University of Crete

Department of Computer Science

Implementing Feature Selection Algorithms for Big Data

by

Panagiotis Tzirakis (AM 714)

Supervisor:

Prof. Ioannis Tsamardinos

Submitted in partial fulfilment of the requirements for the MSc Degree in Computer Science of University of Crete

October 29, 2015

Abstract

In recent years, data has an exponential growth in both the number of instances and the number of features, which brings their scale to the level of terabytes. These amounts of data can be found in many machine learning applications like information retrieval, text categorization and image retrieval. Although such amounts of data are frequent nowadays, classical machine learning algorithms have difficulties to process them.

An important task in machine learning is feature selection and its task is to select the most informative features in a dataset. Feature selection is effective in reducing dimensionality, removing irrelevant features, increasing performance of a learner, and improving our understanding of the model. With the increase of the volume of the data the usability of classical feature selection algorithms deteriorates.

To solve scalability problems, the Map-Reduce model has been proposed. With this model the data can be processed in parallel, and so machine learning algorithms can now be adapted in order to process terabytes of data.

In this thesis we were concerned with the implementation of a feature selection algorithm for big data. More particularly, we used the Map-Reduce model to parallelize the Max-Min Parent and Children (MMPC) algorithm in order to be able to handle big data. MMPC tries, heuristically, with the use of independent tests, to filter out variables. For this thesis we show how two independence tests that can handle categorical and continuous features, can be used with the Map-Reduce model. Finally, we also use a method so that MMPC can be used with any independence test using the Map-Reduce model.

To evaluate our algorithm, we experimented with datasets that contained different number of instances and features. The experimental evaluation showed that our algorithm scales well with these datasets when varying the number of instances and the number of nodes in the cluster. Moreover, the performance of the algorithm is comparable to other feature selection algorithms.

Περίληψη

Στις μέρες μας, τα δεδομενα αυξάνονται εκθετικά τόσο στον αριθμό των δειγματων οσο και στον αριθμό των μεταβλητών, με το μέγεθος τους να φτάνει την κλίμακα των Terabyte. Αυτός ο όγκος δεδομένων μπορεί να βρεθεί σε πολλές εφαρμογές της μηχανικής μάθησης όπως στην ανάκτηση πληροφοριών, κατηγοριοποίηση κειμένου και ανάκτηση εικόνων. Παρόλο που τέτοιου είδους δεδομένα είναι συχνά σήμερα, κλασσικοί αλγόριθμοι μηχανικής μάθησης δεν μπορούν να τα διαχειριστούν.

Μια πολύ σημαντική μέθοδος στην μηχανική μάθηση ειναι η επιλογή μεταβλητών που προσπαθεί να επιλέξει τις μεταβλητές που είναι πιο προβλεπτικές σε ένα σετ δεδομένων. Η επιλογή μεταβλητών είναι σημαντική καθώς μειώνει τις διαστάσεις των δεδομένων, αφαιρεί άσχετες μεταβλητές, αυξάνει την επίδοση ενός ταξινομητή και βοηθάει στην καλύτερη κατανόηση των δεδομένων. Με την αύξηση του όγκου των δεδομένων η απόδοση των κλασσικών αλγόριθμων επιλογής μεταβλητών μειώνεται αισθητά.

Για να λυθούν προβλήματα απόδοσης, το μοντέλο Map-Reduce έχει προταθεί. Πλέον μπορεί να γίνει η επεξεργασία των δεδομένων παράλληλα σε ενα σύμπλεγμα υπολογιστών και οι αλγόριθμοι μηχανικής μάθησης μπορούν να τροποποιηθούν έτσι ώστε να είναι σε θέση να επεξεργαστούν μεγάλο όγκο δεδομένων.

Σε αυτή την εργασία ασχοληθήκαμε με την υλοποίηση αλγορίθμων επιλογής μεταβλητών. Πιο συγκεκριένα, χρισιμοποιήσαμε το Map-Reduce μοντέλο για να παραλληλοποιήσουμε τον αλγόριθμο Max Min Parent and Children (MMPC) έτσι ώστε να μπορεί να διαχειριστεί μεγάλο όγκο δεδομένων. Ο αλγόριθμος αυτός προσπαθεί ευριστικά, με τη χρήση τεστ ανεξαρτησίας, να βρει εξαρτήσεις μεταξύ μεταβλητών. Σε αυτή την εργασία δείχνουμε πως παραλληλοποιήσαμε δύο τεστ ανεξαρτησίας που μπορούν να διαχειριστούν κατηγορικές και συνεχείς μεταβλητές, χρησιμοποιώντας το μοντέλο Map-Reduce. Τέλος, χρησιμοποιήσαμε μία μέθοδο με την οποία ο MMPC μπορεί να χρησιμοποιηθεί με οποιοδήποτε τεστ.

Για να αξιολογήσουμε τον αλγόριθμο χρησιμοποιήσαμε δεδομένα που περιέχουν διαφορετικό αριθμό δειγμάτων και μεταβλητών. Η αξιολόγηση έδειξε ότι ο αλγόριθμος μας κλιμακώνεται καλα οταν αλλάζει ο αριθμός των δειγμάτων και ο αριθμός των κόμβων του δικτύου. Τέλος, η απόδοση του αλγορίθμου είναι συγκρίσιμη με την απόδοση άλλων αλγορίθμων επιλογής μεταβλητών.

Acknowledgements

I would like, first, to thank my supervisor Ioannis Tsamardinos for his help in completing this thesis. Moreover, I would like to thank Mr Borbudakis for our long conversations we had. Finally, I would not have succeeded anything in my life if there was not my family by my side.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Description

In this thesis we are concerned with implementing feature selection algorithms for big data. Most regular feature selection algorithms are not able to cope with big data as their performance decreases significantly. However, it is important to have efficient algorithms that can extract features from such data as this will result in better models and better understanding of the data.

The feature selection algorithm we deal with is called Max Min Parent and Children (MMPC) [43]. Our goal was to expand its usability so it can handle big data efficiently. We chose to expand MMPC for the following reasons:

- MMPC has been proven to be a robust feature selection algorithm for regular datasets. By expanding its usability, we can have a robust algorithm for big data.
- MMPC can be used with any kind of data, if the appropriate statistical test is used.
- MMPC can be easily expanded to extract multiple sets of features with same predictive power. As far as we are concerned, there is no such algorithm for big data and so it would be the first of its kind.
- MMPC has also been used for the creation of the skeleton of Bayesian Networks. This has been accomplished by using MMPC on all the features of the data. So by applying MMPC for big data a Bayesian Skeleton can be created.

To process big data we use the Map-Reduce framework. This framework processes the data in parallel and then combines some intermediate results to get the final output. We chose to use this framework as it is the most well known and most used for performing big data analytics. The most popular platform that implements this framework is called Hadoop, and we use this platform for our purposes.

Using this framework, we expanded the usability of MMPC to handle big data. Due to the fact that MMPC uses independent tests to extract features, different tests need different implementation. In this thesis, we implemented two independent tests. These are the Fisher Z test and the G^2 test. The first is used when the data contain continuous features and the later when the data contain discrete features. Morevover, it is possible

for someone to use MMPC without implementing a statistical test with the Map-Reduce framework. This can be achieved with the Fisher's combined probability test. We describe this method and how we use it in later chapters.

To evaluate our algorithm we used various datasets and a cluster of 7 nodes. The experiments we performed are the following:

- Instances vs Time: In these experiments we varied the number of instances in the dataset and used all the nodes in the cluster.
- Nodes vs Time: In these experiments we varied the number of nodes in the cluster and used all the instances in the datasets.
- Classification Accuracy: Here we evaluated our algorithm using the logistic regression classifier and compare the results with another feature selection algorithm for big data. Moreover, we compare the time taken for both algorithms to select features.
- Using Fisher's method: We use the Fisher's method with the G^2 test and compare the results with the ones produced by the actual G^2 test. More particularly, we compare: the number of features selected, the time taken to finish and the accuracy when the logistic regression classifier is used.

To conclude, our algorithm scales well when varying the number of instances and the number of nodes in a cluster. More particularly, the more instances in a dataset the more time takes for the algorithm to finish when keeping the number of nodes fix in a cluster. In addition, increasing the number of nodes in a cluster and keeping the size of the data fix, the execution time decreases.

1.2 Thesis Structure

The structure of this thesis is the following:

- In chapter 2, we present an introduction to big data. We present the common platforms to perform big data analytics and we describe in more detail the platform Hadoop, which is used for this thesis.
- In chapter 3, we present the background needed for the thesis. We discuss about feature selection algorithms and their categories, and we describe the MMPC algorithm. Finally, we present the related work that has been done until now.
- In chapter 4, we discuss how we used the Map-Reduce model with MMPC and we show how two independent tests can be used. Moreover, we present Fisher's method that can be used with MMPC in order to handle any type of data.
- In chapter 5, we show the results from our experiments. We explain in detail how these results occurred and how they can be interpreted. Moreover, we compare our algorithm with a well known algorithm for big data analytics.

- In chapter 6, we provide our conclusions and the future work. It is a short summary of the most important features of this work and it also presents some more possibilities of this algorithm.

Chapter 2

Big Data

2.1 Introduction

We are living in the big data era, where huge amounts of data are being generated and stored daily. Facebook has over 300 petabytes worth of material from its users and Youtube has to analyze more than 500 petabytes of data. Thompson Reuters News Analytics performed a research and estimated that 35 zettabytes (1 zettabyte = 10^3 exabytes = 10^6 pettabytes) of data will be generated in 2020[1].

However, big data is not only about the size of the data. There are other important characteristics of big data such as data variety, data veracity and data velocity. The following four Vs constitute a comprehensive definition of big data and are described in more detail below:

- Volume: This is the primary characteristic of big data and refers to the large size of the data that can be terabytes or petabytes, as described earlier.
- Variety: Nowadays data are collected from a great variety of sources and can be structured or unstructured and can be of different type/format. That is, it may not always be numbers but may be emails, website links or even pictures.
- Velocity: Refers to the fast rate at which data is being generated, communicated, and stored. Sometimes it is vital to process the data in real-time. Some examples can be found in tracking the visitors of a website in real-time or real-time consumer reaction to events or advertisements via Twitter.
- Veracity: Refers to the uncertainty of the data. Not all data are credible or up to date. Some of the data may be biased or contain noise.

Big data poses a data processing problem and more particularly it can be referred as a data intensive problem. In these kinds of problems the calculations are few and the data cannot be loaded in the main memory. In order to solve scalability and performance problems industries build data centers, which may contain hundreds or thousands of interconnected computers, in order to analyze their data in parallel.

[1]Source: "Thomson Reuters Blog"

The problem of storing and analyzing vast amounts of data was first dealt by Google. Google proposed the Map-Reduce[1] programming model for processing the data and the Google File System[2] for storing the data. Both the Map-Reduce model and the file system support automatic parallelization for processing and storing data efficiently.

The Map-Reduce model consists of two tasks: the map and the reduce. The user need only define these two tasks in order to process the data. The map task processes key-value pairs and emits intermediate key-value pairs. The values of these pairs are then grouped by the same key and are input to the reduce task. Finally, the reduce task processes these key-value pairs and emits another key-value pair, which is the output of the model.

The Google File System (GFS) was developed by Google in order to solve the storage needs of the company. The goals of this file system was performance, scalability, reliability, and availability. The file system was successfully used by Google's cluster that consisted of thousands machines.

2.2 Platforms for Analyzing Big Data

Nowadays, many other companies and organizations followed Google's example and created their own platforms and software in order to perform big data analytics. Some of the most popular are the following:

- Dato: Formerly known as GraphLab, the Dato framework provides parallel computation targeted for machine learning algorithms. The advantage of this framework is that it allows a rapid deployment of distributed algorithms. The disadvantage of this platform is that it is not free.

- DryadLINQ is a system that was created by Microsoft to process data in parallel efficiently. Its goal is to make distributed computing for every programmer easy. It generalizes previous execution environments such as SQL, MapReduce, and Dryad in two ways: by adopting an expressive data model of strongly typed .NET objects; and by supporting general-purpose imperative and declarative operations on datasets within a traditional high-level programming language.

- Flink: An open source platform for scalable batch and stream data processing. This platform distributively and efficiently processes data using powerful programming abstractions in Java and Scala, and automatic program optimization. It has native support for iterations, and incremental iterations. Finally, it supports higher-level tools including a graph processing tool (Gelly) and a Machine Learning Library.

- Hadoop: Hadoop is the most popular platform for analyzing big data and it is used by many big companies such as Google, IBM and Yahoo. For this thesis, we used Hadoop to perform our experiments and due to this fact we describe it in the next section in more detail.

- Spark: This is a fast and general engine for large-scale data processing which provides high-level APIs in Scala, Java, and Python, and an optimized engine that

supports general computation graphs for data analysis. It also supports higher-level tools including Spark SQL for SQL and structured data processing, MLlib which is a scalable machine learning library, GraphX for graphs and graph-parallel computation, and Spark Streaming for streaming applications.

2.3 The Hadoop Platform

The most well-known platform for big data analytics is the open-source distributed data processing platform Apache Hadoop. Hadoop provides a cost-effective way to store and process big data and it is being used worldwide by big companies such as Yahoo and Facebook. The main components of this platform is the Map-Reduce paradigm, which is used to process the data, and Hadoop Distributed File System (HDFS) which is used to store the data in a cluster.

2.3.1 Hadoop Distributed File System (HDFS)

The Hadoop Distributed File System (HDFS) is the core component of Hadoop for storing distributively large amounts of data on a cluster. The main differences from existing distributed file systems is that it is highly fault tolerant and it can be used on low-cost hardware. HDFS is optimized for high throughput and works best when reading and writing large files. One key characteristic of this file system is that the data are stored in blocks. That is, every file is broken into blocks of a fixed size that are stored randomly on the nodes in the cluster. The main goals of the file system are the following:

- Efficiently storing large amounts of data (TB or PB) in a cluster.
- Deal with hardware failures. A file system should store data reliably. In order to deal with failures HDFS uses replication. That is, it stores same blocks in multiple nodes in the cluster.
- Applications that use HDFS are assumed to write a file once and read it many times.

HDFS is based on a master/slave architecture. The master node attaches the NameNode and the Secondary NameNode. The NameNode stores all the metadata for the filesystem across the cluster and there can be only one in a cluster. Moreover, it manages the file system namespace and regulates access to files by clients. The Secondary NameNode serves as a checkpoint mechanism for the NameNode. That is, it stores the state of the NameNode. In case of a failure of the NameNode then in a restart its former state is reconstructed by using the Secondary NameNode.

The slave nodes attach the DataNodes, usually one per node. DataNodes manage the storage of the data in the nodes they are attached on and they perform read and write requests. Also, they perform block creation, deletion, and replication upon instruction from the NameNode. Figure 2.1 shows the architecture of the HDFS.

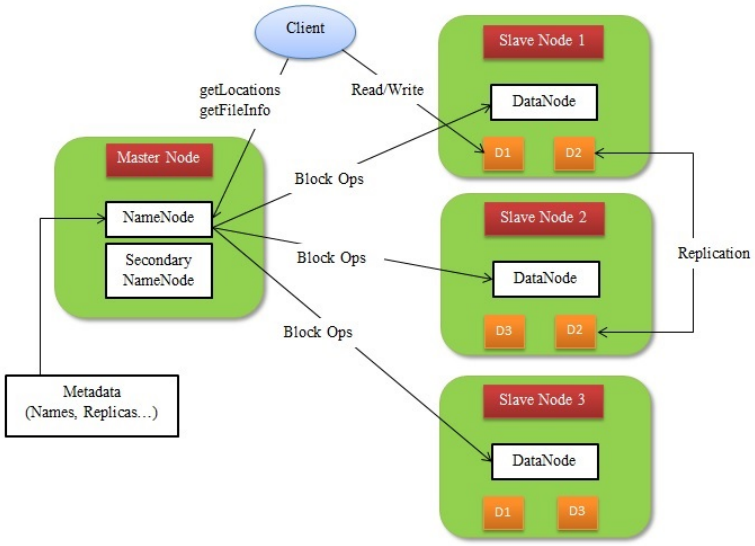

Figure 2.1: HDFS Architecture. NameNode and Secondary NameNode run in master node and DataNode deamons run on slaves. NameNode can perform block operations via DataNodes and clients can make read/write requests. Finally, blocks are replicated in the slaves for fault tolerant issues.

2.3.2 Map-Reduce Model

The Map-Reduce model is used by Hadoop to process the data in parallel and consists of two tasks: the map and the reduce. The key concept on which this model is based on is divide and conquer, where a single task is divided into many independent sub-tasks which are executed in parallel by map tasks and these results are merged in the reduce task for the final outcome.

The mappers and reducers - classes that extend Hadoop-provided base classes to solve a specific problem - are specified by the user. The mapper is necessary to be specified but the reducer need not be. In case the reducer is not specified in a job, the output would be the output of the mapper. The first step of this model is to input the data to the mappers. The output data of the map phase are then sorted and fed to the reduce phase.

The input and output of the map and reduce tasks consist of a list of key-value pairs: (k, v). The output key-value pairs of the map tasks are input to the reduce tasks. Before the pairs are input to the reduce task the system automatically performs a group-by operation on the intermediate key. That is, all the values with the same key are grouped together to form $(k, [v_1, v_2 \dots v_n])$. More particularly, in this phase the following three processes take place:

- Partitioner: As mentioned earlier, map tasks emit key-value pairs which are input to the reduce tasks. The pairs with the same key are input to the same reducer, regardless of which mapper they originated from. The partitioner determines in which reducer a key-value pair will go to.

- Shuffle: This process moves the output of the mappers to the reducers. It is not

necessary for all the mappers to finish in order for this process to start. When one mapper is completed this process starts.

- Sort: This process is responsible for sorting the key-value pairs that are emitted by the mappers.

After the group-by operation, the reduce task is executed with input a list of values associated with each key. The output of the reduce task is also a key-value pair. Figure 2.2 shows the input and output of the map, group-by and reduce phases.

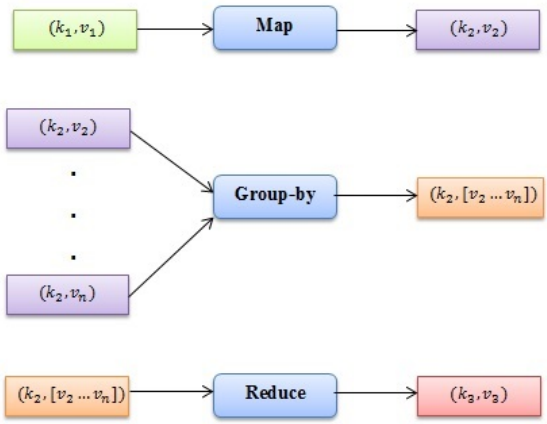

Figure 2.2: Map, Group-by, Reduce Operations. Input and output of Map functions are in the form of key-value pairs. The output values with the same key of each mapper are grouped to the same key and inserted to the Reduce phase which outputs another key-value pair.

The key-value pairs have the following properties:

- Keys are unique but values need not be.
- Values must be associated with one key but keys may not have values.

The main responsibility of the Map-Reduce model is to coordinate the execution. This includes the following:

- Choose appropriate nodes to run map tasks.
- Starting and monitoring the mappers.
- Grouping the key-value pairs emitted by the map tasks.
- Choose appropriate nodes to run reduce tasks.
- Call reduce tasks.
- Starting and monitoring the reducers.

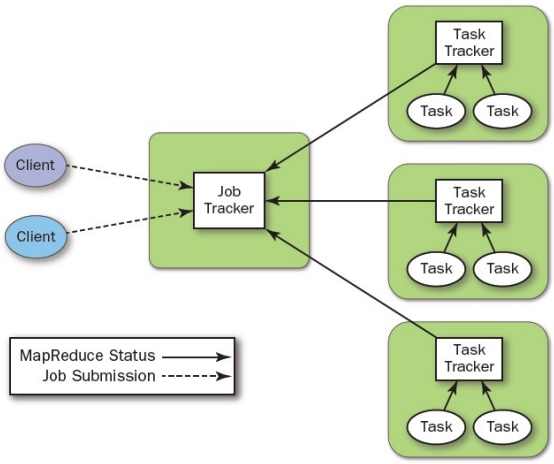

Figure 2.3: Map-Reduce Architecture[4]. JobTracker runs on the master node and a TaskTracker deamon runs on each slave node. The clients submit a job to the JobTracker and it commands the TaskTrackers to start executing the job. Finally, the TaskTrackers send heartbeats to the JobTracker and also the status of the job.

