by using gene expression data using SVMs to exploit prior knowledge of gene function to identify unknown genes of similar function from expression data was introduced in [8]. In [9] an SVM based method was used for predicting the propensity of a protein to be soluble or to form an inclusion body on overexpression in *Escherichia coli*. [10] proposed an SVM approach to stock trading in which an SVM classifier was successfully applied to forecast relative performance within the oil sector. [11] proposed Combining Tag Cloud Learning with SVM classification to achieve Intelligent Search for Relevant Blog Articles. [12] explored the possibility of analyzing Massive Astrophysical Data sets using Hadoop. The applicability of MapReduce to spatial data processing workloads was studied in [13] where the excellent scalability of MapReduce was confirmed in that domain. [14] presents an efficient MapReduce algorithm for analyzing large collections of evolutionary trees this year in 2010. They conclude that MapReduce is a promising

paradigm for developing multi-core phylogenetic applications. Jeff Hammerbacher from Cloudera presented Analyzing petabytes of data with Hadoop at CERN Computing Colloquium in August, 2009 and claimed HDFS was storing over 700 TB of High Energy Physics (HEP) data. [15] concludes that MapReduce is a good choice for basic operations on large data sets for Machine Learning. In this study, we combine Hadoop MapReduce with SVMs to develop a methodology for managing large data sets.

We also discuss the literature on Hierarchical Storage Management, Caching Algorithms and Clustered File Systems in the following sub-sections in this chapter.

2.1.1 Hierarchical Storage Management

CMS Computing Distributed Hierarchy discussed in Section 2.2 is analogous to Hierarchical Storage Management (HSM). HSM supports backing up data to economical storage devices like magnetic tape drives and optical discs. Depending on the usage pattern, data is stored and migrated on different types of storage media. Storage media is categorized based on cost and speed of retrieval when access is required. The usage pattern is monitored to make a decision as to which data can safely be moved to slower devices and which data should stay on the fast devices. For example, when a file is being accessed very frequently it is stored on a high-speed storage device like hard disk drive arrays, which is of high-cost. When the same file is accessed infrequently it is migrated to a slower but less expensive form of storage like magnetic tape. It is transparent to the user when a file is being retrieved from backup storage media; this file is then moved back to a disk drive. HSM is thus a policy based management for archiving data by using storage devices economically. The first implementation of HSM was by IBM on their mainframe computers [16].

2.1.2 Caching Algorithms

The Retention Cost heuristic is in some sense a caching algorithm based on time and size criteria for deleting data sets. Caching algorithm aim to keep information that will be used in the future and to discard information that will not be used for the longest time in future. Some of the caching algorithms are discussed here.

- Least Frequently Used (LFU): In the Least Frequently Used (LFU) algorithm the cache keeps the files which are really needed and discards the ones that aren't needed for the longest period based on the access pattern. A counter is maintained to count how often a file is accessed i.e. the counter corresponding to a file gets incremented each time it is accessed. The file with least counter value is discarded first [17][18].
- Least Recently Used (LRU): The Least Recently Used algorithm removes the file that was referenced the longest time ago with the underlying assumption that the files that have been used recently will need to be used again soon [18][19].
- Least Recently/Frequently Used (LRFU): [18] shows that there exists a spectrum of policies that subsumes the well-known LRU and LFU policies in the form of the LRFU. LRFU is formed by how much more weight we give to the recent history than to the older history.
- LFU with Dynamic Aging (LFUDA): LFUDA is a variant of LFU. It is based on LFU-Aging policy that considers both the access frequency of a file and its age. A dynamic aging policy is used which adds the cache age factor to the reference count when a new object is added to the cache or when an existing object is referenced again. When a file is discarded its key value gets assigned

to the cache age instead of adjusting all key values in the cache. This ensures that the cache age is less than or equal to the minimum key value in the cache [20][21].

- Most Frequently Used (MFU): The MFU algorithm also known as Fetch-and-Discard replacement algorithm, in contrast to LFU discards objects with highest frequency with the assumption that an object with a large frequency counter has had its fair share of disk space and it's time for some other object to get its turn. This algorithm is used to deal with the case such as sequential scanning access pattern, where most of recently accessed objects are not reused in the near future [22][23].
- Most Recently Used (MRU): In contrast to LRU, the MRU algorithm discards the most recently used objects first. This caching mechanism is used when access is unpredictable and determining the least most recently used section of the cache system is a high time complexity operation [24][25].
- First In First Out (FIFO):: The FIFO algorithm has a list which contains the file that arrived first at the front of the list and the rear of the list has the most recent arrival. This algorithm assumes that older files are not used often and when there is requirement for disk space it discards the file at the front of the list [26].
- Greedy-Dual-Size (GDS): The GDS algorithm considers the size and cost information. It assigns a cost/size value to each object. Costs such as latency and network bandwidth can be explored although the cost is set to 1 in the simplest case. GDS assigns a key value to each object which is computed by summing the object reference count, the cost information divided by its size. Thus, recency

for an object is taken into account by inflating a key value (cost/size value) for an accessed object by the least value of currently cached objects [27][28].

- **GDS with Frequency (GDFS):** GDFS is a variant of the Greedy Dual-Size policy, it also takes into account frequency of reference. This algorithm is optimized for more popular, smaller objects in order to maximize object hit rate. A key is assigned to each object computed as the object's reference count divided by its size, plus the cache age factor [20].
- **Lowest Relative Value (LRV):** The LRV algorithm computes the lowest relative value for each object using access time, frequency and size information. The object with the lowest relative value is discarded [29][30].
- **Stor-serv:** [31] proposes the concept of Quality of Service (QoS) ideas used in networking to be applied to storage systems for giving differentiated services to users.

Table 2.1 lists some caching algorithms along with their criteria to make replacement decisions. Time, frequency and size are the most commonly used criteria. Algorithms like FIFO and LIFO do not require any such information to make replacement decisions. There are different possible criteria and the trend in cache replacement algorithms is towards finding functions that integrate most of it to a single value [32].

2.1.3 Clustered File Systems

Clustered File Systems are mounted simultaneously on multiple servers. The cluster is transparent and it appears as just a file system to the clients. We now discuss some

Table 2.1: An extended taxonomy of some existing and proposed caching algorithms.

Criteria	Algorithm
-	Random, FIFO, LIFO
Time	LRU, MRU, LRFU, GDS, GDSF, LFUDA, LRV
Frequency	LFU, MFU, LRFU, GDSF, LRV, LFUDA
Size	SIZE, GDS, GDSF, LRV
Retrieval Cost	GDS, GDSF, LFUDA, LRV
QoS Priority	Stor-serv

of the clustered file systems which are closely related to the Hadoop Distributed File System (HDFS).

- **Lustre:** Lustre is a distributed file system with a “Shared Nothing” architecture. Developed as a research project at Carnegie Mellon University it was later acquired by Sun Microsystems, which in turn was acquired by Oracle Corporation. A Lustre file system consists of a metadata server (MDS) which contains a metadata target (MDT) per filesystem to store metadata such as filename, directories, permissions and file layout. Data is stored on object storage targets (OSTs) on object storage servers (OSSes). Clients are provided with read and write access through the standard POSIX interface [33].
- **dCache:** dCache is a disk-caching system jointly developed by DESY (Deutsches Elektronen Synchrotron) and FNAL (Fermi National Accelerator Laboratory). It consists of an Admin node to run the administrative services and modules to manage pools of storage. There could be one or more pool nodes with pools. Door nodes provide I/O access through SRM or GridFTP to the stores. Door services are installed on the single Admin node. A pnfs node is used to map logical names to physical locations. By providing a single name space across the entire pool of disk servers it makes it look like a single, giant disk system to the users [34].

- **IBM General Parallel File System (GPFS):** The General Parallel File System (GPFS) is a high-performance shared-disk clustered file system developed by IBM. Individual files are striped into blocks and stored across multiple disks. This provides higher input/output performance since the reading and writing of these blocks happens in parallel. Data loss is prevented by having multiple copies of each block written to the physical disks on the individual nodes using RAID controllers. The metadata including the directory tree is distributed and thus avoids single point failure [35]. It is also possible to have just two copies of all files and completely opt out of RAID-replicated blocks [36].
- **Google File System (GFS):** Google File System (GFS) is a distributed file system where clusters are built from commodity hardware developed by Google Inc. A GFS cluster consists of a single master and multiple chunk servers and is accessed by multiple clients. Files are divided into chunks of fixed size. The master assigns each of these chunks with a unique chunk handle. Data loss is prevented by replicating chunks on multiple chunkservers. These chunkservers read and write data using the chunk handles. Metadata of the file system which includes the namespace, access control information, the mapping from files to chunks, and current locations of chunks is maintained by the master. Interaction between the client and the master consists only of metadata operations, all data operations happen at the chunkservers [37].
- **Hadoop Distributed File System (HDFS):** HDFS is highly influenced by GFS. It is a framework written in Java which supports data-intensive distributed applications and is open source. It contains a master server known as the name node which maintains the filesystem namespace and the metadata of all the files and directories present in the filesystem tree. File access by the clients

are regulated by the name node. Files are striped into blocks of 64MB by default and these blocks are stored on worker nodes known as data nodes. For reliability, each block is replicated and stored on multiple data nodes. The data nodes receive instructions from the namenode for block creation, deletion and replication. They manage the storage attached to the nodes on which they run and are responsible for storing and retrieving blocks as requested by the clients and/or namenode. The name node has the information of how many HDFS blocks a file consists of and the data nodes on which these blocks are stored. File access by the clients are regulated by the name node [5]. HDFS is discussed more in detail in Section 2.4.3.

Clustered file systems like IBM GPFS are available only on specific platforms on which the vendor has implemented them [38]. There are a lot of similarities between GPFS and HDFS and it is interesting to discuss their similarities and dissimilarities. Both GPFS and HDFS are designed to store very large quantities of data on datacenters built from commodity hardware. GPFS is commercial whereas HDFS is open source. Files are split into blocks and stored on different nodes under both these file systems. The block-size in HDFS is 64MB or more whereas the block-sizes in GPFS are under 1MB and thus are much smaller. Having many blocks of files fills up the filesystem's indices very quickly. HDFS has lesser numbers of blocks due to larger block-size, this reduces the storage requirement of the Namenode. HDFS does not expect reliable disks, so instead stores copies of the blocks on different nodes. The failure of a node containing a single copy of a block is a minor issue, dealt with by re-replicating another copy of the set of valid blocks, to bring the replication count back up to the desired number. In contrast, while GPFS supports recovery from a lost node, it is a more serious event, one that may include a higher risk of data be-

ing (temporarily) lost. Namenode failure in HDFS becomes a single point of failure, but in GPFS single point of failure is avoided by having distributed metadata. In HDFS the MapReduce programming paradigm is supported and the location of data is made known to MapReduce programs for them to run near the data. Massively parallel programs are written in the MapReduce style. GPFS makes location of the data transparent since the applications do not need this information.

2.1.4 Prominent Users of Hadoop

There are many organizations and applications that use Hadoop extensively. Here is an incomplete list of prominent users of Hadoop [39]:

1. Yahoo: The Yahoo Search Webmap is a Hadoop application that runs on a more than 10,000 core Linux cluster and produces data that is now used in every Yahoo! Web search query. There are multiple Hadoop clusters at Yahoo, each occupying a single datacenter or fraction thereof. No HDFS filesystems or MapReduce jobs are split across multiple datacenters; instead each datacenter has a separate filesystem and workload. Yahoo has the largest installation of Hadoop.
2. Facebook: Facebook has the second largest installation of Hadoop. Hadoop is used to store copies of internal log and dimension data sources and is used as a source for reporting, analytics and machine learning.
3. Amazon Web Services: They provide a web service called Amazon Elastic MapReduce which hosts Hadoop framework running on the web-scale infrastructure of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3) for their customers to perform data-intensive tasks

for applications such as web indexing, data mining, log file analysis, machine learning, financial analysis, scientific simulation and bioinformatics research.

4. Cloudera, Inc: Cloudera provides Hadoop professional training and support for Hadoop. They also provide Cloudera's Distribution for Hadoop to make it easy to set up, configure and use Hadoop [40].
5. LinkedIn: Hadoop is used for discovering People You May Know and other fun facts [41].
6. Twitter: Twitter uses Hadoop for ad hoc analysis of data collected from its famous microblogging service, but the platform also crunches data for use by live tools on the site, including Twitter's name-search function.

Areas of Applications of Hadoop: [40] discusses how Hadoop complements existing data management solutions with new analyses and processing tools. It provides a platform for consolidating data from various sources and delivers immediate value to a variety of vertical markets like:

1. Health and Life Sciences: Hadoop is used in drug discovery and development analysis. It also facilitates patient care quality and program analysis.
2. Telecommunications: Network performance and optimization is computed using Hadoop. Call Detail Record (CDR) analysis can also be done.
3. Financial Services: Risk analysis, fraud detection, security analytics, CRM, credit scoring and customer loyalty are also being determined using Hadoop.
4. Government: Hadoop can be used in determining energy consumption, carbon footprint management, fraud detection and cybersecurity. It can also be used for compliance and regulatory analysis.

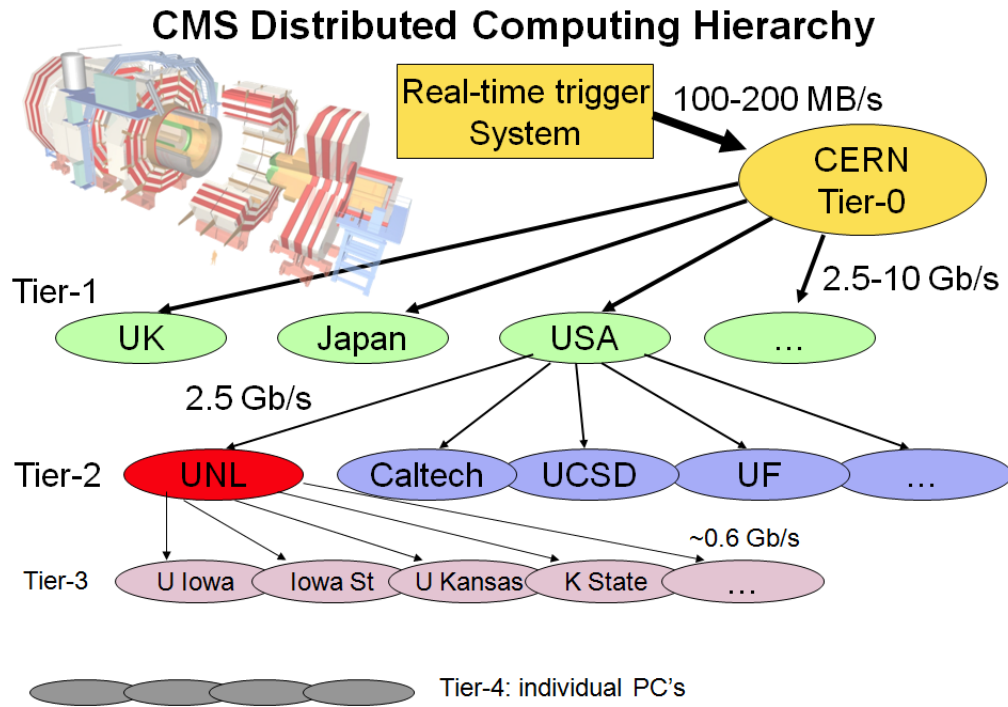


Figure 2.1: CMS Computing Distributed Hierarchy.

