

# Methods For Approximating Forward Selection Of Features In Information Retrieval Problems Using Machine Learning Methods

A thesis  
submitted to the faculty of the graduate school  
of the University of Minnesota  
by

Shruti Pandey

in partial fulfillment of the requirements  
for the degree of  
Master of Science

August 2008

Department of Computer Science  
University of Minnesota Duluth  
Duluth, MN 55812  
USA

UNIVERSITY OF MINNESOTA

This is to certify that I have examined this copy of master's thesis by

Shruti Pandey

and have found that it is complete and satisfactory in all respects,  
and that any and all revisions required by the final  
examining committee have been made.

Dr. Richard Maclin

---

Name of Faculty Advisor

---

Signature of Faculty Advisor

---

Date

GRADUATE SCHOOL



## **Acknowledgements**

I would like to take this opportunity to acknowledge all those people who have helped me during this thesis work. First of all, I would like to thank my advisor, Dr. Richard Maclin for introducing me to the world of Machine Learning. I would also like to thank him for his valuable suggestions, wonderful guidance during the course of this thesis work, and finally for his patience in reviewing my thesis.

I would like to thank my committee members Dr. Carolyn Crouch and Dr. Douglas Dunham for their valuable feedback on my work.

I would like to thank all the faculty and staff at the Department of Computer Science at the University of Minnesota Duluth for their assistance during my master's course-work. I would thank my fellow graduate students who offered great help during my study and stay in Duluth.

Finally, I would like to thank my best friend Nachiket Kamat, my family members and all my friends for encouraging and supporting me throughout my life.

## ABSTRACT

In recent years, with the enormous increase in the number of text databases available online, and the consequent need for better techniques to access this information, there has been strong interest in research done in the area of Information Retrieval (IR). Finding desired information from this vast amount of data is equivalent to finding a needle in a haystack. A document collection can be described by a set of features that may or may not be relevant in representing a document's content. Feature subset selection deals with the identification of those features that are relevant to the learning process.

In this thesis, we examine ways to apply Machine Learning (ML) to IR, focusing on the problem of features selection. We propose methods based on the use of the Random Feature Selection, the  $\chi$  - Square Statistic, the Information Gain statistic, the Term Frequency / Inverse Document Frequency (TF/IDF) statistic and a Boosting Method for Approximating Forward Selection of Features. We examine these methods for a way of approximating Forward Selection (Miller, 1990). Forward Selection's approach for selecting features is extremely computationally expensive making it intractable for large data sets. The results from our experiments show comparable accuracy in document classification and a faster execution time as compared to the Forward Selection algorithm. We also compare our results with experiments performed on the whole data set. These results show that selecting a subset of features leads to better performance in terms of time and accuracy of the learning methods as compared to the whole data set.

# Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
1.1 Thesis Statement.....	3
1.2 Thesis Outline.....	4
<b>2. Background.....</b>	<b>5</b>
2.1 Information Retrieval Problems.....	5
2.1.1 Term Frequency / Inverse Document Frequency (TF/IDF).....	6
2.2 Statistical Ideas.....	9
2.2.1 Chi ( $\chi$ ) Square.....	10
2.2.1.1 Definition of Karl Pearson statistic.....	10
2.2.2 Information Gain.....	11
2.3 Machine Learning.....	15
2.3.1 Classification.....	17
2.3.2 The Naïve Bayes Classifier.....	17
2.3.3 A Decision Tree Classifier.....	22
2.3.4 Support Vector Machines (SVMs).....	24
2.3.5 Boosting Classifiers.....	25
2.3.6 The Random Forest Classifier.....	26
2.4 Feature Selection Methods.....	27
2.4.1 Wrappers.....	27
2.4.2 Forward Selection.....	29
2.5 Related Work.....	31
<b>3. Methods for Selecting Features.....</b>	<b>33</b>
3.1 Selecting Features Randomly.....	33
3.2 Selecting Features Using Chi( $\chi$ ) Square.....	35
3.3 Selecting Features Using Information Gain.....	38
3.4 Selecting Features using Term Frequency / Inverse Document Frequency.....	41
3.5 Selecting Features Using A Boosting Approach.....	45
<b>4. Experiments.....</b>	<b>49</b>

4.1 Our Dataset.....	49
4.2 Results For The Whole Dataset Using Naïve Bayes.....	51
4.3 Feature Selection Results.....	52
4.3.1 Random Feature Selection Results.....	53
4.3.2 Chi( $\chi$ ) Square Results.....	54
4.3.3 Information Gain Results.....	56
4.3.4 Term Frequency / Inverse Document Frequency Results.....	58
4.3.5 Ada Boosting Results.....	59
4.4 Baseline Results Using Forward Selection.....	61
<b>5. Future Work.....</b>	<b>70</b>
5.1 Using Second Order Statistic.....	70
5.2 Limited Forward Selection.....	71
5.3 Genetic Algorithms.....	71
5.4 Frequency / Inverse Document Frequency (TF/IDF) Results.....	71
<b>6. Conclusions.....</b>	<b>72</b>
<b>Bibliography.....</b>	<b>73</b>

## List of Tables

Table 2.1 Contingency Tables.....	11
Table 2.2 Training examples for documents with 4 features belonging to 2 categories Classification, as the proportion, $p_{(+)}$ , of positive examples varies between 0 and 1.....	14
Table 2.3 Naive Bayes Algorithm.....	18
Table 2.4 Training examples for documents with 4 features belonging to 2 categories...20	
Table 2.5 Forward Selection Algorithm.....	30
Table 3.1 Algorithm for Random Feature Selection for Class.....	34
Table 3.2: Algorithm for Selecting Features Randomly.....	35
Table 3.3 Algorithm for Feature Selection Using Chi ( $\chi$ ) Square.....	36
Table 3.4 Algorithm for Chi ( $\chi$ ) Square Feature Selection for Class.....	36
Table 3.5 Contingency matrix.....	37
Table 3.6 Algorithm for Feature Selection using Information Gain.....	39
Table 3.7 Algorithm for Feature Selection using TF/IDF for Class.....	39
Table 3.8 Training examples for documents with 4 features belonging to 2 categories...40	
Table 3.9 Algorithm for Feature Selection using Term Frequency / Inverse Document Frequency.....	43
Table 3.10 Algorithm for Feature Selection Using Term Frequency / Inverse Document Frequency for Class.....	43
Table 3.11 Data with 3 documents.....	44
Table 3.12 Calculation Of TF/IDF Score Using Data In Table 3.12.....	44
Table 3.13 Sorted TF/IDF Score.....	45
Table 3.14 Algorithm For Feature Selection Using Ada-Boosting.....	46
Table 3.15 Algorithm for Feature Selection Using Ada Boosting for A Class.....	47
Table 3.16 Data For Ada Boosting Algorithm.....	47
Table 4.1 Description of Different Categories.....	50
Table 4.2 Results for Random Feature Selection.....	53
Table 4.3 Results for Chi ( $\chi$ ) Square Feature Selection.....	55
Table 4.4 Results for Information Gain Feature Selection.....	57

Table 4.5 Results for Frequency / Inverse Document Frequency Feature Selection.....	58
Table 4.6 Results for Ada Boosting Feature Selection.....	60
Table 4.7 Average accuracy of two-fold test performed for each best feature set that we get using Forward Selection and all the other feature selection methods.....	62
Table 4.8: Accuracy of the Feature Selection Methods evaluated using Naïve Bayes Algorithm.....	65
Table 4.9: Accuracy of the Feature Selection Methods evaluated using J48 Decision Tree Algorithm.....	66
Table 4.10 Accuracy of the Feature Selection Methods evaluated using Random Forests.....	67
Table 4.11 Accuracy of the Feature Selection Methods evaluated using Support Vector Machines.....	68
Table 4.12 Results of Five Highest Accuracy Points For the Feature Selection Methods evaluated using the ML Algorithms.....	69

## List of Figures

Figure 2.1 The entropy function relative to a boolean classification, as the proportion, $p_{(+)}$ , of positive examples varies between 0 and 1.....	13
Figure 2.2 Machine Learning method of classification.....	16
Figure 2.3 Decision tree to predict the class of a given instance based on sorting through the tree formed from the training data.....	22
Figure 2.4 SVM trained with samples from two classes.....	24
Figure 2.5 The Wrapper approach to feature subset selection. The induction algorithm is used as a “black box” by the subset selection algorithm (Kohavi & John 1997).....	28
Figure 4.1 Random Feature Selection Graph.....	54
Figure 4.2 Graph for Chi ( $\chi$ ) Square result.....	56
Figure 4.3 Information Gain Feature Selection Graph.....	57
Figure 4.4 TF/IDF Feature Selection Graph.....	59
Figure 4.5 Ada Boosting Feature Selection Graph.....	60
Figure 4.6 Results of Naïve Bayes on features selected using Forward Selection and all the other feature selection methods.....	64
Figure 4.7: Graph for Accuracy of Naïve Bayes Algorithm for Features Selected Using Proposed Feature Selection Methods.....	65
Figure 4.8 Graph for Accuracy of J48 Decision Tree Algorithm for Features Selected Using Proposed Feature Selection Methods.....	66
Figure 4.9 Graph for Accuracy of Random Forests Algorithm for Features Selected Using Proposed Feature Selection Methods.....	67
Figure 4.10 Graph for Accuracy of Support Vector Machines for Features Selected Using Proposed Feature Selection Methods.....	68
Figure 4.11 Graph Showing the Five Highest Accuracy Points For the Feature Selection Methods evaluated using the ML Algorithms.....	69

# Chapter 1

## INTRODUCTION

As the amount of information available from sources such as the internet has been growing dramatically over the past few years, it is becoming very important to extract good features which can help in classifying documents. As data increases, the number of features to be extracted also increases. The accuracy of any retrieval method is very much dependent on features because the method uses information about the class (category of the document) that is inherent in the features. Thus, the need to efficiently and accurately retrieve relevant information has become a topic of great interest. This has led to lot of interest in the field of Information Retrieval (IR). Salton (1968) defined information retrieval as being “a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information.” The definition itself explains that it is extremely important to put the data in a proper structure so as to organize and store the documents in such a way that it facilitates efficient and accurate searching and analysis.

In order to put the raw data into a good structure, it is very important to extract meaningful features. Features are useful elements present in a document which help in classifying or identifying a given document from a large set of documents. For example, if a document belongs to the category *Cartoon*, then words, such as *Tom*, *Jerry*, *Mickey*, *Goofy* and *Animation* would be useful features for classifying the document to the *Cartoon* category. Forward Selection (Miller, 1990) is a well known technique for providing very good features. Forward Selection (which we detail in Chapter 2) involves starting with no variables in the model, trying out the variables one by one and including them if they are 'statistically significant'. This method, while effective, becomes computationally intractable for large data sets. In this thesis, we have proposed some methods for approximating Forward Selection of features in information retrieval problems using Machine Learning (ML) methods.

Theoretically, having more features should produce a better model and greater accuracy. However, this is not the case in real world. Machine learning algorithms are generally time consuming and computationally expensive. Also the time requirements generally grow dramatically with increase in the number of features. This might also lead to a lot of irrelevant features which could puzzle the learning algorithm by obscuring the actually relevant features and it might also lead to overfitting. Thus the need for selecting meaningful and relevant features is very important.

There is no easy way to determine which features are meaningful. One very good method would be to use human intervention; people could be asked to pick features which are meaningful for the documents. This might help in better classification of the documents, but it is not a practical approach in real world because of the size of the data. Moreover, when human decisions are involved, it might lead to inconsistent selection of features caused by human errors. Machine learning can be used in order to automate this process and get consistent and good results. In a classic machine learning task, there is a training set of labeled feature vectors (or instances), from which a classification model is induced. This model is then used to predict the class label for a set of previously unseen instances. Machine learning plays a very important role in automating the process of deciding whether a word (or a feature) is important or not. Given a set of features and some information about the instances such as the target attribute to which they belong, machine learning algorithms can give back a set of very good features. Hence, if machine learning methods are used initially to determine the attributes that are ‘relevant’ to the target concept, it could save a lot of computation time and resources.

Documents are often represented by a set of words which are nothing but the features which determine the content and classification of the document. Thus, depending on the size of the document the number of features associated with it might vary. But not all of these features are important for the categorization of the document. These features generally have to be filtered to remove extraneous features, for example, removing words which are extremely common, such as *the*, *is*, *a*, *then* which do not contribute in

distinguishing the documents. In addition, approaches such as stemming are used. Stemming is a process of reducing a word to its stem or root form. This means that different variants of a term can be conflated to a single representative form thus leading to decrease in the number of distinct terms. For example, *computation* might be stemmed to *comput*. Further, the frequency of a word might also help in reflecting the importance of a word. For example, if a word occurs only once in a document then it can be filtered out initially because it would not be important for distinguishing the document. But even after all this filtering and pruning of words a huge amount of features are found. Thus, this leads to more of time and resource consumption with no guarantee of increase in the accuracy of the methods used for information retrieval and classification.

The basic idea behind feature selection is to focus on a subset of features from the available features which seem to be most relevant for the target attribute, or in other words which contribute in classifying the documents and ignoring those input features which have little effect on the output. There are various methods for selecting this subset of important features which are under research such as the Wrapper methods (Kohavi and John, 1997), filter methods (John, Kohavi and Pfleger 1994) and weighing methods which are known for its ability of selecting good and meaningful features. Also many of the statistical approaches are used in order to filter out the good features.

## **1.1 Thesis Statement**

In this thesis, we have proposed methods for approximating Forward Selection of features in information retrieval problems using machine learning methods. Forward Selection's approach for selecting features is computationally expensive and also makes use of large number of computer resources. This cost increases dramatically as the number of features increases. Thus, we have proposed various machine learning methods that focus on the most relevant features for use in representing the data, and the problem of selecting the most relevant examples to drive the learning process. The objective of feature selection is improving the prediction performance of the predictors or the learning

algorithms, providing faster and more cost-effective predictors, and providing a better understanding of underlying data.

## **1.2 Thesis Outline**

This thesis is organized as follows. Chapter 2 presents the background information for this thesis with a detailed description of the various information retrieval problems. It describes the basic concepts of machine learning and presents various machine learning methods that are known for contributing in selection of good features. It also contains a detailed description of some feature selection ideas such as Wrappers and Forward Selection. Chapter 3 presents detailed description of our methods such as Random Feature Selection, Chi ( $\chi$ )-square Feature Selection, Information Gain Feature Selection, feature Selection using Term Frequency / Inverse Document Frequency (TF/IDF) statistic and method using a Boosting-based approach. Chapter 4 describes the data that we have used for the experiments and presents the experiments we carried out in order to compare the accuracy of our approach along with some baseline results with Forward Selection. Chapter 5 discusses future work that can be done and Chapter 6 presents conclusions and summarizes the work done in this thesis.

## **CHAPTER 2**

### **BACKGROUND**

This chapter presents the background information for this thesis. In the first section, we introduce information retrieval problems and the concept of term frequency/inverse document frequency (TF/IDF) weighting technique. In the second section, we present basic concepts of machine learning and their application in information retrieval problems. We then present the various machine learning algorithms used in this thesis. In the third section, we introduce well known feature selection methods which lead to accuracy in selecting good features.

#### **2.1 Information Retrieval Problems**

In recent years, with the enormous increase in the number of text databases available online, and the consequent need for better techniques to access this information, there has been strong interest in research done in the area of information retrieval (IR). Information retrieval can be defined as the technique and process of searching, recovering, and interpreting information from large amounts of stored data. The most important issues with information retrieval are the analysis and measurement of the relevance of the stored information.

The general information retrieval problem can be defined as the process of finding a set of documents relevant to a given subject of interest, where a relevant document refers to a document which contains information that is related to what is being searched in the query. For example, we might want to find articles about heart disease and query with a set of terms such as heart, cardiac, pectoral, disease and problems might not be relevant to the query “Heart disease problems”. This is because, there might be documents

containing words such as *heart* and *problem* but are actually not at all related to medical heart problems. This has lead researchers to do a lot of research in the measurement of relevance of the stored information; here relevance refers to the relation between the requested information and the retrieved information. There are various ways of getting relevant data, for example, using various weighing schemes and organizing the data in a particular format so that retrieval becomes faster and more effective. But with the increase in the amount of data, the storage of massive data creates a problem in effective retrieval. This also causes an increase in time and processing power. For example, if we have a very small data set, then it would be comparatively simple to index the data in such a way that we can retrieve the information that we are looking for. But if the collection is very big then it would get difficult to retrieve information from it because the number of features would increase and most of them might not even be relevant. It would need more processing power and time to identify the documents which are most relevant to the information being queried. Thus analyzing the data to get relevant information would be far more difficult.

Thus it is very important to reduce the amount of stored data which is analyzed in order to fetch the documents relevant to a query of interest. This can be done by filtering out features which do not contribute to the classification of the document. This thesis experiments with various techniques for selecting good features for effective retrieval of information.

### **2.1.1 Term Frequency / Inverse Document Frequency (TF/IDF)**

In Information Retrieval, Term Frequency / Inverse Document Frequency (TF/IDF) is a widely used weighing scheme which we have used in order to get features that are most relevant to a document. For each word in a document, TF/IDF calculates values as the product of the frequency of the term in the document times the inverse document frequency. Words which have higher TF/IDF numbers imply a strong relationship with

the document in which they appear. The weight that we get is a statistical measure used to evaluate how important a word is to a document in a collection or a corpus. The importance increases proportionally to the number of times a word appears in the document but is the offset by the frequency of the word in the corpus. For example, if the frequency of a word in a document is high and it occurs in less number of documents in the corpus, then clearly that word is a good classifier for the documents in which it occurs. Whereas, if there is a word whose frequency is same as the previous word but it occurs in almost all the documents in the corpus, then it is more likely to be a common word which might not contribute to the classification of the documents. For example, suppose we are looking for documents which belong to the category *archeology* in a collection of 100 documents belonging to categories *archeology*, *cartoon*, *history*, *movies* and *geography*. If the word *excavation* occurs in 10 documents and *past* occurs in 60 documents, then clearly *excavation* would be a better classifier for the category archeology than the word *past* because of the large number of documents in which it occurs signifies that it a common word for most of the categories and thus would not be a good distinguisher of one particular category.