As mentioned earlier, the user need only define just the map and the reduce tasks. The system is responsible for everything else. The size of the data or the size of the cluster does not matter. If the size of the data is 1 gigabyte or 1 petabyte and the number of nodes available in the cluster are 10 or 1000 thousand, Hadoop will determine the best way to utilize all the nodes to perform the work most efficiently. This is one of the strongest advantages of Hadoop.

In the master/slave architecture, the master node runs the JobTracker (which is the coordinator) and a slave node runs the TaskTracker, which is the "worker". The JobTracker communicates with the NameNode in order to specify where the data are stored in the cluster. Afterwards, it determines the number of the map tasks that are needed and the number of TaskTrackers that are available in order to submit a job. Then, the JobTracker commands the TaskTracker to start executing a job. The TaskTracker runs the map task over the data and at the same time it sends "heartbeats" to the JobTracker, that helps the JobTracker to check the job status and the usage of the resources. Figure 2.3 shows the architecture of the model.

2.3.3 Word Count Example

A very common example in Hadoop is the word count problem. In this problem, the task is to count the number of instances of words that appear in some documents, which are the input. As described earlier, the input to the Map-Reduce model should be in the form of key-value pair. For this example, the key is considered the number of line and the value is considered the line of the document. This pair is the input to the mappers. The output is a key-value pair of the form $(word, 1)$. Then, the group-by operation groups the values that have the same key and associates them to that key. So, the output of the process is the pair $(word, [1\ 1...1])$. This is the input to the reduce task which sums the $1s$ for each word and outputs the results in the key-value form: $(word, sum)$.

Figure 2.4 shows an example with two documents. For each document, each line is

the input to the map function. The map function splits each line and outputs as key the word and as value the number one. Hadoop then sorts and groups each key with the value and inputs it to the reduce function. For example, for the word "Apple" the input to the reduce function is $(Apple, [1\ 1\ 1\ 1])$. Finally, the reduce function counts the ones and outputs $(Apple, 4)$.

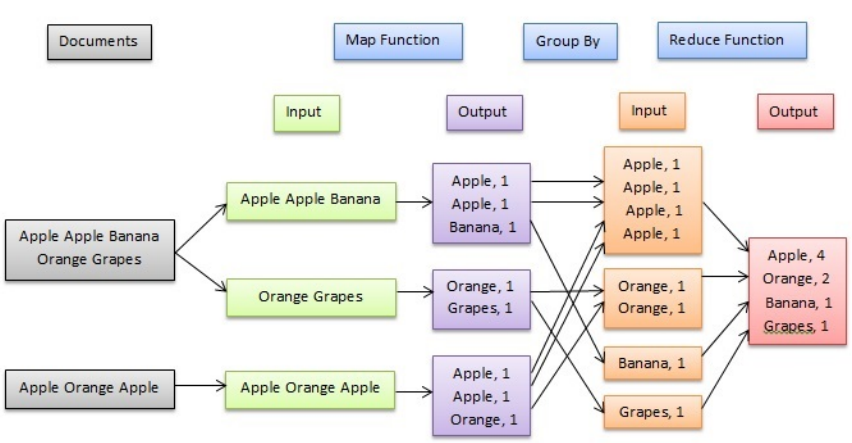

Figure 2.4: Map-Reduce Example [3]. The documents are the input and each line of the document is the input to the Map functions. The mappers split each line to words and extract the key-value pair: $(word, 1)$. Then the Group-By operation groups the values $1s$ to the same key and this is input to the Reduce phase. This phase just sums the $1s$ for each key and outputs the key-value pair: $(word, count)$, where $count$ is the total count of occurances of the $word$ in the documents.

Combiners

Combiners help to optimize Map-Reduce jobs and they are optional. A combiner performs reduce type operations on the output of the mappers before the reducer is executed. Due to the fact that the output of the map tasks is already in memory it makes sense to run the functionality of the reducer on this output. This is performed by combiners and can run on each node and for the same key. However, combiners cannot be used in every job. In order to use combiners the reducer needs to have the following two characteristics:

- Commutative: The order in which the operation processes the values of a key has no effect on the final result.
- Associative: The operation that is applied to the intermediate key value pairs can be applied to any subgroup independently.

In the previous example, the reducer has these two characteristic and so combiners can be used. In this case the input to the reduce function for the word Apple, for example, would be the key-value pair: $(Apple, [2\ 2])$. This is because of the first and third output of the map function.

2.3.4 Applications of Map-Reduce

More and more people are turning to Hadoop to analyze big data in many different areas. One such area is medicine. In [5] they discuss about the challenges and opportunities of big data in the pharmaceutical industry. In this work they propose a platform that integrates and processes clinical data that are collected from various sources. Another example related to medicine can be found in [8], where they describe a new method for creating fetal growth curves and the need for big data techniques in order to scale up the problem.

Much work has also been done in machine learning and more particularly on learning decision trees for big data. A very interesting work was performed by Yin et al. [6], where he describes an algorithm to build regression trees efficiently called "OpenPlanet". A similar work was performed in [7], where they propose two different approaches to parallelize stochastic gradient boosted decision trees. For their first approach they use the Map-Reduce model and for the second they use MPI with Hadoop streaming. Another work related to decision trees is [9], where they describe a process for transforming traditional machine learning algorithms into algorithms for learning distributed data and they apply it to devise algorithms for decision trees. Furthermore, in [10] they propose a new method called MReC4.5 which can be used for classification. This method uses C4.5 decision trees and ensemble methods to achieve ensemble classification.

Several other works have been published in different areas such as computer vision. In [11] they developed a parallel feature extraction algorithm and a parallel averaging stochastic gradient descent to train SVM classifiers. A Map-Reduce algorithm has also been proposed in [12], where they propose a unified framework for predicting attributes and links in social networks, with the use of a two-layer artificial neural network. In addition, an interesting work can be found in [13]. In this work, the authors propose a parallel meta-learning algorithm, to avoid modifying existing machine learning algorithms. Finally, the Map-Reduce model has also been used to implement a variational bayesian probabilistic matrix factorization algorithm [14].

2.3.5 Tools on Top of Hadoop

Hadoop is used massively in the industry and so it is logical to have been expanded in different ways. In [15] the authors developed an open-source data warehousing, called Hive. Hive was created in Facebook in order to make the process of the data easily with Hadoop. It supports queries that are similar to SQL query language. Queries in Hive are compiled using Map-Reduce jobs and the query language is called HiveQL. A similar platform was created in Yahoo! called Pig[16]. This platform also support SQL-like queries that are executed over Hadoop.

One major disadvantage of Hadoop is that it does not support iterations (i.e. passing the data several times), efficiently. The need of efficiently using Hadoop with iterations is imperative. In [17] they proposed Twister, a programming model that supports iterative Map-Reduce computations efficiently. A similar work, that was described in section 2.2, is Spark[18]. Spark is a framework that supports iterations in Map-Reduce. Spark can be up to 100x times faster than Hadoop in memory and 10x faster on disk. A large-scale data warehousing system was build for Spark, called Shark [19]. Shark is compatible is

Apache Hive and supports HiveQL. Moreover, it can be up to 100x times faster than Hive.

A very famous suite that was build on top of Hadoop is Mahout[20]. This suite builds several machine learning algorithms. These include matrix factorization algorithms, clustering techniques like k-means and classification methods like naive Bayes.

Tools that have GUIs have also been build or extended to support Hadoop. One of them is Weka[21]. Weka is a data mining software build on Java and one can run various machine learning algorithms. Weka has been expanded to support Hadoop[22] that can be used easily for various tasks. Another GUI enviroment for Hadoop is Radoop[23]. Radoop is an extension of RapidMiner that can run distributively processes on Hadoop. In addition, it uses data analytics functions from Hive and Mahout. These tools can be easily used by a user to perform big data analytics. The key advantage is that they eliminate the need for learning Hadoop.

2.3.6 Writing Hadoop Applications with R

The main programming language of Hadoop is Java. However, the user can write in any other programming or scripting language, such as Python, Ruby, Perl or C++, with the use of Hadoop streaming. To use this, the language needs to support reading data from standard input and writing to standard output. Independently of the language the main concept is the same. Map and reduce tasks take as input and output key-value pairs.

The language that was used for this thesis is R. R is a widely used programming language for statistical computing and graphics. There are more than 4,400 packages and it is easy to use. R can be integrated with Hadoop with the following ways:

- RHIPE: It stands for R and Hadoop Integrated Programming Environment and was developed by Saptarshi Guha, a former PhD student of Purdue University. The RHIPE package uses the divide and recombine technique to perform data analytics over big data. In this technique, the data is divided into subsets, computation is performed over those subsets by specific R analytics operations and, finally, the output is combined.

- Hadoop streaming: As mentioned earlier, Hadoop streaming is a Hadoop utility for running Hadoop Map-Reduce jobs with executable scripts such as mappers and reducers. The user need to define map and reduce functions to input and output key-value pairs.

- RHadoop: It was developed by Revolution Analytics and it provides large data operations with the R environment. It consists of the following 5 packages:

  - rmr2: This package supports translation of the R language into Hadoop-compliant Map-Reduce jobs.
  - rhdfs: This package provides HDFS usability from R.
  - rhbase: This package supports the database management for HBase stores.
  - plyrmr: This package provides common data manipulation operations.
  - ravro: This package allows R to read and write in avro format.

Not all packages are needed for a Map-Reduce job. The main packages are rmr2 and rhdfs.

- ORCH: This is an Oracle R connector for Hadoop. ORCH can be used on the Oracle big data Appliance or on non-Oracle Hadoop clusters.

2.3.7 Hadoop 2.X

Hadoop 2.X, also known as Hadoop YARN("Yet Another Resource Negotiator"), is the newest version of Hadoop and it differs from Hadoop 1.X in the Map-Reduce model only. The HDFS file system remains unchanged. A short description of this version's Map-Reduce model is presented below.

Hadoop 1.X has two daemons, the Jobtracker and the Tasktracker. On the other hand, Hadoop YARN has three daemons: the ResourceManager, the ApplicationMaster, and the NodeManager. The Jobtracker's responsibilities, which are the resource management and job scheduling, are split into the ResourceManager and the per-application ApplicationMaster. The NodeManager is a per-node slave, like the Tasktracker. These are the three main components of all of YARN's functionality and are described in more detail below:

- ResourceManager: Each cluster has a single ResourceManager which manages all the available cluster resources and helps distributing the applications that run on the system. It has two parts: a scheduler, which is responsible for allocating resources to the various running applications, and an ApplicationManager that manages user jobs on the cluster. The resource requirements of an application determines how the scheduling is performed. This is based on the containers which incorporate many elements such as memory, cpu or disk.

- NodeManager: This is per-machine slave, which tracks the available data processing resources and manages users' jobs. Moreover, it monitors the resource usages, such as memory, disk, and network and sends reports to the ResourceManager.

- ApplicationMaster: This is the "master" of a users' job. It works with the ResourceManager to handle the resources and with the NodeManager to execute and monitor the component tasks. Among others, one responsibility of the ApplicationMaster is to dynamically increase or decrease the resources consumption.

Figure 2.5 shows the YARN architecture. For this thesis, the Hadoop 1.2.1 was used due to the fact that Hadoop 2.X did not exist when the thesis started. The code written in Hadoop 1.2.1 can be used with Hadoop 2.X.

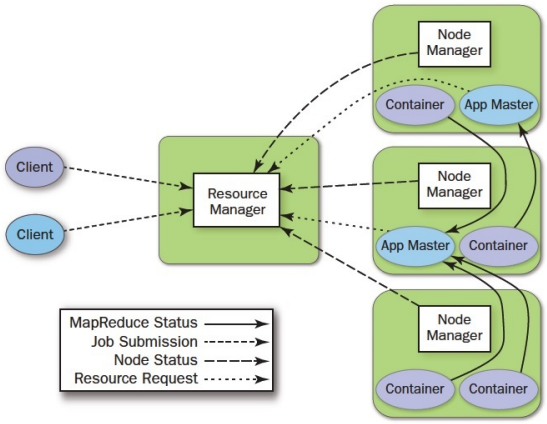

Figure 2.5: YARN Architecture[4]. The master node runs the ResourceManager (RM) where the clients can submit a job and the slave nodes run the NodeManager, which sends the status of the node to the RM and the ApplicationMaster, which requests resources for the job.

Chapter 3

Background

In this chapter we present the background knowledge that is needed for this thesis. First, we describe the notions of feature selection with its main categories. Then, we present the Max-Min Parent and Children (MMPC) algorithm and we describe two independence tests which are used by MMPC. Moreover, we describe two methods that can be used with MMPC. Finally, we present the related work.

3.1 Feature Selection

One major category in machine learning is predictive modeling or supervised learning. In this type of learning, there is a number of training examples (or observations or data points) associated with desired outcomes and the task is to find the relationship between the data points and the outcomes. So, given an input $X = \{x_1, x_2, ..., x_M\}$ we want to predict the target of interest Y. However, it may be the case that the output Y is not determined by all the features in $\{x_1, x_2, ..., x_M\}$ but from a subset of them $\{x_{(1)}, x_{(2)}, ..., x_{(m)}\}$, where $m < M$, which are the more relevant with the task.

In this section we give a brief review about supervised feature (or variable) selection. For more details, a very good review can be found in [24]. However, before defining what feature selection is, it is important to define some conventions. To begin with, datasets are stored in tables, where each row represents an instance, that is a sample, and each column a feature, i.e. a property of the sample. For example, a dataset may contain the height and weight of students. In this case, a sample is a student with specific values in its features height and weight. In our previous example, the target may be whether the student is obesse or not.

The task of feature selection is very important in machine learning. The main reason is that it can decrease computational cost. Imagine a dataset with 4 millions features but only 4 hundred are important for the task we want to accomplish. By applying feature selection the unnecessary features are discarded and the computational cost is reduced. Another advantage of feature selection is that it can improve the predictive performance of classifiers by selecting only the features that are more predictive. This way the training times can be reduced significantly. Furthermore, applying feature selection in a problem can reduce the storage requirements. It is not necessary to store information that is not important to the task. Finally, feature selection can improve our understanding about the data and also we can visualize them better. That is, it is difficult to understand the

structure of the predictive of the model in a thousand feature dataset but it is a lot easier if the number of features are a few hundreds.

The task of feature selection to reduce the number of features in a dataset by eliminating features with no additional predictive information given the selected features. More formally, according to [25] feature selection can be defined as follows:

Definition 3.1.1. [25] A feature selection problem is a tuple $\langle X, \Phi, T, M \rangle$ where X is a sample of input patterns defined over a feature set Φ, $T \in \Phi$ a target variable and M a performance metric of a classifier's model and the selected features. A solution to the problem is a feature subset $\phi \subseteq \Phi$ and a learning algorithm A that maximizes $M(\phi, A(T, X \downarrow \phi))$, where $X \downarrow \phi$ is the projection of the data X on only the features in ϕ.

In the aforementioned definition, the metric M is not specifically defined as it is problem-depended. Different problem requires different metric An example of metric is correlation when the data take continuous values. So, in a given problem we try to find the features ϕ that will maximize a specific metric M given the classifier A.

One major distinction needs to be made between dimensionality reduction and feature selection. The task of feature selection differs from dimensionality reduction techniques, in that dimensionality reduction models build a new set of features from the original feature set, which is usually smaller. However, feature selection reduces the dimensions of the dataset by selecting a subset of the original features.

There are three main methods of feature selection: the filters, the wrappers and the embedded methods. The filters select features by evaluating each feature with respect to an outcome(also called target). The evaluation is performed with some metric like correlation. The wrappers evaluate a classifier with the use of a metric like accuracy and they select the subset of features that achieve the highest performance of the classifier using that specified metric. The last category is the embedded methods. These methods perform feature selection in the training stage of a classifier.

3.1.1 Filter Methods

Filter algorithms use a metric M to select features without evaluating that metric on the output of a classifier A. Examples of metrics are entropy and mutual information, which measure the quality of a feature with respect to the target. The main advantages of these methods are their flexibility, their speed and their robustness to overfitting. However, the downside is that they cannot efficiently find all possible combinations of features that produce the best results. That is, a feature may be useless by itself but combined with others can be very useful.

This is how most filter algorithms behave. However, the algorithm we are concerned(MMPC) tries heuristically to find conditional dependencies between the features and the target using statistical tests.

One representative approach for filter methods is called variable ranking. In this approach the features are evaluated with a metric and then they are sorted according to their evaluation in decreasing order. Then, the first k(user defined) features are selected. For example, assume a dataset with m instances $\{x_{k,i}, y_k\}$ that consists of $i = 1 \dots n$

input features and a scoring function $S(i)$. We evaluate the features $x_{k,i}$ and y_k using $S(i)$ and finally the features are sorted in decreasing order based on that evaluation.

Metrics that can be used with this method are the correlation and the information gain. In the following subsections, we describe these metrics and how they can be used.

Correlation

Correlation can find any dependence between two random variables. The most famous among other types of correlation is the Pearson correlation. The Pearson correlation coefficient, ρ, between two random variables X, Y is defined as follows:

$$\rho_{X,Y} = corr(X,Y) = \frac{cov(X,Y)}{\sqrt{\sigma_X \sigma_Y}}$$

where *cov* is the covariance and σ the variance, and are defined as follows:

$$cov(X,Y) = \sum_{k=1}^{m} (\boldsymbol{x_k} - \overline{\boldsymbol{x}})(\boldsymbol{y_k} - \overline{\boldsymbol{y}})$$

$$\sigma_X = \sqrt{\sum_{k=1}^{m} (\boldsymbol{x_k} - \overline{\boldsymbol{x}})^2}$$

The values of the correlation varies between $[-1, 1]$, where -1 indicates perfect decreasing dependency, that is inverse linear relationship, and 1 indicates perfect increasing dependency, that is linear relationship. When there is small relationship between the variables, the correlation value is near zero.

Mutual Information

Another popular metric for feature selection that comes from the information theory is the mutual information. Given two continuous variables X and Y it is defined as follows:

$$I(X;Y) = \int_X \int_Y p(x,y) \log \frac{p(x,y)}{p(x)p(y)} dxdy$$

where $p(x)$ is the probability density of X, $p(y)$ is the probability density of Y and $p(x,y)$ is the joint probability density of X and Y. In the case of discrete features the mutual information is defined as follows:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(X=x, Y=y) \log \frac{P(X=x, Y=y)}{P(X=x)P(Y=y)}$$

where the probabilities in this case are estimated from frequency counts.

3.1.2 Wrapper Methods

Wrapper methods use predictive models such as classifiers or regressors, which are considered as a black box, in order to perform feature selection. The prediction performance of the model is used to evaluate a subset of features. Each subset of features is used to

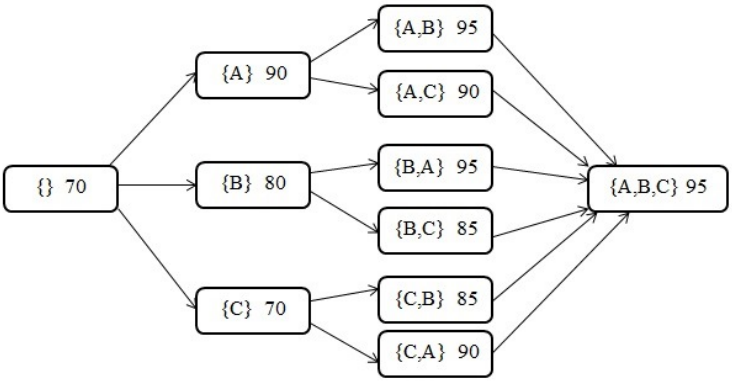

Figure 3.1: Forward Selection Example[28]. In the first step, an algorithm will evaluate the empty set and then will enter every feature and evaluate separately. The feature with the best evaluation is selected (A in this case). Then, the features B and C will enter separately and evaluated. The best evaluation comes when B is entered. Finally, the C enters and has the same score and so the features {A,B} are selected.

train the model and then the model's predictive performance is used to score the subsets. In general, there are 3 things to define for wrappers:

- The model to use.
- A performance estimation technique to evaluate the model.
- A method to find the subset of features.

An exhaustive search is infeasible to find the best subset of features when the number of features is large and it has been shown to be an NP-Hard problem[30]. Greedy search algorithms have been proposed to speed up the procedure. These are the forward selection and the backward selection. In forward selection algorithms, the features are selected and are added in a set of selected features recursively. In each step the feature with the highest score is selected. Figure 3.1 shows an example of the forward selection method. The figure shows all the possible paths but the algorithm takes the path with the highest score in each step. In the first step, the empty set is evaluated and gives a score of 70. In the second step, each feature is added in the empty set and evaluated independently. In this example, the best score is given by feature A. In the next step, the remaining features (B and C) are added one at a time and evaluated. This procedure continuous until either no other feature is left to add in the set or the subset that is evaluated performs worse than the previous evaluated subset.

