MINOR PROJECT

"Predicting and Evaluating the Popularity of Online News"

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

Degree of Bachelor of Technology
in
Software Engineering (SE)

Under the supervision of

Prof. Kusum Lata
(Assistant Professor, CSE)

By:

Arjun Rajpal (2K14/SE/021)

Arpit Jain (2K14/SE/022)

Avinav Goel (2K14/SE/024)

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 **"*Predicting and Evaluating the Popularity of Online News*"** in fulfilment 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 **Prof. Kusum Lata, 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

ARJUN RAJPAL 2K14/SE/021

ARPIT JAIN 2K14/SE/022

AVINAV GOEL 2K14/SE/024

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. Kusum Lata, 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. Kusum Lata, 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 **ARJUN RAJPAL 2K14/SE/021, ARPIT JAIN 2K14/SE/022** and **AVINAV GOEL 2K14/SE/024,** the bonafide students of **Bachelor of Technology in Software Engineering** of **Delhi Technological University** (Formerly Delhi College Of Engineering), New Delhi of **2014–2018 batch** have completed their minor project entitled "***Predicting and Evaluating the Popularity of Online News***" under the supervision of **Prof. Kusum Lata, Assistant Professor, CSE DEPARTMENT**.

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:

Prof. Kusum Lata
Assistant Professor
Computer Science & Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Shahbad, Daulatpur, Bawana Road, Delhi – 110042

ABSTRACT

Consider the situation where an online news publishing agency is browsing submissions of news articles, but can only accept a few without going over budget or without overwhelming the audience. How does the agency determine which news articles will become popular or even viral, and which news articles will be mostly ignored by the general public? Are there any predictors that indicate how many times an article will be shared amongst audiences? Which articles should the agency purchase and publish?

Implied in this question is the classification problem of binning. Our class variable, the number of shares, is a metric that defines how often an article is shared on social media, but it is a continuous variable and so its binning is not obvious. However, we are interested primarily in articles with high popularity as our positive class, more so than articles with low popularity. So how do we bin a numerical attribute into classes of ‘obscure’, ‘mediocre’, popular’, ‘viral’, etc.?

With the expansion of the Internet, more and more people enjoys reading and sharing online news articles. The number of shares under a news article indicates how popular the news is. In this project, we intend to find the best model and set of feature to predict the popularity of online news, using machine learning techniques. Our data comes from Mashable, a well-known online news website. We implemented various learning algorithms on the dataset, ranging from various regressions to SVM. Their performances are recorded and compared. Feature selection method has been used to improve performance and reduce features. SVM turns out to be the best model for prediction, and it can achieve an accuracy of 67% with optimal parameters. Our work can help online news companies to determine news popularity before publication.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1-INTRODUCTION

1.1 OBJECTIVE

The aim of the project is to **— *Predict and Evaluate the Popularity of Online News.***

Description of the problem:

The problem is to build a model that predicts the number of shares on Social Media and in turn help to determine the popularity of Online News articles. Currently media houses rely solely on human judgement to estimate the popularity of any article posted on the web.

Such a model is helpful for the media publishing houses to make important decisions like when to publish an article, if a given article will be popular, which category is famous, etc.

Some prediction approaches are based on analyzing early user's comments [1], or features about post contents and domains [2]. Another proposed method [3] predicted the article's popularity not only based on its own appeal, but also other articles that it is competing with. Prediction models with SVMs, Ranking SVMs [3], Naive Bayes [2] are investigated, and more advanced algorithms such as Random Forest, Adaptive Boosting [4] could increase the precision. This report however, incorporates a broader and more abstracter set of features, and starts with basic regression and classification models to advanced ones, with elaborations about effective feature selection.

File/Dataset:

The data set consists of 39,644 instances of articles published on Mashable.com from 2013 to 2014. The dataset contains 58 total attributes, ranging from the URL of the article, the number of keywords of specific types, what day of the week the article was published, the results of a Latent Dirichlet Allocation algorithm, and the dependent variable: the number of shares of the article. The data set was accessed at the UCI dataset repository referenced in the appendix. There are no missing values in the data set and no significant noise

Fields:

0. url: URL of the article
1. timedelta: Days between the article publication and the dataset acquisition
2. n_tokens_title: Number of words in the title
3. n_tokens_content: Number of words in the content
4. n_unique_tokens: Rate of unique words in the content
5. n_non_stop_words: Rate of non-stop words in the content
6. n_non_stop_unique_tokens: Rate of unique non-stop words in the content

7. num_hrefs: Number of links
8. num_self_hrefs: Number of links to other articles published by Mashable
9. num_imgs: Number of images
10. num_videos: Number of videos
11. average_token_length: Average length of the words in the content
12. num_keywords: Number of keywords in the metadata
13. data_channel_is_lifestyle: Is data channel 'Lifestyle'?
14. data_channel_is_entertainment: Is data channel 'Entertainment'?
15. data_channel_is_bus: Is data channel 'Business'?
16. data_channel_is_socmed: Is data channel 'Social Media'?
17. data_channel_is_tech: Is data channel 'Tech'?
18. data_channel_is_world: Is data channel 'World'?
19. kw_min_min: Worst keyword (min. shares)
20. kw_max_min: Worst keyword (max. shares)
21. kw_avg_min: Worst keyword (avg. shares)
22. kw_min_max: Best keyword (min. shares)
23. kw_max_max: Best keyword (max. shares)
24. kw_avg_max: Best keyword (avg. shares)
25. kw_min_avg: Avg. keyword (min. shares)
26. kw_max_avg: Avg. keyword (max. shares)
27. kw_avg_avg: Avg. keyword (avg. shares)
28. self_reference_min_shares: Min. shares of referenced articles in Mashable
29. self_reference_max_shares: Max. shares of referenced articles in Mashable
30. self_reference_avg_sharess: Avg. shares of referenced articles in Mashable
31. weekday_is_monday: Was the article published on a Monday?
32. weekday_is_tuesday: Was the article published on a Tuesday?
33. weekday_is_wednesday: Was the article published on a Wednesday?
34. weekday_is_thursday: Was the article published on a Thursday?
35. weekday_is_friday: Was the article published on a Friday?
36. weekday_is_saturday: Was the article published on a Saturday?
37. weekday_is_sunday: Was the article published on a Sunday?
38. is_weekend: Was the article published on the weekend?
39. LDA_00: Closeness to LDA topic 0
40. LDA_01: Closeness to LDA topic 1
41. LDA_02: Closeness to LDA topic 2
42. LDA_03: Closeness to LDA topic 3
43. LDA_04: Closeness to LDA topic 4
44. global_subjectivity: Text subjectivity
45. global_sentiment_polarity: Text sentiment polarity
46. global_rate_positive_words: Rate of positive words in the content
47. global_rate_negative_words: Rate of negative words in the content
48. rate_positive_words: Rate of positive words among non-neutral tokens
49. rate_negative_words: Rate of negative words among non-neutral tokens
50. avg_positive_polarity: Avg. polarity of positive words

51. min_positive_polarity: Min. polarity of positive words
52. max_positive_polarity: Max. polarity of positive words
53. avg_negative_polarity: Avg. polarity of negative words
54. min_negative_polarity: Min. polarity of negative words
55. max_negative_polarity: Max. polarity of negative words
56. title_subjectivity: Title subjectivity
57. title_sentiment_polarity: Title polarity
58. abs_title_subjectivity: Absolute subjectivity level
59. abs_title_sentiment_polarity: Absolute polarity level
60. shares: Number of shares (target)

Chapter 2-DATA MINING

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 society as a Whole in recent years, due to the Wide availability of huge amounts of data and the 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.

2.1 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.

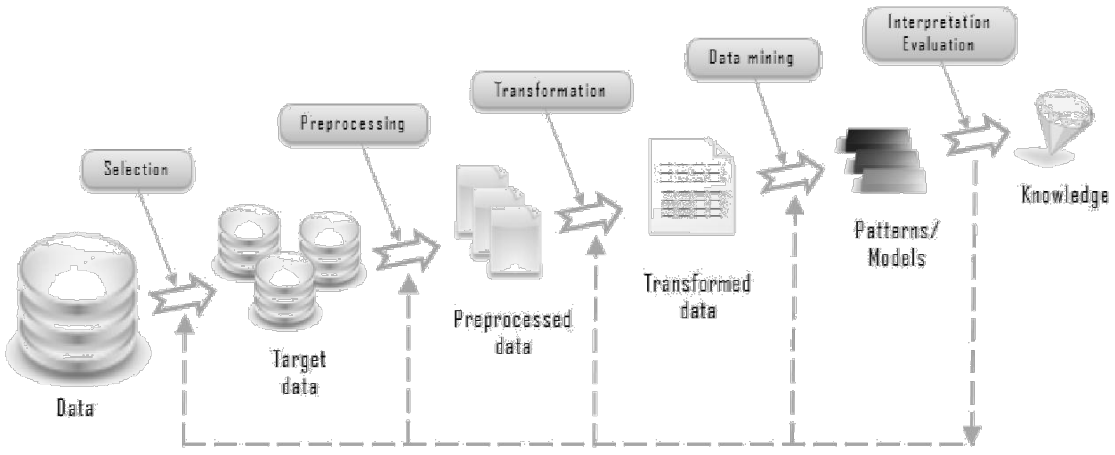


Figure 2.1: Data mining as a step in the process of knowledge discovery.[1]

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)
5. Data mining (an essential process where intelligent methods are applied in order to extract data patterns)
6. Pattern evaluation (to identify the truly interesting patterns representing knowledge based on some interestingness measures)
7. 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

2.2 Data Mining Functionalities-What Kinds of Patterns Can Be Mined?

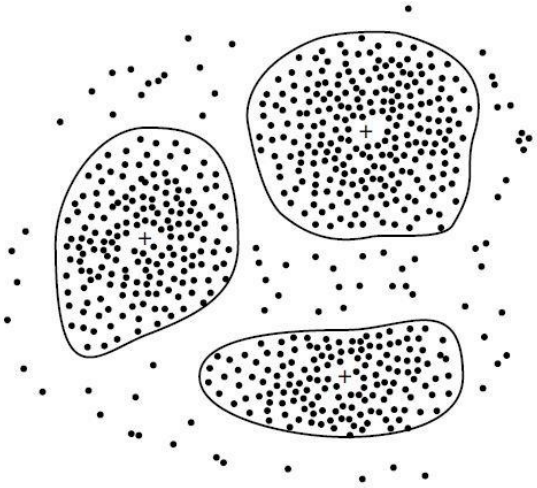
Data mining functionalities include the discovery of concept/class descriptions, associations and correlations, classification, prediction, clustering, trend analysis, outlier and deviation analysis, and similarity analysis. Data mining is the task of discovering interesting patterns from large amounts 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 humans; valid on test data with some degree of certainty; and potentially useful, novel, or validates a hunch about which the user was curious. Measures of pattern interestingness, either objective or subjective, can be used to guide the discovery process. In general, data 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.

2.2.1 Classification and Prediction

Classification is the process of finding a model (or function) that describes and distinguishes data classes or concepts, for the purpose of being able to use the model to predict the class of 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).

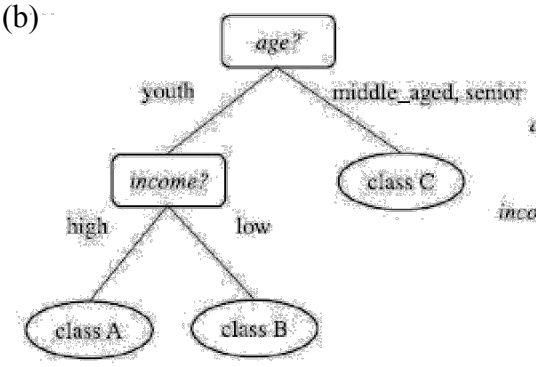
“How is the derived model presented?” The derived model may be represented in various forms, such as classification (IF-THEN) rules, decision trees, mathematical formulae, or neural networks. A decision tree is a flow-chart-like tree structure, where each node denotes a test on an attribute value, each branch represents an outcome of the test, and tree leaves represent classes or class distributions. Decision trees can easily be converted to classification rules. A neural network, when used for classification, is typically a collection of neuron-like processing units with weighted connections between the units. There are many other methods for constructing classification models, such as na'ive Bayesian classification, support vector

machines, and k-nearest neighbor classification.



(a)
age(X, "youth") AND income(X, "high") -> class(X, "A")
agc(X, "youth")ANDincome(X, "low") -> class(X, "8")
age(X, "middlc_agcd") -> class(X, "C")
age(X, "scnior") -> class(X, “C")

(b)



(c)

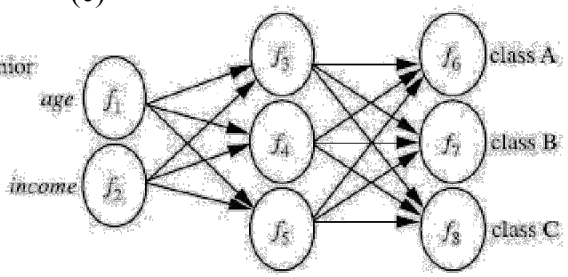


Figure 2.2: A classification model can be represented in various forms, such as (a) IF-THEN rules, (19) a decision tree, or a (c) neural network.”

Whereas classification predicts categorical (discrete, unordered) labels, prediction models continuous-valued functions. That is, it is used to predict missing or unavailable numerical data values rather than class labels.

Regression analysis is a statistical methodology that is most often used for numeric prediction, although other methods exist as well. Prediction also encompasses the identification of distribution trends based on the available data. Classification and prediction may need to be preceded by relevance analysis, which attempts to identify attributes that do not contribute to

the classification or prediction process.

2.2.2 Cluster Analysis

“What is cluster analysis?” Unlike classification and prediction, which analyze class-labeled data objects, clustering analyses 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 to begin with. Clustering can be used to generate such labels. The objects are clustered or grouped based on the principle of maximizing the intra class similarity and minimizing the interclass similarity. That is, clusters of objects are formed so that objects Within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other clusters. Each cluster that is formed can be viewed as a class of objects, from Which rules can be derived. Clustering can also facilitate taxonomy formation, that is, the organization of observations into a hierarchy of classes that group similar events together.
Figure 2.3: A 2-D plot, showing three data clusters. Each cluster “center” is marked with a

2.3 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) novel. An interesting pattern represents knowledge. Several objective measures of pattern interestingness exist.

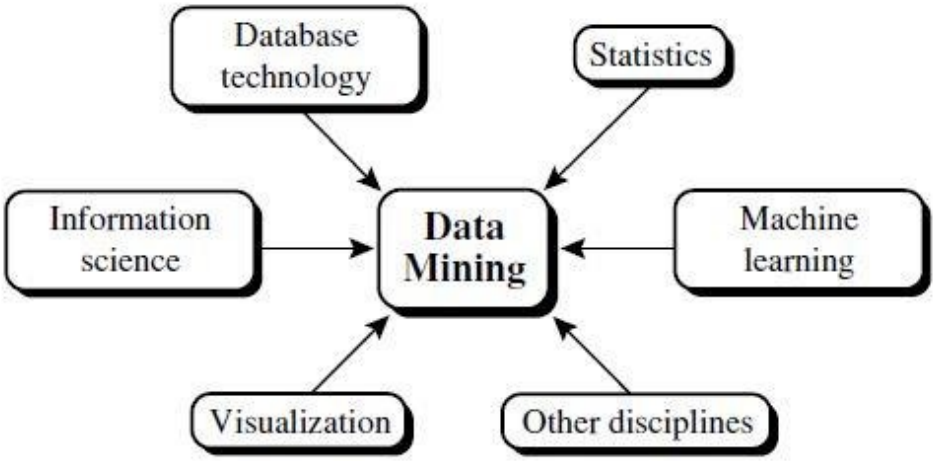
Secondly -“Can a data mining system generate all of the interesting patterns?”- refers to the completeness of a data mining algorithm. It is often unrealistic and inefficient for data mining systems to generate all of the possible patterns. Instead, user-provided constraints and interestingness measures 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 generate only interesting patterns. This would be much more efficient for users and data mining systems, because neither would have to search through the patterns generated in order to identify the truly interesting ones.[”

2.4 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 recognition, spatial data analysis, image databases, signal processing, and many application

fields, such as business, economics, and bioinformatics.



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 techniques utilized: 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.

CHAPTER-3 DATA PROCESSING

Today's real-world databases are highly susceptible to noisy, missing, and inconsistent data due to their typically huge size (often several gigabytes or more) and their likely origin from multiple, heterogeneous sources. Low-quality data will lead to low-quality mining results. "How can the data be preprocessed in order to help improve the quality of the data and, consequently, of the mining results? How can the data be preprocessed so as to improve the efficiency and ease of the mining process?"

There are a number of data preprocessing techniques. Data cleaning can be applied to remove noise and correct inconsistencies in the data. Data integration merges data from multiple sources into a coherent data store, such as a data warehouse. Data transformations, such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency of mining algorithms involving distance measurements. Data reduction can reduce the data size by aggregating, eliminating redundant features, or clustering, for instance. These techniques are not mutually exclusive; they may work together. For example, data cleaning can involve transformations to correct wrong data, such as by transforming all entries for a date field to a common format. Data processing techniques, when applied before mining, can substantially improve the overall quality of the patterns mined and/or the time required for the actual mining"

3.1 Why Preprocess the Data?

The data you wish to analyse by data mining techniques are incomplete (lacking attribute values or certain attributes of interest, or containing only aggregate data), noisy (containing errors, or outlier values that deviate from the expected), and inconsistent (e.g., containing discrepancies in the department codes used to categorize items).

Incomplete, noisy, and inconsistent data are commonplace properties of large real world databases and data warehouses. Incomplete data can occur for a number of reasons. Attributes of interest may not always be available, such as customer information for sales transaction data. Other data may not be included simply because it was not considered important at the time of entry. Relevant data may not be recorded due to a misunderstanding, or because of equipment malfunctions. Data that were inconsistent With other recorded data may have been deleted. Furthermore, the recording of the history or modifications to the data may have been overlooked. Missing data, particularly for tuples With missing values for some attributes, may need to be inferred.

There are many possible reasons for noisy data (having incorrect attribute values). The data collection instruments used may be faulty. There may have been human or computer errors occurring at data entry. Errors in data transmission can also occur. There may be technology limitations, such as limited buffer size for coordinating synchronized data transfer and consumption. Incorrect data may also result from inconsistencies in naming conventions or

data codes used, or inconsistent formats for input fields, such as date. Duplicate tuples also require data cleaning.

3.2 Descriptive Data Summarization

For data preprocessing to be successful, it is essential to have an overall picture of your data. Descriptive data summarization techniques can be used to identify the typical properties of your data and highlight which data values should be treated as noise or outliers. Thus, we first introduce the basic concepts of descriptive data summarization before getting into the concrete workings of data preprocessing techniques. For many data preprocessing tasks, users would like to learn about data characteristics regarding both central tendency and dispersion of the data. Measures of central tendency include mean, median, mode, and midrange, while measures of data dispersion include quartiles, interquartile range (IQR), and variance. These descriptive statistics are of great help in understanding the distribution of the data.

3.2.1 Measuring the Central Tendency

The most common and most effective numerical measure of the "center" of a set of data is the (arithmetic) mean. Let x1, x2, , xN be a set of N values or observations. The mean of this set of values is

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} = \frac{x_1 + x_2 + \cdots + x_N}{N}.$$

Although the mean is the single most useful quantity for describing a data set, it is not always the best way of measuring the center of the data. A major problem with the mean is its sensitivity to extreme (e.g., outlier) values. Even a small number of extreme values can corrupt the mean. For example, the mean salary at a company may be substantially pushed up by that of a few highly paid managers. Similarly, the average score of a class in an exam could be pulled down quite a bit by a few very low scores. To offset the effect caused by a small number of extreme values, we can instead use the trimmed mean, which is the mean obtained after chopping off values at the high and low extremes. For example, we can sort the values observed for salary and remove the top and bottom 2% before computing the mean. We should avoid trimming too large a portion (such as 20%) at both ends as this can result in the loss of valuable information.

For skewed (asymmetric) data, a better measure of the center of data is the median. Suppose that a given data set of N distinct values is sorted in numerical order. If N is odd, then the

median is the middle value of the ordered set; otherwise (i.e., if N is even), the median is the average of the middle two values.

We can, however, easily approximate the median value of a data set. Assume that data are grouped in intervals according to their xi data values and that the frequency (i.e., number of data values) of each interval is known. Let the interval that contains the median frequency be the median interval. We can approximate the median of the entire data set (e.g., the median salary) by interpolation using the formula:

$$median = L_1 + \left(\frac{N/2 - (\sum freq)_t}{freq_{median}}\right) width$$

Where L_1 is the lower boundary of the median interval, N is the number of values in the entire data set, (∑freq); is the sum of the frequencies of all of the intervals that are lower than the median interval, f req_{median} is the frequency of the median interval, and width is the width of the median interval.

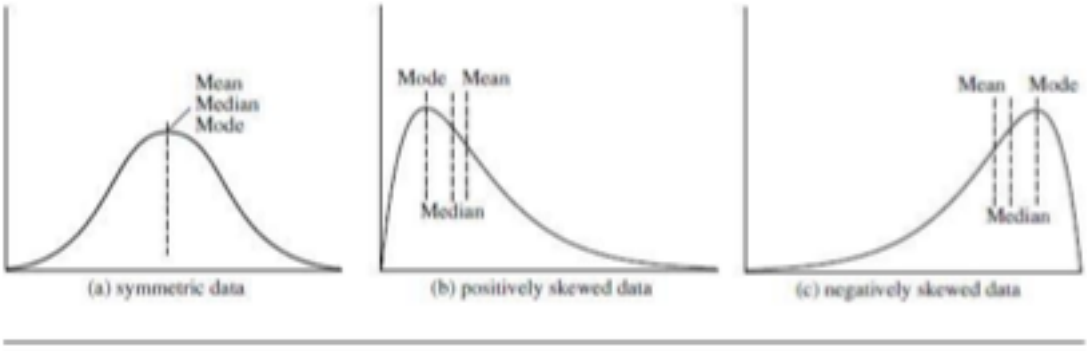


Figure3.1: Mean, median, and mode of symmetric versus positively and negatively skewed data

Another measure of central tendency is the mode. The mode for a set of data is the value that occurs most frequently in the set. It is possible for the greatest frequency to correspond to several different values, which results in more than one mode. For unimodal frequency curves that are moderately skewed (asymmetrical), we have the following empirical relation:

mean X mode = 3 X (mean - median).

This implies that the mode for unimodal frequency curves that are moderately skewed can easily be computed if the mean and median values are known. In a unimodal frequency curve with perfect symmetric data distribution, the mean, median, and mode are all at the same center value, as shown in Figure 3.1(a). However, data in most real applications are not

symmetric. They may instead be either positively skewed, where the mode occurs at a value that is smaller than the median (Figure 3.1(b)), or negatively skewed, where the mode occurs at a value greater than the median (Figure 3.1(c)).

