A Survey of Feature Selection Approaches for Scalable Machine Learning

Master Thesis

by

Steffi Melinda

Submitted to the Faculty IV, Electrical Engineering and Computer Science,
Database Systems and Information Management Group
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

as part of the ERASMUS MUNDUS programme IT4BI
at the
TECHNISCHE UNIVERSITÄT BERLIN

July 31, 2016

Thesis Advisor:
Christoph Boden, M.Sc.

Thesis Supervisor:
Prof. Dr. Volker Markl

A Survey of Feature Selection Approaches for Scalable Machine Learning

by

Steffi Melinda

Submitted to the Department of Computer Science and Electrical Engineering
on July 31, 2016, in partial fulfillment of the requirements for the
degree of Master of Science in Computer Science
as part of the ERASMUS MUNDUS programme IT4BI

Abstract

Feature selection could affect the quality of machine learning model. Various feature selection algorithms were proposed to choose a subset of features effectively. We conducted an analysis of feature selection approaches in terms of their scalability and performance in the distributed machine learning context to tackle large scale data. Our literature study showed that most feature selection algorithms have quadratic complexity and potentially not scalable. We implemented an existing parallel feature selection approach based on variance preservation on Apache Spark and tested it on the classification problem using the internet advertisements and Dorothea dataset from UCI repository to verify its applicability on higher dimensional data and different distributed setting. The result showed that the algorithm was slightly scalable in Spark yet incurring high memory consumption.

Thesis Advisor: Christoph Boden
Thesis Supervisor: Prof. Dr. Volker Markl

Eine Übersicht über die Ansätze der Merkmalsauswahl für Skalierbares Maschinelles Lernen

von

Steffi Melinda

Eingereicht bei der. Fakultät IV für Elektrotechnik und Informatik, 31. Juli 2016,
Arbeit Eingereicht zur Erlangung des Grades Master of Science (M.Sc) im Studiengang
Informatik im Rahmen des Programms Erasmus Mundus IT4BI

Zusammenfassung

Die Feature Selection (Merkmalauswahl) könnte die Qualität des maschinellen Lernmodell beeinflussen. Verschiedene Merkmalsauswahlalgorithmen wurden effektiv eine Untergruppe von Funktionen zur Auswahl vorgeschlagen. Wir analysierten eine Analyse der Merkmalsauswahl Ansätze hinsichtlich ihrer Skalierbarkeit und Leistung im Zusammenhang mit verteilten maschinelles Lernen in groÃ§em Umfang Daten zu bewältigen. Unsere Literaturstudie zeigte, dass die meisten Merkmalsauswahlalgorithmen quadratische Komplexität haben und möglicherweise nicht skalierbar sind. Wir führten eine bestehenden Ansatz zur parallelen Merkmalsauswahl basiert zur Varianz Erhaltung von Apache Spark. Getestet wurde mittels der Internet-Werbung und Dorothea-Datensatz von der UCI repository zur Überprüfung der Anwendbarkeit auf höherdimensionalen Daten und verschiedene verteilte Einstellung. Das Ergebnis zeigte, dass der Algorithmus in Spark leicht skalierbar ist, aber des hohen Speicherplatzverbrauch verbraucht.

Thesis Advisor: Christoph Boden
Thesis Supervisor: Prof. Dr. Volker Markl

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Berlin, July 31, 2016

Steffi Melinda

To my first teacher in life, John and Lisa.

Acknowledgments

I would like to reward my sincere gratitude to my advisor, Christoph Boden, for the opportunity to work together in this study, for your valuable guidance, constructive feedback, and patience. Your broad knowledge in many research areas has become one of my motivation to finish this thesis. I hope this study could be beneficial for many people.

Furthermore, I would like to express my great appreciation to Prof. Dr. Volker Markl and Dr. Ralf Kutsche for the enjoyable experience in the DIMA Group at the TU Berlin and the continuous support throughout the year.

My great appreciation also goes to board members of IT4BI program, Erasmus Mundus and the European Union for making this amazing experience possible and granting me the honour to be a part of this program.

And special thanks to all IT4BI classmates and friends: Diana Matar, Diana Bogantes, Ale, Alejandro, Biplob, Truong, Guven, Stacy, Christina, Phuc, Ghena, Visar, Srdjan, Kate, Daria, Erica, and Eric. Thank you for the friendship, great teamwork, and fun times during the past two years in this study.

I would like to express my deepest gratitude to my beloved family for their continuous prayer, care and encouragement, for showing me the value of hard work, and for making me the way I am now. Last but not least, I would also like to thank my dearest friends and Willy Halim Dinata, for the love, great support and encouragement throughout this study. Thank you all for always believing in me.

Contents

Chapter 1

Introduction and Motivation

As the advancements of technology continuously grow, the amount of generated data, as well as the amount of data to be processed, increases at an unprecedented scale [44]. The process to analyze, make sense and gain insights from such extent of data is widely-known as big data analytics [6]. Real world applications of big data analytics often involve high-dimensional data with large number of features and/or instances. Different types of data (variety) also complicates the analytics process.

Machine learning

Machine learning is commonly used to gain insights or solve prediction task upon those data. Various real-world applications are genomics, biometric recognition, text mining, text classification, etc;

The use of all available features does not guarantee the best accuracy and running time of the machine learning task. Having more features could incur greater computational cost and potentially cause overfitting. To avoid overfitting, many algorithms use the Occam's Razor bias [29] to build a simple model that still achieves an acceptable level of performance on the training data. Using this bias, an algorithm often prefers smaller number of predictive features, with proper combination, that are fully predictive of the class label [37]. Too much irrelevant/redundant information or noisy and unreliable data could make the learning process more difficult. Having a subset of features could potentially produce better accuracy and take less time.

Feature selection

Feature selection is an important technique in data preprocessing for machine learning problems. The basic task of feature selection is to reduce the number of features by removing irrelevant or redundant features and noisy data before model construction. Simplifying learning models is useful to make it more understandable, to provide less learning time (cost-effective) and to enhance generalization by reducing overfitting. The ultimate goal of feature selection is to speed up a machine learning process and potentially improve machine learning model performance, such as predictive accuracy and result comprehensibility [64].

Current technology, which enables real-time analytics to allow faster and more responsive decision-making, produces a strong need to process and analyze huge datasets in a real-time manner. The complexity of an algorithm becomes the key concern to choose a certain feature selection algorithm. Certainly, algorithms with linear complexity is an advantage to achieve scalability. However, in some cases we still need to use a certain algorithm, even though the complexity is not linear, for example: to achieve maximum accuracy or to handle specific machine learning task.

Studies compared top-performing machine learning algorithms, such as logistic regression, Random Forests and SVMs [11]. However, the problem of feasible and efficient computation of feature evaluation remains as current issues, especially when dealing with large scale problems. An example study proposed a parallel large-scale feature selection approach for logistics regression [89]. The approach reaches scalability by parallelizing simultaneously over both features and records, which allows quick evaluation of billions of potential features even for very large data sets.

Parallel systems

Parallel data processing systems, such as Apache Hadoop[1], Apache Spark[2], and Apache Flink[3], provides the cost-efficient technology capable of storing and analyz-

[1]Link: http://hadoop.apache.org
[2]Link: http://spark.apache.org
[3]Link: http://flink.apache.org

ing huge datasets with support for arbitrary data types and complex analysis tasks. They used the principles of parallel databases to exploit parallelism in clusters of commodity hardware computers and offer programming models for custom data analysis tasks. MapReduce [20] as the programming model underlying Hadoop, was criticized for being a bad choice for complex analysis tasks since many tasks do not map naturally to MapReduce. Spark and Flink address this by generalizing MapReduce to a more expressive programming model, called PACT [3], adding support for complex data flows, iterations, and joins.

This thesis focuses on parallel feature selection algorithms for machine learning problems which involve large scale data. We focus on classification problem as the machine learning problem. We analyze the scalability of existing feature selection algorithms designed for parallel settings.

This chapter introduces the thesis and its purpose concerning machine learning. Section 1.1 describes the objectives of this thesis. Section 1.2 describes the outline of this thesis.

1.1 Thesis Objectives

This thesis aims to provide insights over existing feature selection algorithms in distributed machine learning context, especially to tackle large scale data. It begins by listing existing feature selection algorithms and analyze the complexity and performance of each algorithm. Then, we discuss, if there is any, existing proposed parallel approach of each algorithm. An example of feasible feature selection algorithm is then selected. An experimental evaluation of selected feature selection approach aims to test its feasibility when handling large-scale data in practice.

The literature study aims to deliver a summary of how feature selection algorithms contribute to the performance in solving the feature selection problem. We use the classification of feature selection algorithms proposed in [64] and update the list of the algorithms with the most up-to-date improvements of each algorithm. For each of the category, the study thoroughly describes how each algorithm works and analyzes its

scalability and performance from various experiments conducted by other researchers. Then, the study lists down any room for improvement or open problems left for each category that possibly leads to better performance when scaling up the data sets. The description of the improvement on the scalability includes the possibility of optimization or the feasibility of parallelizing the feature selection approaches.

The thesis also conducts an experiment to verify the applicability of the selected approaches in a distributed context. It involves a complete pipeline of training, modeling and evaluating the features to examine the quality of the approximate machine learning model using the selected feature selection approach. Selected comparable approaches are implemented in Apache Spark, to allow parallelizing the computation over the training sets and potential features. We apply the implementations to two high-dimensional datasets: the internet ads dataset and Dorothea (drug discovery) dataset, taken from UCI repository[4]. The objective of the experiment is to evaluate the modeling performance (accuracy) and the scalability when running in a distributed context. Additionally, the experimental evaluation is also useful to identify benefits and drawbacks of the algorithm.

1.2 Thesis Overview

The thesis is organized in five chapters. Chapter 2 provides the background study, which explains the main concepts and common categorization of feature selection. Chapter 3 contains the survey of existing feature selection algorithms, including their scalability analysis. Chapter 4 explains an example of the implementation of a selected algorithm to assess the scalability. Chapter 5 explains the experiment settings, such as the chosen dataset, the classification model; and the evaluation over the model after the applying feature selection implementation. Chapter 6 concludes the thesis by listing the advantages and drawbacks of feature selection algorithms, and also the potential area of future works.

[4]Link: http://archive.ics.uci.edu/ml

Chapter 2

Feature Selection Concepts

We could find feature selection in many areas of data mining and machine learning, such as classification, clustering, association rules, and regression. We focus on classification problem as the machine learning model. Feature selection algorithms aim to produce a subset of features that could improve the performance of the classifier that follows the feature selection process. In the case of the classifier for prediction problem, feature selection could provide faster and more cost-effective predictors.

A paper by Liu [64] described the feature selection process in a simple way. In general, feature selection process firstly selects a subset of original features, then measures the optimality of a subset by using certain evaluation criterion. After selecting a feature selection criterion, finding the subset of useful features becomes the challenge of the feature selection process. Directly evaluating all the subsets of features (2^N) for a given data becomes an NP-hard problem as the number of features grows. Therefore, a suboptimal procedure must be used to remove redundant features with tractable computations.

It becomes even more challenging to apply feature selection on high-dimensional data or big data. A common approach to easily conquer big data problems is to use parallel processing. A recent study by Singh [89] proposed a parallel approach to handling large-scale feature selection specific for logistic regression problem. Other than that, several other parallel feature selection approaches also exist. Up to the moment of writing this thesis, the study of advantages and drawbacks of those feature

selection algorithms are still limited.

This chapter explains the key steps and the characteristics of the feature selection in Section 2.1 and Section 2.2, respectively. Then we explain the commonly known categories of feature selection algorithms (FSA) in Section 2.3.

2.1 Key Steps of Feature Selection

Liu's work in [64] summarizes four basic steps in feature selection process, as depicted in Figure 2-1. Subset generation step produces candidate feature subsets for evaluation. Each candidate subset is evaluated and compared with the previous best one, according to a certain evaluation criterion. If the new subset is better than the previously found one, it replaces the previous best subset. The process of subset generation and evaluation is repeated until a given stopping criterion is reached [64]. The selected best subset usually needs to be validated by prior knowledge or different tests [64]. We explain the detail of the four key steps below.

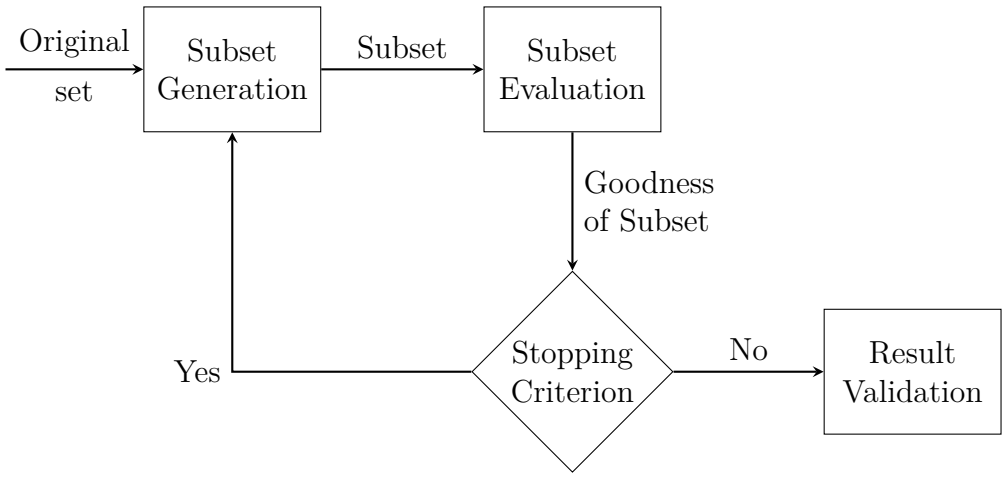


Figure 2-1: Four steps of feature selection (Source: [64])

2.1.1 Subset generation

It is a search problem that we can see as a heuristic search, which produces candidate feature subsets for evaluation. Each state in the search space specifies a candidate

subset for subset evaluation. The quality of searching depends on two basic issues.

First, search starting point determines the search direction. Forward selection starts from an empty set and incrementally add features. Backward selection starts with a full set and incrementally remove features. Bi-directional selection starts with both ends and add and remove features simultaneously. The quality of the subset produced on those search strategies heavily depends on the order of the features in the subset. In the case of getting trapped in the local minima, the subset is no longer the most optimal subset. Random selection tries to avoid that by starting with randomly selected subset [23].

Second, search strategy could affect the processing time. A data set with k features has 2^k candidate subsets. Thus, the search space is exponential and could become not scalable for a large number of k.

We discuss different search strategies in the following.

- **Complete or exponential search** is an optimal search which guarantees to find the best solution. An optimal search does not have to be exhaustive [67]. For example, if an evaluation measure is monotonic a Branch and Bound (B&B) [70] algorithm is optimal. A measure J is monotonic if for any two subsets S_1, S_2, and $S_1 \subseteq S_2$, then $J(S_1) \geq J(S_2)$. Different heuristic functions can reduce the search space without jeopardizing the chances of finding the optimal result. Hence, although the search space is $O(2^n)$, a smaller number of subsets are evaluated.

- **Sequential search** iteratively selects one among all the successors to the current state. Once the state is selected, it is not possible to go back. Although there is no explicit backtracking, the number of such steps are limited [67]. The complexity is determined by taking into account the number k of evaluated subsets in each state change. The cost of this search is, therefore, polynomial $O(n^{k+1})$. Consequently, these methods do not guarantee an optimal result, since the optimal solution could be in a region of the search space that is not visited. There are many variations to the greedy hill-climbing approach, such as

sequential forward selection, *sequential backward elimination*, and *bi-directional selection*. Algorithms with sequential search are simple to implement and fast in producing results [64].

- **Random search** starts with a randomly selected subset and proceeds with two different ways. One is to follow search to inject randomness into the classical sequential approaches, i.e. *random-start hill-climbing* and *simulated annealing*. Another one is to generate the next subset in a completely random manner (*Las Vegas* algorithm). Both approaches use randomness to escape local optima in the search space. The optimality of the selected subset depends on the resources available.

2.1.2 Subset evaluation

After the subset generation, each candidate subset is evaluated using an evaluation criterion. A certain criterion determines the goodness of a subset. An optimal subset selected using one criterion may not be optimal according to another criterion [64]. Evaluation criteria can be categorized based on their dependency on mining algorithms that will be applied to the selected subset.

- **Independent criteria**

  Typically algorithms of the filter model use these criteria. These criteria evaluate feature goodness or subset goodness by exploiting the intrinsic characteristics of the training data without involving any mining algorithm[64]. Some widely-used examples are distance measures, information measures, dependency measures, and consistency measures [64].

- **Dependent criteria**

  These criteria require predetermined mining algorithm and use the performance of the mining algorithm applied on the selected subset to determine which features should be selected. Therefore, these criteria are used in algorithms of the wrapper model and give better performance on suited predetermined mining

algorithm. The drawback of these criteria is that it tends to be more computationally expensive, and may not be suitable for other mining algorithms.

We explain more details on the subset evaluation approaches in Section 2.2.3.

2.1.3 Stopping criterion

The process of subset generation and evaluation is repeated until a given stopping criterion is reached [64]. Example of stopping criterions are:

- The search completes.
- Some given bound is reached. A given bound can be: a minimum number of features or a maximum number of iterations.
- Subsequent addition (or deletion) of any feature does not produce a better subset.
- A sufficiently good subset is selected (e.g. its classification error rate is less than the allowable error rate).

2.1.4 Result validation

One way to validate the selection result is to measure the result directly using prior knowledge about the data. However, in most cases, we usually do not have such prior knowledge. Hence, we have to monitor the change of mining performance with the change of features. For instance, applying the selected features to a classification model.

In general, all feature selection algorithms (FSA) follow this process flow. The key process in determining the quality of the selected features highly depends on the subset evaluation and stopping criterion.

2.2 Characteristics of Feature Selection

Several articles [67, 12] view that the characteristics of feature selection algorithms (FSA) can be described as a search problem with the following aspects:

- **Search Organization.** The general strategy used by FSA to explore the search space. The total number of the search space highly determine the memory and time complexity of the FSA.

- **Generation of Successors.** The mechanism used by FSA to generate a selection of features or subsets.

- **Evaluation Measure.** The function used by FSA to evaluate the successor candidates, which allows comparing different possibilities.

In the following sections, we discuss in detail each of those characteristics.

2.2.1 Search Organization

A search algorithm drives the feature selection process using a specific search strategy. In general, a search procedure examines only a part of the search space [67]. To visit a specific state, a search algorithm uses the information from the previously visited states and eventually heuristic knowledge about non-visited ones [67].

We consider three types of search: *exponential*, *sequential*, and *random*. We explained these search strategies in Section 2.1.1.

2.2.2 Generation of Successors

To generate a successor for each state,we could use five operators: *Forward*, *Backward*, *Compound*, *Weighting*, and *Random*. All of the operators, modify, in some way the weights w_i of the features x_i, with $w_i \in \mathbb{R}$ (in the case of the *weighting* operator), or $w_i \in \{0, 1\}$ (in the case of the rest of operators). In the following descriptions, it is assumed that the evaluation measure J is to be maximized [67].