In the backward elimination method, the algorithm starts with a set that contains all the features and recursively eliminates them one by one. Figure 3.2 shows an example of the backward elimination method. As previously, the figure shows all possible paths. In the first step, the set includes all the features. These are evaluated with the use of a classifier or regressor. In the second step, each feature is eliminated and each subset is evaluated. This procedure continues until either no other feature is left in the set or the subset that is evaluated performs worse than the previous evaluated subset.

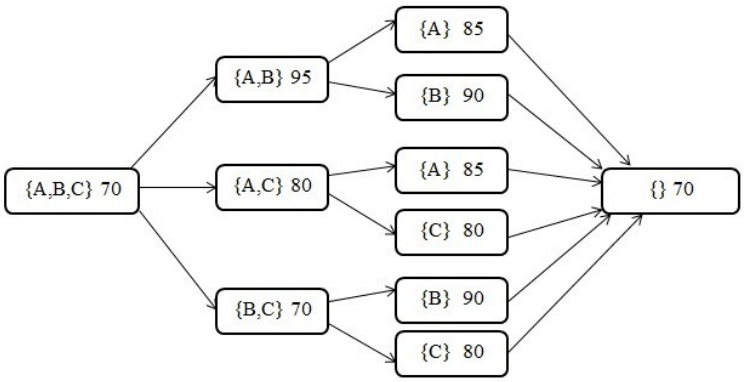

Figure 3.2: Backward Elimination Example[28]. In the first step, an algorithm will evaluate all the features and then will remove every feature and evaluate separately each set. The set with the features {A,B} is selected. Finally, it removes each feature from the set and concluded that the highest score is when features {A,B} are selected.

The advantages of these methods is that they can take into account the dependencies between features and that they interact with the model to produce better results. However, they have high risk of overfitting because of their interaction with the model and also they have high computational cost.

A discrimination needs to be made between wrappers and filters[25]. Considering the definition 3.1.1, a wrapper algorithm will search the space of all possible subsets given an evaluation metric M and all learners A. On the other hand, a filter algorithm selects features without evaluating a metric M on the output of A. However, it is important to notice that it is possible for filter methods to use a learner A' and a metric M' to select features. In this case, the difference with wrappers is that $A \neq A'$ and $M \neq M'$. If there was the case that $A = A'$ and $M = M'$, then the filter algorithm acts as wrapper.

To make it clear, consider the example where Support Vector Machine (SVM) is used to select features. The steps of the feature selection algorithm are the following:

1. Build SVM classifier and find weights for all features
2. Choose half the features with highest weights
3. Repeat steps 1,2 until 1 feature is left

This algorithm acts as wrapper if SVM is used also for classification, and it acts as a filter if other algorithm than SVM is used.

3.1.3 Embedded Methods

In embedded methods the feature selection is a part of the learning procedure of a given model. These methods are computationally more efficient than wrapper methods but they are specific to the model. One successful example of these methods is LASSO[42] (Least Absolute Shrinkage and Selection Operator). This is a regression algorithm which tries to find the coefficients $\boldsymbol{\beta}$ by minimizing the following:

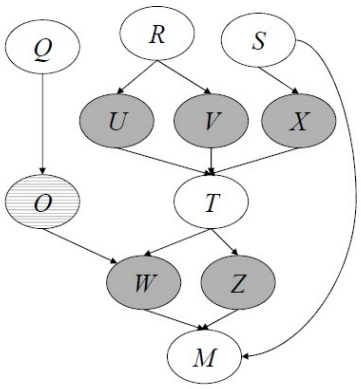

Figure 3.3: An Example of a Bayesian Network[43]. The nodes represent features and the edges conditional dependencies. The nodes with grey color belong to the parents and children of T and along with node O they form the Markov Blanket of T.

$$\boldsymbol{\beta} = \underbrace{argmin}_{\boldsymbol{\beta}}(\sum_{i=1}^{N} y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

where N is the number of instances, y_i are the labels, x_i are the instances, p is the number of features and λ is a regularization parameter. The penalty on the regression coefficients shrinks many of them to zero. The features with non-zero coefficient are selected by this algorithm.

3.2 Max-Min Parent and Children (MMPC) Algorithm

The Max-Min Parents and Children (MMPC)[43] algorithm belongs to the filter methods and its main different from most filter algorithms is that it tries to find conditional dependencies between the features and the target. Its theory stems from the theory of the Bayesian Networks. Before describing the algorithm we will present the appropriate background.

To begin with, a Bayesian Network (BN) is a directed acyclic graph which represents a probability distribution P over a set of random variables $\boldsymbol{\Phi}$ (known as nodes). The edges represent conditional dependencies. So, if there is no edge between two variables (nodes) then these variables are conditionally independent. Two nodes X and T are conditionally independent given the set of variables $\boldsymbol{Z}$, if and only if $P(T|X, \boldsymbol{Z}) = P(T|\boldsymbol{Z})$. In a BN the Markov condition holds:

Definition 3.2.1. Markov condition.[26] Any node is conditionally independent of its non-descendants given its parents.

Figure 3.3 shows an example of a BN. In this example, the parents of nodes T are U, V, and O and so it holds that $P(T|U, V, R) = P(T|U, V)$ and $P(T|O, S) = P(T|O)$.

Definition 3.2.2. Faithfulness Condition.[26] A BN G and a distribution P are faithful to one another if and only if all independence relations in P are entailed by the Markov condition applied to G.

In order to compute the independencies in a BN from the Markov Condition, the d-separation criterion is used. Before defining d-separation it is important to define what a collider and a blocked path are.

A collider node in a path is considered the node with two incoming edges. In figure 3.3, W is considered a collider because it has two incoming edges from O and T. The blocked path can be defined as follows:

Definition 3.2.3. Blocked Path. A path p from node X to node Y is blocked by a set of nodes $\boldsymbol{Z}$, if [26] there is a non-collider node on p that belongs in $\boldsymbol{Z}$ and if no collider nodes of p and none of their descendants belong in $\boldsymbol{Z}$.

Considering the aforementioned definition, d-separation is defined as follows:

Definition 3.2.4. d-separation. Two nodes X and Y are d-separated by $\boldsymbol{Z}$ (denoted as $Dsep(X;Y|\boldsymbol{Z})$) if and only if every path from X to Y is blocked by $\boldsymbol{Z}$.

The independence between two nodes in a BN can be defines as follows:

Theorem 1. If a BN is faithful to a distribution P, then $Dsep(X;T|\boldsymbol{Z}) \Leftrightarrow Ind(X;T|\boldsymbol{Z})$.

A fundamental theorem for MMPC is the following [27]:

Theorem 2. If a BN N is faithful to a distribution P, then:
1. There is an edge between the pair of nodes X and Y in N iff X and Y are conditionally dependent given any other set of nodes.
2. If for the triplet of nodes X, Y, and Z in N, X is adjacent to Y, Y is adjacent to X, and Z is not adjacent to X, $X \rightarrow Y \leftarrow Z$ is a subgraph of N iff X and Z are dependent conditioned on every other set of nodes that contains Y.

The parents and children of T is denoted by $PC(T)$ and it holds that:

Theorem 3. Given two BN C and N that are both faithful to the same distribution then, $PC_C(T) = PC_N(T)$ (proof in []).

The Markov Blanket of a node in a BN can be defined as follows:

Definition 3.2.5. Markov Blanket. In a BN, the Markov Blanket of a node T (noted as $MB(T)$) is the minimal set of nodes on which every other node conditioned on this set is independent of T. i.e. $\forall X \in \boldsymbol{\Phi} \backslash (MB(T) \cup \{T\}, \boldsymbol{Ind}(X;T|MB(T))$.

In figure 3.3, the $MB(T)$ consists of the nodes U, V, X, W, Z and O. So in a BN the following theorem holds:

Theorem 4. In a faithful BN N the $MB(T)$ is unique and it is the set of parents, children and the parents of children (proof in []).

MMPC tries to find the parents and children of a node of interest T in a BN N which is faithful to a given probability distribution P. This is accomplished with two phases: the forward and the backward. In the forward phase (phase I), the algorithm finds the candidate parents and children (CPC) of T with a heuristic function. In this phase all parents and children of T and some non-members will enter CPC. The algorithm is

Algorithm 1 Forward Phase[43]

Input: target T, dataset D
Output: selected variables CPC

1: $CPC = \emptyset$
2: **do**
3: **for** $\forall$ variable X find **do**
4: $minassocset(X)$ = subset s of CPC that minimizes $assoc(X;T|s)$
5: F = variable $\Phi\backslash(\{T \cup CPC\})$ that maximizes $assoc(F;T|minassocset(F))$
6: **if** $Dep(X;T|minassocset(F)$ **then**
7: $CPC = CPC \cup F$
8: **while** CPC has not changed

shown in Algorithm 1. In the backward phase (phase II) the algorithm tries to remove the variables that falsely entered CPC from the first phase. The algorithm is shown in Algorithm 2.

The first phase of the algorithm finds a candidate set of parent and children called CPC for the target variable T. The first variable that is inserted in the CPC set is the one that achieves the highest univariate association with T. Next, the variable that achieves the maximum association with T conditioned on every subset of CPC that has the minimum association. The interpretation[44] of this is to select the variable that despite our best efforts to make it independent of T has the highest minimum association with T among all other candidate variables. Association of X with T given Z in the pseudo-code is denoted with $assoc(X;T|Z)$. Finally, the complexity of the algorithm in this phase is $O(|F| \cdot (|CPC| - 1)^m ax_k)$, where $|F|$ is the number of features in the dataset, $|CPC|$ is the number of features selected by MMPC and max_k is the maximum number of features to be conditioned on each association.

Algorithm 2 Backward Phase[43]

Input: target T, dataset D
Output: selected Variables CPC

1: **for** $\forall X \in CPC$ **do**
2: **if** $\exists s \subseteq CPC, s.t. Ind(X;T|s)$ **then**
3: $CPC = CPC\backslash\{X\}$

The backward phase tries to remove the false positives variables that have entered the CPC set by testing the independence of variable X with target T to all possible subsets in the CPC. That is, it examines if every selected feature can be d-separated from T conditioned on all possibles subsets of CPC. However, it may happen that variables are selected that do not belong to the parents or children (even after the backward phase). Finally, the complexity of the algorithm in this phase is $O(|CPC| \cdot (|CPC| - 1)^m ax_k)$, where $|CPC|$ is the number of features selected by MMPC and max_k is the maximum number of features to be conditioned on each association. Totally, the algorithm's complexity is $O(|F| \cdot (|CPC| - 1)^m ax_k)$

MMPC can be used for different types of variables, as long as the appropriate test is used. For example, it can be used for mixed categorical and continuous variables, and the target can be either categorical, continuous or even censored (survival analysis). For example, in [44] they use the χ^2 test of independence. In the next section, we describe the tests of independence that were used for this thesis.

3.3 Testing Single Hypothesis of Independence

In order to test whether the association in MMPC between a feature and a target, independence tests are used. In this section, we describe the G^2 and the Fisher Z test. The first one is used when variables take discrete values and the latter is used when variables take continuous values. Given a predefined threshold the tests reject the null hypotheses if the pvalue is smaller than the threshold.

3.3.1 G^2 Test

This test is used to find statistical independence between two variables that take discrete values. The null hypotheses H_0 assumes that the relative proportions of one variable are independent of the second variable. On the other hand, the alternative hypotheses H_1 assumes that the relative proportions are dependent.

The first step of the test calculates the contingency table between two variables. Then it calculates the following quantity

$$G^2 = 2 \sum_{ij} O_{ij} \log \frac{O_{ij}}{E_{ij}} \tag{3.1}$$

where O_{ij} is the number of the observations when the values of the variables is i and j, respectively, and E_{ij} is the expected number for the i, j values of the variables. This value is calculated as follows

$$E_{ij} = \frac{O_{i+} O_{+j}}{n} \tag{3.2}$$

where O_{i+} is the number of observations when the value of the first variable is i, O_{+j} is the number of observations when the value of the first variable is j and n is the total number of observations.

Under the Null hypothesis, the G^2 statistic follows the χ^2 distribution with degrees of freedom: $df = (|D(X)| - 1)(|D(Y)| - 1)$, where $D(K)$ is the domain of the variable K.

So far, we described the unconditional G^2 test. For this thesis, we also need to calculate conditional associations. For the conditional case, given a conditional set of variables, the equation 3.1 is used to calculate G for every combination of the values of the conditional variables. Then, the different values of the statistic G are summed to get the final statistic.

More formally, consider the independent test $Ind(X_i; X_j | \mathbf{X}_k)$. Then let O_{ijk} be the number of times where $X_i = a, X_j = b$ and $X_k = \mathbf{c}$. Defining in similar way the O_{ik}, O_{jk} and X_k, the G^2 statistic can be defined as follows [46]:

$$G^2 = 2 \sum_{a,b,\mathbf{c}} O_{ijk} ln \frac{O_{ijk} O_k}{O_{ik} O_{jk}}$$

The degrees of freedom in this case will be:

$$df = (|D(X_i)| - 1)(|D(X_j)| - 1) \prod_{X_i \in \mathbf{X}_k} |D(X_l)|$$

where $D(X)$ is the domain of the variable X.

Lets consider the following dataset and try to find the univariate association between the target T and each feature $F1$.

Id	T	F1	F2
1	0	1	0
2	0	1	0
3	1	0	1
4	0	1	1
5	1	1	0
6	1	0	1
7	1	0	1
8	1	1	0
9	0	0	0
10	0	1	0
11	1	1	1
12	0	1	1
13	0	1	1
14	0	0	0
15	0	0	1

In order to compute the statistic from equation 3.1 we first need to find the contingency tables. The table is shown below.

F1/T	0	1
0	4	5
1	2	4

From these tables we need to find the expected values. This is accomplished with the use of the equation 3.2. For example, the expectation value when $F1 = 0$ and $T = 0$ is:

$$E_{11} = \frac{9 \cdot 6}{15}$$

So the expected values for the table is shown below.

F1/T	0	1
0	3.6	5.4
1	2.4	3.6

Considering the aforementioned tables the statistic for the F_1 feature and the target is computed as follows:

$$G^2 = 2(\sum_{i=0,j=0}^{1} O_{ij} \log \frac{O_{ij}}{E_{ij}}) = 0.08$$

After computing the statistic we need to find the pvalue which follows a chi-square distribution with degrees of freedom for our case $(2-1)\cdot(2-1) = 1$. The pvalue for the statistic is 0.22.

3.3.2 Fisher Z Test

This test is used to check the partial correlation between two variables X and Y by eliminating the effect of set of variables $\mathbf{Z} = \{Z_1, Z_2, \ldots, Z_n\}$. For this test, the null hypotheses H_0 assumes that the partial correlation is equal to zero and the alternative hypotheses H_1 that it is not, i.e. $H_0 : \rho_{XY} = 0, H_1 : \rho_{XY} \neq 0$.

The partial correlation can be found with many different ways. For this thesis, we calculate the covariance matrix Ω. The covariance value for two variables X, Y is computed as follows:

$$\omega_{XY} = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{x_i} - \overline{\boldsymbol{x}})(\boldsymbol{y_i} - \overline{\boldsymbol{y}})^T$$

Computing all the two-pair covariance values for all the variables X, Y, and Z the covariance matrix can be formed.

If we define $P = \Omega^{-1}$, the partial correlation is defined as follows

$$\rho_{XYZ} = -\frac{p_{ij}}{\sqrt{p_{ii}p_{jj}}}$$

The last step is to calculate the Fisher's Z-transform, which it is defined as follows:

$$z(\rho_{XYZ}) = \frac{1}{2}ln(\frac{1+\rho_{XYZ}}{1-\rho_{XYZ}})$$

The null hypotheses is rejected with significant level α if:

$$\sqrt{N - |Z| - 3} * z(\rho_{XYZ}) > \Phi^{-1}(1 - a/2)$$

where $\Phi(\cdot)$ is the cumulative distribution function of a Gaussian distribution with zero mean and unit standard deviation, and N is the sample size.

3.4 Testing Multiple Hypotheses of Independence

MMPC tests multiple hypotheses in the first step where the univariate associations are computed between the target and each feature in the dataset. The features may be a few hundreds or thousands. The disadvantage of using a predefined threshold is that it does not take into account the number of features that are tested. So it may be the case that many features may falsely be selected that follow the null hypothesis. To solve this problem the Storey's method can be used. This method suggests to use a qvalue

threshold to measure each feature's significance instead of pvalues. This threshold is a measure in terms of the false discovery rate (FDR), which is the expected proportion of false positive findings among all the rejected hypotheses.

Consider the case where m features are tested with pvalues $P = \{p_1, p_2, ..., p_m\}$. Storey's method tries to find, the number of k rejected hypotheses such that:

$$k = max\{p_{(i)} \in P, s.t. p_{(i)} \leq \frac{i}{m \cdot \pi_0} A\}$$

where A is a predefined threshold and

$$\pi_0 = \frac{\#\{p_i > \lambda; i = 1, ..., m\}}{m(1 - \lambda)}$$

where $\lambda \in [0, 1]$. The features with pvalue higher than λ are considered to follow the null hypotheses (distributed uniformly, i.e. $U \in [0, 1]$). So, the quantity π_0 measures the number of features that are distributed uniformly, i.e. follow the null hypotheses.

3.5 Fisher's Combined Probability Test

Using MMPC with Map-Reduce model, the Fisher's combined probability test can be very useful. This method combines pvalues produced by independent tests in order to find one statistic. This is accomplished with the following formula:

$$X^2_{2n} = -2 \sum_{i=1}^{n} ln(p_i)$$

where p_i is the pvalue for the i^{th} hypotheses and n is the number of pvalues to be combined. This method assumes that the pvalues are independent and if all the null hypotheses are true then the statistic follows the chi-square distribution with degrees of freedom $2n$.

As mentioned earlier in the Map-Reduce model the mappers run a task in parallel and reducers combine the results. So using Fisher's method, the mappers could run any independent test, compute the pvalues and the reducers would combine the pvalues using this method to compute the total statistic.

3.6 Related Work on Feature Selection for Big Data

There are few works on feature selection and big data. And there are even fewer works on feature selection that use the Map-Reduce model.

To begin with, in [31] the author proposes a simple idea for feature selection based on Map-Reduce. More particularly, he proposes to perform feature selection, using mutual information, in each sub-dataset and then select the feature that was selected most times. In case more than one features have been selected the same number of times then the feature with the highest mutual information is selected. This idea is very simple and not so robust as it may be that a feature may have a low mutual information in the sub-datasets but if it was tested in the whole dataset the same feature may have a high mutual information.

Another feature selection algorithm for big data can be found in [32]. The authors propose an algorithm that is able to efficiently cope with ultrahigh-dimensional datasets and select a small subset of interesting features from them. Their method can tackle two challenging tasks: group-based feature selection, and nonlinear feature selection. The algorithm iteratively activates a group of features, and solves a sequence of multiple kernel learning subproblems. The main disadvantage is that the number of selected features is several orders of magnitude lower than the total of features. Moreover, the algorithm runs on a single machine.

In [33] the authors propose a novel large-scale feature selection algorithm that can read data in distributed form and perform parallel feature selection in symmetric multiprocessing mode via massively parallel processing. The algorithm supports both supervised and unsupervised feature selection and selects features by evaluating their abilities to explain data variance. In both cases, to preserve the variance and select features, they formulate a minimization problem.

In [34] they propose an algorithm which tightens a sparsity constraint by gradually removing variables based on a criterion and a schedule. They formulate the feature selection problem as a constrained optimization problem. More particularly, given a loss function $L(\beta)$ defined on the samples, the problem is the following: $\boldsymbol{\beta} = \underbrace{argmin}_{|\{j:\beta_j \neq 0\}| \leq k} L(\boldsymbol{\beta})$, where $\boldsymbol{\beta}$ is the parameter vector, k is the number of relevant features and $L(\cdot)$ the loss function, which is differentiable with respect to $\boldsymbol{\beta}$. Their algorithm has two steps. In the first step, the parameters are being updated so as to minimize the loss $L(\beta)$. In the second step, the algorithm keeps only the M_ϵ (user defined) features with the highest coefficient magnitudes $|\beta_j|$. This algorithm can be used with regression, classification or ranking problems. The main disadvantage of this method is that it requires to pass the data multiple times to select one feature.

