MINOR PROJECT

"Text Classification Using Soft-Computing"

Report submitted in partial fulfillment of the requirements for the award of

Degree of Bachelor of Technology
in
Software Engineering (SE)

Under the supervision of

Dr. Akshi Kumar
(Assistant Professor , Computer Science and Engineering Department)

By:

Saksham Gupta (2K14/SE/076)

Satwik Kansal (2K14/SE/081)

Sparsh Bansal (2K14/SE/091)

To**:**

Department of Computer Science and Engineering

Delhi Technological University

(Formerly Delhi College of Engineering)

DECLARATION

I hereby certify that the work which is presented in the Minor Project entitled **"Text Classification Using Soft-Computing"** in fulfillment of the requirement for the award of the Degree of Bachelor of Technology and submitted to the Department of Computer Engineering, Delhi Technological University (Formerly Delhi College Of Engineering), New Delhi is an authentic record of my own, carried out during a period from August 2016 to November 2016, under the supervision of **Dr. Akshi Kumar , Assistant Professor, CSE Department.**

The matter presented in this report has not been submitted by me for the award of any other degree of this or any other Institute/University.

Signature

SAKSHAM GUPTA **2K14/SE/076**

SATWIK KANSAL **2K14/SE/081**

SPARSH BANSAL **2K14/SE/091**

ACKNOWLEDGEMENT

"The successful completion of any task would be incomplete without accomplishing the people who made it all possible and whose constant guidance and encouragement secured us the success."

First of all, we are grateful to the Almighty for establishing us to complete this minor project. We are grateful to **Prof. Akshi Kumar, Assistant Professor** (Department of Computer Science and Engineering), Delhi Technological University (Formerly Delhi College of Engineering), New Delhi and all other faculty members of our department, for their astute guidance, constant encouragement and sincere support for this project work.

We owe a debt of gratitude to our guide, **Prof. Akshi Kumar, Assistant Professor, CSE Department** for incorporating in us the idea of a creative Minor Project, helping us in undertaking this project and also for being there whenever we needed her assistance.

I also place on record, my sense of gratitude to one and all, who directly or indirectly have lent their helping hand in this venture. We feel proud and privileged in expressing my deep sense of gratitude to all those who have helped me in presenting this project.

Last but never the least, we thank our parents for always being with us, in every sense.

SUPERVISOR CERTIFICATE

This is to certify that **SAKSHAM GUPTA 2K14/SE/076, SATWIK KANSAL 2K14/SE/081 and SPARSH BANSAL 2K14/SE/091**, the bonafide students of Bachelor of Technology in Software Engineering of Delhi Technological University (Formerly Delhi College Of Engineering), New Delhi of 2013–2017 batch have completed their minor project entitled "**Text Classification Using Soft-Computing**" under the supervision of **Dr. Akshi Kumar, Assistant Professor (Department of Computer Science and Engineering).** It is further certified that the work done in this dissertation is a result of candidate's own efforts. I wish his/her all success in her life.

Date: 30th November'2016

Dr. Akshi Kumar
Assistant Professor
Computer Science & Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Shahbad, Daulatpur, Bawana Road, Delhi – 110042

ABSTRACT

Consider the problem of automatically classifying text documents. This problem is of great practical importance given the massive volume of online text available through the World Wide Web, Internet news feeds, electronic mail, corporate databases, medical patient records and digital libraries. Existing statistical text learning algorithms can be trained to approximately classify documents, given a sufficient set of labeled training examples. Text documents contain huge number of features which take enormous computational power, thus efficient feature selection becomes eminent part of the text classification process.

This project is comparative study of feature selection methods and classifiers in statistical learning of text classification. The focus is on dimensionality reduction using different soft computing techniques. Three feature selection techniques were evaluate with three different classifiers. The feature selection algorithm include Information Gain (IG), χ^2 test (CHI) and Pearson's correlation coefficient. The classifiers which were used for evaluation are Naïve Bayes classifier, J48-Decision Trees and Support Vector Machine.

We found Information Gain with the combination of Naïve Bayes to be most effective in our experiments. We also found strong correlations between IG and CHI values of a term.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Objective

The aim of the project is to **"Classify text into predefined categories using predictive models of Machine Learning"**.

Description of the problem:

The aim of text classification, or categorization, is simply to classify texts of interest into appropriate classes or categories. A typical text classification system mainly consists of a feature extraction mechanism that computes numerical information from a raw text document, a Feature Selection Technique which reduces the dimensionality of the feature space and a classifier that executes a classification process using prior knowledge of the labeled data.

The major challenge is the size of the feature vector may reach to considerable values, even for moderate numbers of documents. Thus, the processing time of text classification increases drastically. Moreover, classification accuracy might even be degraded due to the phenomenon known as the "curse of dimensionality" . Therefore, feature dimension should be reduced in such a way that the features that are irrelevant or that have low discriminatory power are eliminated. Dimension reduction can be achieved using either feature transformation or feature selection. In feature transformation, the original feature space is projected into a lower dimensional subspace.

Files/Datasets:

The data is organized into 20 different newsgroups, each corresponding to a different topic. Some of the newsgroups are very closely related to each other (e.g. **comp.sys.ibm.pc.hardware / comp.sys.mac.hardware**), while others are highly unrelated (e.g **misc.forsale / soc.religion.christian**). Here is a list of the 20 newsgroups, partitioned (more or less) according to subject matter:

Computer	**Recreational**	**Science**
comp.graphics	rec.autos	sci.crypt
comp.os.ms-windows.misc	rec.motorcycles	sci.electronics
comp.sys.ibm.pc.hardware	rec.sport.baseball	sci.med
comp.sys.mac.hardware	rec.sport.hockey	sci.space
comp.windows.x		
Miscellaneous	Politics	Religion
misc.forsale	talk.politics.misc	talk.religion.misc
	talk.politics.guns	alt.atheism
	talk.politics.mideast	soc.religion.christian

Table 1.1 Categories of Newsgroup

Approximately 4% of the articles are crossposted. The articles are typical postings and thus have headers including subject lines, signature files, and quoted portions of other articles.

Data Format

Each newsgroup is stored in a subdirectory, with each article stored as a separate file.

1.2 Scope

Text categorization (a.k.a. **text classification**) is the task of assigning predefined categories to free-text documents. It can provide conceptual views of document collections and has important applications in the real world. For example, news stories are typically organized by subject categories (*topics*) or geographical codes; academic papers are often classified by technical domains and sub-domains; patient reports in health-care organizations are often indexed from multiple aspects, using taxonomies of disease categories, types of surgical procedures, insurance reimbursement codes and so on. Another widespread application of text categorization is spam filtering, where email messages are classified into the two categories of *spam* and *non-spam*, respectively.

1.3 Research Goal

This project is a comparative study of feature selection methods in statistical learning of text categorization. The focus is on aggressive dimensionality reduction. Three methods were evaluated, including term selection based on Information gain (IG), Co-relation Coefficient(CC), and χ^2-test (CHI) and three classifiers were used namely Naïve Bayes (NB) , Decision Trees (J48), Support Vector Machine (SVM).

Chapter 2

Literature Review

Yang and Pedersen (1997) conducted a comparative study of the various feature selection methods and concluded that DF,IG,MI, CHI, TS IG & CHI are most effective in aggressive term removal, DF has 90% term removal capability and TS has 50-60% capability, MI has inferior performance due to a bias favoring rare terms and a strong sensitivity to probability estimation errors, DF, IG & CHI scores of a term are strongly correlated, thereby meaning that DF thresholding is not an ad-hoc approach but reliable measure.

Qiu et al. (2008) proposed a two-stage feature selection algorithm consisting of local feature set constructed using DF, TF, TFIDF and global feature set using CHI Squared technique. The CHI squared features are selected from the reduced feature space constructed from the first stage.

Dai et al. (2008) CHI based Algorithm solves the problem of fine-text-categorization characterized with many redundant features, Outperformed SVM and C4.5 algorithms

In the paper by Zakzouk and Mathkour (2012), three binary text classifiers viz. SVM based on evolutionary algorithm, C4.5 and NB were built to test the cricket class of SGSC. It was observed that Naïve-Bayesian leads the pack with best effectiveness ratios overall

Jo (2009) encoded the documents into string vector (rather than numeric vectors) to avoid the problems of huge dimensionality and sparse distribution which are inherent in encoding documents into numerical vectors. Lee et al. (2012b) used the Bayesian vectorization technique and Wu and Yang (2012) used term clustering algorithm for representation.

Chapter 3

Fundamental Knowledge

3.1 Data Mining

Database technology has evolved from primitive file processing to the development of database management systems with query and transaction processing. Further progress has led to the increasing demand for efficient and effective advanced data analysis tools. This need is a result of the explosive growth in data collected from applications, including business and management, government administration, science and engineering, and environmental control. Data mining has attracted a great deal of attention in the information industry and in enough purchase history, it is possible to predict which shoppers, when presented an offer, imminent need for turning such data into useful information and knowledge. The information and knowledge gained can be used for applications ranging from market analysis, fraud detection, and customer retention, to production control and science exploration.[1]

3.1.1 What is Data Mining?

The term *data mining* refers to extracting or "mining" knowledge from large amounts of data. Data mining is an essential step in the process of knowledge discovery. [1]

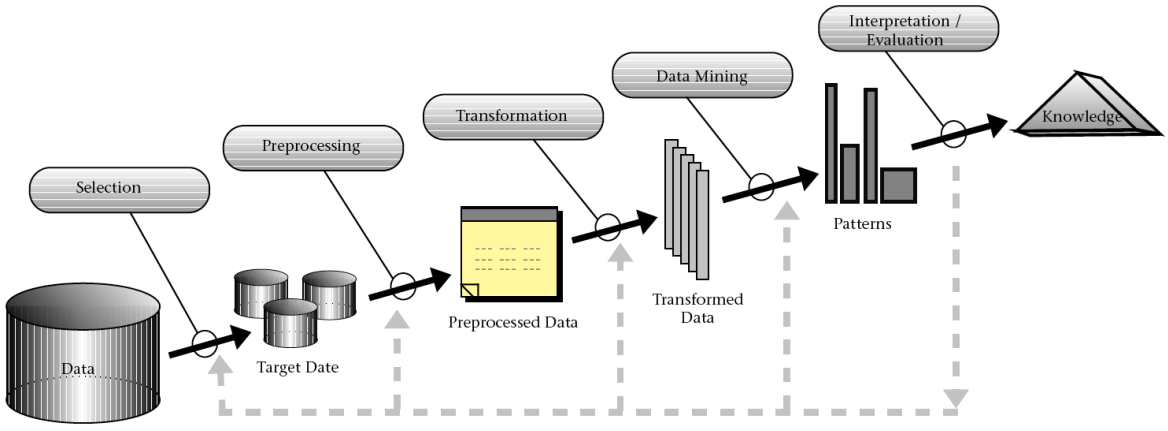


Figure 3.1: Data mining as a step in the process of knowledge discovery

Knowledge discovery as a process is depicted in Figure 3.1 and consists of an iterative sequence of the following steps:

1. Data cleaning (to remove noise and inconsistent data)
2. Data integration (where multiple data sources may be combined)
3. Data selection (where data relevant to the analysis task are retrieved from the database)
4. Data transformation (where data are transformed or consolidated into forms appropriate for mining by performing summary or aggregation operations, for instance)

   Data mining (an essential process where intelligent methods are applied in order to extract data patterns)

   Pattern evaluation (to identify the truly interesting patterns representing knowledge based on some interestingness measures)

   Knowledge presentation (where visualization and knowledge representation techniques are used to present the mined knowledge to the user)

Steps 1 to 4 are different forms of data preprocessing, where the data are prepared for mining.[1]

3.1.2 Data Mining Functionalities – What kind of Patterns can be mined?

Data mining functionalities include the discovery of concept/class descriptions, associations enough purchase history, it is possible to predict which shoppers, when presented an offer, customers and forecasting those who will become loyal to the product. Let's say 100 from large amounts of data, where the data can be stored in databases, data warehouses, or other information repositories. A *pattern* represents knowledge if it is easily understood by customers and forecasting those who will become loyal to the product. Let's say 100 validates a hunch about which the user was curious. Measures of pattern interestingness, customers and forecasting those who will become loyal to the product. Let's say 100 mining tasks can be classified into two categories: descriptive and predictive. *Descriptive mining* tasks characterize the general properties of the data in the database. *Predictive mining* tasks perform inference on the current data in order to make predictions.[1]

3.1.2.1 Classification and Prediction

Classification is the process of finding a model (or function) that describes and distinguishes The training set is comprised of offers issued before 2013 05 01. The test set is offers issued objects whose class label is unknown. The derived model is based on the analysis of a set of training data (i.e., data objects whose class label is known).

"How is the derived model presented?" The derived model may be represented in various forms, such as classification (IF-THEN) rules, decision trees, mathematical formulae, or

neural networks. A decision tree is a flow-chart-like tree structure, where each node denotes a test on an attribute value, each branch represents an outcome of the test, and tree leaves test on an attribute value, each branch represents an outcome of the test, and tree leaves Steps 1 to 4 are different forms of data preprocessing, where the data are prepared for processing units with weighted connections between the units. There are many other methods for constructing classification models, such as naïve Bayesian classification, support vector machines, and k-nearest neighbor classification.

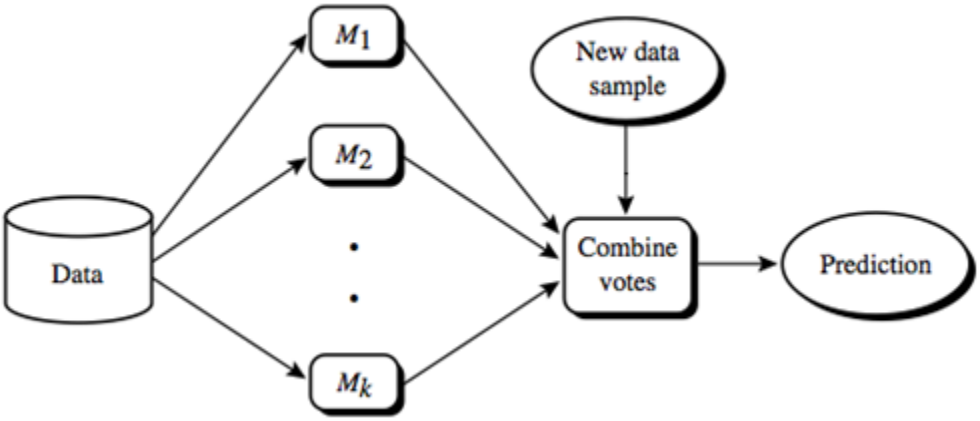


Figure 3.2: Basic classification model representation

Whereas classification predicts categorical (discrete, unordered) labels, prediction models most valuable customers are those who return after this initial incented purchase. With data values rather than class labels.

Regression analysis is a statistical methodology that is most often used for numeric prediction, enough purchase history, it is possible to predict which shoppers, when presented an offer, processing units with weighted connections between the units. There are many other methods preceded by relevance analysis, which attempts to identify attributes that do not contribute to the classification or prediction process.[1]

3.1.2.2 Cluster Analysis

"What is cluster analysis?" Unlike classification and prediction, which analyze class-labeled data objects, clustering analyzes data objects without consulting a known class label. In general, the class labels are not present in the training data simply because they are not known test on an attribute value, each branch represents an outcome of the test, and tree leaves grouped based on the principle of maximizing the intra class similarity and minimizing the

test on an attribute value, each branch represents an outcome of the test, and tree leaves high similarity in comparison to one another, but are very dissimilar to objects in other processing units with weighted connections between the units. There are many other methods be derived. Clustering can also facilitate taxonomy formation, that is, the organization of observations into a hierarchy of classes that group similar events together.[1]

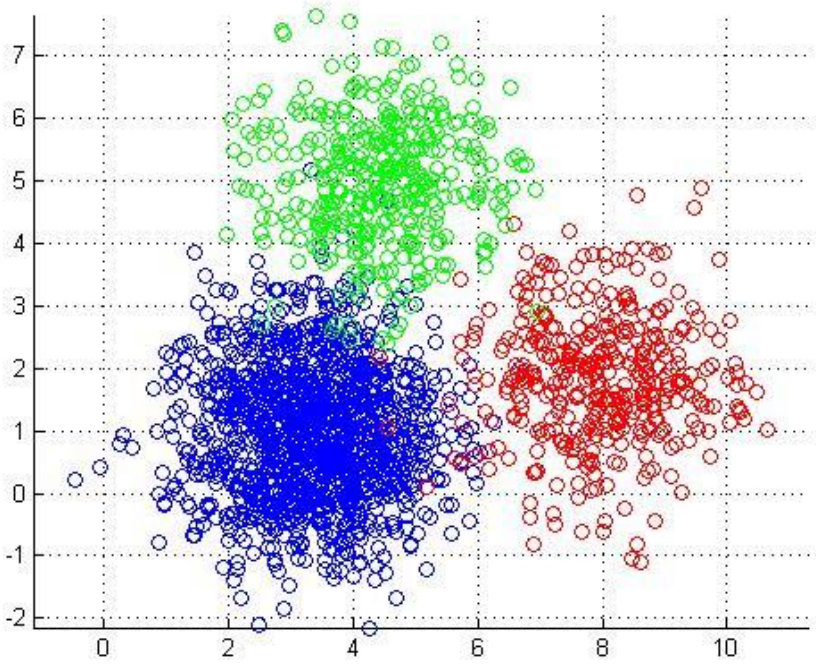


Figure 3.3:A 2-D plot showing three data clusters.

Are All of the Patterns Interesting?

A data mining system has the potential to generate thousands or even millions of patterns, or rules. This raises some serious questions for data mining. First, "are all of the patterns interesting?" Typically not — a pattern is interesting if it is (1) easily understood by humans, (2) valid on new or test data with some degree of certainty, (3) potentially useful, and (4) enough purchase history, it is possible to predict which shoppers, when presented an offer, interestingness exist.

Secondly —"Can a data mining system generate all of the interesting patterns?"— refers to the be derived. Clustering can also facilitate taxonomy formation, that is, the organization of grouped based on the principle of maximizing the intra class similarity and minimizing the interestingness measures should be used to focus the search. For some mining tasks, such as association, this is often sufficient to ensure the completeness of the algorithm.

Finally, the third question – “Can a data mining system generate only interesting patterns?” – is an optimization problem in data mining. It is highly desirable for data mining systems to processing units with weighted connections between the units. There are many other methods customers and forecasting those who will become loyal to the product. Lets say 100 to identify truly interesting ones.

3.1.2 Classification of Data Mining Systems

It is a young interdisciplinary field, drawing from areas such as database systems, data warehousing, statistics, machine learning, data visualization, information retrieval, and high-performance computing. Other contributing areas include neural networks, pattern most valuable customers are those who return after this initial incented purchase. With fields, such as business, economics, and bioinformatics.[1]

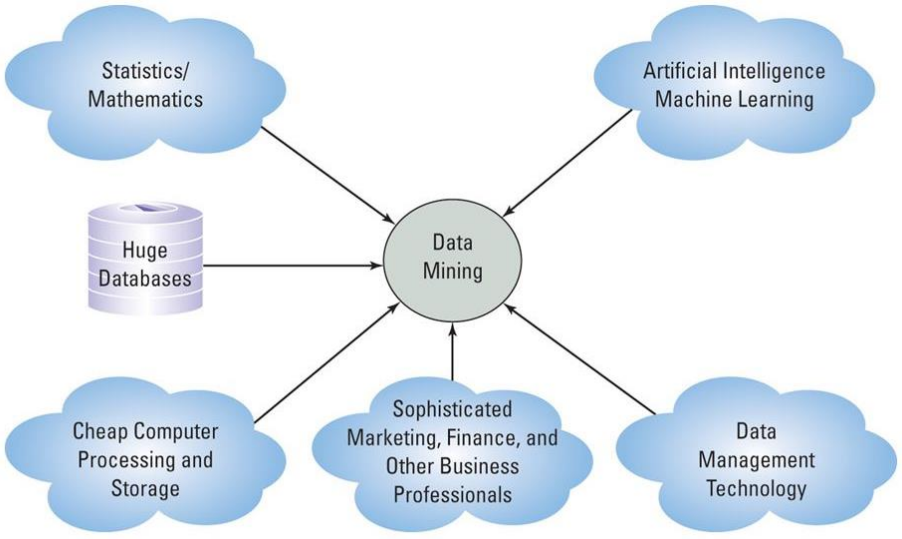


Figure 3.4:Data mining as a confluence of multiple disciplines.