- **Forward selection**

  This operator adds features to the current solution, among those that have not been selected yet. Starting with an empty set, this operator adds a new feature that makes J greater to the solution (set of selected features).

The stopping criterion can be when the number of selected features have reached the fixed parameter, or when the value J has not increased in the last j steps, or when it surpasses a prefixed value J_0. The cost of this operator is $O(n)$.

The main disadvantage is that it is not possible to have in consideration certain basic interaction among features. For example, if x_1, x_2 are such that $J(\{x_1, x_2\}) \gg J(\{x_1\}), J(\{x_2\})$, neither x_1 and x_2 could be selected, in spite of being very useful.

- **Backward elimination**

  This operator removes features to the current solution, among those that have not been removed yet. Starting with a full set of original features, this operator removes a feature that makes J greater from the solution (set of remaining features).

  The stopping criterion can be: when the number of remaining features has reached the fixed parameter, or when the value J has not increased in the last j steps, or when it falls below a prefixed value J_0. This operator remedies some problems, although there still will be many hidden interactions (in the sense of being unobtainable). The cost is $O(n)$, although, in practice, it demands more computation than forward selection [51].

We could generalize both operators (forward and backward) by selecting a subset of k elements in one step. The chosen subset should make J bigger. The cost of the operator for this generalized approach is $O(n^k)$.

- **Compound selection**

  This operator applies f consecutive forward steps and b consecutive backward steps. If $f > b$, the net result is a forward operator. Otherwise, it is a backward operator. Some approach decides whether to perform the forward or the backward steps depending on the respective values of J. This is done to discover new interactions among features.

These methods commonly have "backtracking mechanism" or other stopping conditions when $f = b$. For example, for $f = b = 1$, if x_i is added and x_j is removed, this could be undone in the following steps. A possible stopping criterion is $x_i = x_j$. In sequential FSA, the condition $f \neq b$ assures a maximum of n steps, with a total cost $O(n^{f+b+1})$.

- **Weighting**

  The search space in this operator is continuous. All of the features are present in the solution, and a successor state is a state with a different weighting [67]. It is typically done by iteratively sampling the available set of instances.

- **Random selection**

  This group includes those operators that can potentially generate any other state in a single step. The rest of operators can also have random components, but they are restricted to some criterion of "advance" in the number of features or in improving the measure J at each step.

2.2.3 Evaluation Measure

There are several approaches to evaluating the goodness of a feature subset. The relevance of a feature can be measured through a function and not by a characteristic of the feature itself. The range and scale of J are immaterial. What counts is that the relative values assigned to different subsets reflect the greater of lesser relevance to the objective function [67]. Some measures that assess class separability are the probabilistic, interclass distances, and the consistency. The interclass distance, consistency, entropy and estimations of the probability error may not require the explicit modeling of probability distributions [67].

- **Probability of error**

  These measures build a classifier that can correctly label instances generated by the same probability distribution by minimizing the (Bayesian) probability of error of the classifier. The probability of the error can be reached by the

integral of the maximum unconditional probability distribution of the instances and the posterior probability that certain instance belongs to a certain class label. Since the class-conditional densities are usually unknown, they can either be explicitly modeled (using parametric or non-parametric methods) or implicitly via the design of a classifier that builds the respective decision boundaries between the classes [21]. Some of these classifiers, like the one nearest-neighbor rule, have a direct relation to the probability of error.

The use of (an estimate of) this probability using the construction of a classifier, using a sample dataset, is the base of the *wrapper* methods [49]. The estimation probability may require the use of more elaborate methods than a simple hold-out procedure (cross-validation, bootstrapping) to yield a more reliable value [67].

- **Divergence**

  These measures compute a probabilistic distance or divergence among the class-conditional probability densities $p(\vec{x}|\omega_i)$, using the general formula:

$$J = \int f[p(\vec{x}|\omega_1), p(\vec{x}|\omega_2)]d\vec{x} \tag{2.1}$$

  A valid measure must have a function which value of J satisfies the following conditions:

  (a) $J \geq 0$

  (b) $J = 0$ only when the $p(\vec{x}|\omega_i)$ are equal

  (c) J is maximum when they are non-overlapping

  If the selected features are the good ones, the divergence among the conditional probabilities will be significant[67]. Poor features will lead to very similar probabilities. Hence small J. Examples of divergence measures are Chernoff, Bhattacharyya, Kullback-Liebler, Kolmogorov, Matusita, Patrick-Fisher [21].

- **Dependence**

  These measures quantify how strongly two features are associated with each

other. It means, by knowing the value of one, it is possible to predict the value of the other. The better a feature can predict a class, the better score it gets in the evaluation step. The classical example of this measure is correlation coefficient. A somewhat different approach is to estimate the divergence between the class-conditional and the unconditional densities [67].

- **Interclass distance**

  These measures assume that instances of a different class are distant among instances from other class when plotted as instance space. The most usual distance measure is Euclidean family. These measures do not require the modeling of any density function, but their relation to the probability of error can be very loose [67].

- **Information or Uncertainty**

  These measures compute how much information on certain class has been gained by computing the posterior probabilities with respect to its prior probability. If all the classes become roughly equally probable, then the information gain is minimal and the uncertainty (entropy is maximum)

- **Consistency**

  These measures aim to find the minimum subset of features that produces zero inconsistencies [5]. Inconsistency is defined as two instances having features of only those in the subset, and are equal, yet belongs to different classes. Mathematical formula of the inconsistency count of an instance $A \in S$ is defined as [58]:

$$IC_{X'}(A) = X'(A) - \max_k X'_k(A) \tag{2.2}$$

  where $X'(A)$ is the number of instances in S equal to A using only features in X' and $X'_k(A)$ is the number of instances in S of class k equal to A using only the features in X'. The inconsistency rate of a feature subset in a sample S is then:

$$IR(X') = \frac{\sum_{A \in S} IC_{X'}(A)}{|S|} \tag{2.3}$$

This is a monotonic measure, since $X_1 \subset X_2 \Rightarrow IR(X_1) \geq IR(X_2)$. A possible measure is then $J(X') = \frac{1}{IR(X')+1}$. This measure is in $[0, 1]$ and can be evaluated in $O(|S|)$ time using a hash table [58].

2.3 Feature Selection Categories

Feature selection algorithms can be broadly categorized based on its evaluation criteria into three categories: filter, wrapper, and embedded methods. Figure 2-2 depicts the process for each category.

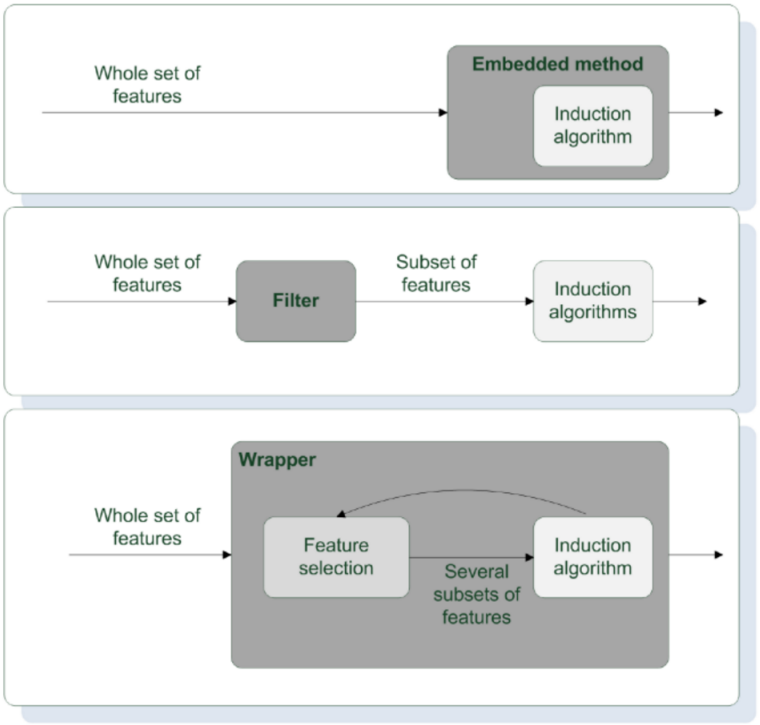


Figure 2-2: Feature selection categories (Source: [43])

1. **Filter methods**

   Filter methods rely on general characteristics of the data to evaluate and select features without involving any mining algorithm. Filter methods use a suitable criterion to score the features. Feature are ranked or ordered based on the evaluation score. This approach is also known as variable ranking. Variable ranking

can be either *univariate* or *multivariate*. In the univariate scheme, each feature is ranked independently from the feature space, while the multivariate scheme evaluates features in a batch way. The multivariate scheme is naturally capable of handling redundant features [2]. After the variable ranking, a threshold is used to remove features having evaluation score below the threshold. It is also known as variable elimination, which is performed before classification to filter less relevant features out.

Some filter methods generate all possible subsets and evaluate the score to determine whether the subsets are useful towards the machine learning problem. The property of feature relevance is important to determine the usefulness of the feature in discriminating different classes [12]. A feature can be considered as irrelevant if it is conditionally independent of the class labels [55]. It means that if a feature is to be relevant, it can be independent of the input data, but cannot be independent of the class labels. The feature that has no influence to the class labels can be discarded.

Some filter methods apply statistical measures to assign a scoring to each feature or subset. These methods often consider each feature independently or concerning the dependent variable. Further on these methods are explained in Section 3.1.

2. Wrapper methods

Wrapper methods prepare different combinations of features, evaluate them using a prediction model, assign a score based on model accuracy and compare with other models. This method considers the selection of a set of features as a search problem, which could use options of searching algorithms, such as best-first search as a methodical algorithm, the random hill-climbing algorithm as a stochastic algorithm, forward/backward passes as a heuristic algorithm, or any other algorithms.

Wrapper methods require a predetermined learning algorithm or a predictor as a black box and uses its performance as the objective function to evaluate subsets

[34]. Suboptimal subsets are found by employing search algorithms, which find a subset heuristically. Since predetermined learning algorithms are used to control the selection of feature subsets, the wrapper model tends to give a superior performance, as the feature subsets found better suited to the predetermined learning algorithm [34], but it is also more computationally expensive than the filter model.

3. **Embedded methods**

   This model attempts to take advantage of the filter and wrapper models by exploiting their different evaluation criteria in different search stages. Embedded methods perform feature selection when training the classifier and are usually specific to given learning machines [43].

   Embedded methods do not separate the learning from the feature selection part. Embedded methods include algorithms, which optimize a regularized risk function on two sets of parameters: the parameters of the learning machine and the parameters indicating which features are selected [43].

   The search for an optimal subset of features is built (embedded) into the classifier construction and can be seen as a search in the combined space of feature subsets and hypotheses. This allows capturing feature dependencies at a lower computational cost than wrapper methods.

Determining best feature selection methods depends on the nature of the data and the machine learning problem. [34] tried to develop a list of things to consider before determining the problem-solving steps feature selection.

Chapter 3

Feature Selection Algorithms

We based our analysis on existing feature selection algorithms on the categorization in the three-dimensional framework by [64]. The three dimensions are the evaluation criteria, the search strategy, and the data mining task (machine learning model) on which an algorithm is more suitable to perform. In our study, we focus on classification problem as the data mining task after the feature selection task.

In this chapter we explain several existing feature selection algorithms that represents characteristics of different categories of feature selection algorithms (FSA) in Section 3.1, Section 3.2, and 3.3. We also discuss some FSA that has been applied or proposed using parallel processing in Section 3.4.

We based our algorithms list on the categorization in Table 3.1, and we also added more up-to-date algorithms to the list.

Current proposed feature selection approaches usually combine several different categories to benefit the most for the feature selection problem. This approach is called hybrid approach. Categorizing an algorithm to certain category could be tricky. The important focus of this study is to understand each algorithm on its scalability and applicability towards the parallel processing.

Evaluation criteria			Search Strategies: Exponential	Search Strategies: Sequential	Search Strategies: Random
	Filter	*Distance*	*Forward:* BFF [104] / *Backward:* BOBRO [7]	*Forward:* Segen [85], SFG[a][21] / *Backward:* SBG[a][21]	*Weighting:* Relief [47], ReliefF [52], ReliefS [60]
		Information	*Backward:* MDLM [86]	*Backward:* Koller [51], FCBF [106]	
		Dependency	*Compound:* SFFS[a][75], SFBS[a][75], ASFFS[a][93]	*Forward:* CFS [36], DVMM [91], POE-ACC [69] / *Weighting:* PRESET [66]	
		Consistency	*Forward:* FOCUS/-2 [28], Schlimmer [83] / *Backward:* ABB [59], B&B [70], MIFES-1 [72]	*Weighting:* Winnow [57], SetCover [16]	*Random:* LVF [62], LVI [63], QBB[b][17]
	Wrapper	*Predictive Accuracy or Cluster Goodness*	*Forward:* Cardie/DTM [9], BS [23] / *Backward:* OBLIVION [54], AMB&B [27] / *Random:* FSBC [42], FSLC [41]	*Forward/Backward:* BDS [23], BSE [10], RACE [68] / *Forward:* WSFG [49], K2-AS [88] / *Backward:* SBS-SLASH [10], WSBG [49], RC [24] / *Compound:* PQSS [23], SS [68], Queiros [76]	*Random:* GA [99], LVW [61], RMHC-PF1 [90], SA [23], RVE [94] / *Forward/Backward:* RGSS [23]
	Embedded	*Filter+Wrapper*		*Forward:* BBHFS [15] / *Weighting:* Xing [102]	

[a] The evaluation measure can be anything, not only distance or dependency.

[b] This algorithm have both Exponential and Random search strategies and the generation of successors can be random and backward.

Table 3.1: Categorization of feature selection algorithms in a three-dimensional framework (Source: [64, 67])

3.1 Filter Methods

As explained in Section 2.3, filter method functions to evaluate and select features without involving any mining algorithm. Simple filter methods are done by computing the evaluation score for each feature, then perform variable/feature ranking. The commonly used methods are statistical measures, such as mutual information, Fisher's score, Laplacian score, Hilbert-Schmidt Independence Criterion (HSIC), Trace Ratio criterion and correlation coefficient scores.

Mutual Information

Due to its simple interpretation, information gain is one of the most popular feature selection methods [51], [106]. It is used to measure the dependence between features and labels. It is done by calculating the information gain between the i-th feature f_i and the class labels C as

$$IG(f_i, C) = H(f_i) - H(f_i|C) \tag{3.1}$$

where $H(f_i)$ is the entropy of f_i and $H(f_i|C)$ is the entropy of f_i after observing C.

$$\begin{aligned} H(f_i) &= -\sum_j p(x_j)\log_2(p(x_j)) \\ H(f_i|C) &= -\sum_k p(c_k)\sum_j p(x_j|c_k)\log_2(p(x_j|c_k)) \end{aligned} \tag{3.2}$$

In information gain, a feature is relevant if it has a high information gain. Features are selected in a univariate way. Therefore, information gain cannot handle redundant features. In [106], a fast filter method FCBF based on mutual information was proposed to identify relevant features as well as redundancy among relevant features and measure feature-class correlation and feature-feature correlation.

Fisher Score and Generalized Fisher Score

Fisher score finds a subset of features, such that in the data space spanned by the selected features, the distances between data points in different classes are as large

as possible, while the distances between data points in the same class are as small as possible. In brief, Fisher score computes the ratio of the between-class variance to the within-class variance. In particular, given the selected m features, the input data matrix $\mathbf{X} \in \mathbb{R}^{d\times n}$ reduces to $\mathbf{Z} \in \mathbb{R}^{m\times n}$. Then the Fisher score is computed as follows [33],

$$F(\mathbf{Z}) = tr\Big\{(\tilde{\mathbf{S}}_b)(\tilde{\mathbf{S}}_t + \gamma\mathbf{I})^{-1}\Big\} \tag{3·3}$$

where γ is a positive regularization parameter, $\tilde{\mathbf{S}}_b$ is called between-class scatter matrix and $\tilde{\mathbf{S}}_t$ is called total scatter matrix, which is defined as

$$\begin{aligned}\tilde{\mathbf{S}}_b &= \sum_{k=1}^{c} n_k(\tilde{\mu}_k - \tilde{\mu})(\tilde{\mu}_k - \tilde{\mu})^T \\ \tilde{\mathbf{S}}_t &= \sum_{i=1}^{n} (\mathbf{z}_i - \tilde{\mu})(\mathbf{z}_i - \tilde{\mu})^T,\end{aligned} \tag{3·4}$$

where $\tilde{\mu}_k$ and n_k are the mean vector and size of the k-th class respectively in the reduced data space, i.e., $\mathbf{Z}$, $\tilde{\mu} = \sum_{k=1}^{c} n_k\tilde{\mu}_k$ is the overall mean vector of the reduced data.

Since there are $\binom{d}{m}$ candidate $\mathbf{Z}$'s out of $\mathbf{X}$, the feature selection problem is a combinatorial optimization problem and very challenging. To reduce the difficulty, heuristic strategies are widely used is to compute a score for each feature independently according to the $\mathbf{F}$. Then the Fisher score of the r-th feature is computed as follows,

$$F_r = \frac{\sum_{i=1}^{c} n_i(\mu_i - \mu)^2}{\sum_{i=1}^{c} n_i\sigma_i^2} \tag{3·5}$$

where $\sigma_i^2 = \sum_{k=1}^{c} n_k(\sigma_k^i)^2$. After computation of the Fisher score for each feature, the top-m ranked features with large scores are selected.

Fisher score is used to rank variables in a classification problem where the covariance matrix is diagonal is optimum for Fisher's linear discriminant classifier [25]. Although many claimed that variable ranking is not optimal, Fisher's score is the most widely used method because of its computational and statistical stability [34]. Computationally, it only requires the computation of m scores and sorting the scores,

so it is efficient. Statistically, it introduces bias, but it may have considerably less variance, so it is robust against overfitting [38].

Fisher score evaluates features individually; therefore, it cannot handle feature redundancy. [33] proposed generalized Fisher score for feature selection to jointly selects features into subsets of features which maximize the lower bound of traditional Fisher score and solve the following problem:

$$\begin{aligned} \left\|\mathbf{W}^T diag(\mathbf{p})\mathbf{X} - \mathbf{G}\right\|_F^2 + \gamma \left\|\mathbf{W}\right\|_F^2, \\ s.t., \mathbf{p} \in \{0,1\}^m, \mathbf{p}^T\mathbf{1} = d, \end{aligned} \tag{3.6}$$

where $\mathbf{p}$ is the feature selection vector, d is the number of features to select, and $\mathbf{G}$ is a special label indicator matrix, as follows:

$$\mathbf{G}(i,j) = \begin{cases} \sqrt{\frac{n}{n_j}} - \sqrt{\frac{n_j}{n}} & \text{if } \mathbf{x}_i \in c_j, \\ -\sqrt{\frac{n_j}{n}} & \text{otherwise} \end{cases} \tag{3.7}$$

Generalized Fisher score is proposed along with an algorithm to solve the multiple kernel learning problem. In each inner iteration, we need to solve one multivariate ridge regression problem, which can be solved efficiently by LSQR (An algorithm for sparse linear equations and sparse least squares) and scales linearly in the number of training samples n. So it has convergence property, which is achieved through iteration over the number of classes and number of features. In summary, the total time complexity of the method is $O(T(cns + s\log m))$, where T is the number of iterations needed to converge, s is the average number of nonzero features among all the training samples, c is the number of classes.