The term frequency in the given document is simply the number of times a given term appears in that document. This count is usually normalized to prevent a bias towards longer documents. For instance, if the word *paper* occurs 100 times in a document which contains 1000 words, then its frequency would anytime be greater than a word which occurs 10 times in a document which has 50 words, even though the second word is clearly a better distinguisher of the documents in which it occurs. The measure of the importance of the term  $t_i$  within the particular document  $d_j$  is given by

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

where  $n_{i,j}$  is the number of occurrences of the considered term in document  $d_j$ , and the denominator is the number of occurrences of all terms in document  $d_j$ .

The inverse document frequency is a measure of the general importance of the term (obtained by dividing the number of all documents by the number of documents containing the term, and then taking the logarithm of that quotient).

$$idf_i = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

With,

- $|D|$  : total number of documents in the corpus
- $|\{d_j : t_i \in d_j\}|$  : Number of documents where the term  $t_i$  appears (i.e.,  $n_{i,j} \neq 0$ ).

Then,

$$tfidf_{i,j} = tf_{i,j} \cdot idf_i.$$

Words that are common in a single or a small group of documents tend to have a higher TF/IDF numbers. For example, suppose we have a feature that occurs only 3 times in a document out of 100 features and it is present in only 1000 document out of 100,000 documents. Then its term frequency can be calculated as  $3/100 = 0.03$  and the inverse document frequency is  $\ln(100,000/1000) = 4.60517019$ . Thus, the TF/IDF score would be  $0.138155106$ . Whereas, if we have a feature that occurs 3 times in a document out of 100 features and it is present in 3 document out of 100,000 documents, then its term frequency can be calculated as  $3/100 = 0.03$  and the inverse document frequency is  $\ln(100,000/3) = 10.4143132$ . Thus, the TF/IDF score would be  $0.312429396$ . Clearly, if we had to choose amongst these two words manually, then we would have chosen the second one because it occurs in a small number of documents in the corpus even though its frequency was same as the first word. Hence, we can say that it is a better classifier of the documents. The TF/IDF results also show that the second word is a better choice for classification of the documents. We exploited this fact because such terms would definitely play a very important role in identifying a document as to which class it belongs to. We thus used it in order to find good features which would be of utmost

importance for a document and might prove to help in classification.

## 2.2 Statistical Ideas

Statistic is a mathematical science pertaining to the collection, analysis, interpretation or explanation, and presentation of data. The field of machine learning and statistic overlap with each other. Given the enormous growth of collected and available data in literally every field, techniques for analyzing such data are becoming ever more important. Statistic plays a very important role in processing large data sets whose sizes and complexities are beyond the ability of humans to handle.

Statistical methods can be used to model the patterns hidden in data and then used to draw inferences about the process or data being studied. Statistical ideas have long been used in machine learning in order to infer knowledge from data. For instance, if we are given a huge database consisting of the medical history of millions of people, or even if the data set contains information of a thousand people, extracting the required information from it would be like finding a needle in the haystack. For example, if we need to know what are the possible symptoms which might cause cancer then it would be very difficult to analyze the data manually and it would take a lot of resources, time and money. But with the increase in computation power, it would be simple to use some statistical method to reveal the hidden patterns pertaining to the symptoms that generally lead to cancer.

In our work we have used the statistical notions such as the  $\chi$ -square statistic in order to gather the information that we need from the huge amount of data. These methods analyze the hidden relation and level of importance of the features with the documents. They provide a statistical approach to reveal the strong correlation of the features and the data set in order to help in the classification of the documents using the features it contains.

## 2.2.1 The Chi-Square Statistic

The Chi-Square ( $\chi^2$ ) statistic is used to investigate whether distributions of categorical variables differ from one another. It tests a null hypothesis that the relative frequencies of occurrence of observed events follow a specified frequency distribution. The events are assumed to be independent and have the same distribution, and the outcomes of each event must be mutually exclusive. We have used Pearson's  $\chi$ -Square model given below.

### 2.2.1.1 Definition of Karl Pearson statistic

A table used to record and analyze the relationship between two or more variables is called a contingency table. Let the two rows of a 2 \* 2 contingency table be denoted as  $i=1$  and  $i=2$ . Let the columns be denoted by  $j = 1$  and  $j = 2$ . Let the observed frequencies  $f_{ij}$  of the corresponding cells be denoted by  $f_{11}$ ,  $f_{12}$ ,  $f_{21}$  and  $f_{22}$ . Let the corresponding estimated expected frequencies be denoted by  $F_{11}$ ,  $F_{12}$ ,  $F_{21}$  and  $F_{22}$ . In terms of these numbers, the Karl Pearson statistic is defined by:

$$\chi^2 = \sum_{i=1}^2 \sum_{j=1}^2 \frac{(f_{ij} - F_{ij})^2}{F_{ij}}$$

$\chi$ -Square is calculated by finding the difference between each observed and expected frequency for each possible outcome, squaring them, dividing each by the expected frequency, and taking the sum of the results. The  $\chi$ -Square statistic of a term is calculated by using the following contingency tables:

**Table 2.1: Contingency Tables**

	Class	~Class	Total
Feature	a	c	a+ c
~Feature	b	d	b+d
	a+b	c+d	a+b+c+d

	Class	~Class	Total
Feature	A	C	a+c
~Feature	B	D	b+d
	a+b	c+d	a+b+c+d

Where:

- a: the observed number of times t and c co-occur,
- b: the observed number of times the t occurs without c,
- c: the observed number of times c occurs without t,
- d: the observed number of times neither c nor t,
- n: the total number of documents.

Where, the expected frequencies A, B,

C and D are calculated as follows:

- $A = ((a+b)/n) * (a+c)$
- $B = ((a+b)/n) * (b+d)$
- $C = ((c+d)/n) * (a+c)$
- $D = ((c+d)/n) * (b+d)$
- n = the total number of documents.

From contingency tables shown in Table 2.1, the  $\chi$ -Square statistic of a word is estimated by the following formula:

$$\chi^2 = \frac{(a - A)^2}{A} + \frac{(b - B)^2}{B} + \frac{(c - C)^2}{C} + \frac{(d - D)^2}{D}$$

We have used Karl Pearson's model for  $\chi$ -Square method for selection of good features for our experiments.

### 2.2.2 Information Gain

Information gain is a statistical property that measures how well a given attribute

separates the training examples according to their target classification (Mitchell 1997). Before we study about information gain, it is important to learn about the entropy statistic. Entropy characterizes the (im)purity of an arbitrary collection of examples. Given a collection  $S$ , containing positive and negative examples of some target concept, the entropy of  $S$  relative to this boolean classification is given by

$$Entropy(S) = \sum_{i=1}^{numclasses} -p_i \log_2 p_i$$

Where  $p_i$  is the proportion of instances in  $S$  that have the  $i^{th}$  class values as output attribute. Entropy, in information theory, is known for specifying the minimum number of bits of information needed to encode the classification of an arbitrary number of  $S$ . The entropy value ranges between 0 and 1. Figure 2.1 shows the form of the entropy function relative to a boolean classification, as  $p_{(+)}$  varies between 0 and 1.

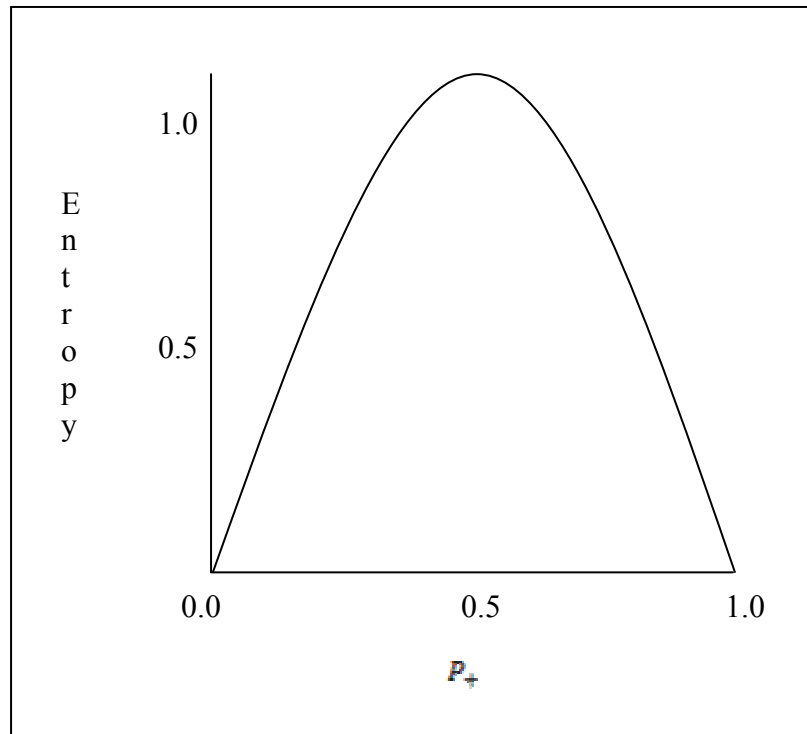
Now, if we know the measure of impurity in a collection of training examples, we can define a measure of the effectiveness of an attribute in classifying the training data. The measure that we have used here is nothing but the information gain which is the expected reduction in entropy caused by partitioning the examples according to this attribute.

More precisely, the information gain,  $InformationGain(S,A)$  of an attribute  $A$  relative to a collection of examples  $S$ , is defined as

$$InformationGain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \left[ \frac{|S_v|}{|S|} Entropy(S_v) \right]$$

Where  $Values(A)$  is the set of all possible values for attribute  $A$  and  $S_v$  is the subset of  $S$

for which attribute  $A$  has value  $v$ . The value  $InformationGain(S,A)$  is therefore the expected reduction in entropy caused by knowing the value of attribute  $A$ .



**Figure 2.1 The entropy function relative to a boolean classification, as the proportion,  $p_{(+)}$ , of positive examples varies between 0 and 1**

For example, refer to the data given below in Table 2.2. The data given consists of 14 documents belonging to 2 categories having 4 features each. For instance, the information gain due to sorting the original 14 examples by the attribute *Feature 4* may be calculated as:

$$\begin{aligned}
 Entropy(S) &= - (9/14) \log_2 (9/14) - (5/14) \log_2 (5/14) \\
 &= 0.940
 \end{aligned}$$

$$\text{Values (Feature 4)} = 0, 1$$

$$S = [9+, 5-]$$

$$S_0 <- [6+, 2-]$$

$$S_1 <- [3+, 3-]$$

$$\begin{aligned} \text{Gain}(S, \text{Feature 4}) &= \text{Entropy}(S) - (8/14) \text{Entropy}(S_0) - (6/14) \text{Entropy}(S_1) \\ &= 0.940 - (8/14)0.8411 - (6/14)1.00 \\ &= 0.048 \end{aligned}$$

**Table 2.2 Training examples for documents with 4 features belonging to 2 categories**

Document	Feature 1	Feature 2	Feature 3	Feature 4	Category
D1	0	1	1	0	Class 2
D2	1	1	1	1	Class 2
D3	0	0	1	0	Class 1
D4	1	1	1	0	Class 1
D5	1	1	0	0	Class 1
D6	1	1	1	1	Class 2
D7	1	1	1	1	Class 1
D8	0	0	0	0	Class 2
D9	1	0	0	0	Class 1
D10	1	0	0	0	Class 1
D11	1	0	1	1	Class 1
D12	1	0	1	1	Class 1
D13	0	0	1	0	Class 1
D14	1	1	1	1	Class 2

The higher the information gain, the greater is its capability of separating the training examples according to their target classification. Thus, in the above mentioned way we calculate the information gain for each attribute and select the one which has the highest information gain. For example, if we had to choose 1000 features, then we calculate the information gain of all the possible features and select the first 1000 features having the highest information gain.

## 2.3 Machine Learning

Learning can be defined as a continuous process of acquiring knowledge through experience. In our day to day life we go through the process of learning. We observe things, people and the changes in our surrounding and adapt to them by changing our behavior in order to survive. In doing so, we make mistakes and learn from them so that we get better and improve ourselves. In this process we could learn ourselves or else be guided by someone. A more concrete definition of learning which is widely accepted is:

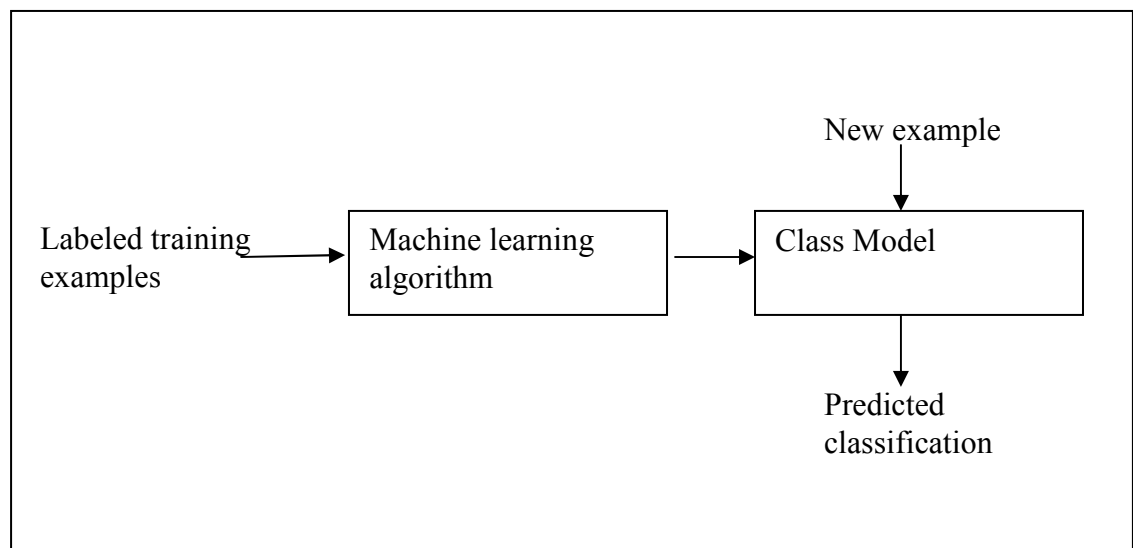
*“Learning is a process in which behavior capabilities are changed as the result of experience, provided the change cannot be accounted for by native response tendencies, maturation, or temporary states of the organism due to fatigue, drugs, or any other temporary factors”* (Runyan, 1977).

Machine Learning (ML) is a sub-field of Artificial Intelligence (AI) concerned with the design, development, understanding and evaluation of algorithms and techniques to allow a computer to learn. Artificial intelligence is a field of computer science whose objective is to develop machines that exhibit intelligent behavior in the tasks it performs. A system is said to be intelligent if it can learn to perform a task assigned to it without any human interaction effectively and accurately. As said by Selfridge (1993):

*“Find a bug in a program, and fix it, and the program will work today. Show the program how to find and fix a bug, and the program will work forever.”*

ML is an effort towards building systems that can learn continuously, self-improve and thereby offer increased efficiency and effectiveness. For example, suppose we want to design a robot which can navigate through a room or perform some task assigned to it such as picking up some things lying on the floor. It would need to learn to navigate through its environment without colliding with external objects. It should also learn to pick up things from the floor. ML algorithms can be used in order to teach the robot to do the above mentioned task. Given all the scenarios, say for example, the kind of object it has to pick up and the layout of the room, the robot can be trained to learn to navigate through the room and pick up objects from the floor. Now, after the robot is trained it can perform the same task in some other location using the information it gathered during the training process.

One ability of ML to automatically learn to make accurate predictions based on past observations makes it very suitable for text classification. Figure 2.2 describes this general process in ML which is called classification.



**Figure 2.2 Machine Learning method of classification**

Figure 2.2 shows that, given a set of labeled documents, the system can learn to build a model that can classify unseen examples based on the classifier that it creates. Thus,

resources and time wasted on manual work can be saved.

### 2.3.1 Classification

ML algorithms that deal with classification of a given instance into a set of discrete categories are called classification algorithms. Given a labeled set of data provided for learning, these algorithms create a model or a set of rules which further help in classifying unseen instances to some target value. There are various classification algorithms such as Naïve Bayes and Decision trees (Mitchell 1997, Witten and Frank, 2005).

After we select the features, in order to validate their goodness, we use some classification algorithms such as Simple Count, Naïve Bayes, J48 Decision Trees, Random Forests and Support Vector Machines (SVM) (Vapnik, 1995) to calculate the accuracy of our selected features. We use WEKA (Witten and Frank, 2005) for the J48 Decision Trees method, Leo Breiman's Random Forests (Breiman, 2001) code ([www.stat.berkeley.edu/~breiman/RandomForests/cc\\_software.htm](http://www.stat.berkeley.edu/~breiman/RandomForests/cc_software.htm)) and SVM<sup>light</sup> software (Vapnik, 1995) for getting out SVM results. In order to use these softwares, we have to change our data in the format required by each software. The following subsections describe the classification algorithms that have been used in this thesis.

### 2.3.2 The Naive Bayes Classifier

Bayes' rule states that

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where  $P(A|B)$  is defined as the probability of observing  $A$  given that  $B$  occurs.  $P(A|B)$  is called posterior probability, and  $P(B|A)$ ,  $P(A)$  and  $P(B)$  are called prior probabilities.

Bayes' theorem gives a relationship between the posterior probability and the prior probability. It allows one to find the probability of observing  $A$  given  $B$  when the individual probabilities of  $A$  and  $B$  are known, and the probability of observing  $B$  given  $A$  is also known.

Naive Bayes (Good, 1965; Langley et al., 1992) is a simple probabilistic classifier based on Bayes' rule. Bayes' theorem provides a way to calculate the probability of a hypothesis based on its prior probability. It uses all attributes and allows them to make contributions to the decision as if they were all equally important and independent of one another. Given a set of attributes as input it predicts the output value, which is one of the target attributes. The Naïve Bayes algorithm is given in Table 2.3.

---

**Table 2.3: Naive Bayes Algorithm**

---

Naive\_Bayes\_Learn(examples)

  For each target value  $v_j$

$$\hat{P}(v_j) \leftarrow \text{estimate } P(v_j)$$

  For each attribute value  $a_i$  of each attribute  $a$

$$\hat{P}(a_i | v_j) \leftarrow \text{estimate } P(a_i | v_j)$$

Classify\_New\_Instance(x)

$$v_{NB} = \arg \max_{v_j \in V'} \hat{P}(v_j) \prod_{a_i \in X} \hat{P}(a_i | v_j)$$


---

Naive Bayes classifiers are among the most successful known algorithms for learning to classify text documents. It takes in a set of training examples as input, in our case each of the training examples are represented by documents with the terms occurring in the document as features or attributes which have been selected by one of the feature selecting methods. To classify a new instance given to it using the Bayesian approach, the Bayes rule is applied to find the probability of observing each output class given the input attributes. The class which has the highest probability is assigned to the instance as the

target class. The probability values used in this are obtained from the counts of the attribute values seen in the training set.