5. Web and Digital Media Services: Ad targeting, analysis, forecasting and optimization could be done using Hadoop.

2.2 HDFS logs used in this study

Nebraska is one of the eight US CMS (Compact Muon Solenoid) Tier-2 sites. CMS is a High-Energy Physics experiment, the results of which are being stored using Hadoop Distributed File System (HDFS) at the University of Nebraska-Lincoln. Figure 2.1 shows the highly distributed computing system being used for processing and analyzing CMS data. The data goes through the following transformations before it reaches a physicist sitting in Nebraska. This holds for the other US CMS Tier-2 sites as well [4]:

- Raw data is processed at the Tier-0 center at CERN - the world's largest particle physics laboratory in Geneva, Switzerland. Firstly, raw data is transformed into reconstructed data.
- Reconstructed data is then transformed to analysis-oriented data (AOD). Computational limitations at CERN make all the necessary transformations impossible at CERN and resort to seven Tier-1 centers worldwide for further transformations.
- Fermilab is the US Tier-1 center; it transforms the AOD data to physicist-specific ntuples. Several copies of AOD data is maintained at Fermilab which undergoes processing every time improved calibrations and algorithms are available. All the available resources are used for this task and there are none available for user analysis tasks, due to which the data is transferred to about thirty five CMS Tier-2 sites, Nebraska being one among eight in the US [42].
- The skimmed sub-samples of data from Fermilab reach Nebraska which provides support for the analysis work of physicists and the required simulation capacity. Along with data transferred from Fermilab it houses user generated data and disk space is made available for temporary caching of samples being simulated by physicists.

Nebraska has made a cautious decision of choosing Hadoop for storing the CMS data. This thesis includes the development of a tool which analyzes gigabytes of Hadoop logs using MapReduce and SVM, is a product of investigation of how to manage data sets of very large scale.

2.3 File Classification Methods via Support Vector Machines

Machine learning is concerned with the development of algorithms which enable the machine to learn and perform tasks. Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression [43]. SVM is a classifier derived from statistical learning theory by Vapnik and Chervonenkis [44]. It belongs to a family of generalized linear classifiers. It is a classification technique that seeks to find a hyperplane that partitions the data by their class labels and at the same time avoid over-fitting the data by maximizing the margin of the separating hyperplane. It makes a binary classification based on a separating hyperplane on a remapped instance space. Thus the goal of the classification is to remap the input vectors onto a multi-dimensional space so that the instances are linearly separable.

SVMs learn from labeled examples from a training set including both positive and negative samples. A hyperplane which classifies the positive and negative data in the training set is found based on the attributes of data in the training set. Many such hyperplanes which separate the data exist. The one that achieves maximum separation needs to be chosen and only one such hyperplane exists [45]. Such a hyperplane achieves good separation since it has the largest distance to the nearest training data points of any class [43].

The data points nearest to the margin on both sides are called support vectors. From the data and its labels the SVMs learn a mapping function. A kernel function, which is a dot product, is used in finding the hyperplane by remapping input feature vectors. Once the hyperplane is found, unlabeled examples from the test are classified based on the support vectors.

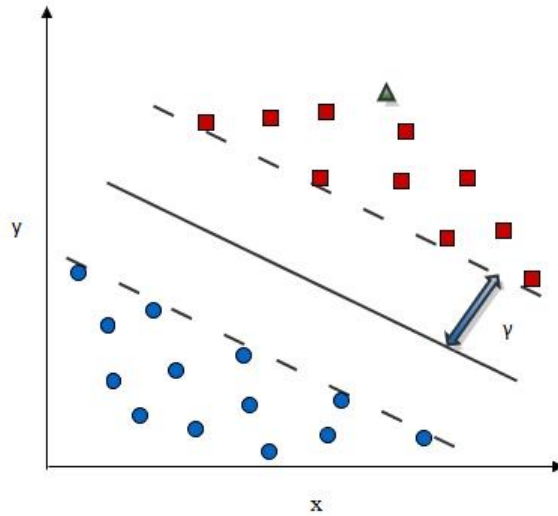


Figure 2.2: In this figure, a hyperplane classifies two classes of training data which is two dimensional. The two classes of data are represented by squares and circles. The bold line, which is separated from the closest training vectors by distance γ is the hyperplane. The classification of triangle, which is an unknown sample, is done by determining which side of the hyperplane it falls. In this example, the prediction for the unknown sample would be square.

Formalization: We represent each sequence by a feature vector which is nothing but a collection of the attributes in vector format. The goal is to correctly classify data in calculating the SVM. The decision boundary must classify all points correctly and should prevent data points from falling into the margin. Let n be the dimension of the feature vector and $\{x_1, \dots, x_n\}$ be the representation of a sequence x and let $y_x \in \{1, -1\}$ be the class label of x . The feature vectors of the training set are used to build the classifier. Weight vector w is of the same dimension as the feature vector and it is represented as $w = \{w_1, \dots, w_n\}$. The label of the sequence is predicted as 1 if $wx + b \geq 1$ and it is predicted as -1 if $wx + b \leq -1$ where, b is a threshold [4]. Therefore,

i) if $yx = 1; wx + b \geq 1$

ii) if $yx = -1; wx + b \leq 1$

The equation of the margin γ_x is given by

$$\gamma_x = y_x(wx + b)$$

The value of γ determines the accuracy of SVMs in predicting the label of the sequence. If γ is positive, it implies that the prediction is correct and a negative value of γ implies that the prediction is incorrect. The values of the weight vector w and threshold b are updated every time there has been an incorrect prediction. We now present a simple algorithm of an SVM [46]. In this algorithm, the weight vector w

Algorithm 1 SVM Algorithm

Require: $w_0 \Leftarrow 0, b_0 \Leftarrow 0, k \Leftarrow 0$

Require: $R \Leftarrow$ Radius of the vector most distant from the origin of vector space

while mistakes are made on training set **do**

for $i = 1$ to N **do**

if $y_i(w_k + b) \leq 0$ **then**

$$w_{k+1} \Leftarrow w_k + \eta y_i x_i$$

$$b_{k+1} \Leftarrow b_k + \eta y_i R^2$$

end if

end for

end while

and the threshold b are initialized to 0. k keeps count of the number of incorrect predictions which is also initialized to 0. R is the radius of the hypersphere and is initialized to the maximum distance between the origin of the hypersphere and a training vector. N is the total number of training vectors. η is the rate of learning. The decision hyperplane is given by the equation

$$h(x) = \text{sgn}(w_k x + b_k) \tag{2.1}$$

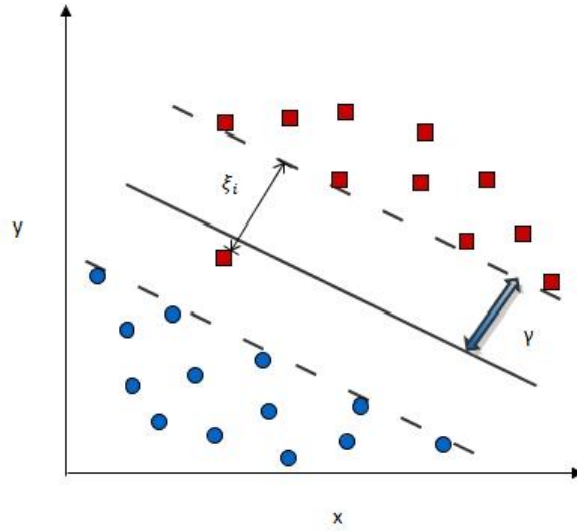


Figure 2.3: In this figure, a hyperplane classifies two classes of training data which is two dimensional by allowing error ξ_i in classification. The two classes of data are represented by squares and circles. The bold line, which is separated from the closest training vectors by distance γ is the hyperplane.

The final weight vector w is in the form [46]

$$w = \sum_{i=1}^N \alpha_i y_i x_i \quad (2.2)$$

where α_i is the number of incorrect predictions made on the example. By using 2.1 and 2.2, the equation for the hyperplane becomes

$$h(x) = \text{sgn} \left(\sum_{i=1}^N \alpha_i y_i \langle x_i, x \rangle + b \right) \quad (2.3)$$

The representation of the data is in dot products.

Soft Margin Classifier [45]: In real world scenarios the decision boundaries might appear curved and there could be no straight line dividing the data points into two different classes in the data space. It is desirable to have a smooth decision boundary

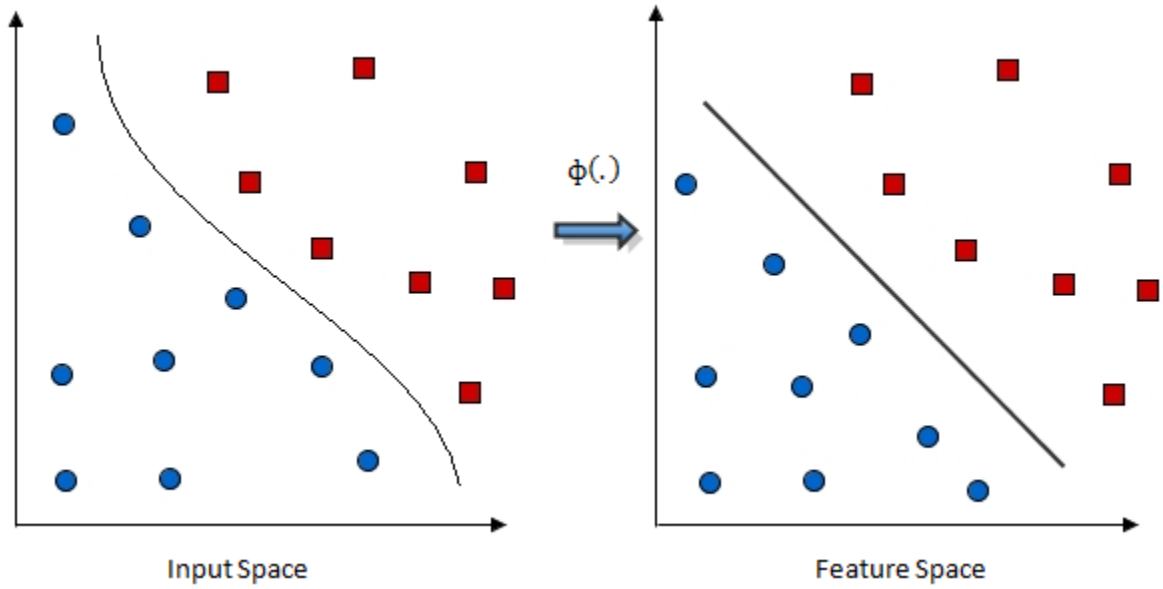


Figure 2.4: The figure shows the mapping $\Phi(\cdot)$ of attributes of the input space to the feature space by a Kernel Function.

which ignores a few data points. By doing this, it is straight instead of being curved around some outliers. Thus, we allow error ξ_i in classification. ξ_i are just “slack variables” in optimization theory [47]

$$\begin{cases} wx_i + b \geq 1 - \xi_i & y_i = 1 \\ wx_i + b \geq \xi_i & y_i = -1 \\ \xi_i \geq 0 \end{cases} \quad (2.4)$$

We need to minimize

$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \quad (2.5)$$

where C is the tradeoff parameter between error and margin.

Kernel Trick: A separating hyper plane is used in the classification of linear data. However, in real world problems the data sets are mostly non-linear. In such cases, kernels are used to non-linearly map the input data to a high-dimensional space. The new mapping is then linearly separable [45]. The space x_i are in is called the Input space and the space of $\Phi(x_i)$ after transformation is called Feature space. Linear operation in the feature space is equivalent to non-linear operation in input space [47].

As discussed, we know that the data is represented in dot products. The dot product allows us to use kernels which implicitly remap and compute dot products. An SVM algorithm using kernel function $K(x, y)$ for some vectors x and y is as below [46]. Thus, [47] Kernel trick allows SVM's to form nonlinear boundaries. The

Algorithm 2 SVM Algorithm with a Kernel Function $K()$

```

while mistakes are made in for loop do
  for  $i = 1$  to  $N$  do
    if  $y_i(\sum_{j=1}^N \alpha_j y_j K(x_j, x) + b) \leq 0$  then
       $\alpha_i \leftarrow \alpha_i + 1$ 
       $b \leftarrow b + y_i R^2$ 
    end if
  end for
end while

```

relationship between the kernel function K and the mapping

$$\Phi(\cdot)$$

is

$$K(x, x') = \langle \phi(x), \phi(x') \rangle \quad (2.6)$$

By specifying K we indirectly specify $\Phi(\cdot)$. $K(x, x')$ needs to satisfy the Mercer condition in order for $\Phi(\cdot)$ to exist. A real valued function $K(x, y)$ is said to fulfill Mercer's

condition if for all square integrable functions $g(x)$ one has [48]

$$\iint K(x, x')g(x)g(x')dxdx' \geq 0 \quad (2.7)$$

Kernel Functions The idea of the kernel function is to enable operations to be performed in the input space rather than the potentially high dimensional feature space. Hence the inner product does not need to be evaluated in the feature space. We want the function to perform mapping of the attributes of the input space to the feature space. The kernel function plays a critical role in SVM and its performance. It is based upon reproducing Kernel Hilbert Spaces [49] [50].

$$K(x, x') = \langle \phi(x), \phi(x') \rangle \quad (2.8)$$

If K is a symmetric positive definite function, which satisfies Mercer's Conditions,

$$K(x, x') = \sum_m^{\infty} a_m \phi_m(x) \phi_m(x'), a_m \geq 0 \iint K(x, x')g(x)g(x')dxdy \geq 0, g \in L_2 \quad (2.9)$$

then the kernel represents a legitimate inner product in feature space. The training set is not linearly separable in an input space. The training set is linearly separable in the feature space. This is called the "Kernel trick" [49] [51].