3.2.2 Measuring the Dispersion of Data

The degree to which numerical data tend to spread is called the dispersion, or variance of the data. The most common measures of data dispersion are range, the five-number summary (based on quartiles), the interquartile range, and the standard deviation. Boxplots can be plotted based on the five-number summary and are a useful tool for identifying outliers.

Range, Quartiles, Outliers, and Boxplots

Let x1, x2, ,xN be a set of observations for some attribute. The **range** of the set is the difference between the largest (maX()) and smallest (min()) values. For the remainder of this section, let's assume that the data are sorted in increasing numerical order. The **kth percentile** of a set of data in numerical order is the value x, having the property that k percent of the data entries lie at or below xi. The median is the 5oth percentile.

The most commonly used percentiles other than the median are **quartiles**. The first quartile, denoted by Q, is the 25th percentile; the third quartile, denoted by Q3, is the 75th percentile. The quartiles, including the median, give some indication of the center, spread, and shape of a distribution. The distance between the first and third quartiles is a simple measure of spread that gives the range covered by the middle half of the data. This distance is called the **interquartile range (IQR)** and is defined as

$$IQR = Q_3 - Q_1.$$

No single numerical measure of spread, such as IQR, is very useful for describing skewed distributions. The spreads of two sides of a skewed distribution are unequal (Figure 3.1). Therefore, it is more informative to also provide the two quartiles Q1 and Q3, along with the median. A common rule of thumb for identifying suspected outliers is to single out values falling at least 1.5 x I QR above the third quartile or below the first quartile.

Because Q, the median, and Q3 together contain no information about the endpoints (e.g., tails) of the data, a fuller summary of the shape of a distribution can be obtained by providing the lowest and highest data values as well. This is known as the five-number summary. The five-number summary of a distribution consists of the median, the quartiles Q1 and Q3, and the smallest and largest individual observations, written in the order Minimum, Q1, Median, Q3, Maximum.

Boxplots are a popular way of visualizing a distribution. A boxplot incorporates the five-

number summary as follows:

- Typically, the ends of the box are at the quartiles, so that the box length is the interquartile range, IQR.
- The median is marked by a line within the box.
- Two lines (called *whiskers*) outside the box extend to the smallest (Minimum) and largest (*Maximum*) observations.

When dealing with a moderate number of observations, it is worthwhile to plot potential outliers individually. To do this in a boxplot, the whiskers are extended to the extreme low and high observations only if these values are less than 1.5 x IQR beyond the quartiles. Otherwise, the whiskers terminate at the most extreme observations occurring within 1.5 x I QR of the quartiles. The remaining cases are plotted individually. Boxplots can be used in the comparisons of several sets of compatible data. The efficient computation of boxplots, or even approximate boxplots (based on approximates of the five-number summary), remains a challenging issue for the mining of large data sets.

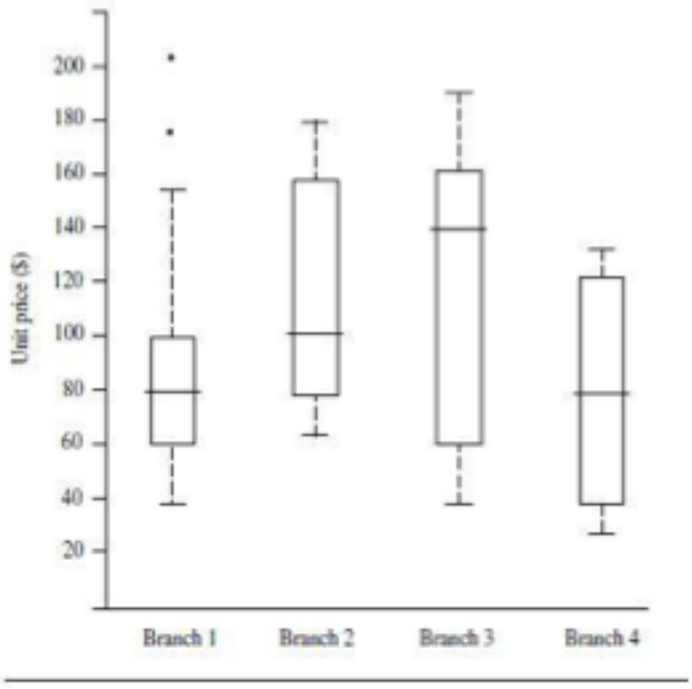


Figure 3.2: Example of boxplot showing the unit price data for items sold at four branches of All Electronics during a given time period

Variance and Standard Deviation

The **variance** of N observations, X_1, X_2,……. ,X_N, is

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \frac{1}{N}\left[\sum x_i^2 - \frac{1}{N}(\sum x_i)^2\right]$$

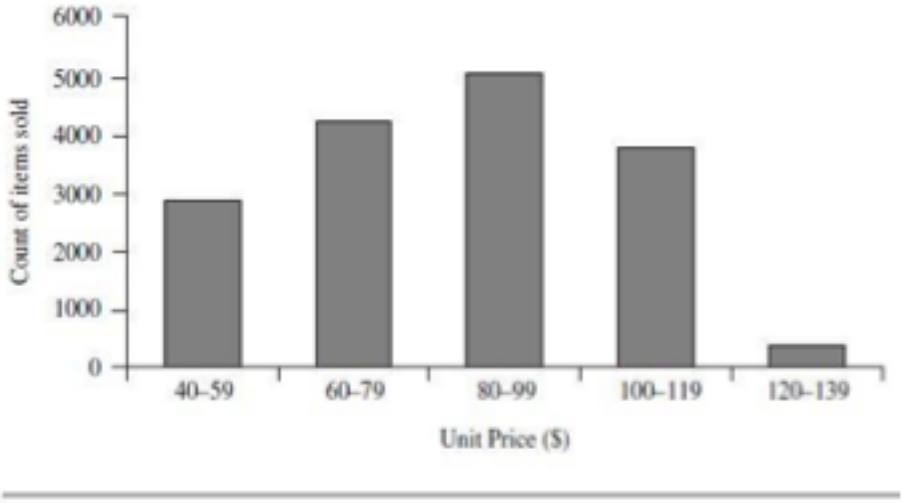
where $\overline{x}$ is the mean value of the observations. The **standard deviation**, σ, of the observations is the square root of the variance, 0'2. The basic properties of the standard deviation, 6, as a measure of spread are

- measures spread about the mean and should be used only when the mean is chosen as the measure of center.
- σ=0 only when there is no spread, that is, when all observations have the same value. Otherwise σ > 0.

The variance and standard deviation are algebraic measures because they can be computed from distributive measures.

3.3 Graphic Displays of Basic Descriptive Data Summaries

Aside from the bar charts, pie charts, and line graphs used in most statistical or graphical data presentation software packages, there are other popular types of graphs for the display of data



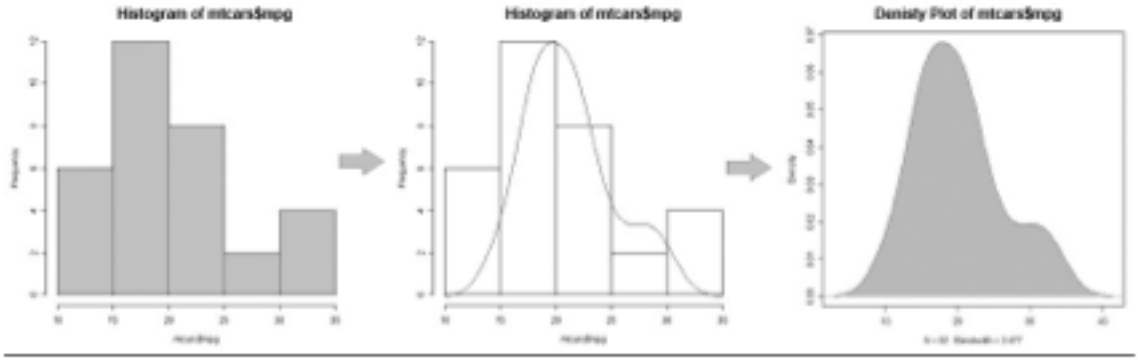
summaries and distributions, such as *histograms*, *quantile plots*, *q-q plots*, *scatter plots*, and *loess curves*. These graphs are very helpful for the visual inspection of your data.

Figure 3.3: Example of histogram

Plotting **histograms**, or **frequency histograms**, is a graphical method for summarizing the distribution of a given attribute. A histogram for an attribute A partitions the data distribution of A into disjoint subsets, or buckets. Typically, the width of each bucket is uniform. Each bucket is represented by a rectangle whose height is equal to the count or

relative frequency of the values at the bucket. If A is categoric, such as automobile model or item type, then one rectangle is drawn for each known value of A, and the resulting graph is more commonly referred to as a **bar chart**. If A is numeric, the term histogram is preferred.

However, histograms can be a poor method for determining the shape of a distribution because it is so strongly affected by the number of bins used. **Kernal density plots** (or just density plots) are usually a much more effective way to View the distribution of a variable.



A **quantile plot** is a simple and effective way to have a first look at a univariate data distribution. First, it displays all of the data for the given attribute (allowing the user to assess both the overall behaviour and unusual occurrences). Second, it plots quantile information. Let x_i, for i = 1 to N, be the data sorted in increasing order so that x_1 is the smallest observation and am is the largest. Each observation, x,, is paired with a percentage, f_i, which indicates that approximately lOOf_i % of the data are below or equal to the value, x_i. We say "approximately" because there may not be a value with exactly a fraction, f_i, of the data below or equal to x_i. Note that the 0.25 quantile corresponds to quartile Q_1, the 0.50 quantile is the median, and the 0.75 quantile is Q_3.
Let

$$f_i = \frac{i - 0.5}{N}.$$

These numbers increase in equal steps of 1/N, ranging from 1/2N (which is slightly above zero) to 1-1/2N (which is slightly below one). On a quantile plot, x_i is graphed against f_i. This allows us to compare different distributions based on their quantiles. For example, given the quantile plots of sales data for two different time periods, we can compare their Q_1, median, Q_3, and other f_i values at a glance.

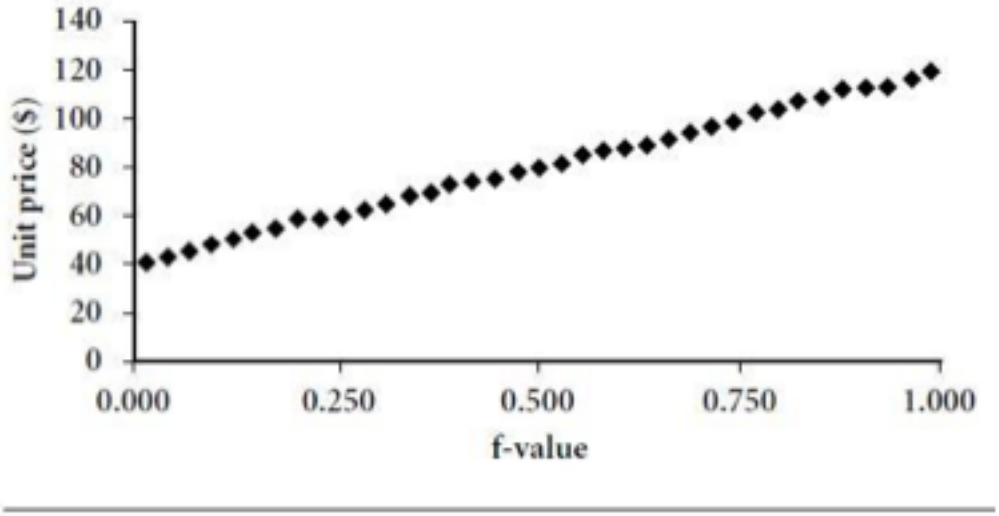


Figure 3.5: Example of quantile plot

A **quantile-quantile plot**, or q-q plot, graphs the quantiles of one univariate distribution against the corresponding quantiles of another. It is a powerful visualization tool in that it allows the user to view whether there is a shift in going from one distribution to another. Suppose that we have two sets of observations for the variable unit price, taken from two different branch locations. Let $x_1,\ldots..,x_N$ be the data from the first branch, and y_1, ..., y_M be the data from the second, where each data set is sorted in increasing order. If M = N (i.e., the number of points in each set is the same), then we simply plot y_i against x_i, where y_i and x_i are both (i-0.5)/N quantiles of their respective data sets. If M < N (i.e., the second branch has fewer observations than the first), there can be only M points on the q-q plot. Here, y_i is the (i-0.5)/M quantile of the y data, which is plotted against the (i-0.5) = M quantile of the x data. This computation typically involves interpolation.

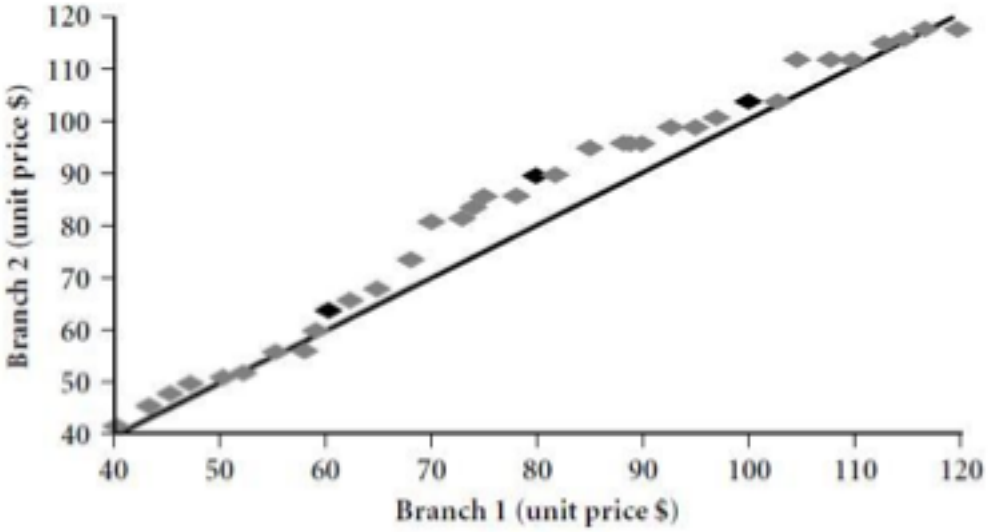


Figure 3.6: Example of quantile-quantile plot.

A **scatter plot** is one of the most effective graphical methods for determining if there appears to be a relationship, pattern, or trend between two numerical attributes. To construct a scatter plot, each pair of values is treated as a pair of coordinates in an algebraic sense and plotted as points in the plane. The scatter plot is a useful method for providing a first look at bivariate data to see clusters of points and outliers, or to explore the possibility of correlation relationships.

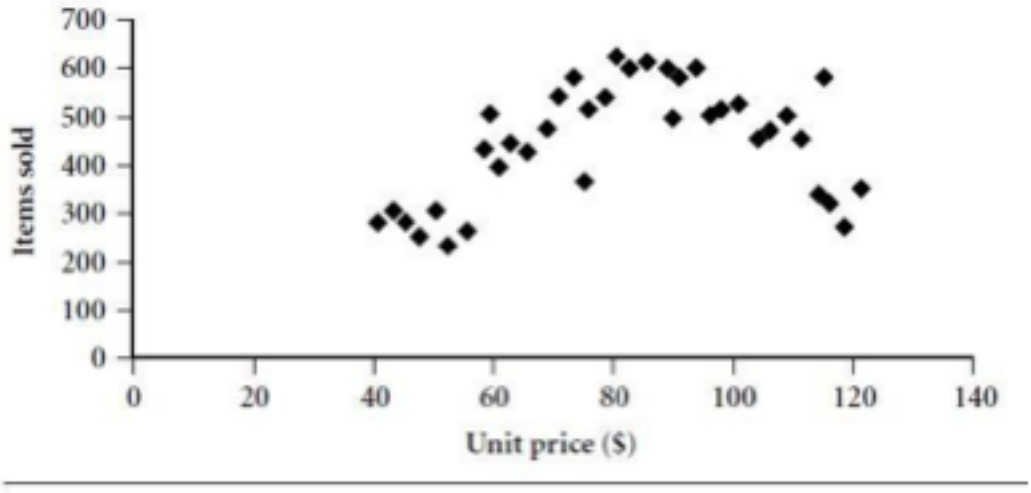


Figure 3.7: Example of scatter plot.

In Figure 3.8, we see examples of positive and negative correlations between two attributes in two different data sets. Figure 3.9 shows three cases for which there is no correlation relationship between the two attributes in each of the given data sets.

When dealing with several attributes, the scatter-plot matrix is a useful extension to the scatter plot. Given n attributes, a scatter-plot matrix is an nxn grid of scatter plots that provides a visualization of each attribute (or dimension) with every other attribute. The scatter-plot matrix becomes less effective as the number of attributes under study grows.

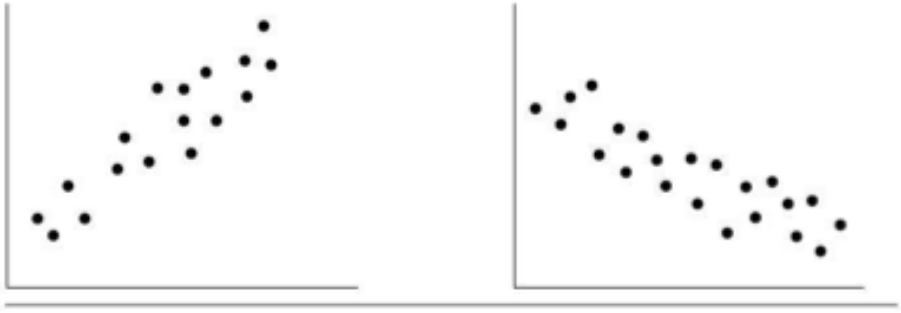


Figure 3.8: Scatter plots can be used to find (a) positive or (b) negative correlations between

attributes

Figure 3.9: Three cases where there is no observed correlation between the two plotted attributes in each of the data sets

A loess curve is another important exploratory graphic aid that adds a smooth curve to a scatter plot in order to provide better perception of the pattern of dependence. The word loess is short for "local regression."

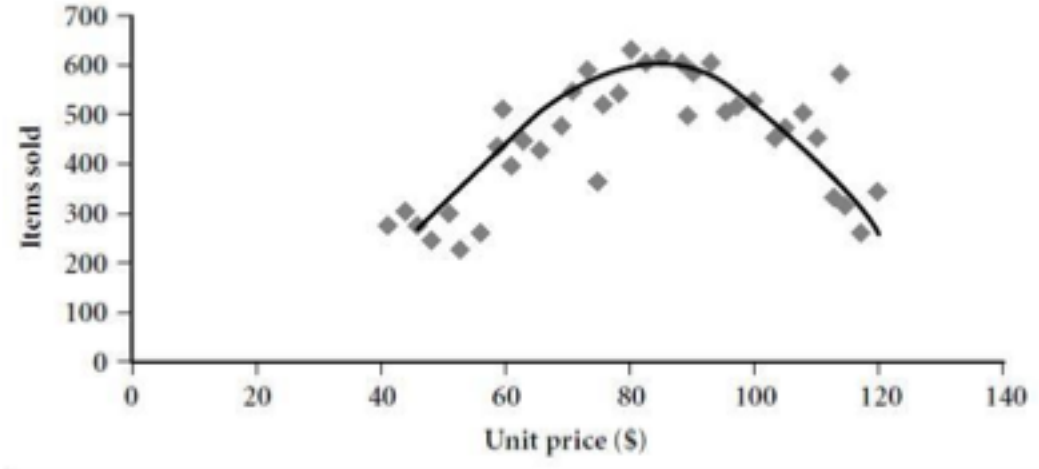


Figure 3.10: Example of 10655 plot

To fit a loess curve, values need to be set for two parameters-oL, a smoothing parameter, and A, the degree of the polynomials that are fitted by the regression. While oL can be any positive number (typical values are between 1/4 and 1), A can be 1 or 2. The goal in choosing oL is to produce a fit that is as smooth as possible without unduly distorting the underlying pattern in the data. The curve becomes smoother as a increases. There may be some lack of fit, however, indicating possible "missing" data patterns. If Cl is very small, the underlying pattern is tracked, yet overfitting of the data may occur where local "wiggles" in the curve may not be supported by the data. If the underlying pattern of the data has a "gentle" curvature with no local maxima and minima, then local linear fitting is usually sufficient (A = 1). However, if there are local maxima or minima, then local quadratic fitting (A = 2) typically does a better

job of following the pattern of the data and maintaining local smoothness. In conclusion, descriptive data summaries provide valuable insight into the overall behaviour of your data. By helping to identify noise and outliers, they are especially useful for data cleaning.

3.4 Forms of Data Preprocessing

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.

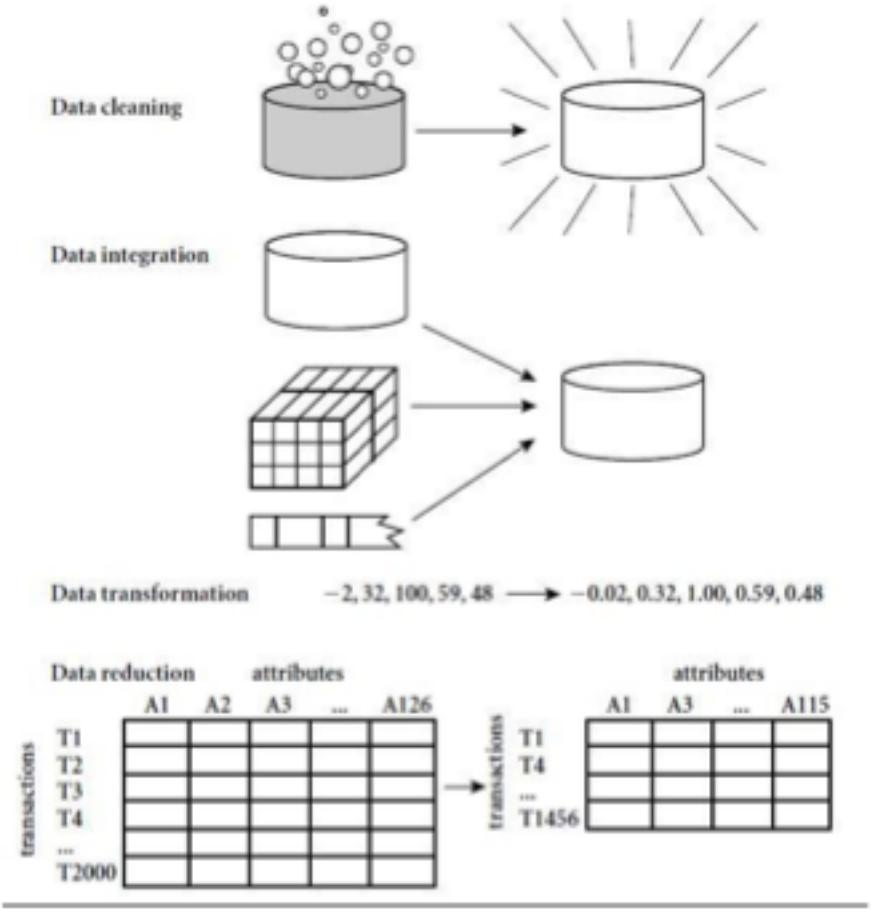


Figure 3.11: Forms of Data Preprocessing

3.4.1 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.