Another algorithm for online feature selection is SAOLA[35] (Scalable and Accurate OnLine Approach). This algorithm processes each feature sequentially, that is one dimension at a time. Taking one feature F_i at time t_i the algorithm tries to maintain the minimum feature subset that maximizes the predictive performance for classification. So they formulate the problem as follows: $S^*_{t_i} = argmin_{S'}\{|S'| : S' = argmax_{\zeta \subseteq S \cup F_i} P(C|\zeta)\}$, where S is the feature subset and C is the class feature. A feature F_i is not inserted in the feature subset if $P(C|F_i) = P(C)$ and if $P(C|S^*_{t_{i-1}}, F_i) = P(C|S^*_{t_{i-1}})$. These equations show that F_i does not carry additional predictive information to C independently and given the selected feature set $S_{t_{i-1}}$. If the feature F_i is inserted to the set then the $S^*_{t_i}$ needs to be pruned so as to satisfy: $S^*_{t_i} = argmax_{\zeta \subseteq S_{t_i}} P(C|\zeta)$. The pruning is performed by checking all subsets of size less than or equal to a predefined number $k (1 \leq k \leq |S^*_{t_i}|)$.

The first feature selection algorithm that used the Map-Reduce framework was introduced by Singh et al [38] and it uses logistic regression to evaluate the features because it has been shown[40] to be a top performing algorithm in high-dimensional data. Kubica et al.[36] describes this algorithm along with other two to evaluate new features that are based on the forward selection wrapper framework. These techniques are: full forward feature selection[37], single feature optimization[38] and grafting[39]. Due to the fact that we compare our algorithm with the work of Singh[38] we describe it in more detail in the following section along with the grafting algorithm which is based on the same idea. We describe in the next section why the full forward feature selection algorithm is not

appropriate for real world applications.

3.6.1 Single Feature Optimization

In a full forward feature selection algorithm the features are added sequentially in a set of selected features. The procedure is the following: evaluate every feature, independently, using a metric and insert the feature that performs the "best" to the set of selected features. The evaluation of the feature has the following steps: add the feature to the feature set, learn a model with the current feature set and evaluate the model on some metric. When the evaluation is completed and a feature is added to the selected features set, an iteration is said to be completed. If we assume that we have D features, then on the dth iteration, $D-(d-1)$ models are build. It is clear that the complexity of this algorithm is very high and cannot be used when the number of features is large.

Single Feature Optimization (SFO) algorithm belongs to the forward selection methods and uses the logistic regression classifier to select features. This classifier tries to model the conditional probability $P(Y|X)$ between the variable Y and X. More formally, given N data points $\{\mathbf{x}_i, \mathbf{y}_i\}, 1 \leq i \leq N$, where $\boldsymbol{x}_i$ is the input vector and y the binary target the model is the following:

$$log\frac{p(\boldsymbol{x})}{1-p(\boldsymbol{x})} = \beta_0 + \boldsymbol{x}\boldsymbol{\beta}$$

where p is the predicted probability $P(Y=1)$ and $\boldsymbol{\beta}$ are the coefficients of the model. Solving with respect to p we get

$$p = f_{\boldsymbol{\beta}}(\boldsymbol{x_i}) = \frac{e^{\boldsymbol{\beta x_i}}}{1+e^{\boldsymbol{\beta x_i}}}$$

So, the logistic regression model is defined by its coefficients $\boldsymbol{\beta}$. The function f maps the output to the range $[0,1]$. To find the coefficients, the maximum likelihood estimation can be used. Therefore, maximizing the log-likelihood we get

$$\boldsymbol{\beta}_{learned} = \underbrace{argmax}_{\boldsymbol{\beta}} \sum_{i=1}^{N} (y_i ln f_{\boldsymbol{\beta}}(\boldsymbol{x_i}) + (1-y_i) ln(1-f_{\boldsymbol{\beta}}(\boldsymbol{x_i}))$$

Ideally, SFO would select features by evaluating a fully learned logistic regression model containing both a candidate feature and the current set of features. However, as described previously the complexity is high.

In order to speed up the evaluation of the features, the method retains the coefficients from the current best model and optimizes only the coefficients $\boldsymbol{\beta}$ of the candidate features. That is, it does not require to find all the coefficients of the model (for the features already selected and the candidate features) to evaluate a single feature, but rather it only finds the coefficient for the new candidate feature by keeping the current coefficients of the best model fix. To do this, the optimization is limited only to the coefficient of the candidate feature. To find the coefficients of the features to be evaluated, the method maximizes the log-likelihood L of the logistic regression with respect to the candidate feature:

$$\underbrace{argmax}_{\beta'_d} \sum_{i=1}^{N} (y_i ln f_{\boldsymbol{\beta}^{(d)}}(\boldsymbol{x_i}^{(d)}) + (1 - y_i) ln(1 - f_{\boldsymbol{\beta}^{(d)}}(\boldsymbol{x_i}^{(d)})))$$

Newton's method is used to maximize the log-likelihood. So, the update rule for the coefficient β would be:

$$\beta'_d = \beta'_d - \frac{\frac{\partial L}{\partial \beta'_d}}{\frac{\partial^2 L}{\partial \beta'^2_d}}$$

where:

$$\frac{\partial L}{\partial \beta'_d} = \sum_{i=1}^{N} x_{id}(y_i - f_{\boldsymbol{\beta}^{(d)}}(\boldsymbol{x_i}^{(d)}))$$

$$\frac{\partial^2 L}{\partial \beta'^2_d} = -\sum_{i=1}^{N} x^2_{id} f_{\boldsymbol{\beta}^{(d)}}(\boldsymbol{x_i}^{(d)})(1 - f_{\boldsymbol{\beta}^{(d)}}(\boldsymbol{x_i}^{(d)}))$$

After finding the coefficient of a feature, the model can be evaluated by measuring a metric like the AUC or the prediction error. Finally, after finding the feature that performs the best, the method learns the full model with the new selected feature.

This method can handle categorical features. To do this, the features need to be binarized. That is, every value of a categorical feature is represented as binary and takes value one if the value exists in the instance or zero if it does not. So every value of the feature would be a new feature. For example, if a feature takes values $\{a, b\}$ and in an instance the feature takes the value $\{a\}$ then this would be presented as two features with values $\{1, 0\}$. For this type of features, SFO algorithm would work independently for each of the resulting binary feature and then the different coefficients can be combined.

The method can also handle multiple class prediction problems. The authors propose to create a separate training dataset for each classification label and run feature evaluation for each of these datasets. Then at prediction time each model can be queried and the class label with the highest predicted value is chosen.

Parallel Single Feature Optimization

The authors parallelized the algorithm using the Map-Reduce model. Their algorithm consists of three steps:

1. **Map phase**: In this phase the algorithm iterates over the training records and computes the predicted probability of the current model. Then, outputs for each candidate feature ($\boldsymbol{x}'_i$) an intermediate dataset that contains the predicted probability, the true outcome and the vector of the feature. Algorithm 3 shows the map phase.

2. **Reduce phase**: This phase takes as input the output of the map phase and computes the coefficient β_d for each feature using the Newton's method. Algorithm 4 shows the reduce phase.

3. **Post-Processing**: In case of categorical features, sum the coefficients that belong to the same feature.

Algorithm 3 Map Phase[36]

for each $\{\boldsymbol{x_i}, y_i, \boldsymbol{x_i'}\}$ in $\{\mathbf{X}, \boldsymbol{y}\}$ **do**
    $p_i = f(\boldsymbol{x_i}, \boldsymbol{\beta})$
    for each $x_{id}^{'} \in \boldsymbol{x_i'}$: **do**
        Output $(x_{id}^{'}, y_i, p_i)$

Algorithm 4 Reduce Phase[36]

$\beta_d^{`} = 0$
Until convergence of $\beta_d^{`}$:
$\frac{\partial L}{\partial \beta_d'} = \frac{\partial^2 L}{\partial \beta_d'^2}$
for each $(\boldsymbol{x_i}, y_i, p_i) \in T_d$ **do**
    $a_i = \log(\frac{p_i}{1-p_i})$
    $p_i^{'} = \frac{e^{\alpha_i + \beta_d^{'}}}{1+e^{\alpha_i+\beta_d^{'}}}$
    $\frac{\partial L}{\partial \beta_d'} = \frac{\partial L}{\partial \beta_d'} + (y_i - p_i^{'})x_{id}^{'}$
    $\frac{\partial^2 L}{\partial \beta_d'^2} = \frac{\partial^2 L}{\partial \beta_d'^2} - p_i^{'}(1-p_i^{'})x_{id}^{'2}$
    $\beta_d' = \beta_d' - \frac{\partial L}{\partial \beta_d'} / \frac{\partial^2 L}{\partial \beta_d'^2}$

To sum up, this algorithm performs single feature optimization using the logistic regression formula and finds the coefficients and the log-likelihood of the model for each feature. Then, the feature that is selected is the feature with the maximum log-likelihood. Finally, it runs the logistic regression model with all the features that have been selected so far in order to find the coefficients of the full model. This step requires multiple passes of the data.

3.6.2 Grafting

This algorithm also uses the logistic regression and adds to the set of features, the feature with the largest magnitude gradient. The interpretation for this is that the new feature that is added to the feature set would help the model the most. Like SFO, grafting uses the coefficients of the current model in order to find the next feature to select. When a feature is selected the model is fully relearned.

The gradient is computed independently for each feature and it can be computed as follows:

$$|\frac{\partial L}{\partial \beta_d'}| = |\sum_i x_{id}^{`}(y_i - p_i)|$$

So the feature that is added to the feature set is the one with the largest gradient. That is,

$$\underbrace{argmax}_{\beta_d'} |\frac{\partial L}{\partial \beta_d'}|$$

Parallel Grafting

The authors propose a three step algorithm:

1. **Map phase**: For each training record and for each new feature, compute the gradient of the model. Algorithm 5 shows the map phase.

Algorithm 5 Map Phase[36]

1: **for** each record $\{\boldsymbol{x_i}, y_i, \boldsymbol{x_i'}\}$ in the data$\{\mathbf{X}, \boldsymbol{y}\}$ **do**
2: Compute the gradient for that record: $g_i = y_i - f_{\boldsymbol{\beta}}(\boldsymbol{x_i})$
3: **for** each active candidate feature $x_{id}' \in \boldsymbol{x_i'}$ **do**
4: Output $g_i : T_d \cup \{g_i\}$

2. **Reduce phase**: Compute the sum of the gradients for each feature. Algorithm 6 shows the reduce phase.

Algorithm 6 Reduce Phase[36]

1: $G_d = 0$
2: **for** each record record $g_i \in T_d$ **do**
3: $G_d = G_d + g_i$

3. **Post-Processing**: In case of categorical features, sum the gradients that belong to the same feature.

To sum up, this algorithm finds the gradients for each feature using the logistic regression formula . Then, the feature that is selected is the feature with the maximum gradient. Finally, it runs the logistic regression model with all the features that have been selected so far in order to find the coefficients of the full model. This step requires multiple passes of the data.

Chapter 4

MMPC for Big Data

In this chapter, we describe how MMPC algorithm (described in section 3.2) can be altered to handle big data with the Map-Reduce model. First, we describe the Map-Reduce jobs that are needed for this task and finally we describe how the G^2, the Fisher Z tests and the Fisher's method can be used with MMPC for big data.

4.1 Using Map-Reduce Model

MMPC is consisting of two phases, where each one is comprised of multiple for loops. So, it is important to have the minimum number of Map-Reduce jobs that is required in order to have accurate results.

The algorithm iteratively computes multivariate (or univariate) associations between a target T, the remaining features R given a subset of the selected features CPC of length max_k. First, for every variable and for every different conditioning subset a Map-Reduce can be performed to compute the appropriate association. However, this is inefficient as several Map-Reduce jobs would be required. So, time will be lost for initializing and completing the job, and for the group-by operation within the job. A more efficient way, is to perform a Map-Reduce jobs to compute all the associations for one feature and for all different conditioning subsets. This will result in more computational intensive Map-Reduce jobs as more associations are needed to be computed. Although this approach is better than the previous, it is not the optimal. The best way in order to exploit fully the Map-Reduce model is to compute all the necessary associations that are needed to find the next variable to be inserted in the set of selected features. That is, the information needed for one iteration of the algorithm.

The information to be extracted from Map-Reduce jobs is dependent on the independence test and so different tests will require different definition of the mapper and the reducer. Moreover, the information to be extracted to compute the associations, needs to have the last selected variable of the algorithm in the conditioning set. This is because in every iteration of the algorithm the tests needed to be performed require the target and each remaining variable given the different sets of the selected variables that contain the last selected variable. All other associations have been computed in previous iterations.

Finally, because of the fact that we are dealing with big data which may have millions of features or instances, small associations in features may be detected and the features will be inserted in the set of selected features. So, it is important to have a number MSF

(user defined) of maximum selected features so that the algorithm will stop when this number is reached. This number bounds the CPC set to have at most MSF features and it is input to the forward phase only. After this phase, the algorithm have selected CPC features and also there are some features that have not yet been filtered out R(remaining features), which would also be in the output of the algorithm because we are concerned with the prediction accuracy of a classifier. Finally, in the CPC set the backward phase would run and output the final set. MMPC pseudocode is shown algorithm 7.

Algorithm 7 MMPC Algorithm

Input: target T, dataset D,max_k,MSF
Output: selected variables CPC, remaining variables R

1: $[CPC, R]$ = ForwardPhase(T,D,max_k,MSF)
2: $[CPC]$ = BackwardPhase(T,D,max_k,CPC)

The forward phase of the algorithm with the use of Map-Reduce model is shown in algorithm 8. Map reduce jobs run in line 5. The *computeAssoc* function extracts the necessary information needed for one iteration of the algorithm. This function input is the dataset (D), the target variable (T), the remaining variables (R) to be tested by the algorithm and the last selected feature F. In this thesis, we use the G^2 and Fisher Z test and so this function can be defined as in algorithm 9. If the data are continuous then the Fisher Z test is used and if the data are discrete G^2 is used. In the first iteration of the algorithm this function will compute the univariate association between T and all variables in D, F is not needed that is why it is initialized to zero. Finally, the conditionally dependence between two variables T and X given a set of variables Z is denoted by $Dep(X;T|Z)$.

Besides using the Map-Reduce model, parallelization can be performed with the use of multiple CPUs. More particularly, we can find the next selected variable to enter the CPC set in parallel (line 6). This parallelization can be performed in the master node.

Finally, for the backward phase one Map-Reduce is needed in every iteration. This is because in every iteration the CPC set may change. Algorithm 10 shows the pseudocode with the Map-Reduce job that is needed (line 2). Now, the remaining variables that is input to the function *computeAssoc* is just the variable that is tested in every iteration. The conditionally independence between two variables T and X given a set of variables Z is denoted by $Ind(X;T|Z)$

The rest of the chapter describes the parallelization of the G^2 and Fisher Z tests. We describe how to extract the necessary information for one iteration of the MMPC algorithm i.e. for the last selected variable and the remaining ones. However, this can be easily expanded to extract the necessary information for not just one iteration. Although, it is not possible to know which variable will be selected in later iterations by the algorithm, the Map-Reduce model can be used to extract information for the top k (user defined) variables. To rank the variables the statistic can be used. So if one of them is to be selected we would not have to perform another Map-Reduce job. The downside of this approach is that it would take more time for the job to finish and also more memory requirements would be needed.

Algorithm 8 Forward Phase[43]

Input: target T, dataset D,max_k, maximum selected features MSF
Output: selected features CPC, remaining features R

$CPC = \emptyset$
$F = 0$
$R =$ all variables in D
do
    $info = computeAssoc(D, T, R, F)$
    parfor $\forall X \in R$ **do**
        $minassocset(X) =$ subset s of CPC that minimizes $assoc(X; T|s)$

    F = variable $\Phi \backslash (\{T \cup CPC\})$ that maximizes $assoc(F; T|minassocset(F))$
    if $Dep(X; T|minassocset(F))$ **then**
        $CPC = CPC \cup F$
        $R = R \backslash F$
    if $|CPC| = MSF$ **then**
        return CPC, R
while CPC has not changed

Algorithm 9 *computeAssoc* Function

Input: dataset D, target T, features R, last selected feature F
Output: information needed $info$

if data continuous **then**
    $info =$ Fisher Z Function
else
    $info = G^2$ Function

Algorithm 10 Backward Phase[43]

Input: target T, dataset D, last selected variable F, max_k, CPC
Output: selected Variables CPC

for $\forall X \in CPC$ **do**
    $info = computeAssoc(D, T, X, F)$
    if $\exists s \subseteq CPC, s.t. Ind(X; T|s)$ **then**
        $CPC = CPC \backslash \{X\}$

4.2 Parallel G^2 Test

The general parallelization algorithm for the G^2 (described in section 3.3.1)f test is shown in algorithm 11. In order to compute the statistic, we first need to compute the appropriate contingency tables. This is performed in the MR_Job (line 1). After computing the tables we need to find the statistics and the pvalues which is performed in lines 2-4. We preferred to compute these values after the Map-Reduce job because that way the reduce function becomes associative and commutative. This means, as described in 2.3.3, that we can use combiners for the job so as to increase performance and reduce the communication cost.

Algorithm 11 G^2 Function

Input: dataset D, remainingVars R, selectedVar s, max_k
Output: pvalues,statistics,tables

1: contingencyTables = MR_Job
2: **for** $\forall cvar \in R$ **do**
3: $statistics[cvar]$ = compute statistic from contingency tables
4: $pvalues[cvar]$ = find pvalue from $statistic[cvar]$
5: **return** *pvalues*, *statistics*

The G^2 test for the MMPC algorithm can be used to find either univariate or multivariate association. To find the univariate association between the target and the other variables the Map-Reduce pseudocode is shown in algorithm 12. The map phase computes all the required contingency tables. That is, between the target and each feature in the dataset. The input to the function is a subset of dataset D, the index of the target feature in the dataset, the number of distinct values each feature takes(stated as *vals*) and the features in the dataset. Line 3 initializes a set *tables* that contain all the contingency tables which are initialized to zero values. For example, $tables\{1\}$ indicates the contingency table for the first feature. It is also worth mentioning that the tables in the set may be of different dimensions as this depends on the different values the feature takes. In addition, from lines 4 - 8 the algorithm passes the data and "fills" the tables. Finally, it outputs for every feature the contingency table for that feature, i.e. the key-value pair $(feature, contingencyTable)$. For instance, a key-value pair is $(1, tables\{1\})$, where the key is equal to one and the value is equal to the contingency table for the first feature. Finally, the reduce function takes a list of all the contingency tables associated with a specific feature and just sums the contingency tables (line 5) for the feature and outputs the complete contingency table for each feature.

In case we need to compute a multivariate association, algorithm 13 is used. For this case the information that is needed to be extracted from the data are the contingency tables between the remaining variables and the target given the set of selected variable of length k that contain the last selected variable. The input to the function is a subset of the dataset(D), the last selected variable (s), the remaining variables (R), the already selected variables (CPC), the max_k, the number of distinct variables each feature takes (V), and the index of the target in the dataset (T). Same notations for the contingency tables have been used as in the univariate association. The only difference is that *tables*

Algorithm 12 Univariate association for G^2 test

Input: dataset D, vals V, target T, features R
Output: contingency tables $contTables$

1: **Map Phase** $(key = 1, value = D)$
2:
3: $tables\{Y|Y \in R\} = \mathbf{0}^{V[T] \times V[Y]}$
4: **for** $\forall$ instance $I \in D$ **do**
5: $t =$ value of target
6: **for** $\forall$ features $j \in I$ **do**
7: $m =$ value of variable j
8: $tables\{j\}[t, m] = tables\{j\}[t, m] + 1$
9: $keyVal = \{< K, V > | K \in R, V = tables\{K\}\}$
10: Output $keyVal$
11:
1: **Reduce Phase** $(key = feature, value = contTables)$
2:
3: $contTable = value\{1\}$
4: **for** $\forall$ table $tab \in value$ **do**
5: $contTable = contTable + tab$
6: Output $(feature, contTable)$

contains besides the feature, also the subset in which the contingency table belongs to. For example, $tables\{1, \{2\}\}$ indicates the contingency table between T, feature 1 conditioned on feature 2. In algorithm 13, initialization of the 3d contingency tables with zeros is performed (line 4). The 3rd dimension of the tables is the multiplication of the distinct values of the variables that are in the conditioning set, which varies from 1 to max_k. From lines 5-12 the algorithm passes the data and "fills" the tables. Finally, it outputs key-value pairs with key a vector that contain the feature and the subset of the conditioning set and values the contingency table associated with that key (line 13). The reduce function just sums all the contingency tables that are associated with the key.