In the case of the data set we explore, all the documents belong to one of the 21 categories. The Naive Bayes' classifier selects the most likely classification amongst the 21 categories given a set of attributes which are the features selected from one of the above mentioned algorithms. The Naïve Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. It applies learning tasks where each instance  $x$  is described as a conjunction of attribute values and where the target function can take on any value from some finite set of values. Learning in the Naive Bayes algorithm involves finding the probabilities of  $P(v_j)$  and  $P(a_i|v_j)$  for all possible values taken by the input and output attributes based on the training set provided.  $P(v_j)$  is obtained from the ratio of the number of time the value  $v_j$  is seen for the output attribute to the total number of instances in the training set. For an attribute at position  $i$  with value  $a_i$ , the probability  $P(a_i | v_j)$  is obtained from the number of times  $a$  is seen in the training set when the output value is  $v$ . Thus we get the Naïve Bayes classifier equation:

$$v_{NB} = \arg \max_{v_j \in V} \hat{P}(v_j) \prod_{a_i \in x} \hat{P}(a_i | v_j)$$

where  $V_{NB}$  denotes the target value output by the Naïve Bayes classifier. The  $P(a_i | v_j)$  is generally estimated from the training data as the number of distinct attribute values for a target times the number of distinct target values. For example  $P(\text{Feature1} | \text{Class1})$  would be number of times Feature1 occurs for Class1 ( $n_c$ ) by the number of times Class1 occurs ( $n$ ). This fraction does provide a good estimate of the probability in many cases but sometimes it provides a poor estimate if  $n_c$  is very small. This causes two main problems, firstly  $n_c/n$  provides a biased underestimate of the probability and secondly, when the probability estimate is zero, this probability term dominates the Bayes classifier if the future query contains that particular feature value. To avoid these difficulties we have adopted a Bayesian approach to estimating the probability. We estimated  $P(a_i | v_j)$  using m-estimate which is defined as :

$$P(a_i/v_j) = \frac{n_c + mp}{n + m}$$

where  $n$  is the number of training examples for which the class  $v$  is  $v_j$ ,  $n_c$  is the number of examples for which class  $v$  is  $v_j$  and the attribute  $a = a_i$ ,  $p$  is a prior estimate for  $P(a_i | v_j)$  and  $m$  is the equivalent sample size which determines how heavily to weight  $p$  relative to the observed data.

**Table 2.4 Training examples for documents with 4 features belonging to 2 categories**

Document ID	Feature 1	Feature 2	Feature 3	Class
D1	1	0	1	Class 1
D2	1	0	1	Class 0
D3	1	0	1	Class 1
D4	0	0	1	Class 0
D5	0	0	0	Class 1
D6	0	1	0	Class 0
D7	0	1	0	Class 1
D8	0	1	1	Class 0
D9	1	1	0	Class 0
D10	1	0	0	Class 1

For example, we have documents represented as instances which has the attributes which are the selected features represented in the form of 0's and 1's which states whether a feature is present in a document or not. Given this document, the Naive Bayes' algorithm uses the prior probability calculated for each feature of a document belonging to a particular class which was calculated during training. Using these prior probabilities it calculates the probability of each class and assigns the class with the highest probability to the training document. This is known as the maximum a posteriori (MAP) rule.

For example, consider the data given in Table 2.4 which has 10 documents having 3 features each belonging to 2 classes.

Suppose we have to classify an instance given below:

DocumentID	Feature1	Feature2	Feature3
2347	1	1	1

Thus we would need to calculate probabilities  $P(\text{Feature 1} | 1)$  ,  $P(\text{Feature 2} | 1)$  ,  $P(\text{Feature 3} | 1)$  ,  $P(\text{Feature 1} | 0)$  ,  $P(\text{Feature 2} | 0)$  and  $P(\text{Feature 3} | 0)$  and then multiply them by  $P(\text{Class 1})$  and  $P(\text{Class 2})$ . Here we have considered the  $m$  value as 3. Both the Classes occur 5 times, thus  $P(\text{Class 1})$  and  $P(\text{Class 2})$  would be 0.5. We calculate rest of the probabilities as follows

$$P(\text{Feature 1} | 1) = (3 + (3*5)) / (5+3) = 0.56$$

$$P(\text{Feature 2} | 1) = (1 + (3*5)) / (5+3) = 0.31$$

$$P(\text{Feature 3} | 1) = (2 + (3*5)) / (5+3) = 0.43$$

$$P(\text{Feature 1} | 0) = (2 + (3*5)) / (5+3) = 0.43$$

$$P(\text{Feature 2} | 0) = (3 + (3*5)) / (5+3) = 0.56$$

$$P(\text{Feature 3} | 0) = (3 + (3*5)) / (5+3) = 0.56$$

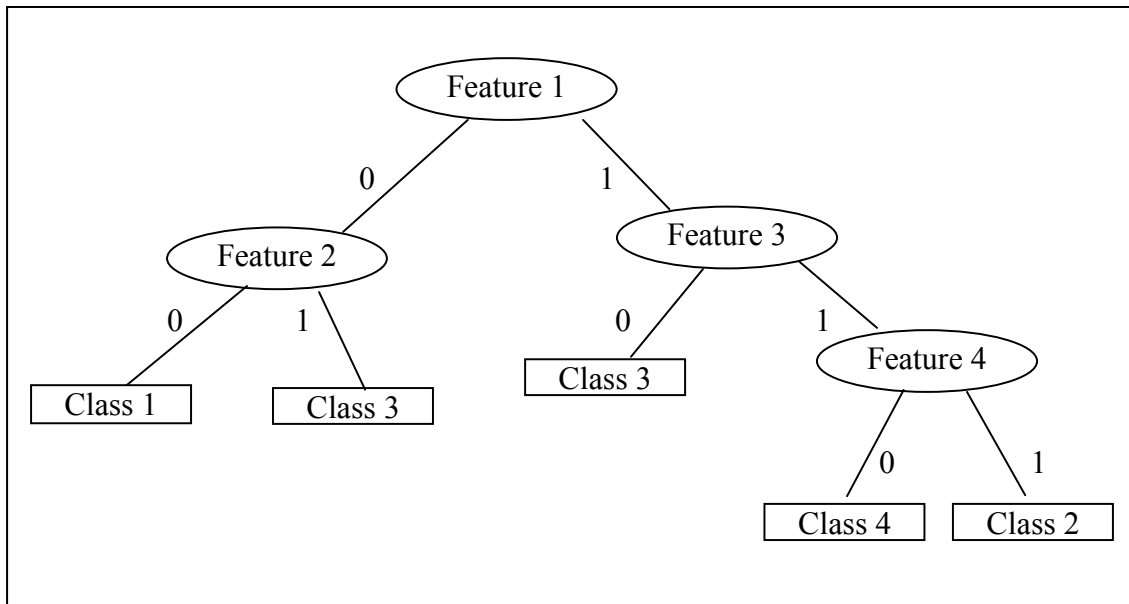
After getting these probabilities we calculate

$$\begin{aligned} &P(\text{Class 1}) * P(\text{Feature 1} | 1) * P(\text{Feature 2} | 1) * P(\text{Feature 3} | 1) \\ &= 0.5 * 0.56 * 0.31 * 0.43 = 0.037 \end{aligned}$$

$$\begin{aligned} &P(\text{Class 0}) * P(\text{Feature 1} | 0) * P(\text{Feature 2} | 0) * P(\text{Feature 3} | 0) \\ &= 0.5 * 0.43 * 0.56 * 0.56 = 0.069 \end{aligned}$$

Now, since  $0.069 > 0.037$  we classify the document as *Class 0*.

### 2.3.3 A Decision Tree Classifier



**Figure 2.3 Decision tree to predict the class of a given instance based on sorting through the tree formed from the training data**

Decision trees are one of the most commonly and widely used methods for classification such as C4.5 (Quinlan, 1993) and Id3 (Quinlan, 1986). Decision trees are graphical structures with nodes and branches. Each node represents an attribute and each branch descending from the node represents one of the possible values of the attribute. The leaf nodes represent the target function. For classifying an instance, the tree sorts the instance starting at the root node. The tree tests the attribute specified by this node and moves down the branch corresponding to the value of the attribute in the given instance. Similarly, it sorts the instances down the tree at each node that it reaches and the terminal node that it reaches decides the class of the instance. Our data was represented in the form of a document vector with a value 0 or 1 for each feature corresponding to whether the feature was present or not in the document respectively. Figure 2.3 shows an example of a tree.

In Figure 2.3, we can see how the tree would look like if formed from our data. Each

non-terminal node represents a feature ID and the branches descending from it represents the value of the node. In our case the values are always 0 or 1 which represents whether the feature is present or not in the document. The leaf nodes represent the class. The data that we used is formed of 21 classes but in the diagram we have just shown 4 classes for simplicity. This tree is formed while training the system using the training set. Once the tree is formed, given an unseen instance it is sorted down the tree in order to get the class of the instance. For example if we want to classify an unseen instance using the tree that is shown in Figure 2.3. The unseen instance is in the format given below:

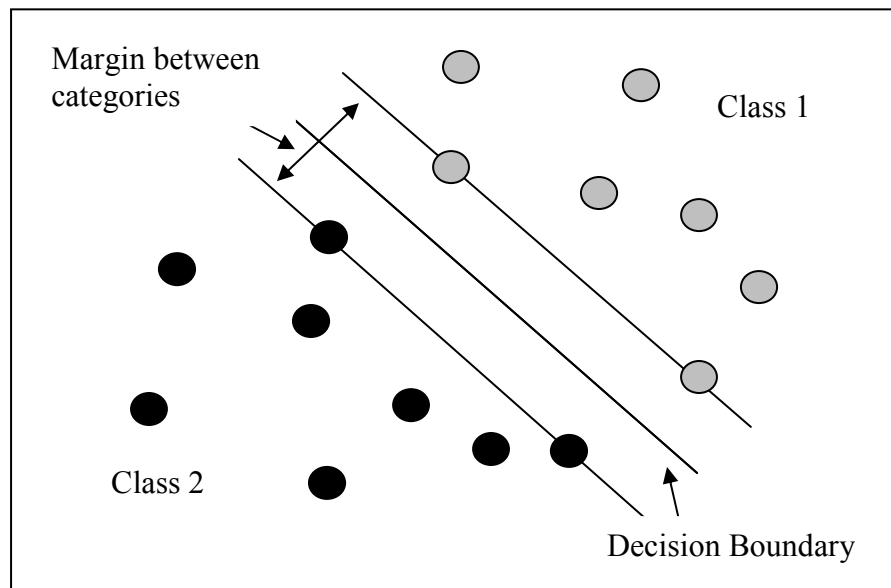
DocumentID	Feature1	Feature2	Feature3	Feature4
2347	1	0	1	0

For classifying this instance we would start at the root node which is Feature1 and test its value given in the unseen instance which is 1. Thus we would move down to Feature3 and repeat the process of testing. Since Feature3 has value 1 we move down to Feature4 and test its value which is 0. Thus we end up at the leaf node having the label Class 4. Thus the instance is classified as Class 4.

For our experiments, we used the J48 Decision Trees algorithm which is available in the WEKA collection of machine learning algorithms (Witten and Frank,2000). J48 Decision Trees is a WEKA implementation of a decision tree learner based on C4.5 (Quinlan, 1993). The tree is formed using a splitting criterion such as *Information Gain* or *Gain Ratio* (Quinlan, 1986) which measures how well a given attribute separates the training examples according to the target classification. At each of the tree building steps, the value of the splitting criteria is calculated for each of the node which has not been already added to the tree. The node with the highest value is chosen to be added to the tree at that particular step. Thus the root node is the feature which separates the data best.

### 2.3.4 Support Vector Machines (SVMs)

Support Vector Machines (SVMs) (Vapnik, 1995) are classifying algorithms based on phrasing a classification or regression problem as an optimization problem. SVMs are known to be a very good classification mechanism for text processing. But SVMs are limited to binary classification, which means at a time it can classify the instances into only two target functions. In general, multi-class functions involving 3 or more classes are solved as a set of binary problems (class 1 vs the remaining classes, class 2 vs the remaining classes, etc.). Given a set of data it chooses a separating plane based on maximizing the notion of a margin based on Probably Approximately Correct (PAC) learning. SVMs represent a data point as a p-dimensional vector and there might be many hyperplanes which might classify the data. SVM picks the best hyperplane in such a way that the distance from the hyperplane to the nearest data points is maximized (basically it tries to maximize the "distance" between the two classes of points).



**Figure 2.4: SVM trained with samples from two classes**

For example, if we have to classify a given set of instances which might belong to two

possible target functions, SVM learning method would create a model using the training instances whose target function is already known. It would then find a separating plane having the maximum margin which best divides the data. This margin serves as the basis for classification. For example in Figure 2.4 we can see that the instances are divided using a separating plane which maximizes the margin. For instance, if we have an unseen instance which lies in the side of the separating plane in which the black solid circles are there, then the instance would be classified as class 2.

For our experiments we have used the software SVM<sup>light</sup> (Joachims, 2002a, Joachims, 1999a). SVM<sup>light</sup> is an implementation of Vapnik's SVM (Vapnik, 1995) in C. It consists of two programs, *svm\_learn* for training and *svm\_classify* for classification. Since SVM is a binary classifier and the data that we used has 21 classes we work with 2 classes at a time. We find the accuracy of the documents belonging to a class with respect to all the other documents which do not belong to this class. For example, we train the system with a data set made up of the members of class 1 labeled as positive and members of all the other classes labeled as negative. We then repeat this process for each of the other classes to get 21 partial models. To label a new point we apply each of the 21 models and pick the one with the highest output as the class. To use the SVM<sup>light</sup> system we had to convert our data into SVM<sup>light</sup> format.

### 2.3.5 Boosting Classifiers

Boosting is a general method which attempts to boost the accuracy of any given learning algorithm. The main idea of Boosting (Freund and Schapire, 1996; Schapire, 1990) is to generate many, relatively weak classification rules and to combine these into a single highly accurate classification rule. It is a popular method to create accurate Ensembles. Ensemble methods are learning algorithms that construct a set of classifiers and then classify new data points by taking a weighted vote of their prediction as given by Hansen and Salamon (1990) is as follows:

*“If we assume classifiers are random in predictions and accuracy greater than 50%, then the accuracy can be pushed arbitrarily high by combining more classifiers”*

We have used boosting mechanisms as one of our method for selecting good features. We used the concept of weak learning (Schapire, 1990), which states that a set of weak learners (learners with greater than 50% accuracy, but not greater) could be combined into a strong learner. The algorithm that we have presented in Chapter 3 is based on the idea in which we weight the data set based on how well it was predicted before. Initially, all the data sets are initialized to negligible weight. Slowly, the weights are updated as per the predictions made. The data sets which were predicted accurately before have very low weight at the end and the data sets which were mispredicted have a very high weight.

### **2.3.6 The Random Forest Classifier**

Random Forests (Breiman 2001) are an ensemble of individual tree predictors which are not influenced by each other when constructed. It is formed of many decision trees. For example, in order to classify a new data points, the data point is evaluated by each of the trees in the forest. While doing so, each tree gives a prediction which is stored. These classifications are considered as votes for the class to which the instance was classified. The forest chooses the classification having the most votes. Random Forests differ from the Bagging method (Breiman,1996a) in that in Bagging each tree would simply be constructed from a random bootstrap sample of the original data. In random forests estimates of the correlation with the other trees and variables are used to inject randomness into the resulting trees. In this way, the trees are more likely to be independent in their predictions. One advantage of this approach is that for each of the trees the unused (out-of-bag) examples can be used to estimate errors and internal correlation amongst the trees.

In it each class is represented as an increasing number. In order to grow a tree which has  $N$  cases in the training set then the  $N$  cases are sampled at random with replacement from the original data. This sample is used as the training set for growing the tree. If there are  $M$  input variables, then a number  $m \ll M$  is specified such that at each node,  $m$  variables are selected at random out of the  $M$  and the best split on these  $m$  is used to split the node. This value  $m$  is held constant throughout the forest growing process. Each tree is grown to the largest extent possible and there is no pruning.

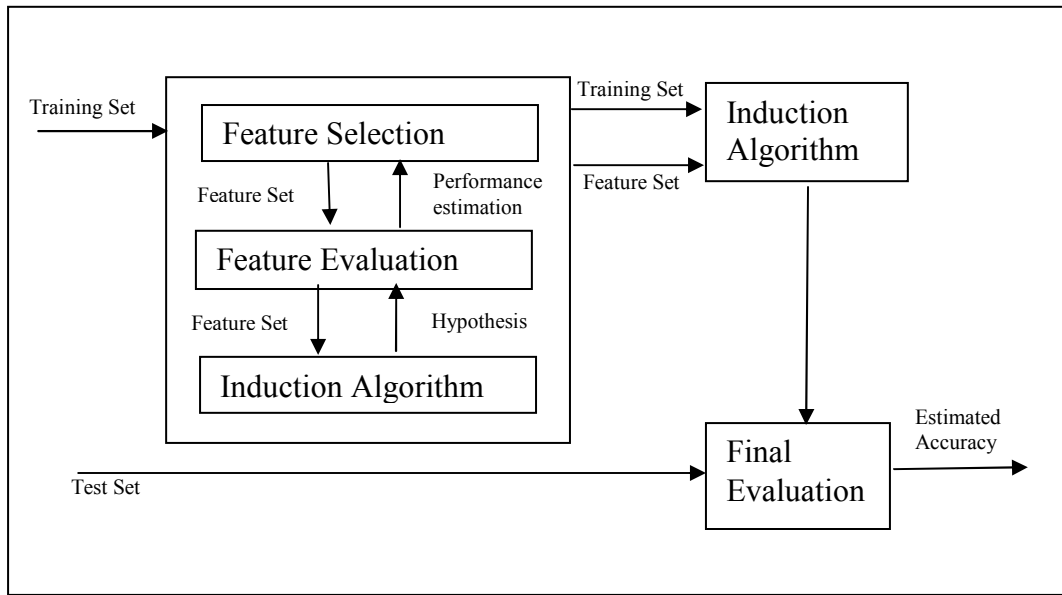
## **2.4 Feature Selection Methods**

Feature selection is a process in which a subset of the features available from the data is selected for application of a learning algorithm. The accuracy of a learning algorithm is highly dependent on the features that are used to train the system. The best subset contains the least number of dimensions that contributes the most to accuracy; we discard the remaining, unimportant dimensions. This is an important stage of pre-processing. The irrelevant input features also induce greater computational cost and also cause over fitting. Thus it is a very important to extract good features in order to improve the performance of the learning model. There are two general approaches which have been used for feature selection and are known for being pretty good are Wrapper methods such as Forward Selection, filtering methods such as the one we present here and hybrid methods.