The experiments, performed by the author, on benchmark data sets indicate that the accuracy of the generalized Fisher score outperforms the traditional Fisher score. Furthermore, the accuracy also outperforms Laplacian score, Trace ratio and Hilbert-Schmidt Independence Criterion (HSIC).

Laplacian Score

Laplacian score (L-score) is a feature scoring method for supervised feature selection problem. It determines the importance of a feature by the construction of a graph using the sample data points and identifies which features best represent the structure of the graph [103]. Laplacian score is computed to assess the power of locality preserving [39].

The algorithm starts by constructing a nearest neighbor graph G with m nodes, with m as the number of data points. An edge is placed between two nodes that are close to each other. If nodes i and j are connected, the edge weight between those data points is calculated as,

$$S_{ij} = e^{-\frac{\left\|x_i - x_j\right\|^2}{t}}, \tag{3.8}$$

where t is a suitable constant. Otherwise, $S_ij = 0$. The weight matrix S of the graph models the local structure of the data space [39]. After the construction of edge weights, for the k-th feature, define:

$$f_k = \left[f_{k1}, f_{k2}, ..., f_{km}\right]^T, D = diag\left(S\mathbf{1}\right), \mathbf{1} = \left[1, ...1\right]^T, L = D - S \tag{3.9}$$

where the matrix L is called graph Laplacian. Let $\tilde{\mathbf{f}}_k = \mathbf{f}_k - \frac{\mathbf{f}_k^T D\mathbf{1}}{\mathbf{1}^T D\mathbf{1}}\mathbf{1}$. Then the L-score of k-th feature, L_k, can be computed as:

$$L_k = \mathbf{f}_k - \frac{\tilde{\mathbf{f}}_k^T L\tilde{\mathbf{f}}_k}{\tilde{\mathbf{f}}_k^T D\tilde{\mathbf{f}}_k} \tag{3.10}$$

Then the feature will be sorted by their L-score and provide the basis for subsequent classification.

Minimum Redundancy Maximum Relevance (mRMR)

mRMR [103] stands for Minimum Redundancy Maximum Relevance. The idea of minimum redundancy is to select the feature such that they are mutually maximally dissimilar. The maximum relevance is to maximize total relevance. mRMR is also

a mutual-information-based method. It selects features according to the maximal statistical dependency criterion [73]. Since it is difficult to directly implementing the maximal dependency condition, mRMR is an approximation to maximizing the dependency between the joint distribution of the selected features and the classification variable [2].

Minimize redundancy for discrete features and continuous features are defined as [2], for discrete features in Equation 3.11 and for continuous features in Equation 3.12.

$$minW_I, \quad W_I = \frac{1}{|S|^2} \sum_{i,j \in S} I(i,j) \tag{3.11}$$

$$minW_c, \quad W_c = \frac{1}{|S|^2} \sum_{i,j \in S} |C(i,j)| \tag{3.12}$$

where $I(i,j)$ and $C(i,j)$ are mutual information and the correlation between f_i and f_j ,respectively. While *maximize relevance* for discrete features and continuous features are defined as [2], for discrete features in Equation 3.13 and for continuous features in Equation 3.14.

$$maxV_I, \quad V_I = \frac{1}{|S|^2} \sum_{i \in S} I(h,i) \tag{3.13}$$

$$maxV_c, \quad V_c = \frac{1}{|S|^2} \sum_{i} F(i,h) \tag{3.14}$$

where h is the target class and $F(i,h)$ is the F-statistic.

Summary

Information gain, Fisher score, Laplacian score, and mRMR merely functions as evaluation score to do features ranking. These algorithms requires at least quadratic time to read d number of features times m number of instances to compute the distance or relevance among each feature towards each data point. When the number of features d grows bigger, the time required for computing the score grows linearly according to the number of features.

The benefit of using feature ranking techniques is that determining the features

to be selected can be done only once. The algorithm then select the $Top - k$ number of features as the result feature subset. However, this technique can not actually guarantee if the quality of the subset collectively is better than the quality of the subset individually. Another important point is that the score for each features heavily depends on the data points used for the learning model. Whenever any change is introduced to the training sample, the feature ranks order could possibly change.

Concerning large dataset, if the number of features d and/or the number of data points m are huge, the algorithm requires all the information to fit in-memory. This becomes another limitation for this technique. This technique requires modification of the mathematical formula to compute the overall score per feature in a distributed setting.

3.1.1 Distance-based filter methods

Recall Table 3.1, examples of feature selection algorithms that use distance as the evaluation measures are best-first strategy for feature selection (BFF), Bobrowski's algorithm (Bobro), Segen's algorithm (Segen), sequential forward generation/selection (SFG or SFS), sequential backward generation/selection (SBG or SBS), Relief and its variations.

Best First strategy for feature selection (BFF)

BFF is an improvement of B&B algorithm. This is a heuristic search strategy that guarantees the globally best subset without exhaustive enumeration for any criterion that satisfies monotonicity [104]. The advantage is that the number of subsets evaluated by BFF is less than those needed by B&B algorithm. However, the complexity of the algorithm remains to be exponential as it utilizes graph and computes the path to select subsets.

Sequential Forward Generation/Selection (SFG or SFS)

SFG or SFS [21] starts with an empty set, iteratively adds one feature which gives the highest value for the objective function J, and then the evaluates the new subset. The measure is computed by taking into account those features already present in the solution. The process is repeated until the required number of features are added. SFG or SFS can produce an ordered list. The algorithm in 1 represents the generic computation for SFG and SBG (see below). This is called a naive SFS algorithm since the dependency between the features is not accounted for. The complexity of this algorithm is exponential.

Sequential Backward Generation/Selection (SBG or SBS)

SBG or SBS [21] is the backward counterpart of SFG algorithm. Instead of adding features to an initial subset, SBG or SBS removes features from a full set of original features. The process is repeated until the required number of features needed is achieved. The Algorithm 1 represents the generic computation for SFG (see above) and SBG. The complexity of this algorithm is quadratic.

Input : $S(X)$ - a sample S described by X, $|X| = n$
J - evaluation measure
Output: X' - solution found
1 $X' := \varnothing$ `// forward`
2 $X' := X$ `// backward`
3 **repeat**
4 $\quad x' := argmax\,\{J(S(X' \cup \{x\}))|x \in X \setminus X'\}$ `// forward`
5 $\quad x' := argmax\,\{J(S(X' \setminus \{x\}))|x \in X'\}$ `// backward`
6 $\quad X' := X' \cup \{x'\}$ `// forward`
7 $\quad X' := X' \setminus \{x'\}$ `// backward`
8 **until** *no improvement in* J *in last* j *steps* **or** $X' = X$ **or** $X' = \varnothing$;

Algorithm 1: SFG and SBG Algorithm

Relief

Relief algorithm selects relevant features using a statistical method, does not depend on heuristics, is accurate even if features interact, and is noise-tolerant [47]. For each

randomly selected instance, find nearest hit and miss to determine quality estimation. It can deal with nominal and numerical attributes, but it cannot deal with incomplete data and is limited to two-class problems. As seen in Algorithm 2, the complexity of this algorithm is $O(kN^2)$ (quadratic).

Input : p - sampling percentage
d - distance measure
$S(X)$ - a sample S described by X, $|X| = n$
Output: W - array of feature weights

```
1 initialize W[] to zero;
2 for p|S| times do
3 |   A := Random_Instance(S);
4 |   A_nh := Near-Hit(A,S);
5 |   A_nm := Near-Miss(A,S);
6 |   for i ∈ [1..n] do
7 |   |   W[i] := W[i] + d_i(A, A_nm) − d_i(A, A_nh);
8 |   end
9 end
```

Algorithm 2: Relief Algorithm

ReliefF

ReliefF algorithm [52] is an improvement of Relief algorithm. ReliefF tries to overcome Relief algorithm's limitation on handling only binary classes. ReliefF handles multiple classes and incomplete and noisy data. For each randomly selected instance, find the nearest hit from the same class. For neighbors that are not in the same class, compute the miss. Update quality estimation for each attribute based on those difference. As the k more similar instances are selected and their averages are computed, the complexity of the algorithm is improved to $O(kN)$. However, the optimal results are still not guaranteed. Besides, an experiment in [106] found that searching for nearest neighbors involves distance calculation, which makes ReliefF more time consuming than those using the calculation of symmetrical uncertainty values.

Relief with selective sampling (ReliefS)

ReliefS was proposed on 2004 [60] as the improvement of Relief algorithm, which creates *kd*-tree upon each bucket of features and adds random sampling. The algorithm in Algorithm 3 shows that ReliefS used either sample size m or bucket size t to control the *kd*-tree splitting process. With N number of instances, $t = N/m$. Therefore, if m is predetermined, the splitting process stops when it reaches the level where each bucket contains N/m or fewer instances. [60] also shows that ReliefS, in general, achieves better performance than ReliefF in terms of learning accuracy and hence selective sampling is an effective approach for active feature selection.

Input : t - bucket size
Set all weights $W[A_i] := 0.0$;
buildKDTree(t);
$m :=$ number of buckets;
for $j := 1$ *to* m **do**
    randomly select an instance X from $Bucket[j]$;
    find nearest hit H and nearest miss M;
    for $i := 1$ *to* k **do**
        $W[A_i] := W[A_i] -$ diff$(A_i, X, H)/m +$ diff $(A_i, X, M)/m$;
    end
end

Algorithm 3: Relief with selective sampling algorithm (ReliefS)

Summary

Distance-based filter algorithms which utilize exponential search strategy, like BFF, has exponential complexity. These algorithms are computationally expensive, in particular when dealing with large number of features. Algorithms with sequential and random search strategies, like SFG, SBG and Relief, have at least quadratic complexity. Moreover, variants of Relief algorithm, like ReliefS, offer an interesting improvement on the efficiency. However, the optimal results are still not guaranteed.

Concerning large dataset, if the number of features d and/or the number of data points m are huge, algorithms that need to compute the distance scores (hit or miss) between two features by considering the whole data points (instances) are not scal-

able, since not everything can fit into the memory. ReliefS is an example of feasible algorithm, in this case, as it could compute the weight of features by only considering a part of the existing instances or data points as buckets.

3.1.2 Information-based filter methods

Examples of FSA that uses information as the evaluation measures are minimum-description-length criterion model (MDLM), Koller's algorithm (Koller), and fast correlation-based filter (FCBF).

Koller

The main idea of Koller's algorithm is that a good subset of the features should present a class probability distribution as close as possible to the distribution obtained with the original set of features [51]. This algorithm starts with all the features and applies backward elimination. At each step, it removes the feature that minimizes the distance between the original and the new class probability distribution, which is measured through cross-entropy [98]. Features are then removed iteratively until the desired number of features is reached. Given the nature of the formulas involved, the method must operate on binary features, and thus may require additional transformations of the data [98].

Fast Correlation-Based Filter (FCBF)

Algorithm 4 shows the pseudo-code of FCBF algorithm. The algorithm applies 3 heuristics, as explained in [106]. *Heuristic 1*, a feature F_p that has already been determined to be a predominant feature can always be used to filter out other features that are ranked lower than F_p and have F_p as one of its redundant peers. *Heuristic 2 and 3*, the iteration starts from the first element in S'_{list} and continues as follows. For all the remaining features (from the one right next to F_p to the last one in S'_{list}), if F_p happens to be a redundant peer to a feature F_q, F_q will be removed from S'_{list}. After one round of filtering features based on F_p , the algorithm will take the currently

remaining feature right next to F_p as the new reference to repeat the filtering process. The algorithm stops until there is no more feature to be removed from S'_{list}."

```
   Input  : S(F_1, F_2, ..., F_N, C)   // a training dataset
          1 δ   // a predefined threshold
   Output: S_best   // an optimal subset
 2 for i = 1 to N do
 3     calculate SU_{i,c} for F_i;
 4     if SU_{i,c} ≥ δ then
 5         append F_i to S'_list;
 6     end
 7     order S'_list in descending SU_{i,c} value;
 8     F_p = getFirstElement(S'_list);
 9     repeat
10         F_q = getNextElement(S'_list, F_p);
11         if F_q <> NULL then
12             repeat
13                 F'_q = F_q;
14                 if SU_{p,q} ≥ SU_{q,c} then
15                     remove F_q from S'_list;
16                     F_q = getNextElement(S'_list, F'_q);
17                 else
18                     F_q = getNextElement(S'_list, F_q);
19                 end
20             until F_q == NULL;
21         end
22         F_p = getNextElement(S'_list, F_p);
23     until F_p == NULL;
24     S_best = S'_list;
25 end
```

Algorithm 4: FCBF Algorithm

The first part of FCBF algorithm has a linear time complexity in terms of the number of features N. In the second part, FCBF can remove a large number of features that are redundant peers to F_p in the current iteration. The best case could be that all of the remaining features following F_p in the ranked list will be removed; the worst case could be none of them. On average, we can assume that half of the remaining features will be removed in each iteration. Therefore, the time complexity for the second part is $O(N \log N)$ in terms of N. Since the calculation of SU for

a pair of features is linear in term of the number of instances M in a data set, the overall complexity of FCBF is $O(MN \log N)$.

An experiment in [106] shows that FCBF is practical for feature selection for classification of high dimensional data. It can efficiently achieve high degree of dimensionality reduction and enhance classification accuracy with predominant features.

Summary

Some information-based filter methods, like FCBF, have at least logarithmic complexity, at worst case quadratic complexity. This makes the algorithm to be more practical for feature selection for classification of high dimensional data.

3.1.3 Dependency-based filter methods

Recall Table 3.1, examples of FSA that uses dependency as the evaluation measures are sequential forward floating selection (SFFS), sequential backward floating selection (SFBS), adaptive sequential forward floating selection (ASFFS), correlation-based feature selection (CFS), discriminative variable memory Markov model (DVMM), combination of probability of error and average correlation coefficient (POE-ACC), and feature selection based on rough set theory (PRESET).

Sequential Forward Floating Selection (SFFS)

This algorithm [75] was proposed as an improvement of SFS algorithm. In each step, SFFS performs a forward step, which selects a feature and unconditionally adds it into the subset, and adds another backward step, which excludes one feature at a time from the subset obtained in the first step and evaluates the new subsets. Therefore, a feature is first unconditionally added and then features are removed as long as the generated subsets are the best among their respective size. The term floating comes from the fact that the algorithm has the characteristic of floating around a potentially good solution of the specified size (see Figure 3-1). The complexity of the algorithm remains to be exponential.

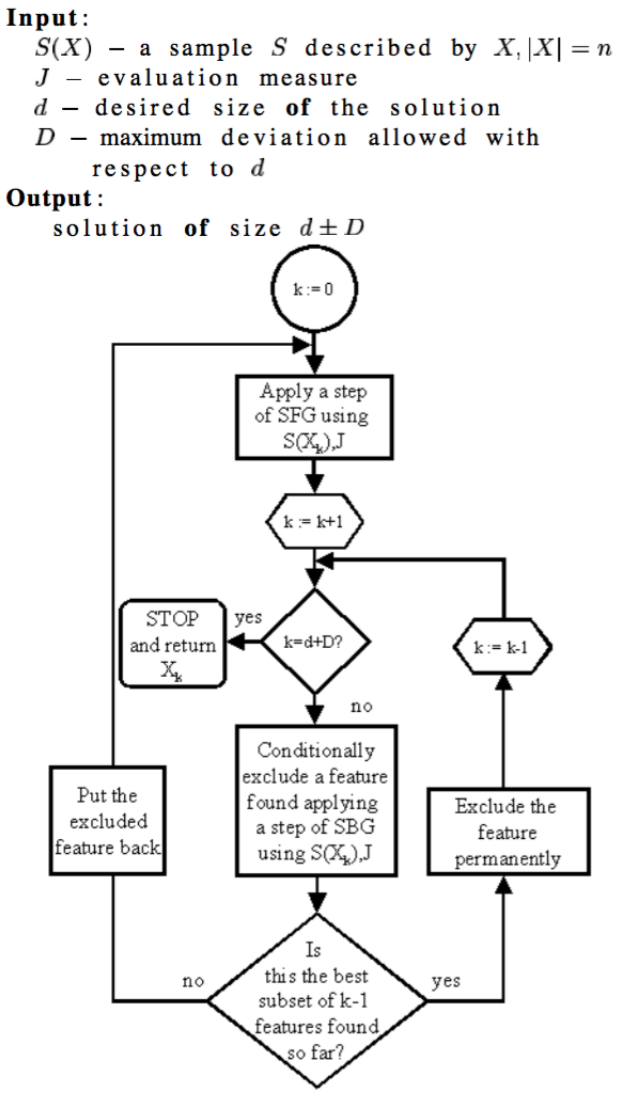


Figure 3-1: SFFS Algorithm (Source: [67])

Sequential Backward Floating Selection (SFBS)

This algorithm [75] was proposed as an improvement of SBS algorithm. Similar to SFFS algorithm, SFBS performs a backward step, which removes a feature from the full feature set and then adds another forward step, which adds one feature from the removed subset (which is the most significant feature) and evaluates the new subset. The complexity of the algorithm remains to be exponential.

Discriminative Variable Memory Markov Model (DVMM)

DVMM stands for discriminative feature selection via multiclass variable memory markov model [91]. This algorithm extend the original generative VMM modeling technique [82], which aims at building a model which hold only a minimal set of relevant suffixes. The feature selection scheme is based on maximizing conditional mutual information. The following explanation of DVMM comes from [91]: "For any subsequence s, the algorithm estimates the information between the next symbol in the sequence and each statistical source $c \in C$, given that subsequence (or suffix) s. DVMM uses this estimate as a new measure for pruning less discriminative features out of the model. The pruning criteria is, however, very different from the one used by the original generative VMM model. Many features may be important for good modeling of each source independently although they provide minor discrimination power. These features are pruned in the DVMM, resulting in a much more compact model which still attains high classification accuracy."

After beginning with a simultaneous ML training for all classes, DVMM selects features that maximize the same objective function. Figure 3-2 explains the pseudo-code for DVMM training algorithm. The author denotes $\hat{T}_s$ as the subtree spanned by s. All the nodes in $\hat{T}_s$ correspond to subsequences with the same suffix, s.

From the algorithm, we could see that DVMM requires quadratic time to span the suffix tree and to trace the span to prune features. Implementing subtree and pruning tree also could become tricky in distributed setting. We did not find any parallel version of this feature selection algorithm yet.