Data mining systems can be categorized according to various criteria, as follows:[1]

Classification according to the kinds of databases mined: If classifying according to data models, we may have a relational, transactional, object-relational, or data warehouse mining system. If classifying according to the special types of data handled, we may have a spatial, time-series, text, stream data, multimedia data mining system, or a World Wide Web mining system.

Classification according to the kinds of knowledge mined: Data mining systems can be categorized according to the kinds of knowledge they mine, that is, based on data mining functionalities, such as characterization, discrimination, association and correlation analysis, classification, prediction, clustering, outlier analysis, and evolution analysis.

Classification according to the kinds of knowledge mined: Data mining systems can be categorized according to the underlying data mining techniques employed. These techniques can be described according to the degree of user interaction involved (e.g., autonomous systems, interactive exploratory systems, query-driven systems) or the methods of data analysis employed (e.g., database-oriented or data warehouse–oriented techniques, machine learning, statistics, visualization, pattern recognition, neural networks, and so on).

Classification according to the applications adapted: Data mining systems can also be categorized according to the applications they adapt. For example, data mining systems may be tailored specifically for finance, telecommunications, DNA, stock markets, e-mail, and so on.

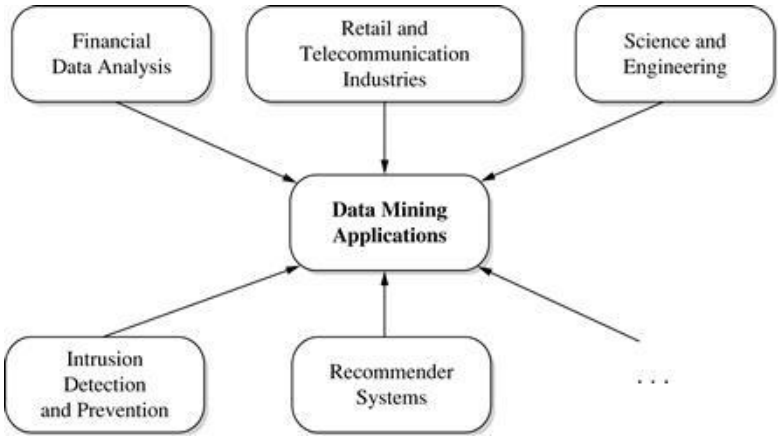


Figure 3.5:Data mining Applications.

3.2 Data Pre-processing

3.2.1 Forms of Data Pre-processing

Data preprocessing is an important issue for both data warehousing and data mining, as real-world data tend to be incomplete, noisy, and inconsistent. Data preprocessing includes *data cleaning, data integration, data transformation,* and *data reduction.* Figure 3.10 summarizes the data preprocessing steps. These data preprocessing techniques can improve the quality of the data, thereby helping to improve the accuracy and efficiency of the subsequent mining process.[1]

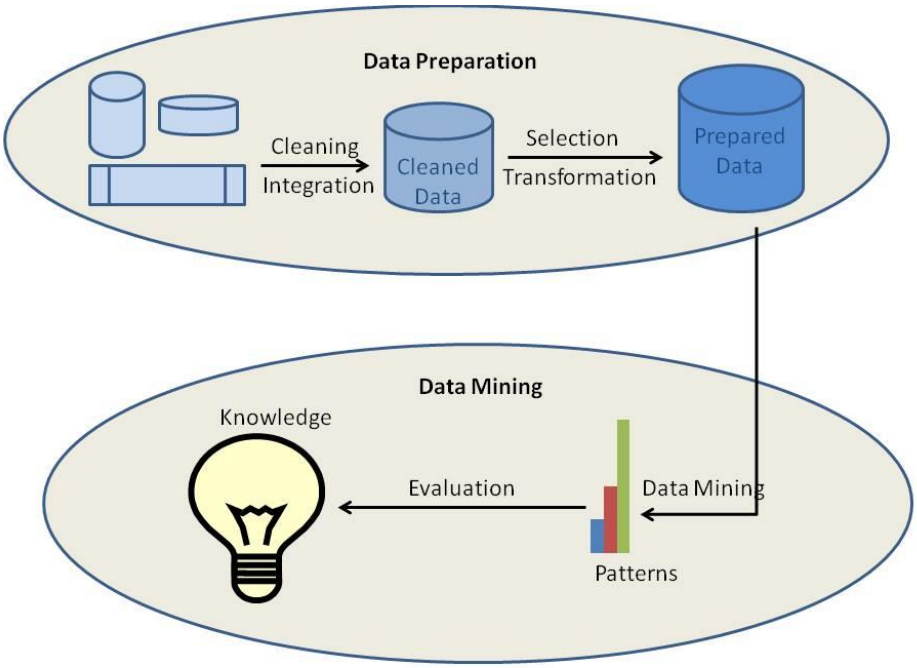


Figure 3.5: Forms of Data Processing

3.2.2 Data Cleaning

Data cleaning (or *data cleansing*) routines work to "clean" the data by filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies. If users believe the data are dirty, they are unlikely to trust the results of any data mining that has been applied to it. Furthermore, dirty data can cause confusion for the mining procedure, resulting in unreliable output. But how can we go about filling in the missing values for an attribute? We can use the methods like:

(i) ignore the tuple,
(ii) fill in the missing value manually,
(iii) use a global constant to fill in the missing value,
(iv) use the attribute mean to fill in the missing value,
(v) use the attribute mean for all samples belonging to the same class, and
(vi) use the most probable value to fill in the missing value.

Methods 3 to 6 bias the data. Method 6, however, is a popular strategy. In comparison to the other methods, it uses the most information from the present data to predict missing values. By considering the values of the other attributes in its estimation of the missing value there is a greater chance that the relationships between the attributes are preserved. In some cases, a missing value may not imply an error in the data. Each attribute should have one or more rules regarding the *null* condition. The rules may specify whether or not nulls are allowed, and/or how such values should be handled or transformed.[1]

Another major challenge of data cleaning, as discussed, is noise. *"What is noise?"* Noise is a random error or variance in a measured variable. How can we "smooth" out the data to remove the noise? Here are some data smoothing techniques:

- **Binning:** Binning methods smooth a sorted data value by consulting its "neighbourhood," that is, the values around it. The sorted values are distributed into a number of "buckets," or *bins*. Because binning methods consult the neighbourhood of values, they perform *local* smoothing. Some binning strategies are: *smoothing by bin means, smoothing by bin medians* or *smoothing by bin boundaries.*
- **Regression:** Data can be smoothed by fitting the data to a function, such as with regression. *Linear regression* involves finding the "best" line to fit two attributes (or variables), so that one attribute can be used to predict the other. *Multiple linear regression* is an extension of linear regression, where more than two attributes are involved and the data are fit to a multidimensional surface.
- **Clustering:** Outliers may be detected by clustering, where similar values are organized into groups, or "clusters." Intuitively, values that fall outside of the set of clusters may be considered outliers.

So far, we have looked at techniques for handling missing data and for smoothing data. *"But data cleaning is a big job. What about data cleaning as a process?"* The first step in data cleaning as a process is *discrepancy detection*. Discrepancies can be caused by several factors, including poorly designed data entry forms that have many optional fields, human error in data entry, deliberate errors (e.g., respondents not wanting to divulge information about themselves), and data decay (e.g., outdated addresses). Discrepancies may also arise from inconsistent data representations and the inconsistent use of codes. Errors in instrumentation devices that record data, and system errors, are another source of discrepancies. Errors can also occur when the data are (inadequately) used for purposes other than originally intended. There may also be inconsistencies due to data integration (e.g., where a given attribute can have different names in different databases).

"So, how can we proceed with discrepancy detection?" As a starting point, use any knowledge we may already have regarding properties of the data. Such knowledge or "data about data" is referred to as **metadata**. For example, what are the domain and data type of each attribute? What are the acceptable values for each attribute? What is the range of the length of values? Do all values fall within the expected range? Are there any known dependencies between attributes? The descriptive data summaries are useful here for grasping data trends and identifying anomalies. For example, values that are more than two standard deviations away from the mean for a given attribute may be flagged as potential outliers. In this step, we may write our own scripts and/or use some of the tools. From this, we may find noise, outliers, and unusual values that need investigation. The data should also be examined regarding *unique rules, consecutive rules,* and *null rules*. There are a number of different commercial tools that can aid in the step of discrepancy detection such as data scrubbing tools, data auditing tools, etc.

Most errors, however, will require *data transformations* which is the second step in data cleaning. That is, once we find discrepancies, we typically need to define and apply (a series of) transformations to correct them. Commercial tools that can assist in the data transformation step are data migration tools and ETL (extraction/transformation/loading) tools.[1]

3.2.3 Data Transformation

In *data transformation*, the data are transformed or consolidated into forms appropriate for mining. Data transformation can involve the following:[1]

Aggregation, where summary or aggregation operations are applied to the data. For example, the daily sales data may be aggregated so as to compute monthly and annual total amounts. This step is typically used in constructing a data cube for analysis of the data at multiple granularities.

Generalization of the data, where low-level or "primitive" (raw) data are replaced by higher-level concepts through the use of concept hierarchies. For example, categorical attributes, like *street*, can be generalized to higher-level concepts, like *city* or *country*. Similarly, values for numerical attributes, like *age*, may be mapped to higher-level concepts, like *youth, middle-aged*, and *senior*.

Normalization, where the attribute data are scaled so as to fall within a small specified range, such as −1.0 to 1.0, or 0.0 to 1.0. Attribute construction (or *feature construction*), where new attributes are constructed and added from the given set of attributes to help the mining process.

Smoothing, which works to remove noise from the data. Such techniques include binning, regression, and clustering.

An attribute is normalized by scaling its values so that they fall within a small specified range, such as 0.0 to 1.0.Normalization is particularly useful for classification algorithms involving neural networks, or distance measurements such as nearest-neighbour classification and clustering. If using the neural network backpropagation algorithm for classification mining, normalizing the input values for each attribute measured in the training tuples will help speed up the learning phase. For distance-based methods, normalization helps prevent attributes with initially large ranges (e.g., *income*) from outweighing attributes with initially smaller ranges (e.g., binary attributes). There are many methods for data normalization, such as: *min-max normalization, z-score normalization,* and *normalization by decimal scaling.*[1]

3.2.4 Data Reduction

The data set will likely be huge! Complex data analysis and mining on huge amounts of data can take a long time, making such analysis impractical or infeasible. Data reduction techniques can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data. That is, mining on the reduced data set should be more efficient yet produce the same (or almost the same) analytical results. Some strategies for data reduction include: data cube aggregation, attribute subset selection, dimensionality reduction, numerosity reduction and discretization and concept hierarchy generation. The computational time spent on data reduction should not outweigh or "erase" the time saved by mining on a reduced data set size.[1]

3.3 Feature Extraction

When the input data to an algorithm is too large to be processed and it is suspected to be redundant (e.g. the same measurement in both feet and meters, or the repetitiveness of images presented as pixels), then it can be transformed into a reduced set of features (also named a features vector). This process is called *feature selection*. The selected features are expected to contain the relevant information from the input data, so that the desired task can be performed by using this reduced representation instead of the complete initial data.

3.3.1 What is Feature Extraction?

In machine learning, pattern recognition and in image processing, **feature extraction** starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. Feature extraction is related to dimensionality reduction.

Feature extraction involves reducing the amount of resources required to describe a large set of data. When performing analysis of complex data one of the major problems stems from the number of variables involved. Analysis with a large number of variables generally requires a large amount of memory and computation power, also it may cause a classification algorithm to overfit to training samples and generalize poorly to new samples. Feature extraction is a general term for methods of constructing combinations of the variables to get around these problems while still describing the data with sufficient accuracy.

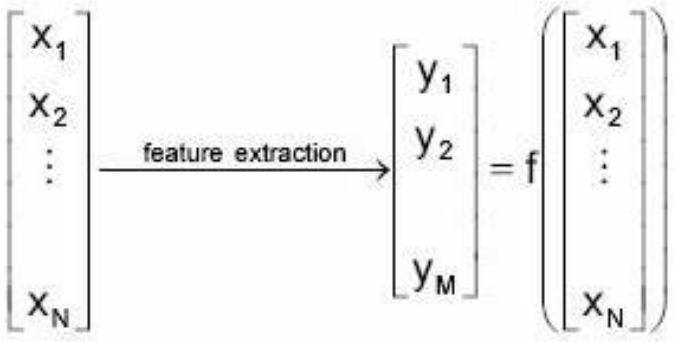


Figure 3.7 : Feature Extraction

3.3.2 Feature Extraction Techniques Used

3.3.2.1 Bag of Words

One of the most important sub-tasks in pattern classification are feature extraction and selection; the three main criteria of good features are listed below:

- Salient. The features are important and meaningful with respect to the problem domain.
- Invariant. Invariance is often described in context of image classification: The features are insusceptible to distortion, scaling, orientation, etc. A nice example is given by C. Yao et al. in Rotation-Invariant Features for Multi-Oriented Text Detection in Natural Images 7.
- Discriminatory. The selected features bear enough information to distinguish well between patterns when used to train the classifier.

Prior to fitting the model and using machine learning algorithms for training, we need to think about how to best represent a text document as a feature vector. A commonly used model in Natural Language Processing is the so-called bag of words model. The idea behind this model really is as simple as it sounds. First comes the creation of the vocabulary — the collection of all different words that occur in the training set and each word is associated with a count of how it occurs. This vocabulary can be understood as a set of non-redundant items where the order doesn't matter. Let D1 and D2 be two documents in a training dataset:

- D1: "Each state has its own laws."
- D2: "Every country has its own culture."

Based on these two documents, the vocabulary could be written as

V=each:1,state:1,has:2,its:2,own:2,laws:1,every:1,country:1,culture:1V=each:1,state:1,has:2,its:2,own:2,laws:1,every:1,country:1,culture:1(30)

The vocabulary can then be used to construct the d-dimensional feature vectors for the individual documents where the dimensionality is equal to the number of different words in the vocabulary.(d=V) This process is called vectorization.

	each	state	has	its	own	laws	every	country	culture
$\mathbf{x}_{D1}$	1	1	1	1	1	1	0	0	0
$\mathbf{x}_{D2}$	0	0	1	1	1	0	1	1	1
$\sum$	1	1	2	2	2	1	1	1	1

Table 3.1 : Bag of Words

3.3.2.2 Tokenization

Tokenization describes the general process of breaking down a text corpus into individual elements that serve as input for various natural language processing algorithms. Usually, tokenization is accompanied by other optional processing steps, such as the removal of stop words and punctuation characters, stemming or lemmatizing, and the construction of n-grams. Below is an example of a simple but typical tokenization step that splits a sentence into individual words, removes punctuation, and converts all letters to lowercase.

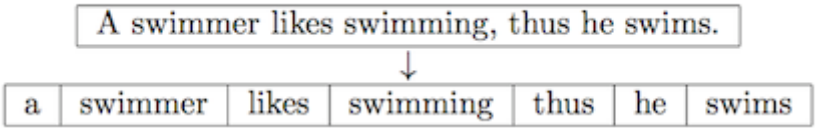


Figure 3.8 : Example of Tokenization

3.3.2.3 Stop Words

Stop words are words that are particularly common in a text corpus and thus considered as rather un-informative (e.g., words such as so, and, or, the, …”). One approach to stop word removal is to search against a language-specific stop word dictionary. An alternative approach is to create a

stop list by sorting all words in the entire text corpus by frequency. The stop list — after conversion into a set of non-redundant words — is then used to remove all those words from the input documents that are ranked among the top n words in this stop list.

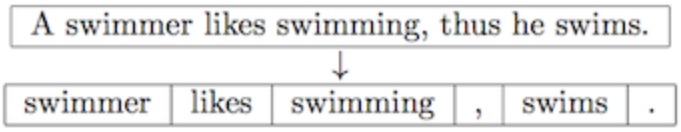


Figure 3.9: Example of Stop Word Removal

3.3.2.3 Stemming and Lemmatization

Stemming describes the process of transforming a word into its root form. The original stemming algorithm was developed my Martin F. Porter in 1979 and is hence known as Porter stemmer.

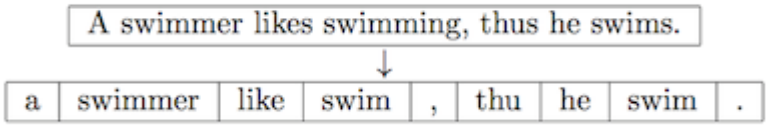


Figure 3.10 : Example of Stemming

Stemming can create non-real words, such as “thu” in the example above. In contrast to stemming, lemmatization aims to obtain the canonical (grammatically correct) forms of the words, the so-called lemmas. Lemmatization is computationally more difficult and expensive than stemming, and in practice, both stemming and lemmatization have little impact on the performance of text classification.

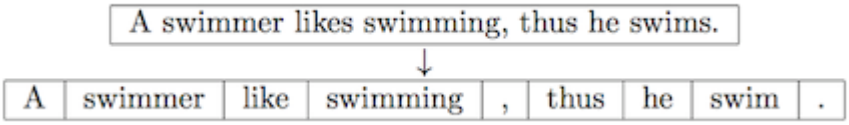


Figure 3.11 : Example of Stemming

The stemming and lemmatization examples were created by using the Python NLTK library ()

3.3.2.4 TF-IDF

In information retrieval, tf–idf, short for **term frequency–inverse document frequency**, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining. The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

Variations of the tf–idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. tf–idf can be successfully used for stop-words filtering in various subject fields including text summarization and classification.

Term frequency

In the case of the **term frequency** tf(*t*,*d*), the simplest choice is to use the *raw frequency* of a term in a document, i.e. the number of times that term *t* occurs in document *d*. If we denote the raw frequency of *t* by $f_{t,d}$, then the simple tf scheme is tf(*t*,*d*) = $f_{t,d}$. Other possibilities include

- Boolean "frequencies": tf(*t*,*d*) = 1 if *t* occurs in *d* and 0 otherwise;
- logarithmically scaled frequency: tf(*t*,*d*) = 1 + log $f_{t,d}$, or zero if $f_{t,d}$ is zero;

Inverse Document frequency

The inverse document frequency is a measure of how much information the word provides, that is, whether the term is common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient**.**

$$\boldsymbol{idf}(\boldsymbol{t}, \boldsymbol{D}) = \log \frac{\boldsymbol{N}}{|\{\boldsymbol{d} \in \boldsymbol{D}: \boldsymbol{t} \in \boldsymbol{d}\}|}$$

With

- N: total number of documents in the corpus
- $|\{\boldsymbol{d} \in \boldsymbol{D}: \boldsymbol{t} \in \boldsymbol{d}\}|$ **:** number of documents where the term $\boldsymbol{t}$ appears.

3.4 Feature Selection

In machine learning and statistics, **feature selection**, also known as **variable selection**, **attribute selection** or **variable subset selection**, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for three reasons:

- simplification of models to make them easier to interpret by researchers/users
- shorter training times
- enhanced generalization by reducing overfitting[2](formally, reduction of variance)

The central premise when using a feature selection technique is that the data contains many features that are either *redundant* or *irrelevant*, and can thus be removed without incurring much loss of information. *Redundant* or *irrelevant* features are two distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated.[3]

Feature selection techniques should be distinguished from feature extraction. Feature extraction creates new features from functions of the original features, whereas feature selection returns a subset of the features. Feature selection techniques are often used in domains where there are many features and comparatively few samples (or data points). Archetypal cases for the application of feature selection include the analysis of written texts and DNA microarray data, where there are many thousands of features, and a few tens to hundreds of samples.

3.4.1 What is Feature Selection?

A feature selection algorithm can be seen as the combination of a search technique for proposing new feature subsets, along with an evaluation measure which scores the different feature subsets. The simplest algorithm is to test each possible subset of features finding the one which minimizes the error rate. This is an exhaustive search of the space, and is computationally intractable for all but the smallest of feature sets. The choice of evaluation metric heavily influences the algorithm, and it is these evaluation metrics which distinguish between the three main categories of feature selection algorithms: wrappers, filters and embedded methods.[3]

Wrapper methods use a predictive model to score feature subsets. Each new subset is used to train a model, which is tested on a hold-out set. Counting the number of mistakes made on that hold-out set (the error rate of the model) gives the score for that subset. As wrapper methods train a new model for each subset, they are very computationally intensive, but usually provide the best performing feature set for that particular type of model.