### **2.4.1 Wrappers**

A Wrapper selection method searches for the best possible feature subset that is customized to a particular algorithm and a domain. Kohavi and John (1997) describes the Wrapper approach in detail for feature subset selection and also discusses its strengths and weaknesses.

In the feature subset selection problem, out of all the possible set of features, a subset of features is selected. The focus is then set on this subset and the other remaining features are ignored. So, this approach simply involves then determining how well the classification method to be used works with the reduced set of features.



**Figure 2.5: The Wrapper approach to feature subset selection. The induction algorithm is used as a “black box” by the subset selection algorithm (Kohavi and John 1997).**

In the Wrapper approach (John, Kohavi and Pflieger 1994), the feature subset selection is a loop wrapped around the induction algorithm. Figure 2.5 explains the basic idea behind the Wrapper method. A data set is partitioned into internal training and holdout sets, with different sets of features removed from the data. The feature subset with the highest evaluation is selected as the final set on which the induction algorithm is executed. The resulting classifier is then assessed on an independent test set that was not used during the search.

Generally, the aim of supervised learning algorithms is to maximize classification accuracy on a test set. John, Kohavi and Pflieger (1994) have taken on this approach in implementing the feature subset selection. Instead of trying to maximize accuracy, they

could have tried to identify which features were relevant and used only those features during the learning phase. One might think that these two goals were equivalent. Please refer John et al., (1994) where they have shown examples where these two goals differ.

Thus, the Wrapper approach carries out a search for a good subset using the induction algorithm (used as a black box) itself as a part of the evaluation function. The induced classifier's accuracy is then estimated using various accuracy estimation techniques (Kohavi 1995b). Refer John, Kohavi and Pfleger (1994) for search space, operators and search engines used by the Wrapper approach. John, Kohavi and Pfleger (1994) have used two induction algorithms: Decision trees (Section 2.3.3) and Naïve Bayes (Section 2.3.2) and the performance using feature subset selection improved both algorithms. The problems in Wrapper approach include over fitting and the large amount of CPU time is required.

## **2.4.2 Forward Selection**

Forward Selection is kind of a Wrapper model. The word Forward Selection refers to a search that begins at the empty set of features. It is a greedy strategy which starts with no variables and at each step the feature that decreases the error the most are added one by one, until any further addition does not significantly decrease the error. In other words, we start by measuring the Leave-One-Out Cross Validation (LOOCV) error of the one component subsets, so that we can find the best individual feature. Next, Forward Selection finds the best subset consisting of two components from the remaining input attributes. The Forward Selection is algorithm is given in Table 2.5.

For example, suppose we have a set of 10 features and we are interested in the best 5 features that we can get using Forward Selection. The algorithm being used in order to find the accuracy is Naïve Bayes. The process would start by creating a model using naïve Bayes for each feature one by one, the feature which had the best accuracy is chosen as the first best feature. Thus we can see that the model was created 10 times in

this case to get the first best feature. Further, the accuracy of model created using the selected feature and each other feature is calculated one by one. Thus a total of 9 models are created this time in order to get the second best feature. This way the process goes on until you get the desired 5 features.

---

**Table 2.5: Forward Selection Algorithm**

---

1. *Shuffle the data set.*
  2. *Break it into user specified P partitions.*
  3. *For each partition (i = 0, 1, ..., P-1)*
    1. *Create OuterTrainset(i), which is all partitions except i.*
    2. *Create OuterTestset(i), which is the i'th partition.*
    3. *From the OuterTrainset(i), randomly choose 70% of data and form the InnerTrain(i).*
    4. *The remaining 30% of the OuterTrainset(i) forms the InnerTest(i).*
    5. *For j = 0.....m*
      1. *Search for the best feature set with j components, fsij. using leave-one-out on InnerTrain(i) using naïve Bayes to calculate the accuracy.*
      2. *Let InnerTest Scoreij be the accuracy which we get using naïve Bayes.*
    6. *End loop of (j).*
    7. *Select the feature set fsij which has the best inner test score.*
    8. *Let OuterScorei be the accuracy of the selected feature set on OuterTestset(i)*
  4. *End of loop of (i).*
  5. *Return the mean Outer Score.*
- 

It is very clear from the above example, that even though the use of this extensive method in order to chose the best features leads to getting very good feature set. But this process needs a lot of resources and time. Moreover, if the data set is really very large then the use of this method to get large number of features becomes impractical. Assuming the cost of a LOOCV evaluation with  $i$  features is  $Cost(i)$ , then the computational cost of Forward Selection for searching a feature subset of size  $m$  out of  $M$  total input attributes will be

$$M Cost(1) + (M - 1)Cost(2) + \dots + (M - m + 1)Cost(m)$$

We selected a subset of feature using Forward Selection using the algorithm given above in order to compare the results of our methods with Forward Selection results. We used Naïve Bayes algorithm to create the model in order to find the accuracy of the features selected during a particular step in the algorithm.

## **2.5 Related Work**

With the continuous growth in the amount of available data, the process of analyzing it and finding desired information is becoming more and more important. Documents are represented by words which are the features which help in distinguishing a document. Thus feature selection has become the focus of much research in areas of Machine Learning (ML), Static. In this chapter discuss some of the work done in feature selection such as, Wrappers (Kohavi and John 1997), Filter Methods, Weighing Schemes and some other work done in feature selection.

The Wrapper (Kohavi and John 1997) methodology offers a simple and powerful way to address the problem of variable selection. It employs a search through the space of feature subsets using the estimated accuracy from an induction algorithm as the measure of goodness for a particular feature subset. Figure 2.5 explains the basic idea behind the Wrapper method. A data set is partitioned into internal training and holdout sets, with different sets of features removed from the data. The feature subset with the highest evaluation is selected as the final set on which the induction algorithm is executed. The resulting classifier is then assessed on an independent test set that was not used during the search. These methods are known for being good for selecting features but they are very computationally expensive which makes them intractable for large data sets. Wrapper methods such as Sequential Forward Selection (Schlimmer, 1993) or Genetic Search Algorithm (Mitchell, 1997) perform search over the space of all possible subset of features, repeatedly calling the induction algorithm as a subroutine to evaluate various subsets of features.

Filter methods are also one of the known good feature selecting methods. Two of the most well-known Filter methods for feature selection are RELIEF (Kira and Rendell 1992) and FOCUS (Almuallim and Dietterich 1991). In RELIEF, a subset of features is not directly selected, but rather each feature is given a relevance weighting indicating its level of relevance to the class label. These methods are also very computationally expensive similar to the Wrapper method and become intractable for finding large number of features.

Weighing Schemes are also used as the basis for selecting good features. These are a form of Heuristic search. Some of these techniques train a predictor model using the Gradient Descent Algorithm (Nelder and Mead, 1965) in which successive passes through the training instances lead to changes in all the weights. At each iteration, a gradient-based heuristic is used to quickly assess which feature is most likely to improve the existing model, that feature is then added to the model, and the model is incrementally optimized using gradient descent. One such attribute weighing method is the Perception Updating Rule (Minsky and Papert, 1969), which adds and subtracts weights on linear threshold unit in response to errors on training instances. Apart from these methods, the Least-Mean squares algorithm (Windrow and Hoff, 1960) for linear units and Backpropagation (Rumerhart. Hinton, and Williams, 1986) are also used as weighing methods for training set. But there is no explicit way in these methods to remove features.

## **CHAPTER 3**

### **METHODS FOR SELECTING FEATURES**

It is important to focus on the most relevant information when working with an overwhelming amount of data. Features represent the data and they play a very important role in the learning process. They determine the performance of the learning model. With the growing amount of available data, it is very important to focus on methods to find good features which contribute to the classification and identification of the documents. There has been a lot of research in this field. There are some approaches used for feature selection used so far, but most of these algorithms are very computation intensive and time consuming which becomes unrealistic while used with large amount of data.

In this chapter we have proposed some methods for selecting features. In the first section, we have described a method for selecting features randomly in order to show that not all the features are good for learning process. Following the random selection method, we have explained the method of selecting feature using a statistical measure called Chi ( $\chi$ ) Square. In the next section, we have described a weighing method of feature selection using the Term Frequency / Inverse Document Frequency weights. And in the last section we have described a boosting ensemble method for selection of features.

#### **3.1 Selecting Features Randomly**

Randomly selecting features is a method in which we randomly selected features from the set of initially selected features. This method was carried out in order to show what accuracy we get if the features are selected in the most basic, unintelligent way. During the time of initial preparation of data, we parse the data and perform various feature extraction methods in order to get initial set of features. In this process, even after removing stop words, removing all the words with very less frequency and performing stemming on the data. We get around more than 100,000 features. To randomly select

features we just select the desired number of features for each class randomly from the set of features that we got during the initial data preparation.

For example if we need to select 10 features per class for this method, then since we have 21 classes, we randomly select 10 features for each class from the features that we extracted from the data initially as mentioned above in the initial data preparation. Thus we get a total of 210 features. While extracting these 210 features randomly we take care that none of the features are selected twice in order to avoid duplication.

**Table 3.1: Algorithm for Random Feature Selection For Class**

---

<p><b>Random_Feature_Selection_For_Class</b>(<i>Examples,Classes,N,#desired_features</i>)  <i>Examples are the data points or the document, Classes are the categories to which the documents belong or to which they are to be classified. N is the total number of features. The program returns a set of features for each class.</i></p> <ul style="list-style-type: none"> <li>• for i = 1 to #classes <ul style="list-style-type: none"> <li>◦ best_feature_Set[i]=<b>Random_Feature_Selection</b>(i,N,examples, #desired_features)</li> </ul> </li> <li>• end</li> <li>• return best_feature_Set</li> </ul>
--

---

Table 3.1 describes the general algorithm for selecting features. It is common for each method, the only difference being that it gives a calls different functions each time depending on the method which is to be used. It takes in the examples which are in the form (document\_ID, features, Class\_ID), classes, total number of distinct features (found by parsing the documents), number of features desired by the user for each class. For each class, it calls the method for feature selection. In this case the method being called is random\_feature\_selection which is described in Table 3.2. It collects the set of features returned by the method for each class and returns it to the calling function.

**Table 3.2: Algorithm for Selecting Features Randomly**

---

**Random\_Feature\_Selection**(Class\_id,N,Examples,#desired\_Features)

- features\_left = { 1 .. N }
- for i = 1 to #desired\_Features
  - best\_feature = Randomly\_select\_from(features\_left)
  - features\_left = features\_left – best\_feature
  - best\_features\_set[i] = best\_feature
- end
- return best\_features\_set

---

Table 3.2 describes the method for selecting features randomly. This method takes the class\_id currently being processed, the remaining features, examples and the desired number of features for each class. From the set of available features, the method returns a random set of distinct desired features for the class being processed.

### 3.2 Selecting Features Using Chi ( $\chi$ ) Square

This method is an implementation of a feature selection method using Chi ( $\chi$ ) Square statistic. The  $\chi$ -Square statistic measures the lack of independence between a term (t) and class (c). We used the  $\chi$ -Square model proposed by Karl Pearson. The Karl Pearson statistic is based upon a comparison of the observed and expected frequencies which would be encountered in each cell of a two dimensional contingency table, given that it is known that no association exists between the two variables of interest.

Table 2.1 describes the algorithm for feature selection using  $\chi$ -Square method. It is called by the general algorithm for feature selection given in Table 3.5 which is same as the algorithm given in Table 3.1. The only difference between the two is the method called for selecting features which is random\_method in Table 3.1 and  $\chi$ -Square in Table 3.5.

As shown in Table 3.3, first of all we calculate and store the frequency or the occurrence of each term belonging to the class being processed. This means for each term we calculate how many times it occurred in the documents belonging to that particular class.

Further, for each feature we calculate the expected frequency using the observed frequency using the equation described in Table 2.1. Then, we calculate the  $\chi$ -Square using Karl Pearson's formula. After finding all the  $\chi$ -Square values we sort all the values. The terms having the highest  $\chi$ -Square values for a particular class would prove to be the best in classification of the documents of that particular class. Thus, we select the first desired number of features which have the highest  $\chi$ -Square values.

**Table 3.3: Algorithm For Feature Selection Using  $\chi$  - Square**

---

**Chi\_Square\_Feature\_Selection**(current\_class\_id, N, examples, #desired\_features)

- for i = 1 to #features
    - for j = 1 to #examples
      - if jth\_example belongs to class current\_class\_id then
        - Observed\_frequency ( $O_{ij}$ ) += Frequency of feature i in j<sup>th</sup> document
      - end
    - end
    - create\_contingency\_table for ith\_feature
    - calculate Expected\_frequency ( $E_{ij}$ ) as shown in Table 3.3
      - $$\text{chi\_square\_values } (\chi_{ij}^2) = \sum_{i=1}^2 \sum_{j=1}^2 (O_{ij} - E_{ij})^2 / E_{ij}$$
  - end
  - Sort(chi\_square\_values)
  - best\_features = first #desired\_features with highest chi\_square\_value
  - return best\_features
- 

**Table 3.4: Algorithm for Chi Square Feature Selection For Class**

---

**Chi\_Square\_Feature\_Selection\_For\_Class**(Examples, Classes, N, #desired\_features)

*Examples are the data points or the document, Classes are the categories to which the documents belong or to which they are to be classified. N is the total number of features. The program returns a set of features for each class.*

- for i = 1 to #classes
    - best\_feature\_Set[i] = **Chi\_Square\_Feature\_Selection** (i, N, examples, #desired\_features)
  - end
  - return best\_feature\_Set
-

For example, suppose we got the observed frequency of feature\_1 for class\_1 as given in the below given matrices. ~Class 1 denotes all the classes other than class 1, In our case since we have 21 classes thus ~Class 1 denotes classes from Class 2 to Class 21. And ~Feature 1 denotes all the features other than Feature 1.

**Table 3.5 Contingency matrix**

**Observed frequency of feature 1  
(with respect to class 1)**

**Expected frequency of feature 1  
(with respect to class 1)**

	Class1	~ Class 1	Total
Feature 1	10,000	20,000	30,000
~ Feature 1	20,000	50,000	70,000
Total	30,000	70,000	100,000

	Class1	~ Class 1	Total
Feature 1	9000	21,000	30,000
~ Feature 1	21,000	49,000	70,000
Total	30,000	70,000	100,000

Using the observed frequency we calculate the expected frequency. For Feature1 and Class 1 the observed frequency is 19, thus we calculate the expected frequency as  $(30,000/100,000) * 30,000 = 9000$ . Similarly for ~Feature1 and Class 1 expected frequency is  $(70,000/100,000) * 30,000 = 21,000$ , for Feature1 and ~Class 1 expected frequency is  $(30,000/100,000) * 70,000 = 21,000$  and for ~Feature1 and ~Class 1 expected frequency is  $(70,000/100,000) * 70,000 = 49,000$ . Once we have calculated the expected frequency, we calculate the  $\chi$ -Square using the above given formula of Karl Pearson.

$$\chi^2 = \frac{(10,000 - 9000)^2}{9000} + \frac{(20,000 - 21,000)^2}{21,000} + \frac{(20,000 - 21,000)^2}{21,000} + \frac{(50,000 - 49,000)^2}{49000}$$

$$= 111.111111 + 47.6190476 + 47.6190476 + 20.4081633$$

$$= 226.757369$$

### 3.3 Selecting Features Using Information Gain

This method is an implementation of a feature selection method using Information Gain statistic. Information gain is a statistical property that measures how well a given attribute separates the training examples according to their target classification (Mitchell 1997).

Before we study about information gain, it is important to learn about the entropy statistic. Entropy characterizes the (im)purity of an arbitrary collection of examples.

Given a collection  $S$ , containing positive and negative examples of some target concept, the entropy of  $S$  relative to this boolean classification is given by

$$Entropy(S) = \sum_{i=1}^{numclasses} -p_i \log_2 p_i$$

Where  $p_i$  is the proportion of instances in  $S$  that have the  $i^{th}$  class values as output attribute. Entropy, in information theory, is known for specifying the minimum number of bits of information needed to encode the classification of an arbitrary number of  $S$ . The entropy value ranges between 0 and 1.

Now, if we know the measure of impurity in a collection of training examples, we can define a measure of the effectiveness of an attribute in classifying the training data. The measure that we have used here is nothing but the information gain which is the expected reduction in entropy caused by partitioning the examples according to this attribute. More precisely, the information gain,  $InformationGain(S,A)$  of an attribute  $A$  relative to a collection of examples  $S$ , is defined as

$$InformationGain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \left[ \frac{|S_v|}{|S|} Entropy(S_v) \right]$$

Where  $Values(A)$  is the set of all possible values for attribute  $A$  and  $S_v$  is the subset of  $S$  for which attribute  $A$  has value  $v$ . The value  $InformationGain(S,A)$  is therefore the expected reduction in entropy caused by knowing the value of attribute  $A$ .

**Table 3.6: Algorithm For Feature Selection Using Information Gain**

---

**Information\_Gain\_Feature\_Selection**(curr\_class\_id, N, examples, #desired\_features)

- $Entropy(S) = \sum_{i=1}^{numclasses} -p_i \log_2 p_i$
  - for i = 1 to #features
    - $InformationGain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \left[ \frac{|S_v|}{|S|} Entropy(S) \right]$
  - end
  - Sort\_descending(InformationGain\_value)
  - best\_features = first #desired\_features with highest InformationGain\_value
  - return best\_features
- 

**Table 3.7: Algorithm for Information Gain Feature Selection For Class**

---

**Information\_Gain\_Feature\_Selection\_For\_Class**(Examples, Classes, N, #desired\_features)

*Examples are the data points or the document, Classes are the categories to which the documents belong or to which they are to be classified. N is the total number of features. The program returns a set of features for each class.*

- for i = 1 to #classes
    - best\_feature\_Set[i] = **Information\_Gain\_Feature\_Selection**(i, N, examples, #desired\_features)
  - end
  - return best\_feature\_Set
- 

Table 3.6 describes the algorithm for feature selection using Information Gain method. It is called by the general algorithm for feature selection given in Table 3.7 which is same as the algorithm given in Table 3.1. The only difference between the two is the method

called for selecting features which is `random_method` in Table 3.1 and Information Gain in Table 3.7.