Summary

Some dependency-based filter methods, like DVMM, have at least quadratic complexity. Computing the dependency measure itself could be feasible in distributed setting. However, those algorithms that involve different data structures, like tree, could become tricky when it comes to tracing the path for each spanning tree and perform pruning.

Initialization and first step—tree growing:

Initialize $\hat{T}$ to include the empty suffix e, $\hat{P}(e) = 1$.

For $l = 1...L$

for every $s_l \in \Sigma^l$, where $s_l = \sigma_1\sigma_2 \cdots \sigma_l$, estimate $\hat{P}(s_l|c) = \prod_{i=1}^{l} \hat{P}(\sigma_i|\sigma_1 \cdots \sigma_{i-1}, c)$

if $\hat{P}(s_l|c) \geq \varepsilon_1$, for some $c \in C$, add s_l into $\hat{T}$.

Second step—pruning:

For all $s \in \hat{T}$, estimate $I_s = \sum_{c \in C} \hat{P}(c|s) \sum_{\sigma \in \Sigma} \hat{P}(\sigma|s, c) \log(\hat{P}(\sigma|s, c)/\hat{P}(\sigma|s))$.

For $l = L...1$

define $\hat{T}_l \equiv \Sigma^l \cap \hat{T}$

for every $s_l \in \hat{T}_l$,

let $\hat{T}_{s_l}$ be the subtree spanned by s_l

define $\bar{I}_{s_l} = \max_{s' \in \hat{T}_{s_l}} I_{s'}$

if $\bar{I}_{s_l} - I_{\text{suff}(s_l)} \leq \varepsilon_2$, prune s_l.

Figure 3-2: Pseudo-code for the DVMM algorithm (Source: [91])

3.1.4 Consistency-based filter methods

Recall Table 3.1, examples of FSA that uses dependency as the evaluation measures are FOCUS, Schlimmer's algorithm, branch and bound (B&B), automatic branch and bound (ABB), minimal set of conjunctive features (MIFES-1), Winnow's algorithm, SetCover, Las Vegas filter (LVF), Las Vegas incremental (LVI), and quick branch and bound (QBB).

FOCUS

The FOCUS [28] algorithm starts with an empty set and carries out breadth-first search until it finds a minimal subset that predicts pure classes. In detail, it starts by evaluating each singleton feature set, then each set of two features and so forth. It stops whenever a sufficiently consistent solution is found. We describe the basic algorithm in Algorithm 5. The same author tried to improve the algorithm into FOCUS-2 [4] algorithm, which introduces the concept of conflict between positive and negative examples to prune the search.

The complexity of FOCUS algorithm is at least quadratic, as it iterates on the number of features n and the feature subsets to evaluate the consistency measure.

Input :
$S(X)$ - a sample S described by X, $|X| = n$
J - evaluation measure (consistency)
J_0 - minimum allowed value of J
Output:
X' - solution found

```
for i ∈ [1..n] do
    for X' ⊂ X, with |X'| = i do
        if J(S(X')) ≥ J_0 then
            stop
        end
    end
end
```

Algorithm 5: FOCUS algorithm

Branch and Bound (B&B)

B&B [70] is originally proposed as an optimal search algorithm. It is a depth-first-search tree-based feature elimination strategy, by evaluating the criterion at each level and sort them, continued with pruning. Given a threshold β (specified by the user), the search stops at each node the evaluation of which is lower than β, so that unnecessary branches are pruned. This algorithm can be used with any evaluation measure that is monotonic.

The advantage of this algorithm is: it guarantees that the selected subset yields the globally best value of any criterion that satisfies monotonicity. However, as it performs the full complete search, the number of subsets evaluated can be huge as the number of features grows. The complexity of the algorithm is exponential.

Automatic Branch and Bound (ABB)

ABB [59] is a variant of B&B in which the threshold is automatically set. We describe this algorithm in Algorithm 6. This algorithm can be used with any evaluation measure that is monotonic.

The complexity of ABB is $O(n \log n)$ as it involves loop over the number of features and then recursively appending features whenever the feature is legitimate and it improves the evaluation measue. Implementing recursive algorithm in parallel setting

Input :
$S(X)$ - a sample S described by X, $|X| = n$
J - evaluation measure (monotonic)
Output:
L - all equivalent solutions found

```
1  Procedure ABB(S(X) : sample; var L' : list of set)
2      for x in X do
3          enqueue(Q,X \ {x}) // remove a feature at a time
4      end
5      while not empty(Q) do
6          X' := dequeue(Q) // X' is legitimate if it is not a subset of a
           pruned state
7          if legitimate(X') and J(S(X')) ≥ J_0 then
8              L' := append(L',X')
9              ABB (S(X'),L')
10         end
11     end
12 end
13 Q := ∅   // Queue of pending states
14 L' := [X]   // List of solutions
15 J_0 := J(S(X))   // Minimum allowed value of J
16 ABB(S(X,L'))     // List of solutions
17 k := smallest size of a subset in L'
18 L := set of elements of L' of size k
```

Algorithm 6: ABB algorithm

would be computationally expensive. What could be done is to split the task into a parallel task whenever they need perform smaller task or calling recursive method. However, the trick is to know when to stop parallelizing.

SetCover

The problem of finding the smallest set of consistent features is equivalent to covering each pair of examples that have different class labels with some feature on which they have different values [18]. SetCover [16] uses that observation to find the smallest set of features that is "consistent" in describing classes. The algorithm is based on Johnson's [45] algorithm for set covering. This algorithm outputs a consistent feature set with size of a log factor of the best possible $O(M \log P)$ and runs in a polynomial time.

Previously, variants of SetCover have been used for learning conjunctions of boolean features. Consistency criterion are defined as: a feature set S is consistent if for any pair of instances with different class labels, there is a feature in S that takes different values. Therefore, including a feature f in S "takes care of" all example pairs with different class labels on which f has different values. Once all pairs are "taken care of" the resulting set S is consistent [18].

The work by [18] shows the advantages of SetCover. SetCover produces close-to-optimal subsets and it is fast and deterministic. This algorithm works well for data where features are rather independent of each other. However, this algorithm may have problems on features which have interdependencies as it selects the best feature in each iteration based on the number of instance-pairs covered. Therefore, a new solution is needed to avoid the problems of exhaustive and heuristic search.

Las Vegas Filter (LVF)

Las Vegas Filter [62] repeatedly generates random feature subsets and then computes their evaluation measure. It was originally implemented by using the consistency of the sample as the evaluation measure. We describe the algorithm in Algorithm 7.

From the algorithm, the complexity of decision problems that have LVF requires

Input :
max - the maximum number of iterations
J - evaluation measure
$S(X)$ - a sample S described by X, $|X| = n$

Output:
L - all equivalent solutions found

```
1  L := []    // L stores equally good sets
2  Best := X    // Initialize best solution
3  J_0 := J(S(X))    // minimum allowed value of J
4  for i ≤ max do
5  |  X' := RandomSubset(Best)      // |X'| ≤ |Best|
6  |  if J(S(X')) ≥ J_0 then
7  |  |  if |X'| < |Best| then
8  |  |  |  Best := X'
9  |  |  |  L := [X']    // L is reinitialized
10 |  |  end
11 |  |  else
12 |  |  |  if |X'| = |Best| then
13 |  |  |  |  L := append(L, X')
14 |  |  |  end
15 |  |  end
16 |  end
17 end
```

Algorithm 7: Las Vegas Filter (LVF) Algorithm

randomized polynomial time to give the correct answer, as it depends on the number of max parameter.

Las Vegas Incremental (LVI)

Las Vegas Incremental [63] is based on the fact that it is not necessary to use the whole sample S to evaluate the measure. If LVF (refer to 18) finds a sufficiently good solution in the first portion S_0 of the whole sample, then LVI halts. Otherwise, the set of samples in the other portion $S \setminus S_0$ making the first portion S_0 inconsistent is added to the first portion S_0. This new portion S_0 is handed over to LVF, and the process is iterated.

Intuitively, the portion can be neither too small nor too big. If it is too small, after the first iteration, many inconsistencies will be found and added to the current subsample, which will hence be very similar to the whole sample S. If it is too big, the result will produce only small amount of computational savings. The authors suggest $p = 10\%$ or a value proportional to the number of features [67]. The evaluation measure could be anything. We describe the algorithm in Algorithm 8.

From the algorithm, the complexity of the FSA again is randomized polynomial time as it involves LVF algorithm to give the correct answer.

Quick Branch and Bound (QBB)

QBB [17] is a hybrid algorithm composed of LVF and ABB. The basic idea consists of using LVF to find good starting points for ABB[67]. It is expected that ABB can explore the remaining search space efficiently. We describe this algorithm in Algorithm 9.

The authors [17] reported that QBB could be, in general, more efficient than LVF, FOCUS, and ABB in terms of average execution time and selected solution.

Summary

Some consistency-based filter methods, like FOCUS, involves iteration on the number of features and the feature subsets to evaluate the consistency measure. Those algo-

Input :
max - the maximum number of iterations
J - evaluation measure
$S(X)$ - a sample S described by X, $|X| = n$
p - initial percentage
Output:
X' - solution found

```
1  S_0 := portion(S, p)    // Initial portion
2  S_1 := S \ S_0    // Test set
3  J_0 := J(S(X))    // Minimum allowed value of J
4  while true do
5  |  X' := LVF(max, J, S_0(X))
6  |  if J(S(X')) ≥ J_0 then
7  |  |  stop
8  |  else
9  |  |  C := { elements in S_1 with low contribution to J using X'}
10 |  |  S_0 := S_0 ∪ C
11 |  |  S_1 := S_1 \ C
12 |  end
13 end
```

Algorithm 8: Las Vegas Incremental (LVI) Algorithm

Input :
$S(X)$ - a sample S described by X, $|X| = n$
J - monotonic evaluation measure
max - the maximum number of iterations
Output:
L - all equivalent solutions found

```
1  L_ABB := [ ]
2  L_LVF := LVF(max,J,S(X))
3  for X' ∈ L_LVF do
4  |  L_ABB := concat(L_ABB, ABB(S(X'), J))
5  end
6  k := smallest size of a subset in L_ABB
7  L := set of elements of L_ABB of size k
```

Algorithm 9: QBB algorithm

rithms require at least quadratic time to complete feature selection task. Although ABB, an improvement of Branch and Bound (B&B), is faster than FOCUS, ABB also takes a long time for a moderate number of irrelevant features. Therefore, for data with large numbers of features, ABB would not terminate in realistic time [18]. Heuristic or probabilistic search is believed to be faster, although these algorithms may not guarantee optimal subsets. As an example, SetCover uses heuristic search, so that it is fast and deterministic. However, it may face problems with data having highly interdependent features [18]. Another example of probabilistic search, LVF does not have the problem with highly interdependent features, but is slow to converge in later stages. The experiment done by [18] shows that LVF can reduce the feature subset size very fast in the beginning but then it slows down in reducing features. Another improvement of ABB, QBB is a welcome modification as it captures the best of LVF and ABB. It is reasonably fast, robust and can handle features with high interdependency [18].

With the interesting complexity, QBB seems to be a good candidate to be parallelized. The remaining challenge is to implement recursive method in parallel setting, which would be computationally expensive. Splitting the task into a parallel task whenever they need perform smaller task or calling recursive method could be one way to solve it. However, we need to know when to stop parallelizing.

3.2 Wrapper Methods

The disadvantage of the filter algorithms is that it totally ignores the effects of the selected feature subset on the performance of the induction algorithm [50]. The wrapper algorithms offers a simple and powerful way to address the problem of variable selection, regardless of the chosen learning model. So, this method considers the learning model as a perfect black box. Wrapper methods use the prediction performance of a given learning model to assess the relative usefulness of subsets of variables.

Given a predefined classifier, a typical wrapper model will perform the following steps:

- *Step 1*: searching a subset of features,
- *Step 2*: evaluating the selected subset of features by the performance of the classifier,
- *Step 3*: repeating Step 1 and Step 2 until the feature set with the highest estimated value is reached.
- *Step 4:* choosing the feature set with the highest estimated value as the final set to learn the classifier.
- *Step 5:* evaluating the resulting classifier on an independent testing set that is not used during the training process.

A general framework for wrapper methods of feature selection for classification is shown in Figure 3-3.

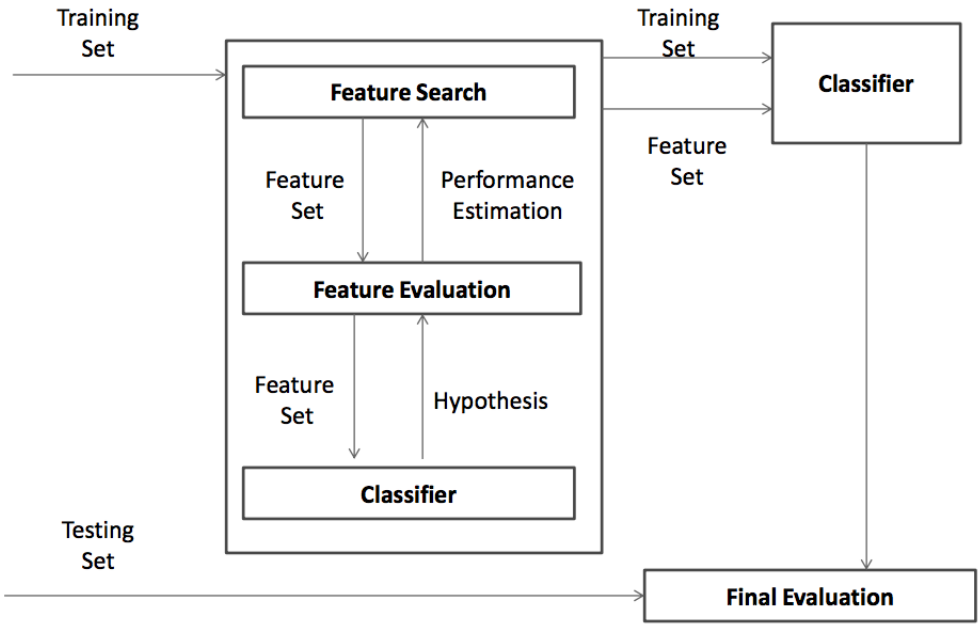


Figure 3-3: A General Framework for Wrapper Methods of Feature Selection for Classification (Source: [95]).

To create or choose a wrapper method, we need to consider three major components [34]:

(a) Feature selection search - How to search the space of all possible variable subsets?

(b) Feature evaluation - How to assess the prediction performance of a learning model to guide the search and stop it?

(c) Induction algorithm - Which predictor to use?

Wrapper methods are often criticized as they seem to be a "brute force" method requiring massive computation. The size of the search space for m features is $O(2^m)$. Performing an exhaustive search is only feasible when the number of variables m is not too large. An exhaustive search problem is known to be NP-Hard [34]. By using efficient search strategies, wrapper methods does not need to sacrifice prediction performance. In some cases: simple sequential search strategies may decrease the potential for the overfitting problem, as illustrated in [79].

A wide range of search strategies can be used, including hill-climbing, best-first, branch-and-bound (B&B), simulated annealing, and genetic algorithms [50]. The Hill-climbing strategy starts with the current set, moves to the subset to get the highest accuracy, and stops when no improvement is found over the current set [95]. The best-first strategy selects the most promising set that has not already been expanded and is a more robust method than hill-climbing [50]. Greedy search strategies can be robust against overfitting. There are two types of the search strategy, as discussed in Section 2.2.2: forward selection and backward elimination. The search finds a feature subset which gives highest evaluation by using heuristic function to guide or direct the search process. The actual accuracy of the classifier is unknown, so wrapper methods use accuracy estimation as both the heuristic function and the valuation function in the feature evaluation phase [95].

The performance assessments in wrapper methods are usually done using a validation set or by cross-validation. Popular predictors include decision trees, naive Bayes, least-square linear predictors, and support vector machines [34].

Although wrapper models obtain better predictive accuracy estimates than filter models [50] [58], these models are very computationally expensive compared to filter models. Since the evaluation process involves maximizing the quality for certain classifier, the selected subset of features is inevitably biased to the predefined classifier.

3.2.1 Exponential Wrapper Algorithms

Recall Table 3.1, examples of Wrapper FSA that uses exponential search are Cardie's algorithm or decision tree-based model (DTM), beam search (BS), OBLIVION, sproximate monotonic branch and bound (AMB&B), feature selection for box classifier (FSBC), and feature selection for linear classifier (FSLC).

Beam Search (BS)

Beam search [23] is a type of best-first search which uses a bounded queue to limit the scope of the search. BS orders the queue from best to worst, with the best states placed at the front of the queue. BS operates by taking the best state and extending the search from that state. Each newly visited state is placed in its proper sorted position in the queue.

If there is no limit on the size of the queue, BS can become an exhaustive search [23]. However, by placing a limit on the queue size, the states with low values, which have high error rates, will eventually fall off the end of the queue, effectively pruning the search at those states. It is interesting to note that, when the limit on the size of the queue is one, beam search is equivalent to SFS and has a complexity of $O(n^2)$.

Oblivious Decision Trees and Abstract Cases (OBLIVION)

OBLIVION [54] tries to handle both irrelevant features and attribute interactions without causing expensive search. OBLIVION uses the concept of *oblivious* tree introduced in [28]. Oblivious decision tree is a tree in which all nodes at the same level test the same attribute [54].

OBLIVION starts by constructing a full oblivious tree that incorporates all potentially relevant attributes. This algorithm estimates the accuracy of the tree in the entire training set using a traditional technique, like n-way cross validation. This algorithm then removes each attribute in turn, estimates the accuracy of the resulting tree in each case, and selects the most accurate [54]. If the tree does not create more error compared to the initial one, OBLIVION replaces the initial tree with it

and continues the process. On each step, the algorithm prunes each of the remaining features, selects the best, and generates a new tree with one fewer attribute. The algorithm stops when the accuracy of the best pruned tree is less than the accuracy of the current one. The complexity of OBLIVION is polynomial in the number of features.

Summary

Most of the exponential search-based wrapper feature selection algorithms have quadratic complexity. Involving the complexity of the classification would trivially incur bigger cost.

One of the challenge in implementing exponential search-based wrapper FSA on distributed setting lies on the mechanism to send the search state on each step to every workers (beam search) and tracing and pruning the tree on different workers (OBLIVION). Letting workers communicate through master each time would not be a preferred solution since it would be as slow as having one machine. So the selection step should be performed in one machine instead.

3.2.2 Sequential Wrapper Algorithms

Recall Table 3.1, examples of Wrapper FSA that uses sequential search are bi-directional search (BDS), backward stepwise elimination (BSE), racing the cross-validation errors (RACE), wrapper sequential forward generation (WSFG), wrapper sequential backward generation (WSBG), K2 algorithm for attribute selection (K2-AS), stepwise backward search-slash (SBS-SLASH), RC, (p,q) sequential selection (PQSS), schemata search (SS), and Queiros's algorithm.