The four types of Kernel Functions usually used with SVM are:

1. Linear Kernel $K(x, x') = (x \cdot x' + 1)$
2. Polynomial Kernel $K(x, x') = (kx \cdot y + c)^p$
3. Sigmoid Kernel $K(x, x') = \tanh(kx \cdot y + c)$
4. Radial Basis Kernel $K(x, x') = e^{-\gamma \|x-y\|^2}$

For this study, we employed a Linear Kernel.

2.4 Hadoop MapReduce for feature extraction

Apache Hadoop is a powerful tool used in the analysis of Big Data. It is a framework written in Java and supports distributed computing. Hadoop provides the power of scanning and analyzing terabytes of data using a collection of commodity servers working in parallel [5]. Hadoop MapReduce is a programming paradigm and software framework which is used for writing applications that rapidly process vast amounts of data in parallel on large clusters of compute nodes [52]. An application is broken down into small fragments of tasks and each of these tasks are executed or re-executed on any of the compute nodes of the cluster [53]. The Hadoop Distributed File System (HDFS) discussed in Section 2.4.3 is used to store data on the compute nodes. Scheduling and monitoring tasks are handled by the framework. The framework also takes care of re-executing failed tasks. In this study, CMS Data is stored using Hadoop Distributed File and HDFS logs are processed using Hadoop MapReduce, discussed in Section 2.4.1.

2.4.1 Hadoop MapReduce

MapReduce is a programming model for data processing and is used to write programs that run in the Hadoop environment [2]. Large volumes of data are processed in parallel by dividing the work into a set of independent tasks [1]. Each process consists of two phases - the Map phase and the Reduce phase and is leveraged by functional programming constructs. Data in MapReduce cannot be subject to any changes and thus are immutable. This is so, since the communication occurs only by generating new output (key, value) pairs which are forwarded to the next phase of execution by

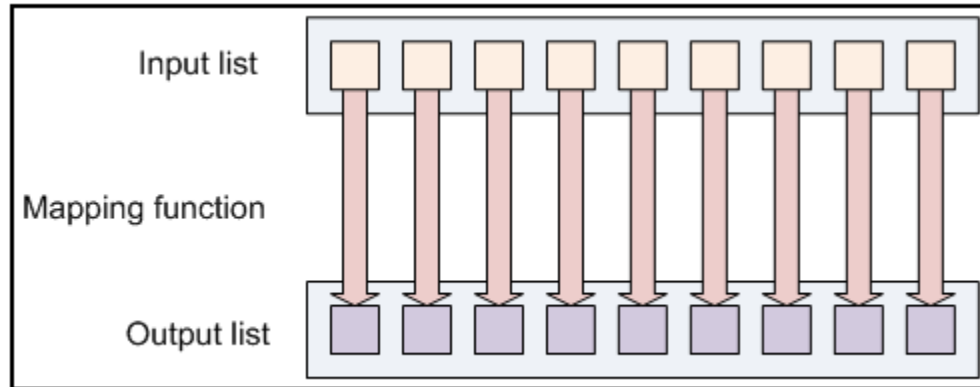


Figure 2.5: Map Function creates a new output list i.e (key, value) pairs from an input list. Courtesy [1]

the Hadoop framework. This approach improves the performance of the system by overcoming the communication overhead which occurs in keeping data on the nodes synchronized [2][1].

2.4.1.1 Map Function

Figure 2.5 shows the input list which consists of FileSplits obtained by splitting the input file. This splitting has no dependency on the internal logical structure of the input file. For each FileSplit a map task, called Mapper, is created [53]. The mapper transforms each element individually to an output data element in a parallel manner. The output from the Mapper is a list of (key, value) pairs. The number of Mappers determines the level of parallelism. In this study, the right level of parallelism was chosen individually for each job based on the size of data being processed and the number of cores that were requested to run the job using Hadoop on Demand (HOD) which is discussed in Section 2.4.4. Mostly, 101 Mappers and 11 Reducers were used, since this resulted in adequate performance.

The output from the Mapper is sorted and partitioned per Reducer by the MapRe-

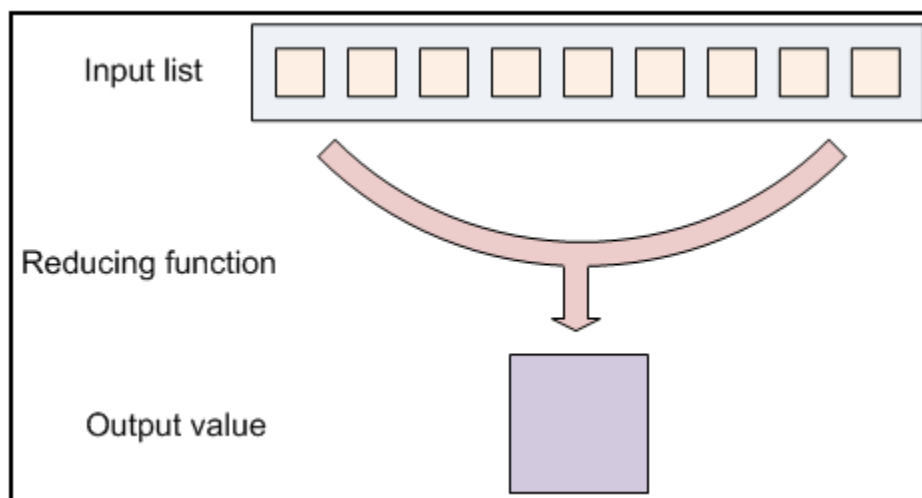


Figure 2.6: Reduce Function processes the output from the Map Function. It sorts the (key, value) pairs and groups them by the key values [2]. Courtesy [1]

duce framework. The number of partitions is equal to the number of reduce tasks for the job.

2.4.1.2 Reduce Function

A reduce task is called a Reducer. As discussed in the previous subsection, the Reducer takes the output of the Mapper which is then sorted by the framework as its input. It iterates through this list of (key, value) pairs and combines the values corresponding to a set of distinct keys. The way these values are combined is dependent on the objective of the job. The Reducer also emits a list of (key, value) pairs. Its output is nothing but a summary of the Mapper's output [1].

2.4.2 Hadoop Streaming

Using Hadoop Streaming any language that can read standard input and write standard output can be used to write a MapReduce program [54][2]. Mappers and Re-

ducers receive their input on stdin and emit output (key, value) pairs on stdout [1]. Communication between the Mappers and Reducers happens via these (key, value) pairs.

In streaming, the representation of input and output is textual. The input to the Mapper can be any text, HDFS logs in our case. Mapper processes this text and writes (key, value) pairs to the stdout. This is in turn taken as the input by a Reducer and emits (key, value) pairs to the stdout. Thus, output from streaming programs is written in the format:

```
key \t value \n
```

2.4.3 Hadoop Distributed File System: HDFS

The Hadoop Distributed File System is a distributed file system which supports redundant storage of large quantities of data across multiple disconnected disks and makes it appear as a single storage unit. Its design is based on the design of the Google File System (GFS) as described in [37]. It is designed for storing very large files that are gigabytes or terabytes in size with streaming data access patterns that run on clusters on commonly available hardware available from multiple vendors, generally known as commodity hardware. Each file is split into HDFS blocks of size 64MB by default and these blocks are stored on datanodes [2]. An HDFS cluster has a master-worker architecture [55][2] where a namenode is the master node and datanodes are worker nodes.

- Name node: It is a master server which maintains the filesystem namespace and the metadata of all the files and directories present in the filesystem tree. The name node has the information of how many HDFS blocks a file consists of and

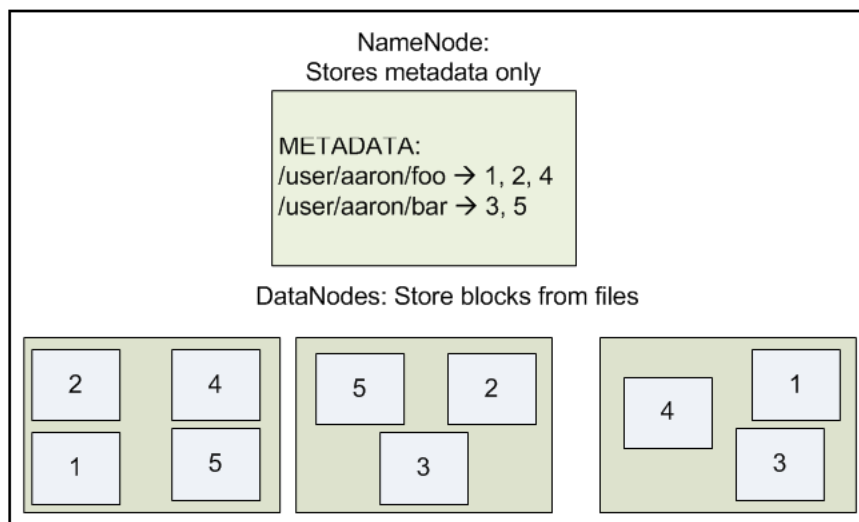


Figure 2.7: HDFS Architecture. Courtesy [1]

the data nodes on which these blocks are stored. File access by the clients are regulated by the name node.

- **Data nodes:** These are the worker nodes which receive instructions from the namenode for block creation, deletion and replication. They manage the storage attached to the nodes on which they run and are responsible for storing and retrieving blocks as requested by the clients and/or namenode.

HDFS Commands: The basic structure of an HDFS command is:

```
hadoop fs <cmd >
```

Some of the basic HDFS commands used are listed below [1]:

- `-ls path`: Lists the contents of the directory specified in the *path*
- `-mv srcdest`: Moves the file or directory within the HDFS from *src* to *dest*
- `-cp srcdest`: Copies the file or directory within the HDFS from *src* to *dest*

- `-rm path`: Removes the file or empty directory specified in the *path*
- `-rmr path`: Removes the file or empty directory specified in the *path* and recursively deletes child entries, if any
- `-put localSrc dest`: Copies the file or directory from the local file system (*localSrc*) into the HDFS (*dest*)
- `-get src localDest`: Copies the file or directory from the HDFS (*src*) into the local file system (*localDest*)
- `-cat filename`: Displays the contents of *filename*

2.4.4 Hadoop On Demand (HOD)

HOD enables the use of Hadoop with batch schedulers to provision virtual Hadoop clusters on a large shared physical cluster of nodes [56][57]. Hadoop clusters are provisioned on demand and the scheduling is handled by resource managers like torque. A request can be made for N number of nodes to a resource manager and the resource manager then provisions them with a Hadoop cluster. Thus, a virtual hadoop cluster is setup quickly and used [57]. After completion of the task at hand, the resources are freed by the HOD and henceforth, these nodes are made available again. While using HOD, Hadoop is not required to be installed on the cluster. A Hadoop tar ball can be provided by the user which is then distributed by HOD to all the nodes which are required to form the virtual Hadoop cluster.

HOD has the following components [57]:

- HOD Client: The HOD client is a Unix command that users use to allocate Hadoop MapReduce clusters. The command provides other options to list allocated clusters and deallocate them. The HOD client generates the hadoop-

site.xml in a user specified directory. The user can point to this configuration file while running Map/Reduce jobs on the allocated cluster.

- RingMaster: The RingMaster is a HOD process that is started on one node per every allocated dynamic cluster. It is submitted as a ‘job’ to the resource manager by the HOD client. It controls which Hadoop daemons start on which nodes. It provides this information to other HOD processes, such as the HOD client, so users can also determine this information. The RingMaster is responsible for hosting and distributing the Hadoop tarball to all nodes in the cluster. It also automatically cleans up unused clusters.
- HodRing: The HodRing is a HOD process that runs on every allocated node in the dynamic cluster. These processes are run by the RingMaster through the resource manager, using a facility of parallel execution. The HodRings are responsible for launching Hadoop commands on the nodes to bring up the Hadoop daemons. They get the command to launch from the RingMaster.
- HOD configuration file: An INI style configuration file where the users first configure various options for the HOD system, including install locations of different software, resource manager parameters, log and temp file directories, parameters for their MapReduce jobs, etc.
- Submit Nodes: Nodes where the HOD Client is run, from where jobs are submitted to the resource manager system for allocating and running clusters.
- Compute Nodes: Nodes which get allocated by a resource manager, and on which the Hadoop daemons are provisioned and started.

The sequence of operations in allocating a cluster and running jobs is pictorially represented in Figure 2.8:

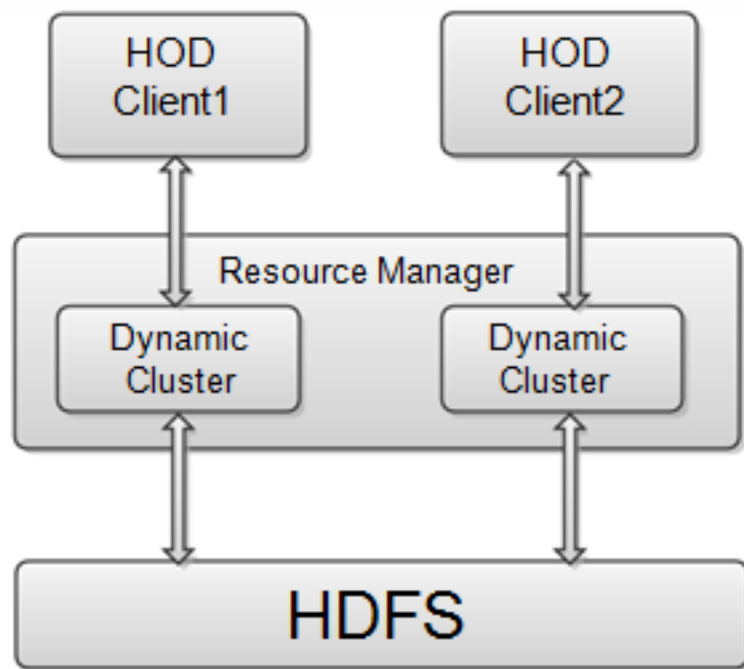


Figure 2.8: Allocation of a virtual Hadoop cluster and running jobs on it.