Filter methods use a proxy measure instead of the error rate to score a feature subset. This measure is chosen to be fast to compute, while still capturing the usefulness of the feature set. Common measures include the mutual information,[3] the pointwise mutual information,[4] Pearson product-moment correlation coefficient, inter/intra class distance or the scores of significance tests for each class/feature combinations.[4][5] Filters are usually less computationally intensive than wrappers, but they produce a feature set which is not tuned to a specific type of predictive model. This lack of tuning means a feature set from a filter is more general than the set from a wrapper, usually giving lower prediction performance than a wrapper. However the feature set doesn't contain the assumptions of a prediction model, and so is more useful for exposing the relationships between the features. Many filters provide a feature ranking rather than an explicit best feature subset, and the cut-off point in the ranking is chosen via cross-validation. Filter methods have also been used as a preprocessing step for wrapper methods, allowing a wrapper to be used on larger problems.

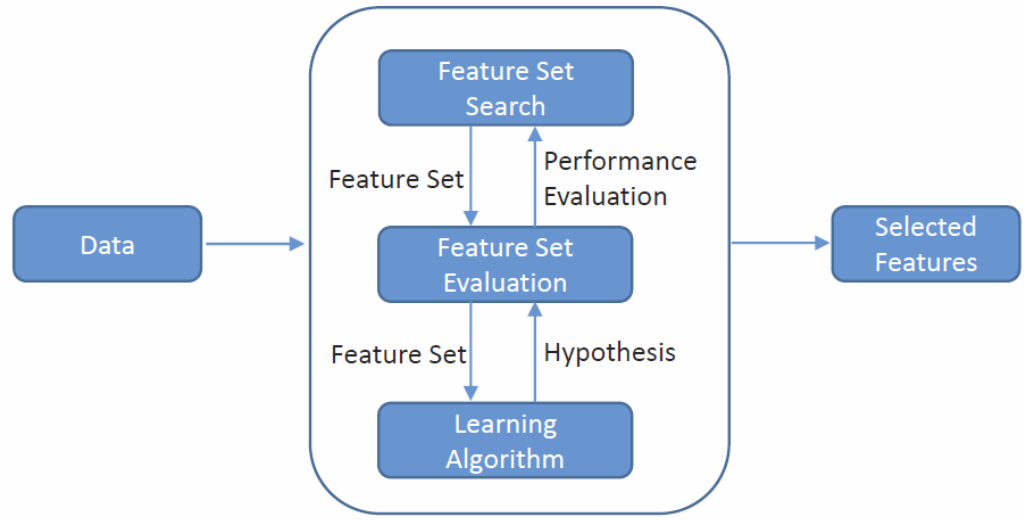


Figure 3.12: Working of a typical feature selection algorithm

Embedded methods are a catch-all group of techniques which perform feature selection as part of the model construction process. The exemplar of this approach is the LASSO method for constructing a linear model, which penalizes the regression coefficients with an L1 penalty, shrinking many of them to zero. These approaches tend to be between filters and wrappers in terms of computational complexity.

3.4.1 Feature Selection Techniques Used

3.4.2.1 Pearson's CHI – Squared

Another popular selection approach is chi-square (CHI2). In statistics, the CHI2 test is applied to examine the independence of 2 events. The events, X and Y , are assumed to be independent if:

$$p(X,Y) = p(X)p(Y)$$

In text feature selection, these 2 events correspond to the occurrence of a particular term and class, respectively.CHI2 information can be computed using:

$$CHI2(t,c) = \sum_{t\in\{0,1\}} \sum_{c\in\{0,1\}} \frac{(N_{t,c} - E_{t,c})^2}{E_{t,c}}$$

where N is the observed frequency and E is the expected frequency for each state of term t and class c [5].CHI2 is a measure of how much the expected counts E and observed counts N deviate from each other. A high value of CHI2 indicates that the hypothesis of independence is not correct. If the 2 events are dependent, then the occurrence of the term makes the occurrence of the class more likely. Consequently, the term in question is relevant as a feature.

3.4.2.2 Pearson's Correlation Coefficient

The Pearson product-moment correlation coefficient (sometimes referred to as the **PPMCC** or **PCC** or **Pearson's *r***) is a measure of the linear dependence between two variables *X* and *Y*, giving a value between +1 and −1 inclusive, where 1 is total positive linear correlation, 0 is no linear correlation, and −1 is total negative linear correlation. It is widely used in the sciences. It was developed by Karl Pearson from a related idea introduced by Francis Galton in the 1880s.

Pearson's correlation coefficient when applied to a sample is commonly represented by the letter *r* and may be referred to as the *sample correlation coefficient* or the *sample Pearson correlation coefficient*. We can obtain a formula for *r* by substituting estimates of the covariances

and variances based on a sample into the formula above. So if we have one dataset $\{x_1,\ldots,x_n\}$ containing n values and another dataset $\{y_1,\ldots,y_n\}$ containing n values then that formula for r is:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Where , n , x_i, y_i are defined as above and $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ (the sample mean); and analogously for $\bar{y}$.

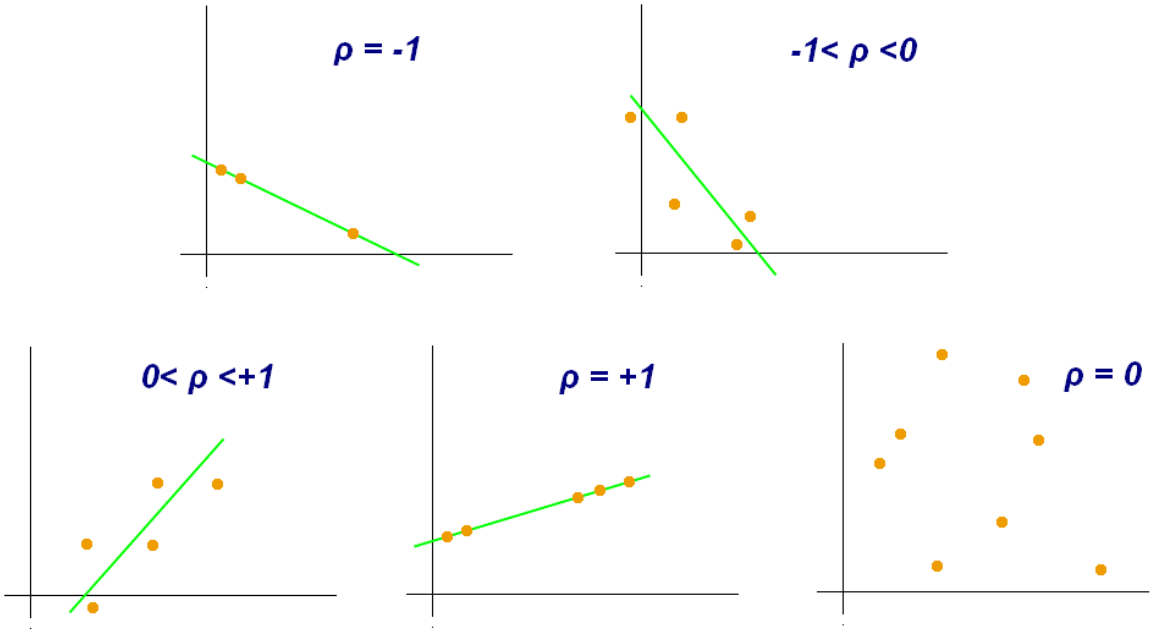


Figure 3.13: Examples of scatter diagrams with different values of correlation coefficient (ρ)

Pearson's correlation coefficient is the covariance of the two variables divided by the product of their standard deviations. The form of the definition involves a "product moment", that is, the mean (the first moment about the origin) of the product of the mean-adjusted random variables; hence the modifier *product-moment* in the name.

3.4.2.3 Information gain Information gain (IG)

Information Gain measures how much information the presence or absence of a term contributes to making the correct classification decision for any class. IG reaches its maximum value if a term is an ideal indicator for class association, that is, if the term is present in a document, if and only if the document belongs to the respective class. IG for term t with respect to class c can be obtained using:

$$\boldsymbol{IG(t,c) = -\sum_{i=1}^{M} P(c)\log P(c) + P(t)\sum_{i=1}^{M} P(c|t)\log P(c|t) + P(\bar{t})\sum_{i=1}^{M} P(c|\bar{t})\log P(c|\bar{t})}$$

where ***M*** is the number of classes, ***P(c)*** is the probability of class **c**, ***P(t)*** and ***P(t)*** are the probability of the presence and absence of term **t**, and ***P(c|t)*** and ***P(c| t)*** are the probability of class ***c*** given the presence and absence of term ***t***, respectively.

3.5 Classification

Databases are rich with hidden information that can be used for intelligent decision making. Classification and prediction are two forms of data analysis that can be used to extract models describing important data classes or to predict future data trends. Such analysis can help provide us with a better understanding of the data at large. Whereas *classification* predicts categorical (discrete, unordered) labels, *prediction* models continuous valued functions. Many classification and prediction methods have been proposed by researchers in machine learning, pattern recognition, and statistics. Classification and prediction have numerous applications, including fraud detection, target marketing, performance prediction, manufacturing, and medical diagnosis. The *generalization* performance of a learning method relates to its prediction capability on independent test data. Assessment of this performance is extremely important in practice, since it guides the choice of learning method or model, and gives us a measure of the quality of the ultimately chosen model.[1][2]

3.5.1 What is Classification? What is Prediction?

The data analysis task is **classification**, where a model or **classifier** is constructed to predict *categorical labels.* These categories can be represented by discrete values, where the ordering among values has no meaning. On the other hand, **prediction** is form of analysis, where the model constructed predicts a *continuous-valued function*, or *ordered value*, as opposed to a categorical label. This model is a **predictor**. Regression analysis is a statistical methodology that is most often used for numeric prediction.

"How does classification work? Data classification is a two-step process. In the first step, a classifier is built describing a predetermined set of data classes or concepts. This is the **learning step** (or training phase), where a classification algorithm builds the classifier by analyzing or "learning from" a **training set** made up of database tuples and their associated class labels. A tuple, $\boldsymbol{X}$, is represented by an n-dimensional **attribute vector**, $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the tuple from n database attributes, respectively, A_1, A_2, …, A_n. Each tuple, $\boldsymbol{X}$, is assumed to belong to a predefined class as determined by another database attribute called the **class label attribute**. The class label attribute is discrete-valued and unordered. It is *categorical* in that each value serves as a category or class. The individual tuples making up the training set are referred to as **training tuples** and are selected from the database under analysis.

Because the class label of each training tuple *is provided*, this step is also known as **supervised learning** (i.e., the learning of the classifier is "supervised" in that it is told to which class each training tuple belongs). It contrasts with **unsupervised learning** (or **clustering**), in which the class label of each training tuple is not known, and the number or set of classes to be learned may not be known in advance.

This first step of the classification process can also be viewed as the learning of a mapping or function, $y = f(\boldsymbol{X})$, that can predict the associated class label y of a given tuple $\boldsymbol{X}$. In this view, we wish to learn a mapping or function that separates the data classes. Typically, this mapping is represented in the form of classification rules, decision trees, or mathematical formulae.[1]

"What about classification accuracy?" In the second step (Figure 6.1(b)), the model is used for classification. First, the predictive accuracy of the classifier is estimated. If we were to use the training set to measure the accuracy of the classifier, this estimate would likely be optimistic, because the classifier tends to **overfit** the data (i.e., during learning it may incorporate some particular anomalies of the training data that are not present in the general data set overall). Therefore, a **test set** is used, made up of **test tuples** and their associated class labels. These tuples are randomly selected from the general data set. They are independent of the training tuples, meaning that they are not used to construct the classifier.

The **accuracy** of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. The associated class label of each test tuple is compared with the learned classifier's class prediction for that tuple. If the accuracy of the classifier is considered acceptable, the classifier can be used to classify future data tuples for which the class label is not known.

"How is (numeric) prediction different from classification?" Data prediction is a two-step process, similar to that of data classification. However, for prediction, we lose the terminology of "class label attribute" because the attribute for which values are being predicted is continuous-valued (ordered) rather than categorical (discrete-valued and unordered). The attribute can be referred to simply as the **predicted attribute**. Prediction can also be viewed as a mapping or function, $y = f(\boldsymbol{X})$, where $\boldsymbol{X}$ is the input (e.g., a tuple describing a loan applicant), and the output y is a continuous or ordered value. That is, we wish to learn a mapping or function that models the relationship between $\boldsymbol{X}$ and y.

Prediction and classification also differ in the methods that are used to build their respective models. As with classification, the training set used to build a predictor should not be used to assess its accuracy. An independent test set should be used instead. The accuracy of a predictor is estimated by computing an error based on the difference between the predicted value and the actual known value of y for each of the test tuples, $\boldsymbol{X}$. There are various predictor error measures.[1]

3.5.2 Preparing the Data for Classification and Prediction

Before we evaluate any model using data, preprocessing steps may be applied to the data to help improve the accuracy, efficiency, and scalability of the classification or prediction process. Preprocessing of the data in preparation for classification and prediction can involve **data cleaning** to reduce noise or handle missing values, **relevance analysis** to remove irrelevant or redundant attributes, and **data transformation**, such as generalizing the data to higher-level concepts or normalizing the data.[1]

3.5.2.1 Overfitting and Data Splitting

In statistics and machine learning, **overfitting** occurs when a statistical model describes random error or noise instead of the underlying relationship. Overfitting generally occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model that has been overfit will generally have poor predictive performance, as it can exaggerate minor fluctuations in the data.

The possibility of overfitting exists because the criterion used for training the model is not the same as the criterion used to judge the efficacy of a model. In particular, a model is typically trained by maximizing its performance on some set of training data. However, its efficacy is determined not by its performance on the training data but by its ability to perform well on unseen data. Overfitting occurs when a model begins to "memorize" training data rather than "learning" to generalize from trend. As an extreme example, if the number of parameters is the same as or greater than the number of observations, a simple model or learning process can perfectly predict the training data simply by memorizing the training data in its entirety, but such a model will typically fail drastically when making predictions about new or unseen data, since the simple model has not learned to generalize at all.[2]

In order to avoid overfitting, it is necessary to use the given dataset in splitted form of two sets: *training set* and *test set*. **Training set** is used to train the model and **test set** is the data exclusively used for evaluating and testing the model. Training set is wider than test set. The learned patterns are applied to this test set, and the resulting output is compared to the desired output. A number of statistical methods may be used to evaluate the algorithm, such as ROC curves. If the learned patterns do not meet the desired standards, subsequently it is

necessary to re-evaluate and change the pre-processing and data mining steps. If the learned patterns do meet the desired standards, then the final step is to interpret the learned patterns and turn them into knowledge.

Now, there are two types of error which helps to determine the overfitting of model. **Test error**, also referred to as *generalization error*, is the prediction error over an independent test sample. **Training error** is the average loss over the training sample. Training error consistently decreases with model complexity, typically dropping to zero if we increase the model complexity enough. However, a model with zero training error is overfit to the training data and will typically generalize poorly. Also as the model becomes more and more complex, it uses the training data more and is able to adapt to more complicated underlying structures. Hence there is a decrease in bias but an increase in variance. There is some intermediate model complexity that gives minimum expected test error.

If we are in a data-rich situation, the best approach for both problems is to randomly divide the dataset into three parts: a training set, a validation set, and a test set. The training set is used to fit the models; the validation set is used to estimate prediction error for model selection; the test set is used for assessment of the generalization error of the final chosen model. Ideally, the test set should be kept in a "vault," and be brought out only at the end of the data analysis. Suppose instead that we use the test-set repeatedly, choosing the model with smallest test-set error. Then the test set error of the final chosen model will underestimate the true test error, sometimes substantially. It is difficult to give a general rule on how to choose the number of observations in each of the three parts, as this depends on the signal-to- noise ratio in the data and the training sample size. A typical split might be 50% for training, and 25% each for validation and testing:[2][3]

Figure 3.14: Training Test and Validation Set

3.5.3 Comparing Classification and Prediction Models

Classification and prediction methods can be compared and evaluated according to the following criteria:

Accuracy: The accuracy of a classifier refers to the ability of a given classifier to correctly predict the class label of new or previously unseen data (i.e., tuples without class label information). Similarly, the accuracy of a predictor refers to how well a given predictor can guess the value of the predicted attribute for new or previously unseen data. Accuracy can be estimated using one or more test sets that are independent of the training set. Because the accuracy computed is only an estimate of how well the classifier or predictor will do on new data tuples, confidence limits can be computed to help gauge this estimate.

Speed: This refers to the computational costs involved in generating and using the given classifier or predictor.

Robustness: This is the ability of the classifier or predictor to make correct predictions given noisy data or data with missing values.

Scalability: This refers to the ability to construct the classifier or predictor efficiently given large amounts of data.

Interpretability: This refers to the level of understanding and insight that is provided by the classifier or predictor. Interpretability is subjective and therefore more difficult to assess.

Recent data mining research has contributed to the development of scalable algorithms for classification and prediction. Additional contributions include the exploration of mined "associations" between attributes and their use for effective classification.[1]

3.5.4 Classification by Decision Tree Induction

Decision tree induction is the learning of decision trees from class-labeled training tuples. A **decision tree** is a flowchart-like tree structure, where each **internal node** (nonleaf node) denotes a test on an attribute, each **branch** represents an outcome of the test, and each **leaf node** (or *terminal node*) holds a class label. The topmost node in a tree is the **root** node. Some decision tree algorithms produce only *binary* trees, whereas others can produce non binary trees.

"How are decision trees used for classification?" Given a tuple, $\boldsymbol{X}$, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple. Decision trees can easily be converted to classification rules.
"Why are decision tree classifiers so popular?" The construction of decision tree classifiers does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. Their representation of acquired knowledge in tree form is intuitive and generally easy to assimilate by humans. The learning and classification steps of decision tree induction are simple and fast. In general, decision tree classifiers have good accuracy. However, successful use may depend on the data at hand. Decision tree induction algorithms have been used for classification in many application areas, such as medicine, manufacturing and production, financial analysis, astronomy, and molecular biology. Decision trees are the basis of several commercial rule induction systems.

During tree construction, *attribute selection measures* are used to select the attribute that best partitions the tuples into distinct classes. When decision trees are built, many of the branches may reflect noise or outliers in the training data. *Tree pruning* attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data.[1]

An **attribute selection measure** is a heuristic for selecting the splitting criterion that "best" separates a given data partition, *D*, of class-labeled training tuples into individual classes. If we were to split *D* into smaller partitions according to the outcomes of the splitting criterion, ideally each partition would be pure (i.e., all of the tuples that fall into a given partition would belong to the same class). Conceptually, the "best" splitting criterion is the one that most closely results in such a scenario. Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split. The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure is chosen as the *splitting attribute* for the given tuples. If the splitting attribute is continuous-valued or if we are restricted to binary trees then, respectively, either a *split point* or a *splitting subset* must also be determined as part of the splitting criterion. The tree node created for partition *D* is labeled with the splitting criterion, branches are grown for each outcome of the criterion, and the tuples are partitioned accordingly. Popular attribute selection measures are — *information gain, gain ratio*, and *gini index*.[1]

3.5.4.1 Implementation of C4.5 Decision Tree Algorithm – J48

J48, an open source Java implementation of the C4.5 decision tree algorithm. C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan. The decision tree generated by C4.5 can be used for classification, and for this reason, C4.5 is also referred to as a statistical classifier. J48 examines the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. To make the decision, the attribute with the highest normalized information gain is used. Then the algorithm recurs on the smaller subsets.[4][5]

Algorithm: Generate decision tree. Generate a decision tree from the training tuples of data partition *D*.[1]

Method:

1. create a node *N*;
2. **if** tuples in *D* are all of the same class, *C* **then**
3. return *N* as a leaf node labeled with the class *C*;
4. **if** *attribute list* is empty **then**
5. return *N* as a leaf node labeled with the majority class in *D*; // majority voting
6. apply Attribute selection method(*D*, *attribute list*) to find the "best" *splitting criterion*;
7. label node *N* with *splitting criterion*;
8. **if** *splitting attribute* is discrete-valued **and**
   multiway splits allowed then // not restricted to binary trees
9. *attribute list* ← *attribute list – splitting attribute*; // remove *splitting attribute*
10. **for** each outcome *j* of *splitting criterion*
    // partition the tuples and grow subtrees for each partition
11. let D_j be the set of data tuples in *D* satisfying outcome *j*; // a partition
12. **if** D_j is empty **then**
13. attach a leaf labeled with the majority class in *D* to node *N*;
14. **else** attach the node returned by Generate decision tree(D_j, *attribute list*) to node *N*;
15. **endfor**
16. return *N*;

Input:

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute list*, the set of candidate attributes;
- *Attribute selection method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting attribute* and, possibly, either a *split point* or *splitting subset*.

