APPROVAL SHEET

Title of Thesis: Parallel Feature Selection of Multiple Class Datasets using Apache Spark

Name of Candidate: Rishi Sankineni
Master of Science, Information Systems, 2017

Thesis and Abstract Approved:

Dr. Jianwu Wang
Assistant Professor
Information Systems

Date Approved: 07/31/17

ABSTRACT

Title of Document: PARALLEL FEATURE SELECTION OF MULTIPLE CLASS DATASETS USING APACHE SPARK

Rishi Sankineni, Master of Science
Information Systems, 2017

Directed by: Assistant Professor, Dr. Jianwu Wang
Department of Information Systems

Feature selection is the task of selecting a small subset from original features that can achieve maximum classification accuracy. This subset of features has some very important benefits like, it reduces computational complexity of learning algorithms, saves time, improve accuracy and the selected features can be insightful for the people involved in problem domain. This makes feature selection as an indispensable task in classification task. In this thesis, we present a two-phase approach for feature selection. In the first phase a batch based Minimum Redundancy and Maximum Relevance (mRMR) algorithm is used with "correlation coefficient" and "mutual information" as statistical measure of similarity. This phase helps in improving the classification performance by removing redundant and unimportant features. In the second phase, we present a stream based tree-based feature selection method that allows dynamic generation and selection of features, while taking advantage of the different feature classes and the fact that they are of different sizes and have different fraction of good features. Experimental results show that this phase is computationally less expensive than comparable "batch" methods that do not take advantage of the feature classes and expect all features to be known in advance.

PARALLEL FEATURE SELECTION OF MULTIPLE CLASS DATASETS USING APACHE SPARK

by

Rishi Sankineni

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Master of Science in
Information Systems
2017

Advisory Committee:
Dr. Jianwu Wang, Chair/Advisor
Dr. George Karabatis, Co-Advisor
Dr. Nirmalya Roy

Acknowledgement

I deeply thank Prof. Jianwu Wang for introducing me to the wonderful world of Machine Learning and for supervising this research. His consistent support, advice, thoroughness and patience have made this work possible. I would also like to sincerely thank Prof. George Karabatis for guiding me through the exciting world of Feature Selection and co-advising to this research. His help had made a crucial contribution to this work.

I would also like to thank the committee member Dr. Nirmalya Roy for being on the examination committee and for providing invaluable feedback. I am grateful to Muthu Kumar and Sai Chaithanya for their immense contribution in developing framework for attack detection and similarity calculation respectively.

I would like to take this opportunity to thank my friends in the Data Informatics lab @ UMBC and Abu Zaher Md Faridee particularly for their tips on research ideas and suggestion.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning deals with the theoretic, algorithmic and applicative aspects of learning from examples. In a nutshell, learning from examples means that we try to build a machine (i.e. a computer program) that can learn to perform a task by observing examples. Typically, the program uses the training examples to build a model of the world that enables reliable predictions[23]. This contrasts with a program that can make predictions using a set of predefined rules (the classical Artificial Intelligence (AI) approach). Most of the problems in machine learning are prediction problems, i.e. problems in which an output Y has to be predicted given an input vector X. If the output Y is a real valued variable the prediction problem is called regression while, if the output Y is a set of classes the prediction problem is called classification. Given a feature vector X, the task is to predict some output vector Y, or its conditional probability distribution P(Y/X). Unfortunately in most of real-world problems, the input feature vector X is constituted by a very large number of features. So the high dimensionality of a problem can cause increased computational complexity, and hinder the performance of the learning algorithm [23]. Therefore, reducing the dimensionality of a problem has certain advantages. Consequently, in any classification task we need to select the relevant subset of features that possibly contains the least number of dimensions which are best for solving the task at hand. Such relevant subset of features is unknown a-priori, instead we need to use some dimensionality reduction method to discover such subset. Feature selection is one of

the dimensionality reduction methods. It is commonly used in applications where original features need to be preserved. Although feature selection can be applied to both supervised and unsupervised learning in this thesis we will focus only on the problem of feature selection in supervised learning (in particular for classification problems).

1.1 Significance of the Problem

The high dimension of today's real-world data poses a serious problem for standard classifiers. Therefore feature selection is a common pre-processing step in many data analysis algorithms. It prepares data for mining and machine learning, which aim to transform data into business intelligence or knowledge. Performing feature selection may have various motivations. For instance, consider the following simple but revealing example of a binary classification problem. Broadly speaking, two factors matter most for effective learning:

- R be the number of features, and
- Z be the number of instances.

When R is fixed, a larger Z means more constraints and the resulting correct hypothesis is expected to be more reliable. When Z is fixed, a reduced R is same as to significantly increase the number of instances. Theoretically, the reduction of dimensionality can exponentially shrink the hypothesis space. Suppose we have four binary (i.e. 0 1) features N1 , N2 , N3 , N4 and class C (e.g.; positive or negative). If the training data is comprised of 4 instances i.e., Z = 4, it is only a quarter of the total number of possible instances $2^4 = 16$. The size of the hypothesis space is $2^{2^4} = 65,536$.

If only two features are relevant to the target concept, the size of the hypothesis space becomes $2^{2^2} = 16$, which is an exponential reduction of the hypothesis space. Now when we are left with only 2 features, the only 4 available instances might be sufficient for perfect learning (assuming there is no repeated instance in the reduced

training data). The resulting model with 2 features can also be less complicated than that with 4 features. Hence, feature selection can effectively reduce the hypothesis space, or virtually increase the number of training instances, and help create a compact model.

1.2 Summary of the Approach

In our approach, we have stressed that a well-designed filter method, such as mRMR [7], can be used to enhance the wrapper feature selection, in achieving both high accuracy and fast speed. It uses an optimal first-order incremental selection to generate a candidate list of features that cover a wider spectrum of characteristic features. These candidate features have similar generalization strength on different classifiers. They facilitate effective computation of wrappers to find compact feature subsets with superior classification accuracy. mRMR algorithm [7] is especially useful for large-scale feature/variable selection problems where there are at least thousands of features/variables, such as gene selection. Of note, the purpose of the mRMR approach studied in this thesis is to maximize the dependency. This typically involves the computation of multivariate joint probability, which is nonetheless difficult and inaccurate. Combining both Max Relevance and Min-Redundancy criteria [7], the mRMR incremental selection scheme provides a better way to maximize the dependency. In this case, the difficult problem of multivariate joint probability estimation is reduced to estimation of multiple bivariate probabilities (densities), which is much easier. In most situations, mRMR reduces the feature selection time dramatically for continuous features and improves the classification accuracy significantly.

Much of "big data" is received in real time, and is most valuable at its time of arrival. For example, a social network may wish to detect trending conversation topics in minutes; a search site may wish to model which users visit a new page; and a service operator may wish to monitor program logs to detect failures in seconds [35]. To enable these low-latency processing applications, there is a need for streaming computation models that scale transparently to large clusters. Spark Streaming is

one such computation model which can be used for real-time feature selection [35].

Streaming feature selection on Apache Spark supports flexible ordering on the generation and testing of features. Features can be generated dynamically based on which features have already been added to the model. One can also test the same feature more than once, as we do in this section(by using multiple batches). New features can be generated in many ways. Each way produces a new feature class for use in streaming. For example, in addition to the n original features, n^2 pairwise interaction terms can be formed by multiplying all n^2 pairs of features together. In practice, we generate three interaction streams: (1) interactions of features that have already been selected with themselves (2) interactions of the selected features with the original features, and (3) all interactions of the original features. This requires dynamic generation of the feature stream, since the interaction terms (1) and (2) can not be specified in advance, as they depend on which features have already been selected.

1.3 Contribution of the Thesis

This thesis contains a number of unique contributions. They are summarized below:

- Applied mRMR feature selection algorithm to KDD'99 dataset.
- Implemented a parallel feature selection algorithm on Apache Spark platform for streaming data.
- Calculated feature selection algorithms efficiency.
- Calculated batch correlation score on streaming data.
- Calculated execution time taken by the cluster to rank the features.

Chapter 2

Background and Related Work

This introductory chapter provides the context and background for the results discussed in the following chapters and defines some crucial notation. The chapter begins with a brief review of machine learning, which is the general context for the work described in this thesis. I explain the goals of machine learning, present the main learning models currently used in the field, and discuss its relationships to other related scientific fields. Then, in section 2.3, I review the field of feature selection, which is the sub field of machine learning that constitutes the core of this thesis. I outline the rationale for feature selection, the different paradigms that are used and survey some of the most important known algorithms.

2.1 Machine Learning

Machine learning deals with the theoretic, algorithmic and applicative aspects of learning from examples[3]. In a nutshell, learning from examples means that we try to build a machine (i.e. a computer program) that can learn to perform a task by observing examples. Typically, the program uses the training examples to build a model of the world that enables reliable predictions. This contrasts with a program that can make predictions using a set of predefined rules (the classical Artificial Intelligence (AI) approach) [23].

"You may not know it, but machine learning is all around you. When you type a query into a search engine, it's how the engine figures out which results to show you (and which ads, as well). When you read your e-mail, you don't see most of the spam, because machine learning altered it out. Go to Amazon.com to buy a book or Netfix to watch a video, and a machine-learning system helpfully recommends some you might like. Facebook uses machine learning to decide which updates to show you, and Twitter does the same for tweets. Whenever you use a computer, chances are machine learning is involved somewhere. Traditionally, the only way to get a computer to do something from adding two numbers to flying an airplane was to write down an algorithm explaining how, in painstaking detail. But machine-learning algorithms, also known as learners, are different: they figure it out on their own, by making inferences from data. And the more data they have, the better they get. Now we don't have to program computers; they program themselves. It's not just in cyberspace, either: your whole day, from the moment you wake up to the moment you fall asleep, is suffused with machine learning"[8].

2.2 Learning Algorithms

Feature selection algorithms can be further categorized into supervised, unsupervised, and semi-supervised, corresponding to different types of learning algorithms. The primary difference between these types of learning is whether the training data has been hand-labeled or not to generate the classifier's output.

2.2.1 Supervised Learning

In supervised learning, we have a training set in which each training example is a (instance, label) pair, and the learner's goal is to learn a mathematical model that represents the mapping function between the instance vectors and the label values. The model is then used to predict the label of a new unseen instance with only a small chance of erring. In essence, supervised feature selection algorithms try to

find features that help separate data of different classes. In case of regression, the feature selection is done by selecting the variables that most reduce the residual sum of squares as in forward stepwise selection or minimizes a penalized criterion[10].

2.2.2 Unsupervised Learning

Unlike supervised learning, unsupervised learning has no labeled data. Learning algorithms base only on input values from the data. It aims to find subset of features according to similar patterns in the data without prior information. This type of learning is used for example in clustering, self-organization, auto- association and some visualization algorithms. Typical approaches to unsupervised learning include clustering, building probabilistic generative models and finding meaningful transformations of the data. Given a fixed number of clusters, we aim to find a grouping of the objects such that similar objects belong to the same cluster. K-means is a classical clustering algorithm which clusters instances according to the Euclidean distance. Unsupervised methods perform poorly in the beginning as compared to supervised learning as they are un-tuned, but performance increases as they tune themselves over some time[12].

2.2.3 Semi-Supervised Learning

The acquisition of labeled instances for a learning problem is often difficult, ex- pensive, or time consuming to obtain, as this requires the efforts of experienced human annotators. Therefore, it may not be feasible to get labeled instances sometimes. On the other hand, acquisition of unlabeled instances is relatively inexpensive to collect. In a scenario when we have a small amount of labeled data with a large amount of unlabeled data for the training, semi-supervised learning technique can be of great practical value to train our classifier. It is observed that when large unlabeled instances used in conjunction with a small number of labeled instances, give higher accuracy. Because semi-supervised learning requires less human effort and gives considerable higher accuracy, it is of great interest both in theory and in

practice[4].

2.3 Feature Selection

A critical issue in data pre-processing is feature selection: instead of using all the features (attributes or variables) in the data, one can selectively choose a subset of features. There are several benefits of feature selection: (1) dimensionality reduction to bring down the computational cost; (2) noise reduction to revamp the classification accuracy; (3) more interpretable features or characteristics that can help identify and monitor the target variables or class types[11]. These advantages are typified on KDDCUP'99 dataset. Out of 42 features, only a smaller number (7) of them show strong correlation with the targeted network attacks. For example, for a two-class network attack subtype classification problem, 42 informative features are usually sufficient. There are studies suggesting that only a few features are sufficient. Therefore, computation is reduced while prediction accuracy is increased via effective feature selection.

There are two essential ways to perform dimension reduction for classification problems. The first way is to basically recognize (by some criterion) those features that contribute most to the class separability. For example, one may select n features out of all the given features, using some method of ranking (the uni-variate approach) or optimizing a criterion function (the multivariate approach), that will most contribute to the classification task. This strategy is termed feature selection[11]. The other way is to find a transformation (linear or nonlinear) from the original high-dimensional input space to a lower dimensional feature space. This approach is termed feature extraction. This transformation may again be supervised or unsupervised. In the supervised case, the task is to find the transformation for which a criterion of class separability is maximized.

For many prediction or regression tasks only a subset of a huge number of candidate features are predictive, and good feature selection methods can give large improvements in predictive accuracy [11]. Feature selection approaches generally

assume that all features are in a single equivalence class. This ignores the key, and useful, fact that very often features are of different type.

Often, samples have hundreds to tens of thousands of variable or features (i.e., they are represented as vectors in a high-dimensional space). The primary task of feature extraction and feature selection is to reduce the dimension of the data as much as possible while still retaining most of the information relevant for the task at hand. There are many reasons to perform such dimension reduction. It may remove redundant or irrelevant information and thus yield a better classification performance; subsequent analysis of the classification results is relatively easier to understand; low dimension results may be visualized, and thus enable better understanding[11].

The process of feature selection can be supervised, unsupervised or semi- supervised based on class labels. In supervised feature selection, the evaluations of features are determined using their correlation with the class while unsupervised algorithm uses data variance or data distribution in its evaluation. In semi- supervised we use limited label information to improve unsupervised feature selection. Depending on how and when the worth of each feature in the subset is evaluated, three models can be proposed. They are filters, wrappers and hybrids. Filters evaluate the worth of a feature without any learning algorithm. Wrappers have a predetermined learning algorithm to evaluate the worthiness of an attribute in the subset. Hybrids are a combination of filters and wrappers. .

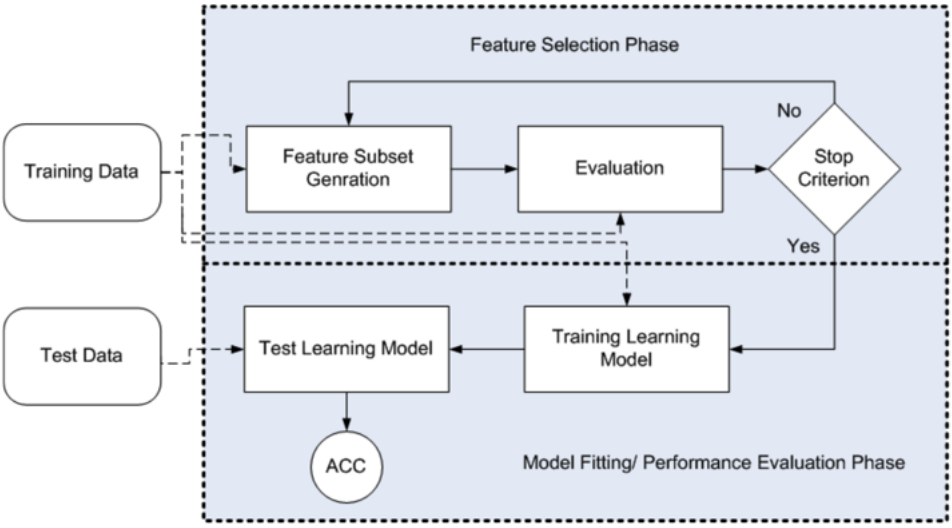


Figure 2.1: A unified view of a feature selection process

2.3.1 Feature Ranking

As the filter approach is the more common one, our study will focus on several filter methods. In this section, we will introduce two common filter based feature ranking techniques.

- **Information Gain**: Information gain (IG)[16] is based on the concept of entropy. The expected value of information gain is the mutual information of target variable (X) and independent variable (A). It is the reduction in entropy of target variable (X) achieved by learning the state of independent variable (A). The major drawback of using information gain is that it tends to choose attributes with large numbers of distinct values over attributes with fewer values even though the latter is more informative.

  To calculate information gain, consider an attribute X and a class attribute Y. The information gain of a given attribute X with respect to class attribute Y is the reduction in uncertainty about the value of Y when the value of X is known. The value of Y is measured by its entropy, H(Y). The uncertainty about Y, given the value of X is given by the conditional probability of Y given X, H (Y—X)[23].

- **Gain Ratio**: The information gain measure is biased towards tests with many outcomes. That is, it prefers to select attributes having many possible values over attributes with fewer values even though the latter is more informative. For example, consider an attribute that acts as a unique identifier, such as a student id in a student database. A split on student id would result in many partitions; as each record in the database has a unique value for student id. So, the information required to classify database with this partitioning would be 0. Clearly, such a partition is useless for classification[16].

$$GainRatio(N) = Gain(N)/SplitInfo(N)$$

2.4 Feature Subset Selection

In contrast to feature ranking, feature subset selection algorithms[22] may automatically find how many features have to select. The rapid advances in several research fields with huge datasets made it essential to select only the most important or descriptive features and the remaining are discarded.

Feature subset selection[22] can be divided into three models: filters, wrappers and embedded. All feature selection models have their own advantages and drawbacks. In general, filters are fast due to the fact they do not incorporate learning and rely on the intrinsic characteristics of the training data to select and discard features (mutual information, data consistency, etc). A wrapper model[19] involves a learning algorithm (a classifier, or a clustering algorithm) to evaluate each subset of features quality. By including the learning algorithm they aim at improving accuracy. However, wrapper models are computationally intensive, which restricts their application to huge datasets. An embedded model embeds feature selection in the training process of the classifier and are usually specific to given learning machines. They are usually faster than wrapper approaches but are also more likely to overfit. In case we have large training set then embedded models can eventually

replace filter models.

There are many strategies for feature selection. For example, one can define an objective function, e.g., one that measures accuracy on a fixed held out set, and use sequential forward or backward selection. A sequential forward selection (SFS)[11] is a bottom-up search where new features are added to a feature set one at a time. At each stage, the chosen feature is one that, when added to the current set, maximizes the objective. The feature set is initially empty. The algorithm terminates when the best remaining feature worsens the objective, or when the desired number of features is reached. The main disadvantage of this method is that it does not delete features from the feature set once they have been chosen. As new features are found in a sequential, greedy way, there is no guarantee that they should belong in the final set.