**Table 3.8 Training examples for documents with 4 features belonging to 2 categories**

Document	Feature 1	Feature 2	Feature 3	Feature 4	Category
D1	0	1	1	0	Class 2
D2	1	1	1	1	Class 2
D3	0	0	1	0	Class 1
D4	1	1	1	0	Class 1
D5	1	1	0	0	Class 1
D6	1	1	1	1	Class 2
D7	1	1	1	1	Class 1
D8	0	0	0	0	Class 2
D9	1	0	0	0	Class 1
D10	1	0	0	0	Class 1
D11	1	0	1	1	Class 1
D12	1	0	1	1	Class 1
D13	0	0	1	0	Class 1
D14	1	1	1	1	Class 2

For example, refer to the data given below in Table 3.8. The data given consists of 14 documents belonging to 2 categories having 4 features each.

For instance, the information gain due to sorting the original 14 examples by the attribute *Feature 4* may be calculated as:

$$\begin{aligned} \text{Entropy}(S) &= - (9/14) \log_2 (9/14) - (5/14) \log_2 (5/14) \\ &= 0.940 \end{aligned}$$

$$\begin{aligned} \text{Values (Feature 4)} &= 0, 1 \\ S &= [9+, 5-] \\ S_0 &<- [6+, 2-] \\ S_1 &<- [3+, 3-] \end{aligned}$$

$$\begin{aligned} \text{InformationGain}(S, \text{Feature 4}) &= \text{Entropy}(S) - (8/14) \text{Entropy}(S_0) - (6/14) \text{Entropy}(S_1) \\ &= 0.940 - (8/14)0.8411 - (6/14)1.00 \\ &= 0.048 \end{aligned}$$

The higher the information gain, the greater is its capability of separating the training examples according to their target classification. Thus, in the above mentioned way we calculate the information gain for each attribute and select the one which has the highest information gain. For example, if we had to choose 1000 features, then we calculate the information gain of all the possible features and select the first 1000 features having the highest information gain.

### **3.4 Selecting Features using Term Frequency / Inverse Document Frequency**

In this method we selected the terms using the TF / IDF. Inverse Document Frequency is a measure of the general importance of the term. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The term frequency in the given document is simply the number of times a given term appears in that document. This count is usually normalized to prevent a bias towards longer

documents to give a measure of the importance of the term  $t_i$  within the particular document  $d_j$ .

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

where  $n_{ij}$  is the number of occurrences of the considered term in document  $d_j$ , and the denominator is the number of occurrences of all terms in document  $d_j$ .

The inverse document frequency is a measure of the general importance of the term (obtained by dividing the number of all documents by the number of documents containing the term, and then taking the logarithm of that quotient):

$$idf_i = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

with

- $|D|$  : total number of documents in the corpus
- $|\{d_j : t_i \in d_j\}|$  : Number of documents where the term  $t_i$  appears (i.e.,  $n_{i,j} \neq 0$ ).

Then,

$$tfidf_{i,j} = tf_{i,j} \cdot idf_i .$$

A high weight for TF/IDF is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents; the weights hence tend to filter out common terms.

After we process the initial raw data (XML documents) we get a document frequency file which contains the number of times a term occurs in a document. Using this data we calculate the TF/IDF score of each term. Further, we rank all the terms with respect to their TF/IDF score and use the first desired number of features which have the highest rank.

**Table 3.9: Algorithm for Feature Selection using Term Frequency / Inverse Document Frequency**

---

**TF/IDF\_Feature\_Selection**(current\_class\_id, N, examples, #desired\_features)

- for i = 1 to #examples
  - for j = 1 to #features\_in\_ith\_example
    - frequency\_ij = frequency of jth feature in ith example
    - term\_frequency\_ij=frequency\_ij /frequency\_of\_features\_in\_ith\_example
    - Num\_jth\_feature = number of documents where the feature j appears.
    - inverse\_document\_frequency\_ij = log(#examples/Num\_jth\_feature)
    - term\_frequency\_inverse\_document\_frequency\_ij = term\_frequency \* inverse\_document\_frequency
  - end
- end
- Rank all the terms between 0 and 1 by normalizing the document frequency for each term.
- best\_feature\_set = #desired\_features the highest rank.
- return best\_feature\_set

---

The algorithm for the TF/IDF method as explained above is given in Table 3.9. It is called by the general algorithm for feature selection given in Table 3.10 which is same as the algorithm given in Table 3.1. The only difference between the two is the method called for selecting features which is random\_method in Table 3.1 and TF/IDF Frequency in Table 3.10.

**Table 3.10: Algorithm for Feature Selection using TF/IDF For Class**

---

**TF/IDF\_Feature\_Selection\_For\_Class**(Examples,Classes,N,#desired\_features)  
*Examples are the data points or the document, Classes are the categories to which the documents belong or to which they are to be classified. N is the total number of features. The program returns a set of features for each class.*

- for i = 1 to #classes
  - best\_feature\_Set[i]=**TF/IDF\_Feature\_Selection**(i,N,examples,#desired\_features)
- end
- return best\_feature\_Set

---

For example, suppose we have the following data given in table 3.11 where D1, D2 and D3 are documents belonging to Class 1.

**Table 3.11: Data with 3 documents**

Document	Term_id	Frequency
D1	1	3
D1	2	2
D2	1	1
D2	3	2
D3	2	2
D3	3	4

Now, we can calculate the term frequency and the inverse document frequency from the given data using the formulas mentioned above in the steps as shown in Table 3.12

**Table 3.12: Calculation Of TF/IDF Score Using Data In Table 3.12**

Document	Term_id	Frequency	TF-score	IDF-score	TF/IDF
D1	1	3	$3/5=0.6$	$\log(3/2)=0.176$	0.1056
D1	2	2	$2/5=0.4$	$\log(3/2)=0.176$	0.0704
D2	1	1	$1/3=0.33$	$\log(3/2)=0.176$	0.05808
D2	3	2	$2/3=0.66$	$\log(3/2)=0.176$	0.11616
D3	2	3	$3/10=0.3$	$\log(3/2)=0.176$	0.0528
D3	3	7	$7/10=0.7$	$\log(3/2)=0.176$	0.1232

After calculating the TF/IDF score as mentioned above in Table 3.12, we sort all the features as per their TF/IDF score and then we rank each term as given below in Table 3.13.

Now, if the user was interested in 2 features then according to the above rankings the best three features selected for Class 1 (Since all these documents belonged to class 1 as mentioned above) would be 3 and 1 since they have the highest rank. Now we can notice that the best two ranks taken by the term 3, thus we ignore one of them and select the next best feature for the particular class.

**Table 3.13: Sorted TF/IDF Score**

Document	Term_id	Frequency	Tf/Idf	Rank
D3	3	7	0.1232	$(5/5) = 1$
D2	3	2	0.11616	$(4/5) = 0.8$
D1	1	3	0.1056	$(3/5) = 0.6$
D1	2	2	0.0704	$(2/5) = 0.4$
D2	1	1	0.05808	$(1/5) = 0.2$
D3	2	3	0.0528	0

### 3.5 Selecting Features Using A Boosting Approach

Boosting [Freund and Schapire,1995] is one of the most powerful learning methods. It is based on the idea of weak learning. A weak learner is one that has accuracy slightly greater than 50%. It focuses on learning classifiers that are trained taking into account the errors of previous classifiers. We used the idea of weak learning, according to which we weight the data set based on how well we have predicted the data points so far. The data points which are predicted accurately are given low weight and the ones which are mispredicted are given high weight. Thus, we give importance to the data points which are constantly being mispredicted and in this process find features that would prove to be useful in predicting such data points. Now if there is a data point which has been misclassified a lot number of times then the weight of that data point would increase considerably. Thus, we penalized the data set which has been misclassified more than 15 times as the weights for these data points would increase to a huge amount.

The algorithm in Table 3.14 describes the process of the algorithm. It is called by the general algorithm for feature selection given in Table 3.15. The boosting value of a feature depends on whether that feature is useful in predicting or mispredicting the

examples or not. As shown here are two functions in Table 3.14, one for the main working of the method and the next one (calc\_boost\_value\_of\_feature) is for calculating the boost value of the feature being processed.

**Table 3.14: Algorithm For Feature Selection Using Ada-Boosting**

---

**Ada\_Boosting\_Feature\_Selection**(class\_id,N,examples,#desired\_features)

- Set weight of examples to  $1/\#examples$
- features\_left = { 1 .. N }
- for i = 1 to #desired\_features
  - best\_feature = ith\_item(1,features\_left)
  - best\_value = calc\_boost\_value\_of\_feature(best\_feature)
  - for j = 2 to | features\_left |
    - new\_value =  
 calc\_boost\_value\_of\_feature(ith\_item(j,features\_left))
    - if new\_value > best\_value then
      - best\_feature = ith\_item(j,features\_left)
      - best\_value = new\_value
    - end
  - end
  - best\_features\_set[i] = best\_feature
  - $\epsilon = \sum \text{weight\_of\_mispredicted\_points}$
  - $\beta = (1 - \epsilon) / \epsilon$
  - multiply the weights of all the misclassified points to  $\beta$
  - normalize weights to sum to 1
- end
- return set of best\_features\_set

**double calc\_boost\_value\_of\_feature**(feature\_num)

- $\text{weight}_{\text{correctly\_predicted}} = \sum \text{weight\_of\_correctly\_predicted\_points by feature\_num}^{\text{th}} \text{ feature}$
  - $\text{weight}_{\text{minpredicted}} = \sum \text{weight\_of\_mispredicted\_points by feature\_num}^{\text{th}} \text{ feature}$
  - $\text{boost\_value} = | \text{weight}_{\text{correctly\_predicted}} - \text{weight}_{\text{minpredicted}} |$
  - return boost\_value
- 

As given in the algorithm in Table 3.14, the Ada boosting method initially sets the weights of all the datapoints to a negligible number. For each class being processed, it returns the best set of features having the highest boost values. The process determines the boost values of each feature depending on how useful it was in predicting or

mispredicting the examples. After finding each feature, weight of all the examples are recalculated as shown in the algorithm. This is done to give more importance to the examples which have been mispredicted so far. Thus, in this way we get a set of features which are useful in predicting the examples.

**Table 3.15: Algorithm for Feature Selection Using Ada Boosting For A Class**

---

**Ada\_Boosting\_Feature\_Selection\_For\_Class**(*Examples, Classes, N, #desired\_features*)  
*Examples are the data points or the document, Classes are the categories to which the documents belong or to which they are to be classified. N is the total number of features. The program returns a set of features for each class.*

- for  $i = 1$  to #classes
  - $\text{best\_feature\_Set}[i] = \text{Ada\_Boosting\_Feature\_Selection}(i, N, \text{examples}, \#desired\_features)$
- end
- return best\_feature\_Set

---

**Table 3.16: Data For Ada Boosting Algorithm**

Document	Feature 1	Feature 2	Feature 3	Class 1	Weight
D1	1	0	1	1	0.14
D2	0	1	0	0	0.14
D3	1	0	1	1	0.14
D4	1	0	1	0	0.14
D5	1	0	1	1	0.14
D6	0	1	0	0	0.14
D7	0	1	1	1	0.14

For example, suppose we have the following data given in table 3.16. The data given has 7 documents having 3 features each and is being evaluated for one class which in this case is class-1.

In the above example, the initial weight is set by the formula  $1/\#examples$ , which in the above case is  $1/7 = 0.14$ . Starting with Feature 1, we can see that it is predicting the class 5 times and mispredicting it the rest 2 times. Thus the boost value for it is calculated as

$(5 - 2) * 0.14 = 0.42$ . Similarly, for feature 2 we can see that it mispredicts the class 5 times and predicts it correctly 2 times. Thus, its value is  $(2 - 5) * 0.14 = 0.42$ . Finally for Feature 3, we can see that it correctly predicts the class 6 times and mispredicts it 1 time. Thus, its boost value is calculated as  $(6 - 1) * 0.14 = 0.70$ . Thus, we choose Feature 3 because it has the highest boost value. Further, to recalculate the weight. We multiply each datapoint (which has been mispredicted by Feature 3) by the total weight of the mispredicted points. Calculating beta as shown above in the algorithm and then multiplying weight of each mispredicted point by beta. Finally we normalize each weight to sum to 1 before we go ahead for finding the next best feature.

## CHAPTER 4

### EXPERIMENTS

This Chapter presents the results of our methods for feature selection. We first describe the data set that we have used to perform our experiments along with the process by which we have extracted the initial set of features. In the next section we discuss experiments which were performed on features selected from our method. Later on we discuss the experiments performed on the features selected using Forward Selection method which was used as a baseline method for comparing our results. For each experiment, we present the methodology, the results obtained and the accuracy achieved.

#### 4.1 Our Data Set

For our experiments we have used the INEX 2007 (document mining track) document collection. The documents are extracted from the Wikipedia XML Corpus (Denoyer, Gallinari, 2007) and split into training and testing parts. The document collection can be downloaded from the XML Mining Website (<http://xmlmining.lip6.fr>). The corpus contains a total of 96,611 documents, out of which 48,306 documents form the training set and the remaining 48,305 documents form the testing set. The documents are organized in 21 categories that correspond to different Wikipedia Portals and basically thematic categories. Table 4.1 displays statistic concerning the categories of the complete corpus.

The category statistic in Table 4.1 shows that the categories are not well balanced, some of them are really very large as compared to others, for example Portal:Law is composed of 25% of the documents and Portal:Music is composed of about 0.4% of the documents. Moreover, the corpus has been built in order to propose some ambiguous categories such

as, for example, Portal:Pornography and Portal:Sexuality or Portal:Chistianity and Portal:Spirituality.

**Table 4.1: Description of Different Categories.**

<b>ID</b>	<b>Category</b>	<b>Number of Documents</b>	<b>Percentage (%) of Overall Data</b>
2112299	Portal:Law	24213	25.4%
1597184	Portal:Literature	16929	17.7%
1484914	Portal:Sports and games	14595	15.3%
1480358	Portal:Art	7624	8.0%
1886386	Portal:Physics	5149	5.4%
3091788	Portal:Christianity	4671	4.9%
2773006	Portal:Chemistry	4567	4.8%
1685758	Portal:History	3246	3.4%
3091127	Portal:Spirituality	2704	2.8%
2914908	Portal:Sexuality	2402	2.5%
2879927	Portal:War	2217	2.3%
1507239	Portal:Aviation	1217	1.2%
2328885	Portal:Formula One	1188	1.2%
1486363	Portal:Astronomy	1105	1.1%
1895383	Portal:Trains	953	1.0%
2314377	Portal:University	605	0.6%
2635947	Portal:Comics	600	0.6%
2257163	Portal:Pornography	458	0.4%
2263642	Portal:Writing	412	0.4%
474166	Portal:Music	401	0.4%

We parse the document collection to extract an initial set of distinct features. First, we remove stop words such as *a*, *and*, *the* and *about*, using a standard stop word list. We also eliminate words whose length is less than three. We then remove all the words whose frequency in the entire collection is not more than one as they would not prove to be of very good help in classifying the document in which they occur. Later on, we strip

off all the punctuation marks from the words and perform stemming on the terms using Porter's algorithm. We also keep a track of all the two word and three word phrases that occur in the documents while selecting the features and finally we keep only the phrases that occur more than 5 times in the whole document set. Using all the features that we have extracted, we create a dictionary file containing all the unique terms with a unique ID for each term. Once our dictionary is formed consisting of all the distinct features that we extracted, we create a term frequency file which contains for each document the term that occurs in it and the frequency of the term occurring in that document.

We then use the different algorithms we describe in Chapter 3 to filter out the best features which would be most helpful in classifying the documents. For doing so we select a subset of 10, 25, 50 and 75 features per class from the original set of features using our feature selection methods such as , Random Feature Selection, Chi ( $\chi$ )-square Feature Selection, Information Gain Feature Selection, feature Selection using Term Frequency / Inverse Document Frequency (TF/IDF) statistic and method using the Boosting-based approach. Each set of selected features are tested for there accuracy using various ML algorithms such as Simple Count, Naïve Bayes, J48 Decision Trees (Witten and Frank, 2005) and Random Forests. We performed baseline experiments with features selected using Forward Selection method in order to compare our results. We have used Naïve Bayes to check the accuracy of the Forward Selection method. The next section describes the results of our experiments.

## **4.2 Results For The Whole Data Set Using Naive Bayes**

We extracted 185,326 features from the initial set of 48,306 training documents. From all the experiments that we performed on the data using various feature selection methods, we noticed that Naïve Bayes, consistently performed well. To establish baseline accuracy we started by performing Naive Bayes testing using the training set and ALL of the features. Because Naive Bayes does not consider any combinations of features but simply computes the contribution of each feature independently it is possible (though

very slow) to run this algorithm on all of the features. For all of the examples and all of the features the accuracy of Naive Bayes is 64.1%. But as can be noted below, similar or better accuracies can be achieved with a much smaller number of features.

Note that using a reduced number of features not only can increase accuracy, but is often useful to be able to show the user the set of features that resulted in better performance. In addition, the Naive Bayes method is one of the few methods that can be tractably used on such a large data set.

### **4.3 Feature Selection Results**

This section presents the results for our methods which are mentioned in Chapter 3. For each feature selection method, we performed five random two -fold validation tests as suggested by Dietterich (1998). We selected a set of 10, 25, 50 and 75 features per class. Which means, 10 features is actually a total of  $10 \times 21 = 210$  features, because our data set has 21 classes. Each set of selected features were tested for accuracy using the following machine learning algorithms: such as Naïve Bayes (Good, 1965; Langley et al., 1992), J48 Decision Trees (Witten and Frank, 2005), Random Forests (Breiman 2001) and Simple Count. The Simple Count method is a simple form of Naïve Bayes in which the prior probabilities of Naïve Bayes are not considered. The frequency of a feature in each class is considered as a measure of importance of that feature. We used WEKA for the J48 Decision Trees (Witten and Frank, 2005) method and Leo Breiman's software for Random Forests and SVM<sup>light</sup> for Support Vector Machines. The results presented in the tables below are average of the 5 random 2 cross-fold results.