Bi-directional search (BDS)

Bi-directional search [23] is a search heuristic that uses both forward and backward sequential techniques. SFS is performed from the empty set and SBS is performed from the full set. Both searches converge in the middle of the search space. At each

step of SFS, the algorithm avoids to add features that have been removed by SBS. Conversely, SBS avoids to remove features that have been added by SFS. If either of these conditions is broken, both algorithms will never converge to the same feature subset [23].

Since both algorithm SFS and SBS share the same $O(n^2)$ complexity, so does the complexity of BDS, although BDS stop once both searches meet in the middle [23].

Wrapper Sequential Forward Generation (WSFG)

The algorithm WSFG [49] is a wrapper version of SFG [21]. WSFG uses the accuracy of an external inducer as the evaluation measure. Since SFG has quadratic complexity, WSFG would have the same complexity, but would incure bigger cost since WSFG has to consider the time to perform classification on each step.

Wrapper Sequential Backward Generation (WSBG)

The algorithm WSBG [49] is a wrapper version of SBG [21]. WSBG uses the accuracy of an external inducer as the evaluation measure. Since SBG has quadratic complexity, WSBG would have the same complexity, but would incure bigger cost since WSBG has to consider the time to perform classification on each step.

(p,q) Sequential Selection (PQSS)

(p, q) Sequential Selection [23] is a generalization of the forward and backward sequential selection techniques. p and q are both positive integers with p referring to the number of features to add at each step and q the number of features to remove. The generalized sequential selection technique tries to compensate for the weaknesses of SFS and SBS by allowing some degree of backtracking, by removing features that have been wrongly chosen by SFS and reconsidering features that have been unwisely discarded [23]. This algorithm can be seen as the wrapper version of floating search selection algorithms (SFFS and SBFS).

The complexity of PQSS is quadratic, similar to SFFS and SFBS. Considering classification time on each step, the actual time of PQSS would be greater.

Summary

Most of the sequential search-based wrapper feature selection algorithms have quadratic complexity. Involving the complexity of the classification would trivially incur bigger cost.

One of the challenge in implementing sequential search-based wrapper FSA on distributed setting lies on the mechanism to send the search state on each step to every workers. Letting workers communicate through master each time would not be a preferred solution since it would be as slow as having one machine. So the selection step should be performed in one machine instead.

3.2.3 Random Wrapper Algorithms

Recall Table 3.1, examples of Wrapper FSA that uses random search are genetic algorithm or genetic search (GA/GS), Las Vegas wrapper (LVW), random mutation hill-climbing and prototype feature selection (RMHC-PF1), simulated annealing (SA), randomized variable elimination (RVE), and random generation plus sequential selection (RGSS).

Genetic Algorithm/Search (GA/GS)

Genetic algorithm [105] treats the whole feature set as the population and generates subsets, performs crossover, mutate the result subset and create the subset as new population. The population is initialized to contain binary vectors representing feature subsets. GA randomly generates feature vectors to initialize the population [23]. The binary vectors represents weather certain feature is included or excluded from the final feature subset. The algorithm iterates through the all the generated combinations (*generations*). A "mating" set is created at the beginning of each generation, which consists of numerous copies of the better feature subsets from the population. The number of copies made of a particular subset is proportional to the worth of the subset. The crossover algorithm needs to be defined to combine various members of the mating set into a new vector (*offspring*). The final step during each generation

is the creation of a new population from the old population and the offspring, by selecting the best members from both the old population and the offspring.

For each generation, an offspring set is created which is then merged with the old population to form the new population. Therefore, only the new subsets need to be evaluated during every new generation [23]. In addition, the offspring set size is not greater than the population size. Given that g is the number of generations and p is the population size (the number of features), the order is $O(gp)$.

Las Vegas Wrapper (LVW)

LVW (Las Vegas Wrapper) [61] is a wrapper algorithm that uses LVF to generate candidate subsets and accuracy of an inducer as the evaluation measure. Since LVF has randomized polynomial complexity, LVW behaves the same way.

Random Generation Plus Sequential Selection (RGSS)

RGSS [23] uses randomness as part of the feature selection process. In particular, it injects randomness into the classical forward and backward sequential selection techniques. Algorithm 10 briefly explains the process.

```
1 for i < number of features do
2 |   randomly generate a feature subset
3 |   perform SFS from this subset
4 |   perform SBS from this subset
5 end
```

Algorithm 10: RGSS Algorithm

RGSS can be seen as breaking the search space into partitions. Inside of these partitions, either SFS or SBS will obtain the locally optimal solution, the best solution in that partition. In this way, a randomly generated subset could be potentially the globally optimal solution that is found by either SFS or SBS [23].

The number of generations should be large for problems of great complexity to increase the probability that a randomly generated subset will fall in a partition containing the optimal or a near optimal solution.

The complexity of this algorithm is determined by observing that both SFS and SBS are performed during each generation. This means that the number of states examined per generation is $O(n^2)$. This if g is the number of generations, the complexity of the algorithm is $O(gn)$.

Summary

As a summary, LVW behaves similarly as LVF, which has randomized polynomial complexity. Taking classifier complexity into account, the polynomial complexity would incur bigger cost (time). RGSS and GA is slightly better by giving quadratic complexity. However, involving the classifier complexity would again incure bigger cost.

Generating populations in GA and RGSS can be implemented as mapping function in MapReduce paradigm. The process of evaluating the result can then be done distributedly in workers. Then master could then gather local optimal solutions to evaluate and select the best solution as the global solution.

3.3 Embedded Methods

Looking back to previously explained methods, filter methods select features that are independent of the classifier and avoid the cross validation step as in the typical wrapper methods. Therefore, filter methods are computationally more efficient than wrappers. However, filter methods do not take into account the biases of the classifiers. For example, the relevance measure of Relief would not be appropriate as feature subset selectors for Naive-Bayes because in many cases the performance of Naive-Bayes improves with the removal of relevant features [34]. Wrapper methods use a predefined classifier to evaluate the quality of features, to avoid biases of the classifier. However, the process involves running the classifier repeatedly to assess the quality of selected subsets of features, which is very computationally expensive [95]. Embedded methods embeds the feature selection as part of the training process of the classifier. This makes embedded methods have the advantages of wrapper methods,

as they include the interaction with the classification model, and filter methods, as they are far less computationally intensive than wrapper methods [64].In addition, embedded method does not split the data into training and validation set. Therefore, they also make better use of the available data.

There are three families of embedded methods: nested subset methods, direct objective optimization and models with a built-in mechanism for feature selection, such as ID3 [77] and C4.5 [78]. We discuss further on the first two families.

3.3.1 Nested Subset Methods

These methods guide the search process by estimating changes in the objective function J value incurred by making moves in feature subset space. Methods that are combined with greedy search strategies (forward selection and backward elimination) yield nested feature subsets. Let us call s the number of features selected at a given step and $J(s)$ the value of the objective function of the trained learning model using such a feature subset. The embedded methods predicts the change in the objective function by using [34]:

- **Finite difference calculation:** Finite difference calculation (method 1) is done by computing the difference between $J(s)$ and $J(s+1)$ or $J(s$âĹŠ1) for the features that are candidates for addition or removal.

  Without retraining new models for each candidate variable, exact differences can be computed efficiently. An example of these method is the Gram-Schmidt orthogonolization procedure, for linear least-square model. This procedure permits the performance of forward variable selection by adding at each step the variable that most decreases the mean-squared-error [34]. Another example is computing approximations of the difference for other algorithms like kernel methods [84].

- **Quadratic approximation of the cost function:** This method was originally proposed to prune weights in neural networks [13]. It can be used for backward elimination, by pruning the features weights w_i. This procedure use

a second-order Taylor expansion of J. At the optimum of J, the first-order term can be neglected, yielding for feature i to the subset $DJ_i = (1/2)\frac{\partial^2 J}{\partial w_i^2}(Dw_i)^2$. The change in weight $Dw_i = w_i$ corresponds to removing feature i. The "optimum brain damage" (OBD) procedure (method 2) is mentioned in [81].

- **Sensitivity of the objective function calculation:** The absolute value or the square of the derivative of J with respect to x_i (or with respect to w_i) is used. Applications of this procedure to a linear model with a cross-entropy objective function are presented in [74]. An interesting variant of the sensitivity analysis method is obtained by replacing the objective function by the leave-one-out cross-validation error [34].

3.3.2 Direct Objective Optimization

These methods formalize the objective function of feature selection and find algorithms to optimize it. Generally, the objective function consists of two terms that compete with each other:

(a) the goodness-of-fit (to be maximized)
(b) the number of variables (to be minimized).

This approach bears similarity with two-part objective functions consisting of a goodness-of-fit term and a regularization term, particularly when the effect of the regularization term is to "shrink" parameter space.

Consider the linear classifier $\mathbf{w}$ in which classification of $\mathbf{Y}$ can be based on a linear combination of features X, such as SVM and logistic regression. In regularization methods, feature selection is achieved by estimating $\mathbf{w}$ with property tuned penalties. The learned classifier $\mathbf{w}$ may have coefficients exactly equal to zero. Each coefficient of $\mathbf{w}$ corresponds to one feature such as $\mathbf{w}_i$ for f_i. Only features with non-zero coefficients in $\mathbf{w}$ will be used in the classifier.

$$\hat{\mathbf{w}} = \min_{\mathbf{w}} c(\mathbf{w}, \mathbf{X}) + \alpha penalty(\mathbf{w}) \tag{3.15}$$

where $c(\cdot)$ is the classification objective function, $penalty(\mathbf{w})$ is a regularization term, and α is the regularization parameter controlling the trade-off between the $c(\cdot)$ and the penalty.

Commonly used embedded methods are Least Absolute Shrinkage and Selection Operator (LASSO), Adaptive LASSO, and Elastic Net regularization.

Least Absolute Shrinkage and Selection Operator (LASSO)

LASSO regularization [97] is based on l_1-norm of the coefficient of $\mathbf{w}$ and defined as

$$penalty(\mathbf{w}) = \sum_{i=1}^{m} |\mathbf{w}_i| \tag{3.16}$$

The feature selection based on LASSO makes use the important property of l_1 regularization, that it can generate an estimation of $\mathbf{w}$ with exact zero coefficients [97]. This means that, there are zero entities in $\mathbf{w}$, which denotes that the corresponding features are eliminated during the classifier learning process [95].

Adaptive LASSO

The LASSO feature selection is consistent if the underlying model satisfies a non-trivial condition, which may not be satisfied in practice [109]. Meanwhile, the LASSO shrinkage produces biased estimates for the large coefficients, thus, it could be sub-optimal in terms of estimation risk [26]. The adaptive LASSO [111] is proposed to improve the performance of as

$$penalty(\mathbf{w}) = \sum_{i=1}^{m} \frac{1}{\mathbf{b}_i} |\mathbf{w}_i| \tag{3.17}$$

The only difference with LASSO is that adaptive LASSO employs a weighted adjustment $\mathbf{b}_i$ for each coefficient $\mathbf{w}_i$.

Elastic Net Regularization

When there are a few features that are highly correlated, LASSO tends to select only one of the correlated features [95]. Elastic net regularization [112] is proposed to

handle features with high correlations, which cannot be handled properly by LASSO.

$$penalty(\mathbf{w}) = \sum_{i=1}^{n} |\mathbf{w}_i|^{\gamma} + (\sum_{i=1}^{n} \left|\mathbf{w}_i^2\right|)^{\lambda} \tag{3.18}$$

with $0 < \gamma \leq 1$ and $\lambda \geq 1$. The elastic net is a mixture of bridge regularization [40] [48] with different values of γ.

Summary

LASSO, adaptive LASSO, and ElasticNet iterates over the number of features m to generate the penalty of the coefficient $\mathbf{w}$. These algorithms functions as regularization and have linear complexity for one step of classification learning process.

Overall, the existing feature selection algorithms are designed for traditional centralized computing environments and most have the quadratic or polynomial complexity. Among the three methods, embedded methods are claimed to have the fastest time. However, they are bonded to the classification model used. Filter methods are on the other hand flexible, but also do not take into account the biases of the classifiers. In terms of scalability, embedded methods and filter methods are more scalable compared to wrapper methods.

Converting existing algorithms to parallel settings requires the consideration over each detail of the algorithm, whether the data structure involved are compatible to distributed setting and whether the computation can be divided into smaller tasks and computed independently by the workers. Converting filter algorithms to a distributed setting is the most feasible choice as most of the filter algorithms works by scoring the features before choosing or filtering out the irrelevant or redundant features. Converting embedded methods to a distributed setting can also be feasible as implementing classifier on distributed setting.

3.4 Parallel Algorithms

As the previous section stated, the existing feature selection algorithms are designed for traditional centralised computing environments. To enhance their efficiency and scalability, most algorithms cannot readily utilise advanced distributed computing frameworks, such as MPI [92], Microsoft's Dryad[1], Google's MapReduce [20] or even Apache Spark and Apache Flink. Converting a centralised-designed algorithms to parallel algorithms requires compatibility. Even if the algorithm is compatible, the analysis over the potential speedup (comparison of running time in a parallel environment to the centralised environment) is the key consideration to migrate the feature selection process.

Parallel Feature Selection from Random Subsets

The study [30] proposed a feature selection algorithm based on support vector machine training time, that can be run on all available processors in parallel. This feature selection method is divided into two stages.

The first stage randomly generates a number of feature sets of fixed size. Then it uses only the features found in these sets to do a 10-fold cross validation in order to determine their suitability to classify the data. The sets of features are then sorted by a given criteria, i.e. the training time or the number of support vectors generated. The best of these randomly generated sets are selected for the second stage of the algorithm.

In the second stage, a union set of the features found in the selected sets is created. The classifier is trained using the newly created feature sets, and tested against a previously unseen test set to see how well it performs. The random feature sets created are completely independent from one another and all of them are evaluated during the same step in the algorithm. For this reason, every single random set created can be evaluated in parallel and thus the time to finish this feature selection method is greatly reduced.

[1]Link: http://www.microsoft.com/en-us/research/project/dryad

This feature selection approach is advantageous if new features need to be selected during data acquisition [43].

Parallel Feature Selection for Logistic Regression

The study [89] developed a wrapper method that employs forward feature selection and logistic regression. In every step, new features are added to an existing model. In order to make it feasible, the evaluation of a single feature is sped up by a heuristic SFO (Single Feature Optimization) that provides approximate models on the effect of adding new features to the model.

The method starts with an empty set of features, and then it selects features that improve performance iteratively until that improvement stops. At that moment, the features that have not been included are discarded. For evaluating new features, three methods are considered: full forward feature selection, single feature optimisation, and grafting. All three methods provide fast and greedy approaches to feature selection problem, but scale poorly with the number of features, becoming computationally expensive as the size of the data grows. For this reason, they are parallelised using the MapReduce framework, distributing the cost over many machines, and being able to scale to much larger feature sets [43].

The author carried out experiments over several UCI datasets to show that the algorithm achieves a speedup of over 15 on a 20-node cluster. This proves that the feature selection algorithm become scalable after the use of MapReduce framework.

Variance Preservation Parallel Feature Selection

Zhao et al. [110] proposed a parallel feature selection algorithm based on variance preservation to tackle both supervised and unsupervised problem. More explanation of this algorithm is explained in Section 4.2.

Chapter 4

Implementation

After surveying various existing algorithms in Chapter 3, we choose to implement the parallel feature selection based on variance preservation algorithm [110]. We implemented the supervised feature selection algorithm to support classification problem. The implementation uses Java programming language and Apache Spark as the parallel processing system. The existence of MLlib library as the scalable machine learning library provides more simplified implementation over the classification model.

As the implementation requires knowledge on parallel data systems, we discuss existing parallel systems in Section 4.1. We then describe the reason for choosing to implement the selected feature selection algorithm in Section 4.2. The implementation of feature selection involves the full machine learning pipeline. In this study, we focus on classification problem as the machine learning model. The implementation of the feature selection algorithm and the classifier is described in Section 4.3 and Section 4.4 respectively.

4.1 Parallel data processing system

Parallel processing divides large problems into smaller problems and carries out the execution or computation of the processes simultaneously (at the same time) [1]. We briefly explain different forms of parallelism [22]. Figure 4-1 illustrates the difference.

- *Data parallelism* splits the data into multiple parts (called *partitions*) and run

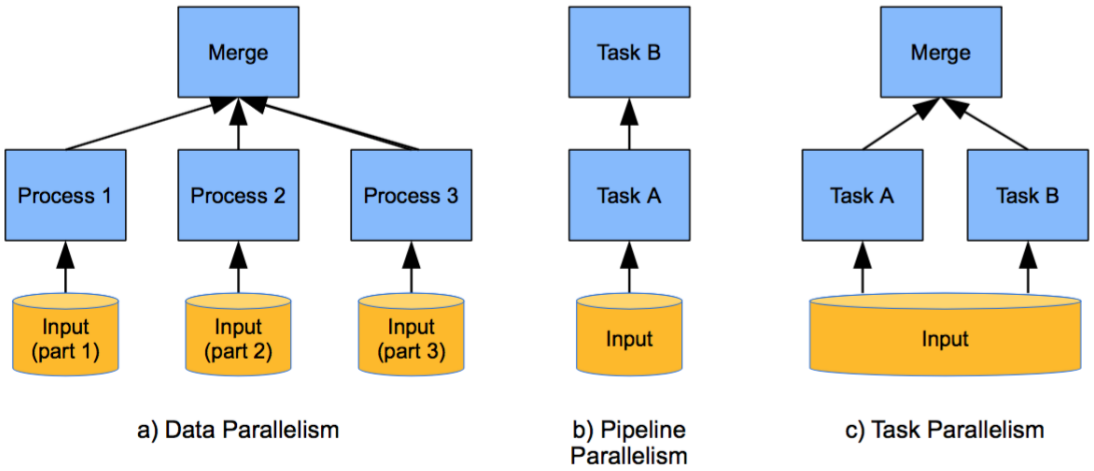


Figure 4-1: Three forms of parallelism (Source: [35])

the same task independently on different partitions of data at the same time.

- *Pipeline parallelism* directly forwards an output of one task to the subsequent task without waiting for the first task to completely finish the computation for the whole input. This could be thought as a stream of data flowing through a pipeline of concurrently running tasks.

- *Task parallelism* divides different tasks (or processes) across different nodes or processors in parallel computing environments. In a cluster, different workloads are executed at the same time and the various tasks are distributed among the nodes.

Most systems actually use more than one form of parallelism explained above. In this study, we focus on task parallelism and data parallelism as the form of the parallel computing.

Parallel data processing systems use a "shared nothing" architecture [65] where multiple independent machines are interconnected within a network and every node usually stores only a part of the overall data. In such environment, the best way to achieve data parallelism is to bring the computation to the data. A common example is the WordCount problem, to count the number of words in total. This problem allows most of the computation work to be done locally (called *data locality*). Each

machine (worker) then sends a single number to a single machine (master) to compute the final sum.