Feature subset selection approach is believed to have better predictive ability than that of feature ranking according to their individual predictive power[22]. As already mentioned, a single feature that is completely useless by itself can strikingly improve performance when taken in account with other features. On the other hand, a good feature which is highly correlated with another feature already in the subset would provide no additional benefit since it would be redundant. Feature ranking approaches can not manage to deal with these scenarios.

Sequential backward selection (SBS)[11] is the top-down analog of SFS: Features are deleted one at a time until d features remain. This procedure has the disadvantage over SFS that it is computationally more demanding, since the objective function is evaluated over larger sets of variables.

Feature selection can be classified into feature subset selection and feature ranking. Feature ranking calculates the score of each attribute and then sorts them according to their scores. Feature subset selection selects a subset of attributes which collectively increases the performance of the model.

Feature selection techniques can be organized into three categories, depending

on the way they combine the feature selection search with the construction of the classification model:

2.4.1 Filter Methods

Filter type methods are essentially data pre-processing or data filtering methods. Features are selected based on the intrinsic characteristics which determine their relevance or discriminant powers about the target classes. Simple methods based on mutual information, statistical tests (t-test, F-test) have been shown to be effective. More sophisticated methods are also developed. Filter methods can be computed easily and very efficiently. The characteristics in the feature selection are uncorrelated to that of the learning methods, therefore they have better generalization property. Filter methods choose the n best individual features, by first ranking the features by some 'informativeness' criterion, for example, using their Pearson Correlation with the target. Then, the top n features are selected. Afterwards, this subset of features is presented as input to the classification algorithm.

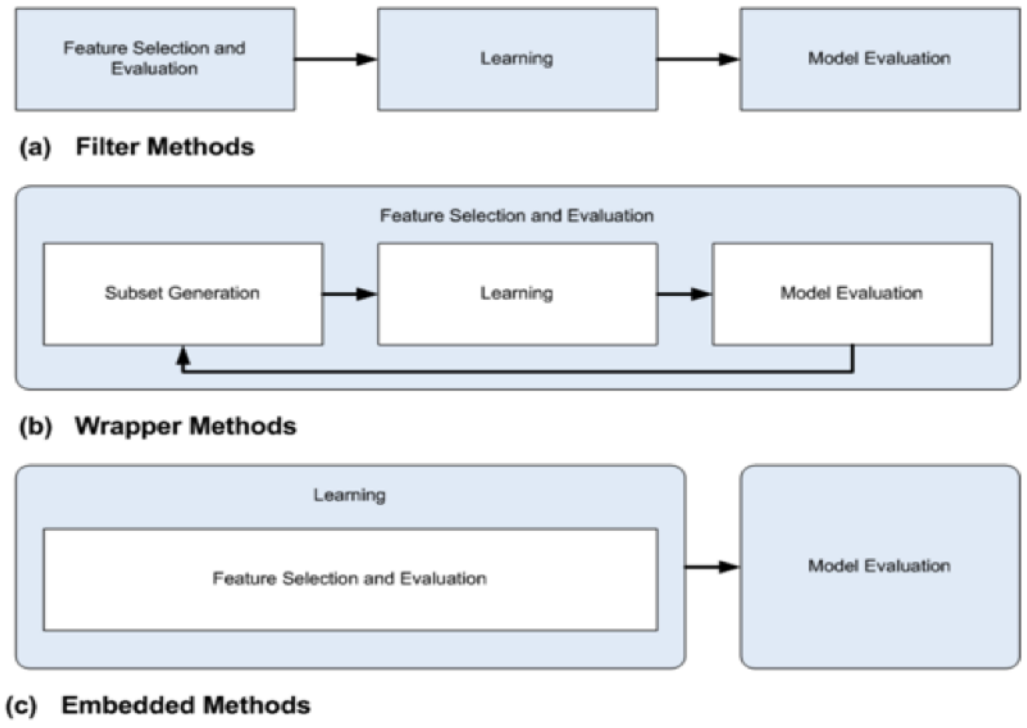


Figure 2.2: Filter, wrapper and embedded feature selection scheme

2.4.2 Wrapper Methods

In wrapper type methods[19], feature selection is "wrapped" around a learning method: the usefulness of a feature is directly judged by the estimated accuracy of the learning method. One can often obtain a set with a small number of non-redundant features, which gives high prediction accuracy, because the characteristics of the features match well with the characteristics of the learning method. Wrapper methods typically require extensive computation to search the best features. Wrapper methods[19] use a search procedure in the space of possible feature subsets using some search strategy such as SFS or SBS, and various subsets of features are generated and evaluated. The evaluation of a specific subset of features is obtained by training and testing a specific classification model. In other words, the search for the desired feature subset is "wrapped" around a specific classifier and training algorithm.

On numerous occasions, wrapper methods[19] are often lambasted because they seem to be a "brute force" method needing enormous amounts of computation, but it is not necessarily so. Cost-effective search strategies may be devised. Using such search strategies does not necessarily mean sacrificing prediction performance. In fact, it appears to be the converse in some cases: coarse search strategies may mitigate the problem of overfitting, as illustrated for instance in this issue by the work of Reunanen (2003). Greedy search strategies seem to be particularly computationally advantageous and robust against overfitting. They come in two flavors: forward selection and backward elimination. In forward selection[11], variables are progressively incorporated into larger and larger subsets, whereas in backward elimination one starts with the set of all variables and progressively eliminates the least promising ones. Both methods yield nested subsets of variables.

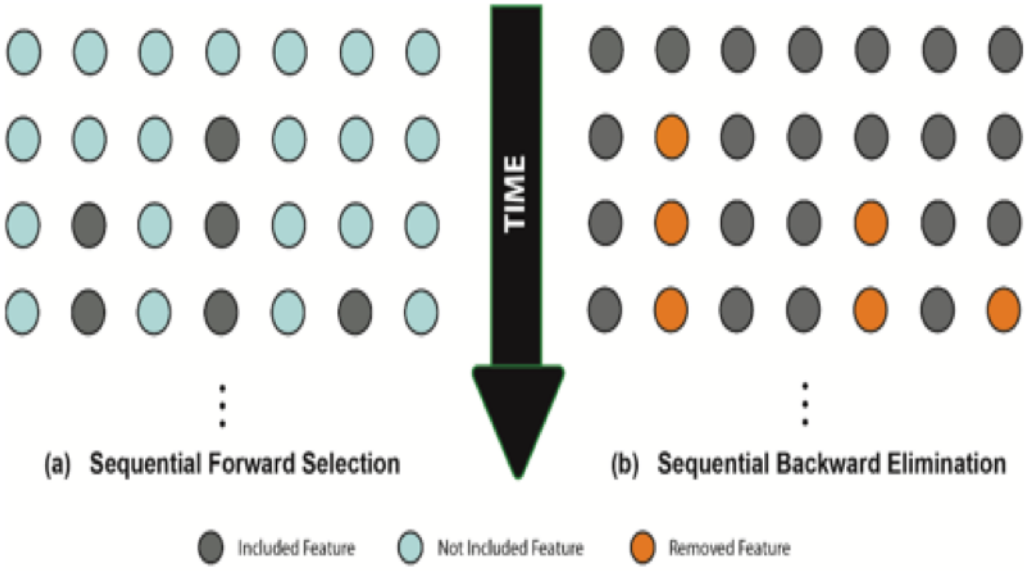


Figure 2.3: Sequential Forward Selection and Sequential Backward Elimination

2.4.3 Embedded Methods

In embedded methods the search for an optimal subset of features is built into the classifier construction. Features are selected as a part of the building of the particular classifier, in contrast to the wrapper approach, where a classification model is used to evaluate a feature subset that is selected without using the classifier. The embedded and wrapper approaches are specific to a given classifier[23].

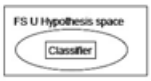
	Model search	Advantages		Disadvantages	Examples
Filter		Univariate	Fast Scalable Independent of the classifier	Ignores feature dependencies Ignores interaction with the classifier	Chi-square Euclidean distance t-test Information gain, Gain ratio
		Multivariate	Models feature dependencies Independent of the classifier Better computational complexity than wrapper methods	Slower than univariate techniques Less scalable than univariate techniques Ignores interaction with the classifier	Correlation based feature selection (CFS) Markov blanket filter (MBF) Fast correlation based feature selection (FCBF)
Wrapper		Deterministic	Simple Interacts with the classifier Models feature dependencies Less computationally intensive than randomized methods	Risk of over fitting More prone than randomized algorithms to getting stuck in a local optimum (greedy search) Classifier dependent selection	Sequential forward selection (SFS) Sequential backward elimination (SBE) Plus q take-away r Beam search
		Randomized	Less prone to local optima Interacts with the classifier Models feature dependencies	Computationally intensive Classifier dependent selection Higher risk of overfitting than deterministic algorithms	Simulated annealing Randomized hill climbing Genetic algorithms Estimation of distribution algorithms
Embedded		Interacts with the classifier Better computational complexity than wrapper methods Models feature dependencies		Classifier dependent selection	Decision trees Weighted naive Bayes Feature selection using the weight vector of SVM

Figure 2.4: A taxonomic summary of feature selection techniques with important characteristics of each technique

2.5 Dimensionality Reduction

Every data entity in a computer is represented and stored as a set of features, for instance, age, height, weight, and so on. Features can interchangeably be termed as dimensions, because an entity with N features can also be represented as a multidimensional point in an N-dimensional space. The process of reducing the initial feature set composed by N features to a feature set composed by K features with $K < N$ is called dimensionality reduction. Ideally, the K reduced features would retain the important characteristics of the original N features[32].

Dimensionality reduction[32] is substantial in many domains like database and machine learning systems and consequently it offers invaluable results like data compression, better data visualization, improved classification accuracy, fast and efficient data retrieval, boosting index performance . There exist two important categories of dimensionality reduction techniques, named feature extraction and feature selection.

Feature extraction also known as feature transformation is the process that finds a new K dimensions that are a combination of the N original dimensions. The best known feature extraction techniques are based on projection and compression methods. Principal component analysis (PCA)[14] and linear discriminant analysis (LDA)[13] are examples of projection methods for unsupervised and supervised learning respectively. Mutual information and information theory is used in compression method.

In contrast to feature extraction, feature selection aims to retain a subset of K best features from an original set of N features and the remaining features are discarded. Feature selection techniques do not alter the original representation of the features. The best known feature selection techniques are filter, wrappers and embedded methods.

The dimensionality reduction technique that we used in this dissertation is the feature selection[11]. While feature selection can be applied to both supervised and unsupervised learning, we merely focus on the problem of classification here. The remaining of this chapter provides a brief survey of the most common feature

selection approaches that can be found in literature.

2.6 Map-Reduce programming paradigm

The MapReduce programming paradigm[6] is a technique for data processing tool for Big data, designed by Google in 2003. MapReduce is based on two separate user-defined primitives: Map and Reduce.

Map function reads the raw data in form of key-value (¡key, value¿) pairs and transforms them into a set of intermediate ¡key, value¿ pairs, where both the key and value types must be defined by the user. In the next stage, MapReduce merges all the values associated with same intermediate key as a list which is called as Shuffle phase[6]. In the last stage, reduce function takes the grouped output from the maps and aggregates it into a smaller set of pairs. This process can be visualized in the below diagram.

The above MapReduce Paradigm is transparent and scalable platform which automatically processes data in a distributed cluster, relieving the user from technical details such as data partitioning, fault tolerance or job communication[30].

Apache Hadoop[29] is a well-known open source implementation of MapReduce for large scale data processing and storage of data across the cluster. Two main module of the Hadoop is Hadoop Distributed File System (HDFS) and MapReduce[6]. HDFS is a distributed file system which enables the user to distribute the files across the several systems. The files in the HDFS[29] are automatically synced throughout the distribution. Its inability to reuse data through in memory primitives makes the application of Hadoop unfeasible for many machine learning algorithms

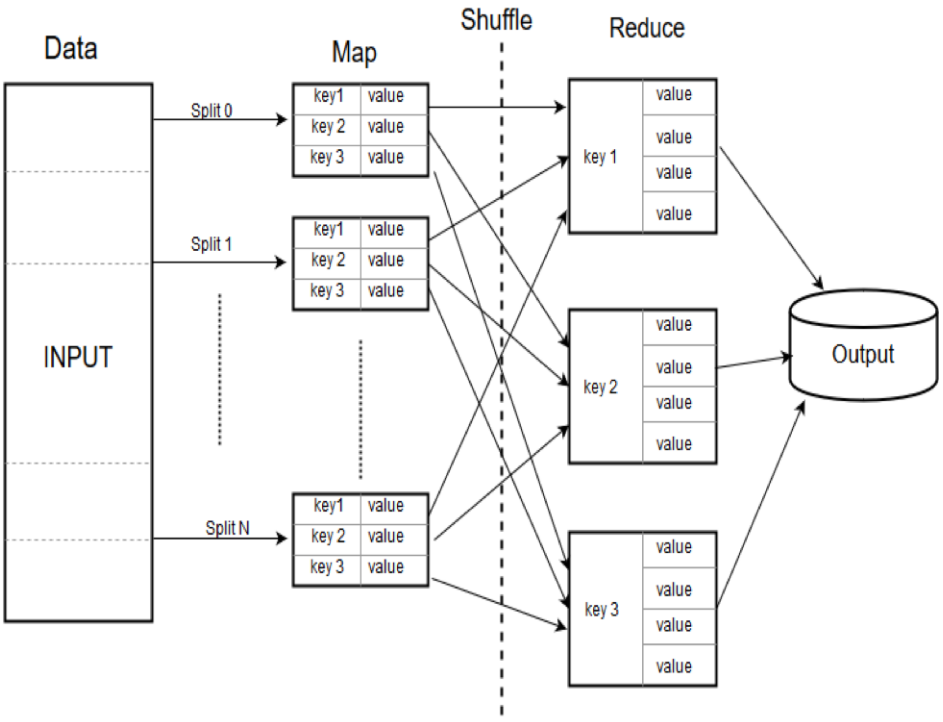


Figure 2.5: Data Flow overview of MapReduce

MapReduce and its variants have been highly successful in implementing large-scale data-intensive applications on commodity clusters. However, most of these systems are built around an acyclic data flow model that is not suitable for other popular applications[30]. Whereas, Apache Spark[34] reuses a working set of data across multiple parallel operations. This includes many iterative machine learning algorithms, as well as interactive data analysis tools. Apache Spark supports these applications while retaining the scalability and fault tolerance of MapReduce. To achieve these goals, Spark introduces an abstraction called resilient distributed datasets (RDDs)[33]. "An RDD is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Spark can outperform Hadoop by 10x in iterative machine learning jobs, and can be used to interactively query a 39 GB dataset with sub-second response time". [33].

Systems like Dyrad and Map-Reduce-Merge particularly achieve their fault tolerance and scalability by providing a programming model where the user creates acyclic data flow graphs to pass input data through a set of operators. This mainly allows the underlying system to manage scheduling and to react to faults without

user intervention. While this data flow programming model is essential for a large class of applications, there are applications that cannot be expressed efficiently as acyclic data flows. In this paper, we focus on one such class of applications: those that reuse a working set of data across multiple parallel operations[34].

2.7 Apache Spark

Apache Spark, a modified large scale data processing which was developed to solve the problems of the Hadoop. Spark was introduced as the part of Hadoop ecosystem which take the advantage of Hadoop by using its distributed file system. Spark framework proposed a set of in-memory computation and analysis with the aim of processing data more rapidly on distributed environments, up to 100x faster than Hadoop[34]. It provides the developer with an easy interface accessible through Scala, Java and Python and has complete machine learning library built-in. MapReduce and its variants have been highly successful in implementing large-scale data-intensive applications on commodity clusters. However, most of these systems are built around an acyclic data flow model that is not suitable for other popular applications. Apache Spark focuses on one such class of applications: those that reuse a working set of data across multiple parallel operations. This includes many iterative machine learning algorithms, as well as interactive data analysis tools. Spark supports these applications while retaining the scalability and fault tolerance of MapReduce. To achieve these goals, Spark introduces an abstraction called resilient distributed datasets (RDDs)[33].

Spark is based on Resilient Distributed Datasets (RDDs)[33], a special type of data structure used to parallelize the computation across the cluster. These parallel structures let us persist and reuse results, cached in memory. A scalable machine learning library (MLlib) was built on top of spark. The spark MLlib[21] contains a large set of standard learning algorithms and statistical tools which has many important functions for knowledge discovery process such as classification, regression, clustering, optimization or data preprocessing. It provides a high-level

API that makes easier for the user to connect multiple machine learning algorithms.

"To efficiently use Apache Spark, developers write a driver program that implements the high-level control flow of their application and launches various operations in parallel. Spark provides two main abstractions for parallel programming: resilient distributed datasets and parallel operations on these datasets (invoked by passing a function to apply on a dataset)". [34]

2.7.1 Resilient Distributed Datasets (RDDs)

Resilient Distributed Datasets (RDDs)[33], a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude. To achieve fault tolerance efficiently, RDDs provide a restricted form of shared memory, based on coarsegrained transformations rather than fine-grained updates to shared state. However, we show that RDDs are expressive enough to capture a wide class of computations, including recent specialized programming models for iterative jobs, such as Pregel, and new applications that these models do not capture.

In simple terms, a resilient distributed dataset (RDD) is a read-only collection of objects partitioned across multiple machines that can be rebuilt if a partition is lost [a]. The elements of an RDD need not exist in physical storage; instead, a handle to an RDD contains enough information to compute the RDD starting from data in reliable storage. This means that RDDs can always be reconstructed if nodes fail[33].

Spark's parallel operations fit into the MapReduce[6] model. However, they operate on RDDs that can persist across operations. The need to extend MapReduce to support iterative jobs was also recognized by Twister, a MapReduce framework that allows long-lived map tasks to keep static data in memory between jobs. How-

ever, Twister does not currently implement fault tolerance. Spark's abstraction of resilient distributed datasets is both fault-tolerant and more general than iterative MapReduce. A Spark program can define multiple RDDs and alternate between running operations on them, whereas a Twister program has only one map function and one reduce function. This also makes Spark useful for interactive data analysis, where a user can define several datasets and then query them. Spark's broadcast variables provide a similar facility to Hadoop's distributed cache, which can disseminate a file to all nodes running a particular job. However, broadcast variables can be reused across parallel operations[34].