Output: A decision tree.

3.5.5 Bayesian Classification

"What are Bayesian classifiers?" Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, such as the probability that a given tuple belongs to a particular class.

Bayesian classification is based on Bayes' theorem, described below. Studies comparing classification algorithms have found a simple Bayesian classifier known as the *naïve Bayesian classifier* to be comparable in performance with decision tree and selected neural network classifiers. Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases.

Naïve Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called *class conditional independence*. It is made to simplify the computations involved and, in this sense, is considered "naïve." *Bayesian belief networks* are graphical models, which unlike naïve Bayesian classifiers, allow the representation of dependencies among subsets of attributes. Bayesian belief networks can also be used for classification.[1]
"How are these probabilities estimated?" $P(H)$, $P(\boldsymbol{X}|H)$, and $P(\boldsymbol{X})$ may be estimated from the given data, as we shall see below. **Bayes' theorem** is useful in that it provides a way of calculating the posterior probability, $P(H|\boldsymbol{X})$, from $P(H)$, $P(\boldsymbol{X}|H)$, and $P(\boldsymbol{X})$. Bayes' theorem is

3.5.5.1 Bayes' Theorem

Let $\boldsymbol{X}$ be a data tuple. In Bayesian terms, $\boldsymbol{X}$ is considered "evidence." As usual, it is described by measurements made on a set of n attributes. Let H be some hypothesis, such as that the data tuple $\boldsymbol{X}$ belongs to a specified class C. For classification problems, we want to determine $P(H|\boldsymbol{X})$, the probability that the hypothesis H holds given the "evidence" or observed data tuple $\boldsymbol{X}$. In other words, we are looking for the probability that tuple $\boldsymbol{X}$ belongs to class C, given that we know the attribute description of $\boldsymbol{X}$. $P(H|\boldsymbol{X})$ is the **posterior probability**, or *a posteriori probability*, of H conditioned on $\boldsymbol{X}$. In contrast, $P(H)$ is the **prior probability**, or *a priori probability,* of H. The posterior probability, $P(H|\boldsymbol{X})$, is based on more information (e.g., customer information) than the prior probability, $P(H)$, which is independent of $\boldsymbol{X}$. Similarly, $P(\boldsymbol{X}|H)$ is the posterior probability of $\boldsymbol{X}$ conditioned on H.

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}$$

"How effective are Bayesian classifiers?" Various empirical studies of this classifier in comparison to decision tree and neural network classifiers have found it to be comparable in some domains. In theory, Bayesian classifiers have the minimum error rate in comparison to all other classifiers. However, in practice this is not always the case, owing to inaccuracies in the assumptions made for its use, such as class conditional independence, and the lack of available probability data. Bayesian classifiers are also useful in that they provide a theoretical justification for other classifiers that do not explicitly use Bayes' theorem. For example, under certain assumptions, it can be shown that many neural network and curve-fitting algorithms output the *maximum posteriori* hypothesis, as does the naïve Bayesian classifier.[1]

3.5.5.2 Naïve Bayes Classifier

Derivation:[7]

D: Set of tuples

- Each Tuple is an 'n' dimensional attribute vector
- $X : (x_1,x_2,x_3,\ldots, x_n)$

Let there be 'm' classes: $C_1, C_2, C_3, \ldots, C_m$.

Naïve Bayes classifier predicts X belongs to Class C_i iff

- $P(C_i/X) > P(C_j/X)$ *for* $1<= j <= m , j <> i$

Maximum Posteriori Hypothesis

- $P(Ci/X) = P(X/C_i)\, P(C_i) / P(X)$
- Maximize $P(X/C_i)\, P(C_i)$ *as* $P(X)$ is constant

With many attributes, it is computationally expensive to evaluate $P(X/C_i)$.

Naïve Assumption of "class conditional independence"

$$P\left(\frac{X}{C_i}\right) = \prod_{k=1}^{n} P(x_k/C_i)$$

$$P\left(\frac{X}{C_i}\right) = P\left(\frac{x_1}{C_i}\right) * P\left(\frac{x_2}{C_i}\right) * \ldots * P\left(\frac{x_n}{C_i}\right)$$

3.5.6 Support Vector Machines

"What are Support Vector Machines?" In machine learning, **support vector machines** (**SVMs**, also **support vector networks**[1]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

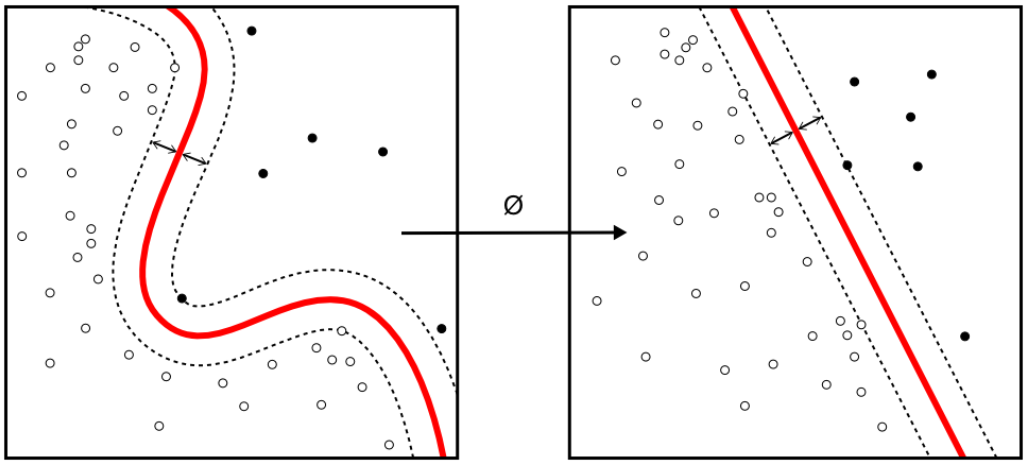


Figure 3.15: Decision Boundary

3.5.6.1 Linear Support Vector Machines

We are given a training dataset of n points of the form

$$(\overrightarrow{x_1}, y_1), \ldots\ldots, (\overrightarrow{x_n}, y_n)$$

where the y_i are either 1 or −1, each indicating the class to which the point $\overrightarrow{x_i}$ belongs. Each $\overrightarrow{x_i}$ is a p-dimensional real vector. We want to find the "maximum-margin hyperplane" that divides the group of points $\overrightarrow{x_i}$ for which from the group of points for which $y_i = 1$, which is defined so that the distance between the hyperplane and the nearest point $\overrightarrow{x_i}$ from either group is maximized.

Any hyperplane can be written as the set of points $\overrightarrow{x_i}$ satisfying

$$\vec{w}.\vec{x} - b = 0$$

where $\vec{w}$ is the (not necessarily normalized) normal vector to the hyperplane. The parameter $\frac{b}{||\vec{w}||}$ determines the offset of the hyperplane from the origin along the normal vector $\vec{w}$.

If the training data are linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the "margin", and the maximum-margin hyperplane is the hyperplane that lies halfway between them. These hyperplanes can be described by the equations

$$\vec{w}.\vec{x} - b = 1$$

And

$$\vec{w}.\vec{x} - b = -1$$

Geometrically, the distance between these two hyperplanes is $\frac{2}{||\vec{w}||}$, so to maximize the distance between the planes we want to minimize $||\vec{w}||$. As we also have to prevent data points from falling into the margin, we add the following constraint: for each i either

$(\vec{w}.\overrightarrow{x_i} - b) \geq 1$ if $y_i = 1$ or

$(\vec{w}.\overrightarrow{x_i} - b) \leq 1$ if $y_i = -1$

These constraints state that each data point must lie on the correct side of the margin.

This can be rewritten as:

$$y_i.(\vec{w}.\overrightarrow{x_i} - b) \geq 1 \text{ for all } 1 \leq i \leq n$$

We can put this together to get the optimization problem:

"Minimize $||\vec{w}||$ subject to $y_i.(\vec{w}.\vec{x_i} - b) \geq 1$ for $i = 1, 2, 3 \ldots . . n$"

The $\vec{w}$ and b that solve this problem determine our classifier, $\vec{x} \mapsto sgn(\vec{w}.\vec{x} - b)$.

An easy-to-see but important consequence of this geometric description is that max-margin hyperplane is completely determined by those $\vec{x_i}$ which lie nearest to it. These $\vec{x_i}$ are called *support vectors*.

3.5.7 Classifier Accuracy Measures

Now that you may have trained a classifier or predictor, there may be many questions going through your mind. How accurately the classifier can predict? What is accuracy? How can we estimate it? Are there strategies for increasing the accuracy of a learned model?

Using training data to derive a classifier or predictor and then to estimate the accuracy of the resulting learned model can result in misleading overoptimistic estimates due to overspecialization of the learning algorithm to the data. Instead, accuracy is better measured on a test set consisting of class-labeled tuples that were not used to train the model. The **accuracy** of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. In the pattern recognition literature, this is also referred to as the overall **recognition rate** of the classifier, that is, it reflects how well the classifier recognizes tuples of the various classes.

We can also speak of the **error rate** or **misclassification rate** of a classifier, M, which is simply $1-Acc(M)$, where $Acc(M)$ is the accuracy of M. If we were to use the training set to estimate the error rate of a model, this quantity is known as the **resubstitution** error. This error estimate is optimistic of the true error rate (and similarly, the corresponding accuracy estimate is optimistic) because the model is not tested on any samples that it has not already seen.[1]

The *confusion matrix* is a useful tool for analyzing how well your classifier can recognize tuples of different classes. Given m classes, a **confusion matrix** is a table of at least size m by m. An entry, $CM_{i,j}$ in the first m rows and m columns indicates the number of tuples of class i that were labeled by the classifier as class j. For a classifier to have good accuracy, ideally most of the tuples would be represented along the diagonal of the confusion matrix, from entry $CM_{1,1}$ to entry $CM_{m,m}$, with the rest of the entries being close to zero. The table may have additional rows or columns to provide totals or recognition rates per class.

Given two classes, we can talk in terms of positive tuples versus negative tuples. **True positives** refer to the positive tuples that were correctly labeled by the classifier, while **true negatives** are the negative tuples that were correctly labeled by the classifier. **False positives** are the negative tuples that were incorrectly labeled. Similarly, **false negatives** are the positive tuples that were incorrectly labeled.

		Predicted class	
		C_1	C_2
Actual class	C_1	true positives	false negatives
	C_2	false positives	true negatives

Table 3.2: A confusion matrix for positive and negative tuples.

"Are there alternatives to the accuracy measure?" The sensitivity and specificity measures can be used. **Sensitivity** is also referred to as the *true positive (recognition) rate* (that is, the proportion of positive tuples that are correctly identified), while **specificity** is the *true negative rate* (that is, the proportion of negative tuples that are correctly identified). In addition, we may use **precision** to access the percentage of tuples. These measures are defined as

$$sensitivity = \frac{t\_pos}{pos}$$

$$specificity = \frac{t\_neg}{neg}$$

$$precision = \frac{t\_pos}{t\_pos + f\_pos}$$

where t_pos is the number of true positives, pos is the number of positive tuples, t_neg is the number of true negatives, neg is the number of negative tuples, and f_pos is the number of false positives. It can be shown that accuracy is a function of sensitivity and specificity:

$$accuracy = sensitivity\frac{pos}{pos + neg} + specificity\frac{neg}{pos + neg}$$

The true positives, true negatives, false positives, and false negatives are also useful in assessing the **costs** and **benefits** (or risks and gains) associated with a classification model. The cost associated with a false negative is far greater than that of a false positive. In such cases, we can outweigh one type of error over another by assigning a different cost to each. Similarly, the benefits associated with a true positive decision may be different than that of a true negative.[1]

3.5.8 Model Selection

Suppose that we have generated two models, M_1 and M_2 (for either classification or prediction), from our data. How can we determine which model is best? It may seem intuitive to select the model with the lowest error rate, however, the mean error rates are just *estimates* of error on the true population of future data cases. Although the mean error rates obtained for M_1 and M_2 may appear different, that difference may not be statistically significant. What if any difference between the two may just be attributed to chance?[1]

3.5.8.1 ROC Curves

ROC curves are a useful visual tool for comparing two classification models. The name ROC stands for *Receiver Operating Characteristic*. An ROC curve shows the trade-off between the true positive rate or sensitivity (proportion of positive tuples that are correctly identified) and the false-positive rate (proportion of negative tuples that are incorrectly identified as positive) for a given model. That is, given a two-class problem, it allows us to visualize the trade-off between the rate at which the model can accurately recognize 'yes' cases versus the rate at which it mistakenly identifies 'no' cases as 'yes' for different "portions" of the test set. Any increase in the true positive rate occurs at the cost of an increase in the false-positive rate. The area under the ROC curve is a measure of the accuracy of the model.

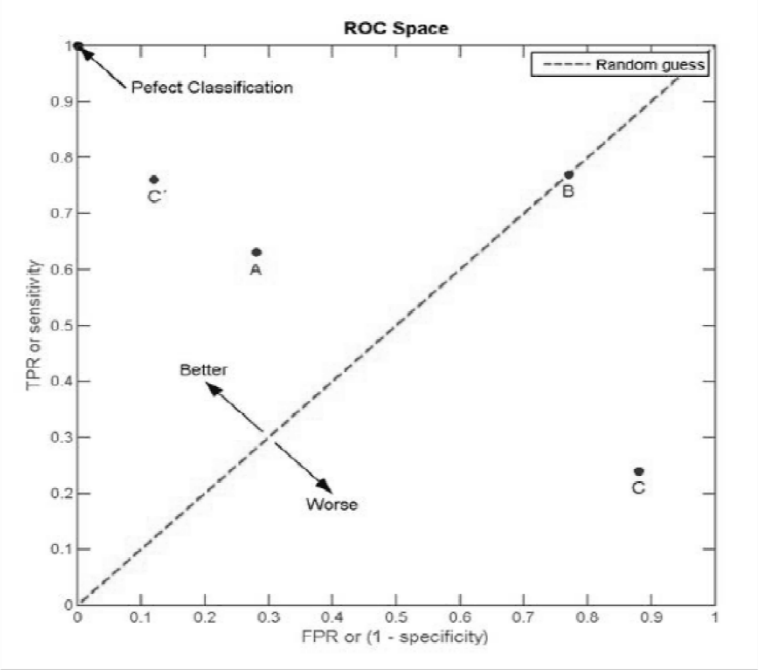


Figure 3.16: ROC Space

In order to plot an ROC curve for a given classification model, *M*, the model must be able to return a probability or ranking for the predicted class of each test tuple. That is, we need to rank the test tuples in decreasing order, where the one the classifier thinks is most likely to belong to the positive or 'yes' class appears at the top of the list. Naive Bayesian and backpropagation classifiers are appropriate, whereas others, such as decision tree classifiers, can easily be modified so as to return a class probability distribution for each prediction. The vertical axis of an ROC curve represents the true positive rate. The horizontal axis represents the false-positive rate. An ROC curve for *M* is plotted as follows. Starting at the bottom left-hand corner (where the true positive rate and false-positive rate are both 0), we check the actual class label of the tuple at the top of the list. If we have a true positive (that is, a positive tuple that was correctly classified), then on the ROC curve, we move up and plot a point. If, instead, the tuple really belongs to the 'no' class, we have a false positive. On the ROC curve, we move right and plot a point. This process is repeated for each of the test tuples, each time moving up on the curve for a true positive or toward the right for a false positive.

Figure 4.4 shows the ROC curves of two classification models. The plot also shows a diagonal line where for every true positive of such a model, we are just as likely to encounter a false positive. Thus, the closer the ROC curve of a model is to the diagonal line, the less accurate the model. If the model is really good, initially we are more likely to encounter true positives as we move down the ranked list. Thus, the curve would move steeply up from zero. Later, as we start to encounter fewer and fewer true positives, and more and more false positives, the curve cases off and becomes more horizontal. To assess the accuracy of a model, we can measure the area under the curve. Several software packages are able to perform such calculation. The closer the area is to 0.5, the less accurate the corresponding model is. A model with perfect accuracy will have an area of 1.0.[1]

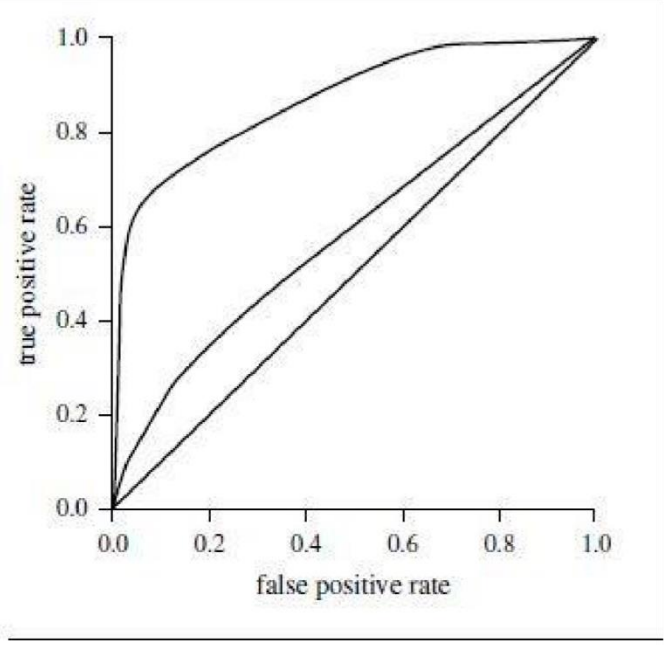


Figure 3.17: The ROC curves of two classification models.

Chapter 4

Implementation

This section describes the implementation approach followed. The 20 Newsgroup dataset is pre-processed now different feature selection techniques are applied and the features thus obtained are fed to different classifiers. All the algorithms were implemented in Python environment.

4.1. Pre-processing

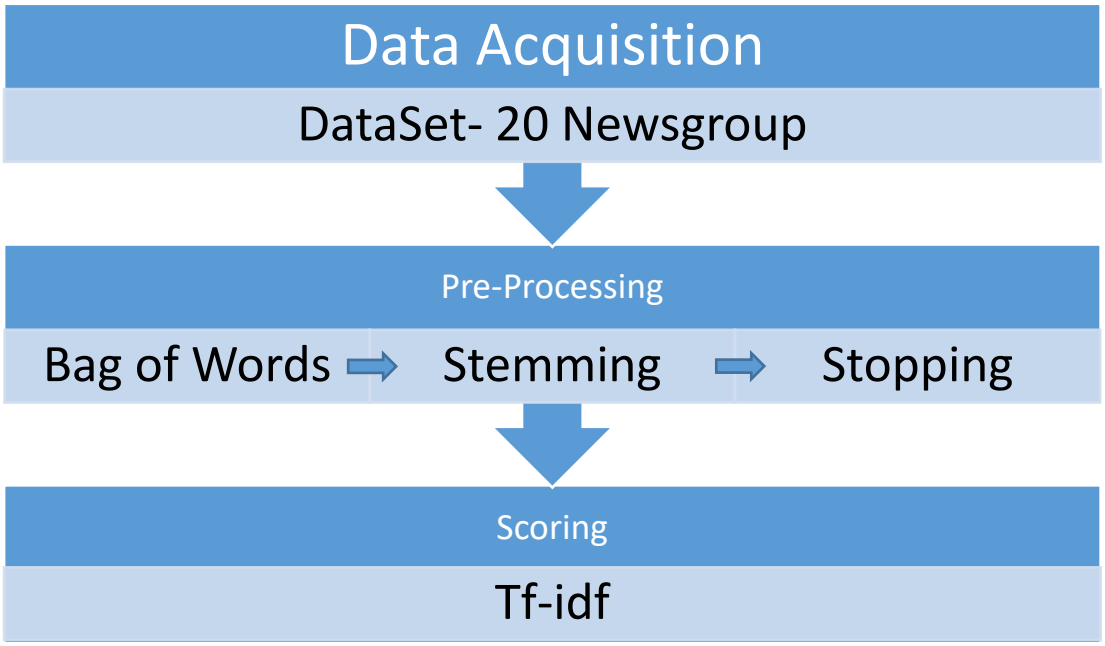


Figure 4.1: Flow of Techniques used for prep-processing

DataSet Desciption :

- Number of articles = 20000
- Number of classes = 20

Data Transformation :

- Tokenizer : Tf-Idf vectorizer
- Stopwords : Rainbow
- Stemming : Porter Stemmer

4.2. Feature Selection

Prominent features are selected using feature selection techniques and the number of features that are passed onto to next stage are 1000.