1. Using the HOD client on the Submit node, the user specifies the required number of cluster nodes.
2. Using a Resource Manager interface, a Resource Manager job which is a HOD process called RingMaster is submitted to the central server of the Resource Manager.
3. Jobs are assigned by the resource manager central server to the resource manager slave daemons on the compute nodes. The RingMaster process is started on one of the nodes.
4. Using another Resource Manager interface, the RingMaster runs the second HOD Component, HODRing.

5. The HODRing gets Hadoop Commands by communicating with the RingMaster.

2.5 Python

Python is defined as an “object-oriented scripting language” [58]. It is an object oriented programming language which is often used for scripting purposes. The programming paradigm of Python resembles that of dynamic languages like Perl and Ruby [59].

Here, python is used for coding the mapper and reducer scripts that are run on the hadoop framework via hadoop streaming [2]. The most frequently used python module is “re”. It provides regular expressions matching operations similar to those found in Perl [60]. Processing the HDFS logs was treated as a text processing task. The data structure “Dictionaries” were extensively used. Dictionaries are nothing but “associative arrays”. The algorithms used to process these HDFS logs in order to extract the features required by the SVMs have been discussed in Section 3.1.3.

2.6 Weka: Machine Learning Software

Weka stands for Waikato Environment for Knowledge Analysis and was developed at the University of Waikato in New Zealand [7]. It is an open source machine learning suite written in Java which provides an interface to various machine learning algorithms. It supports pre-processing of data sets and helps in evaluating performances of learning algorithms [61]. Weka has a graphical user interface, called the Weka Explorer, and a command line interface is also available. Both the Weka Explorer as well as the command line interface were used in this study.

The tools provided by Weka for data pre-processing are known as “filters”. Different filters are available for discretization, normalization, resampling, attribute selection, transforming and combining attributes. After the creation of ARFF files as described in section 3.1.6, they were loaded into Weka and were preprocessed using these filters. ReplaceMissingValues and Discretization filters were used. Data pre-processing is discussed in Section 3.1.2.

Training schemes that perform classification are used to classify the preprocessed data. Many classification schemes like decision trees, rule learners, naive Bayes, decision tables, locally weighted regression, SVMs, instance based learners, logistic regression, voted perceptrons and multi-layer perceptrons are supported by Weka. Weka’s Sequential Minimal Optimization (SMO) implementation of SVM is used in this study.

Graphical and text based results are generated by the explorer. The performance of an SVM classifier can be determined by the confusion matrix which is part of the text based result (Section 3.3.2). The threshold curve with False Positive rate along the x-axis and True Positive rate along the y-axis gives the Receiver Operating Characteristics (ROC) described in Section 3.3.3.

Nine models were built in the Explorer from nine different training sets. After identifying the best classification model among the nine classification models, the unknown data which needed to be classified was used against this model at the command line interface of Weka.

Chapter 3

Materials and Methods

3.1 Data Collection

The SVM learning algorithm used in this thesis was trained on several training sets. The classification performance was examined using the test sets. We now discuss the preparation of these training and testing sets.

3.1.1 Data Source

HDFS logs for ninety days starting February, 2009 to April, 2009 were considered. These logs have information with regard to file operations of every file belonging to the subset of USCMS data stored at Nebraska.

3.1.2 Data Pre-processing

HDFS logs might contain log information pertaining to a file being accessed many number of times within a small period of time which could be due to a faulty process trying to access a file or a set of files over and over again. The logs could also contain information about files which were only used for managing the storage and which

cannot be considered in solving our problem. Features of such files become outliers in this study and it is imperative for the data being used for learning by the SVMs to be clear of such outliers. This data also had to be transformed to a format suitable for SVMs, to improve the performance. Hence, preprocessing was required.

3.1.2.1 Data Cleaning

The Weka Explorer was used for data cleaning. Weka reports the minimum, maximum, mean and standard deviation values for all the attributes in a data set. This information is used to eliminate the outliers. Weka also reports the percentage of missing values in a data set. Using the filter, ReplaceMissingValues, missing values were replaced by the mean value for that attribute.

3.1.2.2 Discretization

It has been shown by [62] that discretization can help improve significantly the classification performance of learning classification algorithms such as SVMs. Weka's discretize filter helps in converting numeric attributes to nominal. Equal-width binning worked well with our data set. In Equal-width binning, the range of possible values is divided into N subranges of the same size [63][7].

$$binwidth = \frac{maximumvalue - minimumvalue}{N}$$

In this study, the value with approximately the best classification performance was the final choice for N which was mostly 60 for each of the nine training data sets.

3.1.3 Training Data

From the HDFS logs, the list of files which were deleted were used to obtain the positive data sets. Likewise, the list of files which were not deleted was used to obtain

negative data sets. The features of deleted and not deleted files were extracted using Hadoop MapReduce. The features which were extracted are as follows:

1. Number of times a file was accessed in one month time frame
2. IP from which a file was accessed
3. Lifetime of a file in the same one month time frame

We also considered the CMS data set to which a file belongs to by using the data set IDs. This information was retrieved from a database maintained by Carl Lundstedt, one of the grid administrators at the Holland Computing Center at the University of Nebraska-Lincoln.

Using Hadoop MapReduce for feature extraction: We start by making two lists of filenames. The first list contains the files which were deleted in the three month time frame. It is used to generate the positive data sets used in the training data. We call this list, the “Delete-List”. The second list, the “Save-List” consists of files which were retained and it is used to generate the negative data sets. The algorithms used to create the Delete-List and Save-list using the MapReduce paradigm [2] are presented in Algorithm 3 and Algorithm 4.

Once these lists are prepared from the HDFS logs, we have mappers and their corresponding reducers written in Python to be used with Hadoop Streaming. Each Mapper-Reducer pair is designed and developed to extract one of the three features listed above. We briefly discuss how each of these features was extracted:

1. Number of times a file was accessed in one month: The number a times a file is being accessed is one of the most important factors to be considered while trying to make a decision with regard to its retentivity. The HDFS logs are considered

Algorithm 3 To create Delete-List - Mapper

Require: Array containing filenames of deleted files, DeleteArray[] $\leftarrow 0$

```

for every line in HDFS logs do
  if cmd=delete in line then
    if word in line is a filename AND not in DeleteArray[ ] then
      add word to DeleteArray[ ]
    end if
  end if
end for
for every filename in DeleteArray[ ] do
  print filename
end for

```

Algorithm 4 To create Save-List - Mapper

Require: Array containing filenames of saved files, SaveArray[] $\leftarrow 0$

Require: Array containing filenames of deleted files, DeleteArray[] created using Algorithm 3

```

for every line in HDFS logs do
  if word in line is a filename AND not in DeleteArray[ ] then
    if word not in SaveArray[ ] then
      add word to SaveArray[ ]
    end if
  end if
end for
for every filename in SaveArray[ ] do
  print filename
end for

```

line-by-line. While generating the positive data sets, every line of the HDFS logs is searched to see if it contains access information for files featured in Delete-List. This is achieved by the Mapper, AccessMapper.py coded in Python. The AccessMappaer.py outputs “ $\langle filename \rangle, 1$ ” even though the $\langle filename \rangle$ might occur multiple times in the log file. This output from the Mapper is then taken as input to the Reducer, AccessReducer.py which sums the occurrences of each filename to a final count. Thus, the number of times a file was accessed is computed from the logs. The same is repeated for the files in Save-List. The

algorithm used to extract the access information is as below:

Algorithm 5 To compute the number of times a file was accessed - Mapper

```

for every filename in Delete-List do
  create a hash map with filename as the key and a counter as the value
end for
for every line in HDFS logs do
  extract the filename in the current line
  look up the hash map for this filename
  if filename is found in hash map then
    increment the respective counter
  end if
end for
print every (key, value) pair of hash map

```

Algorithm 6 To compute the number of times a file was accessed - Reducer

```

Require: counter  $\leftarrow 0$ 
for every line in mapper's output do
  key  $\leftarrow \langle filename \rangle inline$ 
  value  $\leftarrow 1 inline$ 
  while value of the key changes do
    print key, counter
    counter  $\leftarrow 0$ 
  end while
  counter  $\leftarrow counter + value$ 
end for

```

2. IP from which a file was accessed: It was observed that most of the access requests were coming from a distinct set of IPs. It was also observed that files belonging to different data sets were being accessed from the same set of IPs.

Thus, it is interesting to consider the IP from which a file was being accessed. We now discuss the algorithm used to extract the IP from which the corresponding file was accessed from the HDFS logs. It was observed that most of the files were being accessed from a single IP, so, only one IP per file has been considered in this study. Table 3.1 shows these observations from a few samples of HDFS logs.

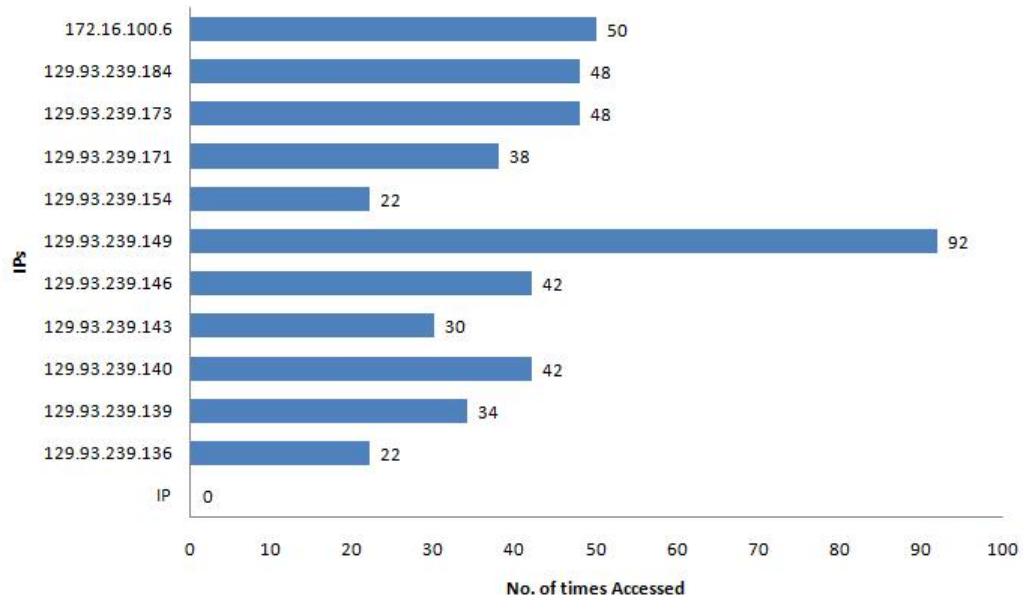


Figure 3.1: The graph shows the pattern of file access requests received from some IPs.

Table 3.1: Percentage of files being accessed by a single IP.

Month/Year	Total No. of Files	No. of Files Accessed by One IP	No. of files accessed by more than one IP	Percentage of files accessed by a single IP
Feb 2009	6145	5874	271	95.59
Mar 2009	6580	6142	708	89.3
Apr 2009	5470	5186	284	94.81

- Age of a file: The duration of existence of a file is also used as a factor here. We provide SVM with this data which is used to determine whether a file needs to be deleted or not. Out of a one month time frame we list the filenames and their corresponding age for both categories of files - those deleted and those which were not deleted. The algorithms used to get this data from the HDFS

Algorithm 7 To extract the IP from which a file was accessed - Mapper

```

for every filename in Delete-List do
  create a hash map with filename as the key and a counter as the value
end for
for every line in HDFS logs do
  extract filename and IP in the current line
  look up the hash map for this filename
  if filename is found in hash map then
    assign IP to this key
  end if
end for
print every (key, value) pair of hash map

```

Algorithm 8 To extract the IP from which a file was accessed - Reducer

```

for every line in mapper's output do
  key ← <filename>inline
  value ← IPinline
  while value of the key changes do
    print key, value
  end while
end for

```

Table 3.2: The training sets used in this study. A, B and C belong to February 2009; D, E and F belong to March 2009; G, H and I belong to April 2009.

Training Set ID	Positive Instances	Negative Instances	Total No. of Instances
A	5164	4085	9249
B	4467	4438	8905
C	4467	4139	8606
D	5752	4959	10711
E	4838	5688	10526
F	4479	5677	10156
G	3800	7680	11480
H	4968	6728	11696
I	3696	6728	10424

logs is presented here.

The algorithmic complexity of all these algorithms is $O(n)$ and it depends on the size of the input log file.

Algorithm 9 To calculate the age of a file - Mapper

```

for every filename in Delete-List do
  FirstAccess: create a hash map with filename as the key to hold first access
  timestamp
  LastAccess: create a hash map with filename as the key to hold first access
  timestamp
end for
for every line in HDFS logs do
  extract filename and timestamp in the current line
  look up the hash map for this filename
  if filename is found in hash map then
    if value for key is unset then
      assign timestamp to this key in FirstAccess
    end if
    if value for key is set then
      assign timestamp to this key in LastAccess
    end if
  end if
end for
print every (key, value) pair of both hash maps

```

Algorithm 10 To calculate the age of a file - Reducer

Require: $diff \leftarrow 0$

Require: $DateArray[] \leftarrow 0$

```

for every line in mapper's output do
   $key \leftarrow \langle filename \rangle inline$ 
   $value \leftarrow date inline$ 
  if value of the key changes then
    calculate the difference between first and last file access timestamps; consider
    just the date leaving out the seconds and microseconds from the timestamp,
    diff
    remove all the elements in  $DateArray[ ]$ 
  end if
  print key, diff
end for

```

3.1.4 Test Data

The test data was also prepared from the HDFS logs. All features of the files i.e. the number of times a file was accessed in one month, the IP from which the file was being accessed and the number of days of existences of the file within the one month period were extracted from these HDFS logs using the algorithms in Section 3.1.3. Thus, the test data contained both positive and negative instances. The performance of each classification algorithm was tested against these test data sets.

3.1.4.1 Testing against training data belonging to the same time frame

Three test sets were generated one each for February, March and April of 2009. The February test set was tested against all the three classifiers that were prepared using the HDFS logs from the same month, February, 2009. This experiment was repeated for March and April of 2009 as well. Results are discussed in Chapter 4.