2.7.2 Discretized-Streams(D-Streams)

"Much of "big data" is received in real time, and is most valuable at its time of arrival. For example, a social network may wish to detect trending conversation topics in minutes; a search site may wish to model which users visit a new page; and a service operator may wish to monitor program logs to detect failures in seconds. To enable these low-latency processing applications, there is a need for streaming computation models that scale transparently to large clusters, in the same way that batch models like MapReduce simplified offline processing". Designing such models is challenging, however, because the scale needed for the largest applications (e.g., realtime log processing or machine learning) can be hundreds of nodes[35]. At this scale, two major problems are faults and stragglers (slow nodes). Both problems are inevitable in large clusters, so streaming applications must recover from them quickly. Fast recovery is even more important in streaming than it was in batch jobs: while a 30 second delay to recover from a fault or straggler is a nuisance in a batch setting, it can mean losing the chance to make a key decision in a streaming setting. Unfortunately, existing streaming systems have limited fault and straggler tolerance. Most distributed streaming systems, including Storm , TimeStream , MapReduce Online , and streaming databases , are based on a continuous operator model, in which long-running, stateful operators receive each record, update internal state, and

send new records. While this model is quite natural, it makes it difficult to handle faults and stragglers. Specifically, given the continuous operator model, systems perform recovery through two approaches]: replication, where there are two copies of each node , or upstream backup, where nodes buffer sent messages and replay them to a new copy of a failed node . Neither approach is attractive in large clusters: replication costs 2 the hardware, while upstream backup takes a long time to recover, as the whole system must wait for a new node to serially rebuild the failed node's state by rerunning data through an operator. In addition, neither approach handles stragglers: in upstream backup, a straggler must be treated as a failure, incurring a costly recovery step, while replicated systems use synchronization protocols like Flux to coordinate replicas, so a straggler will slow down both replicas[35].

Many "big data" applications must act on data in real time. Running these applications at ever-larger scales requires parallel platforms that automatically handle faults and stragglers. Unfortunately, current distributed stream processing models provide fault recovery in an expensive manner, requiring hot replication or long recovery times, and do not handle stragglers[35]. A processing model called, discretized streams (D-Streams), overcomes these challenges. D-Streams enable a parallel recovery mechanism that improves efficiency over traditional replication and backup schemes, and tolerates stragglers. We show that they support a rich set of operators while attaining high per-node throughput similar to single-node systems, linear scaling to 100 nodes, sub-second latency, and sub-second fault recovery. Finally, D-Streams can easily be composed with batch and interactive query models like MapReduce, enabling rich applications that combine these modes. We implement D-Streams in a system called Spark Streaming[35].

Though there has been a wide set of work on distributed stream processing, most previous systems use the same continuous operator model. In this model, streaming computations are divided into a set of long-lived stateful operators, and each operator processes records as they arrive by updating internal state (e.g., a table tracking page view counts over a window) and sending new records in response[35].

While continuous processing minimizes latency, the stateful nature of operators, combined with nondeterminism that arises from record interleaving on the network, makes it hard to provide fault tolerance efficiently. Specifically, the main recovery challenge is rebuilding the state of operators on a lost, or slow, node. Previous systems use one of two schemes, replication and upstream backup [20], which offer a sharp tradeoff between cost and recovery time. In replication, which is common in database systems, there are two copies of the processing graph, and input records are sent to both. However, simply replicating the nodes is not enough; the system also needs to run a synchronization protocol, such as Flux or Borealis's DPC, to ensure that the two copies of each operator see messages from upstream parents in the same order. For example, an operator that outputs the union of two parent streams (the sequence of all records received on either one) needs to see the parent streams in the same order to produce the same output stream, so the two copies of this operator need to coordinate. Replication is thus costly, though it recovers quickly from failures[35].

In upstream backup, each node retains a copy of the messages it sent since some checkpoint. When a node fails, a standby machine takes over its role, and the parents replay messages to this standby to rebuild its state. This approach thus incurs high recovery times, because a single node must recompute the lost state by running data through the serial stateful operator code. TimeStream and MapReduce Online use this model. Popular message queueing systems, like Storm , also use this approach, but typically only provide "at-least-once" delivery for messages, relying on the user's code to handle state recovery[35].

D-Streams[35] avoid the problems with traditional stream processing by structuring computations as a set of short, stateless, deterministic tasks instead of continuous, stateful operators. They then store the state in memory across tasks as fault-tolerant data structures (RDDs) that can be recomputed deterministically. Decomposing computations into short tasks exposes dependencies at a fine granularity and allows powerful recovery techniques like parallel recovery and speculation.

Beyond fault tolerance, the D-Stream model gives other benefits, such as powerful unification with batch processing.

Finally, to recover from faults and stragglers, both DStreams and RDDs track their lineage, that is, the graph of deterministic operations used to build them. Spark tracks this information at the level of partitions within each distributed dataset. When a node fails, it recomputes the RDD partitions that were on it by re-running the tasks that built them from the original input data stored reliably in the cluster. The system also periodically checkpoints state RDDs (e.g., by asynchronously replicating every tenth RDD)5 to prevent infinite recomputation, but this does not need to happen for all data, because recovery is often fast: the lost partitions can be recomputed in parallel on separate nodes. In a similar way, if a node straggles, we can speculatively execute copies of its tasks on other nodes, because they will produce the same result[34].

Because D-Streams are primarily an execution strategy (describing how to break a computation into steps), they can be used to implement many of the standard operations in streaming systems, such as sliding windows and incremental processing, by simply batching their execution into small timesteps. To illustrate, we describe the operations in Spark Streaming, though other interfaces (e.g., SQL) could also be supported. In Spark Streaming, users register one or more streams using a functional API. The program can define input streams to be read from outside, which receive data either by having nodes listen on a port or by loading it periodically from a storage system (e.g., HDFS). It can then apply two types of operations to these streams: Transformations, which create a new D-Stream from one or more parent streams.

Instead of managing long-lived operators, the idea in D-Streams is to structure a streaming computation as a series of stateless, deterministic batch computations on small time intervals. For example, we might place the data received every second (or every 100ms) into an interval, and run a MapReduce operation on each interval

to compute a count.

There are two challenges in realizing the D-Stream model. The first is making the latency (interval granularity) low. Traditional batch systems, such as Hadoop, fall short here because they keep state in replicated, on-disk storage systems between jobs. Instead, we use a data structure called Resilient Distributed Datasets (RDDs), which keeps data in memory and can recover it without replication by tracking the lineage graph of operations that were used to build it.

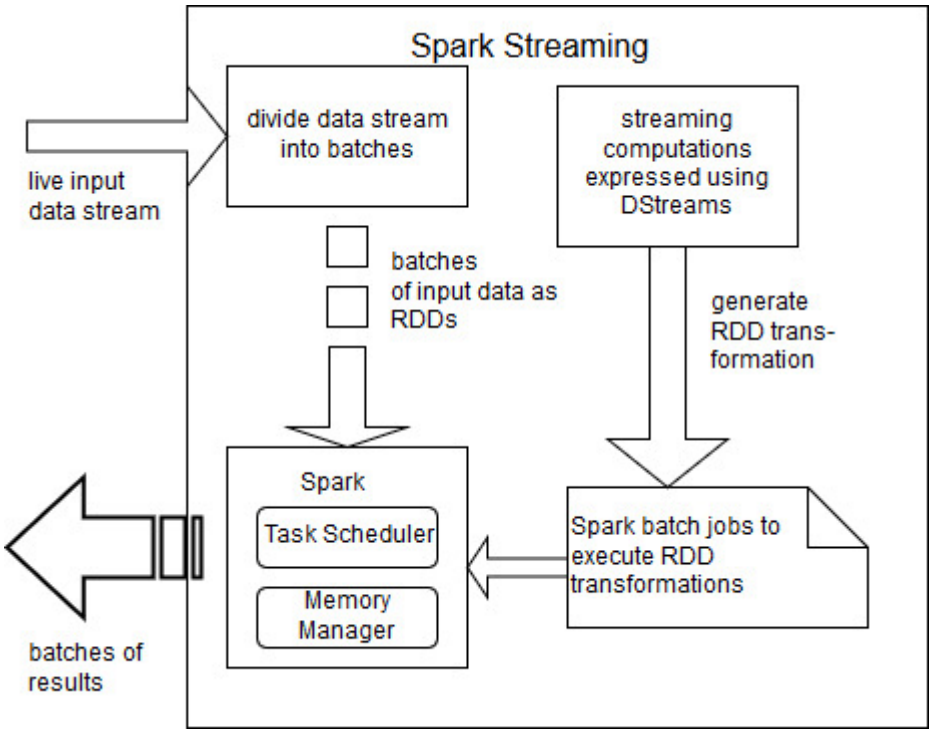


Figure 2.6: High-level overview of the Spark Streaming system. Spark Streaming divides input data streams into batches and stores them in Spark's memory. It then executes a streaming application by generating Spark jobs to process the batches.

Chapter 3

Methodology

This introductory chapter provides the context and approach for the results discussed in the following chapters and defines some crucial notation. The chapter begins with a brief review of our approach, which is the general context for the work described in this thesis. I explain the goals of data discretization, present the main feature selection algorithms used in this thesis.

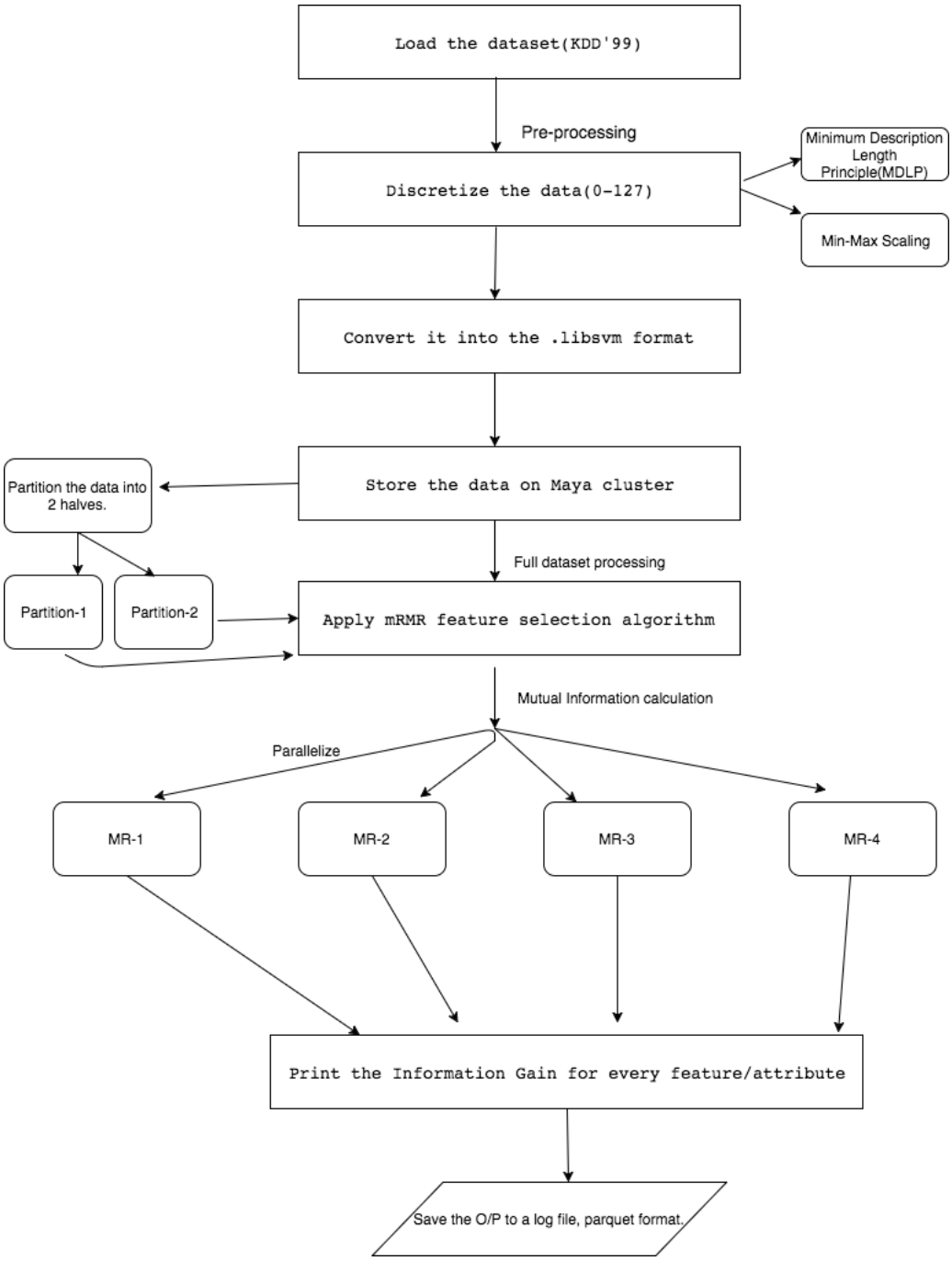


Figure 3.1: Feature selection process for batch data.

3.1 Data Discretization

Discretization of numerical data is one of the most influential data preprocessing tasks in knowledge discovery and data mining. Discretization is considered a data reduction mechanism because it diminishes data from a large domain of numerical values to a subset of categorical values. There Is a necessity to use discretized

data by many algorithms leveraging the Apache Spark platform[26]. Discretization causes that the learning methods show remarkable improvements in learning speed and accuracy. Besides, some decision tree-based algorithms produce shorter, more compact, and accurate results when using discrete values. Even with algorithms that can deal with continuous data, learning is less efficient. Nevertheless, any discretization process generally leads to a loss of information, making the minimization of such information loss is the main goal of a discretizer[26].

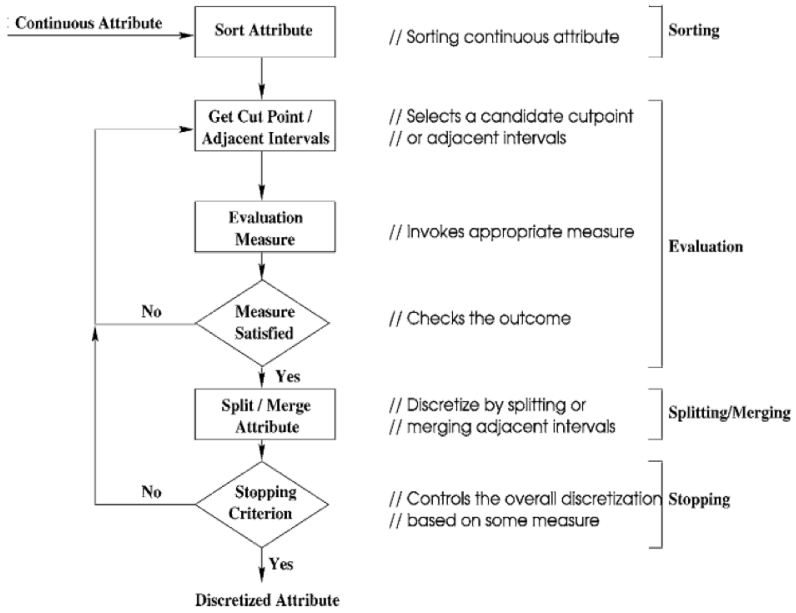


Figure 3.2: A typical discretization process.

3.1.1 MDLP : The Minimum Description Length Principle

Since most real-world applications of classification learning involve continuous-valued attributes, properly addressing the discretization process is an important problem. The Minimum Description Length Principle(MDLP)[9] addresses the use of the entropy minimization heuristic for discretizing the range of a continuous-valued attribute into multiple intervals[fayyad1993multi]. Classification learning algorithms typically use heuristics to guide their search through the large space of possible relations between combination of attribute values and classes. One such heuristic uses the notion of selecting attributes locally minimizing the information entropy of the

classes in a data set[9].

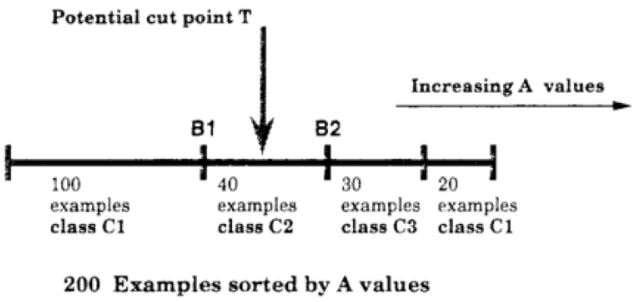


Figure 3.3: A potential within-class cut point.

The attributes in a learning problem may be nominal (categorical), or they may be continuous(numerical). The term "continuous" is used in the literature to refer to attributes taking on numerical values (integer or real); or in general an attribute with a linearly ordered range of values. The above-mentioned attribute selection process assumes that all attributes are nominal. Continuous-valued attributes are discretized prior to selection, typically by partitioning the range of the attribute into subranges. In general, a discretization is simply a logical condition, in terms of one or more attributes, that serves to partition the data into at least two subsets[9].

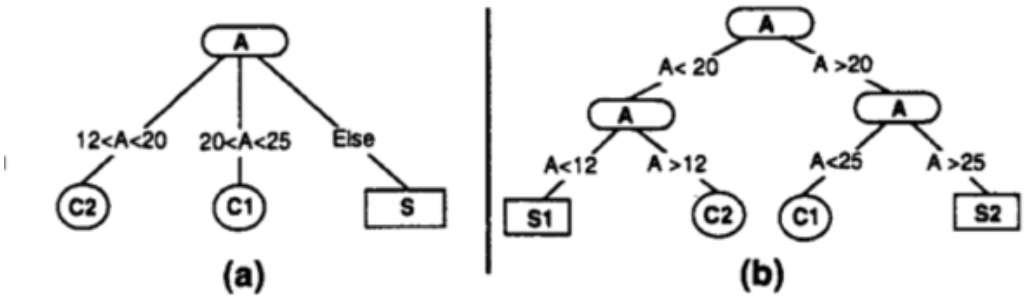


Figure 3.4: Decision Trees and Multi-interval Discretization.

3.1.2 MinMax Scaler

MinMax Scaler[24] transforms features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, i.e. between zero and one. The purpose of attribute discretization is to find concise data representations as categories which

are adequate for the learning task retaining as much information in the original continuous attribute as possible[24].

In this thesis, data has been discretized as integer values in double representation with a maximum of 127 distinct values.

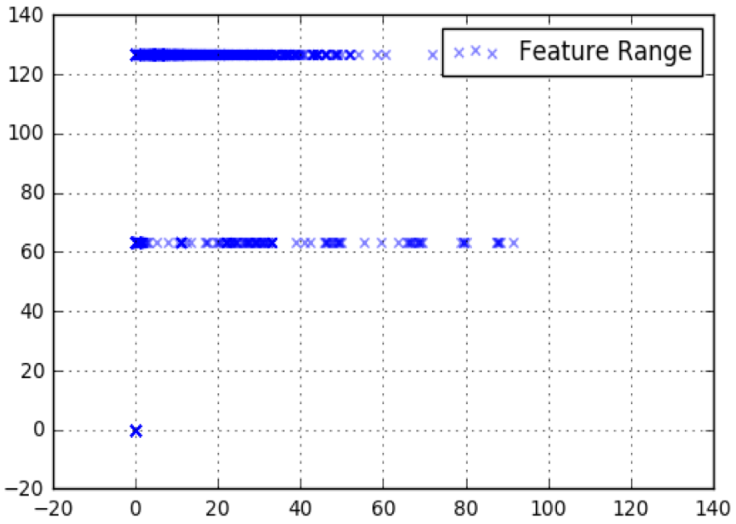


Figure 3.5: MinMax scaling- maximum of 127 distinct values.