The section starts with the results of Random Feature selection method, followed by the  $\chi$ -Square Result and Information Gain statistic. Later on we present results for Term Frequency / Inverse Document Frequency method and Ada Boosting method.

### 4.3.1 Random Feature Selection Results

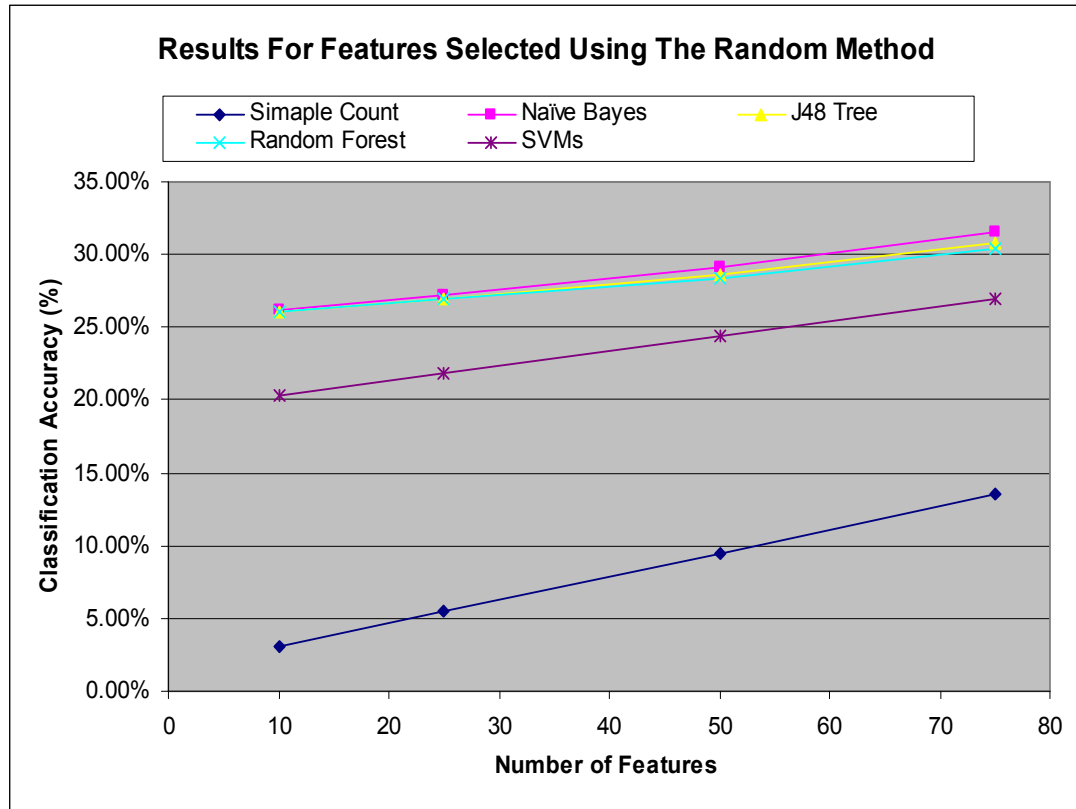
We performed the Random Feature Selection method in order to show how randomly selected features contribute to the learning process. Table 4.2 presents the results for the Random Feature Selection. We performed five random two-fold cross-validation experiments for the data. Table 4.2 presents the averaged results of these experiments and Figure 4.1 graphs the results. We selected a set of 10, 25, 50 and 75 features in order to observe the effect of increasing number of features.

**Table 4.2: Results for Random Feature Selection**

<b>Evaluation Method</b>	<b>10 Features</b>	<b>25 Features</b>	<b>50 Features</b>	<b>75 Features</b>
Simple Count	3.11%	5.49%	9.45%	13.54%
Naïve Bayes	26.14%	27.20%	29.15%	31.53%
J48 Decision Trees	26.07%	26.99%	28.65%	30.78%
Random Forests	26.08%	26.92%	28.35%	30.38%
SVMs	20.26%	21.83%	24.34%	26.96%

Table 4.2 shows that with the increase in the number of features, the accuracy of each method increases gradually. Table 4.2 shows that Random Selection of features is not a very good approach for feature selection and it can be noted that the observed accuracy is very less.

Based on the plot of the results shown in Figure 4.1, Note that Naïve Bayes, J48 Decision Trees and Random Forests perform a lot better as compared to the Simple Count results (though this is not surprising). With the increase in the number of features there is a small gradual increase in accuracy for Naïve Bayes, J48 Decision Trees and Random Forests. In the case of Simple count, the increase in accuracy with the increase in number of features is more prominent, even though the method itself does not perform that well itself as compared to the other methods. Overall the results are significantly worse than the methods that attempt to filter for useful features as shown below.



**Figure 4.1: Random Feature Selection Graph**

### 4.3.2 Chi ( $\chi$ ) Square Results

This section presents the result obtained from features selected using the  $\chi$ -Square statistic. Table 4.3 presents the accuracy of Simple Count, Naïve Bayes, J48 Decision Trees and Random Forests. Note that, for all of the examples and all of the features the accuracy of Naïve Bayes is 64.1%. But as can be noted below in Table 4.3, similar or better accuracies can be achieved with a much smaller number of features. For example, the Subset of 50 and 75 features selected using the  $\chi$ -Square method, when evaluated using Naïve Bayes and Random Forests Method have accuracy more than or equal to 64.1%. Also it can be noted that the J48 Decision Trees also performs comparably with the accuracy achieved for all the examples and all the features. Thus, it can be noted from the results shown in Tale 4.3, that the selection of a subset of features leads to better accuracy.

**Table 4.3: Results for  $\chi$ -Square Feature Selection**

<b>Evaluation Method</b>	<b>10 Features</b>	<b>25 Features</b>	<b>50 Features</b>	<b>75 Features</b>
Simple Count	42.15%	48.38%	51.04%	51.98%
Naïve Bayes	56.64%	61.46%	64.00%	65.28%
J48 Decision Trees	55.93%	60.44%	62.37%	63.26%
Random Forests	57.17%	62.23%	64.24%	64.86%
SVMs	50.09%	53.83%	56.79%	57.40%

Table 4.3 shows the change in accuracy for all the methods are a bit different to the results for features selected using Random Selection method. In the case of Random Features there was around 2% increase in accuracy between the results for 50 and 75 features. But in the case of  $\chi$ -Square it is 1% or less. Figure 4.2 shows that there is a gradual increase in accuracy with the increase in number of features till we reach 50 features. It can be observed that there is not much variation between the accuracy found with 50 features and 75 features but the method itself has an improvement of twice the accuracy of features selected using Random Selection method.

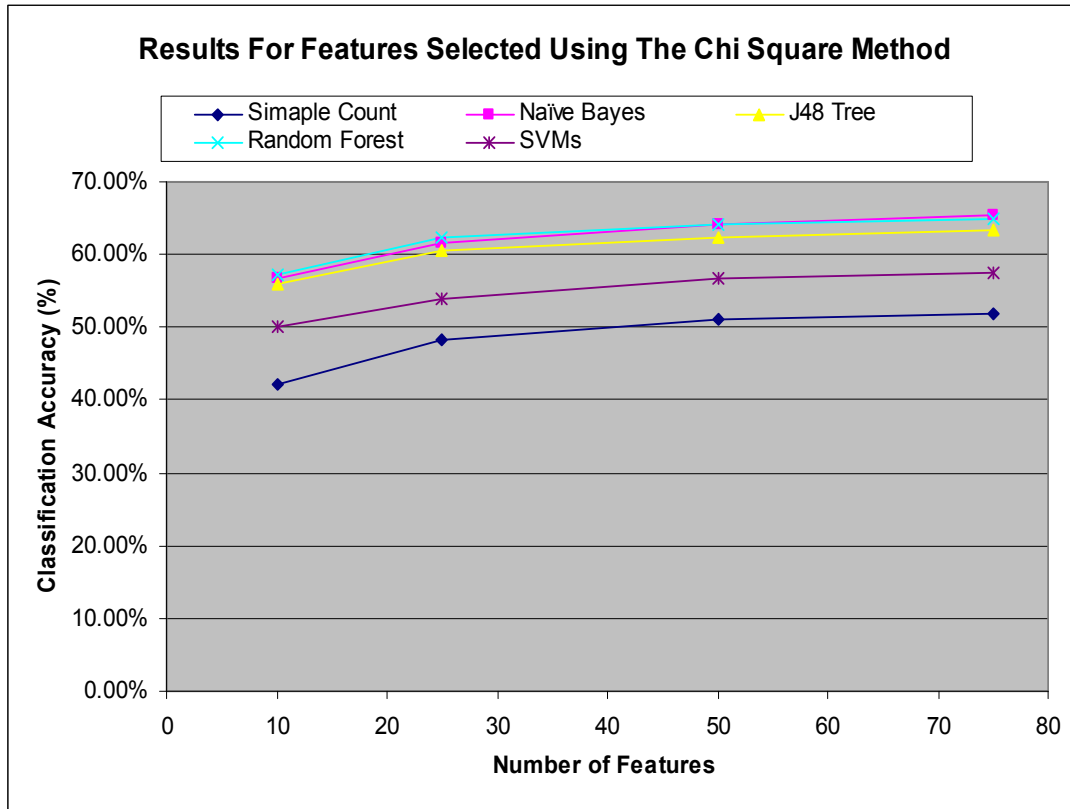


Figure 4.2: Graph for  $\chi$ -Square result

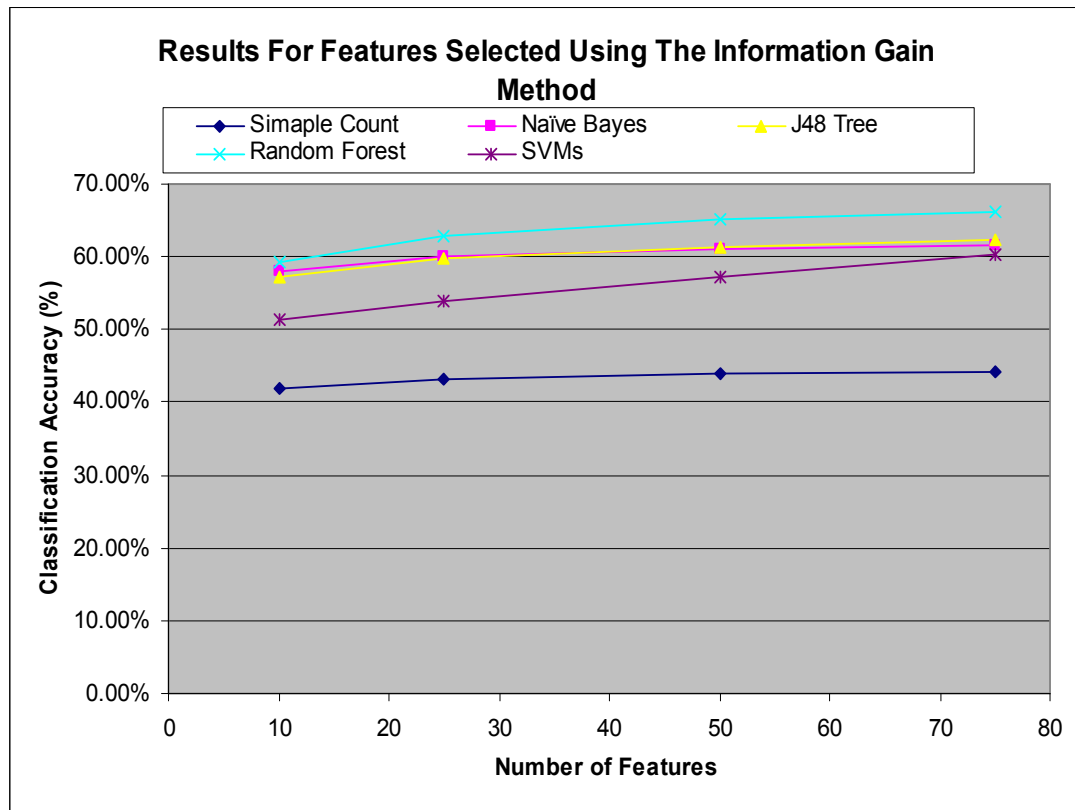
### 4.3.3 Information Gain Results

This section presents the results for features selected using Information Gain method. Table 4.4 presents the results, which are similar to the  $\chi$ -Square results, TF/IDF results and the Ada boosting results. Note that, similar to the results achieved for a subset of features selected using  $\chi$ -Square as mentioned in the Table 4.3, the features selected using Information Gain also perform considerably well as compared to the accuracy (64.1%) that we achieved for all the examples and all the features using Naïve Bayes. It can be noted in Table 4.4 that the accuracy 65.0509% and 66.1622% as obtained for the subset 50 and 75 selected features using the Random Forests method performs better than the accuracy of all the features we obtained using all the features. Also, the methods such as Naïve Bayes and J48 Decision Trees have an accuracy of around 61% to 62% for the subset of 50 and 75 features which is very close to 64.1% that we achieved for all of the examples and all of the features as mentioned in the previous section.

**Table 4.4: Results for Information Gain Feature Selection**

<b>Evaluation Method</b>	<b>10 Features</b>	<b>25 Features</b>	<b>50 Features</b>	<b>75 Features</b>
Simple Count	41.85%	43.28%	43.88%	44.19%
Naïve Bayes	57.95%	59.98%	61.00%	61.52%
J48 Decision Trees	57.22 %	59.83 %	61.28%	62.21 %
Random Forests	59.17%	62.92%	65.05%	66.16%
SVMs	51.42%	53.90%	57.34%	60.29%

Figure 4.3 shows the graph of the results for features selected using Information method. It can be observed that the Naïve Bayes method, J48 Decision Trees method and the Random Forests method perform better than the Simple Count similar to the results for all the other feature selection methods. Similar to the results above, it can be noticed that increasing the number of features result in a small increase in the accuracy. But the results we get with 50 features and 75 features have negligible difference between them.



**Figure 4.3: Information Gain Feature Selection Graph**

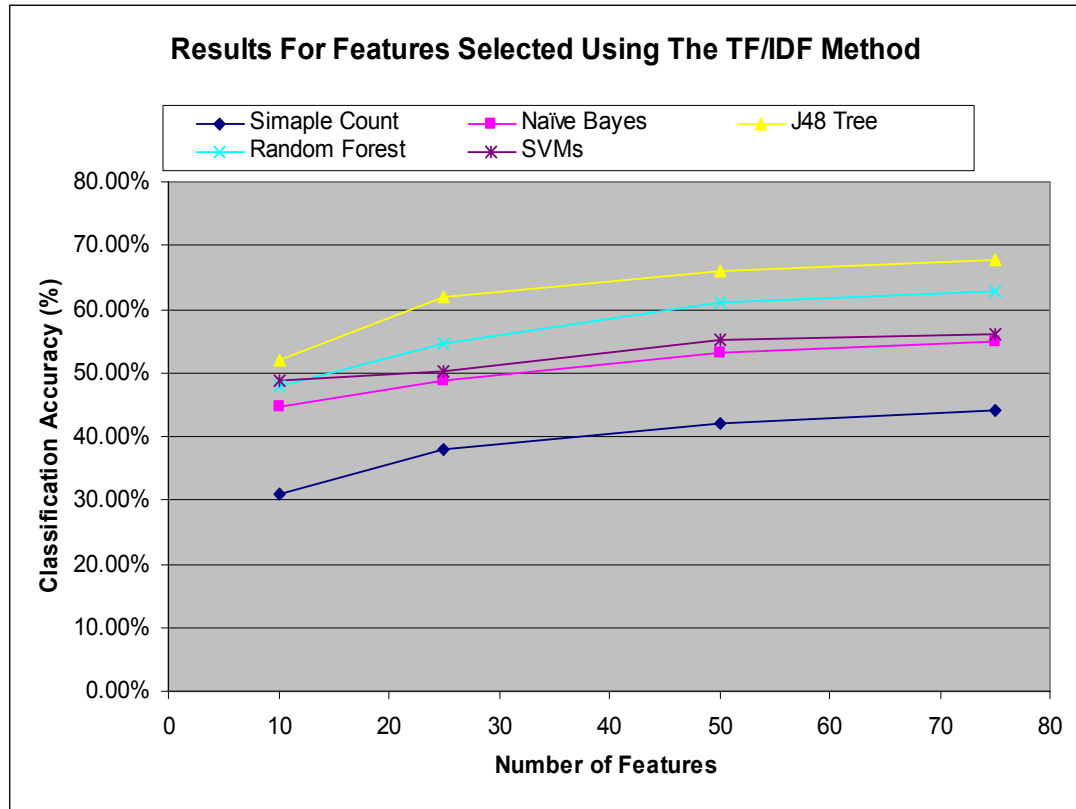
### 4.3.4 Term Frequency / Inverse Document Frequency Results

This section presents the result of features selected using Term Frequency / Inverse Document Frequency (TF / IDF) method. Table 4.5 shows that the accuracy of the features selected using the TF/IDF method is better than the accuracy of Random Feature Selection as also noticed in the case of performance of features selected using the  $\chi$ -Square method and the Information Gain method. Also, it can be noted that the accuracy achieved using a subset of 50 and 75 features when evaluated using the J48 Decision Tree method and the Random Forests method perform better than the accuracy we obtained using all the examples and all the features. It can also be noted in Table 4.5 that the accuracy obtained for 75 features when evaluated with J48 Decision Trees is the highest accuracy we got for all our experiments. The curves which pass through the five highest accuracy points can be noted in Figure 4.10 and the accuracies are listed in Table~ 4.11.

**Table 4.5: Results for Frequency / Inverse Document Frequency Feature Selection**

<b>Evaluation Method</b>	<b>10 Features</b>	<b>25 Features</b>	<b>50 Features</b>	<b>75 Features</b>
Simple Count	31.05%	37.87%	42.14%	44.11%
Naïve Bayes	44.54%	48.74%	53.01%	54.80%
J48 Decision Trees	52.02%	61.85%	65.93%	67.63%
Random Forests	47.79%	54.71%	60.88%	62.78%
SVMs	48.87%	50.36%	55.32%	55.98%

Figure 4.4 shows the change in accuracy with the increase in number of features for each method. We can notice that J48 Decision Trees method performs better than all the other methods for this feature set, followed by Random Forests and Naïve Bayes. The nature of the accuracy of Simple Count is not good as noticed in the results above.