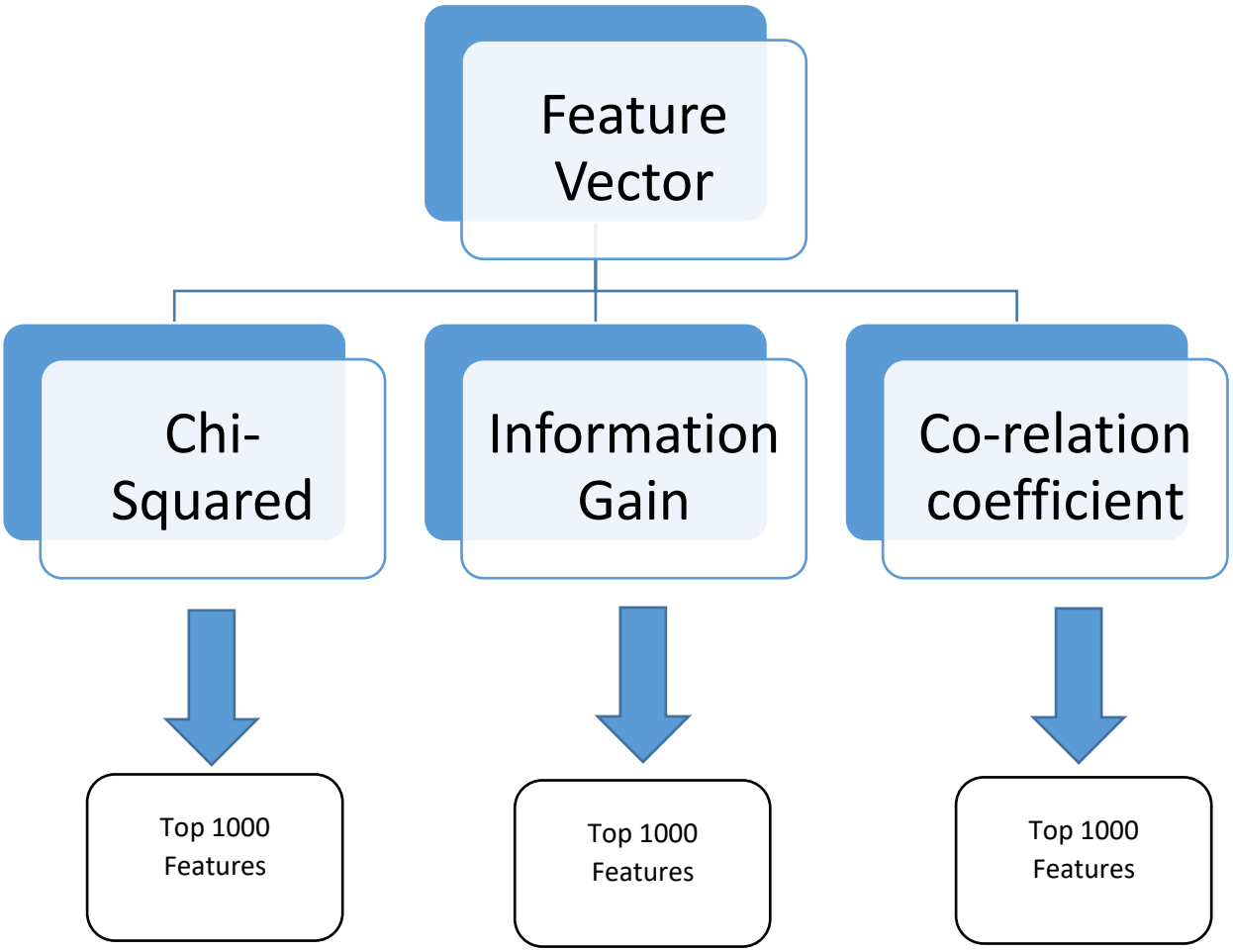


Figure 4.2: Dimensionality reduction using feature selection techniques

4.3 Classification

The training set with reduced feature space is passed into Machine Learning models and 5-fold cross validation is also performed.

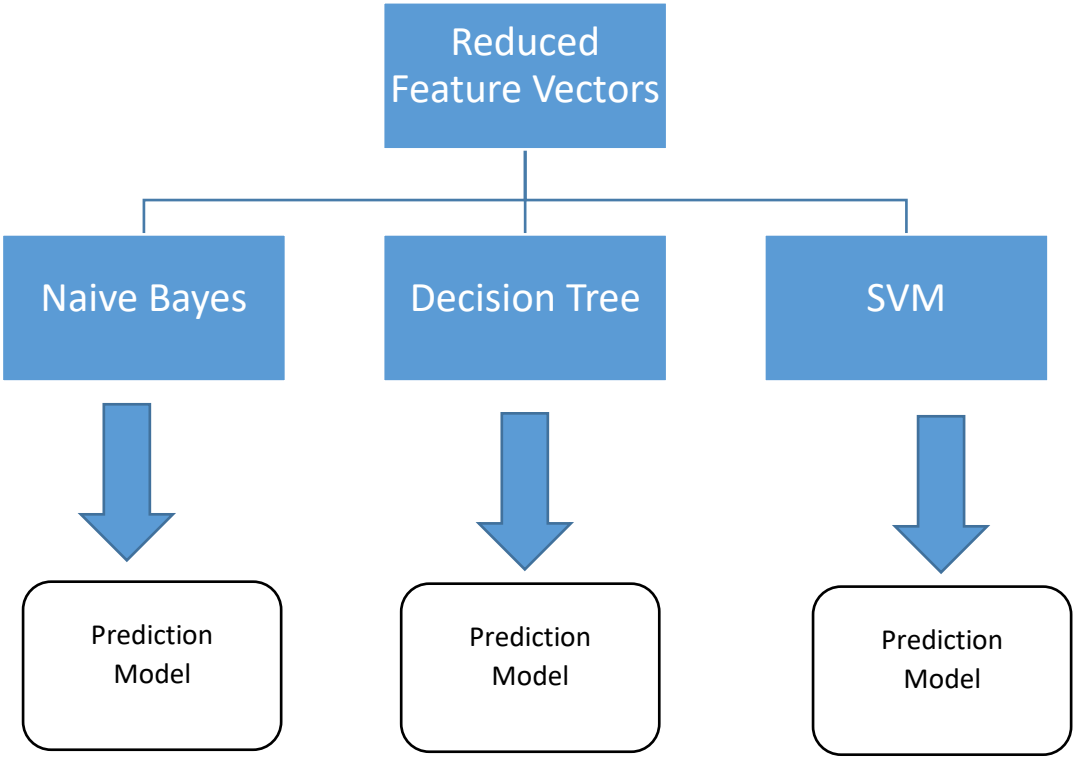


Figure 4.3: Generating prediction models using classifiers

SVM parameters :

- Kernel = Radial Basis Function
- Gamma = 0.0
- Eps = 0.001
- Degree = 3

Decision Tree Parameters:

- Minimum Number of Objects = 2
- confidence factor = 0.25

4.4 Results and Evaluation

The trained model is evaluated against the test samples and finally the values of the evaluation metrics Accuracy, Precision, and ROC curves are generated.

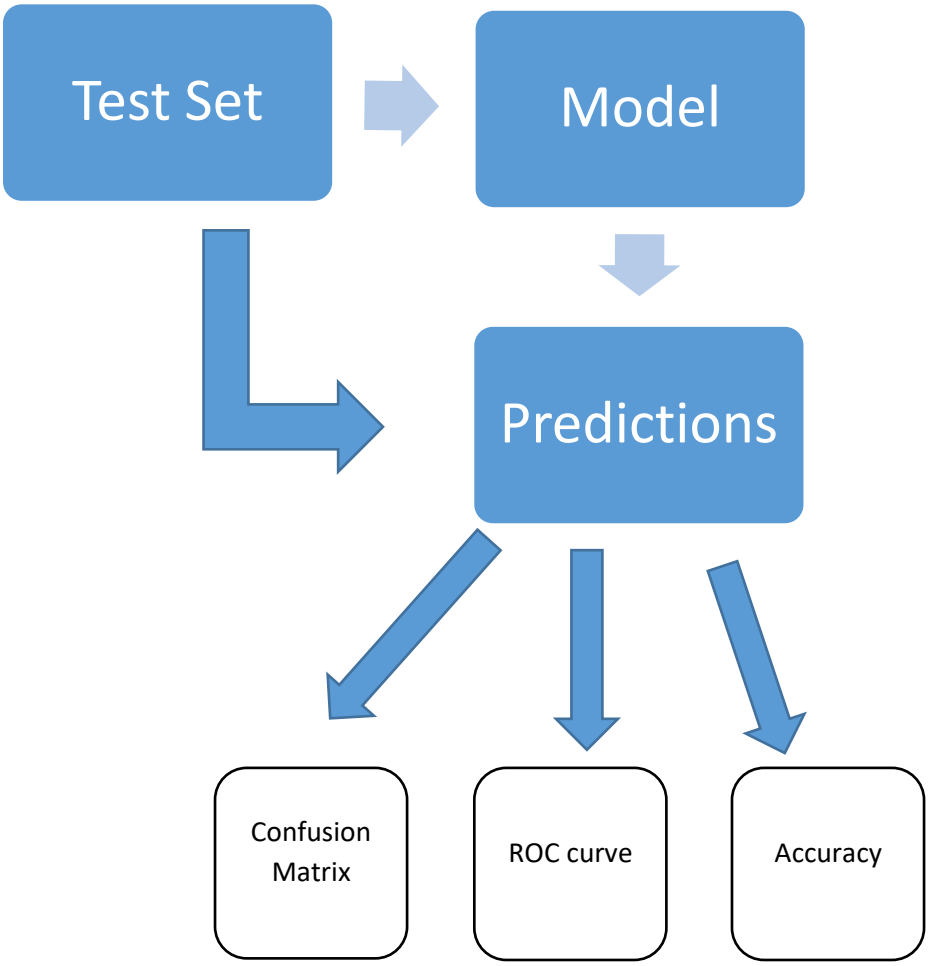


Figure 4.4: Evaluating the trained Models

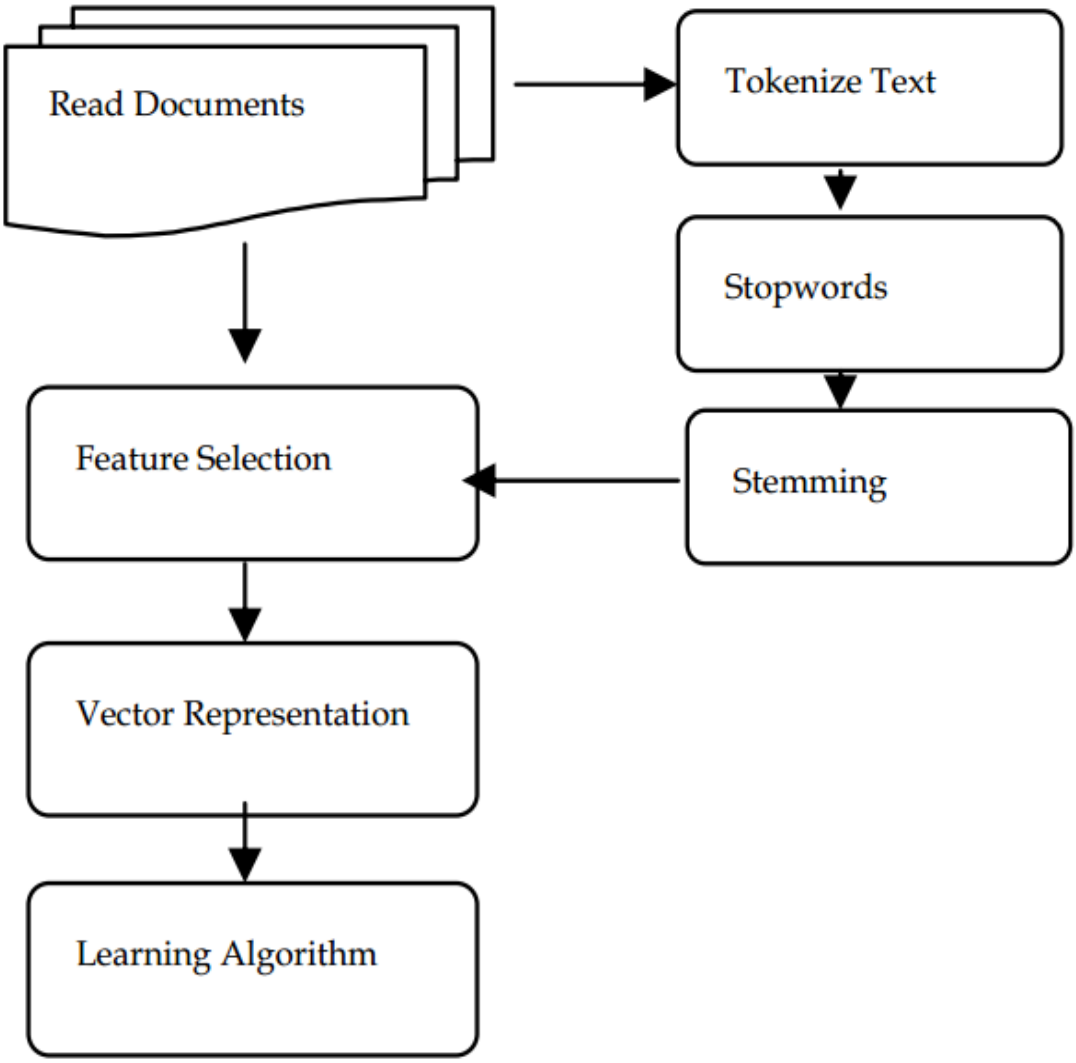


Figure 4.5 Pipeline for evaluating feature selection techniques and classifiers

Result and Analysis

5.1 Results by Naïve Bayes Classifier

5.1.1 Chi Squared

Correctly Classified Instances 9185 81.1826%

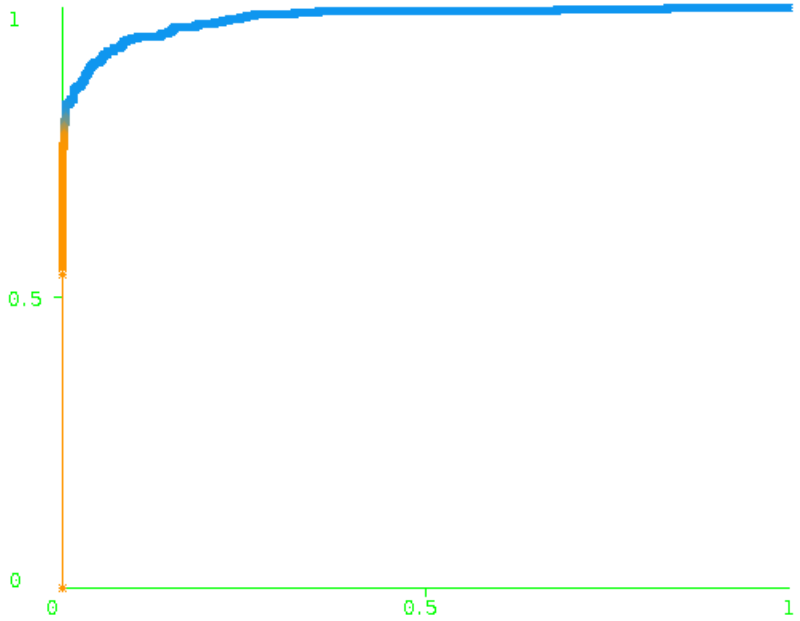
Incorrectly Classified Instances 2129 18.8174%

```
Kappa statistic                          0.8018
Mean absolute error                      0.0194
Root mean squared error                  0.1256
Relative absolute error                 20.473  %
Root relative squared error             57.6335 %
Total Number of Instances            11314
```

```
=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.800	0.004	0.893	0.800	0.844	0.839	0.980	0.874	alt.atheism
	0.936	0.004	0.927	0.936	0.931	0.928	0.992	0.962	sci.crypt
	0.749	0.008	0.831	0.749	0.788	0.778	0.979	0.843	comp.sys.mac.hardware
	0.692	0.016	0.706	0.692	0.699	0.683	0.968	0.762	comp.graphics
	0.894	0.007	0.880	0.894	0.887	0.881	0.991	0.936	sci.space
	0.808	0.005	0.894	0.808	0.849	0.842	0.986	0.916	sci.med
	0.907	0.010	0.826	0.907	0.865	0.858	0.989	0.927	talk.politics.guns
	0.618	0.017	0.666	0.618	0.641	0.622	0.960	0.696	sci.electronics
	0.658	0.008	0.743	0.658	0.698	0.689	0.960	0.744	talk.religion.misc
	0.903	0.004	0.921	0.903	0.912	0.907	0.997	0.966	rec.sport.baseball
	0.891	0.006	0.899	0.891	0.895	0.889	0.994	0.945	rec.motorcycles
	0.639	0.021	0.629	0.639	0.634	0.614	0.963	0.675	comp.sys.ibm.pc.hardware
	0.958	0.004	0.929	0.958	0.943	0.940	0.998	0.986	rec.sport.hockey
	0.815	0.011	0.761	0.815	0.787	0.778	0.979	0.865	talk.politics.misc
	0.781	0.013	0.763	0.781	0.772	0.759	0.982	0.861	rec.autos
	0.726	0.016	0.712	0.726	0.719	0.704	0.969	0.777	misc.forsale
	0.913	0.010	0.835	0.913	0.872	0.866	0.993	0.931	soc.religion.christian
	0.922	0.003	0.940	0.922	0.931	0.928	0.994	0.958	talk.politics.mideast
	0.758	0.022	0.651	0.758	0.701	0.685	0.974	0.779	comp.os.ms-windows.misc
	0.813	0.009	0.838	0.813	0.825	0.816	0.985	0.890	comp.windows.x
Weighted Avg.	0.812	0.010							

ROC Curve



5.1.2 Information Gain

Correctly Classified 9194 81.2622%

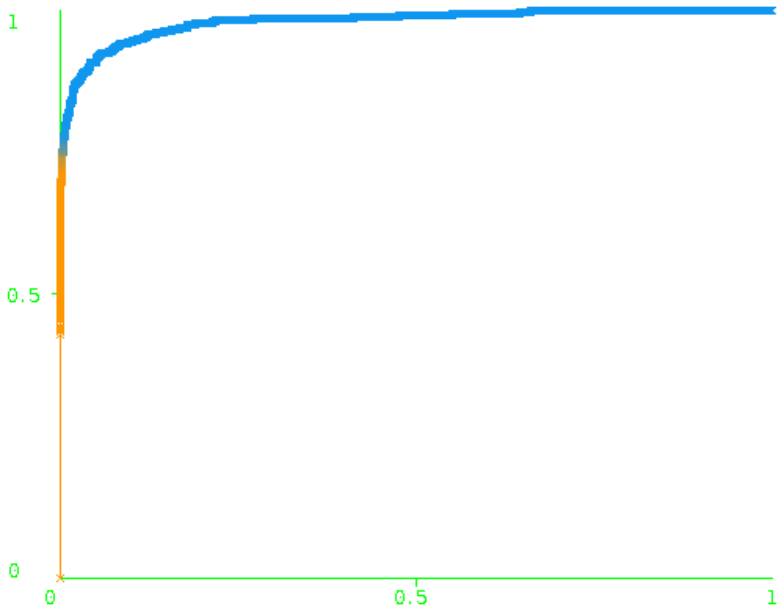
Incorrectly Classified 2120 18.7378%