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.

3.4.2 Data Integration

It is likely that your data analysis task will involve data integration, which combines data from multiple sources into a coherent data store. These sources may include multiple databases, data cubes, or flat files. There are a number of issues to consider during data integration like schema integration and object matching. How can equivalent real-world entities from multiple data sources be matched up? This is referred to as the **entity identification problem.**

Redundancy is another important issue. An attribute (such as *annual revenue*, for instance) may be redundant if it can be "derived" from another attribute or set of attributes. Inconsistencies in attribute or dimension naming can also cause redundancies in the resulting data set.

Some redundancies can be detected by correlation analysis. Given two attributes, such analysis can measure how strongly one attribute implies the other, based on the available data. For numerical attributes, we can evaluate the correlation between two attributes, A and B, by computing the correlation coefficient,

$$r_{A,B} = \frac{\sum_{i=1}^{N}(a_i - \bar{A})(b_i - \bar{B})}{N\sigma_A\sigma_B} = \frac{\sum_{i=1}^{N}(a_i b_i) - N\overline{AB})}{N\sigma_A\sigma_B}$$

where N is the number of tuples, a_i and b_i are the respective values of A and B in tuple i, A and B are the respective mean values of A and B, σ_A and σ_B are the respective standard deviations of A and B and $\sum(a_i b_i)$ is the sum of the AB cross-product (that is, for each tuple, the value for A is multiplied by the value for B in that tuple). Note that $-1 \leq r_{A,B} \leq +1$. If $r_{A,B}$ is greater than 0, then A and B are positively correlated, meaning that the values of A increase as the values of B increase. The higher the value, the stronger the correlation (i.e., the more each attribute implies the other). Hence, a higher value may indicate that A (or B) may be removed as a redundancy. If the resulting value is equal to 0, then A and B are independent and there is no correlation between them. If the resulting value is less than 0, then A and B are negatively correlated, where the values of one attribute increase as the values of the other attribute decrease. This means that each attribute discourages the other. Scatter plots can also be used to view correlations between attributes. Note that correlation does not imply causality. That is, if A and B are correlated, this does not necessarily imply that A causes B or that B causes A. For categorical (discrete) data, a correlation relationship between two attributes, A and B, can be discovered by a X^2 (chi-square) test.

A third important issue in data integration is the *detection and resolution of data value conflicts*. For example, for the same real-world entity, attribute values from different sources may differ. This may be due to differences in representation, scaling, or encoding. When matching attributes from one database to another during integration, special attention must be paid to the structure of the data. This is to ensure that any attribute functional dependencies and referential constraints in the source system match those in the target system. The semantic heterogeneity and structure of data pose great challenges in data integration. Careful integration of the data from multiple sources can help reduce and avoid redundancies and inconsistencies in the resulting data set. This can help improve the accuracy and speed of the subsequent mining process.

3.4.3 Data Transformation

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

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

- **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.

  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.

3.4.4 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.

Chapter-4 MACHINE LEARNING ALGORITHMS SELECTION AND ASSESSMENT

4.1 INTRODUCTION

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.

4.2 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, X, is represented by an n-dimensional **attribute vector, X** = (x_1, x_2,…. x_N), depicting n measurements made on the tuple from n database attributes, respectively, A_1, A_2,…..,A_n. A2, ..., An. Each tuple, 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 (**X**), that can predict the associated class label y of a given tuple **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 formulael"

"*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 (**X**), where **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 **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, **X**. There are various predictor error measures.

4.2 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.

4.2.1 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.

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.

Figure 4.1: Training, test and validation set

4.3 Comparing Classification and Prediction Methods

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.

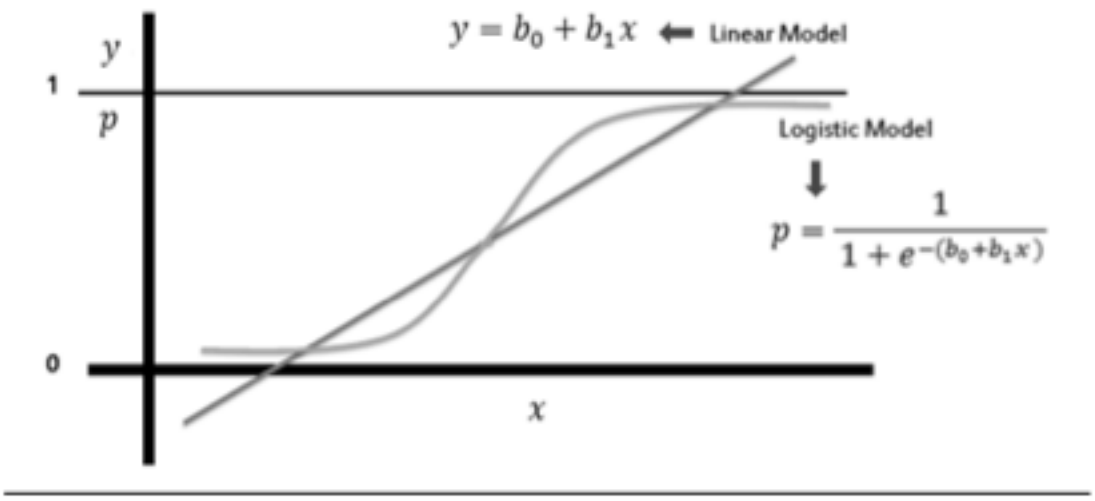
Recent data mining research has contributed to the development of scalable algorithms for classification and prediction. Additional contributions include the exploration of mined "associations" between attributes and their use for effective classification.

4.4 Linear Regression

4.5 Classification by Regression-Based Methods

Linear regression is used to model continuous-valued functions. It is widely used, owing largely to its simplicity. "Can it also be used to predict categorical labels?" **Generalized linear models** represent the theoretical foundation on which linear regression can be applied to the modeling of categorical response variables. In generalized linear models, the variance of the response variable, y, is a function of the mean value of y, unlike in linear regression, where the variance of y is constant. Common types of generalized linear models include **logistic regression** and **Poisson regression**. Logistic regression models the probability of some event occurring as a linear function of a set of predictor variables. Count data frequently exhibit a Poisson distribution and are commonly modeled using Poisson regression.

Log-linear models approximate *discrete* multidimensional probability distributions. They may be used to estimate the probability value associated with data cube cells. Like other forms of regression analysis, logistic regression makes use of one or more predictor variables that may be either continuous or categorical data. Unlike ordinary linear regression, however, logistic regression is used for predicting binary outcomes of the dependent variable (treating the



dependent variable as the outcome of a Bernoulli trial) rather than a continuous outcome. Given this difference, it is necessary that logistic regression take the natural logarithm of the odds of the dependent variable being a case (referred to as the logit or log-odds) to create a continuous criterion as a transformed version of the dependent variable. Thus the logic transformation is referred to as the link function in logistic regression-although the dependent variable in logistic regression is binomial, the logit is the continuous criterion upon which linear regression is conducted.

The logit of success is then fitted to the predictors using linear regression analysis. The predicted value of the logit is converted back into predicted odds via the inverse of the natural logarithm, namely the exponential function. Thus, although the observed dependent variable in logistic regression is a zero-or-one variable, the logistic regression estimates the odds, as a continuous variable, that the dependent variable is a success (a case). In some applications the odds are all that is needed. In others, a specific yes-or-no prediction is needed for whether the dependent variable is or is not a case; this categorical prediction can be based on the computed odds of a success, with predicted odds above some chosen cutoff value being translated into a prediction of a success.

4.5.1 Logistic Regression

Logistic regression predicts the probability of an outcome that can only have two values (i.e. a dichotomy). The prediction is based on the use of one or several predictors (numerical and categorical). A linear regression is not appropriate for predicting the value of a binary variable for two reasons.

- A linear regression will predict values outside the acceptable range (e.g. predicting probabilities outside the range 0 to 1)

$$\frac{p}{1-p} = \exp(b_0 + b_1 x)$$

- Since the dichotomous experiments can only have one of two possible values for each experiment, the residuals will not be normally distributed about the predicted line.

On the other hand, a logistic regression produces a logistic curve, which is limited to values between 0 and 1. Logistic regression is similar to a linear regression, but the curve is constructed using the natural logarithm of the "odds" of the target variable, rather than the probability. Moreover, the predictors do not have to be normally distributed or have equal variance in each group.

Figure 4.2: Comparison of linear regression model and logistic regression model

In the logistic regression the constant (b_o) moves the curve left and right and the slope (b_1) defines the steepness of the curve. By simple transformation, the logistic regression equation can be written in terms of an odds ratio.
Finally, taking the natural log of both sides, we can write the equation in terms of log-odds (logit) which is a linear function of the predictors. The coefficient (b_1) is the amount the logit (log-odds) changes with a one unit change in x.

$$\ln\left(\frac{p}{1-p}\right) = b_0 + b_1 x$$

As mentioned before, logistic regression can handle any number of numerical and/or categorical variables.

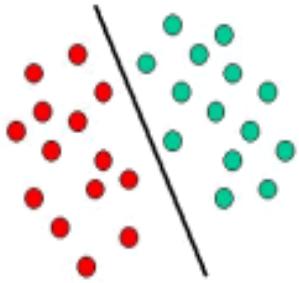
$$p = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + x_2 b_2 + \cdots + b_p x_p)}}$$

There are several analogies between linear regression and logistic regression. Just as ordinary least square regression is the method used to estimate coefficients for the best fit line in linear regression, logistic regression uses maximum likelihood estimation (MLE) to obtain the model coefficients that relate predictors to the target. After this initial function is estimated, the process is repeated until LL (Log Likelihood) does not change significantly.

4.6 Support Vector Machine (SVM)

Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. A schematic example is shown in the illustration below. In this example, the objects belong either to class GREEN or RED. The separating line defines a boundary on the right side of which all objects are GREEN and to the left of which all objects are RED. Any new

object (white circle) falling to the right is labeled, i.e., classified, as GREEN (or classified as RED should it fall to the left of the separating line).



The above is a classic example of a linear classifier, i.e., a classifier that separates a set of objects into their respective groups (GREEN and RED in this case) with a line. Most classification tasks, however, are not that simple, and often more complex structures are needed in order to make an optimal separation, i.e., correctly classify new objects (test cases) on the basis of the examples that are available (train cases). This situation is depicted in the illustration below. Compared to the previous schematic, it is clear that a full separation of the GREEN and RED objects would require a curve (which is more complex than a line). Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers. Support Vector Machines are particularly suited to handle such tasks.

4.6.1 Linear SVM

We are given a training dataset of n points of the form
$(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$

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

Any hyperplane can be written as the set of points $\vec{x}$ satisfying

$$\vec{w} \cdot \vec{x} - b = 0,$$

Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

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 $\vec{w}$

4.6.2 Non Linear Classification

The original maximum-margin hyperplane algorithm proposed by Vapnik in 1963 constructed a linear classifier. However, in 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes. The resulting algorithm is formally similar, except that every dot product is replaced by a nonlinear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. The transformation may be nonlinear and the transformed space high dimensional; although the classifier is a hyperplane in the transformed feature space, it may be nonlinear in the original input space.
It is noteworthy that working in a higher-dimensional feature space increases the generalization error of support vector machines, although given enough samples the algorithm still performs well.

Some common kernels include:

- Polynomial (homogeneous): $k(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j})^d$
- Polynomial (inhomogeneous): $k(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j} + 1)^d$
- Gaussian radial basis function: $k(\vec{x_i}, \vec{x_j}) = \exp(-\gamma \|\vec{x_i} - \vec{x_j}\|^2)$ for $\gamma > 0$. Sometimes parametrized using $\gamma = 1/2\sigma^2$
- Hyperbolic tangent: $k(\vec{x_i}, \vec{x_j}) = \tanh(\kappa \vec{x_i} \cdot \vec{x_j} + c)$, for some (not every) $\kappa > 0$ and $c < 0$

4.6.3 Complexity analysis

Support Vector Machines are powerful tools, but their compute and storage requirements increase rapidly with the number of training vectors. The core of an SVM is a quadratic programming problem (QP), separating support vectors from the rest of the training data. The QP solver used by this libsvm-based implementation scales between $O(n_{features} \times n_{samples}^2)$ and $O(n_{features} \times n_{samples}^3)$ depending on how efficiently the libsvm cache is used in practice (dataset dependent). If the data is very sparse $n_{features}$ should be replaced by the average number of non-zero features in a sample vector.

4.6.4 Evaluation of binary classifiers

4.6.4.1 Sensitivity and Specificity:

1) Sensitivity or True Positive Rate (TPR), also known as recall, is the proportion of people that tested positive and are positive (True Positive, TP) of all the people that actually are positive (Condition Positive, CP = TP + FN).

2) Specificity (SPC) or True Negative Rate (TNR) is the proportion of people that tested negative and are negative (True Negative, TN) of all the people that actually are negative (Condition Negative, CN = TN + FP).

4.6.4.2 Positive and negative predictive values:

1) Positive prediction value answers the question "If the test result is *positive*, how well does that *predict* an actual presence of disease?". It is calculated as TP/(TP + FP); that is, it is the proportion of true positives out of all positive results.

2) Negative prediction value is the same, but for negatives.

4.6.4.3 Single metrics:

1) Fraction Correct (FC) measures the fraction of all instances that are correctly categorized; it is the ratio of the number of correct classifications to the total number of correct or incorrect classifications.

2) F-score is a combination of the precision and the recall, providing a single score.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

4.7 Model Selection

Suppose that we have generated two models, M1 and M2 (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 M1 and M2 may appear different, that difference may not be statistically significant. What if any difference between the two may just be attributed to chance?

4.7.1 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.

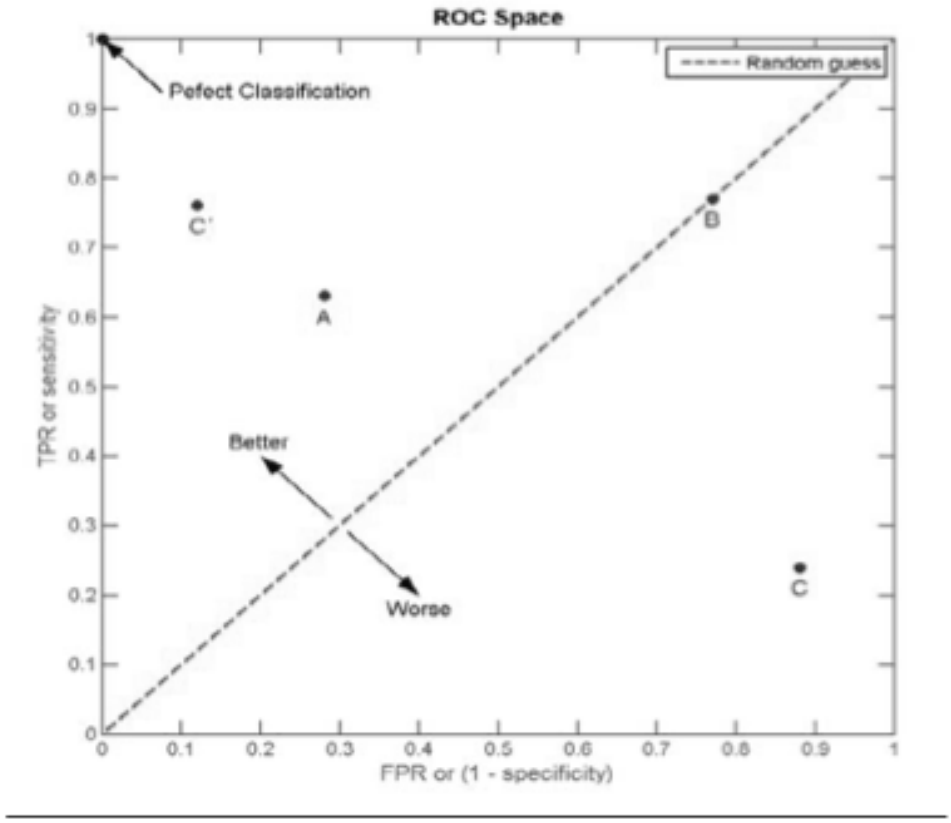


Figure 4.3 ROC Space

*I*n 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 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.

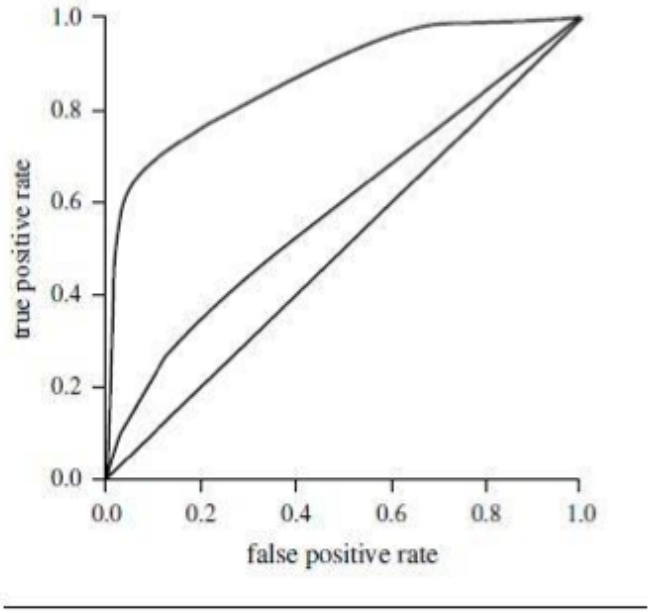


Figure 4.5: The ROC curves of two classification models.

CHAPTER -5 PRACTICAL IMPLEMENTATION

5.1.Dataset Analysis

The initial Dataset consisted of 39,644 observations with 61 features collected over a 2 years period from Jan 2013 - Jan 2015 from mashable website.
After extensive analysis it was discovered that the data of numerous anomalies, for instance:

Data Set	Website
843,330 shares 12 videos 128 videos	792 shares 0 videos 12 videos

Fig. : Comparison of Actual vs Recorded Data for an article(leaked: More Low Cost iPhone photos)

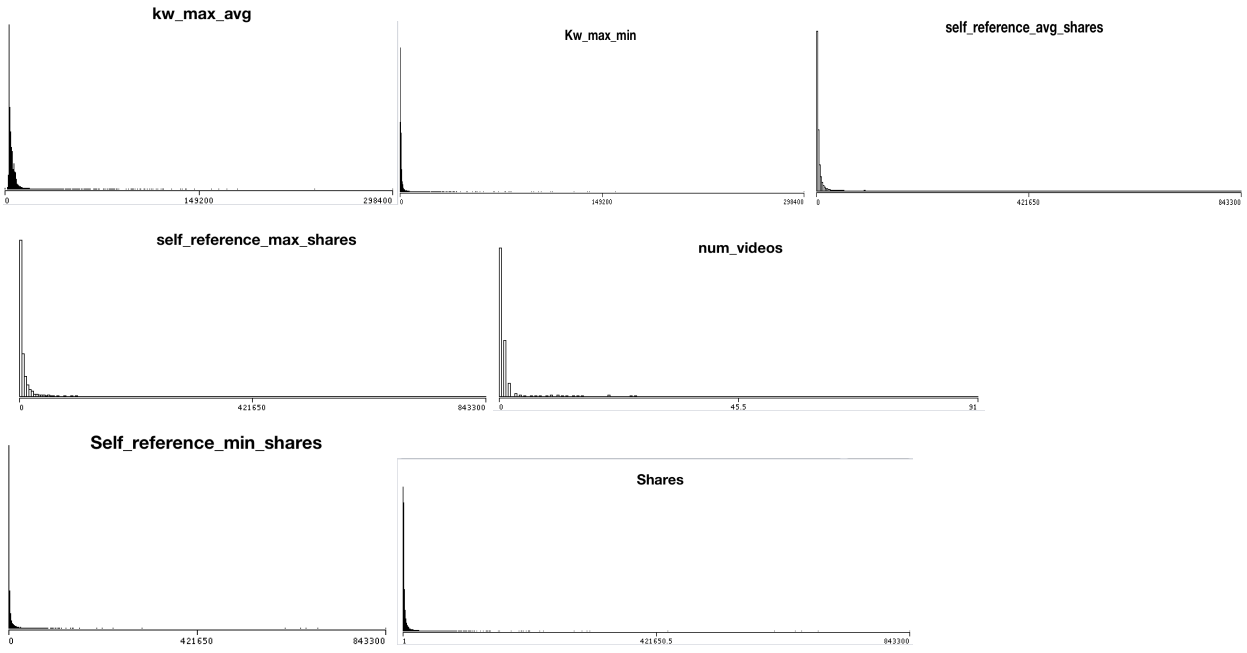
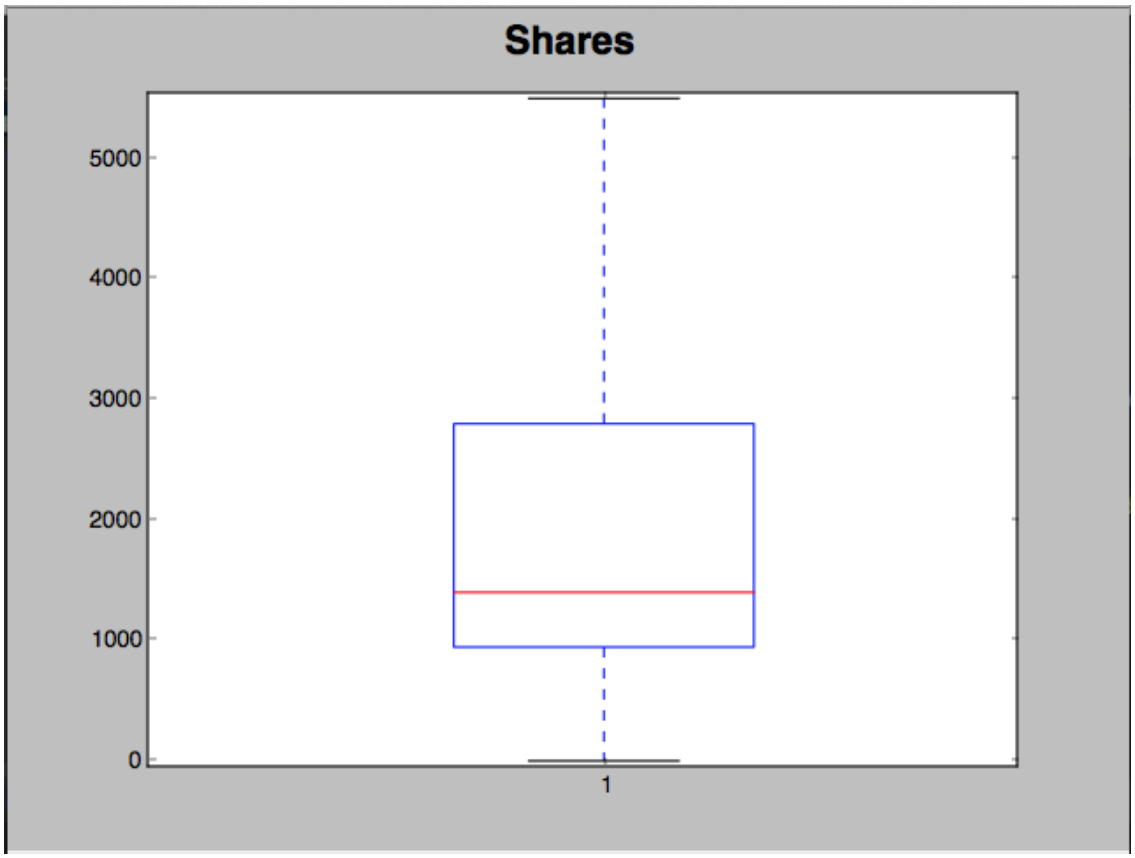