**Figure 4.4: TF/IDF Feature Selection Graph**

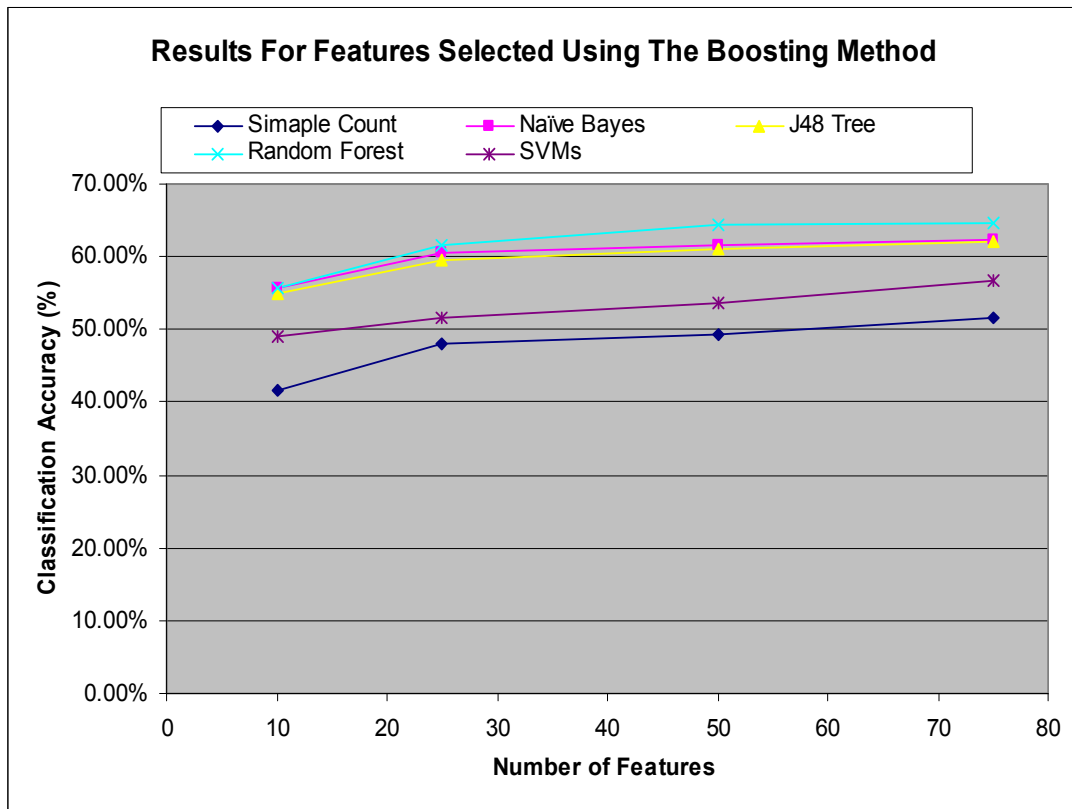
### 4.3.5 Ada Boosting Results

This section presents the results for Features selected using the Ada Boosting method. Table 4.6 presents the result, which are similar to the features selected using the  $\chi$ -Square method, Information Gain method and TF/IDF method as can be noted in the previous Subsections. Also, the accuracies achieved for a subset of 50 and 75 features when evaluated using the Naïve Bayes method, J48 Decision Trees and the Random Forests method perform similar to or better than the accuracy we achieved using all the examples and all the features as mentioned in the previous Section. It can be noted, that the accuracy obtained for 75 features using the Random Forests method is amongst the five highest accuracy points obtained for all our experiments listed in Table 4.11.

**Table 4.6: Results for Ada Boosting Feature Selection**

<b>Evaluation Method</b>	<b>10 Features</b>	<b>25 Features</b>	<b>50 Features</b>	<b>75 Features</b>
Simple Count	41.60%	47.95%	49.34%	51.73%
Naïve Bayes	55.65%	60.51%	61.59%	62.22%
J48 Decision Trees	54.97%	59.61%	61.01%	62.03%
Random Forests	55.74%	61.68%	64.26%	64.70%
SVMs	49.08%	51.57%	53.77%	56.65%

Figure 4.5 shows the graph of the results for features selected using the Ada Boosting method. It can be observed that the methods Naïve Bayes, J48 Decision Trees and Random Forests perform better than the Simple Count method. Similar to the results above, it can be noticed that increasing the number of features result in a small increase in the accuracy. But the results we get with 50 features and 75 features have negligible difference between them.



**Figure 4.5: Ada Boosting Feature Selection Graph**

## 4.4 Baseline Results Using Forward Selection

We know that the Forward Selection algorithm starts with a null feature set and, for each step, the best feature that gets the highest accuracy is included with the current feature set. This leads to a consumption of a large amount of resources and time. Assuming the cost of a Leave-One-Out Cross Validation (LOOCV) evaluation with  $i$  features is  $Cost(i)$ , the computational cost of Forward Selection for searching a feature subset of size  $m$  out of  $M$  total input attributes will be

$$M Cost(1) + (M - 1)Cost(2) + \dots + (M - m + 1)Cost(m)$$

The cost of one prediction with Naïve Bayes as the function approximator is  $O(n^2)$  where  $n$  is the number of datapoints. The cost of computing the mean leave-one-out error, which involves  $N$  predictions, is  $O(N * n^2) \approx O(n^3)$  where  $N$  is the number of features. And so the full cost of feature selection using the above formula is  $O(m * N * n^2) \approx O(n^4)$ , where  $m$  is the desired number of features. Thus we can see that with the increase in number of features, the time complexity of Forward Selection method increases exponentially.

For example, suppose we have to find 2 best features using the Forward Selection method with Naïve Bayes as the function approximator and if we have 180,000 features to start with, given that Naïve Bayes takes around 14 seconds to give the results for 48000 documents, the amount of time to get 2 best features using Forward Selection would be:

$$(180,000 * 14) / (3600) + (179,999 * 14) / 3600 = 1399.99 \text{ hours which is around 58 days.}$$

Thus we can see that the program would take a month to get each good. This makes the Forward Selection Method intractable for being used for finding features for large data sets.

**Table 4.7: Average accuracy of two-fold test performed for each best feature set that we get using Forward Selection and all the other feature selection methods.**

Features	Random Method	$\chi$ -Square	Information Gain	TF/IDF Method	Ada Boosting	Forward Selection
1	25.1343%	28.09%	28.1006%	27.49%	27.029%	28.12385%
2	25.3968%	28.1366%	28.1006%	27.5858%	27.1004%	29.0838%
3	25.4193%	28.1759%	28.3078%	27.69135%	27.5309%	30.0965%
4	25.4193%	28.1711%	28.5347%	27.69135%	27.9923%	30.74%
5	25.4193%	28.2252%	28.51%	27.69135%	28.0192%	31.33075%
6	26.5377%	28.8575%	28.8015%	27.69135%	28.1034%	31.7527%
7	26.5377%	28.9917%	29.6259%	27.7012%	28.3602%	32.14300%
8	26.618%	29.0317%	29.6259%	28.01735%	28.5109%	32.51645%
9	26.6953%	29.5354%	29.778%	28.7279%	28.612%	32.9398%
10	26.7009%	30.0804%	30.187%	28.9335%	29.0356%	33.3667%

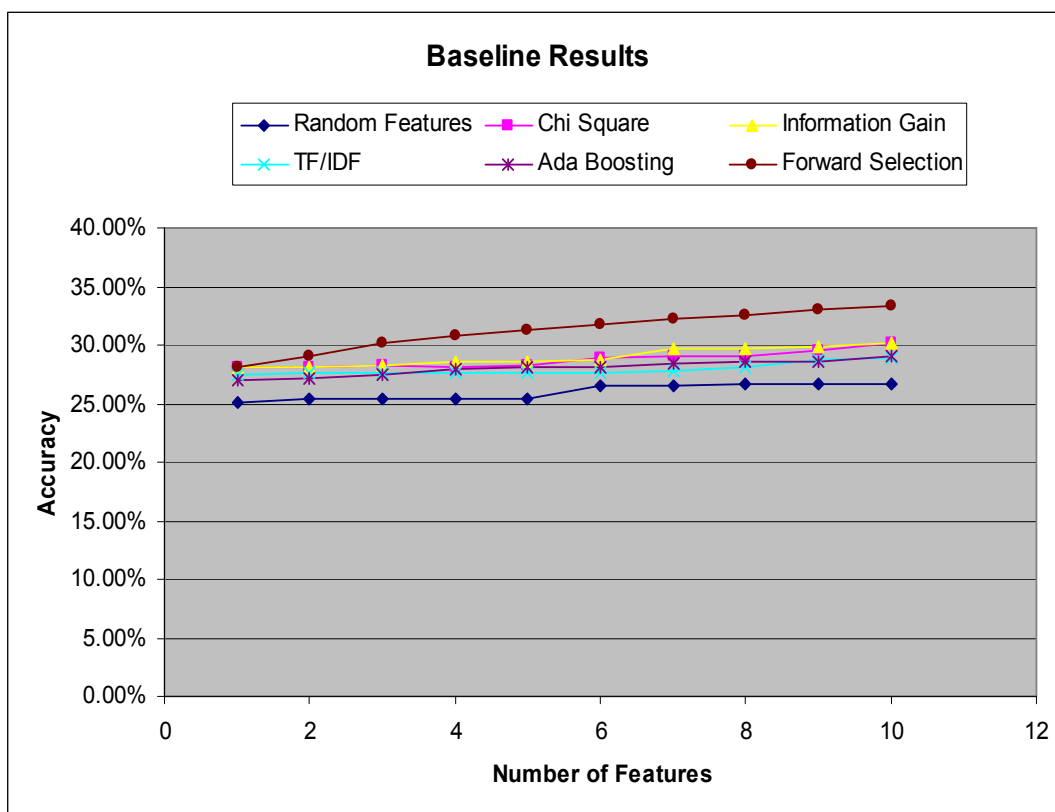
Since we have around 185326 initial features to start with, Forward Selection would take almost a month for each feature feature. Thus we performed our baseline experiments using the Forward Selection Algorithm on a smaller data set consisting of around 16000 documents. This size data set could be run in a few days. We performed a one two-fold test showing the accuracy for each good feature that we get. Thus, the results shows the average of two-fold test for each best feature set that we get (1 feature set, 2 feature set, and so on till 10 feature set). We performed similar two-fold test for all the other methods which include, Random Feature Selection, Chi ( $\chi$ )-square Feature Selection, Information Gain Feature Selection, feature Selection using Term Frequency / Inverse Document Frequency (TF/IDF) statistic and method using the Boosting-based approach.

We calculated the accuracy of each of these feature sets using the Naïve Bayes method. Table 4.7 shows the result of each feature set that we got using the feature selection methods.

Table 4.7 shows that Forward Selection performs better than the other methods, but the difference is only around 3% to 4% when compared with all the other methods other than

the Random feature Selection method. Random Feature Selection selects the features randomly, thus the accuracy varies depending on the goodness of the features that are selected. 3% to 4% accuracy difference is very less as compared to the amount of time Forward Selection takes for each feature. As mentioned in the example given above, for the whole data set, for getting each feature, it would take almost a month which makes Forward Selection intractable for large data sets.

Figure 4.6 shows the graph plot of the baseline results showed in Table 4.7. It plots the performance of each feature selection method we proposed along with the Forward Selection result. It shows that the performance of Forward Selection is the best and the accuracy increases with the increase in the number of features. All the other methods such as, Chi ( $\chi$ )-Square Feature Selection, Information Gain Feature Selection, feature Selection using Term Frequency / Inverse Document Frequency (TF/IDF) statistic and method using the Boosting-based approach also perform comparably good and there accuracy also increases gradually with the increase in the number of features. Random Feature selection method does not perform as good as the other methods because the goodness of the features selected by it, solely depends on random selection which might vary depending on the random number selected.

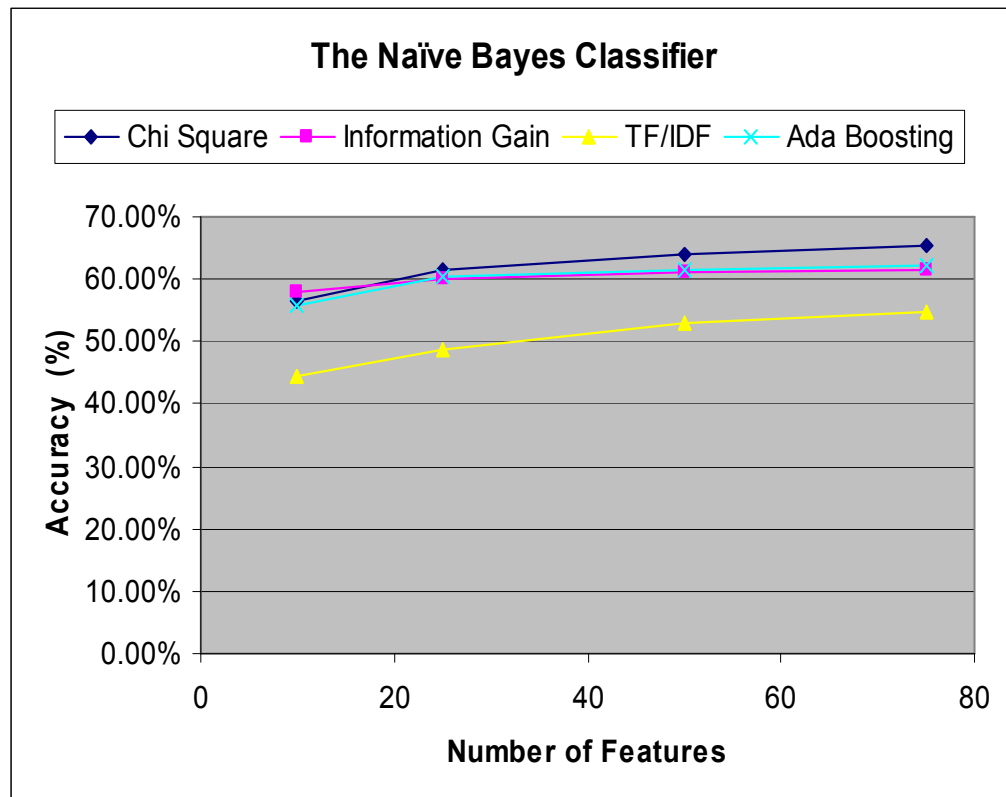


**Figure 4.6: Results of Naïve Bayes on features selected using Forward Selection and our feature selection methods.**

Below given Tables and Figures show the performance of each feature selection method for all the classifiers that we used. Table 4.8 shows the performance of Naïve Bayes method for the proposed feature selection methods and Figure 4.7 shows the graph for the results in Table 4.8. Similarly, Table 4.9 shows the accuracy of the feature selection methods evaluated using J48 Decision Tree Algorithm and Figure 4.8 shows the graph for the results in Table 4.9. Table 4.10 shows the accuracy of the feature selection methods evaluated using Random Forests and Figure 4.9 shows the graph for the results in Table 4.10. Table 4.11 shows the accuracy of the feature selection methods evaluated using SVMs and Figure 4.10 shows the graph for the results in Table 4.11. Table 4.12 and Figure 4.11 shows the accuracy and the graph for the five highest accuracy points achieved for all the experiments performed

**Table 4.8: Accuracy of the Feature Selection Methods evaluated using Naïve Bayes Algorithm.**

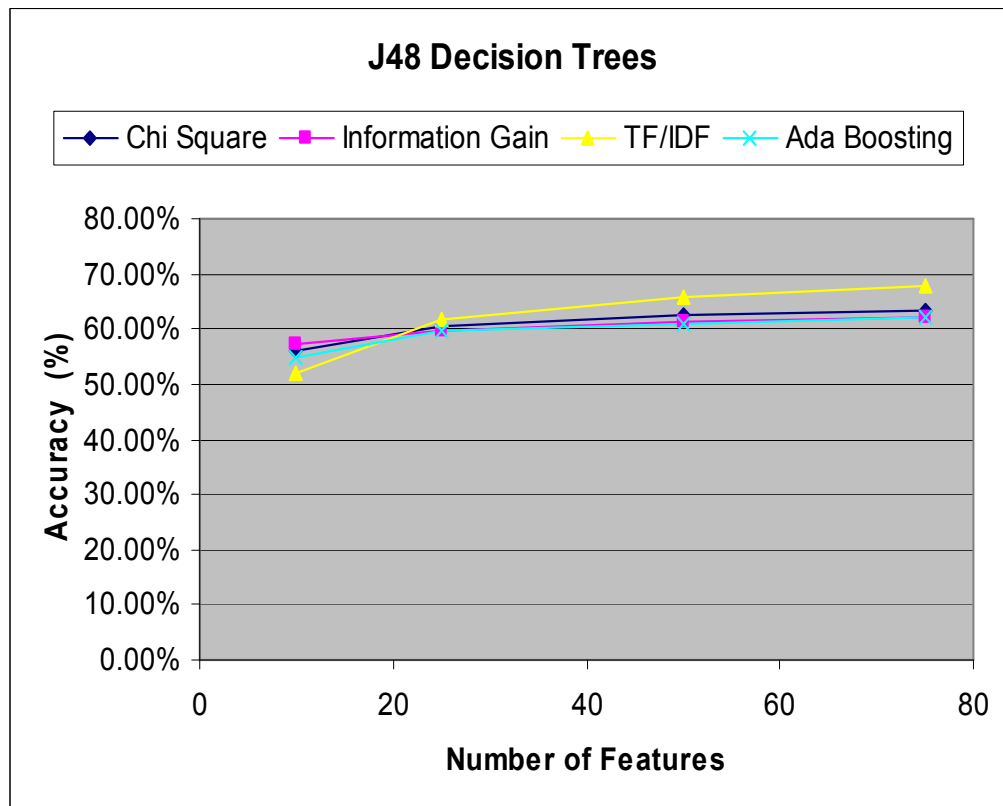
Methods	10 Features	25 Features	50 Features	75 Features
$\chi$ - Square	56.64%	61.46%	64.00%	65.28%
Information Gain	57.96%	59.99%	61.01%	61.53%
TF/IDF	44.54%	48.74%	53.01%	54.80%
Ada Boosting	55.65%	60.51%	61.59%	62.22%