4.2.1 Time Complexity and Communication Cost

Ideally, parallelizing the computation over K machines reduces the runtime by a factor of K. For the computation of the univariate association, the running time of the mapping phase is approximately $O(\frac{N \cdot F}{K})$, where N is the number of records and F is the number of features in the dataset. Finally, the running time of the reduce phase is approximately $O(\frac{V \cdot F \cdot L}{K})$, where L is the number of elements in the largest table among all tables and V the number of tables in the list of values.

The time complexity is different when computing the multivariate association. For the mapping phase the algorithm calculates for all the remaining variables R and for all the different combinations of the selected variables from $k = 1 \dots max_k$ a contingency table. The different combinations of the selected variables are $\sum_{k=1}^{max_k} \frac{n!}{k!((n-k)!} = n^{max_k}$, where n is the number of selected variables minus one, i.e. $n = |CPC| - 1$. So the

Algorithm 13 Multivariate Association for G^2 Test

Input: dataset D, selectedVar s, remainingVars R, selectedVars CPC, max_k, vals V, target T

Output: contingency tables $contTables$

Map Phase $(key = 1, value = D)$

$subsets = \{subs | subs \subseteq S \wedge 1 \leq |subs| \leq max_k \wedge s \in S\}$

$tables\{< Y, subs > | Y \in R, subs \in subsets\} = \mathbf{0}^{V[T] \times V[Y] \times \Pi_{i=1}^{k} V[subs_i]}$

for $\forall$ instance $I \in D$ **do**

  $t =$ value of target

  for $\forall j \in I \cap R$ **do**

    $h =$ value of feature j

    for $k = 1 \cdots max_k$ **do**

      for $\forall sub \in subsets \ \& \ |sub| = k$ **do**

        $o =$ index of the product of values of sub

        $tables\{j, sub\}[t, h, o] = tables\{j, sub\}[t, h, o] + 1.$

$keyVal = \{< K, V > | K \in < R, subsets >, V = tables\{K\}\}$

Output $keyVal$

Reduce Phase $(key =< feature, subset >, value = contTables)$

$contTable = value\{1\}$

for $\forall$ table $tab \in value$ **do**

  $contTable = contTable + tab$

Output $(key, contTable)$

complexity would be $O(\frac{n^{max_k} \cdot N \cdot |R|}{K})$. The time complexity for the reduce phase differs. The worst case is the summation of the largest table, which depends on the max_k value. The larger the max_k the larger the table. Moreover, the larger the max_k the more tables are created for every remaining feature. So the time complexity is $O(\frac{V \cdot L \cdot |R| \cdot n^{max_k}}{K})$, where L is the number of elements in the largest table (this depends on the value of k) and V the number of tables in the list of values.

As mentioned above, for the map phase the algorithm calculates for all the remaining variables R and for all the different combinations of the selected variables from $k = 1 \ldots max_k$ a contingency table. The different combinations of the selected variables are as previously $\sum_{k=1}^{max_k} \frac{n!}{k!((n-k)!} = n^{max_k}$. Lastly, the contingency table is a 3d table with size $L = |W| * |T| * |Z|$, where $|W|$ and $|T|$ is number of values the features take and $|Z| = |Z_1| * |Z_2| * \cdots * |Z_k|$ is the multiplication of the number of values each feature in set Z takes. In our case $|W|$ is a variable from the remaining set, $|T|$ is target variable and $|Z|$ is the conditioning set, which depends on the number k. Taking all these into account, the communication cost is $O(|R| \cdot L \cdot n^{max_k})$.

4.2.2 Example

To make the process of calculating the contingency tables clear we describe a simple example. Consider the following data matrix:

Id	T	F1	F2
1	0	1	0
2	0	1	0
3	1	0	1
4	0	1	1
5	1	1	0
6	1	0	1
7	1	0	1
8	1	1	0
9	0	0	0
10	0	1	0
11	1	1	1
12	0	1	1
13	0	1	1
14	0	0	0
15	0	0	1

The target variable is T and the features of the data are the $F1$ and $F2$. Using the G^2 function we want to calculate the contingency tables. This process is shown in figure 4.1.

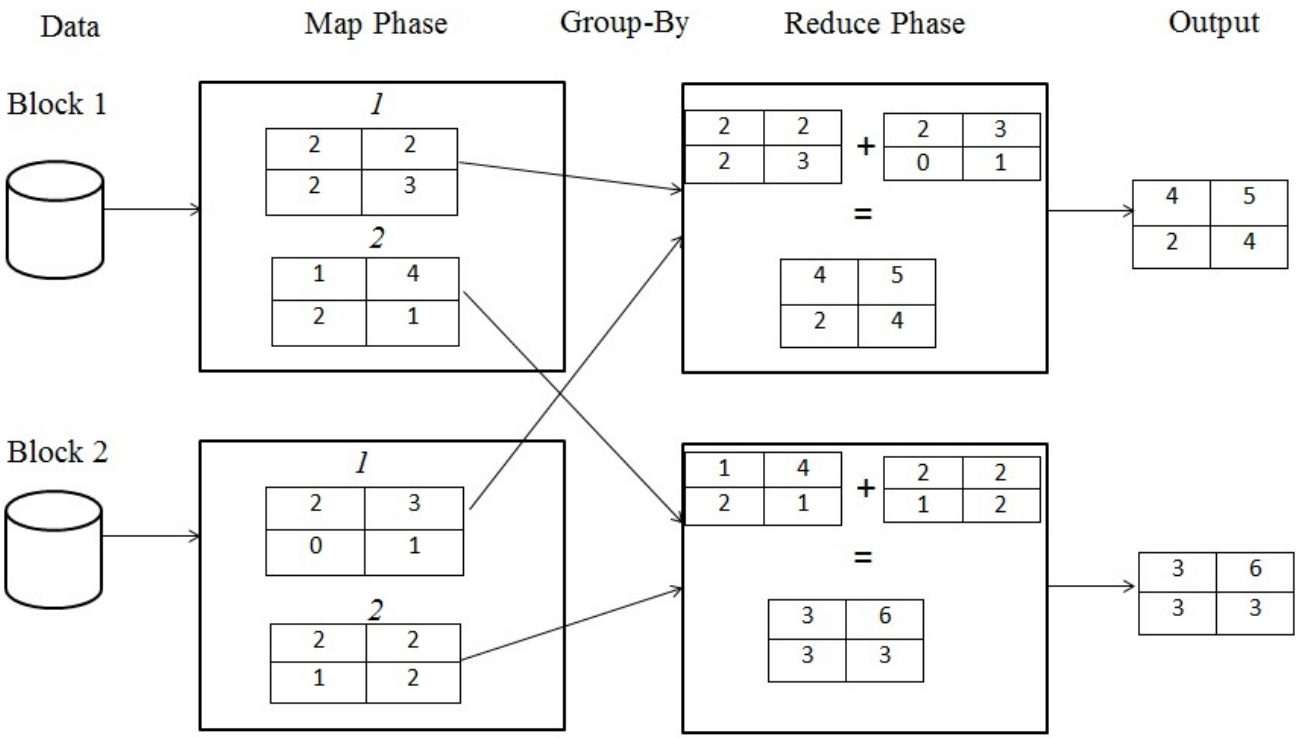

Figure 4.1: Calculating contingency tables. Each block contains a subset of the data. So in the two mappers the contingency tables for each feature are calculated. Then for each feature the tables are grouped and inserted to different reducers which just sum the tables to form the total table and output this table.

To begin with, the data are split in two blocks and each block is processed by a map task. The map phase computes the contingency tables between the target and each of the two features for each block. Then, the group-by operation takes place and for each feature the contingency tables are grouped and input to the reducers. In the reduce phase the tables are summed and output to get the final outcome.

4.3 Parallel Fisher Z Test

In this section, we describe how we parallelized the Fisher Z test (described in section 3.3.2). For the Fisher Z test we only need to find the covariance matrix. For that reason we only need the covariance values between two-pair variables. Because we are programming in R it is vital to vectorize the computations so as to reduce time. The covariance between two features can be written with the following form:

$$\omega_{XY} = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{x_i} - \overline{\boldsymbol{x}})(\boldsymbol{y_i} - \overline{\boldsymbol{y}})^T = \frac{1}{n-1}(\sum_{i=1}^{n}(\boldsymbol{x_i y_i}) - n(\overline{\boldsymbol{xy}}))$$

So in order to find the covariances only the product $(\boldsymbol{x_i y_i})$ is required. However, before computing the covariances, we need to know the mean for every variable, in case the data are not centered. The mean values of the features can be found before running MMPC, so as to compute them only once and not in every iteration of the algorithm. So the mean values would be input when computing those values. Algorithm 14 shows the pseudocode for computing the covariance values. In this algorithm, the first step is to compute the products between the selected variable and the remaining variables. This is performed in line 1 with the use of Map-Reduce. The next step is to subtract from the products the product of the number of instances and the mean values of the selected and the remaining variables (lines 4 - 5).

Algorithm 14 Fisher Z Function

Input: dataset D, #instances N, selectedVar s, remainingVars R, meanValues M
Output: covariances

1: $products = findProducts(D, s, R)$
2:
3: $covValues = \mathbf{0}^{1\times R}$
4: **for** $\forall var \in R$ **do**
5: $\quad covValues[var] = products[s] \cdot products[var] - N(M[s] \cdot M[var])$
6: **return** $covValues$

The pseudocode for computing the mean values of the features is shown in algorithm 15. The map function sums all the rows of the dataset (lines 4 - 6) and the reduce function sums all the vectors the map phase outputs. Finally, the total summation of the variables is divided by the number of instances in the dataset. For this job we used combiners.

After computing the mean values we need to compute the products between the target and the required features. We can find these values with 1 Map-Reduce job which is independent of the association (univariate or multivariate). Algorithm 16 shows the

Algorithm 15 Computing Mean values

Input: dataset D, #instances N
Output: meanValues

Map Phase $(key = 1, value = D)$

$sumValues = \mathbf{0}^{1 \times features}$
for $\forall$ instances $I \in D$ **do**
    for $\forall$ feature $c \in I$ **do**
        $sumValues[c] = sumValues[c] + I[c]$
Output $(1, sumValues)$

Reduce Phase $(key = 1, value =$ vector of sums$)$

$totalSum = \mathbf{0}^{1 \times features}$
for $\forall$ vector $vec \in value$ **do**
    $totalSum = totalSum + vec$
Output $(1, totalSum)$

After MR job

$meanValues = totalSum/N$
return $meanValues$

map and reduce functions. The map function calculates the products for the target and the remaining variables (lines 4 - 6) and outputs a vector with all these values. The reduce function just sums all the vectors. For this Map-Reduce job we used combiners. Moreover, due to the fact that we only need to calculate a vector we set the output key of the map function equal to one.

It is important to mention here that after the algorithm selected the next feature, the covariances related to this feature are stored in memory in order to be used in next iterations and also in the backward phase. This is very important not so much for the forward phase but for the backward phase as it will not be needed to pass the data again.

Algorithm 16 Function findProducts

Input: dataset D, remainingVars R, selectedVar s
Output: covariances

1: **Map Phase** $(key = 1, value = D)$
2:
3: $sums = \mathbf{0}^{1 \times R}$
4: **for** $\forall$ instances $I \in D$ **do**
5: **for** $\forall$ features $c \in I \cap R$ **do**
6: $sums[c] = sums[c] + I[s] \cdot I[c]$
7: Output $(1, sums)$
8:
1: **Reduce Phase** (key = 1, value = sum of products)
2:
3: $products = \mathbf{0}^{1 \times |R|}$
4: **for** $\forall$ vector $vec \in value$ **do**
5: $products = products + vec$
6: Output $(1, products)$
7:

4.3.1 Time Complexity and Communication Cost

Like in G^2, parallelizing this computation over K machines reduces the runtime by a factor of K. The time complexity of the map phase for computing the mean values is $O(\frac{N \cdot F}{K})$, where N is the number of instances and F is the number of features in the dataset. The time complexity for the reduce phase is $O(F \cdot V)$, where V is the number of vectors in the list of values.

As mentioned earlier the mean values will need to be calculated only once. The product of the target and the remaining features, however, is needed to be computed in every iteration. For this computation, the running time of the mapping phase is approximately $O(\frac{N \cdot |R|}{K})$, where N is the number of records and $|R|$ is the number of remaining features to test. Similarly, the running time of the reduce phase is $O(|R| \cdot V)$, where V is the number of vectors in the list of values. When computing the univariate association the remaining variables are considered all the features in the dataset.

Finally, the communication cost is $O(|R|)$, due to the fact that only a vector of length equal to the number of the remaining features is input in the reduce task.

4.3.2 Example

To make the process of calculating the covariances clear we describe a simple example. Consider the following data matrix:

Id	T	F1	F2
1	0	0.28	0.81
2	0	0.3	0.9
3	1	0.13	0.13
4	0	0.22	0.43
5	1	0.02	0.32
6	1	0.52	0.19
7	1	0.2	0.32
8	1	0.12	0.63
9	0	0.17	0.58
10	0	0.32	0.2
11	1	0.41	0.9
12	0	0.48	0.12
13	0	0.21	0.8
14	0	0.32	0.26
15	0	0.18	0.73

Figure 4.2 shows how the product between the target and each feature is computed. To begin with, the data are split in two blocks and each mapper is processing them in parallel. Then each mapper computes the inner product between the target and each feature and the results are stored in a vector. Then, the vector for each map task is grouped by and processed by one reducer. The reducer just sums the two vectors and outputs the final result. Finally, the result is subtracted by the product of the means of the target and each feature in order to compute the covariances. This last step is not shown in the figure.

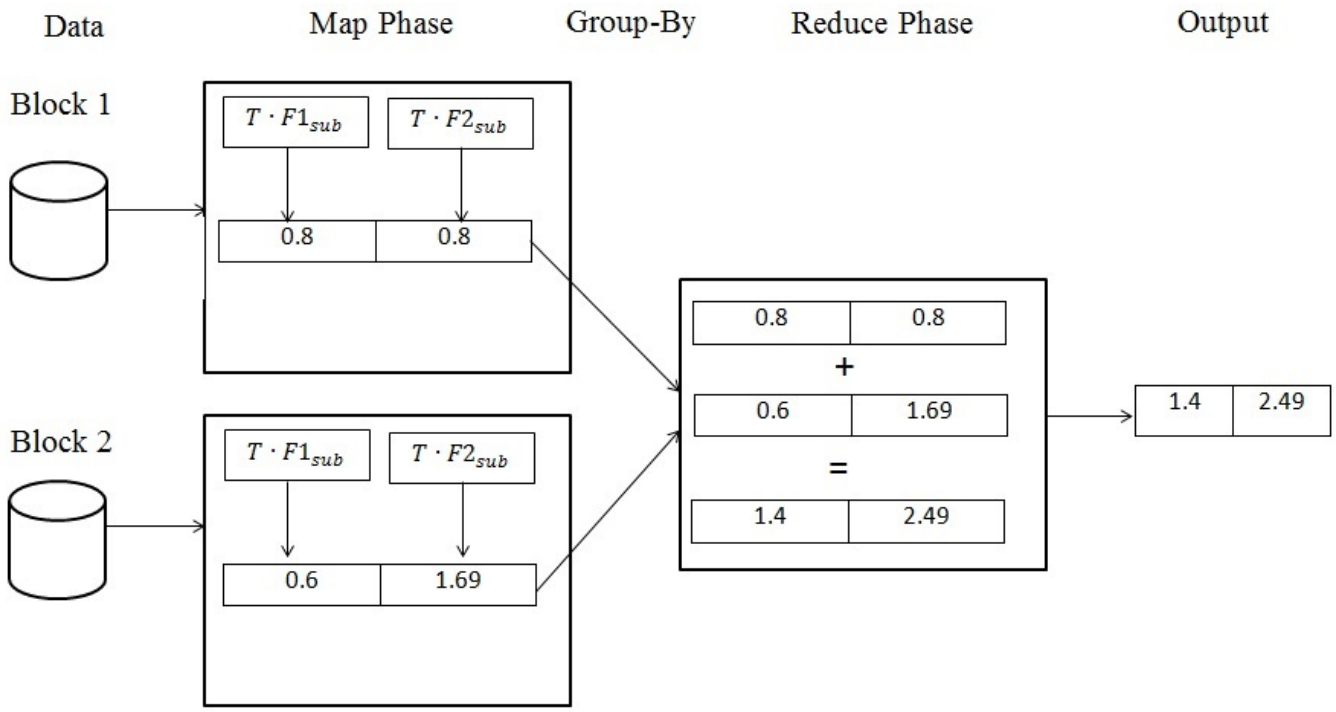

Figure 4.2: Calculating covariance values. On each block the mappers compute the dot product for each feature and the target and insert each value to a vector which is the output of the mappers. The two vectors created from the two blocks are the input to the reducers which just sum them and output the result.

4.4 Using Fisher's Combined Probability Test

In the previous two sections, we described how the Fisher Z and G^2 test can be integrated with MMPC algorithm using the Map-Reduce model. In this section, we use the Fisher's combined test with MMPC. This method combines pvalues that were produced with the same independence test in order to find the total statistic. This method can be very useful for with MMPC because the user can use any test of interest without worrying how to use Map-Reduce model. As a result, MMPC can run with any type of data if the appropriate test is used.

In order to use this method, the map function needs to find the pvalues required using a particular test and the reduce function will then sum the logarithm of the pvalues. Finally, the statistic can be found after the Map-Reduce process by multiplying with -2. The procedure is shown in algorithm 17. Line 1 computes all the necessary logarithmic sums and in lines 3 and 4 the statistic and the pvalue for a particular feature are found.

Algorithm 17 Combining p-values

Input: dataset D, remainingVars R, selectedVar s
Output: $pvalues, statistics$

1: vals = MR_Job
2: **for** $\forall$ values $i \in vals$ **do**
3: $\quad statistics[i] = -2 * vals[i]$
4: $\quad pvalues[i]$ = compute pvalue from statistic
5: **return** $pvalues, statistics$

For the univariate association the remaining variables are all the features in the dataset. In order to find univariate association, the Map-Reduce job is shown in algorithm 18. In this algorithm, all the pvalues between the target feature and all the features in the dataset are computed (lines 4 - 5). These values are stored in a vector which is the output of the map function (line 6). The reduce function takes the vectors of pvalues and computes the product between the vectors element wise (lines 3 - 6). Finally, it outputs the product vector (line 7). The key is set to one because the output is just one vector.

When the multivariate association needs to be computed, algorithm 19 is used. First, an initialization of the vector that will store the pvalues, is performed (line 4). Then, the algorithm computes all the pvalues between the target T and every feature in remaining variables given all the possible subsets (of length 1 to max_k) of the selected variables that contain the last selected variable (lines 5 - 10). Again, in this case the key is set to one for the same reason as previously.

4.4.1 Time Complexity and Communication Cost

Like in the previous tests, parallelizing this computation over K machines reduces the runtime by a factor of K. For the univariate association, the time complexity is $O(\frac{P \cdot |R|}{K})$, where P is the additional time complexity of the test used and |R| the number of remaining features. If the G^2 is used for example the time complexity would be $O(\frac{N \cdot |R|}{K})$, where N is

Algorithm 18 MR for Univariate Association

Input: dataset D, remainingVars R, target T
Output: statistics *stats*

Map Phase

$pvalues = \mathbf{0}^{1\times|R|}$
for $\forall$ features $X \in R$ **do**
    $pvalues[X] = pvalue(X;T)$
Output $(1, pvalues)$

Reduce Phase (key = 1, value = pvalues)

$pvalues = \mathbf{0}^{1\times|R|}$
for $\forall$ vectors $v \in value$ **do**
    for $\forall$ pvalues $p \in v$ **do**
        i = index p in v
        $pvalues[i] = pvalues[i] + ln(p)$
keyval$(1, pvalues)$

Algorithm 19 MR for Multivariate Association

Input: dataset D, remainingVars R, target T, selectedVar s, selectedVars CPC, max_k
Output: statistics *stats*

Map Phase

$subsets = \{subs | subs \subseteq S \wedge 1 \leq |subs| \leq max_k \wedge s \in sub\}$
$pvalues = \mathbf{0}^{1\times|subsets|}$
for $\forall$ features $X \in R$ **do**
    for $ck = 1 \cdots max_k$ **do**
        $condSet = \{sub | sub \subseteq S \wedge |sub| = ck \wedge s \in S\}$
        for $\forall$ conditioning set $sub \in condSet$ **do**
            i = index of X given sub
            $pvalues[i] = pvalue(X;T|sub)$
Output $(1, pvalues)$

Reduce Phase (key = 1, value = pvalues)

for $\forall$ vectors $v \in value$ **do**
    for $\forall$ pvalues $p \in v$ **do**
        i = index p in v
        $stats[i] = stats[i] + ln(p)$
Output $(1, stats)$

the number of records. For the multivariate association the different combinations of the selected variables are as mentioned also in subsection 4.2.1 $\sum_{k=1}^{max_k} \frac{n!}{k!((n-k)!}$. So the time complexity will be $O(\sum_{k=1}^{max_k} \frac{n!}{k!((n-k)!} \frac{R \cdot P}{K})$, where again P is the additional time complexity of the test used. In case G^2 is used the time complexity would be $O(\sum_{k=1}^{max_k} \frac{n!}{k!((n-k)!} \cdot \frac{N \cdot |R|}{K})$, where N is the number of records in the dataset. The time complexity of the reduce function for both univariate and multivariate association is $O(|R| \cdot |V|)$, where $|V|$ is the number of vectors in the list of values.