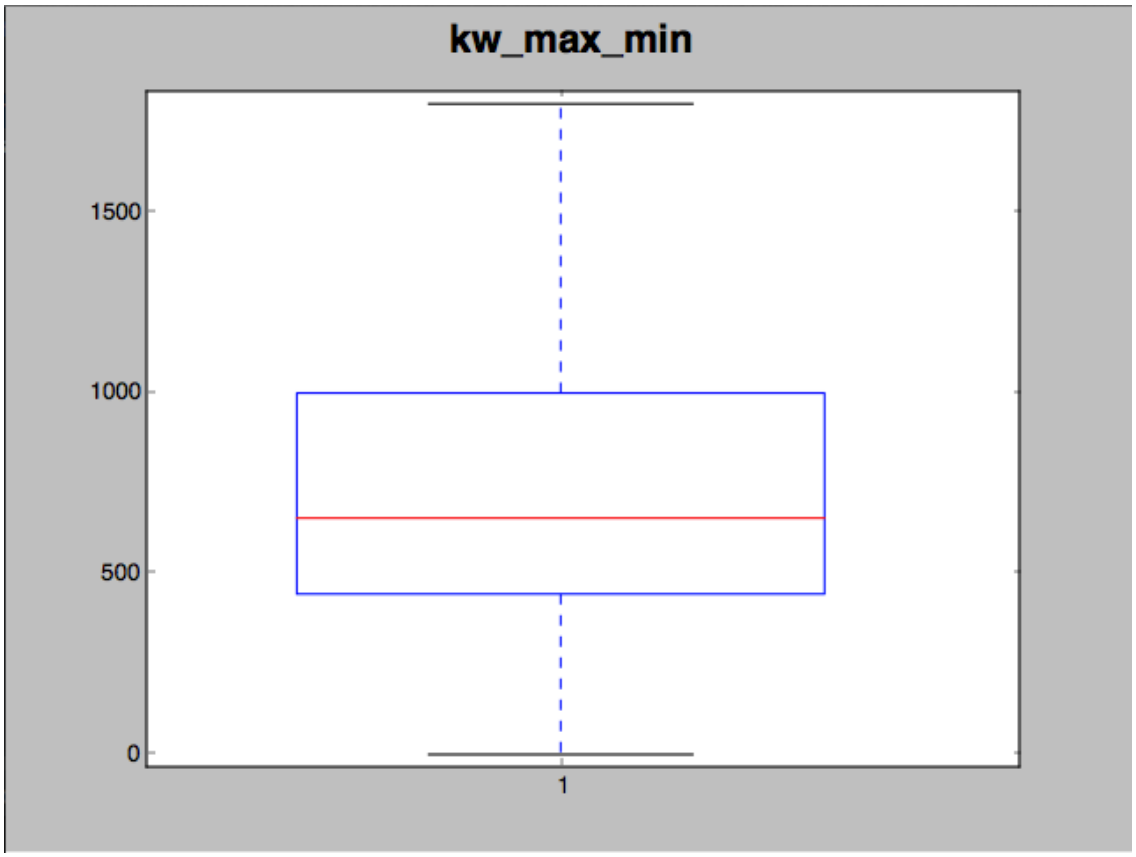
Fig: frequency Distribution histogram showing outlier observations

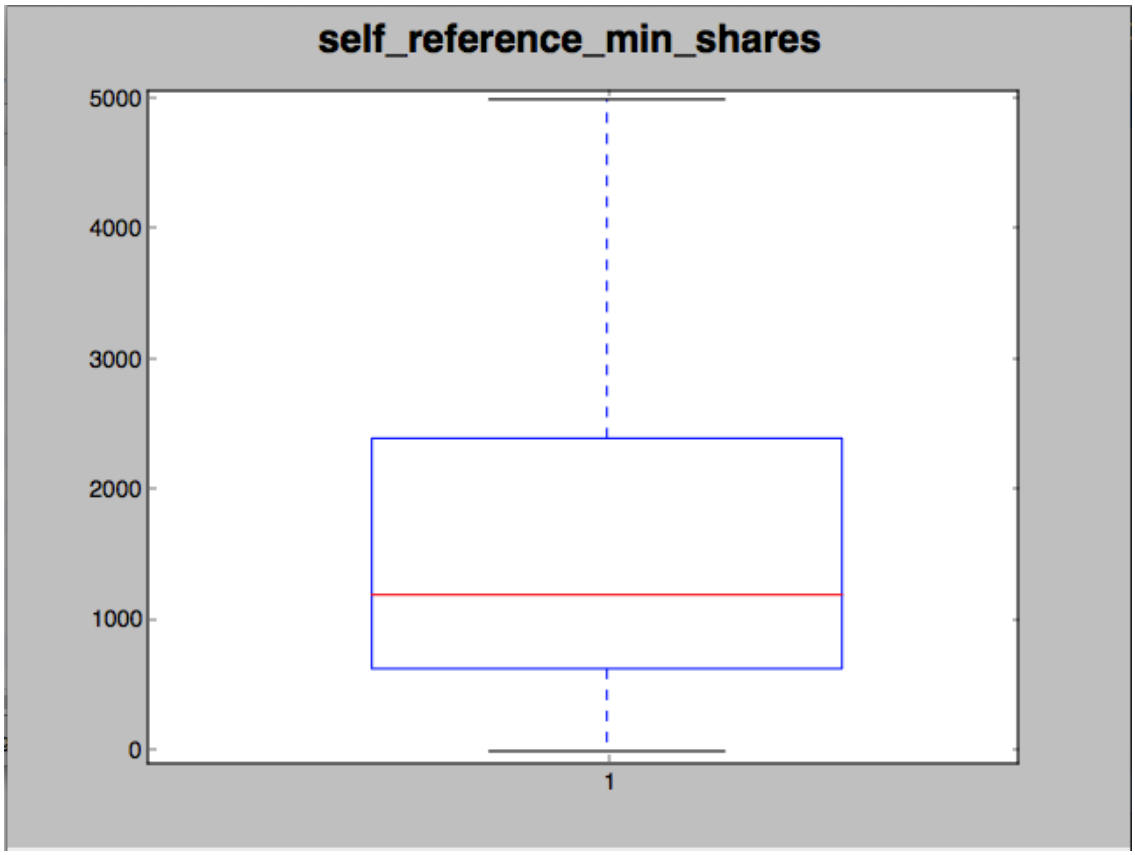
5.2 Data Pre-processing

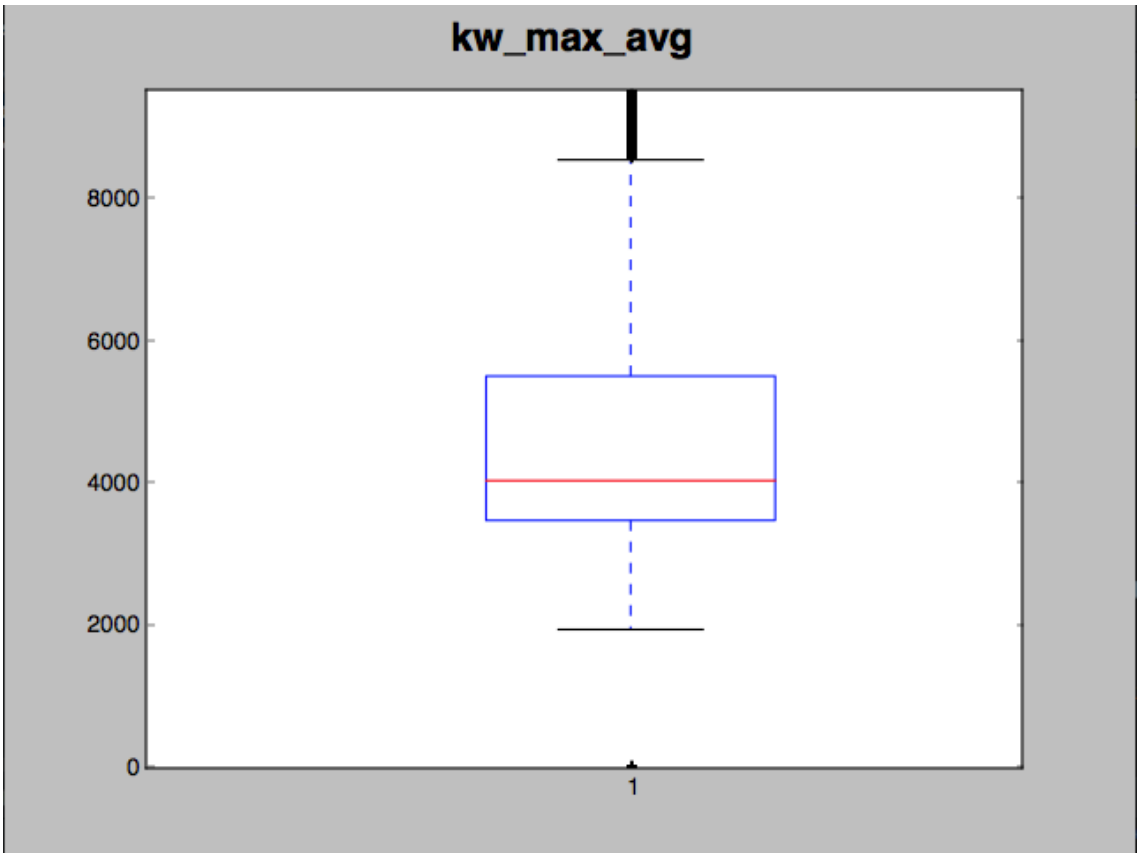
2.1 Outlier Reduction
A **Boxplot** is a convenient way of graphically depicting groups of numerical data through their quartiles. Hence the aforementioned erroneous attributes were box plotted and observations containing outliers were removed.

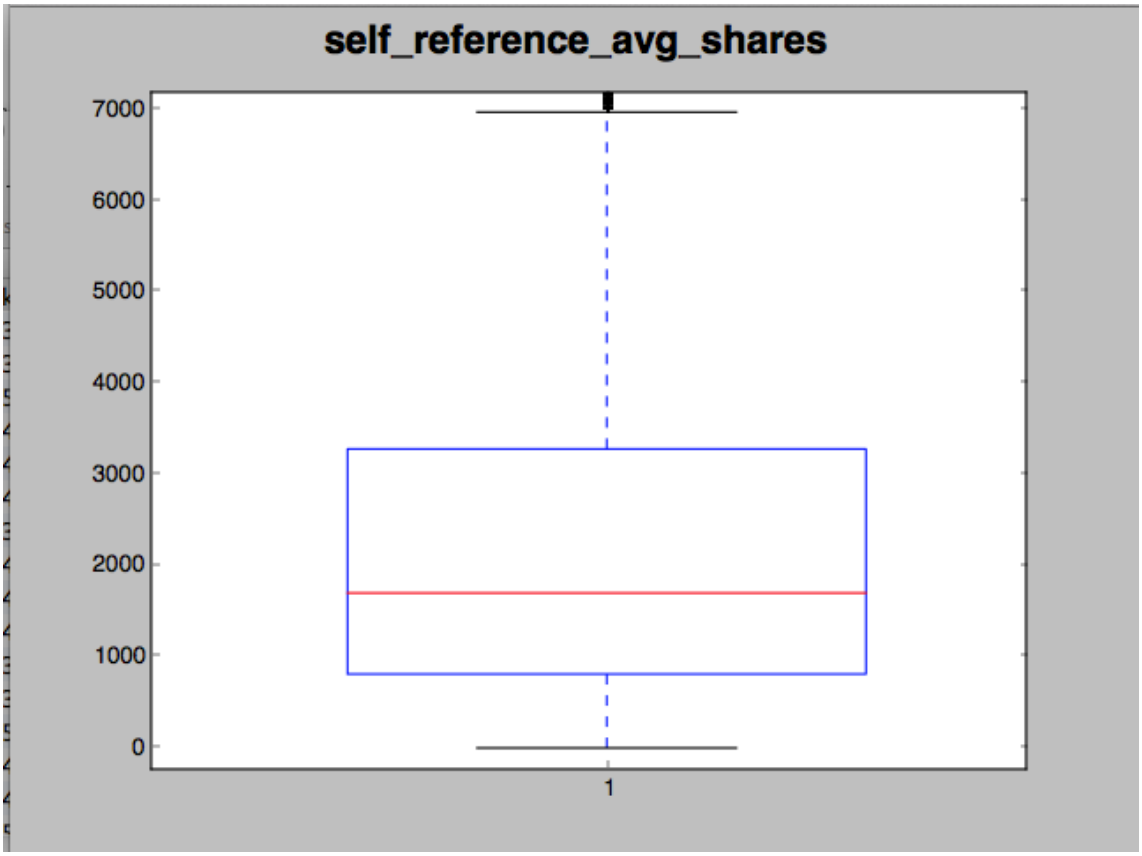
The dataset observations were reduced from 39,644 to 21,105 after removing outliers using boxplotting. The boxplots are as follows:

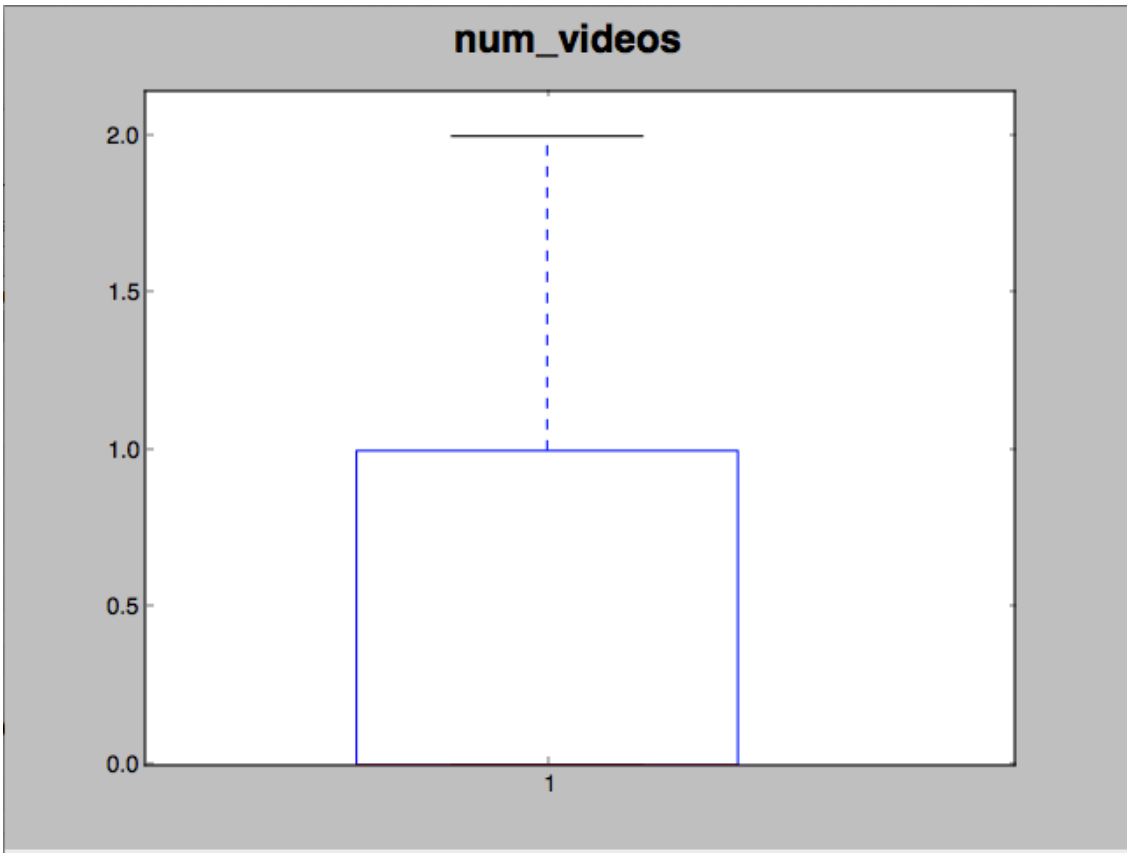


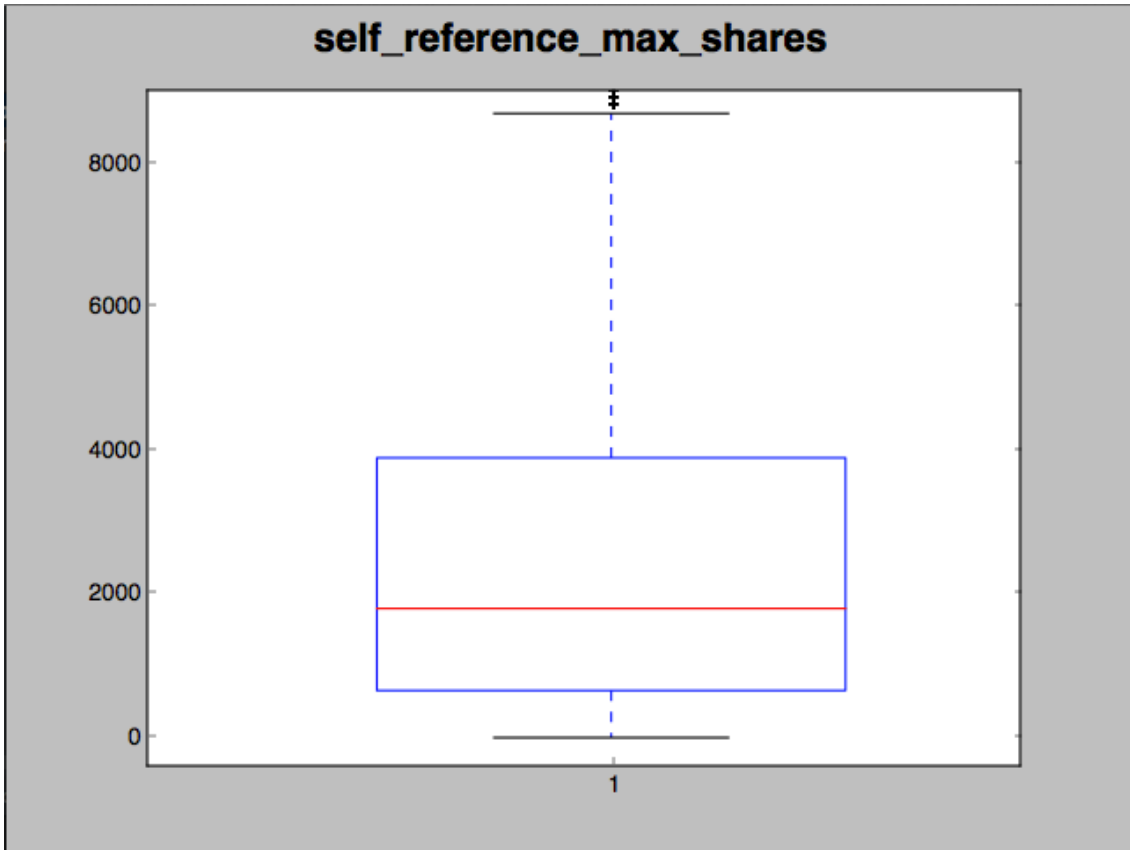

kw_max_min
1500
1000
500
0
1


self_reference_min_shares
5000
4000
3000
2000
1000
0
1

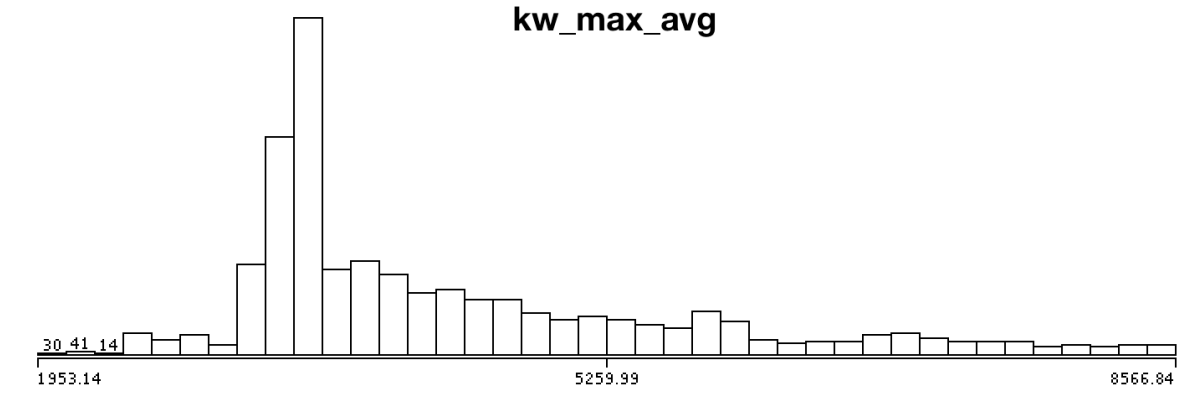

kw_max_avg
8000
6000
4000
2000
0
1

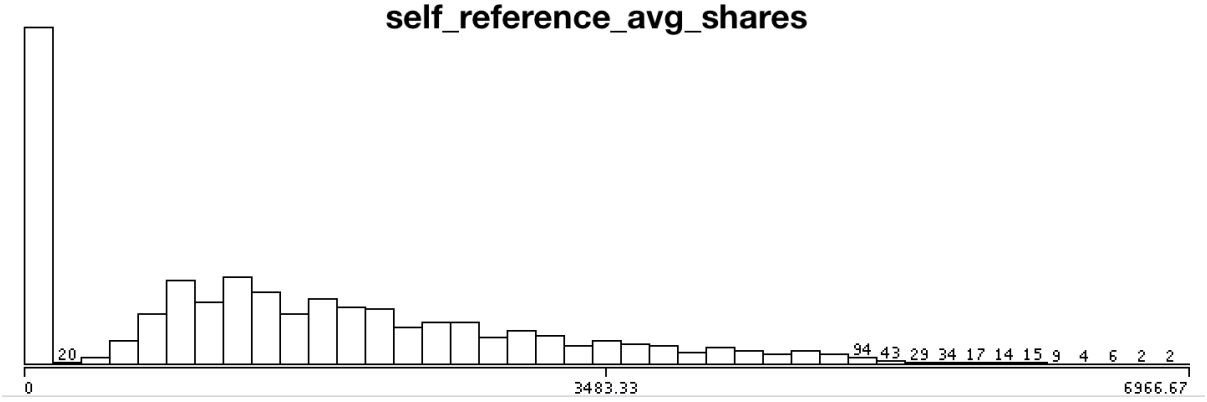

self_reference_avg_shares
7000
6000
5000
4000
3000
2000
1000
0
1

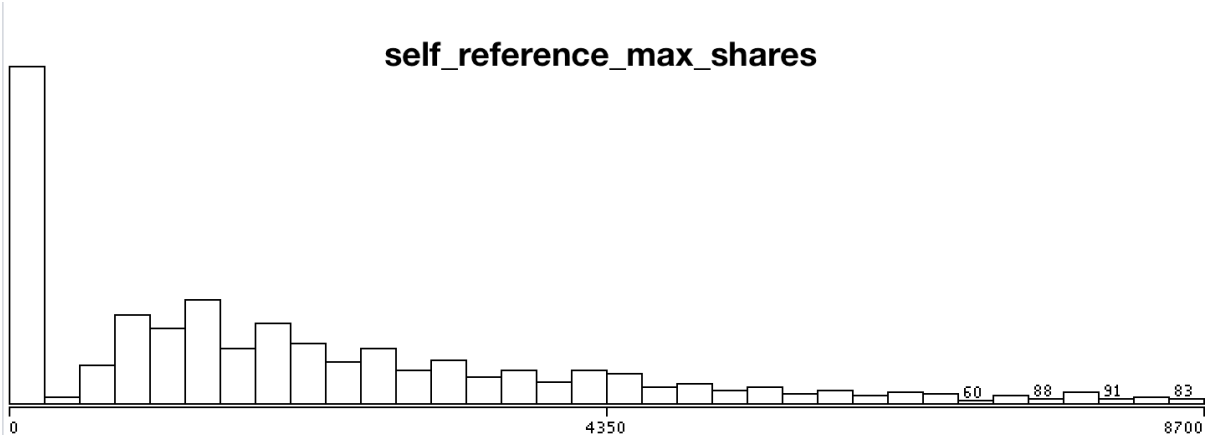

num_videos
2.0
1.5
1.0
0.5
0.0
1

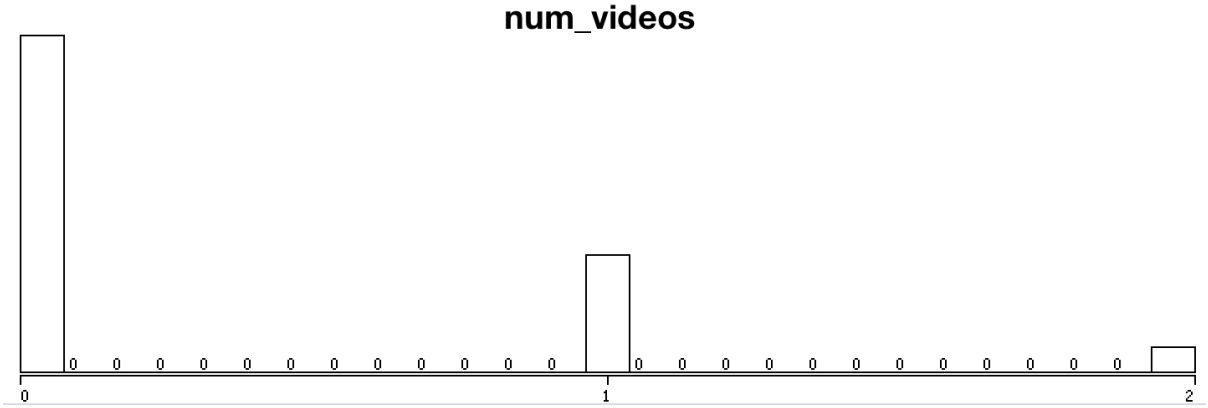

self_reference_max_shares
8000
6000
4000
2000
0
1

After removing outliers frequency distribution histograms showed massive improvements as presented below:

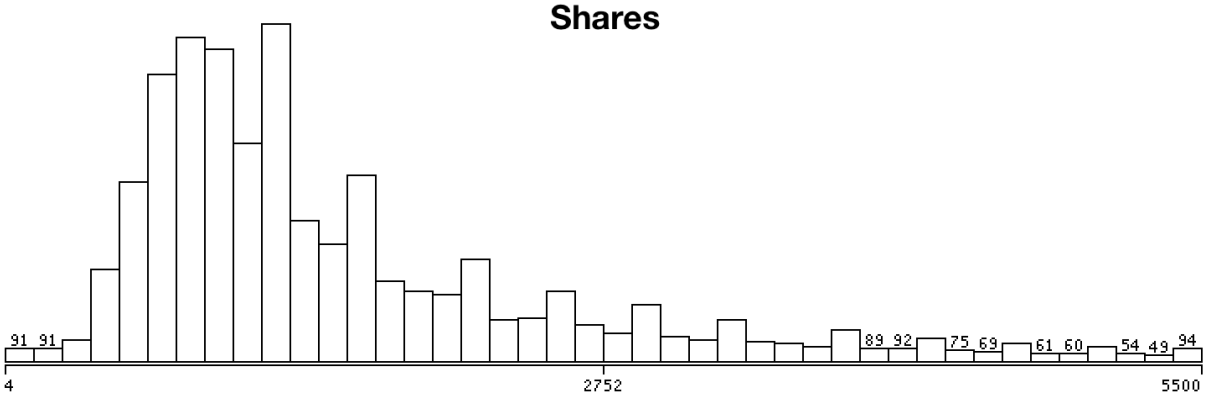







num_videos
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
1
2


self_reference_min_shares
63 99 75 61 94 55 52 72
0
2500
5000


Shares
91 91 89 92 75 69 61 60 54 49 94
4
2752
5500

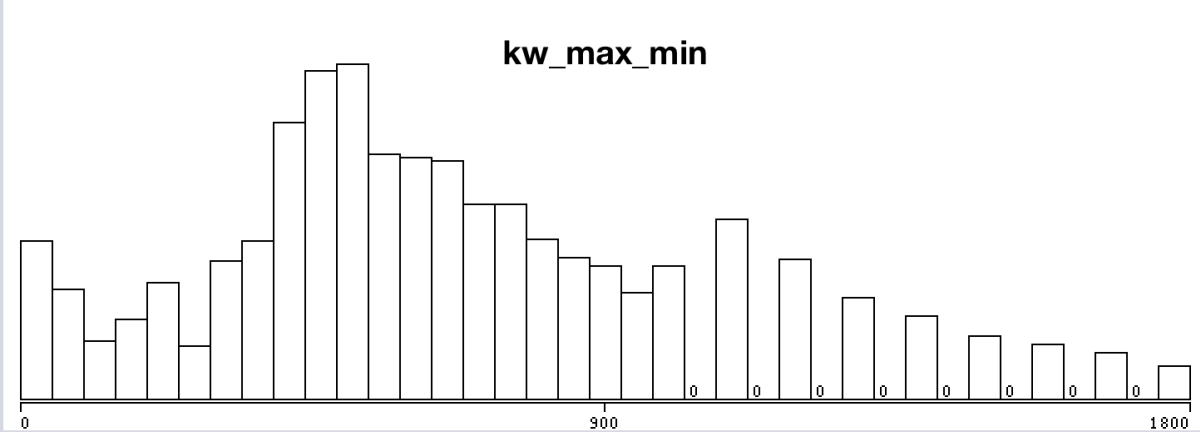


Fig: Frequency Distribution after Box Plot

5.3 Normalization

By looking at the data values, we observed that most of the attributes are greater in magnitude than the number of videos or images. When features differ by orders of magnitude, it is important to perform a feature scaling that can make gradient descent converge much more quickly.
The basic steps are:

- Subtract the mean value of each feature from the dataset.
- After subtracting the mean, additionally scale (divide) the feature values by their respective "standard deviations.

The standard deviation is a way of measuring how much variation there is in the range of values of a particular feature (most data points will lie within ±2 standard deviations of the mean); this is an alternative to taking the range of values (max-min).

$$\mu = \frac{\Sigma x}{n} \qquad \sigma = \sqrt{\frac{\Sigma(x-\mu)^2}{n}} \qquad \bar{x} = \frac{x-\mu}{\sigma}$$

Fig: Mean, Standard Deviation and Normalization Formula

The Following python script was used to normalize and store the data values.

Normalize.py

```
import numpy as np
import csv

with open('dataset.csv', 'r') as f:
    reader = csv.reader(f)
```

```
    X = list(reader)
X = np.delete(X, (0), axis=0)
X = np.delete(X, (0), axis=1)
X = np.delete(X, (-1), axis=1)
X = np.array(X).astype(np.float)

def normalize(X):
    #Normalize the values column-wise
    mean_r = []
    std_r = []

    X_norm = X

    n_c = X.shape[1]
    for i in range(1, n_c):
        m = np.mean(X[:, i])
        s = np.std(X[:, i])

        mean_r.append(m)
        std_r.append(s)
        if s==0:
            s=X[0, i]
        if s == 0:
            s = float("inf")
        X_norm[:, i] = (X_norm[:, i] - m) / s
    np.savetxt("normalizedData.csv", X_norm, delimiter=",")
    print "Saved"
    #return X_norm, mean_r, std_r

normalize(X)
```

5.4 Removal of Collinear Attributes

Collinearity is when attributes are highly correlated such that they represent the same predictor and is expressed in extreme high attribute correlations. These should be avoided at all costs because they make the prediction or classification unstable. Small changes may give very different predictors or classifiers more over eigenvalue and maximum likelihood techniques can't find solutions.

In this project weka classifier was used to remove collinear attributes. The squared coefficient of multiple correlation (Rmc^2) far all attributes that are still in the race is calculated and a threshold is set. The Attribute with the largest Rmc^2 is compared to the threshold first and deselected when larger. Subsequently the Rmc^2 for the remaining attributes is adapted and so forth.

The following thirteen attributes were removed:

1 time_delta
2 n_tokens_title
5 n_non_stop_words
6 n_non_stop_unique_tokens
9 num_imgs
12 num_keywords
13 data_channel_is_lifestyle

15 data_channel_is_bus
18 data_channel_is_world
20 kw_avg_min
22 kw_max_max
23 kw_avg_max
29 self_refence_avg_sharess
31 weekday_is_tuesday
36 weekday_is_sunday
37 is_weekend
41 lda_03
46 global_rate_negative_words

CHAPTER-6 Model Selection and Evaluation

6.1 Multivariate Linear Regression

In multivariate Linear Regression, the aim is to predict the depend variable(number_of_shares) by determining set of parameters (theta). To determine theta, we employed gradient descent to find the point of minima in the hyper-dimensional space.

The python implementation is provided hereunder:

Code:

```
import matplotlib.pyplot as plt
import numpy as np
import csv
from sklearn.model_selection import KFold


with open('normalizedDataWithShares.csv', 'r') as f:
  reader = csv.reader(f)
  X = list(reader)

X =  np.asarray(X)
Y = X[:, -1]  #last column (shares)

X = np.delete(X, (-1), axis=1) #delete last column from x

X = np.array(X).astype(np.float)
Y = np.array(Y).astype(np.float).reshape(-1, 1)

m, n = X.shape  #Number of Instances, Attributes


num_iters = 8000
alpha = 0.2

def compute_cost(X, Y, theta):

   # No of Training Samples
    m = Y.size

    predictions = X.dot(theta)

    errors = (predictions - Y)

    J = (1.0 / (2 * m)) * errors.T.dot(errors)
    return J


def gradient_descent(X, Y, alpha, num_iters):
    '''
    Performs gradient descent to learn theta
    by taking num_items gradient steps with learning
    rate alpha
    '''
    #Resetting Theta
    theta = np.zeros(shape=(X.shape[1], 1))


    m = Y.size
    J_history = np.zeros(shape=(num_iters, 1))

    for i in range(num_iters):

        predictions = X.dot(theta)

        theta_size = theta.size
```

```
        for it in range(theta_size):

            temp = X[:, it]
            temp.shape = (m, 1)

            errors_x1 = (predictions - Y) * temp

            theta[it][0] -= alpha * (1.0 / m) * errors_x1.sum()

        J_history[i, 0] = compute_cost(X, Y, theta)
        # print i, J_history[i, 0]
    return theta, J_history


def LinearCrossValidation(X,Y, k=10):

    kf = KFold(n_splits=k)
    k=0
    error_mae = 0
    error_mrae =0
    error_pred =0

    for train_index, test_index in kf.split(X):
        k+=1
        print "Fold", k
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        theta, J_history = gradient_descent(X_train, Y_train, alpha, num_iters)
        error_mae_fold = calculateMeanAbsoluteError(X_test, Y_test, theta)
        error_mrae_fold = calculateMeanRelativeAbsoluteError(X_test, Y_test, theta)
        error_pred_fold = calculatePred(X_test, Y_test, theta, 0.25)

        error_mae+=error_mae_fold
        error_mrae+=error_mrae_fold
        error_pred+=error_pred_fold
    print "Average Mean Absolute Error %f" % (error_mae / k)
    print "Average Mean Relative Absolute Error%f" % (error_mrae / k)
    print "Average PRED 0.25 %f" % (error_pred / k)


def calculateMeanAbsoluteError(X_test,Y_test, theta):

    dot = X_test.dot(theta)
    error = abs(Y_test - dot)
    # plt.scatter(Y_test, dot)

    totalErr = error.sum()
    # plt.show()
    print "\nMAE%f" %(totalErr/error.shape[0])
    return totalErr/error.shape[0]

def calculateMeanRelativeAbsoluteError(X_test, Y_test, theta):
    dot = X_test.dot(theta)
    error = abs(Y_test - dot)/Y_test

    totalErr = error.sum()
    print "\nMRAE:%f" %(totalErr/error.shape[0])
    return totalErr/error.shape[0]

def calculatePred(X_test,Y_test,theta,q=0.25):

    dot = X_test.dot(theta)
    error = abs((Y_test - dot)/Y_test)
    k=0

    for i in range(0,error.shape[0]):
        if(error[i][0]<=q):
            k=k+1

    predErr = float(k)/float(error.shape[0])

    print "PRED ",q, " ", predErr
    return predErr
```

```
LinearCrossValidation(X, Y)
```

Results:
For gradient descent the parameters learning rate(α) and the number of iterations were chosen as follows:

- α = 0.2
- number of iterations = 8000

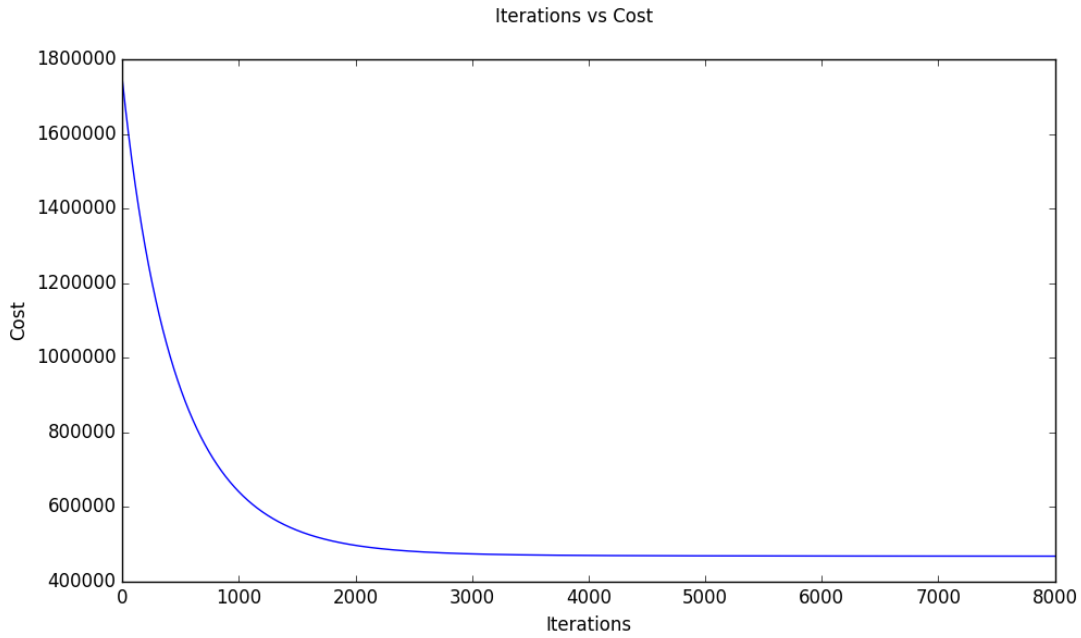


Fig: Minimization of Cost Function in Gradient Descent with Feature Selection

1. Without Feature Selection

Mean Absolute Error	712.830325
Mean Relative Absolute Error	0.681046
PRED(0.25)	0.311377

2. With feature Selection

Mean Absolute Error	722.328057
Mean Relative Absolute Error	0.704558
PRED(0.25)	0.306595

The model achieved an accuracy of:

- 31.89% - without feature selection
- 29.54% - with feature selection

Hence the performance of model degraded slightly after application of feature selection. Hence feature selection was not suitable for linear regression on the given dataset.

6.2 Logistic Regression

We used logistic regression (classification model) to improve our accuracy further and classified the instances into two categories “popular” and “unpopular” based on the median of dependent variable (number of shares).

The python implementation is provided below:

Code:

```
import numpy as np
import csv
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt

with open('normalizedDataWithShares.csv', 'r') as f:
  reader = csv.reader(f)
  X = list(reader)

X =  np.asarray(X)
Y = X[:, -1]  #last column (shares)
X = np.delete(X, (-1), axis=1) #delete last column from x
X = np.array(X).astype(np.float)
Y = np.array(Y).astype(np.float).reshape(-1, 1)
m, n = X.shape  #Number of Instances, Attributes

num_iters = 4000
alpha = 0.003


def hypothesis(X, theta):
    predictions = X.dot(theta)
    return 1/(1+np.e**(-1*predictions))

def cost_function(X, Y, theta):

    predictions = hypothesis(X, theta)

   # No of Training Samples
    m = Y.size

    cost = (Y).T.dot(np.log(predictions)) + (1-Y).T.dot(np.log(1-predictions))

    J = (1.0 / m) * cost
    return J


def gradient_descent(X, Y, alpha, num_iters):
    '''
    Performs gradient descent to learn theta
    by taking num_items gradient steps with learning
    rate alpha
    '''
    #Resetting Theta
    theta = np.zeros(shape=(X.shape[1], 1))


    m = Y.size
    J_history = np.zeros(shape=(num_iters, 1))

    for i in range(num_iters):

        predictions = hypothesis(X, theta)

        theta_size = theta.size

        for it in range(theta_size):

            temp = X[:, it]
            temp.shape = (m, 1)
```

```
            errors_x1 = (predictions - Y) * temp

            theta[it][0] -= alpha * (1.0 / m) * errors_x1.sum()

        J_history[i, 0] = cost_function(X, Y, theta)

    plt.xlabel("Iterations")
    plt.ylabel("Cost")
    plt.suptitle("Iterations vs Cost")
    plt.plot(J_history)
    plt.show()
    return theta, J_history


def LogisticCrossValidation(X,Y, k=10):

    # Cross Validation Coeff
    # Constants for confusion MAtrix
    true_p = 0
    true_n = 0
    false_p = 0
    false_n = 0

    kf = KFold(n_splits=k)
    mae = 0
    fold = 1
    for train_index, test_index in kf.split(X):
        print("\n\nFold %f" %fold)
        fold +=1
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]
        theta, J_history = gradient_descent(X_train, Y_train, alpha, num_iters)

        tp, fp, fn, tn = calculateConfusionMatrix(X_test, Y_test, theta)
        mae_fold = calculateMeanAbsoluteError(X_test.dot(theta), Y_test)
        true_p+=tp
        true_n+=tn
        false_p+=fp
        false_n+=fn
        mae +=mae_fold
    print "Average Confusion Matrix"
    print true_p, false_p
    print false_n, true_n
    displayResultSummary(true_p, false_p, true_n, false_n, mae/k)

def displayResultSummary(tp, fp, tn, fn, mae):
    total = tp+fp+tn+fn
    print "Correctly Classified Instances:", (tp+tn),"\t",((float(tp+tn)/total)*100), "%"
    print "Incorrectly Classified Instances:", (fp + fn), "\t", ((float(fp+fn) / total) * 100), "%"
    print "Mean Absolute Error:\t",mae
    print "TP Rate/Recall", tp/float(tp+fn)
    print "FP Rate", fp/float(fp+tn)
    print "Precision", tp/float(tp+fp)

def calculateMeanAbsoluteError(Y_proba,Y_test):

    error = abs(Y_test - Y_proba)
    # plt.scatter(Y_test, dot)

    totalErr = error.sum()
    # plt.show()
    print "\nMean Absolute Error%f" %(totalErr/error.shape[0])
    return totalErr/error.shape[0]

def calculateConfusionMatrix(X_test,Y_test, theta):
    true_positive = 0
    true_negative = 0
    false_positive = 0
    false_negative = 0
    dot = X_test.dot(theta)

    for i in range(0, dot.shape[0]):
```

```
        if(dot[i][0]>=0.5):
            if(Y_test[i][0]>=0.5):
                true_positive+=1
            else:
                false_positive+=1

        else:
            if(Y_test[i][0]>=0.5):
                false_negative +=1
            else:
                true_negative+=1

    print "Confusion Matrix"
    print true_positive, false_positive
    print false_negative, true_negative
    print "Correct: ", (float)(true_negative + true_positive) / (float)(false_negative + true_negative
+ true_positive + false_positive)
    return true_positive, false_positive, false_negative, true_negative

LogisticCrossValidation(X, Y)
```

Results:
For gradient descent the parameters learning rate(α) and the number of iterations were chosen as follows:

- α = 0.003
- number of iterations = 4000

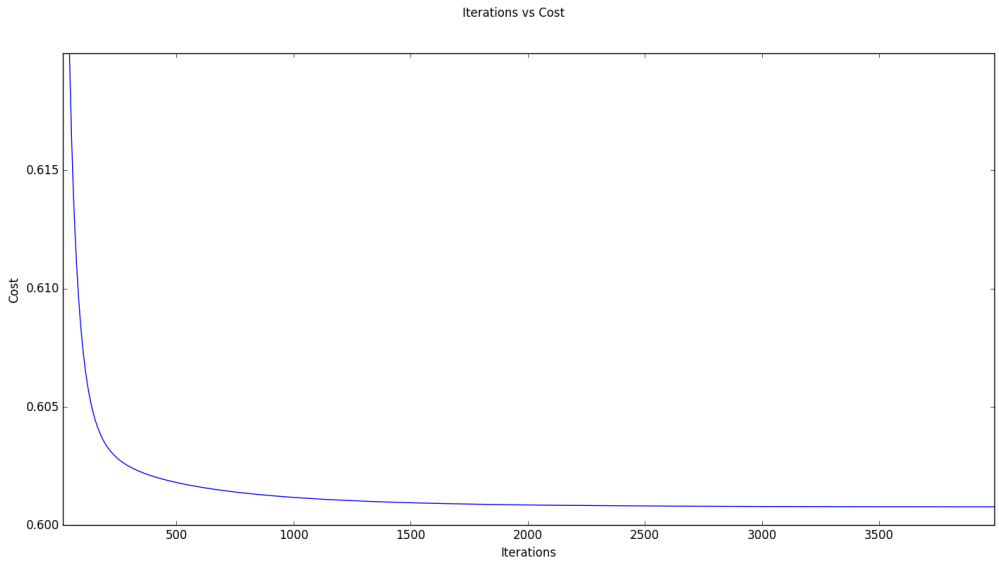


Fig: Minimization of Cost Function in Gradient Descent without Feature Selection

The **receiver operating characteristic** (**ROC**), or **ROC curve** graphical plot illustrates the performance of the logistic binary classifier system as its discrimination threshold is varied. The ROC curves for data with and without feature selection is shown below:

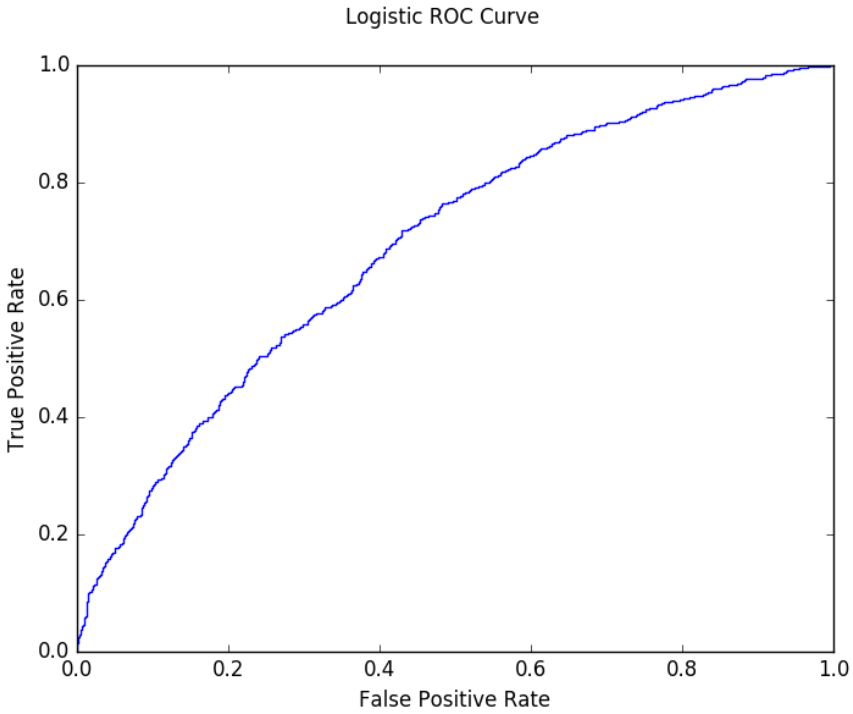


Fig: ROC Curve for Logistic Classifier without Feature Selection

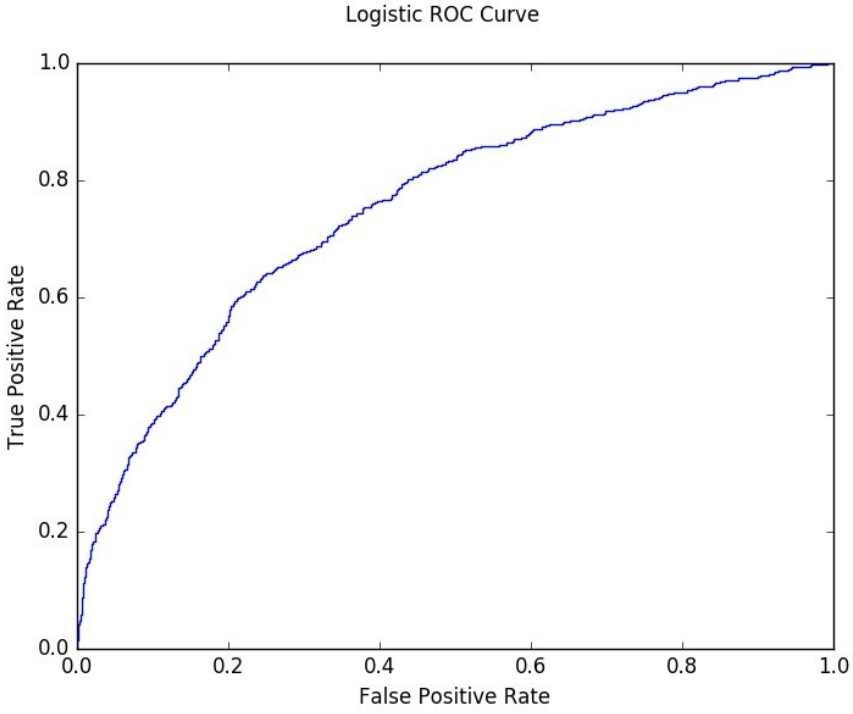


Fig: ROC Curve for Logistic Classifier with Feature Selection

1. Without Feature Selection

Mean Absolute Error 0.615069
Confusion Matrix

		Actual Values	
		1	0
Classified as	1	1616	732
	0	6579	12176

Correctly Classified Instances:	13792	**65.3556366393 %**
Incorrectly Classified Instances:	7311	34.6443633607 %
Mean Absolute Error:		0.837596492743
TP Rate/Recall:		0.197193410616
FP Rate:		0.0567090176635
Precision:		0.688245315162
AUC:		0.697801976868

2. With feature Selection:

Mean Absolute Error 0.909651
Average Confusion Matrix

		Actual Values	
		1	0
Classified as	1	1139	708
	0	6274	12982

Correctly Classified Instances:	14121	**66.9146566839** %
Incorrectly Classified Instances:	6982	33.0853433161 %
Mean Absolute Error:	0.873365774302	
TP Rate/Recall	0.153648995009	
FP Rate	0.0517165814463	
Precision	0.616675690309	
AUC	0.753054188435	

The model achieved an accuracy of:

- 65.35% - without feature selection
- 66.91% - with feature selection

The accuracy of this model displayed significant improvement as compared to the Linear Model. Also the performance of the model increased slightly after application of feature selection. Hence feature selection was suitable for binary logistic regression on the given dataset.

6.3 Support Vector Classification

We used Support vector machine (**SVM**), a supervised learning model with associated learning algorithms to analyze data and classify the instances into two categories "popular" and "unpopular" based on the median of dependent variable (number of shares).

The python implementation is provided below:

Code:

```
import matplotlib.pyplot as plt
import numpy as np
import csv
from sklearn import svm
from sklearn.model_selection import KFold


#      open the csv file for reading data
#      X: stores the whole csv file
with open('normalizedDataWithSharesReducedAttributes.csv', 'r') as f:
    reader = csv.reader(f)
    X = list(reader)

#      dividing X into X and Y where:
#      X:        all the attributes in normailized form
#      Y:        predicted shares in 0 and 1 form
X = np.asarray(X)
Y = X[:, -1]  # last column (shares)

X = np.delete(X, (-1), axis=1)  # delete last column from x

X = np.array(X).astype(np.float)        # convert all values to float
Y = np.array(Y).astype(np.float).reshape(-1, 1)

#      m:        Number of Instances
#      n:        Number of Attributes
m, n = X.shape


# Cross Validaton code that contains
# InputParameters:
#      X:        attribute normalized variables
#      Y:        shares
#      k:        folds in cross validation by default 10
#      Cost:     Cost for SVC classifier
# Return:       None
def SVCCrossValidation(X, Y,Cost=1.0, k=10):

    # Constants for confusion MAtrix
    true_p = 0
    true_n = 0
    false_p = 0
    false_n = 0
    mae = 0

    # KFold library function(Sklearn) for calculating train and test indices
    kf = KFold(n_splits=k)
    fold = 1
```

```
    for train_index, test_index in kf.split(X):
        print("\n\nFold %f" % fold)
        fold += 1
        # X_train:      train attributes data matrix
        # Y_train:      train shares matrx
        # X_test:       test attributes data matrix
        # Y_test:       test shares data matrix
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]

        Y_train = np.array(Y_train)
        print Y_train.shape
        # SVM classifier with paramters
        # Kernel:Rbf
        clf = svm.SVC(probability=True, C=Cost)
        clf.fit(X_train, Y_train)


        # Y_calc as the predicted Y for the test Data by the SVC Classifier
        Y_calc = clf.predict(X_test)
        Y_proba = clf.predict_proba(X_test)
        Y_proba = np.delete(Y_proba, (-1), axis=1)
        print Y_proba.shape

        # Confusion Matrix Calculation
        tp, fp, fn, tn = calculateConfusionMatrix(Y_calc, Y_test)
        mae_fold=calculateMeanAbsoluteError(Y_proba, Y_test)

        mae+=mae_fold
        true_p += tp
        true_n += tn
        false_p += fp
        false_n += fn

    print "Average Confusion Matrix"
    print true_p, false_p
    print false_n, true_n

    displayResultSummary(true_p, false_p, true_n, false_n, mae/k)


def displayResultSummary(tp, fp, tn, fn, mae):
    total = tp+fp+tn+fn
    print "Correctly Classified Instances:", (tp+tn),"\t",((float(tp+tn)/total)*100), "%"
    print "Incorrectly Classified Instances:", (fp + fn), "\t", ((float(fp+fn) / total) * 100), "%"
    print "Mean Absolute Error:\t",mae
    print "TP Rate/Recall", tp/float(tp+fn)
    print "FP Rate", fp/float(fp+tn)
    print "Precision", tp/float(tp+fp)
    print "Recall"

def calculateMeanAbsoluteError(Y_proba,Y_test):

    error = abs(Y_test - Y_proba)
    # plt.scatter(Y_test, dot)

    totalErr = error.sum()
    # plt.show()
    print "\nMean Absolute Error%f" %(totalErr/error.shape[0])
    return totalErr/error.shape[0]


# Calculation of confusion matrix
# Input paramters:
# Y_calc:       Calculated value of share group(0/1)
# Y_test:       Actual Value of test group(0/1)
# Output Parameters:
# confusion matrix Paramters
def calculateConfusionMatrix(Y_calc, Y_test):
    true_positive = 0
    true_negative = 0
    false_positive = 0
    false_negative = 0
```

```
    for i in range(0, Y_calc.shape[0]):
        if (Y_calc[i] >= 0.5):
            if (Y_test[i][0] >= 0.5):
                true_positive += 1
            else:
                false_positive += 1

        else:
            if (Y_test[i][0] >= 0.5):
                false_negative += 1
            else:
                true_negative += 1

    print "Confusion Matrix"
    print true_positive, false_positive
    print false_negative, true_negative
    print "Correct: ", (float)(true_negative + true_positive) / (float)(
        false_negative + true_negative + true_positive + false_positive)
    return true_positive, false_positive, false_negative, true_negative


SVCCrossValidation(X, Y)
```

Results:

The **receiver operating characteristic** (**ROC**), or **ROC curve** graphical plot illustrates the performance of the SVM binary classifier system as its discrimination threshold is varied. The ROC curves for data with and without feature selection is shown below:

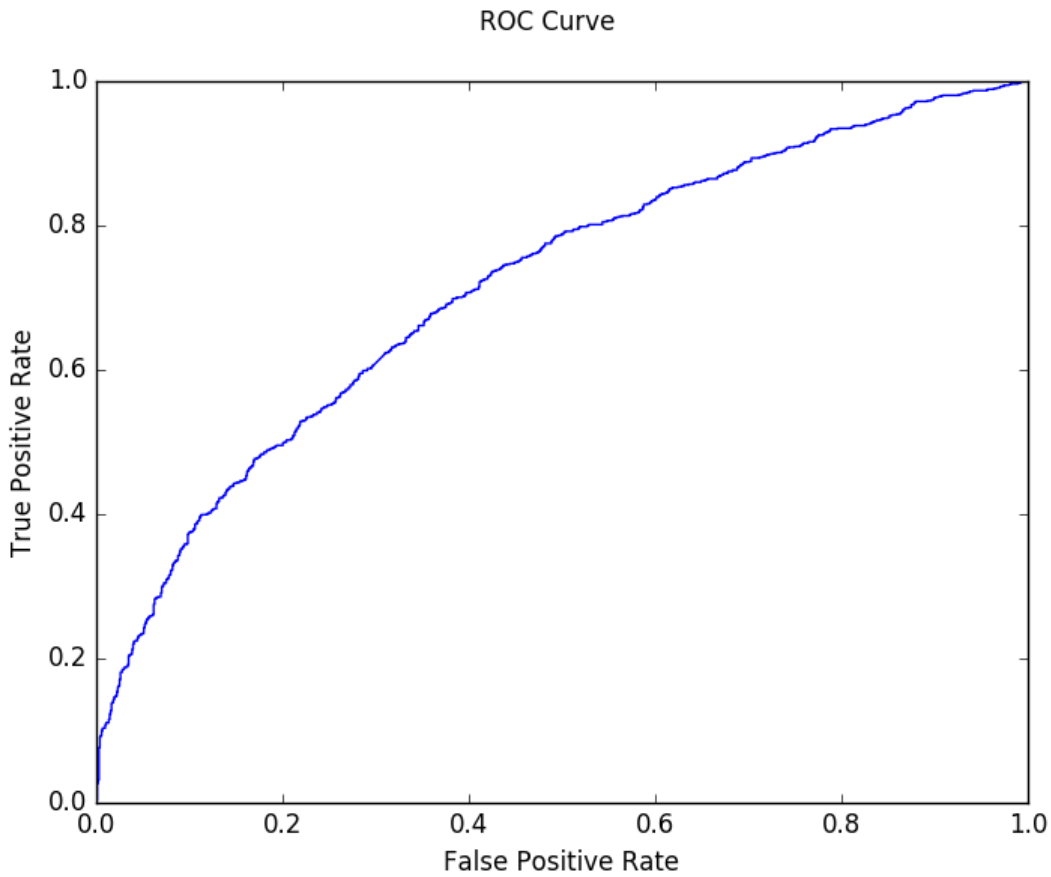


Fig: ROC Curve for SVM Classifier with Feature Selection

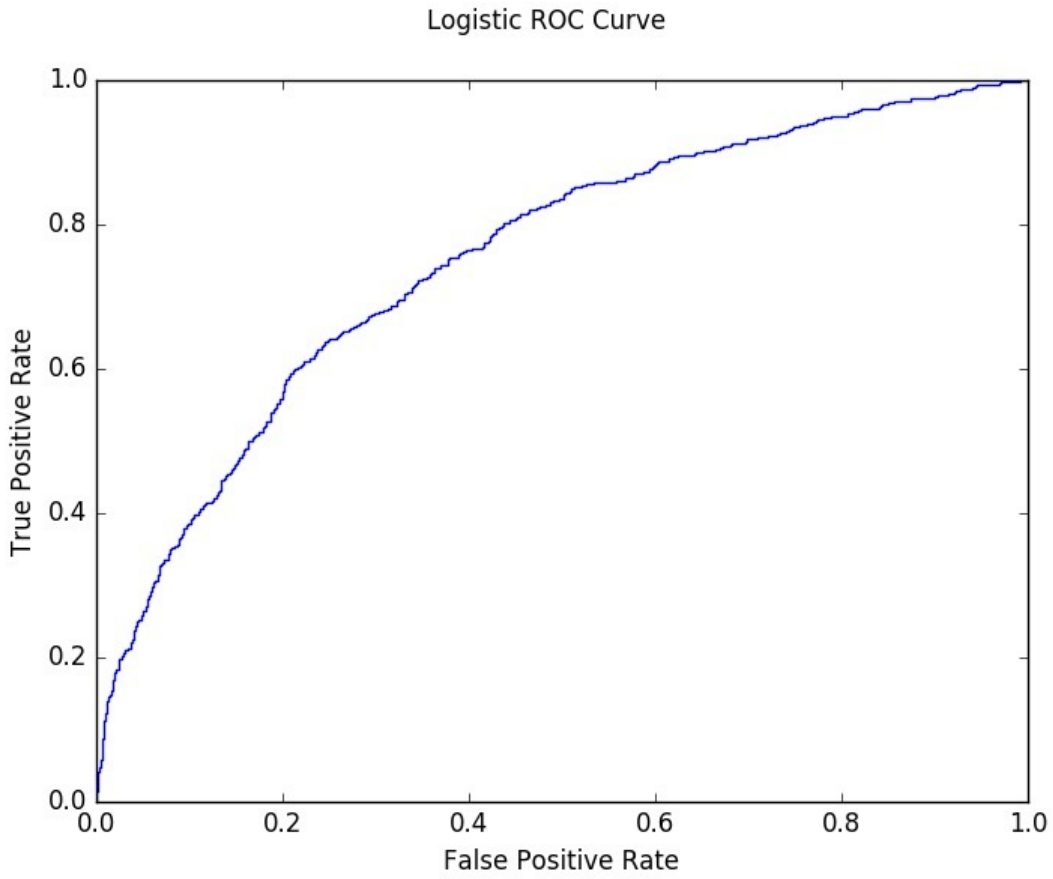


Fig: ROC Curve for SVM Classifier without Feature Selection

1. Without Reduced Attributes

Mean Absolute Error 0.568637

Average Confusion Matrix

		Actual Values	
		1	0
Classified as	1	3071	1751
	0	5124	11157

Correctly Classified Instances 14228 **67.4216935981 %**

Incorrectly Classified Instances 6875 32.5783064019 %

Mean Absolute Error 0.583096599995

TP Rate/Recall 0.374740695546

FP Rate 0.135652308646

Precision 0.636872666943

AUC 0.694588372063

2. With Reduced Attributes

Mean Absolute Error 0.601971

Average Confusion Matrix

		Actual Values	
		1	0
Classified as	1	2133	1198
	0	5280	12492

Correctly Classified Instances: 14625 **69.3029427096 %**

Incorrectly Classified Instances: 6478 30.6970572904 %

Mean Absolute Error: 0.593544224221

TP Rate/Recall 0.287737757993

FP Rate 0.0875091307524

Precision 0.640348243771

AUC 0.716094383573

The model achieved an accuracy of:

- 67.42% - without feature selection
- 69.30% - with feature selection

The accuracy of this model displayed improvement as compared to the other models. Also the performance of the model increased slightly after application of feature selection. Hence feature selection was suitable for Support Vector Classifier on the given dataset.

Chapter-7 RESULT and CONCLUSION

Chapter-8 REFERENCES

[1] Tatar, Alexandru, et al. "Predicting the popularity of online articles based on user comments." Proceedings of the International Conference on Web Intelligence, Mining and Semantics. ACM, 2011.

[2] "Predicting the Popularity of Social News Posts." 2013 cs229 projects. Joe Maguire Scott Michelson.

[3] Hensinger, Elena, Ilias Flaounas, and Nello Cristianini. "Modelling and predicting news popularity." Pattern Anal- ysis and Applications 16.4 (2013): 623-635.

[4] K. Fernandes, P. Vinagre and P. Cortez. A Proactive In- telligent Decision Support System for Predicting the Pop- ularity of Online News. *Proceedings of the 17th EPIA 2015 - Portuguese Conference on Artificial Intelligence*, September, Coimbra, Portugal.

[5] Chang, Chih-Chung, and Chih-Jen Lin. "LIBSVM: A li- brary for support vector machines." *ACM Transactions on Intelligent Systems and Technology (TIST)* 2.3 (2011): 27.

[6] James, Gareth, et al. *An introduction to statistical learning*. New York: springer, 2013.

[7] K. Fernandes, P. Vinagre and P. Cortez. *A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News*. Proceedings of the 17th EPIA 2015 - Portuguese Conference on Artificial Intelligence, September, Coimbra, Portugal.

Chapter-9 FUTURE SCOPE

As is seen from the result, no algorithm can reach 70% accuracy given the data set we have, even though they are state-of-the-art. To improve accuracy, there is little room in model selection but much room in feature selection. In the pre-processing round, 59 features were extracted from news articles, and our later work is based on these features. However, the content of news articles hasn't been fully explored. Some features are related to the content, such as LDA topics (feature #39 - #43), which are convenient to use for learning, but reflect only a small portion of information about the content.

In the future, we could directly treat all the words in an article as additional features, and then apply machine learning algorithms like Naive Bayes and SVM. In this way, what the article really talks about is taken into account, and this approach should improve the accuracy of prediction if combined with our current work.

XXX(Leave)Chapter-5 Multi Instance Learning

5.1 Introduction

In the standard supervised learning task, we learn a classifier based on a training set of feature vectors, where each feature vector has an associated class label. In the Multiple Instance Learning (MIL) task we learn a classifier based on a training set of bags, where each bag contains multiple feature vectors (called instances in the MIL terminology). In this setting, each bag has an associated label, but we do not know the labels of the individual instances that conform the bag. Furthermore, not all the instances are necessarily relevant, i.e., there might be instances inside one bag that do not convey any information about its class, or that are more related to other classes of bags, providing confusing information.

In many fields, we find problems that are most naturally formulated using the multiple instance learning setting. This is the case of drug discovery (pharmacy), classification of text documents (information retrieval), classification of images (computer vision), speaker identification (signal processing) and bankruptcy prediction (economy), to mention a few fields that make use of this framework (see section 2 for a more detailed discussion about real examples). This makes the MIL problem an important topic in the machine learning community, where many methods have been published in the last years. Despite this fact, there is a lack of surveys or analytical studies that compare the performance of the different families of MIL algorithms

The overview of some of the approaches to deal with the advent growth of multi-image multi-label methods are given in this section. Dietterich et al [4] proposed multi instance learning

(MIL) for drug prediction problem in which the objects to be classified were chemical molecules. The system had to classify that whether given chemical molecule is a good drug or not. Andrews et al [5] demonstrated two approaches to modify Support Vector Machines (SVM), mi-SVM and MI-SVM for instance-level classification and bag-level classification respectively. In order to solve multi instance learning, traditional neural networks were used in [3,6]. However they did not apply deep learning methodology nor did they use it for computer vision applications.