3.2 Pearson Correlation

The Pearson product-moment correlation coefficient is a measure of the strength of the linear relationship between two variables[2]. It is referred to as Pearson's correlation or simply as the correlation coefficient. If the relationship between the variables is not linear, then the correlation coefficient does not adequately represent the strength of the relationship between the variables.

The symbol for Pearson's correlation is "p" when it is measured in the population and "r" when it is measured in a sample. Because we will be dealing almost exclusively with samples, we will use 'r' to represent Pearson's correlation unless otherwise noted[2].

Pearson's r can range from -1 to 1. An r of -1 indicates a perfect negative linear relationship between variables, an r of 0 indicates no linear relationship between variables, and an r of 1 indicates a perfect positive linear relationship between variables. Figure 1 shows a scatter plot for which r = 1[2].

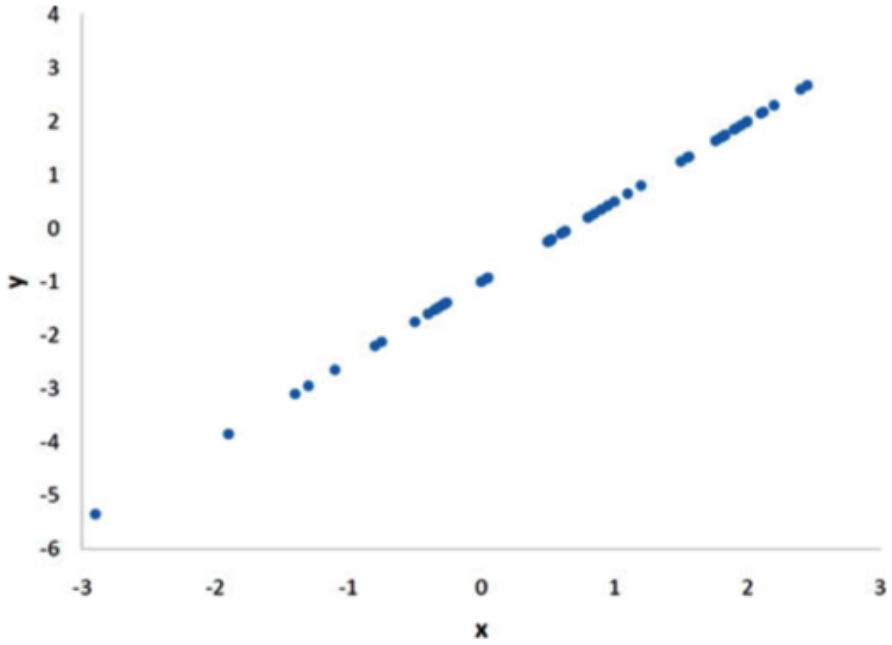


Figure 3.6: A perfect positive linear relationship, r=1.

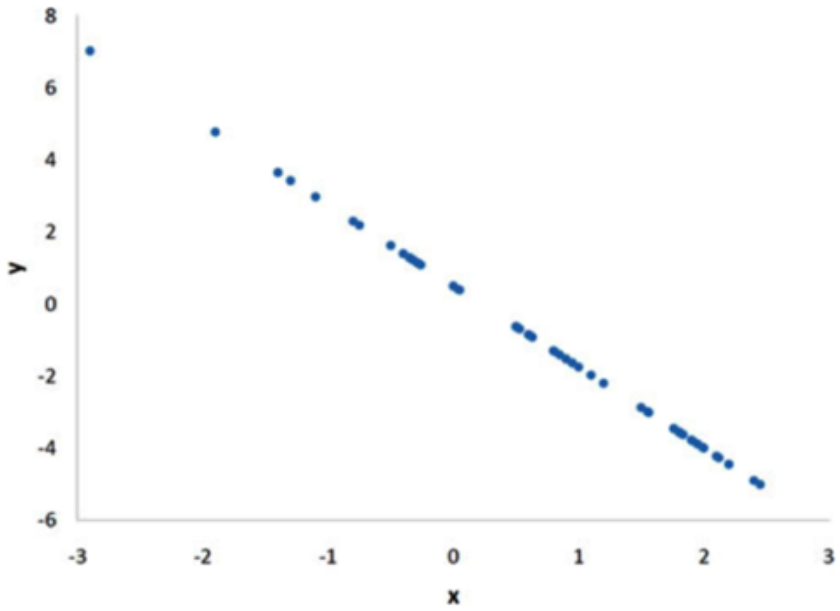


Figure 3.7: A perfect positive linear relationship, r = -1.

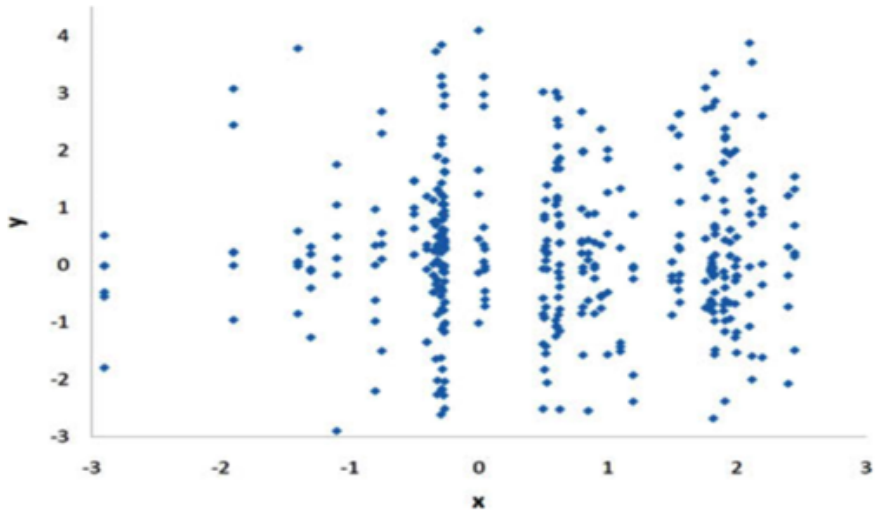


Figure 3.8: A scatter plot for which r = 0. Notice that there is no relationship between X and Y

3.3 Decision Trees

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules[25].

A decision tree consists of three types of nodes[25]:

- Decision nodes – typically represented by squares
- Chance nodes – typically represented by circles
- End nodes – typically represented by triangles

In this thesis, a decision tree model has been designed for network attack classification. Gini impurity and entropy are used as the impurity during the training process of the model. Gini is intended for continuous attributes, and Entropy for attributes that occur in classes. "Gini" will tend to find the largest class, and "entropy" tends to find groups of classes that make up 50 percent of the data[cite]. "Gini" is to minimize misclassification and "Entropy" for exploratory analysis[27].

```
Learned classification tree model:
DecisionTreeModel classifier of depth 15 with 261 nodes
  If (feature 28 <= 0.32)
   If (feature 4 <= 28.0)
    If (feature 4 <= 0.0)
     Predict: 1.0
    Else (feature 4 > 0.0)
     If (feature 2 in {10.0,37.0,25.0,46.0,38.0,41.0,34.0,27.0,7.0,3.0,8.0,19.0})
      If (feature 29 <= 0.5)
       If (feature 36 <= 0.0)
        Predict: 0.0
       Else (feature 36 > 0.0)
        If (feature 0 <= 1.0)
         Predict: 1.0
        Else (feature 0 > 1.0)
         Predict: 0.0
      Else (feature 29 > 0.5)
       Predict: 1.0
     Else (feature 2 not in {10.0,37.0,25.0,46.0,38.0,41.0,34.0,27.0,7.0,3.0,8.0,19.0})
      Predict: 1.0
   Else (feature 4 > 28.0)
    If (feature 2 in {0.0,5.0,10.0,37.0,65.0,41.0,45.0,7.0,3.0,19.0,4.0})
     Predict: 0.0
    Else (feature 2 not in {0.0,5.0,10.0,37.0,65.0,41.0,45.0,7.0,3.0,19.0,4.0})
     If (feature 0 <= 343.0)
      Predict: 1.0
     Else (feature 0 > 343.0)
      Predict: 0.0
  Else (feature 28 > 0.32)
   If (feature 5 <= 0.0)
```

Figure 3.9: Decision Tree model in PySpark for network attack detection using KDD'99

3.4 mRMR Feature Selection

Feature selection is an important pre-processing tool in data mining. It has been an active field of research and development for the past three decades [11]. As the datasets are getting bigger both in terms of instances and feature count in the fields of biomedical research, intrusion detection and customer relationship management, this enormity causes scalability and performance issues in learning algorithms. Feature selection solves the scalability issue and increases the performance of classification models by eliminating redundant, irrelevant or noisy features from high dimensional datasets[11].

Feature selection, identifies subsets of data that are relevant to the parameters used and is normally called Maximum Relevance. These subsets often contain material which is relevant but redundant and mRMR attempts to address this problem by removing those redundant subsets. mRMR has a variety of applications in many areas such as cancer diagnosis and speech recognition [2]. Mutual Information is taken as the basic criterion to find the feature relevance and redundancy. The mutual information between a feature and class labels defines the relevance of that feature. Again, the mutual information among different features defines the correlation i.e., the redundancy among those features. Now our objective is to find such a feature set for which the mutual information among the features and the class labels are maximized and the mutual information among the features are minimized. Therefore, the goal of the proposed method is to find the most relevant and least redundant feature set.[3]

In the era of Big Data, almost every dataset has a characteristic in similar, that is the large number of features. As a result, selecting the relevant features and ignoring the irrelevant and redundant features has become indispensable. However, when dealing with large amounts of data, most existing feature selection algorithms do not scale well, and their efficiency may significantly deteriorate to the point of becoming inapplicable. For these reasons, we propose a distributed approach for partitioned data using Minimum Redundancy Maximum Relevance (MRMR) feature selection algorithm on the Apache Spark platform. MRMR feature selection, as a preprocessing step to machine learning, is highly efficient for dimensionality reduction, removing unrelated data, improving learning accuracy, and increasing result comprehensibility. Nevertheless, the recent surge in dimensionality of data raises a serious challenge to multiple prevailing feature selection methods with respect to coherence and efficacy.

Minimum redundancy feature selection is an algorithm frequently used in a method to accurately identify characteristics of genes and phenotypes and narrow down their relevance and is usually described in its pairing with relevant feature

selection as Minimum Redundancy Maximum Relevance (mRMR)[7].

On the other hand features can be selected to be mutually far away from each other while still having "high" correlation to the classification variable. This scheme, termed as Minimum Redundancy Maximum Relevance (mRMR) selection has been found to be more powerful than the maximum relevance selection[7].

As a special case, the "correlation" can be replaced by the statistical dependency between variables. Mutual information can be used to quantify the dependency. In this case, it is shown that mRMR is an approximation to maximizing the dependency between the joint distribution of the selected features and the classification variable[7]. Studies have tried different measures for redundancy and relevance measures. A recent study compared several measures within the context of biomedical images. "The optimal characterization condition often means the minimal classification error. In an unsupervised situation where the classifiers are not specified,minimal error usually requires the maximal statistical dependency of the target class c on the data distribution in the subspace R^m (and vice versa). This scheme is maximal dependency (Max-Dependency)." [7]

3.4.1 Feature Selection Process

One of the most popular approaches to realize Max Dependency is maximal relevance (Max-Relevance) feature selection: selecting the features with the highest relevance to the target class c. Relevance is usually characterized in terms of correlation or mutual information, of which the latter is one of the widely used measures to define dependency of variables. In this paper, we focus on the discussion of mutual-information-based feature selection. Given two random variables x and y, their mutual information is defined in terms of their probabilistic density functions[7].

In Max-Relevance, the selected features x^i are required, individually, to have the largest mutual information with the target class c, reflecting the largest dependency on the target class. In terms of sequential search, the m best individual features, i.e., the top m features in the descent ordering are often selected as the m features.

In feature selection, it has been recognized that the combinations of individually good features do not necessarily lead to good classification performance. In other words, "the m best features are not the best m features". Some researchers have studied indirect or direct means to reduce the redundancy among features and select features with the minimal redundancy (Min-Redundancy)[7].

Their work in the paper[7] focuses on three issues that have not been touched in earlier work. First, although both Max Relevance and Min-Redundancy have been intuitively used for feature selection, no theoretical analysis is given on why they can benefit selecting optimal features for classification. Thus, the first goal of this paper was to present a theoretical analysis showing that mRMR is equivalent to Max-Dependency for first-order feature selection, but is more efficient. Second, they have proposed how to combine mRMR with other feature selection methods (such as wrappers) into a two-stage selection algorithm. By doing this, we show that the space of candidate features selected by mRMR is more characterizing. This property of mRMR facilitates the integration of other feature selection schemes to find a compact subset of superior features at very low cost. In our approach, we have stressed that a well-designed filter method, such as mRMR, can be used to enhance the wrapper feature selection, in achieving both high accuracy and fast speed. Our method uses an optimal first-order incremental selection to generate a candidate list of features that cover a wider spectrum of characteristic features. These candidate features have similar generalization strength on different classifiers. They facilitate effective computation of wrappers to find compact feature subsets with superior classification accuracy. Our algorithm is especially useful for large-scale feature/variable selection problems where there are at least thousands of features/variables, such as gene selection. Of note, the purpose of the mRMR approach studied in this dissertation is to maximize the dependency. This typically involves the computation of multivariate joint probability, which is nonetheless difficult and inaccurate. Combining both Max Relevance and Min-Redundancy criteria, the mRMR incremental selection scheme provides a better way to maximize the dependency. In this case,

the difficult problem of multivariate joint probability estimation is reduced to estimation of multiple bivariate probabilities (densities), which is much easier. In most situations, mRMR reduces the feature selection time dramatically for continuous features and improves the classification accuracy significantly.

The main benefit of MRMR feature set is that by reducing mutual redundancy within the feature set, these features capture the class characteristics in a broader scope. Features selected within the MRMR framework are independent of class prediction methods, and thus do not directly aim at producing the best results for any prediction method. The fact that MRMR features improve prediction for all four methods we tested confirms that these features have better generalization property. This also implies that with fewer features the MRMR feature set can effectively cover the same class characteristic space as more features in the baseline approach. Additionally, questing the global optimum strictly might lead to data overfitting. On the contrary, mRMR seems to be a practical way to achieve superior classification accuracy in relatively low computational complexity.

According to [27], Peng et al., " Our experimental results show that, although, in general, more mRMR features will lead to a smaller classification error, the decrement of error might not be significant for each additional feature, or occasionally there could be fluctuation of classification errors. For example, in Fig. 3(will be added), the fifth mRMR feature seemingly has not led to a major reduction of the classification error produced with the first four features. Many factors count for these fluctuations. One cause is that additional features might be noisy. Another possible cause is that the mRMR scheme in (6) takes difference of the relevance term and the redundancy term. It is possible that one redundant feature also has relatively large relevance, so it could be selected as one of the top features. A greater penalty on the redundancy term would lessen this problem. A third possible cause is that the cross-validation method used might also introduce some fluctuations of the error curve. While a more detailed discussion on this fluctuation problem and other potential causes is beyond the scope of this paper, a way to solve this prob-

lem is to use other feature selectors to directly minimize the classification error and remove those potentially unneeded features, as what we do in the second stage of our algorithm".

3.5 Streaming Feature Selection

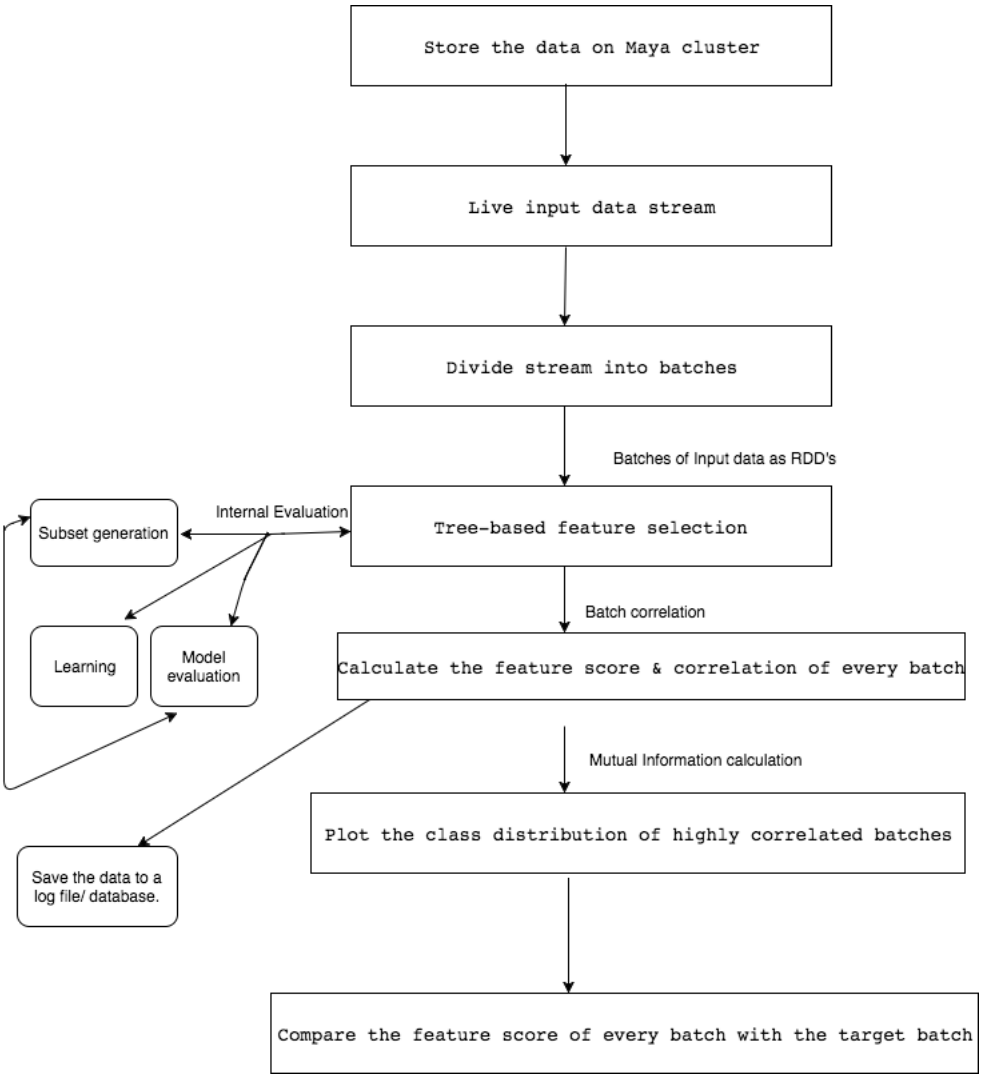


Figure 3.10: Feature selection process for streaming data.

Much of "big data" is received in real time, and is most valuable at its time of arrival. For example, a social network may wish to detect trending conversation topics in minutes; a search site may wish to model which users visit a new page; and a service operator may wish to monitor program logs to detect failures in seconds. To enable these low-latency processing applications, there is a need for streaming

computation models that scale transparently to large clusters. Spark Streaming is one such computation model which can be used for real-time feature selection[35].