Parallel systems are used to achieve speedup and measure scalability. *Speedup* describes the ability to solve a same problem with fixed input size faster, usually by adding more nodes or using any form of parallelism. *Scalability* describes the ability to solve bigger problems by adding more nodes. This is also known as horizontal scalability, or*scaleout*. In contrast, there is also vertical scalability (*scaleup*), where we use more powerful hardware instead of adding more nodes. The following metrics express both properties[1]

$$Speedup = \frac{\text{small system elapsed time}}{\text{big system elapsed time}}$$
$$Scaleout = \frac{\text{small system elapsed time on small problem}}{\text{big system elapsed time on big problem}}$$

Linear speedup means that a problem can be solved twice as fast by doubling the computing resources. Linear scaleout means that a problem doubled in size can be solved in the same time by using twice as many computing resources.

In the following sections, we discuss several existing parallel data processing systems relevant to our feature selection implementation.

4.1.1 Message Passing Interface

Message Passing Interface (MPI) is a standardized portable message-passing communication protocol for parallel computing which supports point-to-point and collective communication [32]. The standard defines the syntax, protocol and semantics of a core of library to allow writing portable message-passing programs in C, C++, and Fortran. MPI goals are high-performance, scalability and portability. MPI systems enables the message-passing parallel programming model, in which the data is moved from the address space of one process to that of another process through operations on each process. The main idea for MPI is to perform all the possible independent

[1]We use the terminology from [56]. In [22], the term scaleup was used for what we call scaleout.

parts of a task in parallel without any communication between the processes [80].

MPI is not suitable for small grain level of parallelism, for example, to exploit the parallelism of multi-core platforms for shared memory multiprocessing [80]. OpenMP is an Application Programming Interface (API) that supports multi-platform shared memory multiprocessing programming [14]. OpenMP became the standard for shared memory parallel programming, but it was still not suitable for distributed memory systems until Coarray Fortran (CAF) [71] was proposed. CAF is a Fortran extension that relied on MPI standard for parallel processing on distributed memory systems. CAF allows two levels of granularity: small grain parallelism with OpenMP and large grain parallelism with MPI-based CAF [80].

MPI uses distributed memory programming model that uses explicit parallelism, which means the developer needs to explicitly address many issues, like task assignment to workers distributed across a potentially large number of machines, synchronization among the different workers, sharing partial results from one worker that is needed by another, and so on. OpenMP uses shared memory parallelism, which means the developer needs to explicitly coordinate access to shared data structures through synchronization primitives, to explicitly handle process synchronization through devices, and to be able to solve common problems, such as deadlocks and race conditions [56]. Developers have to keep track of how resources are made available to workers and must devote a significant amount of attention to low-level system details. In addition, MPI and OpenMP are designed to tackle processor-intensive problems and have limited support for dealing with very large amounts of input data.

4.1.2 MapReduce

Compared to MPI and OpenMP, MapReduce provides an abstraction that hides many system-level details from the programmer. This is one of the most significant advantages of MapReduce. Therefore, a developer can focus on what computations need to be performed, without having to consider how those computations are actually carried out [56]. MapReduce helps in organizing and coordinating large amounts of computation by providing a simple abstraction for the developer, transparently

handling most of the details behind the scenes in a scalable, robust, and efficient manner.

Instead of moving large amounts of data around, MapReduce believes it is far more efficient, if possible, to move the code to the data. MapReduce spreads data across the local disks of nodes in a cluster and runs processes on nodes that hold the data. The complex task of managing storage in such a processing environment is typically handled by a distributed file system that sits underneath MapReduce, like HDFS.

MapReduce can refer to the implementation of the programming model and the execution framework, such as open-source Hadoop implementation in Java. Basically a MapReduce job starts by receiving input as data stored on the underlying distributed file system. MapReduce requires developer to define a mapper and a reducer function. The mapper is applied to every input (split across an arbitrary number of files) to generate a number of intermediate key-value pairs. The reducer is applied to all values associated with the same intermediate key to generate output key-value pairs.

4.1.3 Apache Spark

Although MapReduce is simple to use, the framework has several limitations [87], such as significant overhead for iterative algorithms. Applications such as machine learning and graph analytics iteratively process the data, which means multiple rounds of computation are performed on the same data. Spark is a state-of-the-art framework for high performance parallel computing designed to efficiently deal with iterative computational procedures that recursively perform operations over the same data [108], such as supervised machine learning algorithms. Spark used the concept of maintaining data in memory rather than in disk to reduce data reloading and considerable latencies. Experiments in [101] [107] showed that Spark outperforms by up to two orders of magnitude conventional MapReduce jobs in terms of speed.

Spark introduces a core data unit called Resilient Distributed Datasets (RDDs). RDD is an in-memory abstraction with several important properties:

- *distributed*: RDD can reside in memory, disk or in combination. RDD elements are automatically partitioned across the cluster.
- *immutable*: RDD are immutable once created and can only be created through Spark's deterministic parallel operators.
- *fault-tolerant*: RDD can automatically reconstructs any data lost during failures. Instead of replicating each RDD across nodes for fault-tolerance, Spark remembers the lineage of the RDD (the graph of operators used to build it). This allows Spark to recover lost partitions by recomputing them from base data [107].

Since RDDs can be kept in memory, algorithms can iterate over RDD data many times very efficiently. Each RDD can be either a collection stored in an external storage system, such as a file in HDFS, or a derived dataset created by applying operators (e.g. map, group-by, reduce, hash join) to other RDDs. The operators applied to data elements can either produce other RDDs (transformations) or return values (actions) [80]. However, those operations are computed using the lazy evaluation strategy in order to perform minimal computation and prevent unnecessary memory usage. RDDs are not cached in memory by default. Therefore, when data are reused, a persist method is needed in order to avoid recomputation.

Spark can be run on top of various cluster management options, such as Spark's integrated Standalone Scheduler, Apache Mesos, and Hadoop YARN [46]. Apache Hadoop is an open-source software platform for distributed big data processing over commodity cluster architectures [100], which has three main elements:

- a MapReduce programming model that handles data mapping, shuffling and reduction;
- a distributed file system (HDFS) with high-throughput data access; and
- a cluster manager (YARN) in charge of handling the available computing resources and job scheduling.

HDFS [31] is a hierarchical folder-based API, which stores its file in a shared nothing cluster using commodity hardware. Each file copied to the HDFS is divided

into blocks, the default block size being 128MB, and each block is stored on a few nodes. The replication factor defines to how many nodes it is sent - a high value increases fault tolerance and sometimes data locality but slows down data loading and requires more space.

YARN is the resource management layer of the Apache Hadoop ecosystem, which owns all resources (nodes) in a cluster and overseas and manages all applications running on the cluster. YARN consists of a central ResourceManager and a NodeManager for every node. Applications written for YARN themselves have their own master and worker tasks, but YARN will receive and schedule the job submissions and grant the resources [35].

4.1.4 MPI vs Spark

A study [80] compared MPI on Beowulf against Spark on Hadoop to see how well both distributed computing frameworks could perform. The study evaluated two supervised machine learning algorithms: KNN and Pegasos SVM.

MPI/OpenMP on Beowulf is high-performance oriented and exploits multi-machine or multicore infrastructures. The cluster architecture for this experiment using Beowulf is: all the machines are connected through SSH and one of them has installed CAF implementation accessible from every node. Machines equally store parts of the entire dataset D_n. So, there are n/N_M data samples per machine ($D^i_{n/N_m}, i \in \{1, ..., N_M\}$). There are N_M MPI processes run in parallel, one for each machine, and every process launches N_C OpenMP threads per machine for maximum architecture usage. Reading D^M_{n/N_M} from disk cannot take advantages of the multicore architecture since there is a bottleneck when reading the disk that may depend on the physical implementation of the computing center.

On the other hand, Apache Spark on Hadoop targets iterative algorithms through in-memory computing. Further explanation on this has been covered in Section **??**.

The study showed that MPI/OpenMP outperforms Spark in terms of processing speed and provides more consistent performance. However, Spark has better data management infrastructure and the possibility of dealing with other aspects such as

node failure and data replication. Despite of the result, the same study highlighted that Spark on Hadoop is more still preferrable because it:

- provides a set of tools for data analysis and management that is easy to use, deploy and maintain.
- allows the addition of new nodes at runtime.
- offers a distributed file system with failure and data replication management.

Since in our study we focus on feature selection on machine learning problem, which involves heavy linear algebra computation, Spark becomes a preferrable choice, especially with the existence of MLlib library.

4.2 Selected algorithm

We focus on distributed parallel supervised feature selection algorithm from [110] as seen in 11. The original algorithm was designed to be developed as a SAS High-Performance Analytics procedure [2], which could read data in distributed form and perform parallel feature selection in both symmetric multiprocessing (SMP) mode via multithreading and massively parallel processing (MPP) mode via MPI[92]. The system accomodates high-performance distributed computing frameworks and protocols, such as the Message Passing Interface (MPI) and MapReduce. Further about the concepts of these parallel systems are explained in Section 4.1.

This algorithm was designed to parallelize based on data partitioning to handle large-scale problems. The study in [110] covers the unified approach for both unsupervised and supervised feature selection. In this thesis, we cover only the supervised feature selection for classification problem.

This algorithm is a filter algorithm that uses maximum variance preservation as the criterion, to select features that can best preserve data variance. Assume we want to select k features from a data set $\mathbf{X} \in \mathbb{R}^{n \times m}$ that contains n samples $\mathbf{x}_1, ..., \mathbf{x}_n$ and m features $\mathbf{f}_1, ..., \mathbf{f}_m$. The algorithm treats the data set into 2 parts: $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2)$, where

[2] A SAS procedure is a c-based routine for statistical analysis in the SAS system.

Input : $\mathbf{X}_1, ..., \mathbf{X}_p \in \mathbb{R}^{\frac{n}{p} \times m}, \mathbf{Y}_1, ..., \mathbf{Y}_p \in \mathbb{R}^{\frac{n}{p} \times C}, k$
Output: $\mathbb{L}$, a list of k selected features

On each worker, compute $\mathbf{E}_r \in \mathbb{R}^{C \times m}, \mathbf{v}_r \in \mathbb{R}^{1 \times m}$:

$$\mathbf{E}_r = \mathbf{Y}_r^T \mathbf{X}_r, \quad \mathbf{v}_r = \mathbf{1}^T (\mathbf{X}_r \otimes \mathbf{X}_r); \tag{4.1}$$

Send $\mathbf{E}_r$ and $\mathbf{v}_r$ to the master via `MPI_Reduce` with `MPI_SUM` option:

$$\mathbf{E} = \sum_{r=1}^{p} \mathbf{E}_r, \quad \mathbf{v} = \sum_{r=1}^{p} \mathbf{v}_r; \tag{4.2}$$

On the **master**, compute feature scores

$$\mathbf{s} = \mathbf{1}^T (\mathbf{E} \otimes \mathbf{E}), \quad \mathbf{s} = \mathbf{s} \oslash \mathbf{v}; \tag{4.3}$$

Initialization, $\mathbb{L} = \{F_i\}$, $l = 1$;
while $l < k$ **do**
    The **master** sends $\mathbb{L}$ to all workers via `MPI_Bcast`;
    /* ——————————simultaneously—————————— */
    Workers compute $\mathbf{c}_r^i = \mathbf{X}_r^T \mathbf{f}_r^i, \quad \mathbf{c}_r^i \in \mathbb{R}^{m \times 1}$;
    Workers send $\mathbf{c}_r^i$ to the master via `MPI_Reduce` with `MPI_SUM` option

$$\mathbf{c}^i = \sum_{r=1}^{p} \mathbf{c}_r^i, \quad \mathbf{c}^i \in \mathbb{R}^{m \times 1} \tag{4.4}$$

    /* —————————————————————————————————— */
    On the **master**, construct
    $\mathbf{A}^{-1} \in \mathbb{R}^{l \times l}, \mathbf{C}_{\mathbf{Y},1} \in \mathbb{R}^{C \times l}, \mathbf{C}_{1,2} \in \mathbb{R}^{l \times (m-l))}, \mathbf{C}_{\mathbf{Y},2} \in \mathbb{R}^{C \times (m-l))}, \mathbf{v}_2 \in \mathbb{R}^{1 \times (m-l))}$;
    On the **master**, compute

$$\mathbf{B} = \mathbf{A}^{-1} \mathbf{C}_{1,2}, \quad \mathbf{H} = \mathbf{C}_{\mathbf{Y},1} \mathbf{B}, \quad \mathbf{G} = \mathbf{C}_{\mathbf{Y},2} - \mathbf{H}; \tag{4.5}$$

$$\mathbf{g} = \mathbf{1}^T (\mathbf{G} \otimes \mathbf{G}), \quad \mathbf{w} = \mathbf{v}_2 - \mathbf{1}^T (\mathbf{C}_{1,2} \otimes \mathbf{B}), \quad \mathbf{s} = \mathbf{g} \oslash \mathbf{w}; \tag{4.6}$$

    Master selects $i = \arg\max(s_i | s_i \in \mathbf{s}), \mathbb{L} = \mathbb{L} \cup \{F_i\}, l++$;
end

Algorithm 11: Parallel feature selection based on variance preservation. (Source: [110])

$\mathbf{X}_1 \in \mathbb{R}^{n \times k}$ contains the selected features and $\mathbf{X}_2 \in \mathbb{R}^{n \times (m-k)}$ contains the remaining ones. The criterion selects features by minimizing the following expression:

$$\arg\min_{\mathbf{X}_1} Trace(\mathbf{Y}^T(\mathbf{I} - \mathbf{X}_1(\mathbf{X}_1^T\mathbf{X}_1)^{-1}\mathbf{X}_1^T)\mathbf{Y}) \tag{4.7}$$

Expression 4.7 measures the response variance that cannot be explained by the features in $\mathbf{X}_1$. Minimizing this measure can potentially selects the features that can jointly gives the maximum amount of the response variance.

In classification case, the response vector y with C different values is used to construct a response matrix $\mathbf{Y} \in \mathbb{R}^{n \times C}$ using the following expression:

$$\mathbf{Y}_{i,j} = \begin{cases} \left(\sqrt{\frac{1}{n_j}} - \frac{\sqrt{n_j}}{n}\right) & \text{if } y_i = j \\ -\frac{\sqrt{n_j}}{n} & \text{if } y_i \neq j \end{cases} \tag{4.8}$$

where n_j is the number of instances in class j, and $y_i = j$ denotes that the ith instance belongs to the jth class. Applying it to the Expression 4.7 makes the algorithm selects features that maximize the discriminant criterion of linear discriminant analysis (LDA).

The author [110] uses Sequential Forward Selection (SFS) strategy to generate a suboptimal solution for the problem in an efficient way. This strategy reduces the time complexity of the algorithm into

$$CPU\left(mk\left(\frac{n}{p} + k^2\right)\right) + NET(m(C+k)\log p) \tag{4.9}$$

assuming that $m \gg k > C$ and p is the number of workers (parallel processing units or executor nodes). The complexity suggests that when the number of instances is large and the network is fast enough, the algorithm can speed up feature selection linearly as the number of parallel processing units increases.

The experiment conducted by the author [110] in distributed settings gives almost linear speedup (speedup ratio is almost 1). This becomes the main reason why we choose this algorithm. The other reason is because this algorithm has been designed

for parallel settings to accommodate MPI paradigm in a different system. We would like to know whether the same speed up would apply to an implementation in Apache Spark.

4.3 Feature selection implementation

Using Algorithm 11 as the pseudo-code, we implement the selected feature selection in Apache Spark with Java. The Spark MLlib library offers local matrices and distributed matrices as data types representation for matrices. A distributed matrix has row and column indices and its values that are stored distributively in one or more RDDs. Variants of `DistributedMatrix` that are relevant for our implementation are:

- `BlockMatrix`, since we need to make partitions over the input matrix into several groups of matrices based on the number of workers we are using.
- `RowMatrix`, since we need to build vectors into a matrices after computing covariance vectors on each workers.

We noticed that "Converting a distributed matrix to a different format may require a global shuffle, which is quite expensive."[3]. With that reason, we decided to implement the algorithm with another way. Since Spark MLlib used JBLAS library[4] for the underlying linear algebra operations in Java language, we explored the same library. JBLAS provides `DoubleMatrix` class to represents both matrix and vector, with `double` typed values, and also implemented commonly-used arithmetic operations for matrix.

In our implementation, we created three different object classes: `FeatureScore`, `CovarianceMatrices` and `XYMatrix` to bind several matrices or vectors belong to the same row into one object. This makes is easier to pair the matrix to the original feature values $\mathbf{X}$ and class labels $\mathbf{Y}$. Figure 4-2 depicts the class diagram of those object classes.

[3]Source: http://spark.apache.org/docs/latest/mllib-data-types.html#distributed-matrix
[4]Link: http://jblas.org/

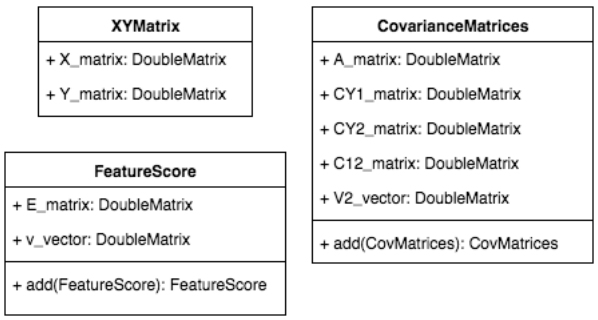


Figure 4-2: Implemented object classes

The process started by reading the data from text file using `textFile(String fileName, int minPartitions)` function, which reads a text file from HDFS or a local file system and return it as an RDD of Strings and distribute the RDD across `minPartitions` partitions. Note that in the pseudocode proposed, the whole matrix is splitted into grids (small matrices) by dividing the rows equally to each partition. So now we have data with $\frac{n}{p}$ rows and 1 column in each partition.

Then we read the input RDD using mapPartitions to form the data into RDD of List of String array. This allowed us to separate the feature matrix $\mathbf{X}$ and the response matrix $\mathbf{Y}$. We constructed Y matrix using the `mapPartitionsToPair()` function to convert the data rows (RDD of List of String array) into RDD of (String, Integer) to represents the class label and the Y value calculated using the Expression 4.8. We used `mapPartitions()` function to construct RDD of `XYMatrix`, which represents the binded matrix $\mathbf{X}$ (feature matrix), with dimension of $\frac{n}{p} \times m$, and $\mathbf{Y}$ (response matrix), with dimension of $\frac{n}{p} \times C$.

For the linear algebra operations, we used the functions implemented in JBLAS library. Table 4.1 shows the list of implemented functions that we used from JBLAS library. For instance, in step 1 to 4, to calculate E matrix, on each worker, we use a mapper function and multiply the matrix $\mathbf{Y}_r^T$ of dimension $C \times \frac{n}{p}$ with the matrix $\mathbf{X}_r$ of dimension $\frac{n}{p} \times m$. And then we apply reducer, which calls the `add()` method in `FeatureScore`.