Maron and Lozano-Pérez [7] presented the Diverse density algorithm for solving MIL problem. In this approach a point is found in feature space such that it is near to instances of positive bags and at the same time is at a greater distance from instances of negative bags. Hence, the learning phase involves searching for an optimal point which is the one that has the maximum diverse density which is measured in terms of how many different instances of positive bags are located near that point, and how many negative instances are far away from that point. The algorithm has been applied to stock selection and content based image retrieval [8].

Deep learning architectures comprises various layers consisting of feature detectors. Simple and local features are detected by lower layers which are then fed to higher layers that calculate more complex features [9, 10, 11, 12]. Convolution neural networks (CNN) have been applied for a variety of computer vision applications including object recognition [10] and video classification [12]. Xu et al. [13] presented deep learning method to compute features for multi-instance learning in medical imaging. Song et al. [14] applied CNN features for weakly supervised object localization.

Wang et al [15] employed k-nearest neighbour (kNN) algorithm for MIL framework by utilizing a bag-level distance metric, the minimum Hausdorff distance that is defined as the shortest distance between any two bag instances. Two multi instance learning algorithms were presented: citation-kNN which predicts the label of a bag based not only on the nearest neighbours (references) but also on the bags that consider the concerned bas as a neighbour (citers), which proved to be more efficient than the kNN based only on references and the Bayesian method, which estimates the labels of a bag based only on the abels of its neighbours. Chevaleyre et al [16] applied decision trees and decision rules in the MIL framework. A multi instance version of decision tree algorithm ID3 called ID3-MI and that of rule learning algorithm RIPPER called RIPPER-MI was introduced. The growth of the decision tree was based on the information gain of a feature with respect to set of instances that was given by a multiple instance coverage function, which was related to the entropy of the instances, where the entropy was given by multiple instance entropy function.

Amores [1] presented an analysis of different multiple instance learning methods that comprised of both review and comparison of different paradigms which included instance space paradigm,

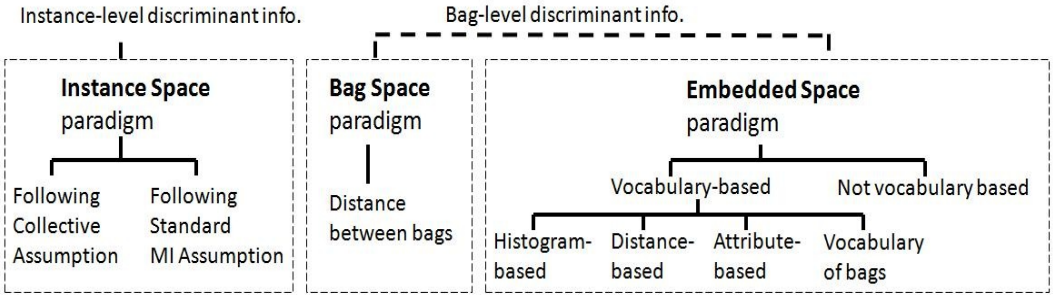
bag space paradigm and embedded space paradigm. In the embedded space paradigm we utilize a vocabulary-based mapping, in bag space paradigm we consider the similarity between all instances and in instance space paradigm we infer an instance based classifier from the training data. Under particular constraints, the bag space paradigm becomes computationally more expensive and therefore vocabulary based ES methods are better suited for certain type of data.

5.2 Overview of Paradigms

A bag is a set X = {~x1, . . . , ~xN}, where the elements ~xi are feature vectors called instances in the MIC terminology, and the cardinality N can vary across the bags. All the instances ~xi live in 3 a d-dimensional feature space, ~xi ∈ R d , called instance space.

The objective of the MIC problem is to learn a model, at training time, that can be used to predict the class labels of unseen bags. In this work, we only consider the binary classification problem, where a bag X can be either positive or negative. Our objective is to estimate a classification function F(X) ∈ [0, 1] that provides the likelihood that X is positive. In order to learn such a function, we are given a training set with M bags and their corresponding labels, T = {(X1, y1), . . . , (XM, yM)}, where yi ∈ {0, 1} is the label of Xi (yi = 0 if Xi is negative, and yi = 1 if it is positive).

In addition to the bag-level classification function F(X), many methods try to learn an instance level classification function f(~xi) that operates directly on the instances ~xi . Throughout this work we will use uppercase to refer to bags X and to the bag-level classifier F, and we will use lowercase to refer to instances ~x and to the instance-level classifier f.



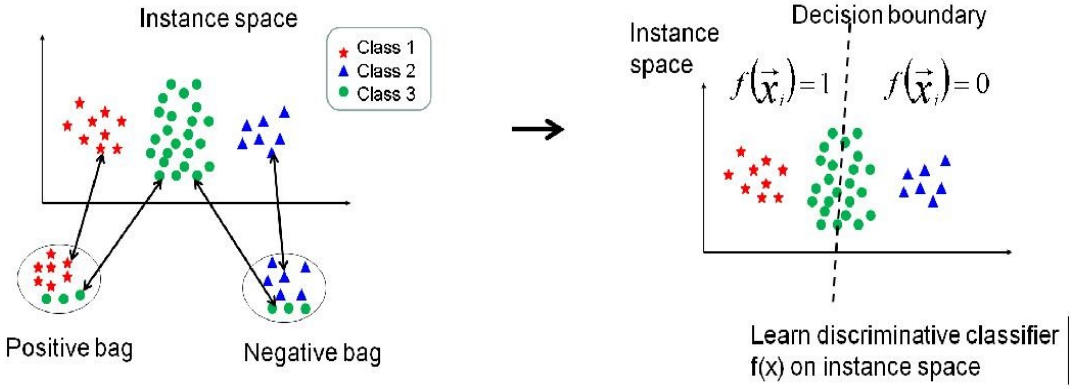
5.2.1 Instance Space Paradigm

In the Instance Space (IS) paradigm, the discriminative information is considered to lie at the instance-level. Therefore, the discriminative learning process occurs at this level: a discriminative instance-level classifier f(~x) is trained to separate the instances in positive bags

from those in negative ones (see Fig. 1). Based on it, given a new bag X the bag-level classifier F(X) is obtained by simply aggregating instance-level scores f(~x), ∀~x ∈ X. We say that this type of paradigm is based on local, instance-level information, in the sense that the learning process considers the characteristics of individual instances, without looking at more global characteristics of the whole bag.

The methods falling in this category must address the question of how to infer an instancelevel classifier f(~x) without having access to a training set of labelled instances. In order to solve this issue, some assumption must be made about the relationship between the labels of the bags in the training set and the labels of the instances contained in these bags. In this sense, two sub-categories of IS methods emerge clearly in the literature: the ones following the Standard MI (SMI) assumption and the ones following the Collective assumption.

The SMI assumption states that every positive bag contains at least one positive instance (i.e. an instance belonging to some target positive class), while in every negative bag all of the instances are negative. This is an asymmetrical assumption which is used in many MIC problems such as the traditional one of drug discovery. Note that this assumption is that one of the instances has some desirable properties that make the bag positive. Therefore, the methods following this assumption try to identify the type of instance that makes the bag positive

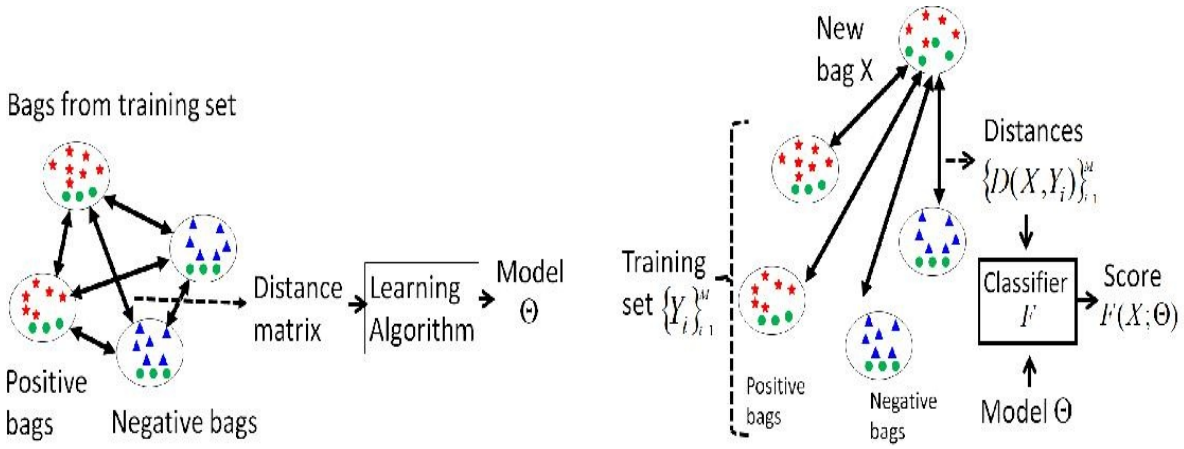


5.2.2 Bag Space Paradigm

In the Bag Space (BS) paradigm, the discriminative information is considered to lie at the bag-level. In this paradigm each bag X is treated as a whole entity, and the learning process

discriminates between entire bags. As a result, it obtains a discriminative bag-level classifier F(X) which makes use of the information from the whole bag X in order to take a discriminative decision about the class of X. We say that this type of paradigm is based on global, bag-level information, because the discriminative decision is taken by looking at the whole bag, instead of aggregating local instance-level decisions.

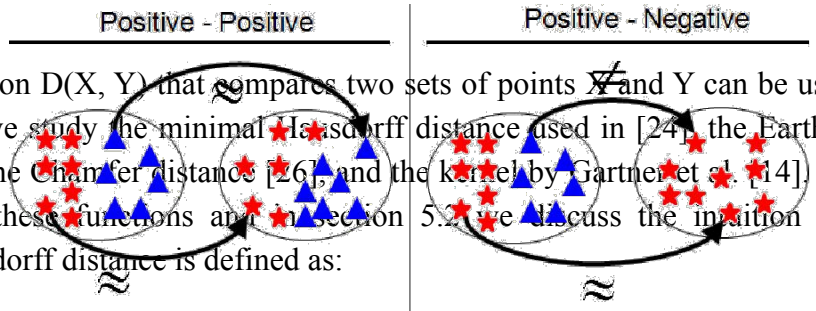
Given the fact that the bag space is a non-vector space, the BS methods make use of nonvectorial learning techniques. As far as we know, all the existent non-vectorial techniques work through the definition of a distance function D(X, Y) that provides a way of comparing any two non-vectorial entities X and Y (where these entities are bags in our problem). Once this distance function has been defined, it can be used into any standard distance-based classifier such as KNearest Neighbor (K-NN), or similarly into any kernel-based classifier such as SVM. Fig. 2 illustrates the idea under this paradigm. Although we use the term "distance" in Fig. 2, the BS paradigm also includes methods that use other types of pairwise comparisons between bags, such as kernel-based comparisons K(X, Y) in SVM-based methods.



The idea of the IS paradigm just reviewed is to estimate a model that summarizes the properties of the single instances, by discriminating those typically found in positive bags versus those found in negative ones. This makes this type of methods consider local information, in the sense that the obtained model is about instances and not about bags as a whole. At classification time, the classifier F(X) is obtained as an aggregation of local responses f(~x), where each of them consider only one instance ~x at a time.

In contrast, the methods of the BS paradigm treat the bags X as a whole, and the discriminant learning process is performed in the space of bags. This allows the algorithm to take into account more information while performing the inference of F(X). In order to learn a non-vectorial entity such as a bag, we can define a distance function D(X, Y) that compares any two bags X and Y, and plug this distance function into a standard distance-based classifier such as K-NN or SVM

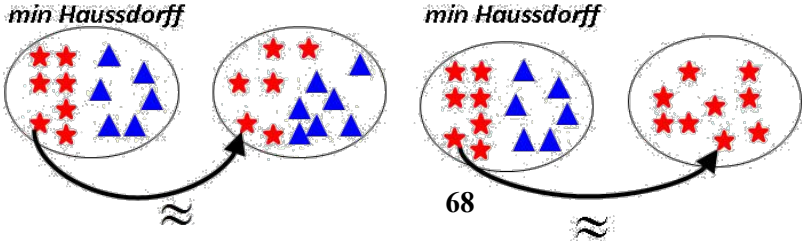
Note that a bag X is nothing else than a set of points in a d-dimensional space. Therefore, any



distance function D(X, Y) that compares two sets of points X and Y can be used in this context. In this work we study the minimal Hausdorff distance used in [24], the Earth Movers Distance (EMD) [25], the Chamfer distance [26], and the kernel by Gartner et al. [14]. Let us first see the definition of these functions and in section 5.2 we discuss the intuition behind them. The minimal Hausdorff distance is defined as:

This is the distance between the closest points of X and Y. The EMD distance, on the other hand, is the result of an optimization process. Let X = {~x1, . . . , ~xN}, and Y = {~y1, . . . ,~yM}. The EMD distance is defined as:

where the weights wi j are obtained through an optimization process that globally minimizes D(X, Y) subject to some constraints

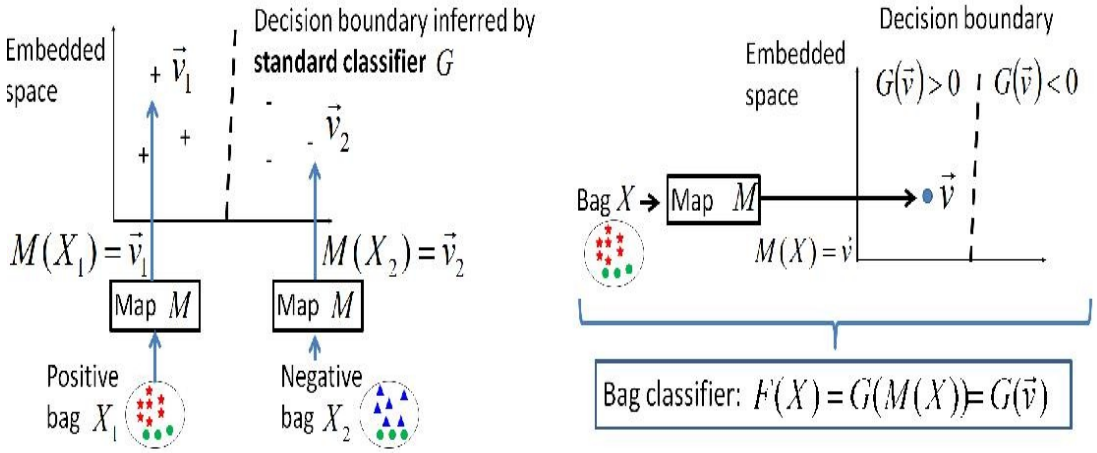


5.2.3 Embedded Space Paradigm

Both the last paradigm and the one presented in this section are based on extracting global information about the bag. In the BS paradigm this is done in an implicit way through the definition of the distance function D(X, Y) or kernel function K(X, Y). This function defines how bags are compared, and therefore, how the information about them is considered in the matching.

In the ES paradigm, this is done in an explicit way, by defining a mapping M : X 7→ ~v from the bag X to a feature vector ~v which summarizes the characteristics of the whole bag. Different definitions of this mapping function put emphasis on different types of information, and have a high impact on the performance of the method.

In this sense, we can split the existing ES methods in roughly two sub-categories. In the first one the methods simply aggregate the statistics of all the instances inside the bag, without making any type of differentiation among instances. In contrast, in the vocabulary-based paradigm the mapping is constructed by analyzing how the instances of the bag match certain prototypes that have been previously discovered in the data.



XXX(Leave)Chapter 6-HYBRID ALGORITHM

In this section, the description of approaches to machine learning that combine and thus club the advantageous features of several previously used algorithmic ideas along with some smart human decisions pertaining to the specificity of the problem such as the handling of two highly correlated features etc. These approaches resulted in accuracy gain at the cost of a negligible

increase in both the learning and testing time. It is important to note that the gain from the hybrid approach is bound to increase while no expected noticeable change to the processing time, when it comes to application on the real world data, which might vary so much that using a single fixed approach might not result into good enough results.

Before proceeding to our Hybrid algorithms, the general as well as significant drawbacks from the previous approaches which motivated our approach are given below.

6.1 K-Means Algorithm:

The assumptions on which this algorithm works are as follows

- k-means assumes the variance of the distribution of each attribute (variable) is spherical;
- all variables have the same variance;
- the prior probability for all k clusters is the same, i.e., each cluster has roughly equal number of observations;

If any one of these 3 assumptions are violated, then k-means will fail to give the expected result accuracy.

This has been illustrated on the next page.

Broken Assumption: Non - Spherical Data

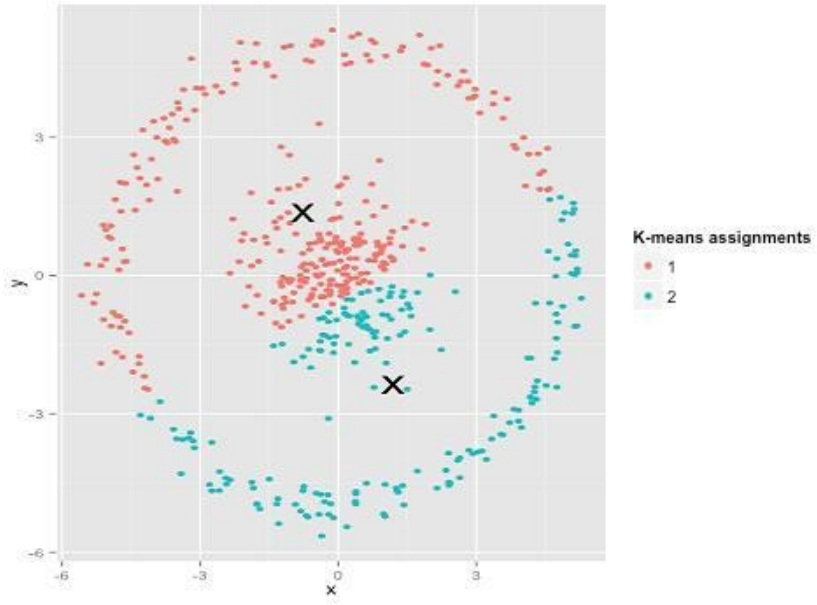


Figure 5.1 Calculated Result

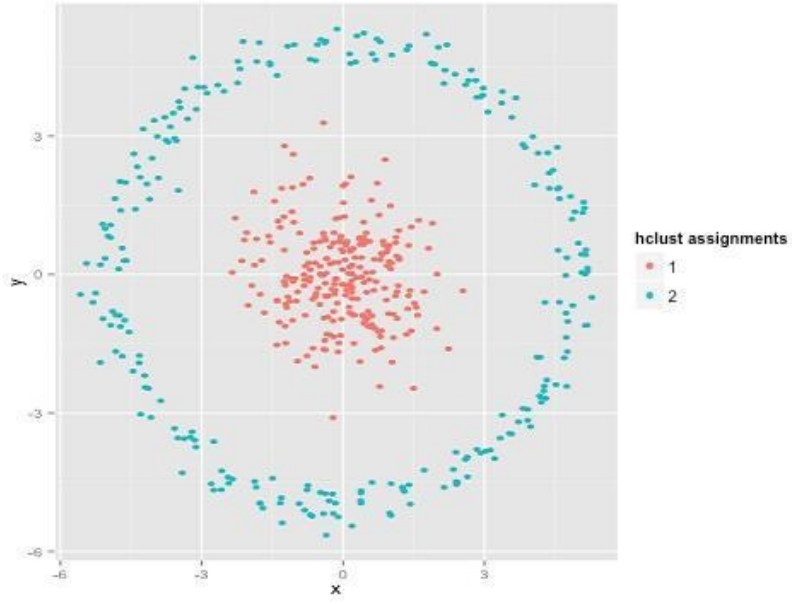


Figure 5.2 Expected Result

There can be several such examples where the K-Means clustering alone may result out to be an inappropriate approach. It is clear that the failure is a possibility due to the assumptions turning out to be wrong and not the improper working of the algorithm.

Complexity Analysis of k-means Clustering

The k-means problem is solved using Lloyd's algorithm. The **average complexity** is given by **O(k n T)**, were n is the number of samples and T is the number of iteration. The **worst case complexity** is given by **O(n^(k+2/p))** with n = n_samples, p = n_features. In practice, the k-means algorithm is very fast (one of the fastest clustering algorithms available), but it falls in local minima. That's why it can be useful to restart it several times.

6.2 Naïve Bayesian Classifier

This algorithm is based on the following equation.

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}.$$

1) It assumes the probability distribution of the attributes is independent of each other.
2) Zero Frequency Problem: To make sure that zero frequency problem does not cause issues, the implementation used Laplacian Smoothing, thus making the result an approximation, especially for the classes with smaller total frequencies.

So if you have no occurrences of a class label and a certain attribute value together then the frequency-based probability estimate will be zero.

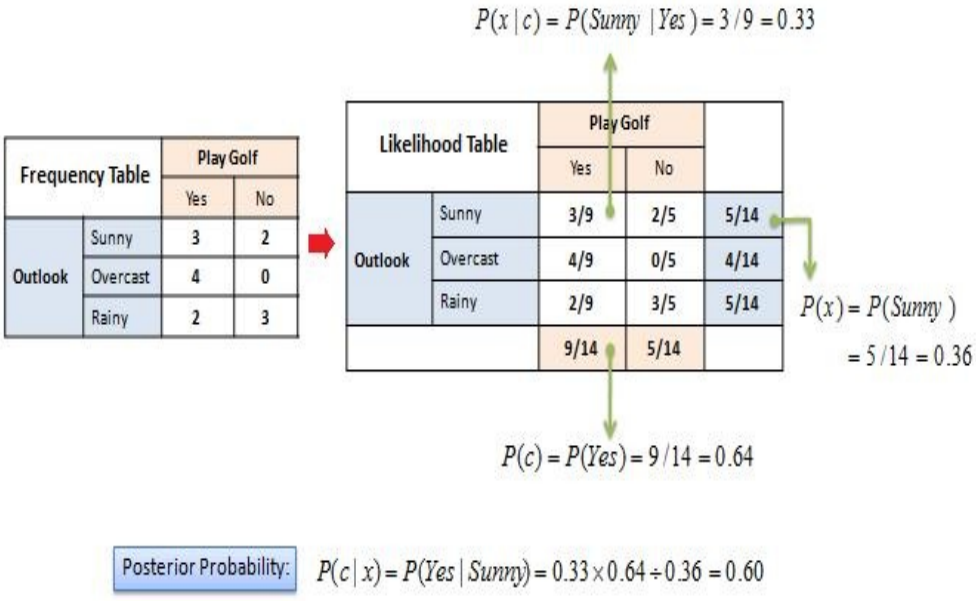


Figure 5.3 Zero Probability Problem

So to assure the independence between the various features, we decided to apply the clustering algorithm first, then followed by the Bayesian Classification, as this helps us eliminate the Bayesian drawbacks, while minimising the K-means error probability.