3.1.4.2 Testing against training data belonging to a different time frame

Among the nine classifiers, two classifiers with the best Classification Accuracy Percentage and Receiver Operating Characteristic (ROC) were short listed and these were tested against test data prepared using a different month's log data.

Having different test data helps in identifying the best classifier as it helps in evaluating the performance of the classifier when tested against test data belonging to a different time frame. This is a very important factor since the model which is developed here is intended for practical use at the Holland Computing Center to address a maintenance issue - creating more space for storage by eliminating files which are of least value. In other words, the classifier which has been developed


```

#$-N hadoop-on-demand
#$-l h_rt=12:00:00,excl=true
#$-S /bin/bash
#$-pe hadoop4
#$-cwd
#$-M ranjinis@cse.unl.edu
#$-m abe
#$-v HADOOP_HOME="/home/swanson/ranjinis/hadoop-on-demand2.0/hadoop_home"
#$-v JAVA_HOME="/util/comp/sun/jdk1.6.0_10"

/home/swanson/ranjinis/hadoop-on-demand2.0/hadoop_home/bin/hadoop jar
/util/comp/hadoop/hadoop-0.19.0/contrib/streaming/hadoop-0.19.0-streaming.jar -mapper
/home/swanson/ranjinis/work/hdfs_original_logs/correct/nowWorkingIP.py -reducer
/home/swanson/ranjinis/work/hdfs_original_logs/correct/WORKING/NEW/IPReducer.py -input
/users/ranjinis/FEBone/ -output /users/ranjinis/myOutputDirs103 -file
/home/swanson/ranjinis/work/hdfs_original_logs/correct/del100

```

Figure 3.2: Figure shows a script used for submitting Hadoop Streaming jobs using SGE on Prairiefire.

based on log data of 2009 should be useful in classifying files which are created and accessed in the future months after the classifier was created.

3.1.5 Running MapReduce using Hadoop On Demand (HOD)

MapReduce programs written in Python were run with Hadoop Streaming using Hadoop On Demand. Sun Grid Engine (SGE), an open source batch-queuing system [64] was used to submit Hadoop Streaming jobs to Prairiefire - a 90-node Beowulf cluster [65]. Figure 3.2 is an example of an SGE script which was used to submit jobs to the queue. The sge script helps in requesting the required number of cores for the virtual hadoop cluster.

-pe hadoop 4

It also contains the Hadoop Streaming command. Hadoop streaming jar comes with the Hadoop distribution and it facilitates MapReduce jobs. The option `-mapper` is used to specify the mapper, a python script that needs to be distributed to the nodes which perform the map function. Likewise, the option `-reducer` is used to specify the reducer used by the nodes to perform the reduce function. Options `-input` and `-output` give the HDFS path of the input and output files respectively. `-file` option is used to package an auxiliary file required by the mapper.

3.1.6 Creating an ARFF file for WEKA

An ARFF, Attribute Relation File Format was developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the WEKA machine learning software. It is an ASCII file which contains a list of instances sharing a set of attributes [66], which is a standard way of representing data sets whose instances are independent, unordered and there is no relationship among these instances [7]. ARFF file structure is described using Figure 3.3. The name of the relation appears at the beginning of the file. `@relation` declaration associates a name with the data set. Here, the name of the relation is “logdata” and it is represented as

```
@relation logdata
```

Following the name of the relation is a block defining the attributes such as ID, AccessNumber, Age and IP. `@attribute` declaration specifies the name and type of an attribute.

```
@attribute ID numerical
```

```
@attribute AccessNumber numerical
```

```
@attribute Age numerical
```

```

%ARFF file for the log data with numeric features

@relation logdata

@attribute ID numeric
@attribute accessno numeric
@attribute IP numeric
@attribute age numeric
@attribute delete? {yes, no}

@data

%15 instances

1 3 172161104 0 yes
2 3 172161121 0 yes
3 149 172161008 5 no
4 3 172161155 0 yes
5 167 172161008 7 no
6 176 172161008 8 no
7 3 172161169 0 yes
8 3 172161156 0 yes
9 3 172161105 0 yes
10 128 172161008 1 no
11 159 172161008 6 no
12 3 172161140 0 yes
13 201 172161008 3 no
14 3 172161152 0 yes
15 3 172161179 0 yes

```

Figure 3.3: ARFF file for the log data.

@attribute IP numerical

@attribute delete? yes, no

All the attributes take numeric values and are followed by the keyword `numeric`. `delete?` is the class value which is predicted from the values of other attributes.

The attribute definitions are followed by `@data` declaration in a single line denoting the start of the data segment. Every line corresponds to an instance and each instance is represented by the values taken by its attributes. Missing values, if any, are represented by a single question mark.

ARFF files for our study were created from the outputs of MapReduce processes which were discussed in Section 3.1.6.

Algorithm 11 To create ARFF file for the log data

Require: create a file with .arff extension

```

write
"@relation logData"
into ARFF the file
write
"@attribute ID numeric
@attribute accessno numeric
@attribute IP numeric
@attribute age numeric
@attribute delete? yes, no"
into the ARFF file.

```

Require: AccessArrayDel [] \Leftarrow *accessNumberofeachfilefromthefilecreatedusing5*

Require: AccessArraySave [] \Leftarrow *accessNumberofeachfilefromthefilecreatedusing5*

Require: AgeArrayDel [] \Leftarrow *accessNumberofeachfilefromthefilecreatedusing6*

Require: AgeArraySave [] \Leftarrow *accessNumberofeachfilefromthefilecreatedusing6*

Require: IPArrayDel [] \Leftarrow *accessNumberofeachfilefromthefilecreatedusing7*

Require: IPArraySave [] \Leftarrow *accessNumberofeachfilefromthefilecreatedusing7*

write the contents of these arrays line by line into the ARFF file; each line corresponding to a file.

3.2 File Classification Method - Support Vector

Machines

Weka is a collection of machine learning algorithms for data mining tasks [61]. It implements John Platt's sequential minimal optimization algorithm for training a support vector classifier [7]. This SVM implementation of Weka was used in this study. Both the positive and the negative instances from the training sets were used to train the SVM.

Linear and RBF Kernel functions were used to create a hyperplane for classifica-

tion in this study [67]. Simple linear SVM was used since it was the fastest to learn and it is known to provide good generalization accuracy [68]. RBF was the second fastest algorithm to learn and it was used to observe benefits, if any, like the difference in accuracies between the two.

Using the Weka command prompt, the following command was used for training the SVM with linear function:

```
java weka.classifiers.functions.SMO -C 1.25 -L 0.0015 -N 1 -t testOneWithID.arff -T
testTwoID.arff -p 1
```

where,

-C is the complexity constant whose default value is 1.0

-L is the tolerance parameter whose default value is 0.0010

-N is option to specify whether to 0=normalize/1=standardize/2=neither, default value is 0=normalize

-V is the number of folds for the internal cross-validation

-K is the kernel to use

-T is used to set the training file i.e. testOneWithID.arff

-t is used to set the test file i.e. testTwoID.arff

-p is used to output only the predictions for test instances, this helps in tracking the test instances

For training the SVM with the RBF Kernel, the following command was used:

```
java weka.classifiers.functions.SMO -C 1.25 -L 0.0015 -N 1 -K
"weka.classifiers.functions.supportVector.RBFKernel -C 250007 -E 1.0" -t
testOneWithID.arff -T testTwoID.arff -p 1
```

here,

-C which is used in -K is used to set the cache size

-E which is also used in -K is used to set the exponent

3.3 Performance Analysis

For each pair of training and test data which were prepared as described in Section 3.1.3, the classification performance was analyzed for the Linear Kernel method. The types of analyses that were performed are Cross-Validation Analysis, Confusion Matrix and Accuracy Rate where the accuracy rate gives the measure for classification performance and Receiver Operating Characteristic (ROC) curve for more detailed analysis.

3.3.1 Cross-Validation

Cross-Validation was used in analyzing the classification performance [69]. It helps in estimating generalization error based on “resampling”. The resulting generalization error of each of the nine models that were prepared as described in Section 3.1.3 were considered and the model with the smallest estimated generalization error was chosen [69].

In the “leave-one-out” method one item from the training data set is left out and the learning algorithm is trained on the rest of the items. The trained model is then used to predict the label of the one left out earlier. This process is repeated for each item of the training set by leaving it out and predicting its label from the trained model prepared from the rest of the items in the training set. It was shown that although this method of cross-validation works well for estimating generalization error for continuous error functions such as the mean squared error, it performs poorly for discontinuous error functions such as the number of misclassified cases [70]. Thus,

Table 3.3: Confusion Matrix

		Predicted	
		Positive	Negative
Actual	Positive	True Positive (TP)	False Negative (FP)
	Negative	False Positive (FN)	True Negative (TN)

a k -fold cross-validation was preferred where the training data set is broken into k sets of data, each of size n/k , where n is the size of the training data set. The learning algorithm is trained on $k - 1$ sets and tested against 1. This process is repeated k times after which the mean accuracy is calculated. A small value of k makes the analyses more pessimistic and this helps in selecting the best model [70]. Choosing too small a value for k , for instance, 3-fold is shown to result in wastage of data and more expensive [69]. Thus, a value of 10 for k was chosen for estimating the generalization error.

3.3.2 Confusion Matrix and Accuracy Rate

A Confusion Matrix is a 2 dimensional matrix which represents the actual and predicted classifications done by a classifier [71]. The performance of a model is evaluated based on the data in the confusion matrix. The structure of a confusion matrix for a two class classifier is represented in Table 3.3.

This is a confusion matrix, with true positive, false positive, true negative, and false negative.

- True Positive (TP) is the number of correct predictions that an instance is positive

- False Negative (FN) is the number of incorrect predictions that an instance is negative
- False Positive (FP) is the number of incorrect predictions that an instance is positive
- True Negative (TN) is the number of correct predictions that an instance is negative

Some of the common performance metrics that can be calculated using a confusion matrix are [71]:

- The accuracy (AC) is the proportion of the total number of predictions that were correct. It is determined using the equation:

$$AC = \frac{TP + TN}{TP + FN + FP + TN} \quad (3.1)$$

- The true positive rate (TP) is the proportion of correct predictions of positive instances. It is given by the equation:

$$TP = \frac{TP}{TP + FN} \quad (3.2)$$

- The false positive rate (FP) is the proportion of negative instances that were incorrectly predicted as positive. It is given by the equation,

$$FP = \frac{FP}{TN + FP} \quad (3.3)$$

- The true negative rate (TN) is the proportion of negative instances that were correctly predicted as negatives. It is given by the equation,

$$TN = \frac{TN}{TN + FP} \quad (3.4)$$

- The false negative rate (FN) is the proportion of positive instances that were incorrectly predicted as negative. It is given by the equation,

$$FN = \frac{FN}{TP + FN} \quad (3.5)$$

- The precision (P) is the proportion of positive instances that were correctly classified. It is given by the equation,

$$P = \frac{TP}{TP + FN} \quad (3.6)$$

All the above performance metrics were calculated for both Linear and RBF Kernel Methods on all training sets. They were calculated from independently prepared test sets and from the cross-validation test.

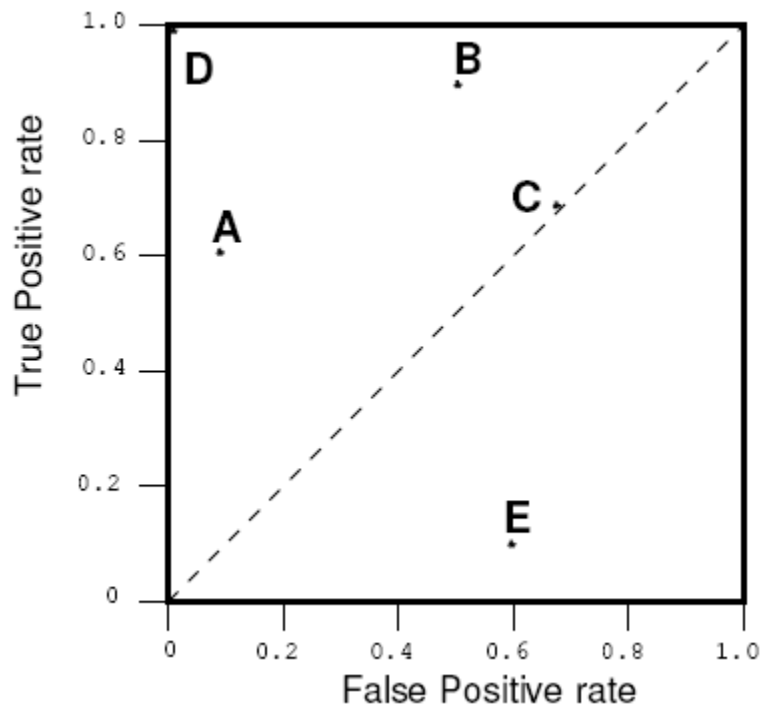


Figure 3.4: A basic ROC graph showing five discrete classifiers. A discrete classifier is one that outputs only a class label. Each discrete classifier produces an (FP rate, TP rate) pair, which corresponds to a single point in ROC space. (courtesy [3]).

3.3.3 Receiver Operating Characteristic

Receiver operating characteristic (ROC) analysis is being used to evaluate the performance of classifiers [3]. An ROC graph is a plot with the false positive rate along the X axis and the true positive rate along the Y axis. It helps in visualizing relative trade-offs between benefits (true positives) and costs (false positives). Figure 3.4 shows an ROC graph with five classifiers labeled A through E. Now, we discuss this basic ROC graph shown in Figure 3.3. In the graph, the point (0, 1) is the perfect classifier, it classifies all positive and negative instances correctly. *D*'s performance is perfect as shown in the figure. The point (0, 0) implies that both the false positive and true positive rates are 0, which means that the classifier gives out neither false

positive error nor does it give out any true positives. Thus, it represents a classifier that predicts all instances to be negative. Likewise, the point $(1, 1)$ implies that both false positive and true positive rates are 1, and thus, the classifier predicts every instance to be positive. Point $(1, 0)$ is the classifier that is incorrect for all classifications. Classifier *A* will have more true positives than false positives and hence the it has more number of correct predictions than incorrect predictions. Classifier *B* has higher true positive and false positive rates compared to Classifier *A*. Classifier *C* classifies only half of the instances correctly and classifier *E* has the worst classifying performance since it classifies most of the instances incorrectly.