The communication cost for the univariate association is $O(|R|)$, due to the fact that only a vector of length equal to the number of features is input in the reduce task. Moreover, for the multivariate association the communication cost is $O(|R| \cdot |subsets|)$, where $|subsets|$ is the number of different conditioning subsets.

4.4.2 Example

To make the process of calculating the statistic clear we describe a simple example. Consider the following data matrix:

Id	T	F1	F2
1	0	0.28	0.81
2	0	0.3	0.9
3	1	0.13	0.13
4	0	0.22	0.43
5	1	0.02	0.32
6	1	0.52	0.19
7	1	0.2	0.32
8	1	0.12	0.63
9	0	0.17	0.58
10	0	0.32	0.2
11	1	0.41	0.9
12	0	0.48	0.12
13	0	0.21	0.8
14	0	0.32	0.26
15	0	0.18	0.73

Figure 4.3 shows how to compute the sum of the logarithms of the association between the target and each feature. To begin with, the data are split in two blocks and each map task is processing them in parallel and computes the association between the target and each feature. Then the results are stored in a vector. The association is defined by the user. In this case, we used the logistic regression. Then, the vector for each map task is grouped by and processed by one reducer. The reducer just logarithm the values in the vectors and then sums the two vectors and outputs the final result. Finally, the result is multiplied by -2 to get the statistic for each association. This last step is not shown in the figure.

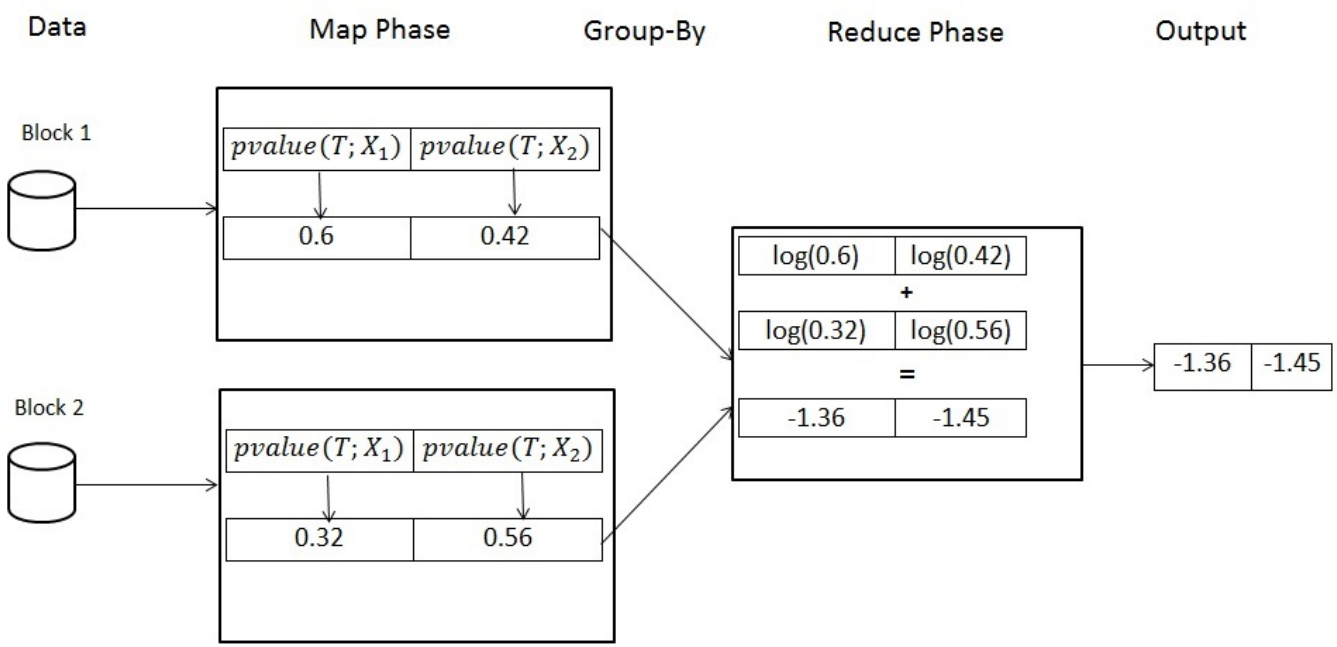

Figure 4.3: Computing the sum of log-pvalues. On each block the pvalues between the target and each feature is calculated and inserted in a vector. Then this vector is inserted to the reducers which find the log for each value, sum the vectors and output the result.

Chapter 5

Experiments

In this chapter, we present the experimental evaluation of our feature selection algorithm. First, we describe the datasets we used and the architecture of the cluster. Then, we show the results for three kind of experiments: changing the number of nodes, the number of instances and using a classifier to evaluate the feature selection. Finally, we present experiments using the Fisher's method.

5.1 Datasets

The datasets that were used for the thesis are shown in table 5.1. We tried to evaluate our algorithm with datasets of different number of features and size.

Dataset	#Instances	#Features	#Classes	Type	Size(Gb)
HIGGS	11,000,0000	28	2	Real	8
SUSY	5,000,0000	18	2	Real	2.4
RCV1	534,135	47,236	2	Real	0.8
Epsilon	500,000	2,000	2	Real	9
Zeta	500,000	2,000	2	Real	7.6
Mnist8	8,100,000	784	10	Real	19
URL	2,396,130	3,231,961	2	Real/Binary	2.1

Table 5.1: Datasets used for experimentation with information about the number of instances/features/classes,type and the size of each one.

The HIGGS[47] and SUSY[48] datasets are related to Physics & Astronomy and have been built from official ATLAS full detector simulation. They contain features that are kinematic properties measured by the particle detectors in the accelerator and high-level features derived by physicists. The labels indicate if the instance is signal or background. The RCV1[49] dataset is a text dataset and each instance is a document. The features are the within document weights, assigned to each term in the document. The Epsilon[49] and Zeta[50] datasets are artificial datasets that were created for the Pascal large scale learning challenge in 2008. The features of these datasets are scaled to N(0,1) and each instance is normalized to a unit vector. Furthermore, Mnist8[49] consists of images of size

28x28 scaled to $[0, 1]$, where each pixel is a feature. The labels indicate a number from 0 to 9. Finally, the URL[49] dataset consists of approximately 2 millions URLs (instances). The label of the dataset is whether the url is malicious(phishing, spam etc) or not. The features contain different kind of information like DNS, and geographic information.

Due to the fact that all datasets contain real values and the G^2 test is used with discrete features, we discretized the datasets. That is, we convert the real values to discrete values and in our case to four values. This was accomplished with equal bins method. This method takes a vector that contains real values and first finds the minimum and maximum values of the vector. Then, it divides it to k equal width intervals, and in our case to three. The values that are in the first interval take value zero, those in the next interval take value one and so forth. We did not run the G^2 test for the URL dataset, because G^2 test cannot handle millions of features, at the time. By discretizing the datasets their size is decreased. Table 5.2 shows their size after discritization.

Dataset	Size(Mb)
HIGGS	608
SUSY	185
RCV1	140
Epsilon	1,860
Zeta	1,950
Mnist8	5,680

Table 5.2: Size of datasets after discretization.

5.2 Cluster Architecture

Our cluster contains 7 nodes and on each node a virtual machine(VM) runs. The characteristics of the VMs are the following: 6 cores, 6 Gb RAM, and 100 Gb of disk space. Each VM runs Ubuntu server 12.04 and the nodes were connected with 1Gbps Ethernet. Finally, the Hadoop 1.2.1 version was used. The configurations we used for Hadoop are described in Appendix A.3.

5.3 Evaluation

The experiments we performed to evaluate our algorithm were the following:

- Instances vs Time: Using the whole cluster (i.e. 7 nodes), we varied the percentage of instances used for each dataset to take the following values: 25%, 50%, 75% and 100%. For these experiments it is important to notice that for different percentages of the datasets different features may be selected. This is because the features to be selected are dependent on their distribution in the dataset. In addition, for these experiments it is important to take into account the number of map tasks that are initialized for each of the percentages of the datasets because this directly affects the time. Tables 5.3 and 5.4 shows the map tasks that are initialized for both the

Fisher Z test and the G^2 test, respectively. From table 5.4 we can see that the SUSY and RCV1 datasets are very small, and so there is no point of running this experiment. Moreover, from table 5.3 the dataset RCV1 is also small and so also for this dataset we did not run this kind of experiment.

Dataset	25%	50%	75%	100%
HIGGS	30	60	90	120
SUSY	9	18	27	36
RCV1	3	7	10	13
Epsilon	37	73	102	146
Zeta	30	59	88	117
Mnist8	73	145	218	291

Table 5.3: Number of map tasks need to be initialized for the different percentages of the datasets when using the Fisher Z test.

Dataset	25%	50%	75%	100%
HIGGS	3	5	8	10
SUSY	2	2	3	3
RCV1	2	2	2	3
Epsilon	8	16	24	30
Zeta	8	15	23	30
Mnist8	23	46	69	91

Table 5.4: Number of map tasks need to be initialized for the different percentages of the datasets when using the G^2 test.

- Nodes vs Time: For these experiments we varied the number of nodes in the cluster and we use all the instances for each dataset. The nodes we used were 1,3,5 and 7. As noted earlier, on each node 6 mappers could run in parallel. Table 5.5 shows the number of map tasks that could run in parallel when changing the number of nodes.

Nodes	#Maps
1	6
3	18
5	30
7	42

Table 5.5: Map tasks that can run in parallel when changing the number of nodes in the cluster

- Classification Accuracy: For these tests we split the datasets to two sets: 75% training and 25% test. We run the logistic regression classifier and measured its

accuracy before and after the feature selection using MMPC and SFO. For both algorithms we had an upper limit of 100 features to select so as for the algorithms to finish in 4 days period. This was the case for the RCV1, Mnist8 and Zeta datasets. For these datasets and for the MMPC algorithm we found the classification accuracy by performing two tests. For the first test we used only the 100 features the algorithm selects and for the second we used these 100 features and the remaining features, i.e. the ones not filtered out by MMPC when stopped at 100th iteration. The classifier could not run on the URL dataset for the millions of features and we do not present any results. Finally, for the Mnist8 dataset we run the algorithm with the G^2 test because it has multiple classes and Fisher Z test can run only with binary class. For the other datasets the Fisher Z test was used.

- Using Fisher's Combined Probability Test: For this test we run the G^2 test and compare it with the Fisher's combined probability method. We did three comparisons. First, we show the number of features that are selected by both methods and the number of those that are same between them. Moreover, we show the classification accuracy using the logistic regression classifier, in order to evaluate the method. Finally, we present the time taken for both methods to finish.

Finally, MMPC runs with the following settings: for the Fisher Z test $max_k = 2$, $threshold = 0.01$ and for the G^2 test $max_k = 1$, $threshold = 0.01$. We chose to change the value of max_k for the G^2 test because it requires substantial amount of memory that is not available when one reduce task is available. Moreover, we also run MMPC with the threshold that was set with the use of Storey's method (described in Chapter 3). We present the results in the appendix. For this experiment, we set $fdr = 0.05$ and $\lambda = 0.05$. The results using the Storey's method are shows in appendix B.

It is important to notice that the parameters we use with MMPC affects the number of features that are to be selected. The fewer the threshold the less features are to be selected and also the lower the max_k the higher the number of selected features. This is an important factor for MMPC because the user can control the number of features he or she wants to select.

5.3.1 Instances vs Time

For these experiments, the time that the algorithm takes to finish, for different datasets, varies significantly. Because of this for each dataset we divided the time of each experiment with the time taken with 25% of the instances for each dataset. That is,

$$relative\ running\ time = \frac{time\ with\ X\%\ instances}{time\ with\ 25\%\ instances}$$

where $X = 25\%, 50\%, 75\%, 100\%$. Figures 5.1 and 5.2 shows the results for the Fisher Z test and the G^2, respectively.

From figure 5.1 we can see that the fewer instances a dataset has, the less time takes for the algorithm to finish. This is due to the decrease of the size of the datasets. Moreover, the relative time increases when the percentage of datasets is also increased. In addition, the larger the size of the dataset the higher the increase of the relative time. For example, for the Mnist8, Epsilon and Zeta datasets the increase is higher than the other 2 datasets.

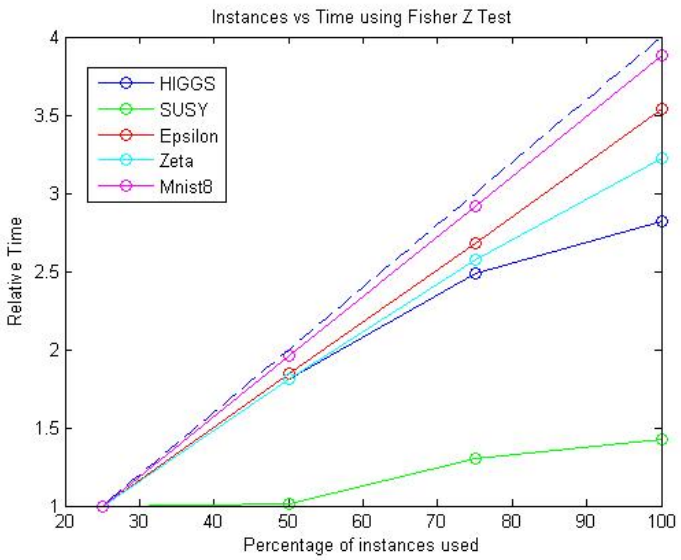

Figure 5.1: Relative running time using Fisher Z test

These were expected as the larger the dataset the more mappers are initialized for the job. Furthermore, for the HIGGS dataset we can see that when 100% of the instances are used the increase is not as high as when going from 50% to 75%. This is because fewer features are selected and so fewer iterations are performed. Finally, for the SUSY dataset the relative time increases when 75% of the dataset is used. This is because of the small size of the dataset.

From figure 5.2 the same conclusions as in Fisher Z Test can extracted. That is, the less the instances of the datasets the smaller their size and less time takes for the algorithm to finish. This is the case for the Mnist8 and Epsilon datasets. Because of the fact that Mnist8 has larger size than the Epsilon dataset the increase in time is higher when more instances are used. For the HIGGS dataset we see something peculiar. When 50% of the instances are used the less time it takes for the algorithm to finish. This happens because of two factors. First, the algorithm for this percentage of instances needs one iteration less than when 25% of the instances are used. Second, because the size of the dataset is small, the difference of map tasks initialized between 25% and 50% is also small and one iteration less affects the graph. Finally, in contrast to the results in Fisher Z test, the increase in relative time does not approach the optimal case. This is because the size of the datasets is much smaller in this case.

5.3.2 Nodes vs Time

For these experiments, the time that the algorithm takes to finish, for different datasets, varies significantly. Because of this for each dataset we divided the time when 1 node is used with the time when different nodes are used. This is called the speed up and it is defined as follows:

$$Speed\ Up = \frac{time\ with\ 1\ Node}{time\ with\ K\ Nodes}$$

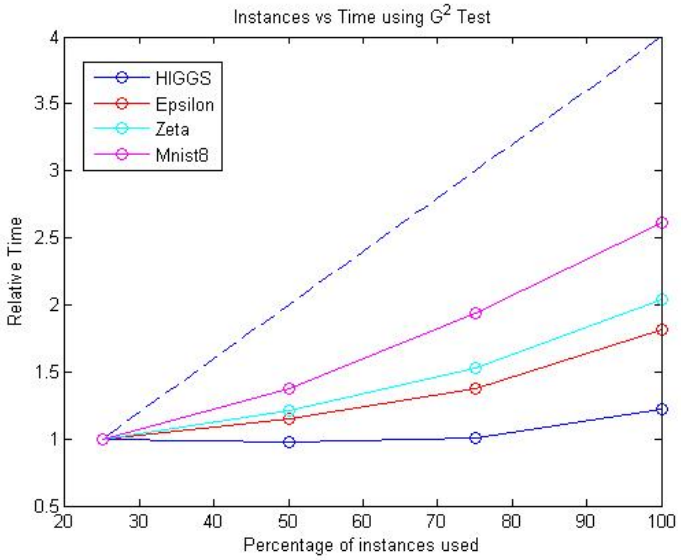

Figure 5.2: Relative running time using G^2 test

where $K = 1, 3, 5, 7$. Figure 5.3 and 5.4 shows the results for the Fisher Z test and the G^2 test, respectively. The ideal speed up is the diagoanl in the figures.

From figure 5.3 we can conclude that the more nodes we use the less time it takes for the algorithm to finish. This is because more map tasks can run in parallel. However, this is not the case for the RCV1 dataset where the relative time decreases when 3 nodes are used and then remains unchanged when more nodes are used. This is due to the fact that the dataset is small and only 13 mappers are needed. When 1 node is used 6 mappers can run in parallel and when 3 nodes are used 18 can run in parallel which are more than enough for this dataset. Moreover, for the other datasets we can see that the larger the size of the dataset the higher the decrease of the relative time when more nodes are used. For example, for the Mnist8 dataset, which is the largest dataset, the relative time approaches the optimal time when 3 and 5 nodes are used, but it seems to move away from the optimal when 7 nodes are used. This is also the case for the Zeta and Epsilon datasets. Finally, the smaller the dataset the higher the distance from the optimal(e.g. SUSY).

As in the Fisher Z test, from figure 5.4 we can conclude that the more nodes we use the less time it takes for the algorithm to finish. More particularly, for the HIGGS and SUSY datasets we can see that after 3 nodes the speed up remains the same. This is due to the small size of the datasets. In addition, the larger the size of the dataset the higher the decrease in the speed up. This is the case for the Epsilon and Mnist8 datasets, where the time reaches the optimal. Finally, as the number of nodes are increased the relative time for the datasets moves away from the optimal time. This can be seen clearly if we compare the speed up of the Mnist8 dataset when the Fisher Z test and the G^2 test is used. In the Fisher Z test the speed up is closer to the optimal than when the G^2 test is used.

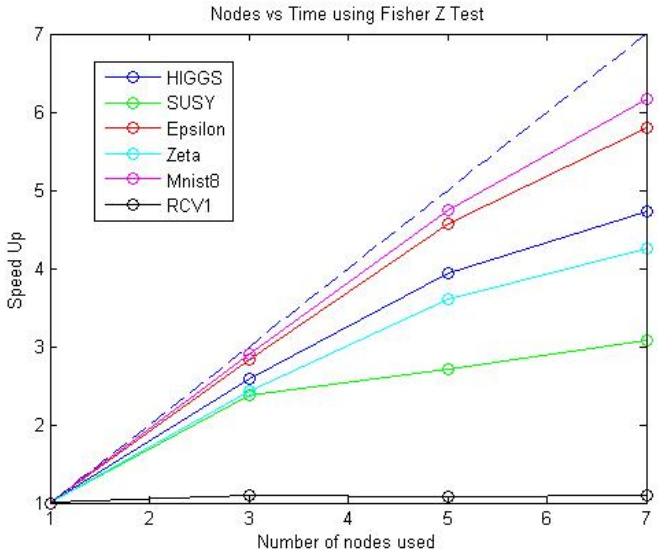

Figure 5.3: Speed up when using Fisher Z test. The diagonal shows the optimal case. The larger the size of the dataset the greater the decrease in time which approximates the optimal case.