Streaming feature selection on Apache Spark supports flexible ordering on the generation and testing of features. Features can be generated dynamically based on which features have already been added to the model. One can also test the same feature more than once, as we do in this section(by using multiple batches). New features can be generated in many ways. Each way produces a new feature class for use in streaming. For example, in addition to the n original features, n^2 pairwise interaction terms can be formed by multiplying all n^2 pairs of features together. In practice, we generate three interaction streams: (1) interactions of features that have already been selected with themselves (2) interactions of the selected features with the original features, and (3) all interactions of the original features. This requires dynamic generation of the feature stream, since the interaction terms (1) and (2) can not be specified in advance, as they depend on which features have already been selected.

The dynamic feature generation and selection schemes, namely (1) and (2) above yield significantly more accurate models on real data sets compared to the approaches which do not use these dynamic interactions. Interaction terms are one example of a more general class of generated features, including features formed from transformations of the original features (square root, log, etc.), or combinations of them including, for instance, PCA. Such generated features frequently lead to substantially better predictive models, but it is not obvious which of the transformations will be most useful. By putting each into its own stream, one can try many transformations at relatively little cost. In contrast, in a conventional batch method, one would need to look at all the features in all the streams, at significant computational cost and, worse, at the cost of statistical power of needing to use a larger penalty to control against overfitting. Including separate feature classes for original features, gives improvement in predictive power.

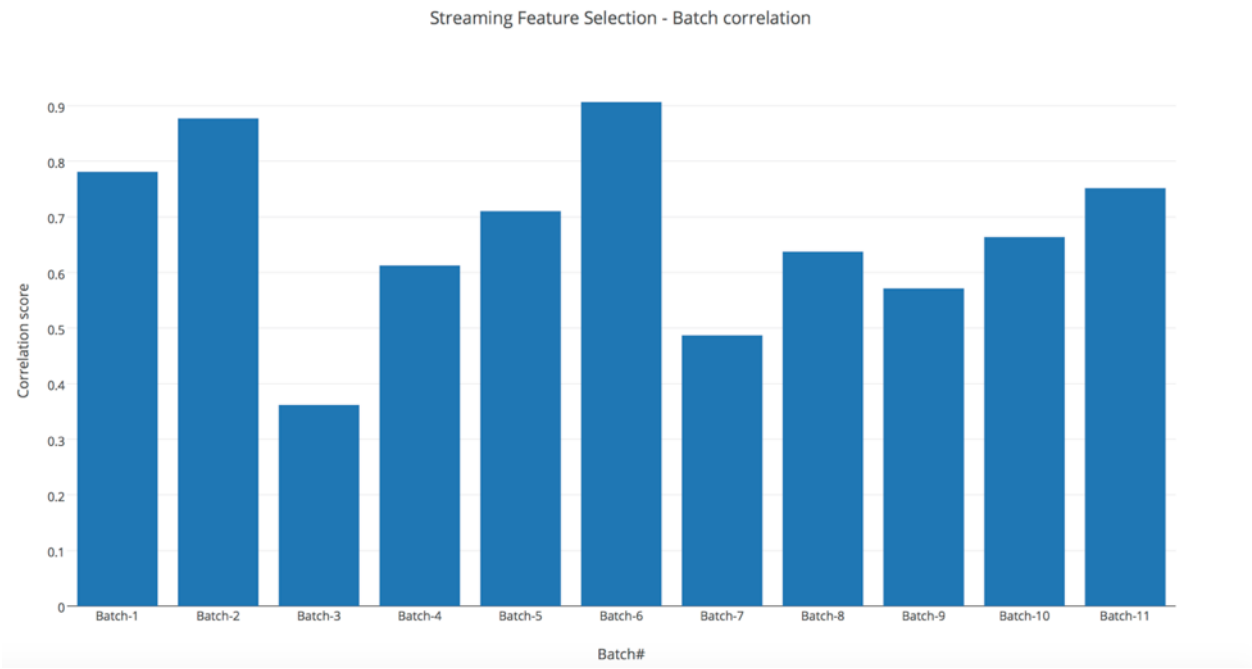


Figure 3.11: Batch correlation on streaming data.

Chapter 4

Implementation and Evaluation

This chapter mainly focuses on the implementation and evaluation of the prototype system that detects intrusions with a limited set of features. To carry out our experimentation we have used the dataset provided by UCI KDD Archive[5]. It is a labeled dataset containing 494021 instances of packet flows. Preprocessing techniques like removing duplicates, removing null values and normalization have been applied on our data set by using Python. Once the data set was preprocessed, multiple(KNN,SVM,J-48) classifiers have been created with the help of the training instances and the respective labels. Finally, we analyzed the performance of our features by running it in on various number of nodes and discovered that as we increase the number of nodes the time taken to perform feature selection diminishes by a large margin.

4.1 Implementation

The dataset which we got from UCI KDD[5] is a labelled comma separated file. With the help of Spark context, we converted the file into RDD (Resilient Distributed Dataset) which is the datatype which resides in memory for computation. Spark gives us the flexibility to convert RDD into data frames which helps us to perform computation in efficient manner. Once all our dataset is converted into Spark data frame it can be distributed across the worker nodes for computation.

4.1.1 Data set description

The dataset which we used in our experiment to access K-NN classifier for Network Intrusion Detection is KDDCup'99 dataset[5] and it is developed by MIT at Lincoln's laboratory. This dataset is derived from the Defense Advanced Research Project Agency (DARPA) packet traces which comprises of variety military network territory simulated intrusions. The KDD dataset is also utilized in the Third International competition that happened on Knowledge Discovery and Data Mining Tools. The goal of this competition was to establish a network detector to find "good" connections and "bad" connections[**nskh2016principle**].

The entire KDDCup'99 dataset (extract the kddcup.data.gz file [5]) consist of 4,898,431 records in which every record is of 41 features which are detailed in the below table. We utilized only the 10% part (extract the kddcup.data_10_percent.gz file[**nskh2016principle**]) of KDD dataset for the purpose of training and testing. The 10% KDDCup'99 data consist of 494,069 records (each containing 41 features) which are categorized into 4 types of attack. The categories of attack and their distinct types are presented in Table below.

No	Features	Description
1	duration	Duration of the Connection
2	protocol type	Connection protocol (e.g. TCP, UDP, ICMP)
3	service	Destination service
4	flag	Status flag of the connection
5	source bytes	Bytes sent from source to destination
6	destination bytes	Bytes sent from destination to source
7	land	1 if successfully logged in; 0 otherwise
8	wrong fragment	Number of wrong fragment
9	urgent	Number of urgent packets
10	hot	Number of "hot" indicator
11	failed logins	Number of failed logins
12	Logged in	1 if successfully logged in; 0 otherwise
13	num_compromised	Number of "compromised" condition
14	root shell	1 if root shell is obtained; 0 otherwise
15	su_attempted	1 if "su root" command attempted; 0 otherwise
16	num root	Number of "root" accesses
17	num file creations	Number of file creation operations
18	num shells	Number of shell prompts
19	num access file	Number of operations on access control files
20	num_outbound cmds	Number of outbound commands in a ftp session
21	is hot login	1 if login belongs to the "hot" list; 0 otherwise
22	is guest login	1 if login is the "guest" login; 0 otherwise
23	count	Number of connections to the same host as the current connection in the past 2 seconds
24	srv count	Number of connections to the same service as the current connection in the past two seconds
25	serror rate	% of connections that have "SYN" errors
26	srv serror rate	% of connections that have "SYN" errors
27	rerror rate	% of connections that have "REJ" errors
28	srv rerror rate	% of connections that have "REJ" errors
29	same srv rate	% of connections to the same service
30	diff srv rate	% of connections to different services
31	srv diff host rate	% of connections to different hosts
32	dst host count	Count of connections have the same destination host
33	dst host srv count	Count of connections have the same destination host and using the same service
34	dst host same srv rate	% of connections having the same destination host and using the same service
35	dst host diff srv rate	% of different service on the current host

Table 4.1: Features list and Description of KDDCup'99 Dataset [**nskh2016principle**]

No	Features	Description
36	dst host same src port rate	% of connections to the current host having the same src port
37	dst host srv diff host rate	% of connections to the same service coming from different host
38	dst host serror rate	% of connections to the current host that have an S0 error
39	dst host srv serror rate	% of connections to the current host and specified service that have an S0 error
40	dst host rerror rate	% of connections to the current host that have an RST error
41	dst host srv rerror rate	% of connections to the current host and specified service that have an RST error

Table 4.2: Features list and Description of KDDCup'99 Dataset [**nskh2016principle**]

Table 4.3: Attack Classification & data types

[Normal]

Attack Types	Class
Normal	Normal
apache2	
Back	
land	
mailbomb	
neptune	
pod	
processtable	
smurf	
teardrop	
udpstorm	

[U2R]

Attack Types	Class
buffer overflow	U2R
loadmodule	
perl	
ps	
rootkit	
sqlattack	
xterm	

[Probe]

Attack Types	Class
ipsweep	Probe
mscan	
portsweep	
saint	
satan	
nmap	

[R2L]

Attack Types	Class
ftp write	R2L
guess passwd	
sendmail	
imap	
multihop	
named	
phf	
snmpgetattack	
snmpguess	
warezmaster	
worm	
xlock	
httptunnel	
xsnoop	
wazerclient	

Class	Training	Test
1	19.69%	19.48%
2	0.83%	1.34%
3	79.24%	73.90%
4	0.01%	0.07%
5	0.23%	5.20%

Figure 4.1: Class distribution(skewed) of KDD Cup 1999.

In the machine learning literature, it has been pointed out that little work has been done in the area of classification by machine learning when there is a highly skewed distribution of the class labels in the data set. In many cases, a classifier tends to be biased towards the majority class resulting in poor classification rates on minority classes. As we can see in the training and test class distribution of the KDD cup 1999 data U2R and R2L attacks constitute 0.24 percent of the training dataset but these attacks take up 5.27 percent in the test data[5].

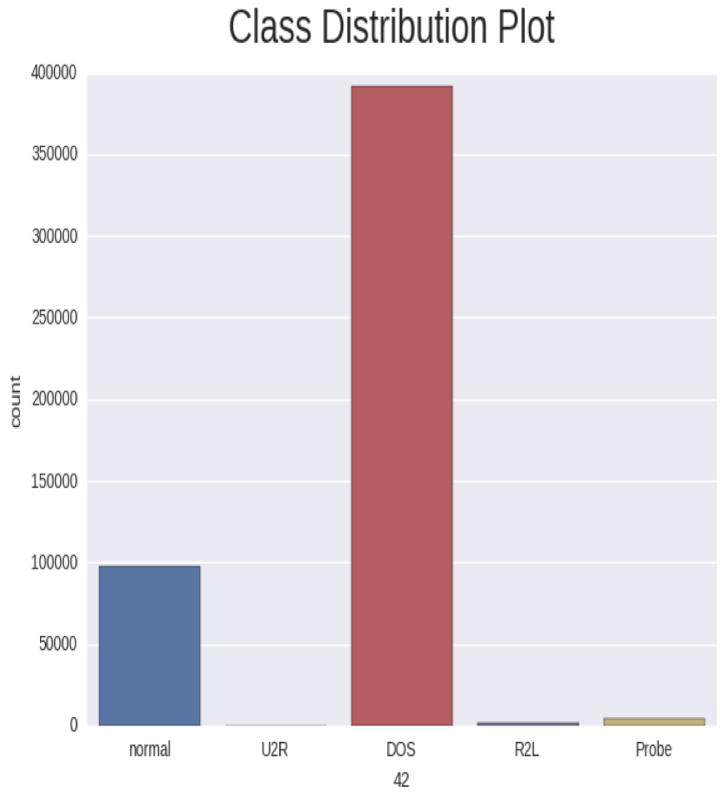


Figure 4.2: Class distribution of training data.

It is important to note that the test data is not from the same probability distribution as the training data, and it includes specific attack types not in the training data. This makes the task more realistic. Some intrusion experts believe that most novel attacks are variants of known attacks and the "signature" of known attacks can be sufficient to catch novel variants. The datasets contain a total of 24 training attack types, with an additional 14 types in the test data only.

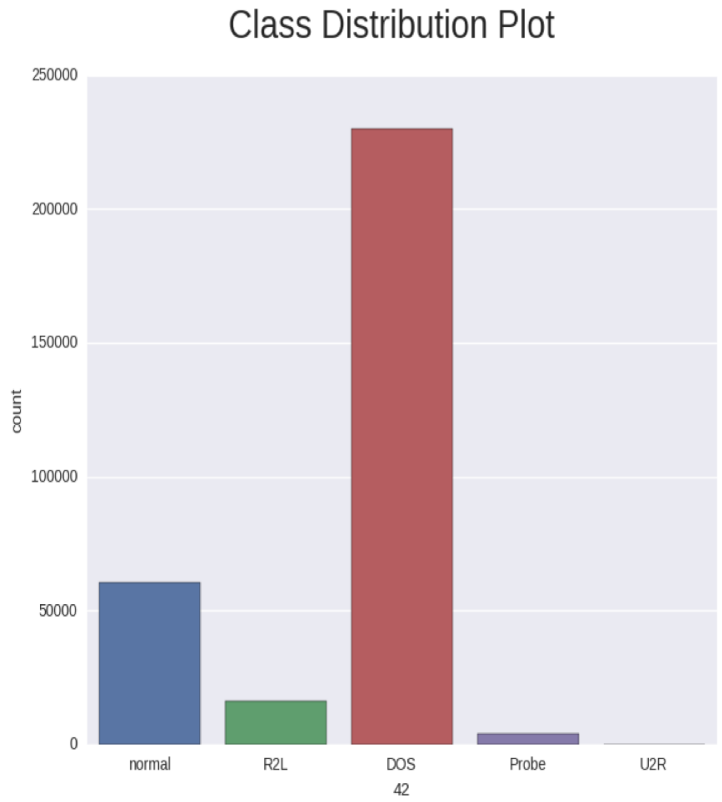


Figure 4.3: Class distribution

4.2 Evaluation

In the KDD'99 dataset we have applied Minimum Redundancy Relevance Feature selection[7] algorithm to calculate which features give us more information about our datasets. After application of the above algorithm we found that the following columns give us the more information about our dataset as shown in Table 4.4. We used the above 25 columns to train our KNN classifier.

We have applied two feature selection techniques (MRMR feature selection and Tree-based feature selection) to the KDD'99 intrusion detection dataset. We have selected the top k (k is set to 7,25) feature subsets for the experiments. After the feature selection, we used 3 learners, KNN, Decision-Tree J48, SVM to build classification models on the datasets with various selected subset of features. The classification models are evaluated in terms of the AUC performance metric, Precision, Recall, F-Measure, Overall Accuracy. The results of the experiments are displayed

in this section. The process of calculating AUC value for a table is performed in three steps:

- Identify the row and column for which the AUC needs to be calculated. This helps in selecting a ranker and a learner.
- Ranker is applied to the dataset to get the ranking list. The top k features are selected from the ranking list. The value of k can be determined by checking the table for which the AUC is calculated.
- Classification model is built using the dataset with selected features from the previous step.

We also compared the results from the subset of features with the results from the complete set of features (base dataset). We found that the classification performance is improved even after a significant number of features were removed from the original dataset. This demonstrates that feature selection was successfully applied to the KDD'99 Intrusion detection dataset.

In our dataset, we have total of 42 features including class attribute. Most of the data mining algorithms had shown inefficiency and are not effective due to high dimensionality [11]. To deal with this problem we performed feature selection process using information-gain [11] to reduce the features. We performed feature selection and achieved success in ranking the attributes based on information gain. After the dimensionality reduction, we split the dataset into two parts training and testing sets to validate our model. The composition of training set from the dataset is 60 percent and the rest is the testing set. After attaining the training set we had removed the zero-day attacks to train classifier and named all the attacks in testing set as 'zero-days' which were removed from training set.

Metric Name	Definition	Formula
Precision P	**Fraction of retrieved instances that are relevant**	**TP/(TP+FP)**
Recall R	**Fraction of relevant instances that are retrieved**	**TP/(TP+FN)**
F-Measure	**Harmonic mean of precision and recall**	**2*P*R/(P+R)**
ROC Curve	**Graphical plot that illustrates the performance of classifier**	**TPR vs FPR**

Figure 4.4: Different types of metrics used.

In the above table the term TP denotes True Positive, TN denotes True Negative, FP denotes False Positive and FN denotes False Negative. TPR, FPR denotes true positive rate and false positive rates respectively. In cyber-security analytics prediction is true positive if the predicted value is an attack and actual value is also an attack, false positive if the predicted value is an attack and actual value is benign, true negative if the predicted value is a benign and actual value is also a benign, false negative if the predicted value is a benign and actual value is an attack. True positive rate is same as Recall and false positive rate is defined as the probability of false alarm [16]. ROC is region of convergence and more bump on the top of diagonal line represents high performance of the classifier and vice versa. Diagnostic type of ROC is illustrated in below figure,

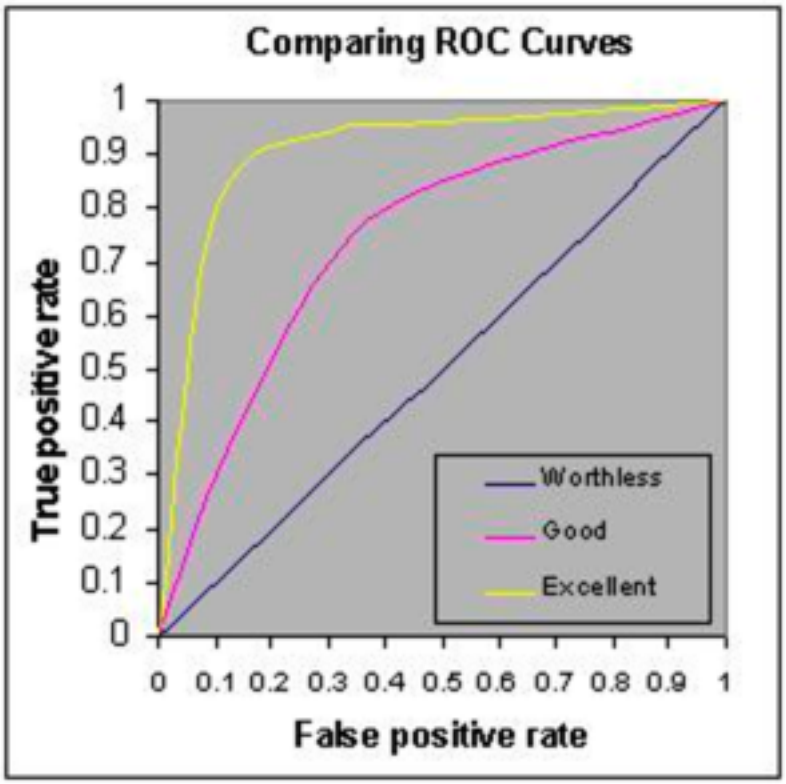


Figure 4.5: AUC-ROC curve.