Operation	Method
Transpose	x.transpose()
Element-wise multiplication	x.mul(y)
Matrix-matrix multiplication	x.mmul(y)
Element-wise division	x.div(y)

Table 4.1: Matrix operations functions in JBLAS

4.4 Classification

We use linear support vector machines (SVM) as the supervised learning models in our experiment. An SVM model represents the instances as points in space. The training instances are mapped so that points from separate categories/classes could form a clear gap as wide as possible. To test/validate the model, new instances are mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

Linear SVM implementation from `spark.mllib` supports binary classification [5]. The input is written in `libsvm` format, to easily represent sparse matrix. The training used the SVM model with Stochastic Gradient Descent (SGD) to allow more optimized classification model.

The classification process is evaluated using Area under ROC and area under Precision-Recall implemented by `BinaryClassificationMetrics` class in MLlib.

Area under ROC

Receiver Operator Characteristics (ROC) curve plots the false-positive rate as the X-axis and true-positive rate as the Y-axis, to illustrates the performance of a binary classifier system. The accuracy of a test depends on how well the test separates the group being tested into those with positive result (e.g. ad) and negative result (e.g. nonad). The accuracy is measured by the are under ROC curve. An example of an ROC curve in Figure 4-3 shows that an area of 1 represents a perfect test, while an area of 0.5 represents a worthless test. A rough guide for classifying the accuracy of a diagnostic test is the traditional academic point system [96]:

[5]Source: http://spark.apache.org/docs/latest/mllib-linear-methods.html

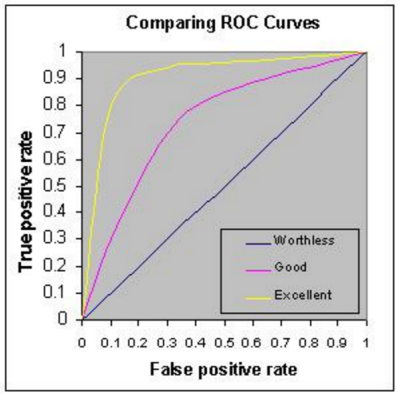


Figure 4-3: Area under ROC (Source: [96])

- 0.90 - 1 = excellent (A)
- 0.80 - 0.90 = good (B)
- 0.70 - 0.80 = fair (C)
- 0.60 - 0.70 = poor (D)
- 0.50 - 0.60 = fail (F)

Area under Precision-Recall curve

For binary classification task, precision is the fraction of retrieved instances that are relevant, while recall is the fraction of relevant instances that are retrieved. Basically the following expression can be used to compute precision and recall.

$$Precision = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} \tag{4.10}$$

$$Recall = \frac{\text{true positive}}{\text{true positive} + \text{false negative}} \tag{4.11}$$

Precision-Recall (PR) curves are useful alternatives to ROC curves on imbalanced or skewed data sets [8], as it can highlight performance differences that are lost in ROC curves. [19] showed that an algorithm that optimizes the area under the ROC curve is not guaranteed to optimize the area under PR curve. That is why we additionally used this metric.

Chapter 5

Evaluation

To validate the accuracy and to assess the effect of feature selection towards classification problem, we conducted small experiments to run our feature selection implementation upon two datasets taken from UCI machine learning dataset repository[1].

In the following sections, we explain the characteristics of the dataset in Section 5.1. We also explain the experiment setup (Section 5.2) and the result (Section 5.3). The experiment was done both on distributed cluster settings.

5.1 Dataset

We utilized two different datasets taken from UCI machine learning dataset repository:

1. *Internet advertisements dataset:* This dataset represents a set of possible advertisements in Internet pages. It was originally used in [53] to predict whether a certain image from a page is an internet advertisements or not. The dataset has the following properties:

    - Number of instances: 3279 instances
    - Number of features: 1558 features (3 continuous, others binary).

[1]Link: https://archive.ics.uci.edu/ml/datasets.html

- Missing attribute values: One or more of the three continuous features are missing in 28% of the instances. We treat these missing values as "0" to simplify the problem.
- Class distribution: 2821 nonads, 458 ads

The features encode the geometry of the image (if available) as well as phrases occuring in the URL, the image's URL and alt text, the anchor text, and words occuring near the anchor text. Complete description of the features can be seen in [53].

2. *Dorothea dataset:* This dataset represents a drug discovery dataset. The chemical compounds represented by structural molecular features must be classified as active (binding to thrombin) or inactive. The dataset was originally used as one of 5 datasets of the NIPS 2003 feature selection challenge. The dataset has the following properties:

   - Number of instances: 800 instances
   - Number of features: 100000 features (50000 real features, 50000 probe/-dummy features)
   - Class distribution: We use only the training data with 78 positive examples and 722 negative examples

   The dataset are represented as a sparse matrix showing only the index of the feature/column with non-zero values in each row.

To easily represent sparse matrix, we converted all dataset into LibSVM format[2] using our own implementation.

To treat the real values in Internet advertisement dataset, we implemented binarization function in the input formatter. The binarization threshold is fixed to 320 as the real values ranges from 0 to 640.

[2]Link: http://www.csie.ntu.edu.tw/ cjlin/libsvm/

5.2 Experiment Setup

In the experiment we wanted to measure the scaleout factor of the selected feature selection algorithm. We used Google Cloud Dataproc for Spark using the configuration in Table 5.1.

Master node	
Machine type	n1-highmem-4
Virtual CPUs	4
Memory	26 GB 6.50$GBpervirtualCPU$
Primary disk size	500 GB
Worker nodes	
Total worker nodes	(varied 1 to 5)
Machine type	n1-highmem-4
Virtual CPUs	4
Memory	26 GB 6.50$GBpervirtualCPU$
Primary disk size	500 GB
Spark	
spark.executor.cores	4
spark.executor.instances	$1-5$

Table 5.1: Cluster configuration

We configured the system to use all available resources, especially memory and disk on both Spark driver and executors. Number of virtual CPUs can be seen as number of cores per CPU. For example, with 2 worker nodes, we have 2×4 cores. The total of 8 cores must be divided among number of executors used in the experiment. We set 4 cores to be used for each executor, so we have 2 executors in total.

In our implementation, we used Java 1.8 as the programming language, Spark core 2.11 as the large data processing system, Spark MLlib 2.11 version 1.6.0 as the machine learning library and JBLAS 1.2.4 as the linear algebra library.

5.3 Experiment Results

We conducted the experiment with different number of selected features and number of nodes to measure the scaleout factor of the selected feature selection algorithm.

Figure 5-1 shows the running time for selecting 50 to 300 features of Internet

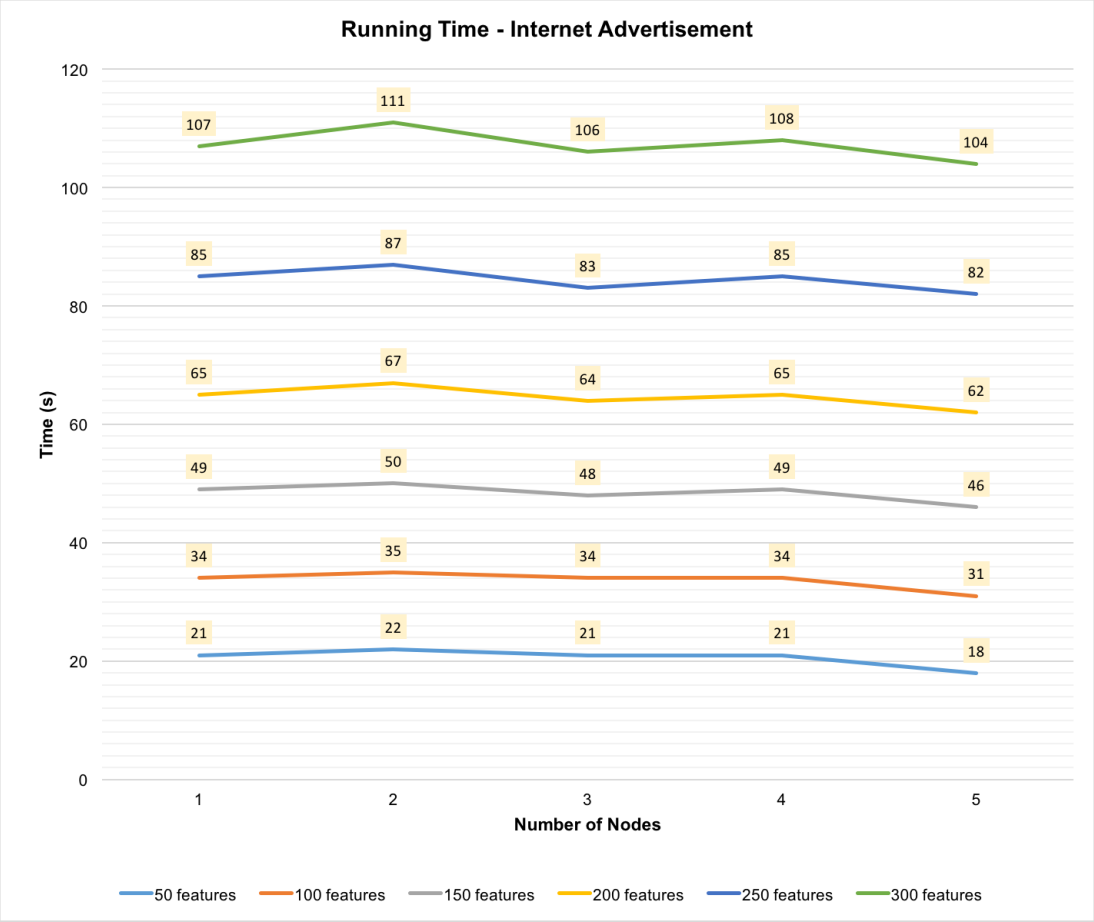


Figure 5-1: Running time for Internet Advertisements datasets

Advertisement datasets. From the figure, when we add more nodes, the running time can be less than running on a single node, or at the worst case, the same as running on a single node. Our experiment is limited to the number of nodes, up to 5 number of nodes, since the experiment on Google is limited to having maximum 25 cores in total (5×4 cores for workers and 4 cores for driver/master).

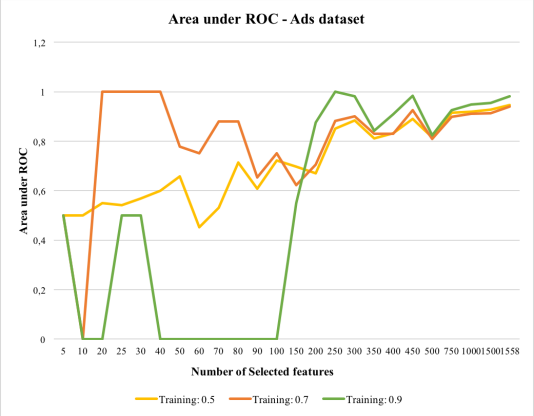
During the experiment, we also noticed that the selected algorithm consumed huge amount of memory. In the beginning, we used standard CPU with less RAM provided by Google Cloud. Our implementation reached OutOfMemoryException when we tried to select more than 30 features on high parallelism degree for Dorothea dataset. Then we tried to upgrade the memory of the CPU and get bigger number of selected features before reaching an error. Since Google Cloud has limitations on the free service, we have to limit our experiment to the specfied machine in Table 5.1.

The reason is because the algorithm process the huge matrix into submatrices with numerous rows instead of processing the matrix per line/data points. This leads to forcing the heavy computation of matrix operations has to be performed on each node. Each node has to compute several matrices, such as: feature matrix, response matrix, and covariance matrices; and keep them as an RDD in memory buffers.

Huge memory space taken up also affects the performance of the algorithm in Spark. When the limit of memory used for caching is exceeded, Spark will drop older partitions from the memory. This means the next time a certain matrix in the older partition is needed, Spark has to recompute the RDD based on the lineage.

After getting the selected features, we run an SVM classifier to test the importance of the selected features. The training algorithm to build the SVM model is run for 100 times for every experiment that we make. We used three different training data proportion: 0.5, 0.7 and 0.9 from the whole data points (instances). We measure the area under ROC and area under Precision-Recall curve to measure how well the classes are separated based on the given selected features.

For Internet advertisements dataset, the area under ROC (AUROC) of the original dataset (without feature selection) is between 0.89 to 0.98 and the area under Precision-Recall curve (AUCPR) is between 0.79 to 0.92. Figure 5-2 shows that



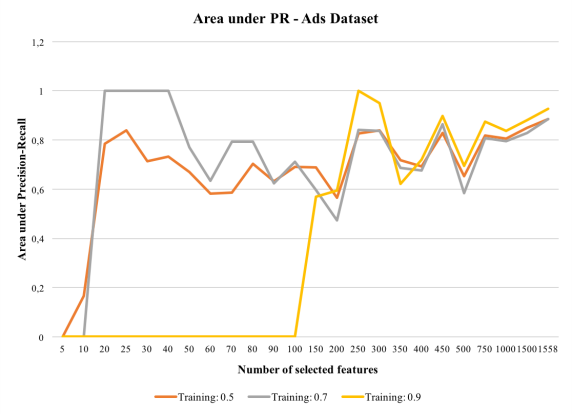
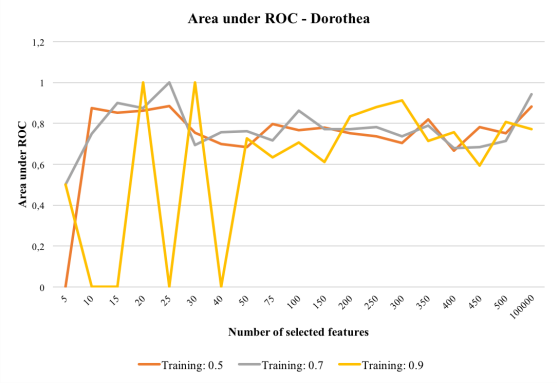


Figure 5-2: Classification result for Internet Advertisement dataset



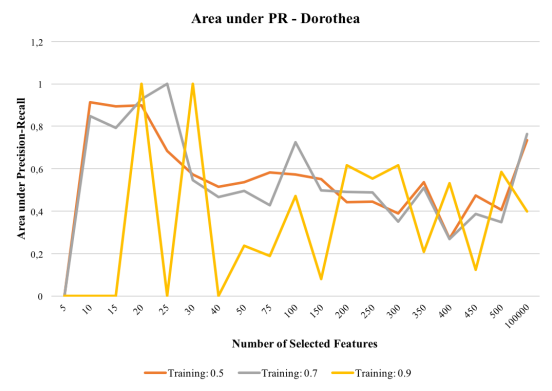


Figure 5-3: Classification result for Dorothea dataset

when the number of features selected is less than 100, for training proportion of 0.9, the SVM model can not correctly classify the labels, because the learning features are not enough.

The highest score in both AUROC and AUCPR is when we have 250 selected features and the training proportion is 0.9, which covers 2951 out of 3279 instances. The samples are enough to cover the characteristics of both classes, since 14% of the whole instances belong to "ad" class and the rest to "nonad" class.

For Dorothea dataset, the AUROC of the original dataset (without feature selection) is between 0.77 to 0.94 and the area under Precision-Recall curve (AUCPR) is between 0.4 to 0.76. Figure 5-3 shows that when the number of features selected is less than 50, for training proportion of 0.9, the SVM model can not correctly classify the labels, because the learning features are not enough.

As we know from the dataset description in 5.1, Dorothea dataset has 50000 dummy features and 50000 real features, while our test only covers until 1000 selection, we could not see whether the algorithm could correctly remove all the dummy features. Besides, the dataset sponsor did not provide the information on the real features. Based on our experiment, the highest score in AUROC and AUCPR still cannot defeat the score of the original dataset with all the features.

Chapter 6

Conclusion

This chapter concludes our study. As stated in the beginning (Section 1.1), the goal of this thesis was to provide insights over existing feature selection algorithms in distributed machine learning context. We listed several existing feature selection algorithms and analyzed the complexity and performance of each algorithm.

From computational efficiency aspect, most filter methods have quadratic to exponential complexity. However, generally the implementation of filter algorithms are simple and could be easily converted to distributed settings. Some dependency-based filter algorithms used tree as the data structure. These algorithms need to be heavily modified to be implemented in a distributed setting, since implementation that requires tracing the tree spanned across multiple nodes would incur huge communication cost.

Consistency-based filter algorithms, like QBB, seems to be a good candidate to be parallelized. The remaining challenge is to implement recursive method in parallel setting, which would be computationally expensive if not handled properly. Splitting the task into a parallel task whenever they need perform smaller task or calling recursive method could be one way to solve it. However, we need to carefully define when to stop parallelizing.

Other filter algorithms that use heuristic or probabilistic search are believed to be faster. However, these algorithms could not guarantee optimal subsets. As an example, SetCover is fast and deterministic, but it may face problems with data

having highly interdependent features [18].

Most wrapper algorithms have quadratic complexity. Beside more complicated implementation, the drawback of wrapper algorithms is that it heavily depends on the classification problem. If the complexity of the classifier is big, it would trivially incur bigger cost along the number of iterations made. Converting wrapper algorithms to distributed setting would heavily depends on the nature of the classifier used. Therefore, the scalability depends on the scalability of the classifier.

Embedded methods that iterates over the number of features and functions as regularization, have linear complexity for one step of classification learning process. So depending on the number of iteration, embedded methods are another feasible option to be implemented in distributed settings.

Overall, the existing feature selection algorithms are designed for traditional centralized computing environments and most have the quadratic or polynomial complexity. Among the three methods, embedded methods are claimed to have the fastest time. However, they are bonded to the classification model used. Filter methods are on the other hand flexible, but also do not take into account the biases of the classifiers. In terms of scalability, embedded methods and filter methods are more scalable compared to wrapper methods.

There are still limited number of algorithms designed and implemented in distributed settings. Parallel feature selection from random subsets and parallel feature selection for logistic regression, and variance preservation parallel feature selection are some examples.

To the best of our knowledge, an independent experiments performed over these algorithms on Spark does not exist yet. Our thesis conducts an experiment on the variance based preservation algorithm to verify the applicability in Spark. The result showed that the algorithm could run slightly faster when the number of executor nodes are added. However, the algorithm has the drawback of having consumed too much memory space, which could also affects the performance. To make the algorithm becomes more scalable, one has to change the partitioning mechanism and reduce the matrix computation on the executor nodes. This way, the algorithm might utilize

less memory and therefore, could incur better speedup.

6.1 Future Works

In this section we briefly discuss potential areas of future work related to the feature selection itself and to the implementation. We lists down the room for improvement that possibly leads to better performance when scaling up the feature selection:

- Design new parallelized feature selection algorithms

  Although there are tons of feature selection algorithms, currently the number of parallel feature selection algorithms are still very limited, especially since the algorithms usually heavily depends on which distributed system they used. Especially for Spark, as shown in the experiment, the memory efficiency could also be a factor that determines how fast the parallelized algorithm could run.