```
Kappa statistic                          0.8026
Mean absolute error                      0.0192
Root mean squared error                  0.1271
Relative absolute error                 20.2281 %
Root relative squared error             58.3314 %
Total Number of Instances           11314
```

```
=== Detailed Accuracy By Class ===
```

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.744	0.004	0.884	0.744	0.808	0.803	0.979	0.861	alt.atheism
0.928	0.004	0.929	0.928	0.929	0.925	0.992	0.962	sci.crypt
0.760	0.011	0.790	0.760	0.774	0.763	0.981	0.845	comp.sys.mac.hardware
0.728	0.022	0.646	0.728	0.684	0.667	0.971	0.748	comp.graphics
0.880	0.008	0.860	0.880	0.870	0.863	0.991	0.938	sci.space
0.810	0.004	0.918	0.810	0.860	0.855	0.987	0.924	sci.med
0.897	0.010	0.815	0.897	0.854	0.848	0.989	0.908	talk.politics.guns
0.665	0.017	0.688	0.665	0.676	0.659	0.965	0.752	sci.electronics
0.613	0.009	0.698	0.613	0.653	0.643	0.958	0.653	talk.religion.misc
0.893	0.003	0.940	0.893	0.916	0.912	0.997	0.964	rec.sport.baseball
0.891	0.008	0.868	0.891	0.880	0.873	0.994	0.946	rec.motorcycles
0.659	0.019	0.657	0.659	0.658	0.639	0.965	0.693	comp.sys.ibm.pc.hardware
0.953	0.004	0.932	0.953	0.942	0.939	0.998	0.984	rec.sport.hockey
0.813	0.011	0.762	0.813	0.787	0.778	0.977	0.854	talk.politics.misc
0.798	0.012	0.790	0.798	0.794	0.783	0.985	0.876	rec.autos
0.766	0.012	0.783	0.766	0.774	0.762	0.977	0.819	misc.forsale
0.903	0.011	0.822	0.903	0.861	0.854	0.993	0.927	soc.religion.christian
0.887	0.002	0.952	0.887	0.918	0.915	0.995	0.958	talk.politics.mideast
0.761	0.017	0.709	0.761	0.734	0.719	0.976	0.780	comp.os.ms-windows.misc
0.820	0.010	0.817	0.820	0.818	0.808	0.986	0.894	comp.windows.x
Weighted Avg. 0.813	0.010	0.815	0.813	0.813	0.804	0.983	0.868	

ROC Curve



5.1.3 Correlation Coefficient

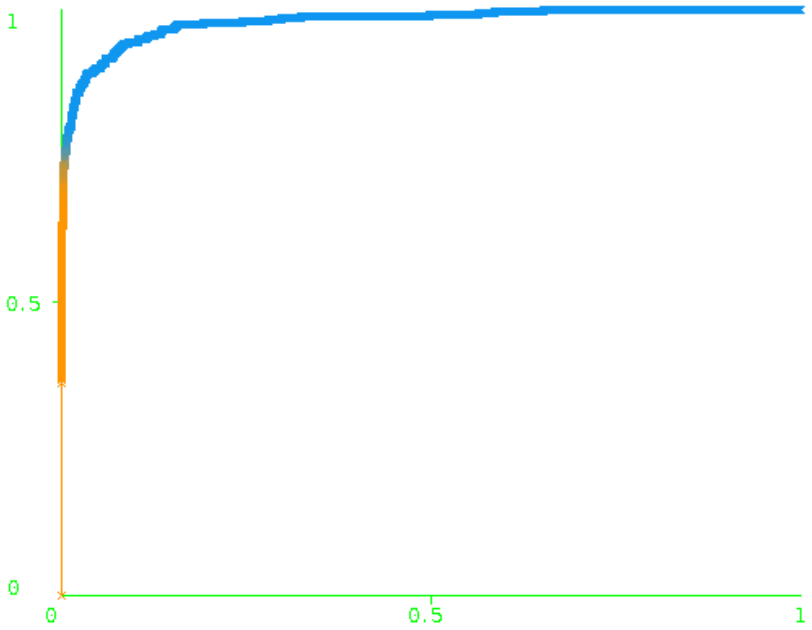
Correctly Classified	9163	80.9882%
Incorrectly Classified	2151	19.0118%

```
Kappa statistic                          0.7997
Mean absolute error                      0.0195
Root mean squared error                  0.1278
Relative absolute error                 20.4952 %
Root relative squared error             58.645  %
Total Number of Instances           11314

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.742    0.006    0.852      0.742    0.793      0.786    0.978     0.845     alt.atheism
                 0.931    0.004    0.930      0.931    0.930      0.926    0.991     0.958     sci.crypt
                 0.751    0.013    0.760      0.751    0.755      0.742    0.980     0.834     comp.sys.mac.hardware
                 0.743    0.021    0.659      0.743    0.698      0.682    0.972     0.752     comp.graphics
                 0.889    0.008    0.861      0.889    0.875      0.868    0.990     0.935     sci.space
                 0.806    0.004    0.921      0.806    0.860      0.855    0.986     0.917     sci.med
                 0.890    0.010    0.821      0.890    0.854      0.847    0.989     0.903     talk.politics.guns
                 0.670    0.017    0.689      0.670    0.679      0.662    0.965     0.758     sci.electronics
                 0.589    0.009    0.683      0.589    0.632      0.623    0.959     0.649     talk.religion.misc
                 0.896    0.003    0.939      0.896    0.917      0.913    0.997     0.968     rec.sport.baseball
                 0.900    0.008    0.857      0.900    0.878      0.871    0.994     0.948     rec.motorcycles
                 0.649    0.020    0.646      0.649    0.648      0.628    0.964     0.684     comp.sys.ibm.pc.hardware
                 0.950    0.004    0.931      0.950    0.941      0.937    0.998     0.984     rec.sport.hockey
                 0.819    0.011    0.768      0.819    0.793      0.784    0.978     0.852     talk.politics.misc
                 0.791    0.011    0.799      0.791    0.795      0.784    0.985     0.869     rec.autos
                 0.750    0.012    0.770      0.750    0.760      0.747    0.976     0.813     misc.forsale
                 0.903    0.011    0.818      0.903    0.859      0.852    0.992     0.926     soc.religion.christian
                 0.881    0.002    0.952      0.881    0.915      0.912    0.995     0.958     talk.politics.mideast
                 0.733    0.017    0.710      0.733    0.721      0.706    0.975     0.773     comp.os.ms-windows.misc
                 0.823    0.010    0.819      0.823    0.821      0.811    0.986     0.896     comp.windows.x
Weighted Avg.    0.810    0.010    0.812      0.810    0.810      0.800    0.983     0.866
```

ROC Curve



5.2 Results by Decision Tree Classifier

5.2.1 Chi Squared

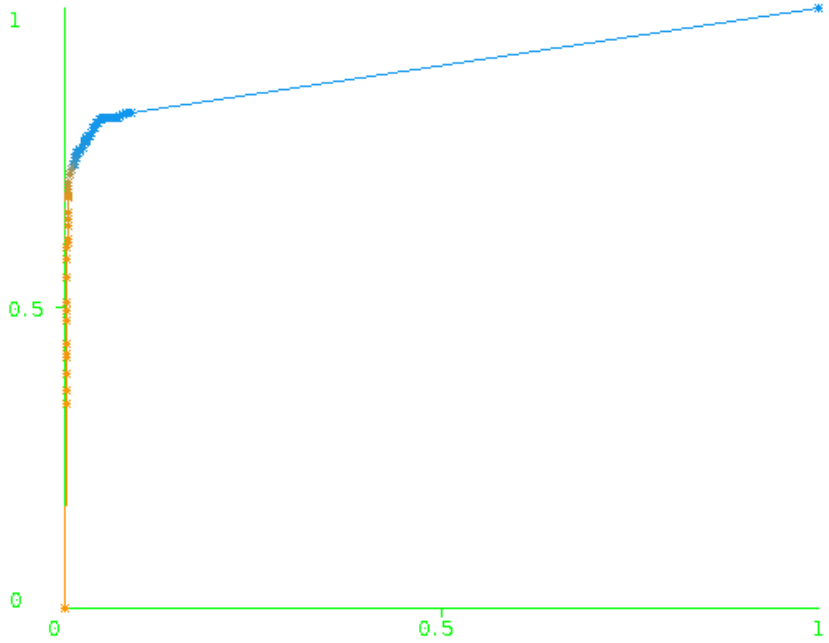
Correctly Classified	8025	70.9298%
Incorrectly Classified	3289	29.0702%

```
Kappa statistic                          0.6938
Mean absolute error                      0.0334
Root mean squared error                  0.1552
Relative absolute error                 35.1706 %
Root relative squared error             71.227  %
Total Number of Instances           11314

=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.733	0.012	0.726	0.733	0.730	0.717	0.900	0.692	alt.atheism
	0.877	0.009	0.842	0.877	0.859	0.851	0.947	0.843	sci.crypt
	0.720	0.019	0.668	0.720	0.693	0.676	0.877	0.646	comp.sys.mac.hardware
	0.567	0.022	0.585	0.567	0.576	0.553	0.814	0.475	comp.graphics
	0.786	0.015	0.744	0.786	0.765	0.751	0.902	0.723	sci.space
	0.734	0.019	0.679	0.734	0.706	0.689	0.886	0.678	sci.med
	0.782	0.012	0.775	0.782	0.778	0.767	0.916	0.712	talk.politics.guns
	0.491	0.028	0.489	0.491	0.490	0.462	0.789	0.380	sci.electronics
	0.501	0.013	0.580	0.501	0.538	0.524	0.840	0.428	talk.religion.misc
	0.744	0.014	0.746	0.744	0.745	0.731	0.889	0.674	rec.sport.baseball
	0.819	0.009	0.842	0.819	0.831	0.821	0.931	0.746	rec.motorcycles
	0.464	0.023	0.526	0.464	0.493	0.468	0.780	0.355	comp.sys.ibm.pc.hardware
	0.848	0.009	0.847	0.848	0.848	0.839	0.944	0.797	rec.sport.hockey
	0.634	0.015	0.641	0.634	0.638	0.622	0.842	0.552	talk.politics.misc
	0.709	0.017	0.698	0.709	0.703	0.687	0.874	0.587	rec.autos
	0.668	0.020	0.646	0.668	0.657	0.638	0.842	0.522	misc.forsale
	0.815	0.013	0.775	0.815	0.794	0.783	0.924	0.727	soc.religion.christian
	0.817	0.006	0.880	0.817	0.847	0.840	0.926	0.803	talk.politics.mideast
	0.692	0.019	0.664	0.692	0.678	0.660	0.859	0.541	comp.os.ms-windows.misc
	0.698	0.012	0.760	0.698	0.728	0.714	0.894	0.656	comp.windows.x
Weighted Avg.	0.709	0.015	0.708	0.709	0.708	0.693	0.880	0.631	

ROC Curve



5.2.2 Information Gain

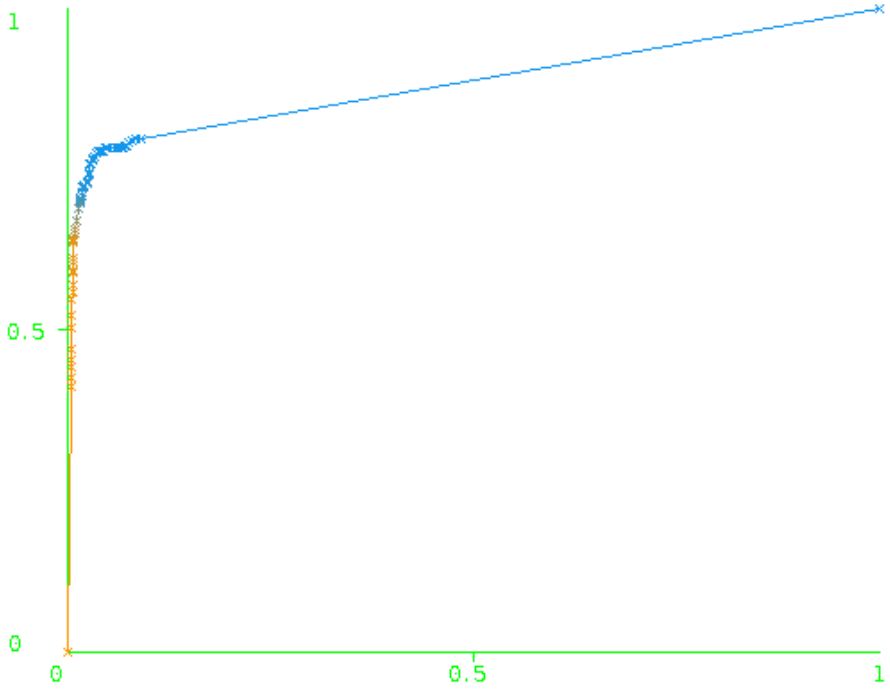
Correctly Classified	7839	69.2858%
Incorrectly Classified	3475	30.7142%

```
Kappa statistic                          0.6765
Mean absolute error                      0.0342
Root mean squared error                  0.1616
Relative absolute error                 36.0531 %
Root relative squared error             74.1778 %
Total Number of Instances           11314

=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.673	0.014	0.684	0.673	0.679	0.664	0.882	0.590	alt.atheism
	0.869	0.008	0.855	0.869	0.862	0.854	0.938	0.847	sci.crypt
	0.689	0.018	0.671	0.689	0.680	0.662	0.856	0.603	comp.sys.mac.hardware
	0.524	0.026	0.525	0.524	0.524	0.499	0.809	0.395	comp.graphics
	0.767	0.016	0.730	0.767	0.748	0.734	0.889	0.689	sci.space
	0.741	0.021	0.667	0.741	0.702	0.685	0.871	0.633	sci.med
	0.773	0.011	0.776	0.773	0.774	0.763	0.897	0.668	talk.politics.guns
	0.489	0.031	0.468	0.489	0.478	0.449	0.754	0.352	sci.electronics
	0.448	0.017	0.483	0.448	0.465	0.448	0.790	0.319	talk.religion.misc
	0.764	0.016	0.730	0.764	0.746	0.732	0.897	0.661	rec.sport.baseball
	0.813	0.009	0.841	0.813	0.827	0.817	0.922	0.754	rec.motorcycles
	0.459	0.027	0.487	0.459	0.473	0.445	0.771	0.319	comp.sys.ibm.pc.hardware
	0.848	0.007	0.872	0.848	0.860	0.852	0.948	0.814	rec.sport.hockey
	0.585	0.015	0.620	0.585	0.602	0.585	0.814	0.454	talk.politics.misc
	0.709	0.017	0.692	0.709	0.700	0.684	0.871	0.545	rec.autos
	0.650	0.018	0.669	0.650	0.659	0.641	0.843	0.486	misc.forsale
	0.783	0.013	0.769	0.783	0.776	0.763	0.904	0.707	soc.religion.christian
	0.812	0.007	0.853	0.812	0.832	0.824	0.912	0.757	talk.politics.mideast
	0.660	0.022	0.620	0.660	0.639	0.619	0.832	0.470	comp.os.ms-windows.misc
	0.688	0.011	0.768	0.688	0.726	0.713	0.890	0.672	comp.windows.x
Weighted Avg.	0.693	0.016	0.693	0.693	0.693	0.677	0.866	0.593	

ROC Curve



5.2.3 Correlation Coefficient

Correctly Classified 7800 68.9411%

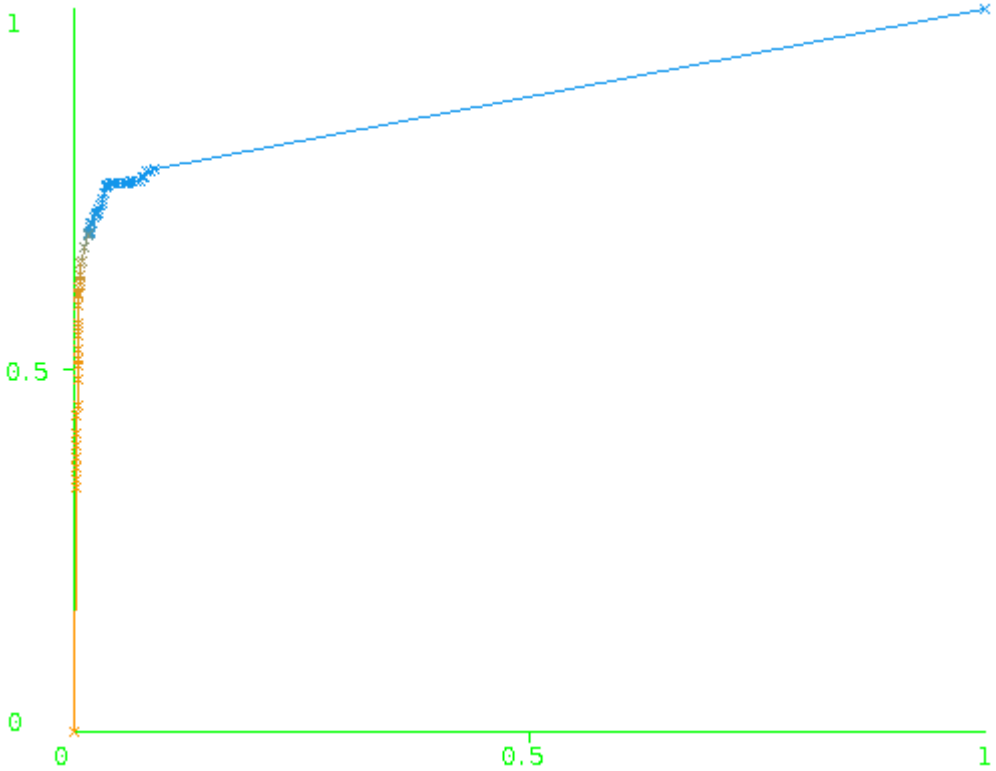
Incorrectly Classified 3514 31.0589%

```
Kappa statistic                          0.6729
Mean absolute error                      0.0345
Root mean squared error                  0.1625
Relative absolute error                 36.3183 %
Root relative squared error             74.5895 %
Total Number of Instances           11314