**Figure 4.7: Graph For Accuracy of Naïve Bayes Algorithm for Features Selected Using Proposed Feature Selection Methods.**

**Table 4.9: Accuracy of the Feature Selection Methods evaluated using J48 Decision Tree Algorithm.**

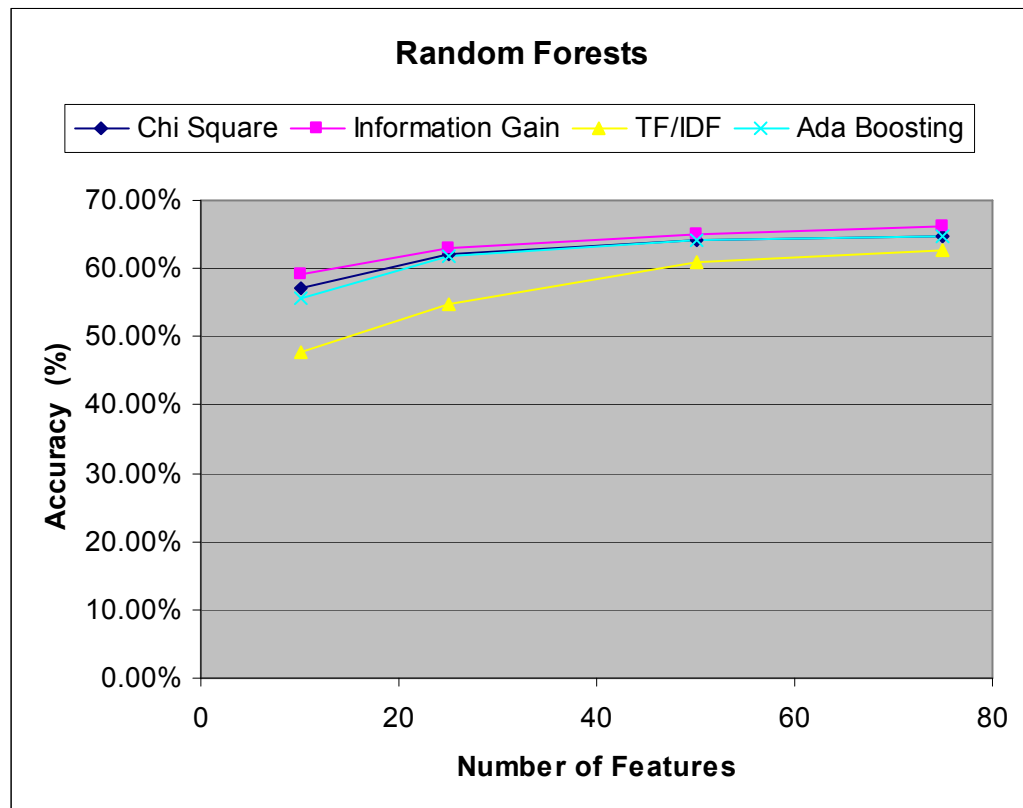
Methods	10 Features	25 Features	50 Features	75 Features
$\chi$ - Square	55.93%	60.44%	62.37%	63.26%
Information Gain	57.2294 %	59.8336 %	61.2841 %	62.2163 %
TF/IDF	52.02%	61.85%	65.93%	67.63%
Ada Boosting	54.97%	59.61%	61.01%	62.03%



**Figure 4.8: Graph For Accuracy of J48 Decision Tree Algorithm for Features Selected Using Proposed Feature Selection Methods.**

**Table 4.10: Accuracy of the Feature Selection Methods evaluated using Random Forests**

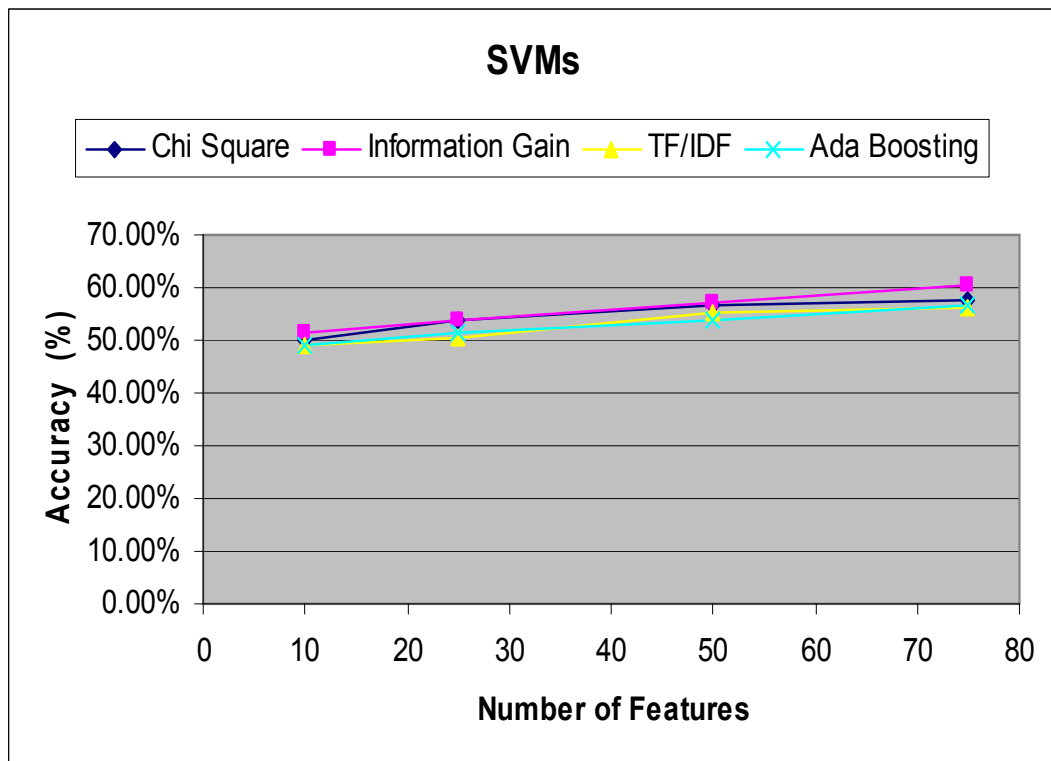
Methods	10 Features	25 Features	50 Features	75 Features
$\chi$ - Square	57.17%	62.23%	64.24%	64.86%
Information Gain	59.1798%	62.9281%	65.0509%	66.1622%
TF/IDF	47.79%	54.71%	60.88%	62.78%
Ada Boosting	55.74%	61.68%	64.26%	64.70%



**Figure 4.9: Graph For Accuracy of Random Forests Algorithm for Features Selected Using Proposed Feature Selection Methods.**

**Table 4.11: Accuracy of the Feature Selection Methods evaluated using Support Vector Machines**

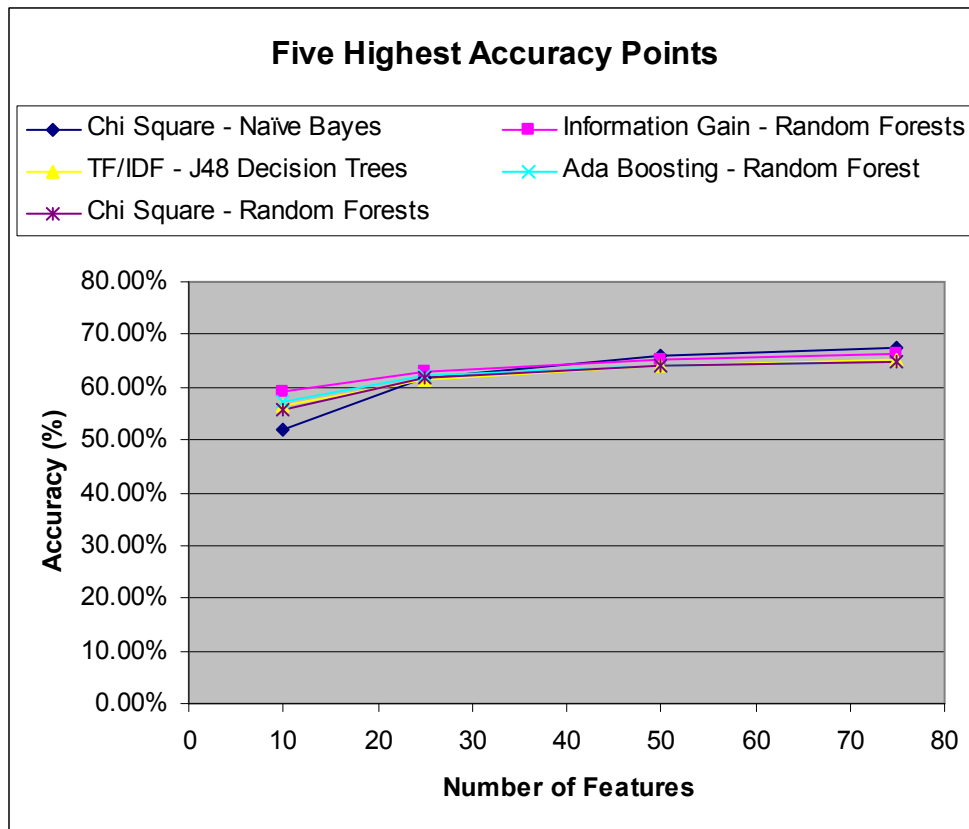
Methods	10 Features	25 Features	50 Features	75 Features
$\chi$ - Square	50.09%	53.83%	56.79%	57.40%
Information Gain	51.42%	53.90%	57.34%	60.29%
TF/IDF	48.87%	50.36%	55.32%	55.98%
Ada Boosting	49.08%	51.57%	53.77%	56.65%



**Figure 4.10: Graph For Accuracy of Support Vector Machines for Features Selected Using Proposed Feature Selection Methods.**

**4.12: Results of Five Highest Accuracy Points For the Feature Selection Methods evaluated using the ML Algorithms.**

Feature Selection	Evaluation Method	10 Features	25 Features	50 Features	75 Features
TF/IDF	J48 Decision Trees	52.02%	61.85%	65.93%	67.63%
Information Gain	Random Forests	59.1798%	62.9281%	65.0509%	66.1622%
$\chi$ - Square	Naïve Bayes	56.64%	61.46%	64.00%	65.28%
$\chi$ - Square	Random Forests	57.17%	62.23%	64.24%	64.86%
Ada Boosting	Random Forests	55.74%	61.68%	64.26%	64.70%



**Figure 4.11: Graph Showing the Five Highest Accuracy Points For the Feature Selection Methods evaluated using the ML Algorithms.**

## CHAPTER 5

### FUTURE WORK

In this thesis we have proposed methods for approximating Forward Selection of features in Information Retrieval (IR) problems using Machine Learning (ML) methods. This chapter discusses some work that we can do in future. They include, using second order statistic for selecting features, feature selection using Limited Forward Selection and Genetic Algorithms for selecting relevant features.

#### 5.1 Using Second Order Statistic

The phrase *order statistic* refers to statistical methods that depend only on the *ordering* of the data. The first order statistic of a random sample is the smallest element of the sample. The second order statistic is the second smallest. We can use second order statistic in order to get good features for classification. For instance, we can select a pairs of features based on some criteria such as, frequency or the similarity of the context in which they occur and use them. In doing so, it gives importance to the best feature as well as the second best, which might also prove to be a good discriminator of the target function. For example, if we consider frequency of a word as the measure of importance of the features. Suppose we have document which belong to two categories *Automobile* and *Petroleum*. If we get the words *oil* and *motor* as the first and second best feature using the frequency statistic. Then it is more likely that they would belong to the *Automobile* category. Whereas, if we get the words *oil* and *drilling* as the two best features, then it is more likely that they belong to the *Petroleum* category.

## **5.2 Limited Forward Selection**

In this thesis we observed that Forward Selection (Miller, 1990) is a computationally expensive method and in the case of large data sets it might be intractable. As future work we can explore Limited Forward Selection (Deng, 1998) for selecting features. At each step, instead of considering all the remaining features for selecting one feature, the Limited Forward Selection method only tries a part of them which are more promising. For example, in the case of Forward Selection we execute the function approximator (Naïve Bayes in our case) for each feature in order to select one feature. But in the case of Limited Forward Selection, it sorts the accuracy of the model created by each feature and gives more importance to the ones with more accuracy.

## **5.3 Genetic Algorithms**

Apart from all the methods that we used for selecting features, we can also use Genetic Algorithm (Mitchell, 1997) for selecting good features. Genetic Algorithms gives the flexibility of finding new features using its various techniques such as, cross over and mutation which might help in classifying the documents better. We can use some fitness function to determine the relevancy of a selected feature. It can also be used in order to create new features from the selected relevant feature which would classify the documents better.

## **5.4 Term Frequency / Inverse Document Frequency (TF/IDF) Results**

The TF/IDF results in Table 4.5 show a lot of variation as compared to the other method results. For example, TF/IDF performs very well for J48 Decision Trees but not good for rest of the methods. Thus, in future, more research can be done to understand the reason behind the variation seen in the TF/IDF results.

## CHAPTER 6

### CONCLUSIONS

In this thesis we proposed some methods for selecting relevant features that would approximate the results of Forward Selection in a tractable way. Forward Selection is a very effective method for selection of relevant features; but it is impractical to use this approach in the case of large data set because of the time that it takes. We implemented the Forward Selection algorithm as a baseline method to compare the performance of our methods. Based on the experiments that we conducted, our results show comparable accuracy in classification of documents and significant gain in the process time of the algorithms as compared to the Forward Selection algorithm.

We also performed experiments with the whole data in order to observe the affect of using a subset of relevant features instead of all of them. The results showed that the selection of a subset of features leads to better performance of the learning methods as compared to the results that we got using the whole data set.

The methods that we proposed use the  $\chi$  - Square statistic to filter features, the Information Gain statistic to filter, the Term Frequency / Inverse Document Frequency (TF/IDF) statistic to filter and a Boosting-based technique for selecting relevant features. We evaluated each of these methods of feature selection using various ML algorithms including, Naïve Bayes, Random Forests, J48 Decision Tree classification algorithm and a simple form of Naïve Bayes algorithm called Simple Count.

## BIBLIOGRAPHY

- [Breiman, 2001] Leo Breiman, “Random Forests”, *Machine Learning*, 45(1), p. 5-32, 2001.
- [Deng and Moore, 1998] Kan Deng and Andrew W. Moore, “On the Greediness of Feature Selection Algorithms”, in *the Proceedings of the Fifteenth International Conference on Machine Learning (ICML)*, Madison, Wisconsin, USA, p.62-77, June 1998.
- [Denoyer and Gallinari, 2006] Ludovic Denoyer and Patrick Gallinari, “The Wikipedia XML Corpus”, *INitiative for the Evaluation of XML Retrieval (INEX)*, 2006.
- [Denoyer and Gallinari, 2007] Ludovic Denoyer and Patrick Gallinari, “Document Mining Track”, *INitiative for the Evaluation of XML Retrieval (INEX)*, 2007.
- [Dietterich 1998] Thomas G. Dietterich, “Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms”, *Neural Computation*, 10(7), p. 1895-1923, 1998.
- [Freund and Schapire, 1996] Yoav Freund and Robert E. Schapire, “Experiments with a New Boosting Algorithm”, in *the Proceedings of the Thirteenth International Conference on Machine Learning (ICML)*, p. 148-156, Bari, Italy, July 1996.
- [Hansen and Salamon, 1990] N.K. Hansen and P. Salamon, “Neural Network Ensembles”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12, p 993-1001, 1990.
- [Joachims/1999a] T. Joachims, “Making Large-Scale SVM Learning Practical”, in *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola (ed.), MIT Press, 1999.
- [Joachims/2002b] T. Joachims and Thorsten, “Learning to Classify Text Using Support Vector Machines”, 668, Kluwer, 2002.
- [John, Kohavi and Pflieger, 1994] G. John, R. Kohavi, and Pflieger, K, “Irrelevant features and the subest selection problem”, in *the Proceedings of the Eleventh International Conference on Machine Learning (ICML)*, San Francisco, p. 121-129, 1994.
- [Kohavi and John, 1997] Ron Kohavi and George H. John, “Wrappers for Feature Subset Selection”, *Artificial Intelligence*, 97(1-2), p. 273-324, December 1997.

- [Langley, 1992] Pat Langley, Wayne Iba, Kevin Thompson, “An Analysis of Bayesian Classifiers”, in *the Proceedings of the Tenth Conference on Artificial Intelligence*, San Jose, California, USA, p. 233-228, 1992.
- [Miller, 1990] A. J. Miller, “Subset Selection in Regression”, Chapman and Hall, 1990.
- [Minsky and Papert, 1969] M. Minsky and S. Papert, “Perceptrons – An Introduction to Computational Geometry”, The MIT Press, Cambridge, Massachusetts, USA, 1969.
- [Mitchell, 1977] T. M. Mitchell, *Machine Learning*, McGraw Hill, 1997.
- [Quinlan, 1986] J. R. Quinlan, “Induction of decision trees”, *Machine Learning*, 1, p. 81-106, 1986.
- [Quinlan, 1993] *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, Inc., Los Altos, California, USA, 1993.
- [Runyon, 1977] K. E. Runyon, *Consumer Behavior and the Practice of Marketing*, Columbus, Ohio, USA, Charles E. Merrill Publishing Company, 1977.
- [Salton, 1968] J., Gerard Salton and Michael Lesk, “Computer Evaluation of Indexing and Text Processing”, *J. ACM*, 15(1), p. 8-36, 1968.
- [Schapire, 1990] Robert E. Schapire, “The Strength of Weak Learnability”, *Machine Learning*, 5(2), p. 197-227, 1990.
- [Schlimmer, 1993] J. C. Schlimmer, “Efficiently Inducing Determinations: A Complete and Systematic Search Algorithm that Uses Optimal Pruning”, in *the Proceedings of Tenth International Conference on Machine Learning (ICML)*, Amherst, Massachusetts, USA, p.284-290, 1993.
- [Selfridge, 1993] Oliver G. Selfridge, “The Gardens of Learning: A Vision for AI”, *AI Magazine Date*, 14(2), p. 36 – 48, 1993.
- [Vapnik, 1995] Vladimir N. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag New York, Inc, New York, New York USA, 1995.
- [Witten and Frank, 2005] Ian H. Witten; Eibe Frank, *Data Mining: Practical machine learning tools and techniques, 2nd Edition*, Morgan Kaufmann, San Francisco, California, USA, 2005.