4.2.1 mRMR feature selection on full dataset(non-partitioned)

MRMR feature selection has been performed on the KDD'99 full dataset and the feature score for every attribute it in the dataset is displayed in the image below. Features with highest mutual information have high score. When k = 25, we get the following feature subset out of 42 features.

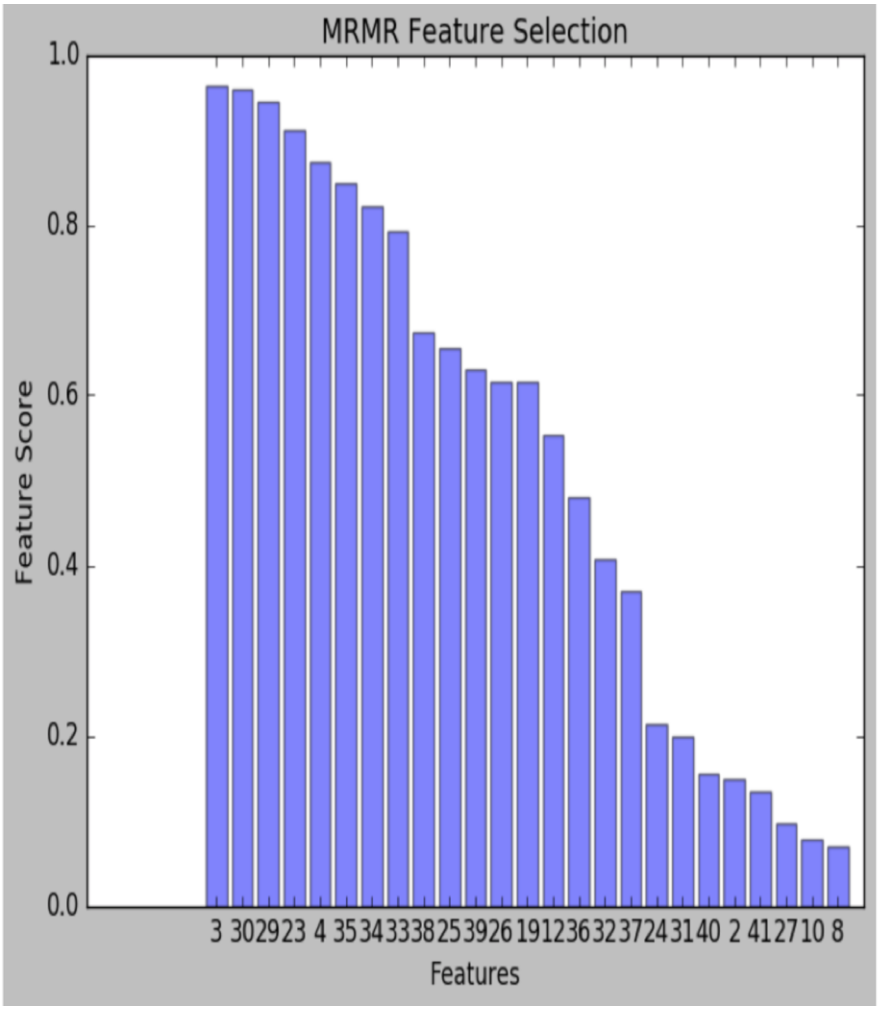


Figure 4.6: mRMR feature selection on full dataset.

score	Features	score	Features
0.9630	service	0.5531	logged_in
0.9452	same_srv_rate	0.4068	dst_host_count
0.9119	count	0.3708	dst_host_srv_diff_host_rate
0.8747	flag	0.2134	srv_count
0.8498	dst_host_diff_srv_rate	0.1999	srv_diff_host_rate
0.8226	dst_host_same_srv_rate	0.1546	dst_host_rerror_rate
0.7932	dst_host_srv_count	0.1489	protocol_type
0.6735	dst_host_serror_rate	0.1338	dst_host_srv_rerror_rate
0.6554	serror_rate	0.0959	rerror_rate
0.6302	dst_host_srv_serror_rate	0.0783	hot
0.6158	srv_serror_rate	0.0704	wrong_fragment
0.6158	num_access_files		

Table 4.4: Feature Score of Top 25 Attributes

4.2.2 mRMR feature selection on partitioned dataset(n=2)

MRMR feature selection has been performed on the KDD'99 partitioned (2 halves) dataset and the feature score for every attribute it in the KDD 1st half dataset is displayed in the image below. Features with highest mutual information have high score. When k = 25, we get the following feature subset out of 42 features.

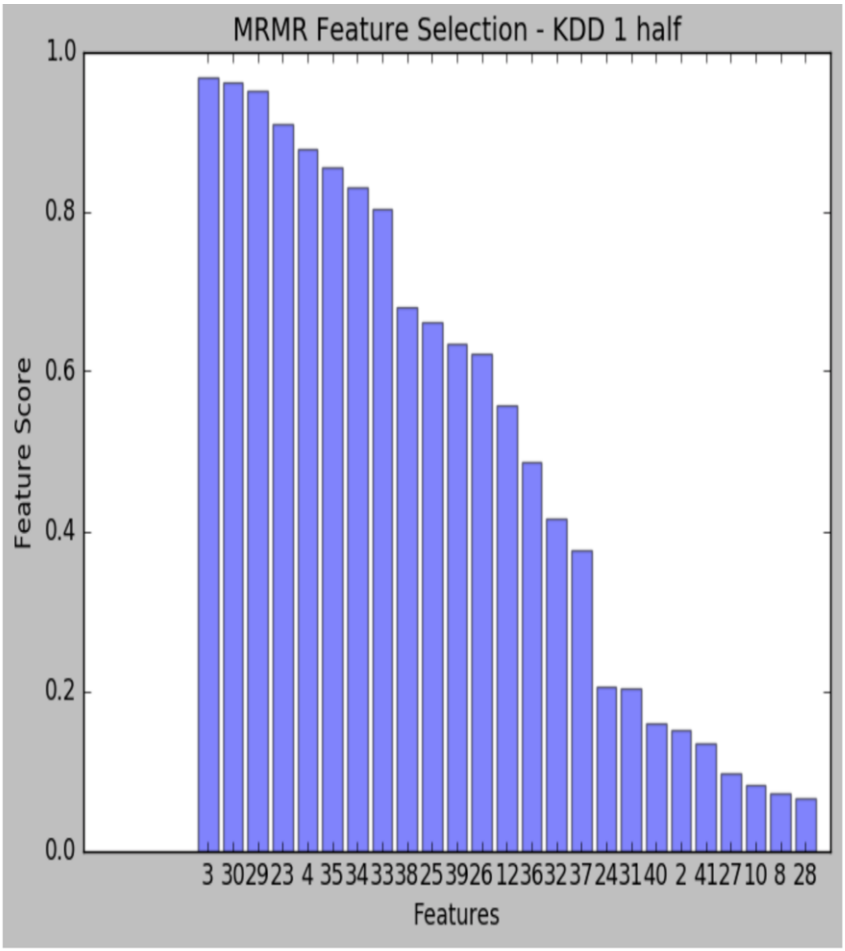


Figure 4.7: mRMR feature selection on partitioned dataset.

The feature score for every attribute it in the KDD 2nd half dataset is displayed in the image below. Features with highest mutual information have high score. When k = 25, we get the following feature subset out of 42 features.

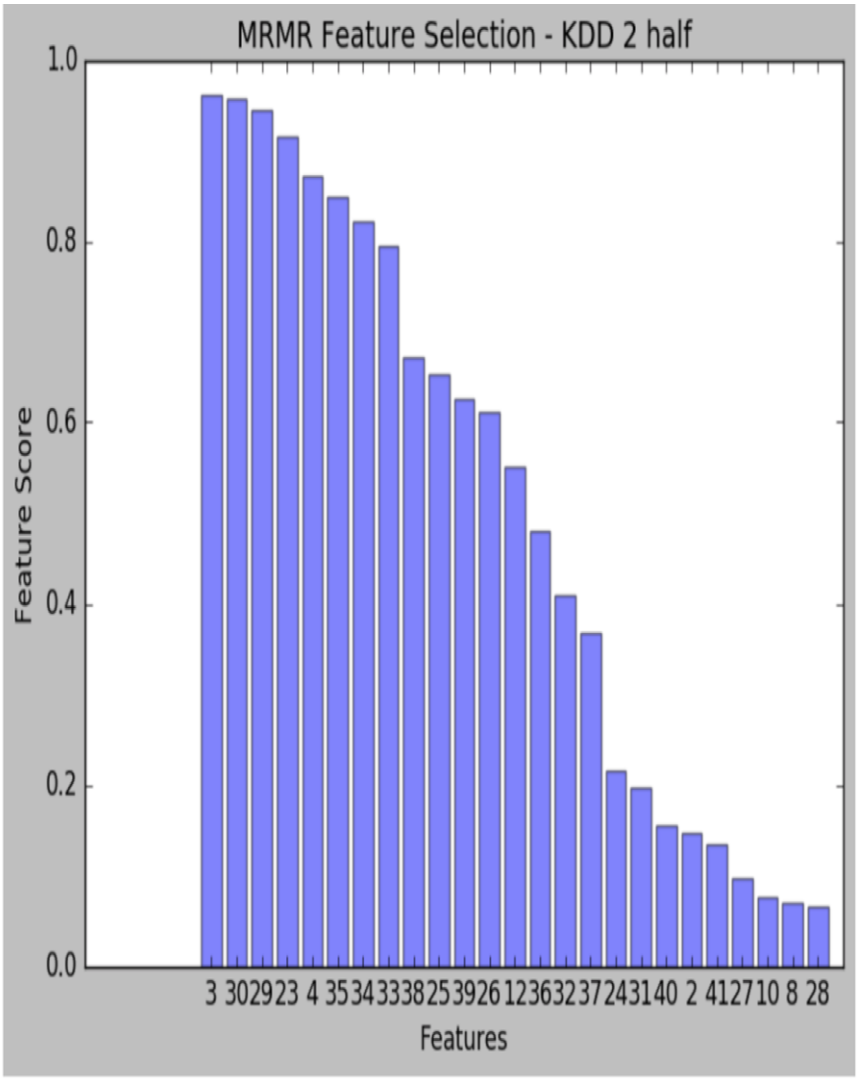


Figure 4.8: mRMR feature selection on partitioned dataset.

4.3 Classifier Results

The tables below summarize the classification performance in terms of AUC, Precision, Recall, and F-1 measure for the three classifiers with top k(k=7,25) features. The tables also display model performance on base dataset.

For our experimentation, we have used 10 percent KDD 1999 dataset [18]. This dataset contains half a million data points. It contains 42 features including label feature which comprises of 22 attack types and a normal. To train and test the classifier we had split the dataset into 60 percent training and 40 percent training set. To identify the zero-day attacks we trained the classifier with 8 attacks out of 22 attack types. In test dataset, we had marked 14 attacks that were removed from training dataset as zero-day attacks. The below table gives a list of the attacks that were marked as zero-day attack and non-zero-day attacks. For our experimental graphical representation, we had given unique number to each attack as shown in the below tables.

4.3.1 K-nearest neighbors' (k=3)

We deployed a KNN classifier and set k value as 3. We split the whole dataset into two parts, training set and testing set in the ration of 3:2. After the creation of training set, we had removed the zero-day attacks which are mentioned in the above table 1. After training we had tested the classifier using test data that is streamed continuously using spark streaming context. Out of 14 zero-day attacks our classifier had predicted 10 zero-day attacks and for rest of them it predicted as benign.

Sample type	Number of instance
Test data	36398
Zero day attacks	5683
Non-Zero attacks	6665
Attack samples	12348
Normal samples	24050

Table 4.5: Depicts the values and the classified instances with k=3

Metric	Count
True positives	11702
False positives	50
True negatives	24000
False negatives	646

Table 4.6: Depicts the metrics and its count with k=3

Classification metric	Value in percentage
Precision	99.572
Recall	94.768
F-measure	97.313
Accuracy of Zero-day prediction	91.836
Accuracy of Non-zero day prediction	97.267
Overall Accuracy	98.225

Table 4.7: Depicts the classification accuracy of our model with k=3.

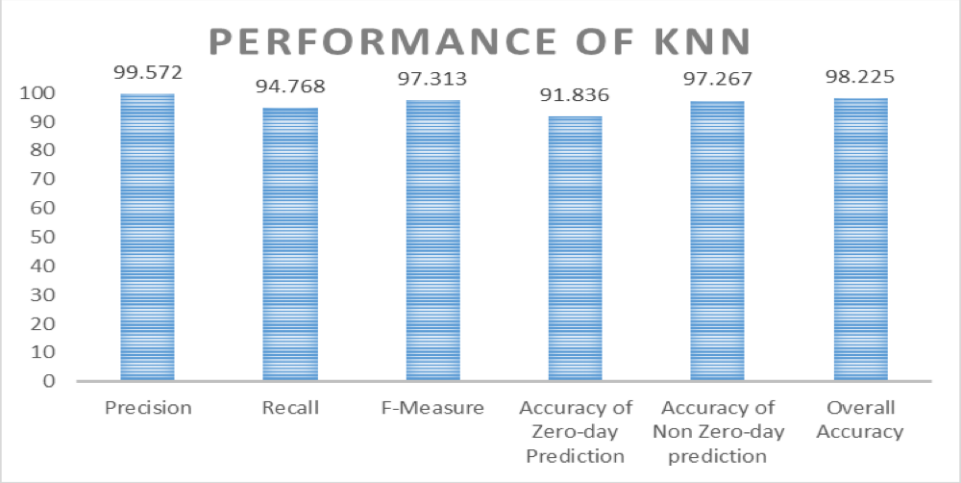


Figure 4.9: Performance of KNN with k=3 (bar-chart).

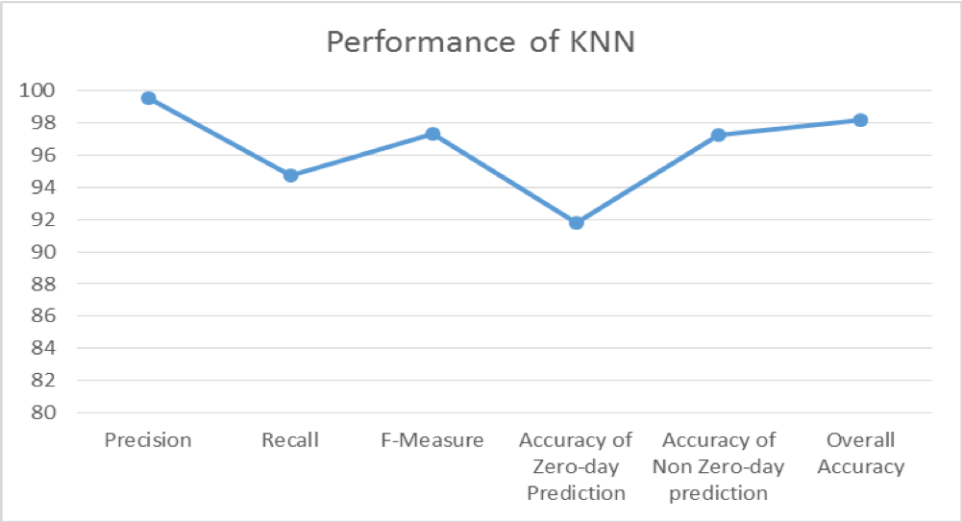


Figure 4.10: Performance of KNN with k=3 (line-graph).

4.3.2 Decision-tree J-48

Sample type	Number of instance
Test data	36398
Zero day attacks	5683
Non-Zero attacks	6665
Attack samples	12348
Normal samples	24050

Table 4.8: Depicts the values and the classified instances using J48

Metric	Count
True positives	10827
False positives	1366
True negatives	22684
False negatives	1521

Table 4.9: Depicts the metrics and its count using J48

Classification metric	Value in percentage
Precision	88.796
Recall	87.6822
F-measure	88.471
Accuracy of Zero-day prediction	79.92
Accuracy of Non-zero day prediction	94.83
Overall Accuracy	92.06

Table 4.10: Depicts the classification accuracy of our J48 model.

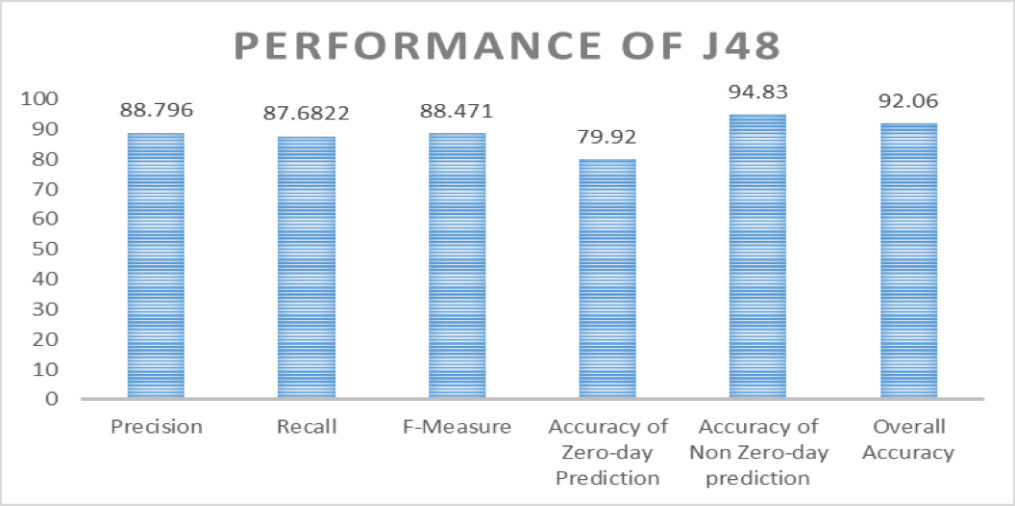


Figure 4.11: Performance of J48 (Bar-chart).