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.675    0.015    0.671      0.675    0.673      0.658    0.872     0.599     alt.atheism
               0.874    0.009    0.846      0.874    0.860      0.852    0.940     0.850     sci.crypt
               0.682    0.020    0.646      0.682    0.663      0.645    0.852     0.585     comp.sys.mac.hardware
               0.526    0.030    0.490      0.526    0.507      0.480    0.795     0.385     comp.graphics
               0.761    0.014    0.745      0.761    0.753      0.739    0.891     0.680     sci.space
               0.729    0.017    0.701      0.729    0.715      0.699    0.869     0.625     sci.med
               0.778    0.013    0.751      0.778    0.764      0.752    0.903     0.677     talk.politics.guns
               0.491    0.028    0.493      0.491    0.492      0.464    0.759     0.352     sci.electronics
               0.454    0.016    0.494      0.454    0.473      0.456    0.808     0.353     talk.religion.misc
               0.745    0.016    0.725      0.745    0.735      0.720    0.882     0.658     rec.sport.baseball
               0.819    0.010    0.822      0.819    0.821      0.811    0.921     0.719     rec.motorcycles
               0.431    0.028    0.459      0.431    0.444      0.415    0.757     0.299     comp.sys.ibm.pc.hardware
               0.845    0.007    0.864      0.845    0.854      0.846    0.946     0.812     rec.sport.hockey
               0.578    0.015    0.623      0.578    0.600      0.584    0.803     0.444     talk.politics.misc
               0.717    0.016    0.710      0.717    0.714      0.698    0.869     0.547     rec.autos
               0.656    0.018    0.662      0.656    0.659      0.641    0.842     0.488     misc.forsale
               0.770    0.013    0.770      0.770    0.770      0.757    0.900     0.700     soc.religion.christian
               0.824    0.009    0.826      0.824    0.825      0.816    0.920     0.749     talk.politics.mideast
               0.643    0.021    0.630      0.643    0.637      0.616    0.827     0.456     comp.os.ms-windows.misc
               0.681    0.012    0.764      0.681    0.720      0.707    0.892     0.647     comp.windows.x
Weighted Avg.  0.689    0.016    0.689      0.689    0.689      0.673    0.864     0.587
```

ROC Curve



5.3 Results by Support Vector Machine

5.3.1 Chi Squared

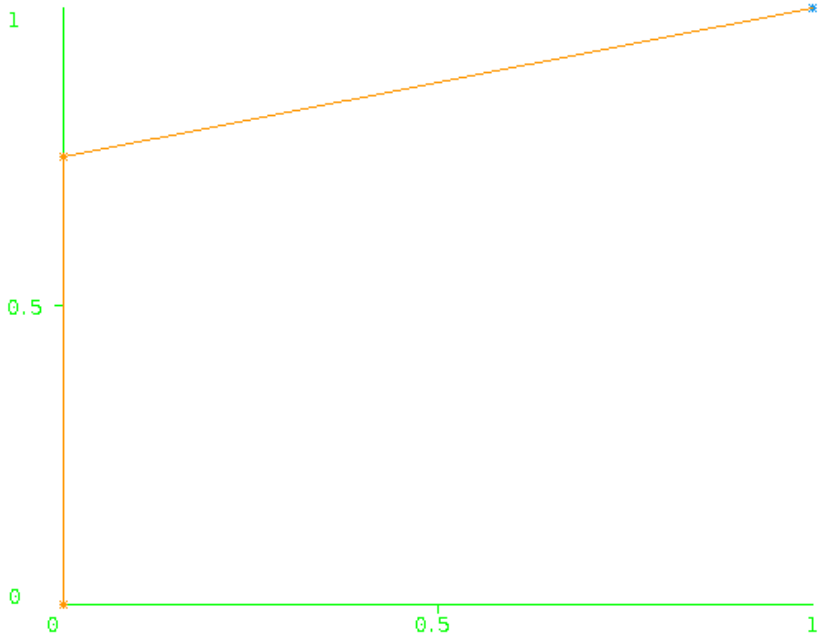
Correctly Classified	8451	74.6951%
Incorrectly Classified	2863	25.3049%

```
Kappa statistic                          0.7334
Mean absolute error                      0.0253
Root mean squared error                  0.1591
Relative absolute error                 26.6509 %
Root relative squared error             73.0081 %
Total Number of Instances           11314
```

```
=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.750	0.002	0.952	0.750	0.839	0.839	0.874	0.725	alt.atheism
	0.874	0.001	0.979	0.874	0.924	0.921	0.936	0.862	sci.crypt
	0.663	0.005	0.885	0.663	0.758	0.755	0.829	0.603	comp.sys.mac.hardware
	0.688	0.017	0.685	0.688	0.687	0.669	0.836	0.487	comp.graphics
	0.786	0.003	0.934	0.786	0.853	0.850	0.891	0.745	sci.space
	0.680	0.003	0.924	0.680	0.784	0.784	0.839	0.646	sci.med
	0.841	0.004	0.922	0.841	0.879	0.875	0.919	0.783	talk.politics.guns
	0.827	0.159	0.223	0.827	0.351	0.376	0.834	0.194	sci.electronics
	0.531	0.003	0.844	0.531	0.651	0.661	0.764	0.463	talk.religion.misc
	0.787	0.003	0.940	0.787	0.857	0.853	0.892	0.751	rec.sport.baseball
	0.789	0.002	0.963	0.789	0.868	0.866	0.894	0.771	rec.motorcycles
	0.571	0.014	0.692	0.571	0.626	0.610	0.779	0.418	comp.sys.ibm.pc.hardware
	0.855	0.001	0.988	0.855	0.917	0.915	0.927	0.853	rec.sport.hockey
	0.763	0.007	0.833	0.763	0.797	0.789	0.878	0.646	talk.politics.misc
	0.724	0.007	0.851	0.724	0.783	0.774	0.858	0.631	rec.autos
	0.692	0.009	0.812	0.692	0.747	0.737	0.842	0.578	misc.forsale
	0.835	0.009	0.842	0.835	0.838	0.829	0.913	0.711	soc.religion.christian
	0.814	0.001	0.975	0.814	0.887	0.886	0.906	0.802	talk.politics.mideast
	0.724	0.016	0.716	0.724	0.720	0.704	0.854	0.533	comp.os.ms-windows.misc
	0.673	0.003	0.924	0.673	0.779	0.779	0.835	0.639	comp.windows.x
Weighted Avg.	0.747	0.014	0.843	0.747	0.779	0.775	0.867	0.644	

ROC Curve



5.3.2 Information Gain

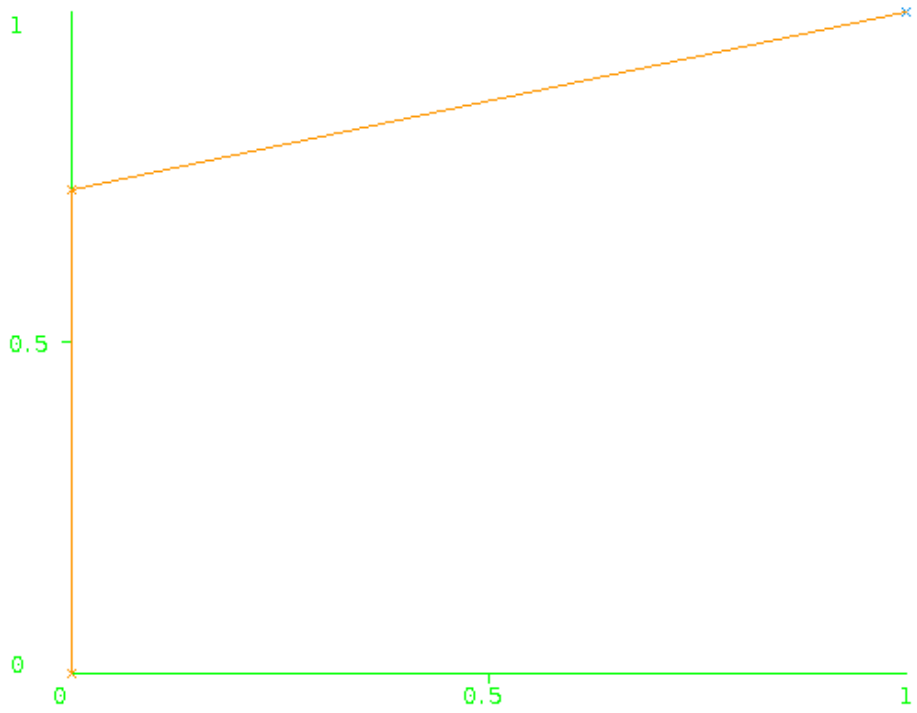
Correctly Classified	8833	78.0714%
Incorrectly Classified	2481	21.9286%

```
Kappa statistic                          0.769
Mean absolute error                      0.0219
Root mean squared error                  0.1481
Relative absolute error                 23.0949 %
Root relative squared error             67.9632 %
Total Number of Instances           11314

=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.729	0.002	0.938	0.729	0.821	0.821	0.864	0.696	alt.atheism
	0.874	0.001	0.981	0.874	0.924	0.922	0.937	0.864	sci.crypt
	0.706	0.006	0.861	0.706	0.776	0.769	0.850	0.623	comp.sys.mac.hardware
	0.753	0.029	0.586	0.753	0.659	0.644	0.862	0.454	comp.graphics
	0.836	0.004	0.925	0.836	0.879	0.874	0.916	0.783	sci.space
	0.727	0.004	0.909	0.727	0.808	0.804	0.862	0.676	sci.med
	0.846	0.004	0.922	0.846	0.883	0.878	0.921	0.788	talk.politics.guns
	0.832	0.103	0.308	0.832	0.450	0.466	0.865	0.265	sci.electronics
	0.493	0.004	0.802	0.493	0.611	0.619	0.745	0.412	talk.religion.misc
	0.854	0.004	0.919	0.854	0.885	0.880	0.925	0.793	rec.sport.baseball
	0.831	0.002	0.967	0.831	0.894	0.891	0.915	0.813	rec.motorcycles
	0.649	0.016	0.686	0.649	0.667	0.650	0.816	0.464	comp.sys.ibm.pc.hardware
	0.857	0.000	0.990	0.857	0.919	0.917	0.928	0.856	rec.sport.hockey
	0.798	0.008	0.817	0.798	0.807	0.799	0.895	0.660	talk.politics.misc
	0.773	0.007	0.861	0.773	0.815	0.806	0.883	0.677	rec.autos
	0.761	0.008	0.833	0.761	0.795	0.786	0.876	0.646	misc.forsale
	0.858	0.010	0.824	0.858	0.841	0.832	0.924	0.714	soc.religion.christian
	0.837	0.001	0.967	0.837	0.897	0.895	0.918	0.818	talk.politics.mideast
	0.739	0.014	0.751	0.739	0.745	0.731	0.863	0.569	comp.os.ms-windows.misc
	0.750	0.004	0.921	0.750	0.827	0.823	0.873	0.704	comp.windows.x
Weighted Avg.	0.781	0.012	0.838	0.781	0.798	0.793	0.884	0.668	

ROC Curve



5.3.3 Correlation Coefficient

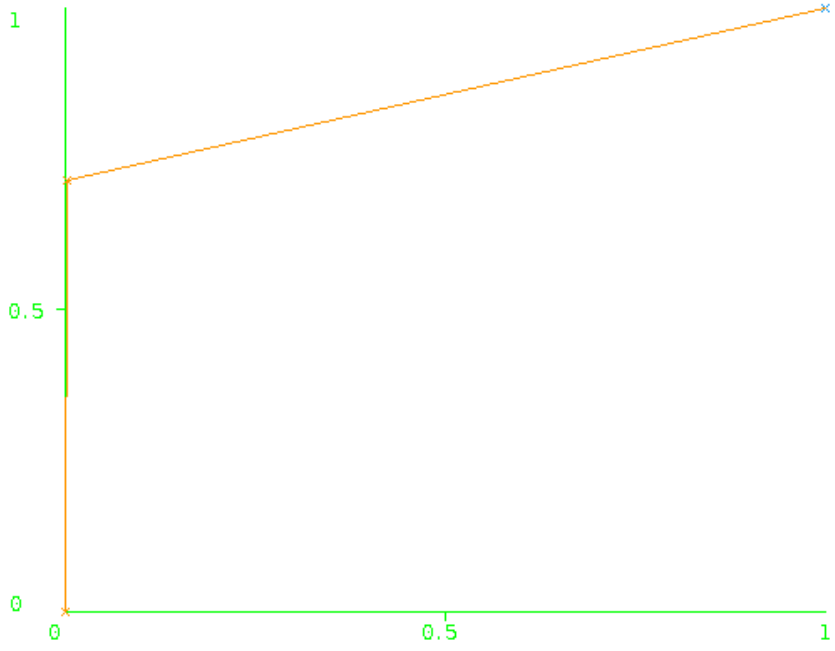
Correctly Classified	8821	77.9654%
Incorrectly Classified	2493	22.0346%

```
Kappa statistic                          0.7678
Mean absolute error                      0.022
Root mean squared error                  0.1484
Relative absolute error                 23.2066 %
Root relative squared error             68.1273 %
Total Number of Instances           11314

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.713    0.002    0.927      0.713    0.806      0.806    0.855     0.673     alt.atheism
                 0.877    0.001    0.978      0.877    0.925      0.922    0.938     0.864     sci.crypt
                 0.706    0.007    0.841      0.706    0.768      0.759    0.849     0.609     comp.sys.mac.hardware
                 0.759    0.035    0.542      0.759    0.632      0.619    0.862     0.424     comp.graphics
                 0.830    0.003    0.930      0.830    0.877      0.872    0.913     0.781     sci.space
                 0.741    0.004    0.907      0.741    0.816      0.811    0.868     0.686     sci.med
                 0.837    0.004    0.920      0.837    0.876      0.871    0.917     0.777     talk.politics.guns
                 0.819    0.094    0.325      0.819    0.466      0.478    0.863     0.276     sci.electronics
                 0.507    0.004    0.809      0.507    0.623      0.631    0.751     0.426     talk.religion.misc
                 0.868    0.006    0.895      0.868    0.881      0.875    0.931     0.783     rec.sport.baseball
                 0.826    0.002    0.965      0.826    0.890      0.887    0.912     0.806     rec.motorcycles
                 0.658    0.018    0.668      0.658    0.663      0.644    0.820     0.457     comp.sys.ibm.pc.hardware
                 0.858    0.000    0.994      0.858    0.921      0.920    0.929     0.861     rec.sport.hockey
                 0.787    0.008    0.812      0.787    0.799      0.791    0.890     0.648     talk.politics.misc
                 0.776    0.007    0.862      0.776    0.817      0.808    0.885     0.681     rec.autos
                 0.757    0.009    0.823      0.757    0.789      0.779    0.874     0.636     misc.forsale
                 0.856    0.010    0.829      0.856    0.842      0.834    0.923     0.717     soc.religion.christian
                 0.830    0.001    0.977      0.830    0.897      0.896    0.914     0.819     talk.politics.mideast
                 0.734    0.014    0.748      0.734    0.741      0.727    0.860     0.563     comp.os.ms-windows.misc
                 0.745    0.004    0.917      0.745    0.822      0.818    0.871     0.697     comp.windows.x
Weighted Avg.    0.780    0.012    0.833      0.780    0.795      0.790    0.884     0.663
```

ROC Curve



Chapter 6

Conclusion

6.1 Conclusion

From our evaluation on 20 Newsgroup dataset, the use of **Information Gain** with **the Naive Bayes** classifier gave out the best accuracy of 81.2622%.

IG measures the number of bits of information obtained by knowing the presence or absence of a term in a document. The strong DF-IF correlations means that common terms are often informative, and vice versa (this statement of course does not extend to stop words). This is contrary to a widely held belief in information retrieval that common terms are non-informative.

The chief merit of this Coefficient is that it not only gives an idea about the co-variation of the two series but also indicates the direction of relationship. Thus this Coefficient measures both the degree and direction of the correlation between two variables.

The excellent performance of DF, IG and CHI indicates that common terms are indeed informative for text categorization tasks.

ACCURACY RESULTS	CHI-SQUARED	INFORMATION GAIN	CORRELATION COEFFICENT
NAÏVE BAYES	81.1826%	**81.2622%**	80.9882%
DECISION TREE	70.9298%	69.2858%	68.9411%
SUPPORT VECTOR MACHINE	74.6951%	78.0714%	77.9654%

Table 6.1 Accuracy Comparison of Various Techniques

6.2 Limitations

1. CHI computes local scores of the term over each category and then takes maximum or average value of these scores as the global term-goodness criterion. Now there is no explicit explanation on how to choose maximum or average, Secondly, CHI cannot reflect the degree of scatter of a term.

2. The poor performance of IG is also informative its bias towards low frequency terms is known , but whether or not this theoretical weakness will cause significant accuracy loss in text categorization has not empirically examined.

3. Correlation Coefficent assumes a linear relationship between the variables even though it may not be there. It is liable to be misinterpreted, as a high degree of correlation does not necessarily mean very close relationship between the variables. Also it is tedious to calculate and it is unduly affected by the values of extreme items.

6.3 Future Scope

1. The enormous rise in computational capabilities have opened plethora of options such as Deep Learning Techniques which were previously to computationally expensive. New Methods like Char-CNN , VD-CNN and many more can be applied to drastically increase the accuracy of our system. Although they might be slow in providing the results.

2. A less computationally expensive approach can be to develop a Hybrid feature Selection scheme which is composed of filter and wrapper selection so that a combination of various type of features can be accessed. This incorporation of other features and feature weighing schemes is a possible extension of this research and remains as an interest of future study.

References

[1] Yang, Y., & Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. *Proceedings of the Fourteenth International Conference on Machine Learning ICM*, 412-420.

[2] Zhen, Z., Zeng, X., Wang, H., & Han, L. (2011). A global evaluation criterion for features selection in text categorization using Kullback-Leibler Divergence. *IEEE International Conference of Soft Computing and Pattern Recognition (SoCPaR)*. a.

[3] Qiu, L.Q., Zhao, R.Y., Zhou, G., & Yi, S.W. (2008). An extensive empirical study of feature selection for text categorization. *Seventh IEEE/ACIS International Conference on Computer and Information Science*.

[4] Dai, L., Hu, J., & Liu, W.C. (2008). Using modified chi square and rough set for text categorization with many redundant features. *IEEE International Symposium on Computational Intelligence and Design*.

[5] Zakzouk, T.S., & Mathkour, H.I. (2012). Comparing text classifiers for sports news. *Procedia Technology*, 1, 474 – 480.

[6] Kumar, M.A., & Gopal, M. (2010). An investigation on linear SVM and its variants for text categorization. *Second IEEE International Conference on Machine Learning and Computing*.

[7] Ferrer, J.A., & Juan, A. (2010). Constrained domain maximum likelihood estimation for naïve Bayes text classification. *Pattern Analysis and Applications*, 13(2), 189–196.

[8] Shanahan, J. (2001). Modeling with words: an approach to text categorization. *IEEE International Fuzzy Systems Conference*.

[9] Wang, T.Y., & Chiang, H.M. (2009). One-against-one fuzzy support vector machine classifier: An approach to text categorization. *Expert Systems with Applications*, 36, 10030–10034.

[10] G. Salton, A. Wong, C. Yang, "A vector space model for automatic indexing", Communications of the ACM, Vol.18, pp. 613–620, 1975.

[11] UCI Machine Learning Repository: Twenty Newsgroups Data Set : archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups

[12] Scikit-learn: machine learning in Python – GitHub : https://github.com/scikit-learn/scikit-learn

[13] PyQt - Python Wiki : https://wiki.python.org/moin/PyQt

[14]The Stanford Natural Language Processing Group: nlp.stanford.edu/

Appendix A : Code Snippets

Python Implementation

```
from sklearn.datasets import fetch_20newsgroups
import re
import pickle
import logging
logging.basicConfig()

from nltk import word_tokenize
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords

cachedStopWords = stopwords.words("english")

print("Fetching data...")
twenty_train = fetch_20newsgroups(subset='train', shuffle=True,
random_state=42)
print("Done!")

def tokenize(text):
    '''Input is text output is a list of words stemmed and tokenized'''
    min_length = 3
    words = map(lambda word: word.lower(), word_tokenize(text));
    tokens = [word for word in words if word not in cachedStopWords]
    tokens =(list(map(lambda token: PorterStemmer().stem(token), words)));
    p = re.compile('[a-zA-Z]+');
    filtered_tokens = list(filter(lambda token: p.match(token) and
len(token)>=min_length, tokens));
    return filtered_tokens

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_selection import SelectKBest, chi2
from sklearn import svm

def predict(text,model_num):
    model = 0
    if(model_num=="Naive Bayes"):
        model = pickle.load(open('_model_naive.p',"rb"))
    elif model_num=="Decision Tree":
        model = pickle.load(open('model_j48.p',"rb"))
    else:
        model = pickle.load(open('model_svm.p',"rb"))
    li = []
    li.append(text)
    res = model.predict(li)
    print(twenty_train.target_names[res])
    return twenty_train.target_names[res]
```

#QT GUI CODE