- Convert existing feature selection algorithms

  Converting existing algorithms to parallel settings requires the consideration over each detail of the algorithm, whether the data structure involved are compatible to distributed setting and whether the computation can be divided into smaller tasks and computed independently by the workers. Converting filter algorithms to a distributed setting is the most feasible choice as most of the filter algorithms works by scoring the features before choosing or filtering out the irrelevant or redundant features. Converting embedded methods to a distributed setting can also be feasible as implementing classifier on distributed setting.

Appendix A

Java Implementation

Listing A.1: Abstract class

```
1  import org.apache.spark.SparkConf;
2  import org.apache.spark.api.java.JavaRDD;
3  import org.apache.spark.api.java.JavaSparkContext;
4
5  public abstract class FSInputReader
6  {
7      private JavaSparkContext sc;
8
9      public FSInputReader(String fileName, int numOfExecutors) {
10         SparkConf conf = new SparkConf().setAppName("Feature Selector");
11           sc = new JavaSparkContext(conf);
12           rawData = sc.textFile(fileName, numOfExecutors).cache();
13       }
14
15       abstract public void process(int loopNumber, String outputFileName,
             String datasetName, String bucketName);
16 }
```

Listing A.2: Feature Selection class

```
1  public class AdsInputReader extends FSInputReader
2  {
3      private Broadcast bcFeatures;
4      private Broadcast<double[]> bcInstances;
5      private JavaRDD<XYMatrix> xyMatrix;
6
7      public AdsInputReader(String filename, int numOfExecutors)
8      {
9          super(filename, numOfExecutors);
10     }
11
12     public void process(int loopNumber, String outputName, String
           datasetName, String bucketName)
13     {
14         // get number of original features in the dataset
15         int numberOfFeatures;
16         bcFeatures = getSparkContext().broadcast(numberOfFeatures);
17
18         // read data from input file
19         JavaRDD<List<String[]>> rawData =
               getRawData().mapPartitions(iterator -> {
20             List<String[]> list = new ArrayList<>();
21
22             while(iterator.hasNext()){
23                 list.add(iterator.next().split(" "));
24             }
25
26             return Collections.singleton(list);
27         });
```

```
28
29          countClasses(rawData); // build response matrix Y
30
31          // actually do feature selection
32          DoubleMatrix score = computeFeatureScores(rawData, bcFeatures,
                bcInstances);
33          Set<Integer> selectedFeatures = getBestFeatures(score, loopNumber);
34          DoubleMatrix subMatrix = getSubMatrix(selectedFeatures);
35
36          // write output to file and statistics
37      }
```

Listing A.3: Step 1 to 3 in computeFeatureScores method

```
1  // map values into pairs of X (features matrix) and Y (response matrix)
2  xyMatrix = logData.mapPartitions(iterator -> {
3      ArrayList<Double[]> featureMatrix = new ArrayList<>();
4      ArrayList<Double[]> responseMatrix = new ArrayList<>();
5      while(iterator.hasNext()) {
6          List<String[]> list = iterator.next();
7          for(String[] splittedLine : list) {
8              featureMatrix.add(getFeatures(splittedLine, bcFeatures));
9              // if splittedLine is belongs to class j, add responseMatrix with
                   Y value for class j.
10             }
11     }
12
13     // initiate DoubleMatrix x and y from featureMatrix and responseMatrix
14
15     return Collections.singleton(new XYMatrix(x, y));
```

```
16  }).cache();
17
18    JavaRDD<FeatureScore> fScoreMatrix = xyMatrix.map(matrix -> {
19        DoubleMatrix x = matrix.getX();
20        DoubleMatrix y = matrix.getY();
21        DoubleMatrix ones = DoubleMatrix.ones(x.getRows());
22        return new FeatureScore(y.transpose().mmul(x),
              ones.transpose().mmul(x.mul(x)));
23  });
24
25          FeatureScore totalScore = fScoreMatrix.reduce((a, b) -> a.add(b));
26
27          DoubleMatrix e = totalScore.getEMatrix();
28          DoubleMatrix v = totalScore.getVMatrix();
29          DoubleMatrix s =
                DoubleMatrix.ones(e.getRows()).transpose().mmul(e.mul(e));
30
31          s = s.div(v); // Element-wise division on matrix
```

List of Figures

List of Tables

Bibliography

[1] Parallel computing. http://en.wikipedia.org/wiki/Parallel_computing. Accessed: 2016-07-03.

[2] C. C. Aggarwal. *Data Classification: Algorithms and Applications*. Chapman & Hall/CRC, 1st edition, 2014.

[3] A. Alexandrov, S. Ewen, M. Heimel, F. Hueske, O. Kao, V. Markl, E. Nijkamp, and D. Warneke. Mapreduce and pact-comparing data parallel programming models. In *Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW)*, volume 14, pages 25–44, 2011.

[4] H. Almuallim and T. G. Dietterich. Efficient algorithms for identifying relevant features. Technical report, Corvallis, OR, USA, 1992.

[5] H. Almuallim and T. G. Dietterich. Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2):279–305, September 1994.

[6] M. A. Beyer and D. Laney. The importance of big data: A definition. Technical report, Gartner, 2012.

[7] L. Bobrowski. Feature selection based on some homogeneity coefficient. In *Proc. of the 9th International Conf. on Pattern Recognition*, volume 1, pages 544–546, Nov 1988.

[8] K. Boyd, K. H. Eng, and C. D. Page. *Area under the Precision-Recall Curve: Point Estimates and Confidence Intervals*, pages 451–466. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[9] C. Cardie. Using decision trees to improve case-based learning. In *Proc. of the 10th International Conf. on Machine Learning*, pages 25–32. Morgan Kaufmann, 1993.

[10] R. Caruana and D. Freitag. Greedy attribute selection. In *Proc. of the 11th International Conf. on Machine Learning*, pages 28–36. Morgan Kaufmann, 1994.

[11] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proc. of the 25th International Conf. on Machine Learning*, ICML '08, pages 96–103, New York, NY, USA, 2008. ACM.

[12] G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers and Electrical Engineering*, 40(1):16–28, January 2014.

[13] Y.L. Cun, J. S. Denker, and S. A. Solla. Advances in neural information processing systems 2. chapter Optimal Brain Damage, pages 598–605. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[14] L. Dagum and R. Menon. Openmp: An industry-standard api for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1):46–55, January 1998.

[15] S. Das. Filters, wrappers and a boosting-based hybrid for feature selection. In *Proc. of the 18th International Conf. on Machine Learning*, ICML '01, pages 74–81, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[16] M. Dash. Feature selection via set cover. In *Proc. of Knowledge and Data Engineering Exchange Workshop*, pages 165–171, Nov 1997.

[17] M. Dash and H. Liu. *Hybrid search of feature subsets*, pages 238–249. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[18] M. Dash, H. Liu, and H. Motoda. *Consistency Based Feature Selection*, pages 98–109. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[19] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 233–240, New York, NY, USA, 2006. ACM.

[20] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[21] P. A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach*. Prentice Hall, 1982.

[22] D. DeWitt and J. Gray. Parallel database systems: The future of high performance database systems. *Communications of the ACM*, 35(6):85–98, June 1992.

[23] J. Doak. An Evaluation of Feature Selection Methods and Their Application to Computer Security. Technical report, UC Davis Department of Computer Science, 1992.

[24] Pedro Domingos. Why does bagging work? a bayesian account and its implications. In *In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 155–158. AAAI Press, 1997.

[25] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, 2nd edition, 2001.

[26] J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001.

[27] I. Foroutan and J. Sklansky. Feature selection for automatic classification of non-gaussian data. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(2):187–198, March 1987.

[28] T. Fountain, H. Almuallim, and T. G. Dietterich. Learning with many irrelevant features. Technical report, Corvallis, OR, USA, 1991.

[29] D. Gamberger and N. Lavrac. Conditions for occam's razor applicability and noise elimination. In *Proc. of the 9th European Conf. on Machine Learning*, ECML '97, pages 108–123, London, UK, 1997. Springer-Verlag.

[30] D. J. Garcia, L. O. Hall, D. B. Goldgof, and K. Kramer. A parallel feature selection algorithm from random subsets. In *Proc. of the 17th European Conf. on Machine Learning and the 10th European Conf. on Principles and Practice of Knowledge Discovery in Databases*, 2006.

[31] S. Ghemawat, H. Gobioff, and S. Leung. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[32] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.

[33] Q. Gu, Z. Li, and J. Han. Generalized fisher score for feature selection. *Computing Research Repository (CoRR)*, abs/1202.3725, 2012.

[34] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.

[35] A. Hacker. Evaluating parallel data processing systems for scalable feature selection. unpublished thesis, 2013.

[36] M. A. Hall. Correlation-based feature selection for machine learning. Technical report, 1998.

[37] M. A. Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *Proc. of the 17th International Conf. on Machine Learning*, ICML '00, pages 359–366, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[38] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2nd edition, February 2009.

[39] X. He, D. Cai, and P. Niyogi. Laplacian score for feature selection. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 507–514. MIT Press, 2006.

[40] J. Huang, J. L. Horowitz, and S. Ma. Asymptotic properties of bridge estimators in sparse high-dimensional regression models. *The Annals of Statistics*, pages 587–613, 2008.

[41] M. Ichino and J. Sklansky. Feature selection for linear classifier. *Proc. of the 7th International Conf. on Pattern Recognition*, 1:124–127, 1984.

[42] M. Ichino and J. Sklansky. Optimum feature selection by zero-one integer programming. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14(5):737–746, Sept 1984.

[43] B. Igelnik and J. M. Zurada. *Efficiency and Scalability Methods for Computational Intellect*. IGI Global, Hershey, PA, USA, 1st edition, 2013.

[44] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big data and its technical challenges. *Commun. ACM*, 57(7):86–94, July 2014.

[45] D. S. Johnson. Approximation algorithms for combinatorial problems. In *Proc. of the 5th Annual ACM Symposium on Theory of Computing*, STOC '73, pages 38–49, New York, NY, USA, 1973. ACM.

[46] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia. *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, Inc., 1st edition, 2015.

[47] K. Kira and L. A. Rendell. A practical approach to feature selection. In *Proc. of the 9th International Workshop on Machine Learning*, ML92, pages 249–256, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[48] K. Knight and Fu W. Asymptotics for lasso-type estimators. *Annals of Statistics*, pages 1356–1378, 2000.

[49] R. Kohavi. *Wrappers for Performance Enhancement and Oblivious Decision Graphs*. PhD thesis, Stanford, CA, USA, 1996. UMI Order No. GAX96-11989.

[50] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, December 1997.

[51] D. Koller and M. Sahami. Toward optimal feature selection. In *Proc. of the 13th International Conf. on Machine Learning*, pages 284–292, 1995.

[52] I. Kononenko. Estimating attributes: Analysis and extensions of relief. In *Proc. of the European Conf. on Machine Learning*, ECML-94, pages 171–182, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[53] Nicholas Kushmerick. Learning to remove internet advertisements. In *Proceedings of the Third Annual Conference on Autonomous Agents*, AGENTS '99, pages 175–181, New York, NY, USA, 1999. ACM.

[54] P. Langley and S. Sage. *Oblivious Decision Trees and Abstract Cases*, pages 113–117. AAAI Press, Seattle, WA, 1994.

[55] M. H. C. Law, M. A. T. Figueiredo, and A. K. Jain. Simultaneous feature selection and clustering using mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1154–1166, Sept 2004.

[56] J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan and Claypool Publishers, 2010.

[57] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.

[58] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, 1st edition, 1998.

[59] H. Liu, H. Motoda, and M. Dash. A monotonic measure for optimal feature selection. In *Proc. of European Conf. on Machine Learning*, pages 101–106, 1998.

[60] H. Liu, H. Motoda, and L. Yu. A selective sampling approach to active feature selection. *Artificial Intelligence*, 159(1–2):49 – 74, 2004.

[61] H. Liu and R. Setiono. Feature selection and classification - a probabilistic wrapper approach. In *Proc. of the 9th International Conf. on Industrial and Engineering Applications of AI and ES*, pages 419–424, 1996.

[62] H. Liu and R. Setiono. A probabilistic approach to feature selection – a filter solution. In *Proc. of the 13th International Conf. on Machine Learning*, pages 319–327. Morgan Kaufmann, 1996.

[63] H. Liu and R. Setiono. Scalable feature selection for large sized databases. In *Proc. of the 4th World Congress on Expert Systems*, pages 521–528. Morgan Kaufmann, 1998.

[64] H. Liu and L. Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502, April 2005.

[65] Stonebraker M. The case for shared nothing. *Database Engineering*, 9:4–9, 1986.

[66] M. Modrzejewski. Feature selection using rough sets theory. In *Proc. of the European Conf. on Machine Learning*, ECML '93, pages 213–226, London, UK, UK, 1993. Springer-Verlag.

[67] L. C. Molina, L. Belanche, and A. Nebot. Feature selection algorithms: a survey and experimental evaluation. In *Proc. of IEEE International Conf. on Data Mining 2002 (ICDM)*, pages 306–313, 2002.

[68] A. W. Moore. Efficient algorithms for minimizing cross validation error. In *Proc. of the 11th International Conf. on Machine Learning*, pages 190–198. Morgan Kaufmann, 1994.

[69] A. N. Mucciardi and E. E. Gose. A comparison of seven techniques for choosing subsets of pattern recognition properties. *IEEE Transactions on Computers*, C-20(9):1023–1031, Sept 1971.

[70] P. M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, C-26(9):917–922, Sept 1977.

[71] R. W. Numrich and J. Reid. Co-array fortran for parallel programming. *SIGPLAN Fortran Forum*, 17(2):1–31, August 1998.

[72] A. L. Oliveira and A. Sangiovanni-Vincentelli. Constructive induction using a non-greedy strategy for feature selection. In *Proc. of the 9th International Workshop on Machine Learning*, ML92, pages 355–360, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[73] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, Aug 2005.

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

[75] P. Pudil, J. Novovičová, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15(11):1119–1125, 1994.

[76] C. E. Queiros and E. S. Gelsema. On feature selection. In *Proc. of the 7th International Conf. on Pattern Recognition*, pages 128–130, 1984.

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

[78] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[79] J. Reunanen. Overfitting in making comparisons between variable selection methods. *Journal of Machine Learning Research*, 3:1371–1382, March 2003.

[80] J. L. Reyes-Ortiz, L. Oneto, and D. Anguita. Big data analytics in the cloud: Spark on hadoop vs mpi/openmp on beowulf. *Procedia Computer Science*, 53:121 – 130, 2015.

[81] I. Rivals and L. Personnaz. Mlps (mono layer polynomials and multi layer perceptrons) for nonlinear modeling. *Journal of Machine Learning Research*, 3:1383–1398, March 2003.

[82] D. Ron, Y. Singer, and N. Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2):117–149, 1996.

[83] J. C. Schlimmer. Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In *Proc. of the 10th International Conf. on Machine Learning*, pages 284–290. Morgan Kaufmann, 1993.

[84] B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, Cambridge MA, 2002.

[85] J. Segen. Feature selection and constructive inference. In *Proc. of the 7th International Conf. on Pattern Recognition*, pages 1344–1346, 1984.

[86] J. Sheinvald, B. Dom, and W. Niblack. A modeling approach to feature selection. In *Proc. of the 10th International Conf. on Pattern Recognition*, volume i, pages 535–539, 1990.

[87] J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, B. Reinwald, and F. Özcan. Clash of the titans: Mapreduce vs. spark for large scale data analytics. *Proc. of the VLDB Endowment*, 8(13):2110–2121, September 2015.

[88] M. Singh and G. M. Provan. Efficient learning of selective bayesian network classifiers. In *Proc. of the 13th International Conf. on Machine Learning*, pages 453–461. Morgan Kaufmann, 1996.

[89] S. Singh, J. Kubica, S. Larsen, and D. Sorokina. Parallel large scale feature selection for logistic regression. In *Proc. of 2009 SIAM International Conf. on Data Mining*, pages 1172–1183, 2009.

[90] D. B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proc. of the 11th International Conf. on Machine Learning (ICML)*, pages 293–301. Morgan Kaufmann, 1994.

[91] N. Slonim, G. Bejerano, S. Fine, and N. Tishby. Discriminative feature selection via multiclass variable memory markov model. *EURASIP Journal of Applied Signal Processing*, 2003:93–102, January 2003.

[92] M. Snir. *MPI–the Complete Reference: The MPI core*, volume 1. MIT press, 1998.

[93] P. Somol, P. Pudil, J. Novovičová, and P. Paclík. Adaptive floating search methods in feature selection. *Pattern Recognition Letters*, 20(11–13):1157–1163, 1999.

[94] D. J. Stracuzzi and P. E. Utgoff. Randomized variable elimination. *Journal of Machine Learning Research (JMLR)*, 5:1331–1362, December 2004.

[95] J. Tang, S. Alelyani, and H. Liu. Feature selection for classification: A review. *Data Classification: Algorithms and Applications*, page 37, 2014.

[96] T. MD Tape. The area under an roc curve. `http://gim.unmc.edu/dxtests/ROC3.htm`. Accessed on: 1 July 2016.

[97] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[98] E. Triantaphyllou and G. Felici. *Data Mining and Knowledge Discovery Approaches Based on Rule Induction Techniques (Massive Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[99] H. Vafaie and I. F. Imam. Feature selection methods: Genetic algorithms vs greedy-like search. In *Proc. of the International Conf. on Fuzzy and Intelligent Control Systems*, 1994.

[100] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[101] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: Sql and rich analytics at scale. In *Proc. of 2013 ACM SIGMOD International Conf. on Management of Data*, SIGMOD '13, pages 13–24. ACM, 2013.

[102] E. P. Xing, M. I. Jordan, and R. M. Karp. Feature selection for high-dimensional genomic microarray data. In *Proc. of the 18th International Conf. on Machine Learning*, pages 601–608. Morgan Kaufmann, 2001.

[103] L. Xu, W. Lin, and C.C. J. Kuo. *Visual Quality Assessment by Machine Learning*. Springer Singapore, 2015.

[104] L. Xu, P. Yan, and T. Chang. Best first strategy for feature selection. In *Proc. of the 9th International Conf. on Pattern Recognition*, pages 706–708 vol.2, Nov 1988.

[105] J. Yang and V. Honavar. Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems and their Applications*, 13(2):44–49, Mar 1998.

[106] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proc. of the 20th International Conf. on Machine Learning (ICML-2003)*, pages 856–863, 2003.

[107] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. 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*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[108] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proc. of the 2nd USENIX Conf. on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[109] Peng Zhao and Bin Yu. On model selection consistency of lasso. *Journal of Machine Learning Research*, 7:2541–2563, December 2006.

[110] Z. Zhao, J. Cox, D. Duling, and W. Sarle. Massively parallel feature selection: An approach based on variance preservation. In *Proc. of 2012 European Conf. on Machine Learning and Knowledge Discovery in Databases - Volume Part I*, ECML PKDD'12, pages 237–252. Springer-Verlag, 2012.

[111] Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101(476):1418–1429, 2006.

[112] Hui Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 67(2):301–320, 2005.