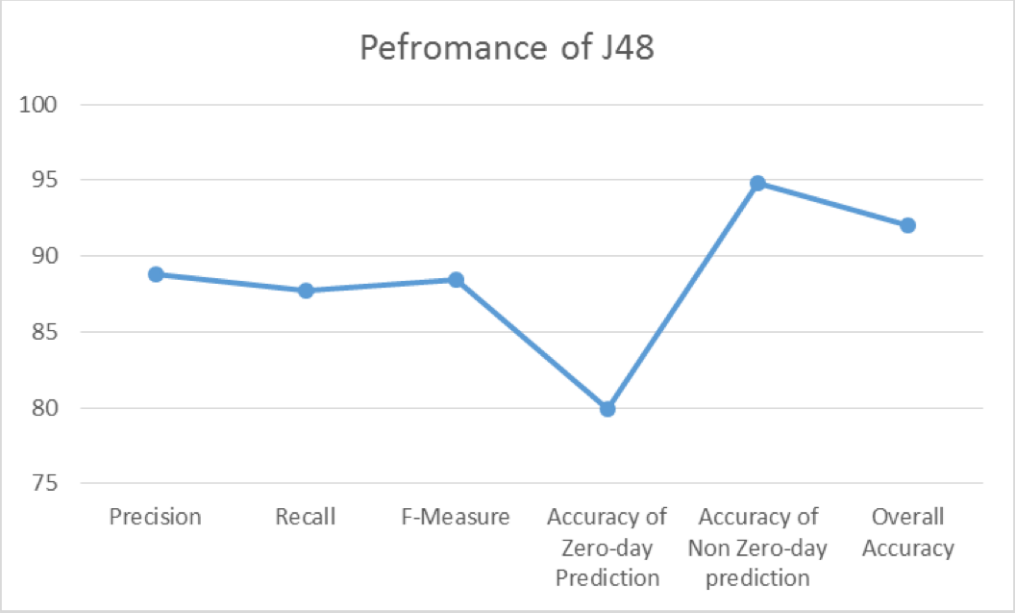


Figure 4.12: Performance of J48 (Line-graph).

4.3.3 Support Vector Machine

Sample type	Number of instance
Test data	36398
Zero day attacks	5683
Non-Zero attacks	6665
Attack samples	12348
Normal samples	24050

Table 4.11: Depicts the values and the classified instances

Metric	Count
True positives	11125
False positives	11
True negatives	24039
False negatives	1223

Table 4.12: Depicts the metrics and its count using SVM

Classification metric	Value in percentage
Precision	99.902
Recall	90.095
F-measure	88.471
Accuracy of Zero-day prediction	88.210
Accuracy of Non-zero day prediction	91.70
Overall Accuracy	96.609

Table 4.13: Depicts the classification accuracy of our SVM model.

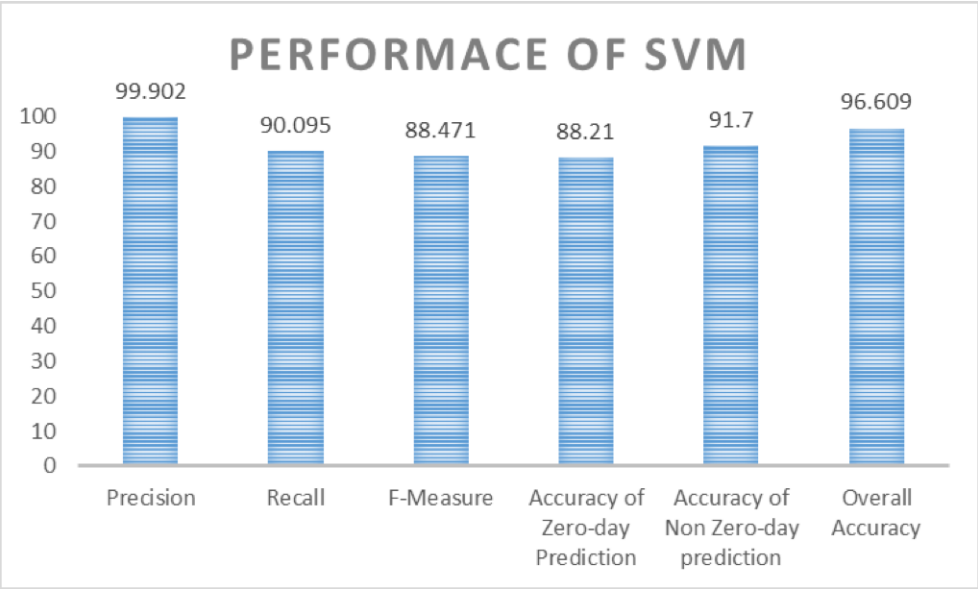


Figure 4.13: Performance of SVM (Bar-chart).

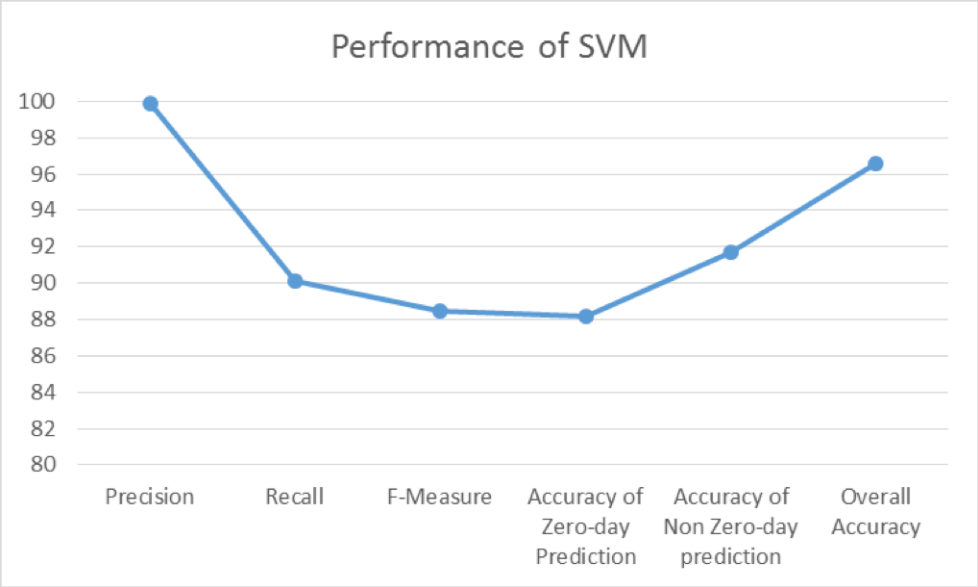


Figure 4.14: Performance of SVM (Line-graph).

4.3.4 Classifier Comparison

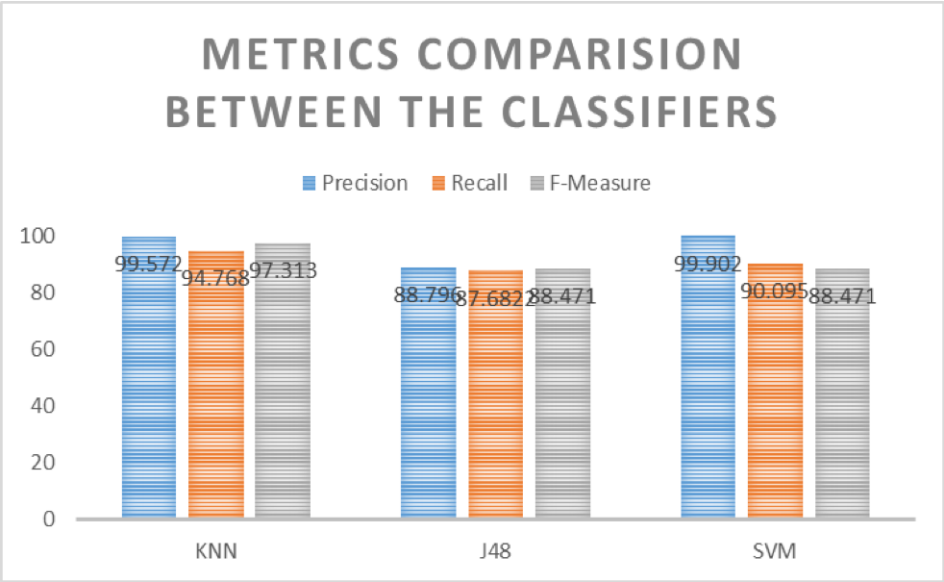


Figure 4.15: Performance of all three classifiers(Bar-chart).

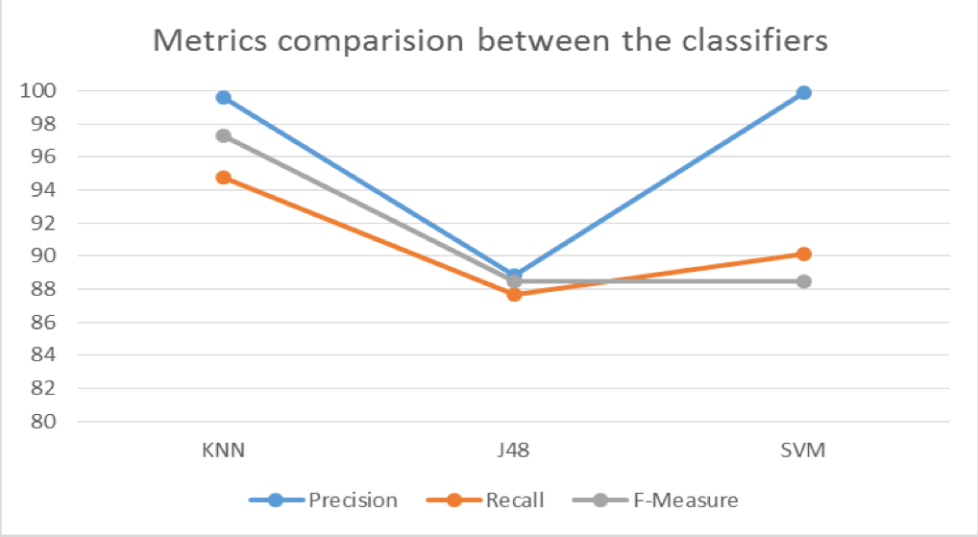


Figure 4.16: Performance of all three classifiers(Line-graph).

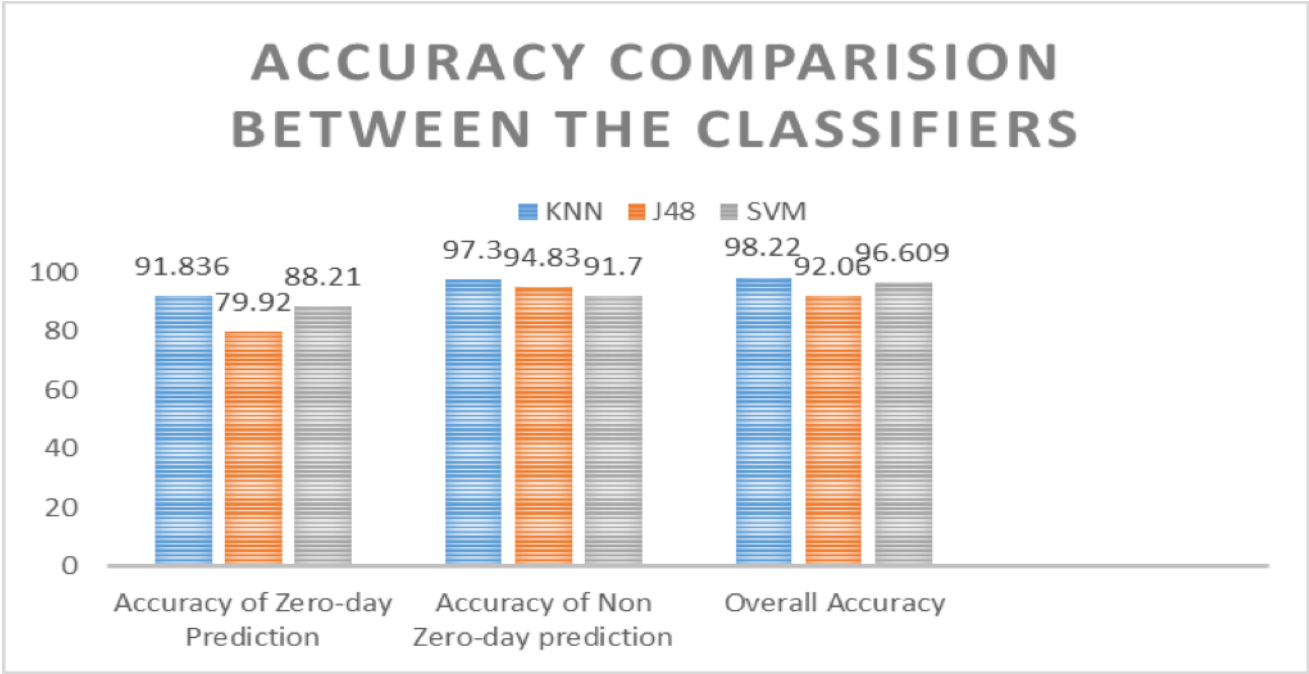


Figure 4.17: Accuracy score of all three classifiers.

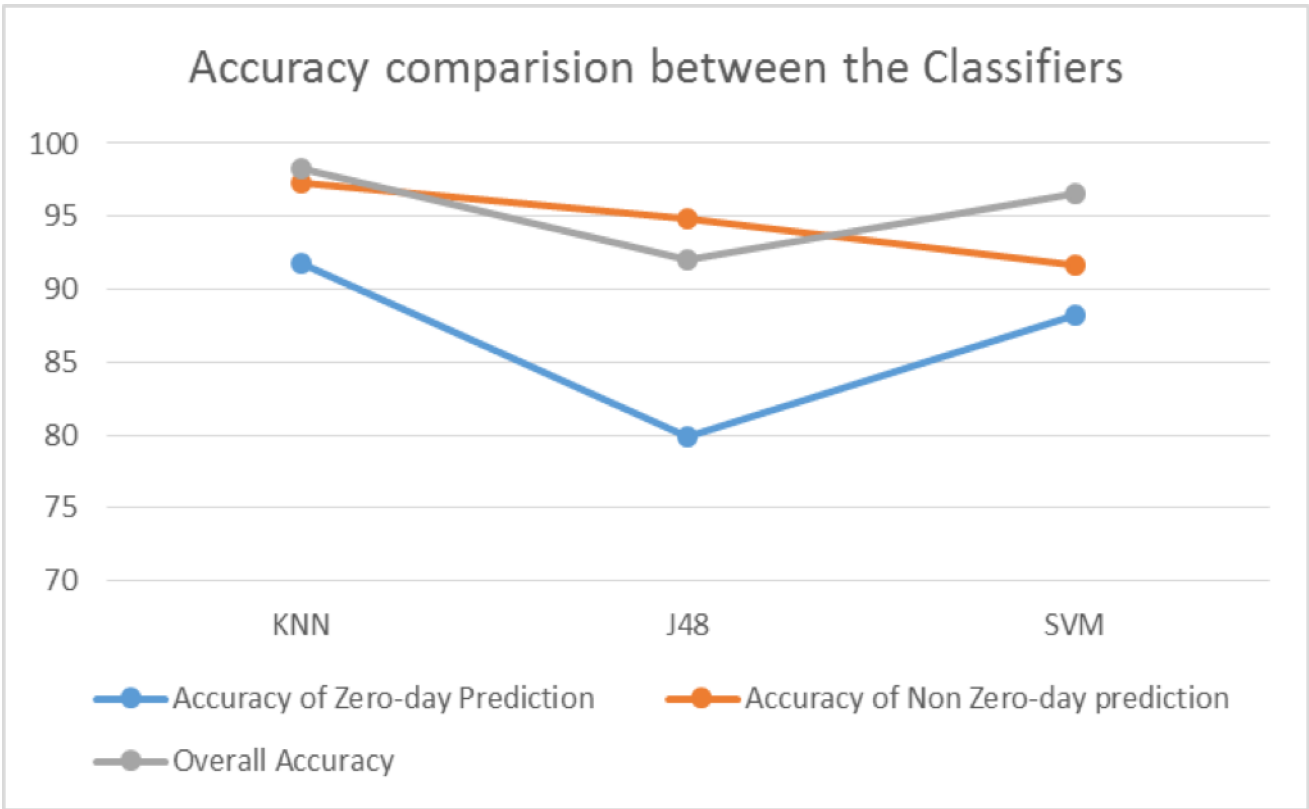


Figure 4.18: Accuracy score of all three classifiers.

From the above tables, we can extrapolate that the KNN classifier which follows the nearest neighbor heuristics has been highly efficient in terms of attack identification when compared to the other classifiers Decision-Tree J-48 and Support Vector Machines(SVM). The reason why J-48 wasn't as efficient as KNN is because the data is continuous and the concept of information has no effect on data distribution. In the same way SVM has not set the effective boundary to separate the classes.

For the KNN classifier it is shown that the precision is 99.572 percent, which means our system had predicted the normal samples correctly with almost 0 percent error rate. After plotting the accuracy and classifier performance we had calculated true positive rate and false positive rate for different values of k ranging from 0 to 3 and then we had plotted ROC curve to make sure that our system is stable in identification of zero day attacks as in the below table.

K value	TPR	FPR
1	0.92	0.12
2	0.939	0.09
3	0.94768	0.03

Table 4.14: ROC Metrics for KNN(k=1,2,3)

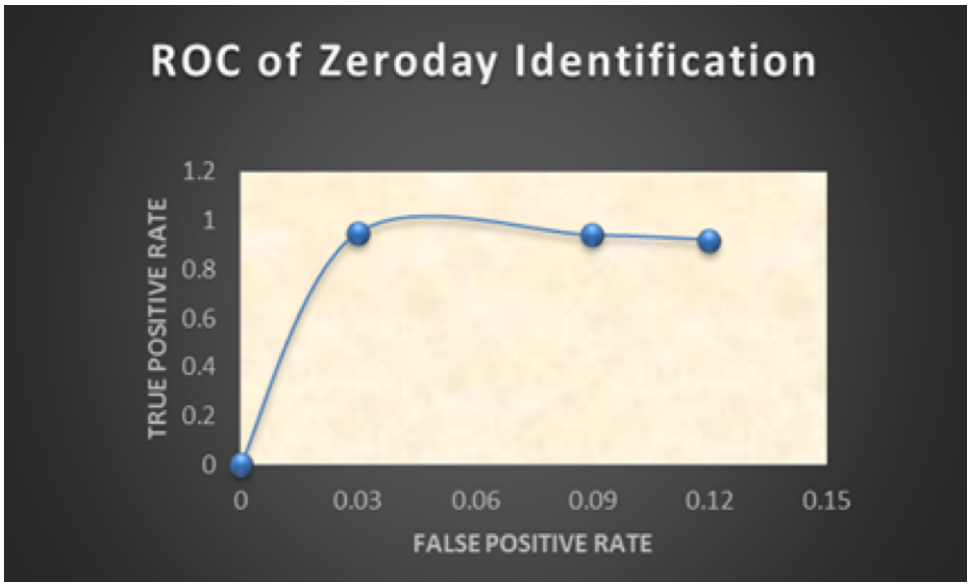


Figure 4.19: ROC-Curve.

4.4 Streaming Feature Selection

Streaming feature selection on Apache Spark supports flexible ordering on the generation and testing of features. Features can be generated dynamically based on which features have already been added to the model. One can also test the same feature more than once, as we do in this section (by using multiple batches). New features can be generated in many ways. Each way produces a new feature class for use in streaming. For example, in addition to the n original features, n^2 pairwise interaction terms can be formed by multiplying all n^2 pairs of features together. In practice, we generate three interaction streams: (1) interactions of features that have already been selected with themselves (2) interactions of the selected features with the original features, and (3) all interactions of the original features. This requires dynamic generation of the feature stream, since the interaction terms (1) and (2) cannot be specified in advance, as they depend on which features have already been selected.