```
from PyQt4 import QtCore, QtGui

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig,
_encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName(_fromUtf8("MainWindow"))
        MainWindow.resize(800, 600)
        self.centralwidget = QtGui.QWidget(MainWindow)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
        self.line = QtGui.QFrame(self.centralwidget)
        self.line.setGeometry(QtCore.QRect(0, 70, 801, 20))
        self.line.setFrameShape(QtGui.QFrame.HLine)
        self.line.setFrameShadow(QtGui.QFrame.Sunken)
        self.line.setObjectName(_fromUtf8("line"))
        self.line_2 = QtGui.QFrame(self.centralwidget)
        self.line_2.setGeometry(QtCore.QRect(180, 80, 20, 491))
        font = QtGui.QFont()
        font.setPointSize(18)
        font.setBold(True)
        font.setItalic(True)
        font.setWeight(75)
        self.line_2.setFont(font)
        self.line_2.setFrameShape(QtGui.QFrame.VLine)
        self.line_2.setFrameShadow(QtGui.QFrame.Sunken)
        self.line_2.setObjectName(_fromUtf8("line_2"))
        self.label = QtGui.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(400, 10, 371, 51))
        font = QtGui.QFont()
        font.setFamily(_fromUtf8("Waree"))
        font.setPointSize(28)
        font.setItalic(True)
        self.label.setFont(font)
        self.label.setObjectName(_fromUtf8("label"))
        self.label_2 = QtGui.QLabel(self.centralwidget)
        self.label_2.setGeometry(QtCore.QRect(50, 110, 131, 25))
        self.label_2.setObjectName(_fromUtf8("label_2"))
        self.textEdit = QtGui.QTextEdit(self.centralwidget)
        self.textEdit.setGeometry(QtCore.QRect(200, 90, 561, 171))
        self.textEdit.setObjectName(_fromUtf8("textEdit"))
        self.label_3 = QtGui.QLabel(self.centralwidget)
        self.label_3.setGeometry(QtCore.QRect(10, 270, 181, 71))
        self.label_3.setObjectName(_fromUtf8("label_3"))
        self.label_4 = QtGui.QLabel(self.centralwidget)
        self.label_4.setGeometry(QtCore.QRect(40, 320, 101, 25))
        self.label_4.setObjectName(_fromUtf8("label_4"))
        self.label_5 = QtGui.QLabel(self.centralwidget)
        self.label_5.setGeometry(QtCore.QRect(50, 380, 99, 25))
```

```
        self.label_5.setObjectName(_fromUtf8("label_5"))
        self.comboBox = QtGui.QComboBox(self.centralwidget)
        self.comboBox.setGeometry(QtCore.QRect(220, 300, 331, 35))
        self.comboBox.setEditable(False)
        self.comboBox.setObjectName(_fromUtf8("comboBox"))
        self.comboBox.addItem(_fromUtf8(""))
        self.comboBox.addItem(_fromUtf8(""))
        self.comboBox.addItem(_fromUtf8(""))
        self.comboBox.addItem(_fromUtf8(""))
        self.comboBox_2 = QtGui.QComboBox(self.centralwidget)
        self.comboBox_2.setGeometry(QtCore.QRect(220, 370, 331, 35))
        self.comboBox_2.setObjectName(_fromUtf8("comboBox_2"))
        self.comboBox_2.addItem(_fromUtf8(""))
        self.comboBox_2.addItem(_fromUtf8(""))
        self.comboBox_2.addItem(_fromUtf8(""))
        self.pushButton = QtGui.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(580, 310, 181, 91))
        self.pushButton.setObjectName(_fromUtf8("pushButton"))
        self.pushButton.clicked.connect(self.button_clicked)
        self.line_3 = QtGui.QFrame(self.centralwidget)
        self.line_3.setGeometry(QtCore.QRect(0, 420, 801, 20))
        self.line_3.setFrameShape(QtGui.QFrame.HLine)
        self.line_3.setFrameShadow(QtGui.QFrame.Sunken)
        self.line_3.setObjectName(_fromUtf8("line_3"))
        self.line_4 = QtGui.QFrame(self.centralwidget)
        self.line_4.setGeometry(QtCore.QRect(-13, 360, 201, 20))
        self.line_4.setFrameShape(QtGui.QFrame.HLine)
        self.line_4.setFrameShadow(QtGui.QFrame.Sunken)
        self.line_4.setObjectName(_fromUtf8("line_4"))
        self.line_5 = QtGui.QFrame(self.centralwidget)
        self.line_5.setGeometry(QtCore.QRect(0, 260, 191, 20))
        self.line_5.setFrameShape(QtGui.QFrame.HLine)
        self.line_5.setFrameShadow(QtGui.QFrame.Sunken)
        self.line_5.setObjectName(_fromUtf8("line_5"))
        self.label_6 = QtGui.QLabel(self.centralwidget)
        self.label_6.setGeometry(QtCore.QRect(60, 460, 99, 25))
        self.label_6.setObjectName(_fromUtf8("label_6"))
        self.lineEdit = QtGui.QLineEdit(self.centralwidget)
        self.lineEdit.setGeometry(QtCore.QRect(220, 450, 261, 35))
        self.lineEdit.setObjectName(_fromUtf8("lineEdit"))
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtGui.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 33))
        self.menubar.setObjectName(_fromUtf8("menubar"))
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtGui.QStatusBar(MainWindow)
        self.statusbar.setObjectName(_fromUtf8("statusbar"))
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def textchanged(self):
        self.mytext = self.textEdit.toPlainText()
        print self.mytext

    def button_clicked(self):
        out = "predicting"
        print out
        self.lineEdit.setText(out)

        self.mytext = self.textEdit.toPlainText()
        text = str(self.comboBox_2.currentText())
```

```
        print text
        output = predict(unicode(self.mytext) ,text)
        self.lineEdit.setText(output)

    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow",
None))
        self.label.setText(_translate("MainWindow", "News Classifier",
None))
        self.label_2.setText(_translate("MainWindow", "Enter Text", None))
        self.label_3.setText(_translate("MainWindow", "Feature Selection
Technique", None))
        self.label_4.setText(_translate("MainWindow", "Technique", None))
        self.label_5.setText(_translate("MainWindow", "Classifier", None))
        self.comboBox.setItemText(0, _translate("MainWindow", "Chi-Squared",
None))
        self.comboBox.setItemText(1, _translate("MainWindow", "Information
Gain", None))
        self.comboBox.setItemText(2, _translate("MainWindow", "Corelation
Coefficient", None))
        self.comboBox.setItemText(3, _translate("MainWindow", "Hybrid",
None))
        self.comboBox_2.setItemText(0, _translate("MainWindow", "Naive
Bayes", None))
        self.comboBox_2.setItemText(1, _translate("MainWindow", "Decision
Tree", None))
        self.comboBox_2.setItemText(2, _translate("MainWindow", "Support
Vector Machine", None))
        self.pushButton.setText(_translate("MainWindow", "Predict", None))
        self.label_6.setText(_translate("MainWindow", "Result", None))

def textchanged(text):
    print text


def button_clicked(mytext):
   print "Button 1 clicked"


if __name__ == "__main__":
    import sys
    app = QtGui.QApplication(sys.argv)
    MainWindow = QtGui.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

PRE MODEL COMPUTATION GENERIC CODE

```
from sklearn.datasets import fetch_20newsgroups
import re
import logging
logging.basicConfig()
from nltk import word_tokenize
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords

cachedStopWords = stopwords.words("english")
print("Fetching data...")
twenty_train = fetch_20newsgroups(subset='train', shuffle=True,
random_state=42)
print("Done!")

def tokenize(text):
    '''Input is text output is a list of words stemmed and tokenized'''
    min_length = 3
    words = map(lambda word: word.lower(), word_tokenize(text));
    tokens = [word for word in words if word not in cachedStopWords]
    tokens =(list(map(lambda token: PorterStemmer().stem(token), words)));
    p = re.compile('[a-zA-Z]+');
    filtered_tokens = list(filter(lambda token: p.match(token) and
len(token)>=min_length, tokens));
    return filtered_tokens

from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer(tokenizer=tokenize, analyzer='word')
X_train_counts = count_vect.fit_transform(twenty_train.data)
X_train_counts.shape


count_vect.vocabulary_.get(u'algorithm')

from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_selection import SelectKBest, chi2

from sklearn import svm
from sklearn.pipeline import Pipeline
text_clf = Pipeline([('vect', CountVectorizer(tokenizer=tokenize,
analyzer='word')),
                     ('tdf-idf', TfidfTransformer()),
                     ('chi2', SelectKBest(chi(),k=1000
                     ('clf', svm.SVC()),
])
text_clf = text_clf.fit(twenty_train.data, twenty_train.target)

import numpy as np
twenty_test = fetch_20newsgroups(subset='test', shuffle=True,
random_state=42)
docs_test = twenty_test.data
predicted = text_clf.predict(docs_test)

ac = np.mean(predicted == twenty_test.target)
print ac*100
pickle.dump(text_clf, open("model.p", "wb" ))

model = pickle.load('model.p')
res = model.predict(["Some text"])
print res
```

Appendix B : System Snapshots

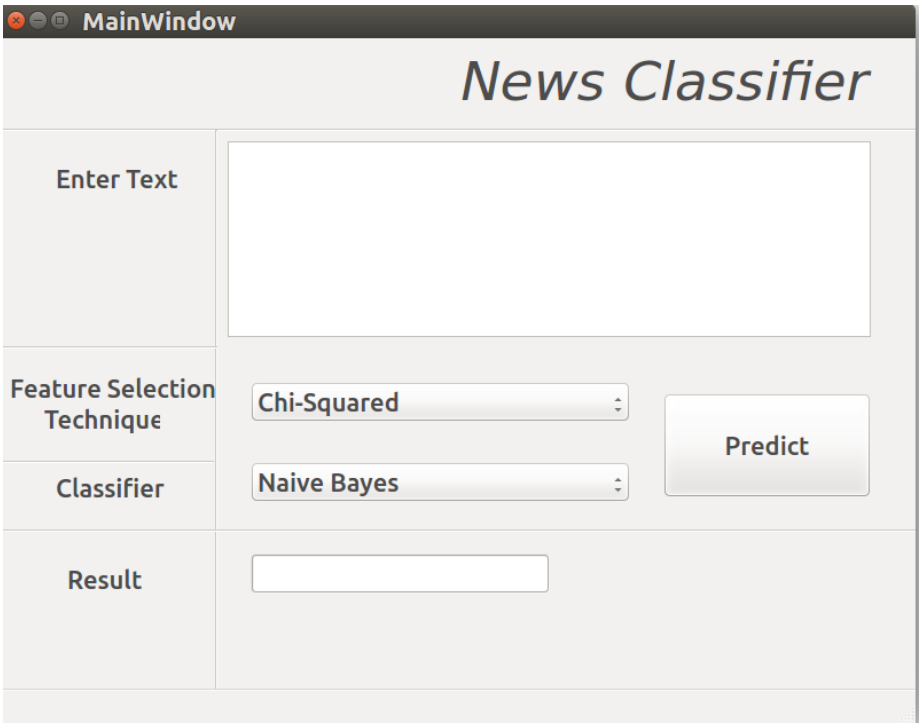


Figure B.1 Basic User Interface

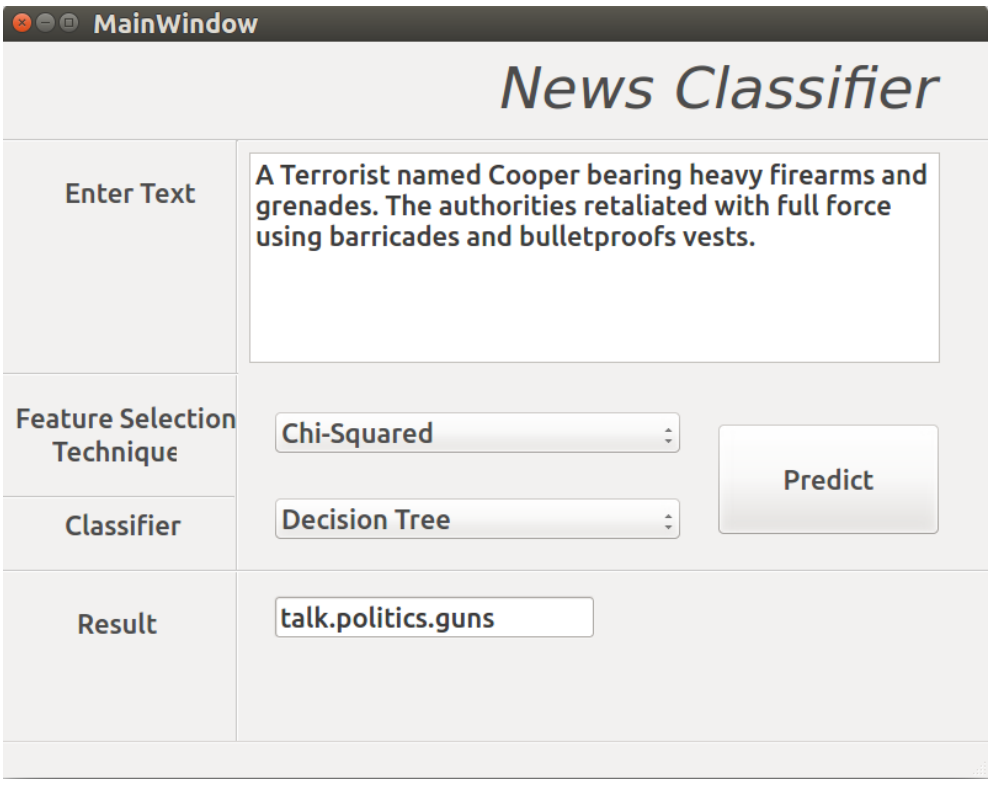


Figure B.2: A query related to politics and guns

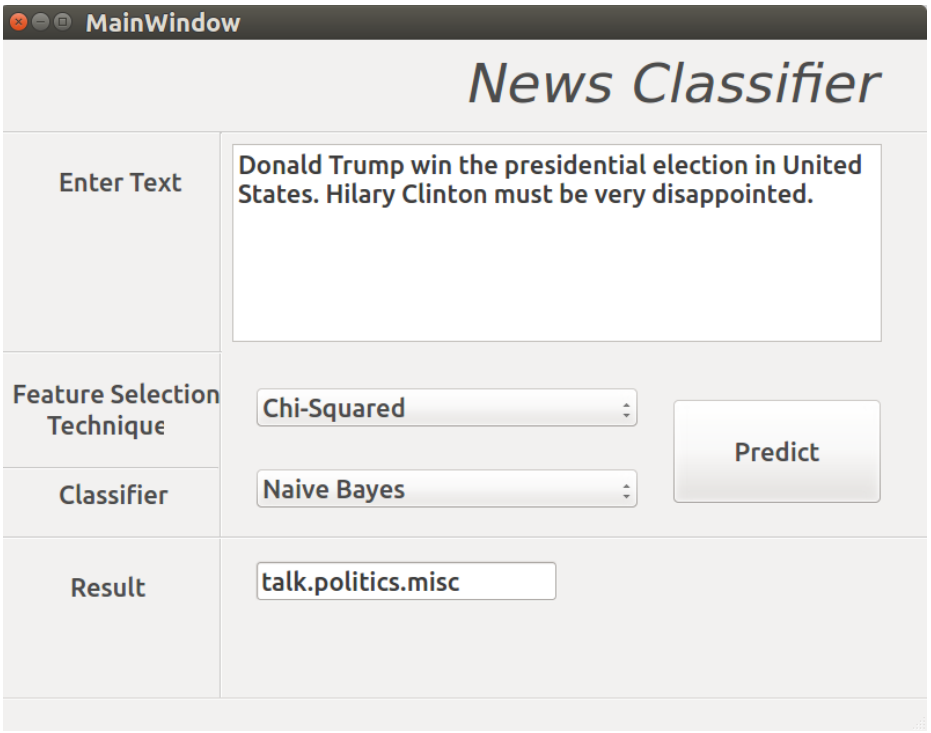


Figure B.3: A query related to politics

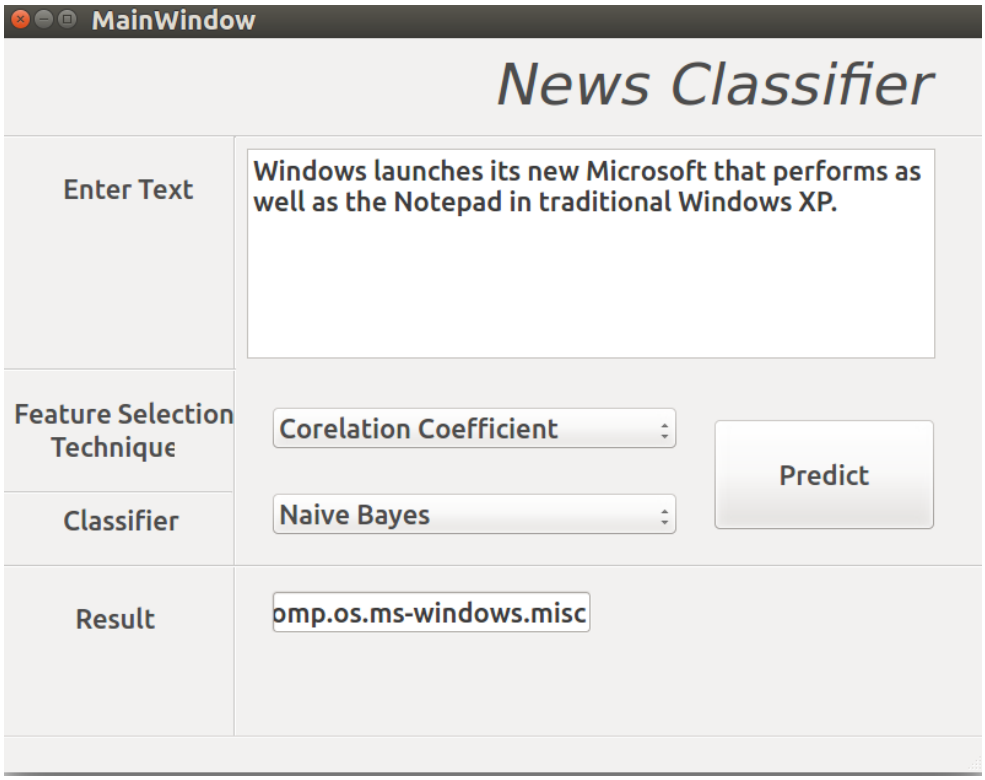


Figure B.4: A query related to Microsoft Windows

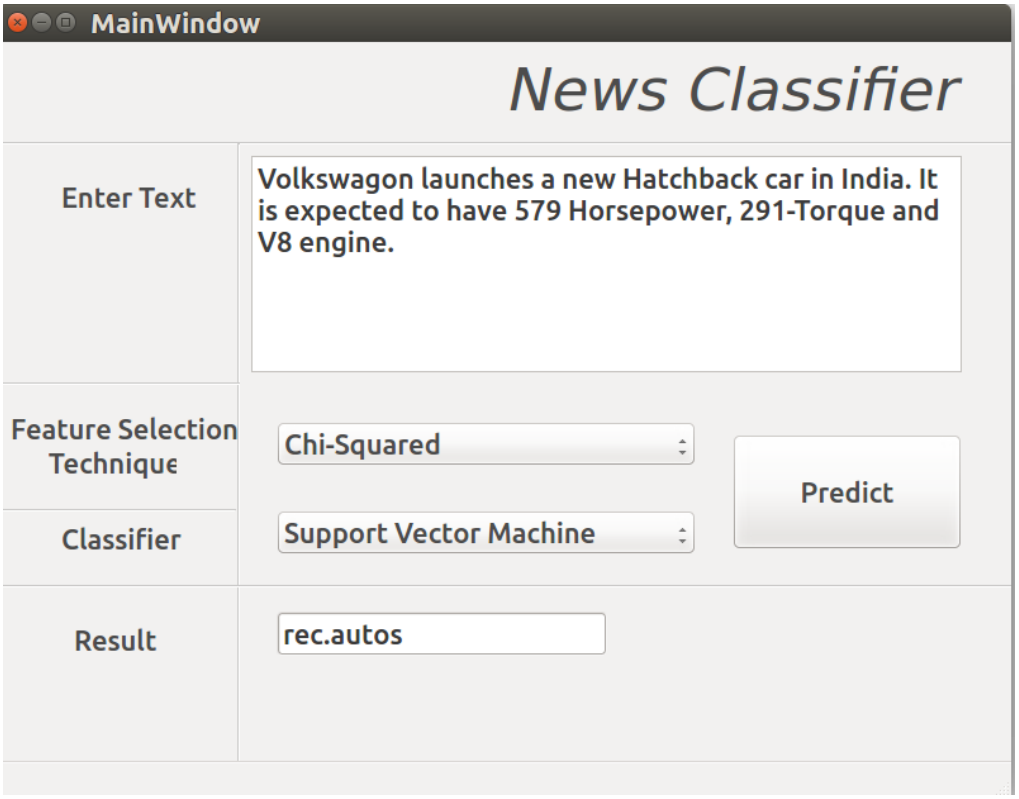


Figure B.5: A query related to automobiles