3.4 CMS Data Sets to be Deleted

All the files considered in this study belong to some data set which belong to the CMS Physics data. There exists a Dataset Bookkeeping Service (DBS) comprising a database and the services used to store and access metadata related to its physics data [72]. Metadata which provides the mapping between the files and the data set they belong to was recovered from a dump received from Carl Lundstedt, one of the Grid administrators at the Holland Computing Center. This dump was generated by querying the DBS.

Firstly, all the files which were marked to be deleted by the SVMs were mapped to their respective data sets. Secondly, these data sets were ranked according to the number of files that belonged to them. In other words, the data set with the maximum number of files to be deleted was ranked as the number one candidate for deletion. Lastly, a report was produced containing the data sets and their respective sizes in the order of their ranks. This report is intended for the use of grid administrators at the Holland Computing Center in managing the storage for the CMS project.

3.5 Retention Cost Heuristic

As described in Section 3.4, a list of data sets to which Nebraska subscribes was obtained by querying the DBS. This list also contains the corresponding size of the data sets. A perl script was developed to obtain the number of days since latest access D_{LA} for each data set. Retention Cost R_C was calculated as below:

$$R_C = \frac{\text{size}}{\text{totaldiskspace}} \times D_{LA}$$

Since the total disk space is constant, this can be ignored. Size of data sets is expressed in terms of GB. A threshold value T_{RC} for R_C is identified. Computation of T_{RC} has been discussed in Section 4.4. Data sets with value of $R_C > T_{RC}$ were predicted for deletion and data sets with $R_C < T_{RC}$ were predicted for retention.

Chapter 4

Results and Discussion

All the nine classification models were developed as discussed in 3. This Chapter discusses the accuracies of each of these nine classification models and their respective false positive rate analysis. We also discuss building a tool from the best classifier and how it works.

4.1 Identification of Data sets to be deleted

The Classifier with the best accuracy rate was chosen and used as the training data set in the tool.

4.1.1 Accuracy rates

The accuracies of all nine classifiers have been represented in Table 4.1.

Thus, training data set “A” has the best accuracy rate of 97.40% and training data set “B” has an accuracy rate of 92.50% which is only next to that of “A”.

Table 4.1: Cross Validation Accuracy rates of the nine classifiers

Training Set ID	Accuracy Rate
A	97.40
B	92.50
C	67.11
D	64.66
E	71.04
F	68.23
G	69.39
H	62.43
I	68.49

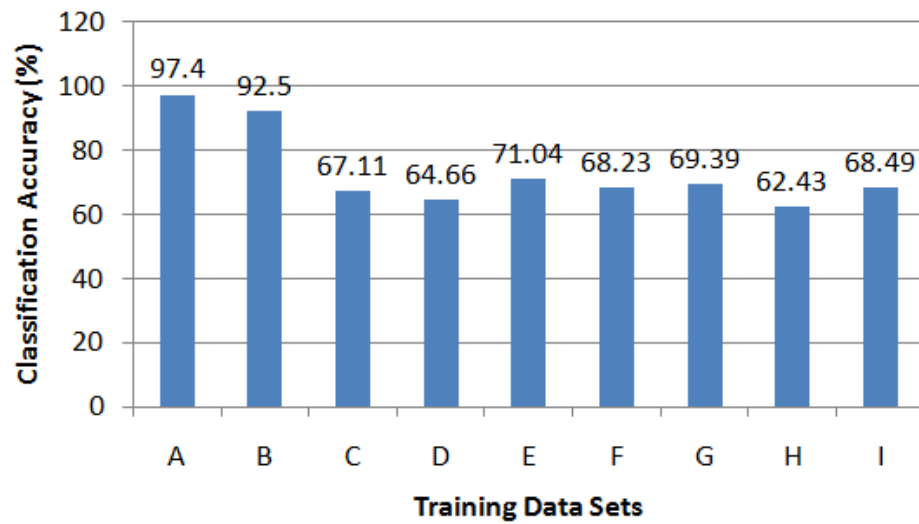


Figure 4.1: Accuracy rates of cross-validation test of all the nine SVM classifiers trained on different data sets.

Table 4.2: Accuracy rates of the top two classifiers against real data.

Training Set ID	Testing Set ID	Accuracy Rate
A	Btest	90.20
B	Atest	87.42
A	Ctest	89.05
B	Ctest	94.41
A	Dtest	91.25
B	Dtest	90.54
A	Etest	86.18
B	Etest	83.86
A	Ftest	75.03
B	Ftest	68.97
A	Gtest	66.52
B	Gtest	66.58
A	Htest	57.40
B	Htest	57.51
A	Itest	64.43
B	Itest	64.55

Top two classifiers, A and B were tested further against various eight other test sets as described in Chapter 3, Section 3.1.4 to determine which of these perform best. Table 4.2 contains these results. We still find that A outperforms B and hence, is chosen to be the classifier with best performance.

Another classifier was developed by picking 600 records randomly from each of these nine training sets i.e. A, B, C, D, E, F, G, H and I, 5400 files in total. This training set could contain log info for any day for those three months. The performance of this classifier is represented in Figure 4.3. This shows that no significant improvement in classification performance could be achieved by developing a classifier from a training set sampled from different months.

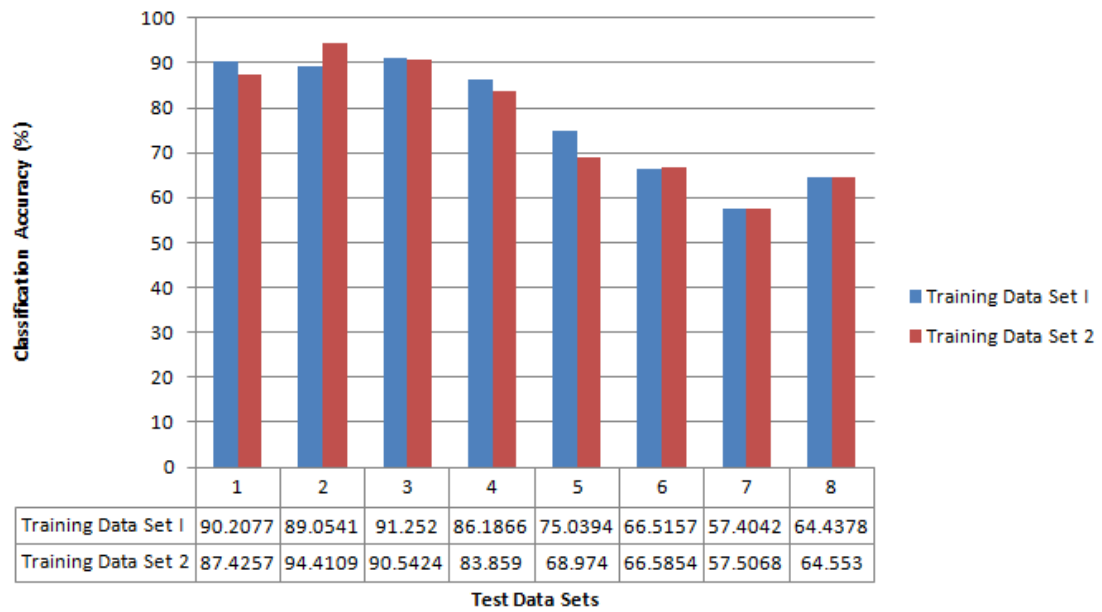


Figure 4.2: Accuracy rates of all the nine SVM classifiers tested against different test data sets.

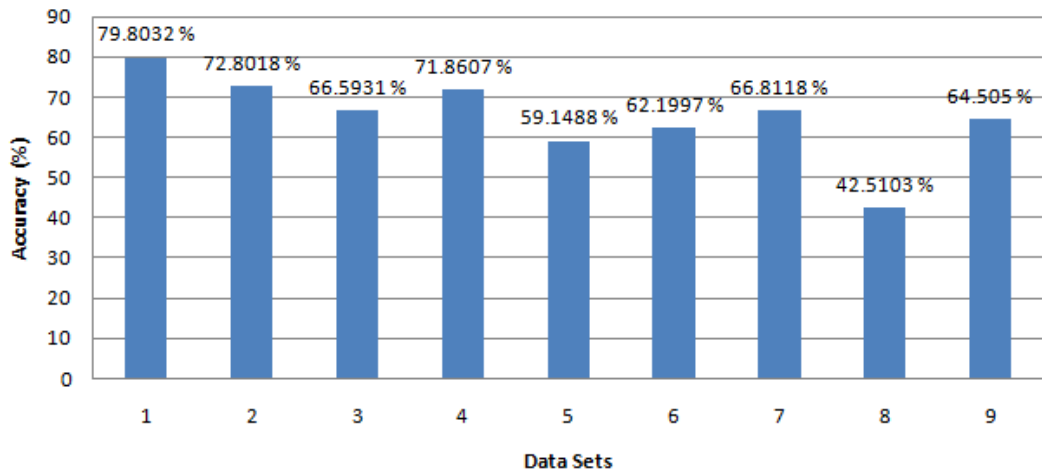


Figure 4.3: Accuracy rates of top two classifiers “A” and “B” tested against different test data sets.

Table 4.3: Confusion Matrix of Classifier A

		Predicted	
		Positive	Negative
Actual	Positive	5148 (TP)	16 (FP)
	Negative	224 (FN)	3861 (TN)

Table 4.4: Performance Metrics calculated using Confusion Matrix as discussed in 3.3.2.

Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	97.4
True Positive Rate	$\frac{TP}{TP+FN}$	95.8
False Positive Rate	$\frac{FP}{TN+FP}$	0.4
True Negative Rate	$\frac{TN}{TN+FP}$	99.6
False Negative Rate	$\frac{FN}{TP+FN}$	4.1

4.1.2 Confusion Matrix Analysis

The classifier developed from training set “A” has shown to have the best overall performance - cross validation accuracy and its accuracies against eight test data sets. The confusion matrix of this chosen classifier is represented in Table 4.3.

4.1.3 Receiver Operating Characteristic (ROC) Analysis

Figure 4.4 compares ROC of all the nine classifiers. As discussed in Section 3.3.3 of Chapter 3,

- Point (0, 1) is the perfect classifier, it classifies all positive and negative instances correctly
- Point (0, 0) implies that both the false positive and true positive rates are 0, which means that the classifier gives out neither false positive error nor does

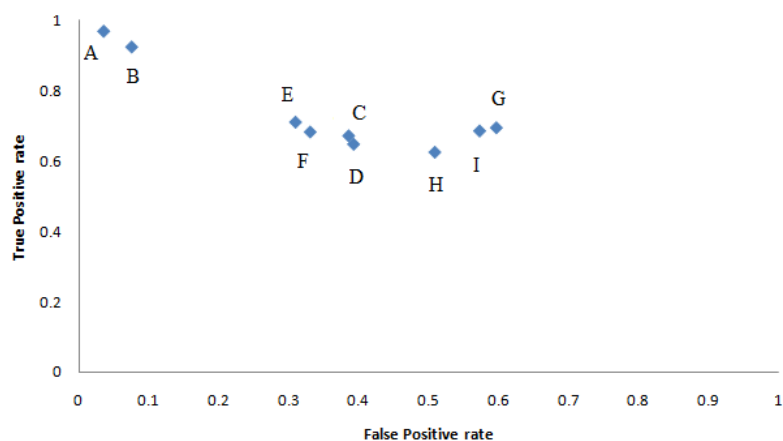


Figure 4.4: ROC of all the nine SVM classifiers.

it give out any true positives. Thus, it represents a classifier that predicts all instances to be negative

- Point (1, 1) implies that both false positive and true positive rates are 1, and thus, the classifier predicts every instance to be positive
- Point (1, 0) is the classifier that is incorrect for all classifications

Thus, A (0.036, 0.97) can be concluded as the best classifier which classifies most positive and negative instances correctly.

4.2 Testing

4.2.1 Unit Testing - Testing Python Scripts

All the python scripts used as Mappers and Reducers with Hadoop Streaming were first tested manually using simple unix commands.

```
cat data | python mapper.py | sort | python reducer.py
```

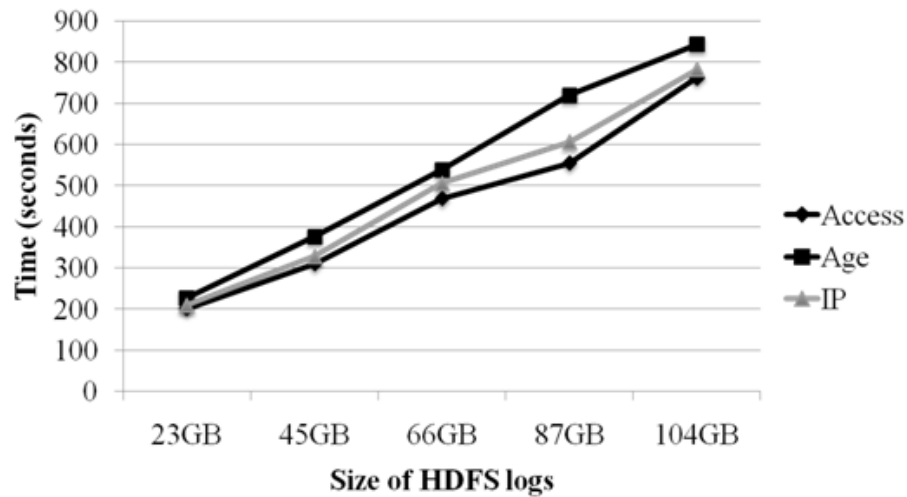


Figure 4.5: This figure shows the performance of the MapReduce programs used in this study and also shows that they are scalable

These unix commands used in the above specified sequence simulate the MapReduce framework which help in verifying and validating the functionality of mappers and reducers used.

4.2.2 Performance and Scalability Testing

The performance and scalability of the MapReduce programs used in this study are represented in Figure 4.5.

4.2.3 Integration Testing

Hadoop MapReduce using Hadoop Streaming and machine learning using Support Vector Machines were integrated using shell scripting and tested against valid and invalid input files of different sizes. Execution, error handling and creation of log file was verified for both scenarios.