Algorithm 1 Pseudo code for the random forest algorithm

To generate c classifiers:
for $i = 1 \rightarrow c$ **do**
    Randomly sample the training data D with replacement to produce D_i
    Create a root node, N_i containing D_i
    Call BuildTree(N_i)
end for
BuildTree(N):
if N contains instances of only one class **then** *return*
else
    Randomly select x ▷ of the possible splitting features in N
    Select the feature F with the highest information gain to split on
    Create f child nodes of $N, N_1, ..., N_f$, where F has f possible values $(F_1, ..., F_f)$
    for $i = 1 \rightarrow f$ **do**
        Set the contents of $N_i \rightarrow D_i$, where D_i is all instances in N that match F_i
        Call BuildTree(N_i)
    end for
end if

Figure 4.20: Tree based feature selection algorithm psuedo-code

The dynamic feature generation and selection schemes, namely (1) and (2) above yield significantly more accurate models on real data sets compared to the approaches which do not use these dynamic interactions. Interaction terms are one example of a more general class of generated features, including features formed from transformations of the original features (square root, log, etc.), or combinations of them including, for instance, PCA. Such generated features frequently lead to substantially better predictive models, but it is not obvious which of the transformations will be most useful. By putting each into its own stream, one can try many transformations at relatively little cost. In contrast, in a conventional batch method, one would need to look at all the features in all the streams, at significant computational cost and, worse, at the cost of statistical power of needing to use a larger penalty to control against overfitting. Including separate feature classes for original features, gives improvement in predictive power

Streaming feature selection has been performed on multiple batches of data (KDD'99). For our experiments, the batch size is set to 10,000 with the total number of batches being 11. So, our total training values are 110,000. Every batch is streamed with an interval of 2 seconds. A tree-based approach is used to calculate the features and Pearson correlation for calculating the correlation score of every batch with respective to the full batch feature scores. The below figure depicts the correlation score of all the batches (1-11).

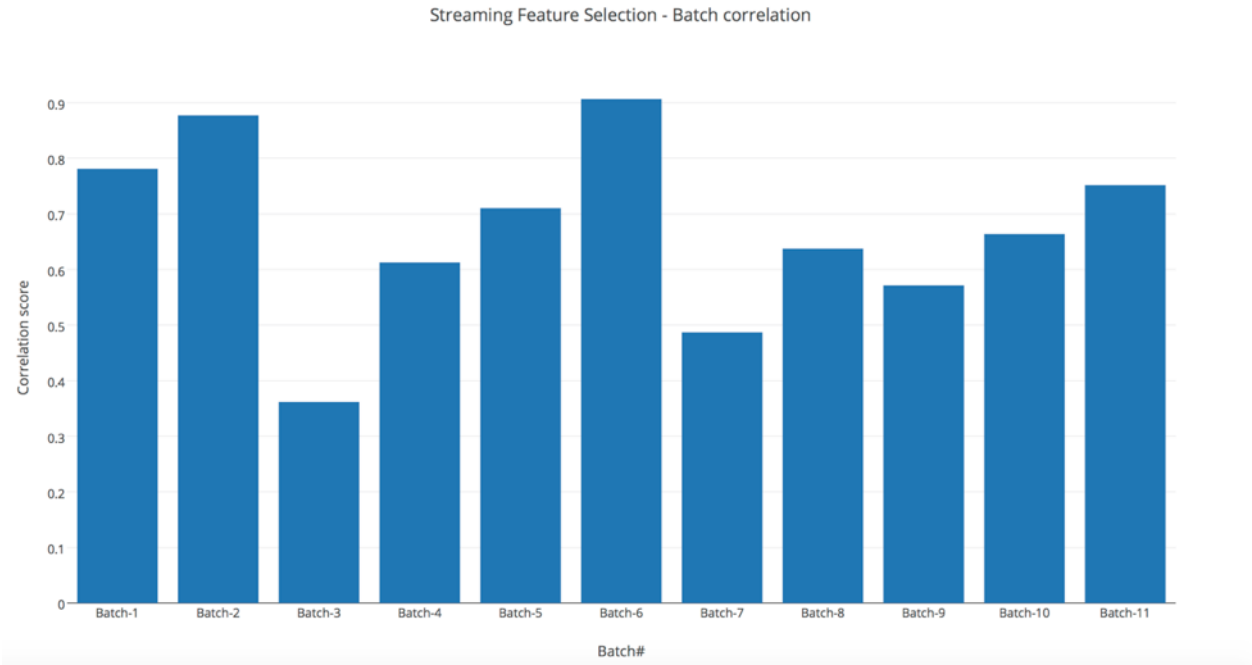


Figure 4.21: Batch correlation on streaming data.

4.4.1 Feature score comparison of highly correlated batches

From figure 4.20, we can say that the Batch-1, Batch-2, and Batch-6 are highly correlated with the full dataset. It basically means that the samples from these batches are highly influential in predicting the class label (i.e; attack v/s normal in our case). Since these batches are highly correlated, we can compare the feature scores of top 5 features from these 3 batches with the feature scores of the target(full-dataset) to see if they are actually similar to each other. Figure 4.21, 4.22, and 4.23 display the feature scores of top 5 features from highly correlated batches and target(full-dataset).

- Correlation score for batch-1 : 0.781127
- Correlation score for batch-1 : 0.877196
- Correlation score for batch-1 : 0.906813

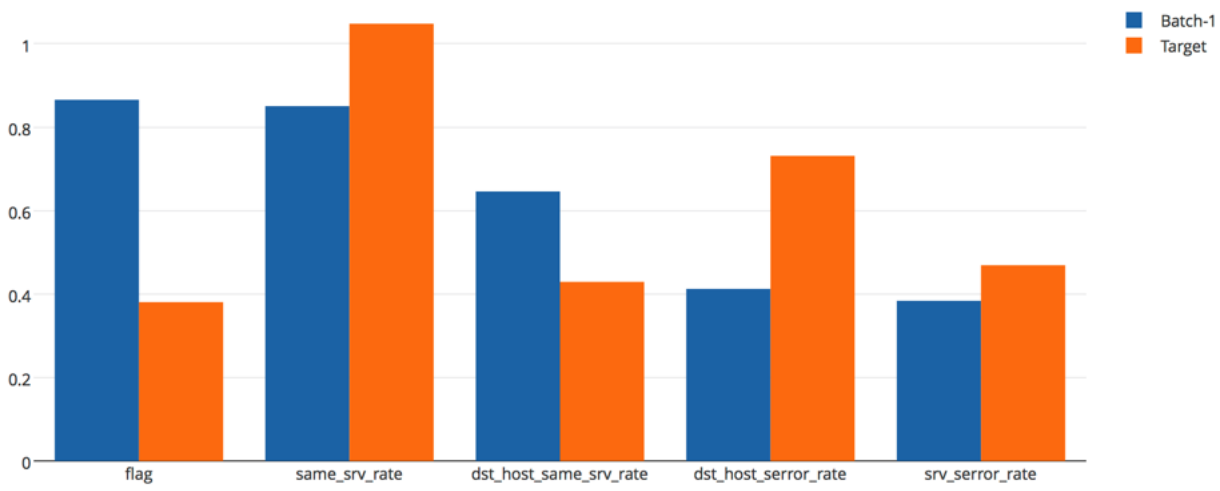


Figure 4.22: Batch-1 v/s Target(full dataset).

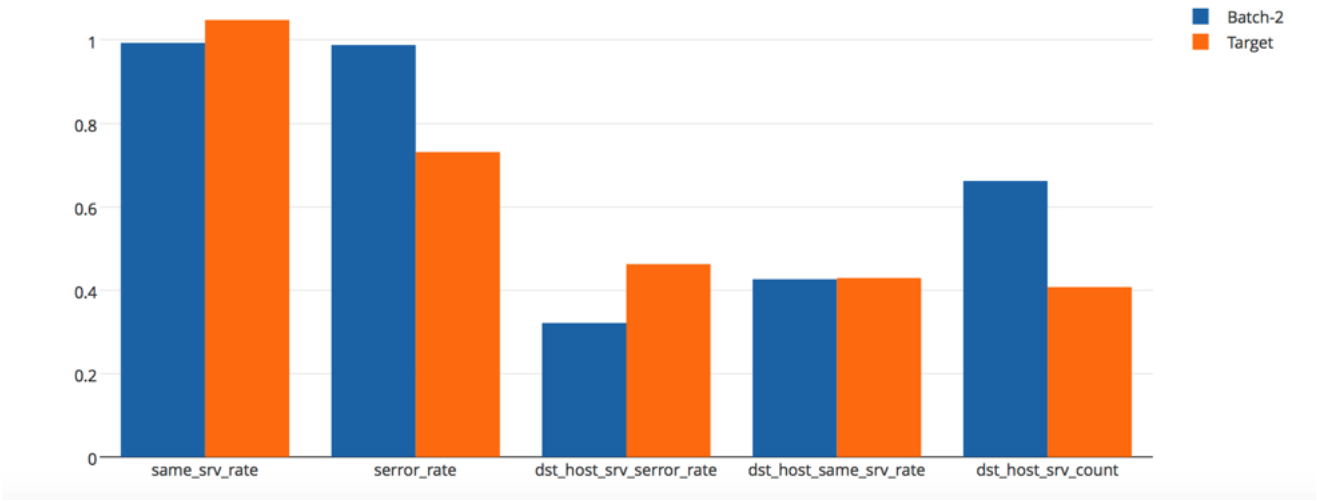


Figure 4.23: Batch-2 v/s Target(full dataset)

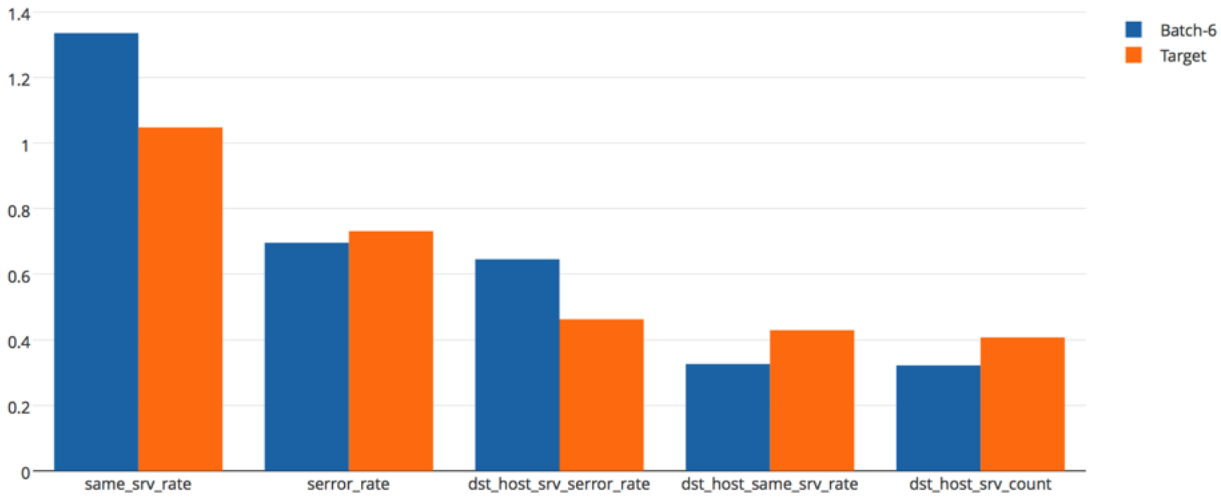


Figure 4.24: Batch-6 v/s Target(full dataset).

Chapter 5

Conclusion & Future Works

In this thesis, we presented a two-phase approach for feature selection. In the first phase a batch based Minimum Redundancy and Maximum Relevance (mRMR) algorithm is used with "correlation coefficient" and "mutual information" as statistical measure of similarity. This phase helped in improving the classification performance by removing redundant and unimportant features. In the second phase, we presented a stream based tree-based feature selection method that allowed dynamic generation and selection of features, while taking advantage of the different feature classes and the fact that they are of different sizes and have different fraction of good features. Experimental results showed that this phase was computationally less expensive than comparable "batch" methods that do not take advantage of the feature classes and expect all features to be known in advance. The k-nearest neighbors' algorithm (k-NN) classifier (linear and nonlinear), Decision-tree J-48, Support Vector Machines(SVM)[31] were used to evaluate the classification accuracy of our approach.

We believe that the selection algorithms we suggested in this thesis can work well on many problems, but it is important to understand that any selection algorithm is based on some assumptions. If these assumptions are violated the algorithm can fail. On the other hand, if a stronger assumption holds, another algorithm that assumes this stronger assumption might outperform the first one. For example, a method that ranks individual features by assigning a score to each feature independently

assumes that complex dependency on sets of features does not exist or is negligible. This assumption narrows the selection hypothesis space, and therefore allows for generalization using fewer instances. Thus, if this assumption is true, we would expect such a ranking to work better than methods that do not assume this, i.e. methods that consider subsets and are able to reveal complex dependencies (as these methods look in a larger hypothesis space). However, we cannot expect such rankers to work well when this independency assumption is not true.

We have also reviewed feature selection and explained the basic concept of different feature selection methods: filter, wrapper and hybrid model. We reviewed two filter based feature ranking techniques and one streaming based feature ranking technique. They are information gain, gain ratio, mutual information evaluation. We examined classification models that are built using various classification techniques such as Decision-tree J48, k-nearest neighbor, support vector machine. We took a brief review of the evaluation criteria used to evaluate the classification models. We have also introduced methods for feature ranking technique that can help build stable and robust classification models.

Future work will involve experiments on the datasets from different domains. The mRMR feature selection and tree-based feature selection algorithm will be tested on more datasets with different backgrounds. The difference in performance and accuracy of different feature selection approaches will be evaluated. Statistical analysis tests can be extended to different tests. ANOVA tests will be performed on individual fold values for each classifier.

At present our thesis has mainly concentrated on filter based feature ranking techniques. In the future, we would like to explore different approaches such as embedded feature selection techniques and its applicability to our streaming approach.

Bibliography

[1] Aijun An and Nick Cercone. "Discretization of continuous attributes for learning classification rules". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 1999, pp. 509–514.

[2] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. "Pearson correlation coefficient". In: *Noise reduction in speech processing*. Springer, 2009, pp. 1–4.

[3] Christopher M Bishop. "Pattern recognition". In: *Machine Learning* 128 (2006), pp. 1–58.

[4] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]". In: *IEEE Transactions on Neural Networks* 20.3 (2009), pp. 542–542.

[5] KDD Cup. "Dataset". In: *available at the following website http://kdd. ics. uci. edu/databases/kddcup99/kddcup99. html* 72 (1999).

[6] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[7] Chris Ding and Hanchuan Peng. "Minimum redundancy feature selection from microarray gene expression data". In: *Journal of bioinformatics and computational biology* 3.02 (2005), pp. 185–205.

[8] Pedro Domingos. *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Basic Books, 2015.

[9] Usama Fayyad and Keki Irani. "Multi-interval discretization of continuous-valued attributes for classification learning". In: (1993).

[10] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin, 2001.

[11] Isabelle Guyon and André Elisseeff. "An introduction to variable and feature selection". In: *Journal of machine learning research* 3.Mar (2003), pp. 1157–1182.

[12] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "Unsupervised learning". In: *The elements of statistical learning*. Springer, 2009, pp. 485–585.

[13] Alan Julian Izenman. "Linear discriminant analysis". In: *Modern multivariate statistical techniques*. Springer, 2013, pp. 237–280.

[14] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[15] Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia. *Learning spark: lightning-fast big data analysis*. " O'Reilly Media, Inc.", 2015.

[16] John T Kent. “Information gain and a general measure of correlation”. In: *Biometrika* 70.1 (1983), pp. 163–173.

[17] Kenji Kira and Larry A Rendell. “A practical approach to feature selection”. In: *Proceedings of the ninth international workshop on Machine learning*. 1992, pp. 249–256.

[18] Kenji Kira and Larry A Rendell. “The feature selection problem: Traditional methods and a new algorithm”. In: *AAAI*. Vol. 2. 1992, pp. 129–134.

[19] Ron Kohavi and George H John. “Wrappers for feature subset selection”. In: *Artificial intelligence* 97.1-2 (1997), pp. 273–324.

[20] Ron Kohavi and Dan Sommerfield. “Feature Subset Selection Using the Wrapper Method: Overfitting and Dynamic Search Space Topology.” In: *KDD*. 1995, pp. 192–197.

[21] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. “Mllib: Machine learning in apache spark”. In: *Journal of Machine Learning Research* 17.34 (2016), pp. 1–7.

[22] Patrenahalli M. Narendra and Keinosuke Fukunaga. “A branch and bound algorithm for feature subset selection”. In: *IEEE Transactions on Computers* 26.9 (1977), pp. 917–922.

[23] Amir Navot. “On the role of feature selection in machine learning”. PhD thesis. Hebrew University, 2006.

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. “Scikit-learn: Machine learning in Python”. In: *Journal of Machine Learning Research* 12.Oct (2011), pp. 2825–2830.

[25] J. Ross Quinlan. “Induction of decision trees”. In: *Machine learning* 1.1 (1986), pp. 81–106.

[26] Sergio Ramırez-Gallego, Salvador Garcıa, Héctor Mouriño-Talın, David Martınez-Rego, Verónica Bolón-Canedo, Amparo Alonso-Betanzos, José Manuel Benıtez, and Francisco Herrera. “Data discretization: taxonomy and big data challenge”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 6.1 (2016), pp. 5–21.

[27] Alfréd Rényi et al. “On measures of entropy and information”. In: *Proceedings of the fourth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 1961, pp. 547–561.

[28] James G Shanahan and Laing Dai. “Large scale distributed data science using apache spark”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2015, pp. 2323–2324.

[29] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. “The hadoop distributed file system”. In: *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*. IEEE. 2010, pp. 1–10.

[30] Manas Srivastava, C Sabarinathan, Rishi Sankineni, and TM Manoj. "Mining Of Big Data Using Map-Reduce Theorem". In: *IOSR Journals (IOSR Journal of Computer Engineering)* 1.17 (), pp. 49–55.

[31] Johan AK Suykens and Joos Vandewalle. "Least squares support vector machine classifiers". In: *Neural processing letters* 9.3 (1999), pp. 293–300.

[32] Joshua B Tenenbaum, Vin De Silva, and John C Langford. "A global geometric framework for nonlinear dimensionality reduction". In: *science* 290.5500 (2000), pp. 2319–2323.

[33] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing". In: *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association. 2012, pp. 2–2.

[34] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. "Spark: Cluster Computing with Working Sets." In: *HotCloud* 10.10-10 (2010), p. 95.

[35] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. "Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters." In: *HotCloud* 12 (2012), pp. 10–10.

[36] Xiaojin Zhu. "Semi-supervised learning literature survey". In: (2005).