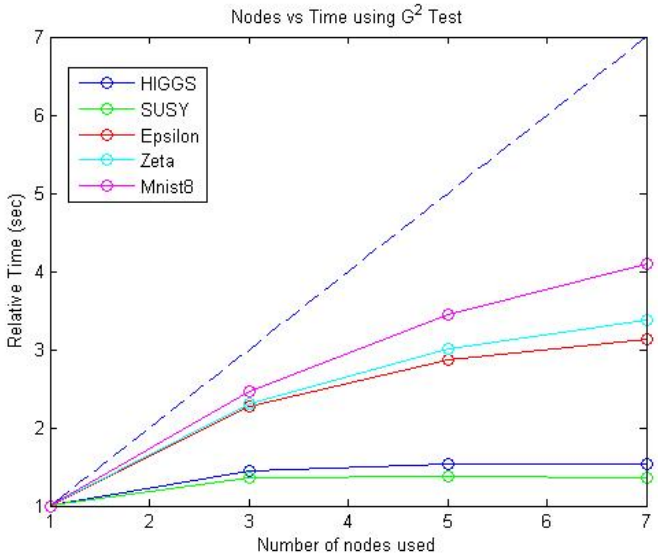

Figure 5.4: Speed up when using G^2 test. Increasing the number of nodes decreases the total time needed for the algorithm to finish. Also, the greater the size of the dataset the greater the decrease of time (approaches optimal case - diagonal).

5.3.3 Classification Accuracy

The classifier we used to evaluate the feature selection for both MMPC and SFO is the logistic regression. This classifier finds the probability $P(Y|X)$, where Y is the feature we want to predict and can take discrete values and X is any vector containing continuous or discrete values. The number of features selected by MMPC and SFO are shown in table 5.6. The number in the parentheses for the MMPC algorithm is the number of remaining

features.

Dataset	#Features	#MMPC	#SFO
HIGGS	28	12(0)	17
SUSY	18	11(0)	3
RCV1	47,236	100(6712)	-
Epsilon	2,000	82(0)	100
Zeta	2,000	100(88)	100
Mnist8	784	100(643)	-
URL	3,231,961	100(2242)	-

Table 5.6: Number of features of each dataset and the selected features for MMPC and SFO. In parenthesis for MMPC is the number of remaining features.

From table 5.6 we can see that MMPC algorithm selects less than half the features for the HIGGS and SUSY datasets. The SFO algorithm selects more features for the HIGGS dataset (17) and only 3 features are selected for the SUSY. In addition, the selected features using the MMPC algorithm for the Epsilon dataset are less than 5% of the total features. For the SFO algorithm we use the first 100 selected features. In addition, for the Zeta dataset we use the first 100 features that are selected by both MMPC and SFO. For the other datasets the SFO cannot run in a reasonable time.

For the Mnist8 dataset MMPC selects 100 features and the remaining ones are 643. The large number of features that are selected can be explained if we consider the fact that this dataset consists of 10 classes. In general, the more classes a dataset has the more features are needed to discriminate between them. So in order to discriminate between the classes many features are needed. Finally, for the URL dataset the algorithm selected 2242 features. The number of selected features are small. This may be the case because this dataset has only two classes.

Figure 5.5 shows the classification accuracy when the logistic regression classifier is used for the selected features of both the MMPC and SFO algorithms but also when all the features are used. Moreover, for MMPC the figure shows the results when the remaining variables are used along with the selected ones.

From the results, we can see that the classifier for the datasets HIGGS and SUSY perform almost the same and it is independent of the feature selection algorithm. This is not a surprise due to the fact that these datasets consists of few features. For the Epsilon dataset the accuracy decreases (only) slightly, which is important fact because few features are selected from both MMPC and SFO algorithms. The accuracy of the classifier for this dataset is higher when the SFO features are used. However, this is expected because 100 features are selected by SFO in contrast to the 82 features that MMPC selects. Finally, for the Zeta dataset the features selected by both MMPC and SFO increase the performance of the classifier significantly.

For the RCV1 and the Mnist8 dataset the SFO could not finish in reasonable time and so we run only MMPC. RCV1 dataset has a high decrease in its accuracy when only the 100 selected variables are used for classification. However, if the remaining variables are added the classification accuracy decreases slightly. Finally, the Mnist8 dataset has

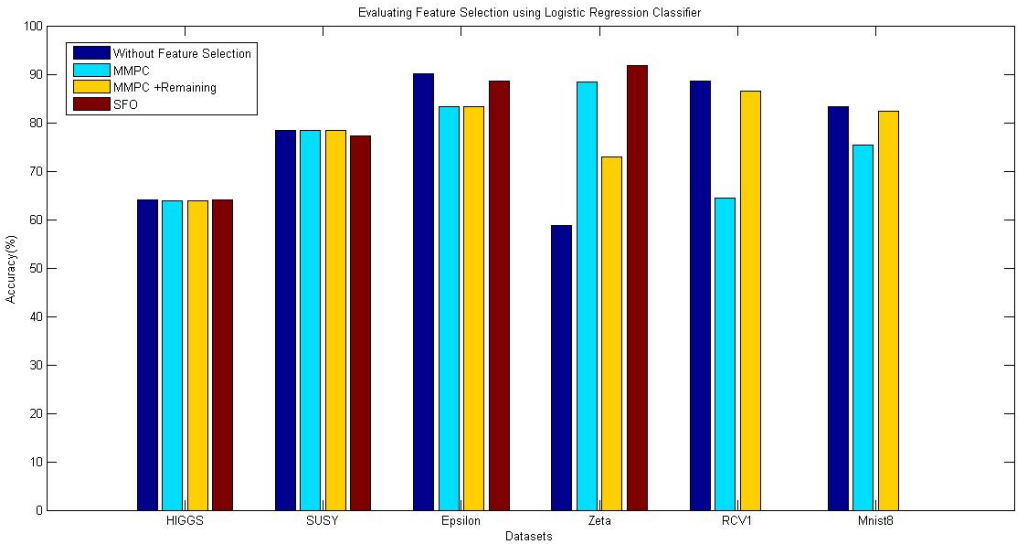

Figure 5.5: Classification accuracy of logistic regression using the features selected by MMPC, the features selected and the remaining of MMPC (indicated as MMPC+), the selected features of SFO. Finally, for the binary datasets the distribution of the most frequent class is displayed.

a slight decrease in its accuracy when the first 100 features are selected. This is an important fact because the dataset consists of 10 classes and only 100 features are used.

Due to the fact that 5 datasets contain binary target it is important to show the class distribution for the test set of these dataset so as to conclude that the feature selection benefits the classification. Figure 5.5 shows the distribution of the most likely class. From the results we can see that the distribution of the two classes for all the datasets is nearly 50%. As a result, the feature selection benefits the classification.

The last comparison we make is the time taken for the MMPC and for SFO to finish. Figure 5.6 shows the logarithmic time taken by both algorithms. From the results we can see than MMPC takes less time in all datasets to finish. This was expected as SFO needs to pass the data more times than MMPC. More particularly, it is worth mentioning that although for the SUSY dataset MMPC selects more features than SFO the time taken is lower than SFO. Moreover, for the Epsilon dataset the difference is the highest. This was expected as MMPC selects fewer features than SFO. In Zeta dataset the difference is smaller than in Epsilon dataset because the features selected by MMPC are the same as for the SFO.

5.3.4 Using Fisher's Combined Probability Method

As described in section 4.4, Fisher's combined probability method can be used with any statistical test. We compare this method with the G^2 test because we expect to take less time the computation of the Fisher's combined probability method. When computing the contingency tables in G^2 test different key is used for different contingency table and so

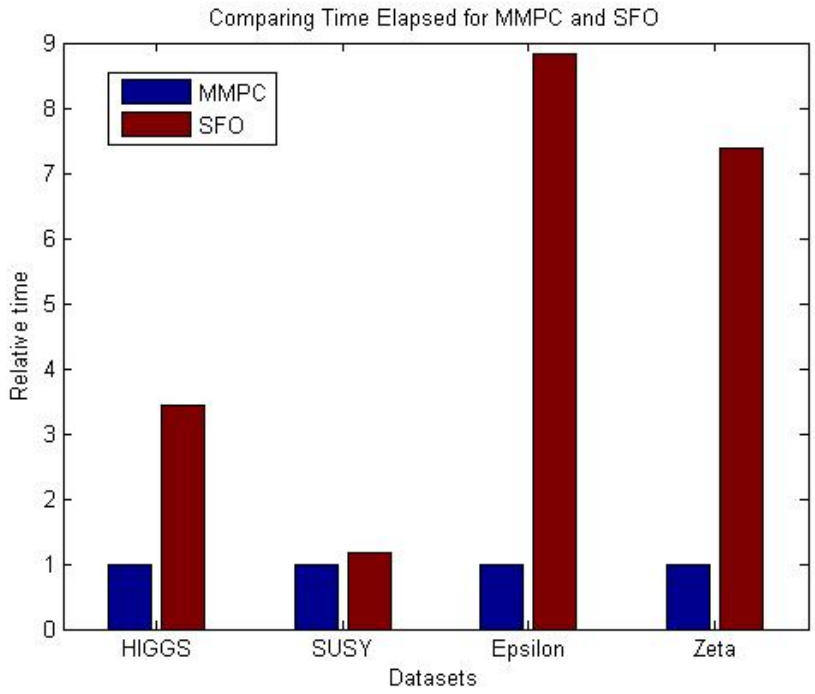

Figure 5.6: Comparing relative time between MMPC and SFO. For the large datasets MMPC is at least 7 times faster.

the group-by operation in Hadoop takes a substantial amount of time. Using combining pvalues the key is equal to one and only a vector needs to be output from the map phase. That is why we expect this method to take less time.

Before presenting the results it is vital to show the number of features each method selects. This information is shown in table 5.7 along with the number of features that are the same in these two methods. From the results we can see that for the HIGGS and SUSY datasets the number of features selected by G^2 test is larger than when the combined pvalues method are used. However, it is important to notice that for both datasets all the features selected by combined pvalues method are in the selected features by G^2. For the other two datasets we stop the process when 100 features were selected. Most of the selected features are the same for the two methods. More particularly, for the Epsilon dataset are 67 and for the Zeta 83.

Dataset	#Features	$\#G^2$	#Combined	#Same
HIGGS	28	16	11	11
SUSY	18	11	10	10
Epsilon	2,000	100	100	67
Zeta	2,000	100	100	83

Table 5.7: Total number of features for each dataset and the number of features selected by MMPC when G^2 test and the Fisher's combined probability method are used.

Figure 5.7 shows the logarithmic time that takes for the algorithm to finish when the

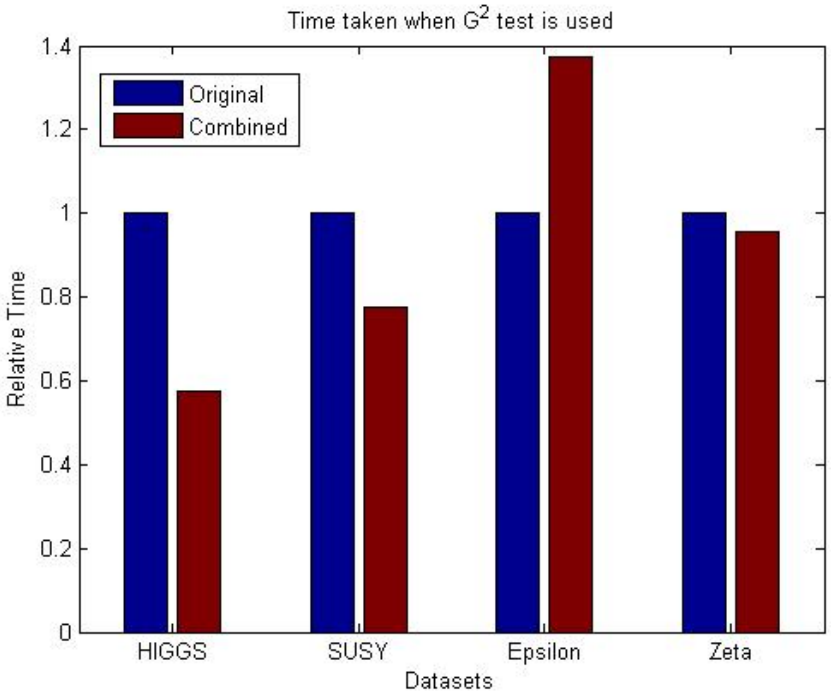

Figure 5.7: Relative time taken when G^2 test and Fisher's combined probability method are used. In most dataset Fisher's combined probability method is faster than the G^2 test.

actual G^2 test is used and for the combined pvalues method. From the results we can see that combining pvalues method takes less time in three out of four datasets. Only in Epsilon dataset the time is more.

In order to evaluate the method we run the logistic regression classifier using the selected features of the G^2 test and the combining pvalues method. The results are shown in figure 5.8. From the results we can conclude that when the features selected by combining pvalues method are used, the classifier performs slightly worse than the features selected by G^2 test. The worse performance is for the HIGGS dataset. However, this was expected because the number of features selected are 6 less than the number of features selected when G^2 test is used. Although the performance of the Fisher's combined probability method is slightly worse, it can be used with any complicated statistical test beyond the simple ones we use in this thesis.

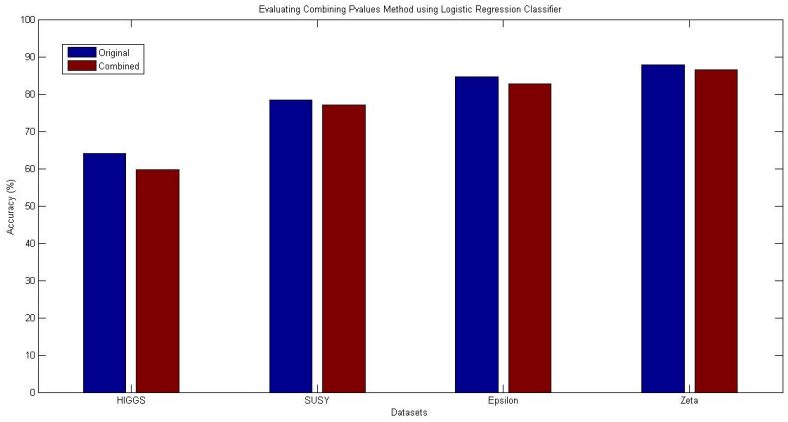

Figure 5.8: Accuracy using logistic regression classifier when running MMPC with G^2 test and Fisher's combined probability method. Accuracy of Fisher's combined probability method reduces slightly than the actual G^2 method.

Chapter 6

Conclusions and Future Work

This thesis was concerned with feature selection algorithms for big data. Its main contribution is the parallelization of the Max-Min Parent and Children algorithm for big data and more particularly with the use of the Map-Reduce model and Hadoop. We chose to use Hadoop because it is the most well-known platform and most frequently used for big data analytics.

We showed how the parallelization can be performed efficiently for the MMPC algorithm using two independent tests: the G^2 and the Fisher Z test. Moreover, the user can extent the functionality of the algorithm to support any other test of interest, the same way we used the aforementioned tests.

The experimental evaluation demonstrated that our algorithm is robust and can scale well when different size of datasets and nodes are used. More particularly, with a fixed size dataset the more nodes in the cluster the less time takes for our algorithm to finish execution. In addition, with a fixed number of nodes in a cluster, the higher the size of the dataset the higher the execution time of the algorithm.

MMPC outperforms SFO in different aspects which are illustrated below:

- MMPC needs to perform only one Map-Reduce job in order to select a feature, in contrast to the SFO algorithm that needs several jobs and so more time is required. This is because SFO runs a logistic regression model in each iteration, which requires multiple passes of the data.

- Our algorithm keeps a set of features that have the maximum minimum association with the target other than the set of selected features that SFO also keeps.

- For continuous/binary data our algorithm can be used with millions of features (using Fisher Z test), in contrast to SFO which can be used with only thousands of features.

- MMPC can handle any type of data if the appropriate statistical independence test is used but SFO can handle data that have continuous or categorical features.

- The user can control the number of features that will be selected from MMPC. This can be accomplished by fixing the parameters threshold and max_k. The higher the threshold the more features are selected and the higher the max_k the smaller the size of selected features.

One major limitation of our algorithm is when multiple statistics are required for a test and the sufficient statistic extracted from a Map-Reduce job needs substantial amount of memory. In this case if the nodes does not have the needed memory the algorithm will not be able to run. This is the case for the G^2 test where if millions of contingency tables are needed and the nodes does not have the required memory to store all tables, the algorithm will crash.

Another limitation is the implementation in R. R is an excellent language to perform statistical analysis, but it is very slow. Moreover, using RHadoop some functionalities of Hadoop are hidden within the package. So, the solution is to implement MMPC using Java in order to accelerate its performance.

For future work, the MMPC algorithm for big data can be extended to output multiple set of features that have same predictive power. In [51], the authors present SES, a statistical algorithm that attempts to identify a set of equivalent, highly predictive features for a given target outcome. The algorithm is very similar to MMPC and it is easy to extend the usability of MMPC to output multiple features.

Moreover, MMPC could be used to find a Bayesian network for big data. It has been used for small datasets in [52]. The algorithm first finds the skeleton of a Bayesian Network with the use of MMPC and then they use a Bayesian-scoring greedy hill-climbing search to find the orientation of the edges. It would be interesting to show how this algorithm performs and scales for big data as well.

Finally, because of the fact that G^2 test is slow and requires significant amount of memory it is imperative the need to try and speed up this test. This can be accomplished with the use of data structures and more particularly with the AD-trees. Building this structure the algorithm would not need to pass the data to find the p-value of the test. We tested these trees using R but we did not had any improvement in time. Implementing this structure in another language like Java for example may have better results.

Bibliography

[1] Dean, J., Ghemawat, S. D. Greene, G. Cagney, N. Krogan, and P. Cunningham *"MapReduce: simplified data processing on large clusters",* Communications of the ACM, 51(1), 107-113, 2008.

[2] Dean, J., Ghemawat, S. Ghemawat, S., Gobioff, H., Leung, S. T. *"The Google file system",* In ACM SIGOPS Operating Systems Review, Vol. 37, No. 5, pp. 29-43, 2003.

[3] http://javax4u.blogspot.gr/2012/11/hadoop.html

[4] Markham, J., Niemiec, J., Vavilapalli, V.K., Murthy,A., and Eadline,D., *"Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2",* Addison-Wesley Professional, 2014

[5] Seebode, C., Ort, M., Regenbrecht, C., and Peuker, M., *"BIG DATA infrastructures for pharmaceutical research"*, IEEE International Conference on Big Data, pp. 59-63, 2013.

[6] Yin, W., Simmhan, Y., and Prasanna, V. K. , *"Scalable regression tree learning on Hadoop using OpenPlanet",* In Proceedings of third international workshop on MapReduce and its Applications Date,pp. 57-64, 2012.

[7] Ye, J., Chow, J. H., Chen, J., and Zheng, Z., *"Stochastic gradient boosted distributed decision trees",* In Proceedings of the 18th ACM conference on Information and knowledge management, pp. 2061-2064, 2009.

[8] Bochicchio, M. A., Longo, A., Vaira, L., Malvasi, A., and Tinelli, A., *"Multidimensional analysis of fetal growth curves",* IEEE International Conference on Big Data, pp. 23-28, 2013.

[9] Caragea, D., Silvescu, A., and Honavar, V., *"A framework for learning from distributed data using sufficient statistics and its application to learning decision trees",* International Journal of Hybrid Intelligent Systems, 1(1), 80-89, 2004.

[10] G. Wu, H. Li, X. Hu, Y. Bi, J. Zhang, and X. Wu, *"Mrec4.5: C4.5 Ensemble Classification with Map-ReduceâĂİ,* Proc. Fourth China-Grid Ann. Conf., pp. 249-255, 2009.

[11] Lin, Y., Lv, F., Zhu, S., Yang, M., Cour, T., Yu, K., Cao, L. and Huang, T., *"Large-scale image classification: fast feature extraction and svm training",* IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1689-1696, 2011.

[12] Yin, X., Wu, B., and Lin, X. *"A unified framework for predicting attributes and links in social networks",* IEEE International Conference on Big Data, pp. 153-160, 2013.

[13] Liu, X., Wang, X., Matwin, S., and Japkowicz, N., *"Meta-learning for large scale machine learning with MapReduce",* IEEE International Conference on Big Data, pp. 105-110, 2013.

[14] Tewari, N. C., Koduvely, H. M., Guha, S., Yadav, A., and David, G., *"MapReduce implementation of Variational Bayesian Probabilistic Matrix Factorization algorithm",* IEEE International Conference on Big Data, pp. 145-152, 2013.

[15] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Suresh, A., Hao, L., and Murthy, R., *"Hive-a petabyte scale data warehouse using hadoop",* In Data Engineering (ICDE), 2010 IEEE 26th International Conference on (pp. 996-1005), 2010.

[16] Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A., *"Pig latin: a not-so-foreign language for data processing",* In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 1099-1110, 2008.

[17] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S. Bae, J. Qiu, and G. Fox, *"Twister: a runtime for iterative mapreduce",* In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pages 810âĂŞ818. ACM, 2010

[18] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I., *"Spark: cluster computing with working sets",* In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, pp. 10-10, 2010.