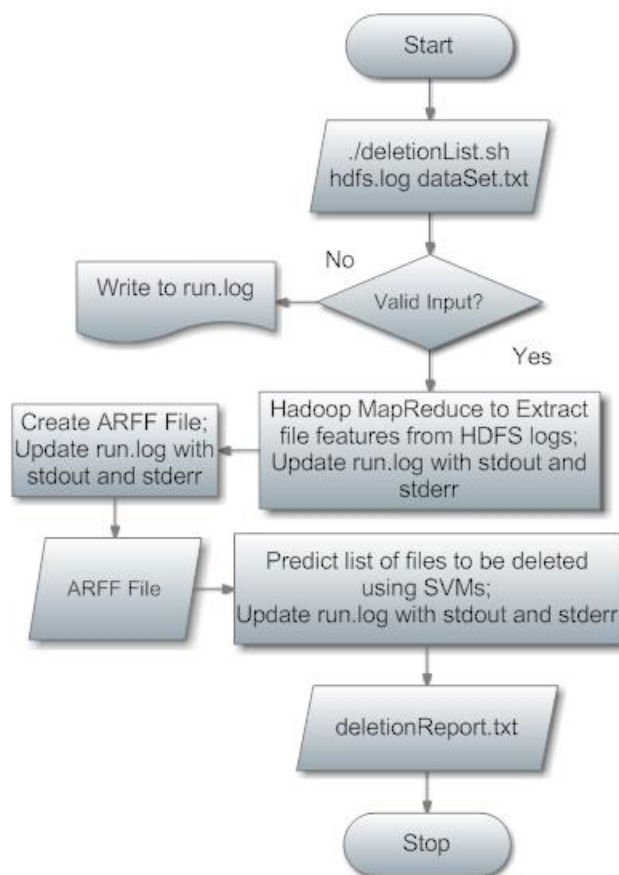


Figure 4.6: Flowchart depicting how Large Data Sets are managed using Support Vector Machines in this study.

4.3 Tool Usage and Result Interpretation

The product of this study is a shell script called “deletionList.sh” which can be run from the bash shell. It takes two input files, namely an HDFS log file and its corresponding CMS Data Set dump and outputs a list of CMS data set names predicted for deletion. Logs are maintained by logging each step of log data processing like Hadoop MapReduce and machine learning using Support Vector Machines. The log file, “run.log” can be found in the same directory where the shell script resides. The flow of execution is as shown in Figure 4.6.

4.3.1 Setting up the Environment for Tool Deployment

For successful execution of “deletionList.sh” the following prerequisites need to be set up on the cluster:

1. Python 2.6.2
2. Weka 3.6
3. Hadoop 0.19.0
4. Hadoop On Demand
5. Sun Grid Engine

All the python scripts being called by the shell script, deletionList.sh are shipped with it. deletionList.sh accepts two inputs:

1. HDFS Log File as shown in Figure 4.7. Size of log file for each day is in the range of 1.5GB to 4GB. In this study, we have considered log files for one month for each experiment.
2. Data Set Dump File as shown in Figure 4.8.

Final output gives a report of the best data sets to be deleted along with their sizes and the total number of files which were classified to be deleted. Figure 4.9 gives an example of the Final Output Report.

```

2009-04-29 00:02:03,477 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit:
ugi=root,root,bin,daemon,sys,adm,disk,wheel ip=/172.16.100.8 cmd=setReplication
src=/user/uscms01/pnfs/unl.edu/data4/cms/store/unmerged/mc/Summer08/Summer08/QCD_EMenric
hed_Pt30to80/GEN-SIM-RECO/IDEAL_V6_RECO_v1/0001/CC2321B8-AF75-DD11-AEC9-
00E081300824.root dst=null perm=null

2009-04-29 00:02:03,477 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit:
ugi=root,root,bin,daemon,sys,adm,disk,wheel ip=/172.16.100.8 cmd=setReplication
src=/user/uscms01/pnfs/unl.edu/data4/cms/store/unmerged/mc/Summer08/Summer08/QCD_EMenric
hed_Pt30to80/GEN-SIM-RECO/IDEAL_V6_RECO_v1/0001/E039E806-9D75-DD11-A1B4-
001E68041C1F.root dst=null perm=null

2009-04-29 00:02:03,478 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem.audit:
ugi=root,root,bin,daemon,sys,adm,disk,wheel ip=/172.16.100.8 cmd=setReplication
src=/user/uscms01/pnfs/unl.edu/data4/cms/store/unmerged/mc/Summer08/Summer08/QCD_EMenric
hed_Pt30to80/GEN-SIM-RECO/IDEAL_V6_RECO_v1/0001/EE8BC3FF-C275-DD11-8F9F-
00188B7AC633.root dst=null perm=null

```

Figure 4.7: Contents of Input HDFS Log File.

```

PATH  FILES_FILESIZE FILES_LOGICALFILENAME
-----
/Zprime_ttbar_semileptonic_m4000_w40_RECO/CMSSW_2_1_8-Madgraph-FullSim-IDEAL_V9-
10TeV/RECO 3537184170
/store/results/icmstop/mc/BoostedTop/2008/CMSSW_2_1_8/fullsim/Zprime_ttbar_s1semi4000w40_1
0TeV_reco/s1semi4000w40_5000evt_10TeV_50442_50_99_RECO.root

/Zprime_ttbar_semileptonic_m4000_w40_RECO/CMSSW_2_1_8-Madgraph-FullSim-IDEAL_V9-
10TeV/RECO 3530692439
/store/results/icmstop/mc/BoostedTop/2008/CMSSW_2_1_8/fullsim/Zprime_ttbar_s1semi4000w40_1
0TeV_reco/s1semi4000w40_5000evt_10TeV_50442_0_49_RECO.root

/Zprime_ttbar_semileptonic_m4000_w40_RECO/CMSSW_2_1_8-Madgraph-FullSim-IDEAL_V9-
10TeV/RECO 3540767474
/store/results/icmstop/mc/BoostedTop/2008/CMSSW_2_1_8/fullsim/Zprime_ttbar_s1semi4000w40_1
0TeV_reco/s1semi4000w40_5000evt_10TeV_33749_50_99_RECO.root

/Zprime_ttbar_semileptonic_m4000_w40_RECO/CMSSW_2_1_8-Madgraph-FullSim-IDEAL_V9-
10TeV/RECO 3538449610
/store/results/icmstop/mc/BoostedTop/2008/CMSSW_2_1_8/fullsim/Zprime_ttbar_s1semi4000w40_1
0TeV_reco/s1semi4000w40_5000evt_10TeV_33749_150_199_RECO.root

/Zprime_ttbar_semileptonic_m4000_w40_RECO/CMSSW_2_1_8-Madgraph-FullSim-IDEAL_V9-
10TeV/RECO 3527999496
/store/results/icmstop/mc/BoostedTop/2008/CMSSW_2_1_8/fullsim/Zprime_ttbar_s1semi4000w40_1
0TeV_reco/s1semi4000w40_5000evt_10TeV_33749_100_149_RECO.root

```

Figure 4.8: Contents of Input Data set Dump.

/TkAlCosmics0T/Summer08_COSMMC_21X_v1_v1/GEN-SIM-RAW	207	3581423897
/TTJets-madgraph/Fall08_IDEAL_V11_redigi_v10/GEN-SIM-RECO	53	1245891207
/QCD_BCtoMu_Pt20/Summer08_IDEAL_V9_v1/GEN-SIM-RECO	38	2017869978
/Wc-madgraph/Fall08_IDEAL_V9_reco-v1/GEN-SIM-RECO	26	1255435460
/InclusiveMu5Pt50/Summer08_IDEAL_V9_v1/GEN-SIM-RECO	18	2263565803
/InclusiveMuPt15/Summer08_IDEAL_V9_v1/GEN-SIM-RECO	17	2547074495
/Zjets-madgraph/Fall08_IDEAL_V9_reco-v2/GEN-SIM-RECO	5	1200519543
/QCD_pt_0_15/SAM_IDEAL_V9_SAM/GEN-SIM-RAW-RECO	4	1074702886
/QCD_BCtoE_Pt20to30/Summer08_IDEAL_V9_PAT_v1/USER	3	1070378779
/WJets-madgraph/Fall08_IDEAL_V9_v1/GEN-SIM-RECO	3	2607059710
/AstarJets-madgraph/Fall08_IDEAL_V9_v2/GEN-SIM-RECO	2	1089224947
/TkAlCosmics4T/Summer08_COSMMC_21X_v1_v1/GEN-SIM-RAW	2	3236543319

Figure 4.9: Contents of Final Deletion Report.

4.4 Comparing SVM based Methodology with Retention Cost Heuristic

A few data sets were picked randomly and as discussed in Section 4.4 the R_C for each of these data sets was calculated.

The Threshold value T_{R_C} was arrived at by using the binary search technique, i.e. searched for an R_C value which predicted deletion with high accuracy. If a data set predicted for deletion was not accessed in the following one month time frame, the accuracy was considered high. Thus, T_{R_C} was computed to be 16.0 which is an approximation of 15.675. 219 data sets are subscribed by Nebraska, 14 data sets were randomly picked. HDFS logs for March 2010 were considered. Deletion predictions from SVM methodology and R_C heuristic were computed and their accuracies were compared which resulted in Table 4.6.

Table 4.5: Retention Cost Metric calculated for some randomly picked data sets.

Data Set	Size (GB)	D_{LA}	R_C
QCD_EMEnriched_Pt80to170/Summer09-MC_31X_V3_7TeV-v1/GEN-SIM-RECO	1.040860589	22	22.89893296
QCD_EMEnriched_Pt30to80/Summer09-MC_31X_V3_7TeV-v1/GEN-SIM-RECO	1.32829925	7	9.29809475
TTbarJets-madgraph/Summer09-MC_31X_V3_7TeV-v2/GEN-SIM-RECO	2.001488867	5	10.00744434
ZJets-madgraph/Summer09-MC_31X_V3_7TeV-v2/GEN-SIM-RECO	2.030603647	4	8.122414588
PhotonJet_Pt170to300/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	1.668406269	17	28.36290657
PhotonJet_Pt30to50/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	1.690302816	8	13.52242253
PhotonJet_Pt80to120/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	1.608302404	10	16.08302404
PhotonJet_Pt300to500/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	1.508702934	6	9.052217604
PhotonJet_Pt120to170/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	1.279241541	7	8.954690787
Zee/Summer09-MC_31X_V3_7TeV_TrackingParticles-v1/GEN-SIM-RECO	1.996696187	11	21.96365806
PhotonJet_Pt20to30/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	1.70895395	3	5.12686185
WJets-madgraph/Summer09-MC_31X_V3-v1/GEN-SIM-RECO	2.162174109	13	28.10826342
PhotonJet_Pt50to80/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	1.221273414	1	1.221273414
PhotonJet_Pt15to20/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	1.583614414	1	1.583614414

Table 4.6: Comparing SVM methodology with Retention Cost Heuristic.

Data Set	SVM Deletion Prediction	R_C Deletion Prediction	Not Accessed in April, 2010
QCD_EMEnriched_Pt80to170/Summer09-MC_31X_V3_7TeV-v1/GEN-SIM-RECO	Yes	Yes	Yes
QCD_EMEnriched_Pt30to80/Summer09-MC_31X_V3_7TeV-v1/GEN-SIM-RECO	Yes	No	Yes
TTbarJets-madgraph/Summer09-MC_31X_V3_7TeV-v2/GEN-SIM-RECO	No	Yes	Yes
ZJets-madgraph/Summer09-MC_31X_V3_7TeV-v2/GEN-SIM-RECO	Yes	No	No
PhotonJet_Pt170to300/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	No	No	No
PhotonJet_Pt30to50/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	Yes	No	Yes
PhotonJet_Pt80to120/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	No	Yes	No
PhotonJet_Pt300to500/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	No	No	No
PhotonJet_Pt120to170/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	No	No	No
Zee/Summer09-MC_31X_V3_7TeV_TrackingParticles-v1/GEN-SIM-RECO	No	Yes	No
PhotonJet_Pt20to30/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	No	No	No
WJets-madgraph/Summer09-MC_31X_V3-v1/GEN-SIM-RECO	No	Yes	No
PhotonJet_Pt50to80/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	No	No	No
PhotonJet_Pt15to20/Summer09-MC_31X_V9_7TeV-v1/GEN-SIM-RECO	No	No	No



Figure 4.10: Performance of different classifiers against test data J

According to Table 4.6, twelve out of fourteen data sets had accurate predictions from SVM Methodology whereas only nine out of fourteen data sets had accurate predictions from R_C Heuristic. This experiment was repeated for different sets of data sets and resulted in SVM methodology having more accuracy for each repetition.

4.5 Classifier A's accuracy against test set J

Classifier *A* was built using training sets from February 2009. We tested the performance of this classifier against test set *J* from March 2010 which is more than a year later. Figure 4.10 shows that the classification accuracy dropped for classifier *A* which had the best classification accuracy for February, March and April of 2009. The combined classifier had slightly better accuracy. A new classifier *K*, developed in March 2010 seemed to be the best for test data *J*.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

Categorizing data sets by SVM methodology expedites and makes objective management of large data sets. It is also highly scalable and also improves the accuracy in categorizing. From this study it can be concluded that this problem is a good fit for machine learning using Support Vector Machines. Python scripts used as mappers and reducers that implemented algorithms with complexity $O(n)$ worked well with Hadoop Streaming and were successful in utilizing the processing capacity of the Hadoop MapReduce framework. Nine different training sets were so created. Using various statistics their classification performance was evaluated and the classifier with best the performance was chosen. From HDFS log files, attribute relation format files were created and used with Weka where the classifier helps in categorizing the data sets to two categories, namely “delete” and “save”. Another methodology which used the Retention Cost heuristic for categorizing data sets was developed and compared with the SVM methodology. SVM methodology proved to have higher accuracy and outperformed the Retention Cost heuristic. However, a new classifier needs to be cre-

ated when a performance drop is observed. The Classifier developed from training set *A* (February, 2009) was shown to have the best classification accuracy, 97.4 percent, for data from dates up to April 2009. When this classifier was tested against test data developed from HDFS logs of March, 2010 its classification accuracy dropped to 44 percent. A new classifier *K* from training set *K* (March, 2010) had to be developed for classifying test data *J* (March, 2010) for higher classification accuracy of 94 percent.