[19] Engle, C., Lupher, A., Xin, R., Zaharia, M., Franklin, M. J., Shenker, S., and Stoica, I., *"Shark: fast data analysis using coarse-grained distributed memory.",* In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 689-692, 2012.

[20] http://mahout.apache.org/

[21] http://www.cs.waikato.ac.nz/ml/weka/

[22] http://markahall.blogspot.co.nz/2013/10/weka-and-hadoop-part-1.html

[23] Prekopcsak,Z., Makrai, G., Henk, T., and Gaspar-Papanek, C., *"Radoop: Analyzing Big Data with RapidMiner and Hadoop",* In Proceedings of the 2nd RapidMiner Community Meeting and Coference (RCOMM 2001), 2011.

[24] Guyon, I., Elisseeff, A., *"An introduction to variable and feature selection",* The Journal of Machine Learning Research, 3, 1157-1182, 2003.

[25] Tsamardinos, I. and Aliferis, C F. *"Towards principled feature selection: Relevancy, filters and wrappers",* In: Proceedings of the ninth international workshop on Artificial Intelligence and Statistics. Morgan Kaufmann Publishers: Key West, FL, USA, 2003.

[26] Pearl, J., *"Probabilistic Reasoning in Expert Systems"*, Morgan Kaufmann, San Mateo, CA 1988.

[27] Spirtes,P., Glymour, C., and Scheines, R., *"Causation, Prediction, and Search"*, The MIT Press, second edition, 2000.

[28] Tsamardinos, I. *"Machine Learning Course",*(2013-2014).

[29] Fawcett, T., *"ROC graphs: Notes and practical considerations for researchers",* Machine learning, 31, 1-38, 2004.

[30] Amaldi, E., and Kann, V. *"On the approximation of minimizing non zero variables or unsatisfied relations in linear systems",* Theoretical Computer Science, 209, pp. 237-260, 1998

[31] Sun, Zhanquan. *"Parallel feature selection based on MapReduce"*, Computer Engineering and Networking. Springer International Publishing, 2014. 299-306.

[32] Tan, M., Tsang,I. W., and Wang,L., *"Towards ultrahigh dimensional feature selection for big data"*, Journal of Machine Learning Research, vol. 15, pp. 1371âĂŞ1429, 2014.

[33] Zhao, Z., Zhang, R., Cox, J., Duling, D., and Sarle, W. *"Massively parallel feature selection: an approach based on variance preservation"*, Machine learning, 92(1), 195-220, 2013.

[34] Barbu, A., She, Y., Ding, L., and Gramajo, G. *"Feature Selection with Annealing for Big Data Learning"*, arXiv preprint arXiv:1310.2880, 2013.

[35] Yu, K., Wu, X., Ding, W., and Pei, J.*"Towards Scalable and Accurate Online Feature Selection for Big Data."*, Data Mining (ICDM), 2014 IEEE International Conference on. IEEE, 2014.

[36] Bekkerman, R., Bilenko, M., and Langford, J. (Eds.). *"Scaling up machine learning: Parallel and distributed approaches",* Chapter 17(pp. 352-369), Cambridge University Press,2011.

[37] Whitney, A.W., *"A direct method of nonparametric measurement selection",* IEEE Trans. Comput. 20 (9), 1100âĂŞ1103, 1971.

[38] Singh, S., Kubica, J., Larsen, S., and Sorokima, D., *"Parallel Large Scale Feature Selection for Logistic Regression",* In SDM ,pp. 1172-1183, 2009.

[39] Perkins, S., Lacker, K., and Theiler, J., *"Grafting: Fast, incremental feature selection by gradient descent in function space",* The Journal of Machine Learning Research, 3, 1333-1356 2003.

[40] Caruana, R., Karampatziakis, N., and Yessenalina, A, *"An empirical evaluation of supervised learning in high dimensions",* In Proceedings of the 25th international conference on Machine learning (pp. 96-103), ACM, 2008.

[41] Hoi, S. C., Wang, J., Zhao, P., and Jin, R. *"Online feature selection for mining big data",*, In Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, pp. 93-100, ACM, 2012

[42] Tibshirani, R. , *"Regression Shrinkage and Selection via the Lasso",* Journal of the Royal Statistical Society, 58(1), 267âĂŞ288, 1996.

[43] Tsamardinos, I., Aliferis, C. F., and Statnikov, A., *"Time and sample efficient discovery of Markov blankets and direct causal relations",* In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 673-678, 2003.

[44] Tsamardinos, I., Aliferis, C. F., Statnikov, A., and Brown, L. E., *"Scaling-up bayesian network learning to thousands of variables using local learning techniques",* Vanderbilt University DSL TR-03-02,2003.

[45] Guilford J. P., Fruchter B. *"Fundamental statistics in psychology and education,"* Tokyo: McGraw-Hill Kogakusha, LTD, 1973.

[46] Spirtes, P., Glymour, C., and Scheines, R., *"Causation, prediction, and search",* The MIT Press, second edition

[47] https://archive.ics.uci.edu/ml/datasets/HIGGS

[48] https://archive.ics.uci.edu/ml/datasets/SUSY

[49] https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/

[50] http://largescale.ml.tu-berlin.de/instructions/

[51] Tsamardinos, I., Lagani, V., and Pappas, D., *"Discovering multiple, equivalent biomarker signatures",* The Hellenic Society for Computational Biology and Bioinformatics, 2012.

[52] Tsamardinos, I., Brown, L., and Aliferis, C., *"The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm",* Machine learning, 65(1), 31-78, 2006.

Appendices

Appendix A

Installing Hadoop and RHadoop

For this thesis the Hadoop 1.2.1 and the RHadoop were used. This appendix describes how Hadoop was installed and tuned. Moreover, it presents the installation of RHadoop and the packages that were used for the implementation of the MMPC algorithm.

A.1 Installing Hadoop

Hadoop can run in a single node or in a cluster of nodes. First, we describe how it can be install in a single node and then in a cluster of nodes.

A.1.1 Single node installation

The node need to have Java and ssh installed. Firstly, the user need to configure ssh, so as to create a sshless connection. In order to do that the user have to generate a SSH key. This can be done with the command: *ssh-keygen -t rsa -P ""*. This will create a RSA key pair with empty password. After creating the key the user have to enable SSH access to the local machine with this newly created key. This can be done with the command: *cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys*.

After configuring the SSH, it is time to install Hadoop. The user can find the version of his choice in the http://hadoop.apache.org/releases.html. Later the user have to extract the downloaded file. This will create a folfer "hadoop-X.Y". The final step is to configure Hadoop. To do that the user needs to modify some of the files in the folder hadoop/conf. The files are: hadoop-env.sh, mapred-site.xml, hdfs-site.xml and core-site.xml.

The *hadoop-env.sh* file contains some environment variable settings used by Hadoop, such as the *JAVA_HOME* which specifies the path to java installation. The file is shown below.

```
<!-- File hadoop-env.sh -->
export JAVA_HOME=/usr/lib/jvm/java-6-sun
```

The *mapred-site.xml* file contains configuration options for MapReduce, such as the host and port the jobtracker runs. The files is presented below.

```
<!-- File mapred-site.xml -->

<property>
```

```
<name>mapred.job.tracker</name>
<value>master:54311</value>
<description>The host and port that the MapReduce job tracker runs
    at.  If "local", then jobs are run in-process as a single map
    and reduce task.
</description>
</property>

<property>
<name>mapred.tasktracker.reduce.tasks.maximum</name>
<value>1</value>
</property>

<property>
<name>mapred.tasktracker.map.tasks.maximum</name>
<value>6</value>
</property>

<property>
<name>mapred.compress.map.output</name>
<value>true</value>
</property>

<property>
<name>mapred.map.output.compression.codec</name>
<value>org.apache.hadoop.io.compress.BZip2Codec</value>
</property>
```

The *hdfs-site.xml* file contains configuration options for the HDFS, such as the block size or the number of replication. The file is shown below.

```
<!-- File hdfs-site.xml -->

<property>
<name>dfs.replication</name>
<value>2</value>
<description>Default block replication.
The actual number of replications can be specified when the file
   is created.
The default is used if replication is not specified in create time
   .
</description>
</property>
```

Finally, the *core-site.xml* file contains configuration that override the default values for core Hadoop properties. The file is presented below.

```
<!-- File core-site.xml -->
<property>
<name>hadoop.tmp.dir</name>
<value>/app/hadoop/tmp</value>
<description>A base for other temporary directories.</description>
</property>

<property>
<name>fs.default.name</name>
```

```
<value>hdfs://master:54310</value>
<description>The name of the default file system. A URI whose
scheme and authority determine the FileSystem implementation. The
uri's scheme determines the config property (fs.SCHEME.impl)
    naming
the FileSystem implementation class. The uri's authority is used
    to
determine the host, port, etc. for a filesystem.</description>
</property>
```

After configuring Hadoop, the user needs to format Namenode. This is done only the first time and it can be done with the command: *hadoop namenode -format.*

Finally, hadoop can now start on a signle node with the command: *bin/start-all.sh.*

A.1.2 Multi-node installation

In order to make a multi-node installation the master node needs to define the IP of the nodes in the cluster, including itself. This can be done by modifying the file */etc/hosts.* An example of the file is shown below.

```
192.168.0.1     master
192.168.0.2     slave1
192.168.0.3     slave2
```

Moreover, the master node needs to have the ability to log without a password to the slave nodes. This can be done by adding the master's key to the authorized keys of the slave nodes. The command to accomplish this is the following: *ssh-copy-id -i $HOME/.ssh/id_ rsa.pub user@slave1.*

The final step is to configure the Hadoop of the master so as to know which is the master and which are the slave nodes. To do that the user need to define the *conf/masters* to be:

```
master
```

and to configure the slaves the user needs to update the *conf/slaves* to be:

```
master
slave1
slave2
```

Finally, the user needs to format the namenode and start the hadoop cluster. The commands are the same as previously for the single-node installation.

A.2 Installing RHadoop

As mentioned in the Introduction, RHadoop is consisting of 5 packages. For this thesis we used the two main packages: rmr2 and rhdfs. The packages can be downloaded from the link: *https://github.com /RevolutionAnalytics/RHadoop/wiki/Downloads.*

All the following installations need to be performed in each node in the cluster. To install the two packages, the R language needs to be installed. Moreover, these packages

are dependent on other packages, which should be installed. The user can install these packages with the command:

```
install.packages(c("rJava", "Rcpp", "RJSONIO", "bitops", "digest",
    "functional","stringr", "plyr","reshape2","dplyr","R.methodsS3
    ","caTools", "Hmisc"))
```

After installing these packages the user can install rmr2 and rhdfs with the commands:

```
install.packages("rhdfs_1.0.8.tar.gz", repos=NULL,type="source")
install.packages("rmr2_2.2.2.tar.gz", repos=NULL, type="source")
```

Before running a Map-Reduce job the user needs to define the following parameters to the *.R file:

```
Sys.setenv("HADOOP_PREFIX"="/User/hadoop-1.1.2")
Sys.setenv("HADOOP_CMD"="/User/hadoop-1.1.2/bin/hadoop")
Sys.setenv("HADOOP_STREAMING"="/User/hadoop-1.1.2/contrib/
    streaming/hadoop-streaming-1.1.2.jar")
```

A.2.1 Word count example

Below it is shown the word count example which was presented in the introduction using RHadoop[1].

```
Sys.setenv("HADOOP_PREFIX"="/User/hadoop-1.1.2")
Sys.setenv("HADOOP_CMD"="/User/hadoop-1.1.2/bin/hadoop")
Sys.setenv("HADOOP_STREAMING"="/User/hadoop-1.1.2/contrib/
    streaming/hadoop-streaming-1.1.2.jar")

library(rmr2)

## map function
map <- function(k,lines) {
wordsList <- strsplit(lines, '\\s')
words <- unlist(wordsList)
keyval(words, 1)
}

## reduce function
reduce <- function(word, counts) {
keyval(word, sum(counts))
}

wordcount <- function (input, output=NULL) {
mapreduce(input=input, output=output, input.format="text",
map=map, reduce=reduce)
}

## Submit job
hdfsRoot <- 'wordcount'
hdfsData <- file.path(hdfsRoot, 'data')
```

[1]Source: "Building an R Hadoop System"

```
hdfsOut <- file.path(hdfsRoot, 'out')
out <- wordcount(hdfsData, hdfsOut)

## Fetch results from HDFS
results <- from.dfs(out)
```

A.3 Configuring Apache Hadoop

There are more than 160 parameters in Hadoop which can affect the performance of cluster. We run some test experiments for two important parameters of Hadoop. These are the number of cores to use and the size of the block size. The Map-Reduce job finds the mean of features in the dataset, which is used for the Fisher Z test. The dataset we used was the SUSY.

Figure A.1 shows the difference in seconds of the mapreduce job when changing the number of map tasks to run in parallel (2,4,6,8,10,12,16) and keeping the block size constant to 64MB. With this block size the number of maps that are needed to run to complete the job are 36.

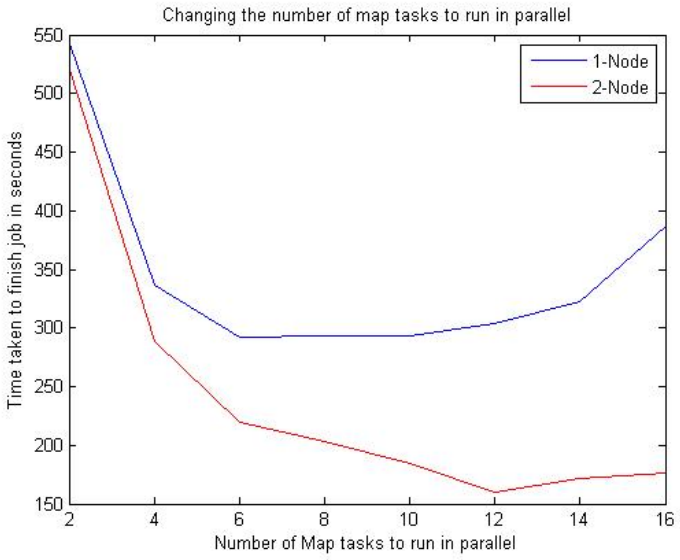

Figure A.1: Changing the number of map tasks to run in parallel for 1- and 2-Node. When the number of map tasks that run in parallel is equal to the number of cores in the nodes we have maximum performance.

From the figure A.1 we can see that using 1-Node the time reduces until 6 map tasks run in parallel and with 2-Nodes the time reduces until 12 map tasks run in parallel. This was expected as 1-Node has 6 cores and 2-Node has 12 cores available for the task.

Figure A.2 shows the difference in seconds of the mapreduce job when changing the block size to [64 128 256 512] and keeping the number of map tasks to run in parallel equal to the number of cores available.

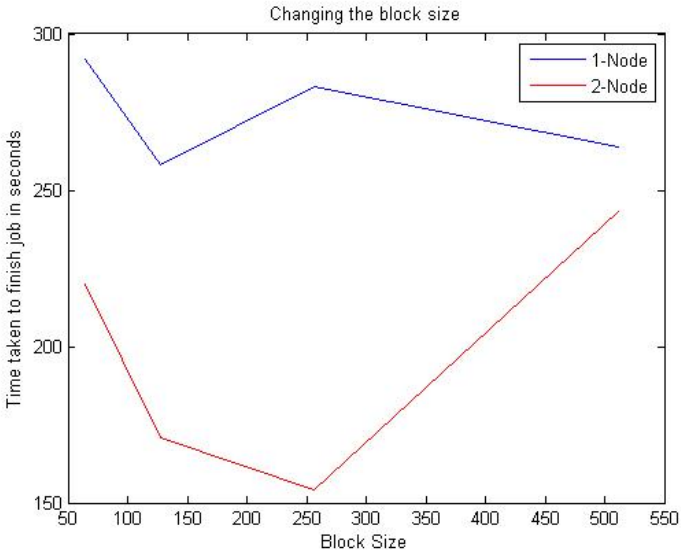

Figure A.2: Relative time when changing the block size and using 1 and 2 nodes. With 2 nodes and block size 256Mb the time is decreased the most.

From the figure A.2 we can conclude that using 1-Node the time reduces until the block size is 128MB and with 2-Nodes until the block size is 256MB. The result for the 1-Node show that 128MB is the best choice. However, with 2-Nodes the time is reduced with block size greater than 256MB. This is due to the fact that with block size 512MB only 10 maps are needed to complete the job and 12 map tasks can run in parallel. That is, the resources of the 2-Node cluster are not fully exploited.

Performing these tests and changing just two configurations of Hadoop, we saw that the performance was significantly changed. For this thesis we consider the following configurations:

- mapred.tasktracker.map.tasks.maximum : The default number of map tasks per job is two. For our cluster we changed it to six.
- mapred.child.java.opts: Java opts for the task tracker child processes. For our case we initialized it to 1 Gb.
- dfs.replication: The block replication in the cluster. We set it to two.
- dfs.block.size: The default block size is 64MB. We did not change this value as some datasets that we use are small.

All other configuration had the default values.

Appendix B

Using Storey's method

The experiments in chapter 5 were performed with threshold 0.01. However, it would be interesting to use the Storey's method to adjust the threshold and see the performance of the selected variables. Table B.1 shows the number of selected features along with the new threshold that was used. The FDR threshold is higher than 0.01 in all of the datasets but SUSY which is exactly 0.

Datasets	#Features	Adjusted Threshold	#MMPC
HIGGS	28	$3.9 \cdot 10^{-2}$	14(0)
SUSY	18	0	5(0)
RCV1	47,236	$3.6 \cdot 10^{-2}$	100(7512)
Epsilon	2,000	$1.1 \cdot 10^{-2}$	83(0)
Zeta	2,000	$1.1 \cdot 10^{-2}$	100(88)
Mnist8	784	1	100(684)
URL	3,231,961	$1.7 \cdot 10^{-3}$	100(2242)

Table B.1: Total number of features for each dataset, the adjusted threshold when Storey's method is used and the number of features selected by MMPC. In parenthesis for MMPC is the number of remaining features.

Figure B.1 shows the performance of the logistic regression for the selected features by adjusting the threshold with Storey's method.

From the results we can see that the performance of the classifier is almost the same as when the threshold was set to 0.01 for all datasets but SUSY which dips significantly. This was expected as fewer features are selected than the previous experiment.

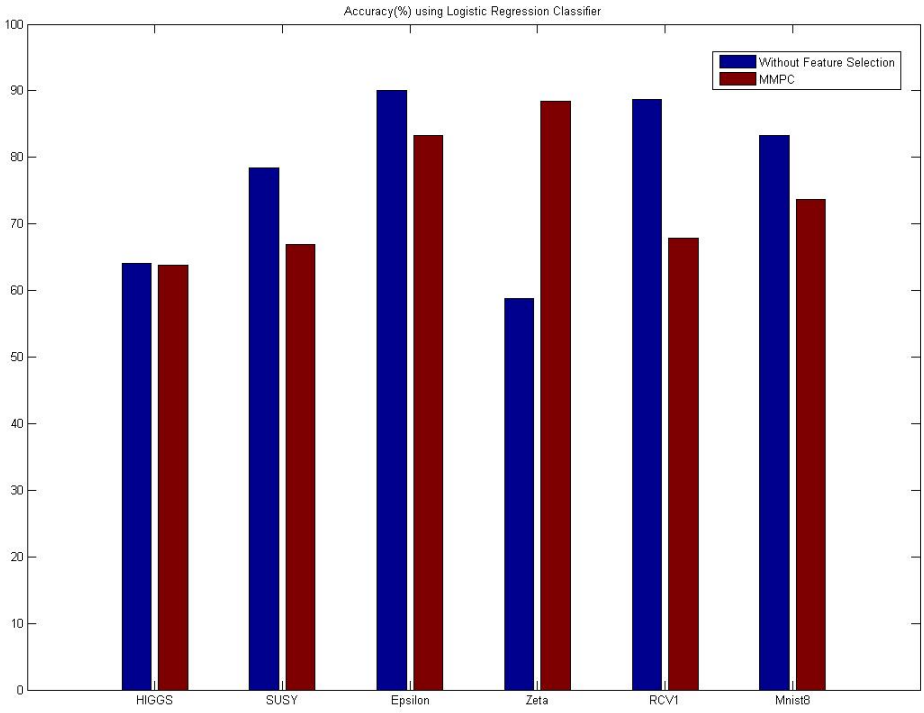

Figure B.1: Evaluating feature selection using Storey's method, using logistic regression classifier's accuracy on the features selected by MMPC and when no feature selection is performed.