5.2 Future Work

This study used sequential minimal optimization to find the separating planes. Other choices such as quadratic programming and least squares should also be tested. Classification using other Machine Learning algorithms like decision tree learning, association rule learning, artificial neural networks, clustering and bayesian networks could be tested to see if higher classification accuracy could be achieved. Sampling of file names were random in this study, using different sampling methods and repeating the experiments might be useful in developing a classifier with more accuracy. More sophisticated data cleaning techniques could be employed. A more complete feature set for training and a thorough investigation into the time frame over which a given classifier remains valid could be complemented. Having a single pair of Map and Reduce algorithms would reduce the number of input data scans from four to two. To be able to solve similar problems involving large data sets a generic algorithm could be designed which could take the desired features as inputs and output the values associated with those features. This would involve more sophisticated log parsing techniques.

Bibliography

- [1] “Module 4: MapReduce.” <http://developer.yahoo.com/hadoop/tutorial/module4.html>. [Online; accessed 8-September-2009].
- [2] T. White, *Hadoop: The Definitive Guide*. O’Reilly Media, 1 ed., June 2009.
- [3] T. Fawcett, *ROC Graphs: Notes and Practical Considerations for Data Mining Researchers*. HP Laboratories Palo Alto, January 2003. Copyright Hewlett-Packard Company 2003.
- [4] K. Bloom, “US CMS Tier-2 Computing,” in *International Conference on Computing in High Energy and Nuclear Physics (CHEP07)*, Journal of Physics: Conference Series 119 (2008) 052004, (Lincon, NE 68588-0111, USA), Department of Physics and Astronomy, University of Nebraska-Lincoln, IOP Publishing, 2008.
- [5] “Apache Hadoop.” <http://hadoop.apache.org/>. [Online; accessed 10-August-2009].
- [6] A. W. Moore, *Support Vector Machines*. School of Computer Science, Carnegie Mellon University, Nov 2001.
- [7] I. H. Witten and E. Frank, *Data mining: practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann Publisher, second edition ed., 2005.

- [8] M. P. S. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. W. Sugnet, T. S. Furey, M. Ares, Jr., and D. Haussler, “Knowledge-based Analysis of Microarray Gene Expression Data By Using Support Vector Machines.” <http://www.cs.iastate.edu/~honavar/geneexpression-svm.pdf>, 2000.
- [9] S. Idicula-Thomas, A. J. Kulkarni, B. D. Kulkarni, V. K. Jayaraman, and P. V. Balaji, “A support vector machine-based method for predicting the propensity of a protein to be soluble or to form inclusion body on overexpression in escherichia coli,” *Bioinformatics*, vol. 22, no. 3, pp. 278–284, 2006.
- [10] C. King, Christophe, and V. J. Weng, “A SVM Approach to Stock Trading.” <http://www.stanford.edu/class/cs229/proj2009/KingVandrotWeng.pdf>.
- [11] A. Ammari and V. Zharkova, “Combining Tag Cloud Learning with SVM Classification to achieve Intelligent Search for Relevant Blog Articles.” <http://www.socialgamingplatform.com/msm09/proceedings/paper7.pdf>.
- [12] S. Loebman, D. Nunley, Y. Kwon, B. Howe, M. Balazinska, and J. P. Gardner, “Analyzing Massive Astrophysical Datasets: Can Pig/Hadoop or a Relational DBMS Help?.” <http://nuage.cs.washington.edu/pubs/iasds09.pdf>.
- [13] A. Cary, Z. Sun, V. Hristidis, and N. Rishe, “Experiences on Processing Spatial Data with MapReduce.” <http://users.cis.fiu.edu/~vagelis/publications/Spatial-MapReduce-SSDBM2009.pdf>, 2009.
- [14] S. Matthews and T. Williams, “MrsRF: an efficient MapReduce algorithm for analyzing large collections of evolutionary trees,” *BMC Bioinformatics*, vol. 11, no. Suppl 1, p. S15, 2010.

- [15] D. Gillick, A. Faria, and J. Denero, “MapReduce: Distributed Computing for Machine Learning.” http://www.icsi.berkeley.edu/~arlo/publications/gillick_cs262a_proj.pdf, 2006.
- [16] M. Kaczmarek, T. Jiang, and D. A. Pease, “Beyond backup toward storage management,” *IBM Syst. J.*, vol. 42, no. 2, pp. 322–337, 2003.
- [17] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*. John Wiley and Sons, Inc, 7th ed., 2005.
- [18] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, “LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies,” *IEEE TRANSACTIONS ON COMPUTERS*, vol. 50, pp. 1352–1361, December 2001.
- [19] P. R. Jelenkovic and A. Radovanovic, “Least-Recently-Used caching with dependent requests,” *Theoretical Computer Science*, vol. 326, pp. 293–327, October 2004.
- [20] J. Dilley, M. Arlitt, and S. Perret, “Enhancement and Validation of Squid’s Cache Replacement Policy,” *Internet Systems and Applications Laboratory*, vol. HP Laboratories Palo Alto, May 1999.
- [21] M. Arlitt, R. Friedrich, and T. Jin, “Workload Characterization of a Web Proxy in a Cable Modem Environment,” tech. rep., ACM Performance Evaluation Review, 1999.
- [22] P. J. Denning, “The working set model for program behavior,” *Communications of the ACM*, vol. 11, pp. 323–333, May 1968.

- [23] J. Edward G. Coffman and P. J. Denning, *Operating Systems Theory*. Prentice Hall Professional Technical Reference, 1973.
- [24] A. V. Aho, P. J. Denning, and J. D. Ullman, “Principles of Optimal Page Replacement,” *J. ACM*, vol. 18, no. 1, pp. 80–93, 1971.
- [25] A. S. Tanenbaum, *Modern Operating Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.
- [26] S. Podlipnig and L. Böszörményi, “A survey of Web Cache Replacement Strategies,” *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, 2003.
- [27] P. Jayarekha and T. R. G. Nair, “An Adaptive Dynamic Replacement Approach for a Multicast based Popularity Aware Prefix Cache Memory System,” *CoRR*, vol. abs/1001.4135, 2010.
- [28] S. Jin and A. Bestavros, “Popularity-Aware Greedy Dual-Size Web Proxy Caching Algorithms,” in *ICDCS '00: Proceedings of the The 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, (Washington, DC, USA), p. 254, IEEE Computer Society, 2000.
- [29] M. Conti, M. Kumar, S. K. Das, and B. A. Shirazi, “Quality of Service Issues in Internet Web Services,” *IEEE Transactions on Computers*, vol. 51, pp. 593–594, 2002.
- [30] L. Rizzo and L. Vicisano, “Replacement Policies for a Proxy Cache,” *IEEE/ACM Transactions on Networking*, vol. 8, pp. 158–170, 1998.
- [31] O. Ercetin and L. Tassiulas, “Pricing strategies for differentiated services content delivery networks,” *Comput. Netw.*, vol. 49, no. 6, pp. 840–855, 2005.

- [32] I. Ari, A. Amer, E. Miller, S. Brandt, and D. Long, “Who Is More Adaptive? ACME: Adaptive Caching using Multiple Experts (Ext. Abstract),” in *In Workshop on Distributed Data and Structures (WDAS)*, pp. 143–158, 2002.
- [33] P. Schwan, “Lustre: Building a File System for 1,000-node Clusters,” in *Proceedings of the Linux Symposium*, 2003.
- [34] M. Ernst, P. Fuhrmann, M. Gasthuber, T. Mkrtchyan, and C. Waldman, “dCache, a distributed storage data caching system,” September 2001.
- [35] F. Schmuck and R. Haskin, eds., *GPFS: A Shared-Disk File System for Large Computing Clusters*, (San Jose, CA), IBM Almaden Research Center, Proceedings of the Conference on File and Storage Technologies (FAST02), January 2002.
- [36] Wikipedia, “IBM General Parallel File System — Wikipedia, The Free Encyclopedia.” http://en.wikipedia.org/w/index.php?title=IBM_General_Parallel_File_System&oldid=358425304", 2010. [Online; accessed 30-April-2010].
- [37] S. Ghemawat, H. Gobioff, , and S.-T. Leung, “The Google File System,” *ACM*, vol. Proceedings of the 19th ACM Symposium on Operating Systems Principles, pp. 29–43, 2003.
- [38] P. H. Carns, W. B. Ligon, III, R. B. Ross, and R. Thakur, “PVFS: A Parallel File System for Linux Clusters,” in *In Proceedings of the 4th Annual Linux Showcase and Conference*, pp. 317–327, USENIX Association, 2000.
- [39] “Powered By Hadoop.” <http://wiki.apache.org/hadoop/PoweredBy>. [Online; accessed 28-July-2010].

- [40] I. Cloudera, “Cloudera.” <http://www.cloudera.com/>, 2010. [Online; accessed 30-April-2010].
- [41] R. Journey, “Linkedin Blog.” <http://blog.linkedin.com/tag/hadoop/>. [Online; accessed 28-July-2010].
- [42] B. Bockelman, “High Energy Hadoop.” <http://www.cloudera.com/blog/2009/05/01/high-energy-hadoop/>. [Online; accessed 23-September-2009].
- [43] Wikipedia, “Support Vector Machine — Wikipedia, The Free Encyclopedia.” http://en.wikipedia.org/w/index.php?title=Support_vector_machine&oldid=329070374, 2009. [Online; accessed 2-December-2009].
- [44] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer, N.Y., 1995.
- [45] V. Jakkula, “T.”
- [46] P. Khati, “Comparative Analysis of Protein Classification Methods,” a masters thesis, Department of Computer Science and Engineering, University of Nebraska - Lincoln, Lincoln, NE, 2004.
- [47] K.-W. Cheung, J. T. Kwok, M. H. Law, and K.-C. Tsui, “Mining customer product ratings for personalized marketing,” *Decis. Support Syst.*, vol. 35, no. 2, pp. 231–243, 2003.
- [48] Wikipedia, “Mercer’s condition — Wikipedia, The Free Encyclopedia.” http://en.wikipedia.org/w/index.php?title=Mercer%27s_condition&oldid=308387225, 2009. [Online; accessed 5-January-2010].
- [49] N. Aronszajn, “Theory of reproducing kernels,” 686, pp. 337–404, *Trans. Amer. Math. Soc*, N. Aronszajn, 1950.

- [50] R. Rosipal and L. J. Trejo, “Kernel partial least squares regression in reproducing kernel hilbert space,” *J. Mach. Learn. Res.*, vol. 2, pp. 97–123, 2002.
- [51] K. M. Cnrs, Y. Kodratoff, and S. Moscatelli, “Machine Learning For Object Recognition and Scene Analysis,” *Internationa Journal of Pattern recognition and AI*, vol. 8, pp. 8–1, 1994.
- [52] “Hadoop MapReduce.” <http://hadoop.apache.org/mapreduce/>. [Online; accessed 10-August-2009].
- [53] “Apache Hadoop.” <http://wiki.apache.org/hadoop/>. [Online; accessed 12-August-2009].
- [54] “Hadoop Streaming.” <http://hadoop.apache.org/common/docs/current/streaming.html>. [Online; accessed 30-August-2009].
- [55] “HDFS Architecture.” http://hadoop.apache.org/common/docs/current/hdfs_design.html. [Online; accessed 23-September-2009].
- [56] S. Paranjpye and M. Konar, “Grid Computing at Yahoo!,” tech. rep.
- [57] “Hadoop On Demand.” <http://hadoop.apache.org/common/docs/r0.17.1/hod.html>. [Online; accessed 5-November-2009].
- [58] M. Lutz, *Learning Python*. O’Reilly, third ed., October 2007.
- [59] Wikipedia, “Python (programming language) — Wikipedia, The Free Encyclopedia.” [http://en.wikipedia.org/w/index.php?title=Python_\(programming_language\)&oldid=346759876](http://en.wikipedia.org/w/index.php?title=Python_(programming_language)&oldid=346759876), 2010. [Online; accessed 28-January-2010].
- [60] D. Mertz, *Text Processing in Python*. Addison Wesley.

- [61] Wikipedia, “Weka (machine learning) — Wikipedia, The Free Encyclopedia.” [http://en.wikipedia.org/w/index.php?title=Weka_\(machine_learning\)&oldid=338751970](http://en.wikipedia.org/w/index.php?title=Weka_(machine_learning)&oldid=338751970), 2009. [Online; accessed 25-December-2009].
- [62] J. L. Lustgarten, V. Gopalakrishnan, H. Grover, and S. Visweswaran, “Improving Classification Performance with Discretization on Biomedical Datasets,” *AMIA Annual Symposium Proceedings*, vol. 2008, no. PMC2656082, pp. 445–449, 2008.
- [63] I. H. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. J. Cunningham, “Weka: Practical Machine Learning Tools and Techniques with Java Implementations,” in *Proc ICONIP/ ANZIIS/ANNES99 Future Directions for Intelligent Systems and Information Sciences*, pp. 192–196, Morgan Kaufmann, 1999.
- [64] Wikipedia, “Sun Grid Engine — Wikipedia, The Free Encyclopedia.” http://en.wikipedia.org/w/index.php?title=Sun_Grid_Engine&oldid=339082418, 2010. [Online; accessed 1-January-2010].
- [65] H. C. Center, “Prairiefire.” <http://hcc.unl.edu/prairiefire/>, 2010. [Online; accessed 1-January-2010].
- [66] “Attribute-Relation File Format (ARFF).” <http://www.cs.waikato.ac.nz/~ml/weka/arff.html>, November 2008. [Online; accessed 22-December-2009].
- [67] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning)*. The MIT Press, 1st ed., December 2001.
- [68] M. A. Hearst, “Support Vector Machines,” *IEEE Intelligent Systems*, vol. 13, pp. 18–28, 1998.

- [69] A. Moore, “Cross-validation for detecting and preventing overfitting.” <http://www.autonlab.org/tutorials/overfit09.pdf>.
- [70] J. Shao, “Linear Model Selection by Cross-Validation,” *Journal of the American Statistical Association*, vol. 88, no. 422, pp. 486–494, 1993.
- [71] F. Provost, T. Fawcett, and R. Kohavi, “The Case Against Accuracy Estimation for Comparing Induction Algorithms,” in *In Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 445–453, Morgan Kaufmann, 1997.
- [72] V. Kuznetsov, D. Evans, and S. Metson, “The CMS data aggregation system,” *Procedia Computer Science*, vol. 1, 2010.