6.3 K-means with naïve bayes

We initially implemented K-means individually. Then we worked on hybrid by combining K-means with naïve bayes. K-means is used to group together similar data. Then naïve bayes was implemented on each cluster and model was made. For each new testcase, it was first determined to which cluster it belongs. Then the naïve bayes model for that particular cluster was used to make prediction for the given testcase. K-means was used with the hope that grouping together similar data will help in increasing accuracy of naïve bayes algorithm. Here we had a tradeoff between time of computation and accuracy but additional gained accuracy was preferred.

For this algorithm, we first discretized the data as naïve bayes requires data which is in discrete form. We couldn‘t implement Gaussian naïve bayes as data distribution was not Gaussian and still using the algo would have resulted in poor accuracy. We had two choices for data discretization, equal width discretization and equal frequency discretization. Equal width discretization resulted in better performance.

We ran our algorithm on different number of clusters but found that 5 number of clusters resulted in highest accuracy. Hence 5 clusters have been used for this algorithm.

On running this hybrid algorithm on Cleveland heart disease dataset we got an accuracy of 78.4% which was an improvement over naïve bayes implementation without k-means.

6.4 Decision Tree

Decision trees have been used to obtain some of the most accurate solutions by clubbing them with other statistical learning techniques. Still we highlight some of the drawbacks of Decision tree Algorithms

- Decision trees tend to overfit on data with a large number of features. Getting the right ratio of samples to number of features is important, since a tree with few samples in high dimensional space is very likely to overfit.
- In general, the run time cost to construct a balanced binary tree is $O(n_{samples} n_{features} \log(n_{samples}))$ and query time $O(\log(n_{samples}))$. Although the tree

construction algorithm attempts to generate balanced trees, they will not always be balanced.

Assuming that the subtrees remain approximately balanced, the cost at each node consists of searching through $O(n_{features})$ to find the feature that offers the largest reduction in entropy. This has a cost of $O(n_{features} n_{samples} \log(n_{samples}))$ at each node, leading to a total cost over the entire trees (by summing the cost at each node) of $O(n_{features} n_{samples}^2 \log(n_{samples}))$.

So we first use K-Means to form the clusters, and then apply the Decision tree algorithms, so that the overfitting can be minimized as well as the smaller and more closely related cluster sizes enable us to improvise both pre-pruning and post-pruning of the tree for the same given size.

6.5 Decision tree with Kmeans

We initially implemented decision tree individually. Then we worked on hybrid by combining Kmeans with decision tree. In random forest, data is randomly divided in different groups and decision tree is made on each group and result is determined. However in this algorithm, Kmeans is used to group together similar data. Then decision tree was implemented on each cluster and model was made. For each new testcase, it was first determined to which cluster it belongs. Then the decision tree model for that particular cluster was used to make prediction for the given testcase. Kmeans was used with the hope that grouping together similar data will help in increasing accuracy of decision tree algorithm. Decision tree will have fewer branches due to grouping of similar data and will not overfit. The hybrid algorithm was run on both continuous dataset and dataset after discretization. Continuous dataset resulted in higher accuracy. We ran our algorithm on different number of clusters but found that 5 number of clusters resulted in highest accuracy. Hence 5 clusters have been used for this algorithm.

On running this hybrid algorithm on Cleveland heart disease dataset we got an accuracy of 79.1% which was an improvement decision tree implementation without kmeans and random forest implementation.

6.6 Photo Classifier

Yelp hosts tens of millions of photos uploaded by Yelpers from all around the world. The wide variety of these photos provides a rich window into local businesses, a window we're only just peeking through today.
One way we're trying to open that window is by developing a photo understanding system which allows us to create semantic data about individual photographs. The data generated by the system has been powering our recent launch of tabbed photo browsing as well as our first attempts at

content-based photo diversification.

One can imagine a variety of ways to tackle the ambitious goal of holistically understanding pictures. To help simplify our problem, we focused initially on only sorting photos into a handful of predefined classes. Further, we focused only on categories of photos directly relevant to restaurants as shown below




food **drink** **inside** **outside** **menu**

To develop a classifier that can put a photo into one of these groups, we need to first collect many photos with known labels. We collected this information through a few different ways:

- **Photo captions**: A good number of "menu" photos have the word "menu" in their captions. Similarly, we can find photos titled "sushi" or "burger" that are likely to be food. For the former, we had to worry about false positives because it's not uncommon to see "food" or "drink" photos whose captions are of the pattern "Best on Their Menu!", and as a result some cleanup was necessary. To aid us in identifying food items, we relied on Yelp's menu structures which maintain each business' list of food items. We found that matching food items from the list to the captions of photos yielded a dataset of high precision.

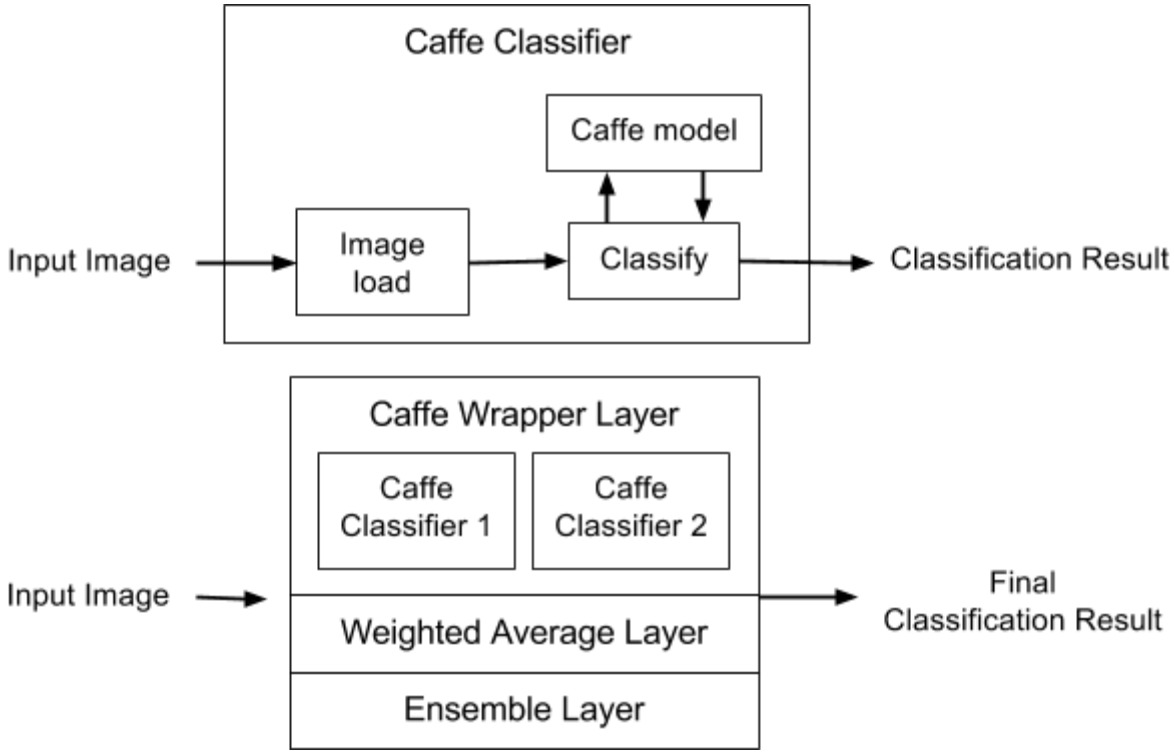
- **Photo attributes**: When uploading photos to Yelp, users are allowed to mark a few attributes about the photo, such as *"Is it the storefront?"* They are not always accurate, but still serve as a good source of candidate photos.

- **Crowdsourcing**: We ran additional tasks through a crowdsourcing partner to correct our guesses for what label should be applied to each photo and to collect more "inside" and "outside" photos. We have found that this led to generally good quality labels at a reasonable cost (both in time and money).

Once we had our labeled data, we employed deep convolutional neural networks (CNNs) in the form of "AlexNet" to recognize those classes. CNNs usually consist of a deep stack of multiple**convolutional layers** (for extracting spatially local and translation-invariant features), **ReLU (Rectified Linear Units) layers** (for non-saturating activations), **pooling layers** (for down-sampling and translation-invariance), **local response normalization layers** (for better generalization) and **fully-connected layers** as in conventional feedforward neural networks. Softmax outputs and regularization methods such as dropout are also commonly used. Our CNN was built on AWS EC2 GPU instances based on the Caffe framework. We like Caffe because it's easy to use, performant, open source (BSD 2-

clause), and under active development. To address Caffe's software dependencies, we wrapped our CNN using Docker so that it could be more easily deployed.

We also created abstractions to ensure that our CNN could be easily integrated with other possible forms of classifiers, including different instances of CNNs. As illustrated below, our baseline is a "Caffe Classifier" that runs the CNN by means of Caffe; it's a special form of an abstract classifier that can take different signals and perform different classification algorithms. Our current "facade" classifier is an ensemble that takes the weight average of classification results from two independently trained Caffe Classifiers. It would be quite straightforward if we decide to further incorporate new classifiers relying on other signals, such as photo captions.



On an evenly split gold test set of 2,500 photos, our current classifier shown above has an overall precision of 94% of precision and recall of 70%. While these numbers can definitely be improved, we found them reasonably good for the applications described below.

Chapter 7-MACHINE LEARNING TOOLS and IDE

7.1 DEVELOPMENT SOFTWARE

- Why **Python**?

  Python is far more powerful and superior to many other languages(scientific): it's a general-purpose language that include a well suited GUI along with powerful libraries. It is free licensed and open-source, fundamentally object-oriented, highly portable, extensible and embeddable and easily maintainable.

7.2 IDE

- **PyCharm (IntelliJ)**

  - ❖ It is primarily used for scientific programming in python language.
  - ❖ It incorporates NumPy, SciPy and matplotlib
  - ❖ Includes support for interactive features along with well supported plugins.
  - ❖ It is available through cross platform on Windows with Python(x,y) and using PyQt on Linux Distributions.
  - ❖ Syntax highlighting along with high introspection for code completion.
  - ❖ Supports multiple Python consoles including IPython.

7.3 SOFTWARE LIBRARY

- **Scikit-learn**

  - ❖ It is an open source and free licensed machine learning library for the Python programming language.
  - ❖ It includes various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy along with matplotlib to incorporate tools for creating interactive results that include graphs, pie charts and histograms.
  - ❖ It is largely written in Python.
  - ❖ It was created as an open source project in GSOC (Google Summer of Code).
  - ❖ Simple and efficient tools for data mining and data analysis
  - ❖ Accessible to everybody, and reusable in various contexts.

- **Numpy**

  It is the one of the most fundamental package for scientific computing with Python and is incorporated in scikit-learn. It consists of some other things including:

  - ❖ A powerful N-dimensional array object.
  - ❖ Sophisticated functions.
  - ❖ Useful for linear algebra and random number capabilities.
  - ❖ Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined.

- **Matplotlib**

  Matplotlib is a python 2D plotting library that contains a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in python scripts with ease to provide interactive features.

Chapter 7-Proposed Methodology

The proposed algorithm consists of four main steps. The features are extracted from images using a seven layer CNN architecture which are then represented using the BoW model with earth mover's distance as a metric each bag. A custom kernel binary SVM classifier along with association rules are then used to predict labels for each restaurant.

7.1 Feature Extraction

For feature extraction we use the pre-trained model of Caffe [17] open source deep learning framework to extract features from images. A 4096-dimensional feature vector, fc7 feature is extracted from images through five convolution layers and two fully connected layers using the CNN architecture given by Krizhevsky et al. [18]. In order to compute fc7 features for each image, the images are converted into a form that is compatible with the CNN as its architecture requires a fixed input size of 227x227 pixels.

7.1 Bag-of-Words Representation

In order to classify images, the Bag-of-Words model is employed which treats image features as words. It represents the data items (images) as a histogram over features (words). The BoW algorithm was initially presented for text extraction domain in word document analysis, and was later modified further for computer vision applications [19]. The BoW classification model analogous to word dictionary is based on the process of vector quantization image features extracted from local regions or points, like texture, color etc. The BoW model can be described as follows. Given a dataset containing m images characterized by where is the features extracted using the above method,a unsupervised learning algorithm k-means, is used to cluster based on the number of clusters which depicts the categories for classification. Thus, each image in is represented as a vector v of counts over each feature.

)) … (1)

The standard BoW pipeline is implemented. Firstly, the features are extracted using the method defined above. The feature vectors are quantized into words by mapping them to the nearest codevector in the codebook by using Earth Mover's Distance (EMD).

where are the weights obtained through a global optimisation process that globally minimizes [20]. Given the bag of features representation of images from different classes, a classifier is trained to classify the images.

7.3 Label Classification

We train a SVM for each attribute to obtain binary SVM classifiers which can then be used to predict the labels for an unknown class. We transform the previously computed EMD, between the bags into a kernel using extended Gaussian kernel [21].

For each binary SVM classifier, Let where denotes an n-dimensional bag of features vector for the class, label where 1 denotes that the current attribute is associated with this class and -1 indicates that the attribute is not associated with the given class. Hence a total of binary SVM classifiers are obtained where gives the number of attributes. For kth classifier model, we can represent it by decision function. Using the representor theorem [] we have,

The decision function can then be given as

For the chosen symmetric kernel (), using mercers theorem.

The decision function is represented in the dual form

The variable vector can be determined by minimizing the following standard support vector formulation

s.t.

The parameters and are found by cross validation during the training phase.

7.4 Multi Label Training

For each binary SVM classifier we do a 10 fold cross validation to find parameters and and in each fold, the ratio of positive samples to negative samples is kept the same. We use LibSVM [ ] for training the SVM with a precomputed Gram matrix [ ]. For each iteration of the binary SVM classifier, an average F1 score is computed and if it is better than the previous iteration then the new values of and are kept. Thus for each of the binary classifiers we get pairs of and values which are used in the predicting the labels for whole of the training set to determine the optimal threshold value. The obtained, and values are then used to predict the labels/attributes for the test set.

7.5 Association Rules

We mine association rules between the labels for further increasing the accuracy of the proposed system. Two types of rules have been considered, positive association rules and negative association rules. We take positive association rules to be those which predict the presence of a label for a class given the presence (absence) of certain other labels. Conversely, negative association rules are those which predict the absence of a label for a class given the presence (absence) of other associated labels. We have used PrefixSpan[ ] algorithm to mine data patterns in the training data with support value greater than , where is the minimum support value. For the resulting data patterns, those with a single item are not considered. For each label occurring

in the remaining data patterns, confidence is computed for positive and negative sequences where confidence for a sequence is given as

here is the label under consideration, is the sequence containing for which confidence is being computed. Sequences with confidence values above minimum confidence value are added to the set of positive or negative association rule depending on the sequence. Minimum confidence values for individual labels are decided by their relative F1 scores. Labels with high F1 score have higher minimum confidence values and those with a low F1 score have relatively lower confidence values such that the difference between the minimum confidence value and F1 score is kept constant. Table 1 shows the positive and negative rules mined between different labels along with their corresponding support and confidence.

Pattern	0	0 8	0 ~4 8	0 ~4 ~7 8	1	1 2	1 2 4	1 2 5	1 2 5 6
Support	130	109	104	101	190	164	99	156	156

(a)

Rule	**Confidence**
2 4 □ 6	1.0
1 2 □ 5	0.951
1 2 5 □ 6	1.0

(b)

Rule	**Confidence**
0 8 □ ~4	0.954
0 ~4 8 □ ~7	0.971

(c)

Table 1 : a) subset of patterns mined from the input label sequence with support value=0.25. b) Positive association rules for given subset. c) Negative association rules for the given subset. ~ symbol before a label number represents the absence of the label.

Chapter-9 RESULT and CONCLUSION

We evaluate the efficiency of the proposed algorithm on the basis of F1 score. We tested the approach on Yelp Kaggle database [25] in order to include realistic images of a variety of resolution and sizes. The experiments are conducted using Caffe deep learning framework.

9.1 Dataset

Yelp Kaggle database contains user uploaded restaurant images. The problem is of multiimage multi label classification with each restaurant having multiple labels that need to be automatically tagged to each business/restaurant. The database contains around 230,000 images in the training dataset with 1996 businesses and around 240,000 images in the test dataset with 10,000 businesses. The label set is limited to size of 9 with each label depicting different

meaning.

Label #	Label Name
0	Good for lunch
1	Good for dinner
2	Takes reservations
3	Outdoor seating
4	Restaurant is expensive
5	Has alcohol
6	Has table service
7	Ambience is classy
8	Good for kids

Table 2. Labels for businesses

The images shown in Fig 1. correspond to the images of particular business. The images not only contain food, but also the restaurant name, menu cards and drinks. As the images are user-uploaded photos, they do not have consistency. Some images are in portrait mode, some in landscape mode, some are in square shape etc. To make the images consistent and compatible with CNN they are reduced to fixed size of 227x227 pixels. The images in the training dataset are split into 70-30 training validation set. The proposed algorithm is trained on the training set, validated and the F1 score evaluated on the test set is reported as the performance measure.

9.2 Implementation Details and Results

We employed Caffe deep learning framework to compute the feature vector for all the training images. The standard BoW model is used to construct bag of features for each business. For vector quantization of features we use the EMD to calculate the distance of bag from every other bag in the training set. We use LibSVM and employ a custom kernel SVM for classification. Finally, we utilized association rule mining to extract rules between labels to further increase the accuracy of the proposed system. The SPMF open source java library is used for mining rules One of the critical parameters of our system is the threshold value ′ ′ defined above, we experimented for various values of ″ and found 0.47 to be the optimal value. It is noticed that on increasing the threshold value there is a significant drop in correct labelling (true positives) for the system while decreasing the threshold results in degradation of performance due to an increase in mislabeling(false positives) as shown in Fig 2. It is further noticed that a slight increase in threshold value leads to a small improvement in the prediction of training dataset but

the system performance degrades for test dataset. This can be justified as there is some noise in the training dataset and such fine-tuning increases the probability of overfitting.

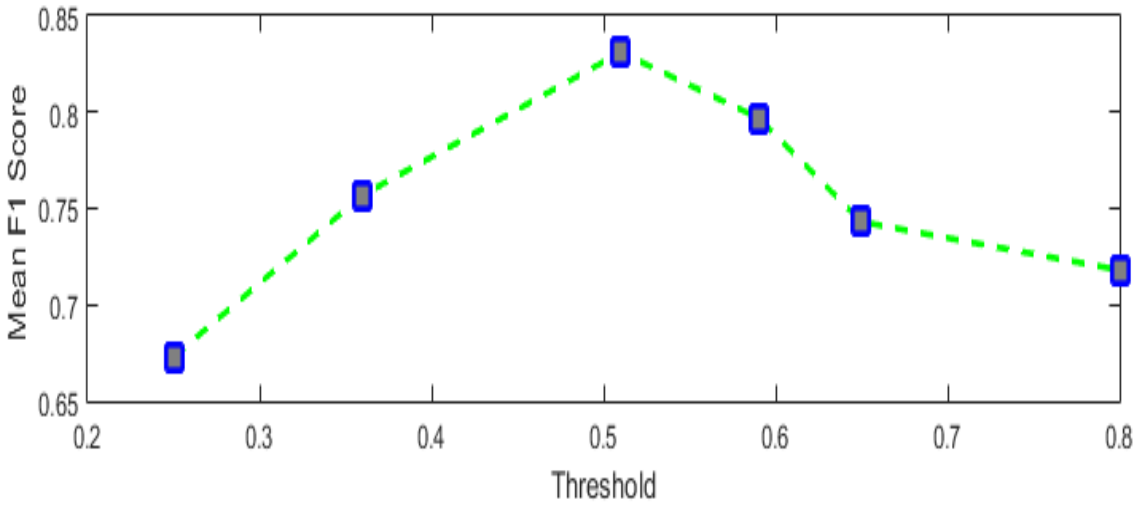


Fig 2. F1 score vs Threshold

From Table 3, we observe that our system performs better for certain labels in comparison to other labels. Maximum F1 score is achieved for label 6:has_table_service while the minimum F1 score is obtained for label 3:outdoor_seating. This can be attributed to the fact that majority of the images in the dataset of businesses with label 3 are in an indoor setting. It is also observed that the maximum gain in accuracy from association rules is obtained for label 3. For other labels there is a marginal to moderate increase in accuracy.

From Table 4, we observe that our system suffers from a higher chance of misclassification when the number of attributes associated with a business are low and there is strong variation in ambient illumination. This is due to limited applicability of association rules under the given conditions coupled with sensitivity of certain labels namely 0,1,3 and 7 on lighting conditions.

Label	**F1 score w/o Association Rules**	**F1 score with Association Rules**
0:good_for_lunch	0.75660	0.75682
1:good_for_dinner	0.85372	0.85372
2:takes_resrvations	0.89410	0.89410

3:outdoor_seating	0.70193	0.74726
4:restaurant_is_expensive	0.80407	0.80715
5:has_alcohol	0.87823	0.87893
6:has_table_service	0.94421	0.94844
7:ambience_is_classy	0.78620	0.79104
8:good_for_kids	0.88831	0.88831

Table 3: mean F1 score of each label for our model without Association rules and our model with Association rules. The mean F1 score is computed for 30% of the training data reserved for validating the model

From Table 4, we observe that our system suffers from a higher chance of misclassification when the number of attributes associated with a business are low and there is strong variation in ambient illumination. This is due to limited applicability of association rules under the given conditions coupled with sensitivity of certain labels namely 0,1,3 and 7 on lighting conditions.

Business Photos	**Correct Labels**	**Predicted Labels**
	1,2,4,5,6,7	1,2,4,5,6,7
	0,1,2,3,5,6,7	0,1,2,3,5,6,7
	3	0

	1 3 8	0 3 8

It can be observed from Table 5 that our model outperforms the listed rival models in terms of mean F1 score. Also, there is a significant increase in accuracy over rival models. The 2 Layer model uses fully connected neural network (FCNN) to extract features. The VGG model is trained on a relatively small dataset and uses GoogleNet framework to extract features. This is mainly due to the fact that we use separate binary SVM classifiers for prediction of each label.

Comparison Table

Model	**Mean F1 Score**
Random Guesser [ ]	0.41337
2 Layer FC NN [ ]	0.49
VGG CNN-S + Fine-tuning [ ]	0.60881
Benchmark with Color Features [ ]	0.64598

Our model	0.82245

Table 5: mean F1 score for various models along with the provided benchmark

Moreover the relationships between different labels are utilized in the form of association rules to further enhance the system performance by predicting missing labels (positive association rules) and removing misclassified labels (negative association rules) . Also, tenfold cross validation for parameters increases the robustness of the system.

Chapter-11 REFERENCES

[1] Jaume Amores. Multiple instance classification: Review, taxonomy and comparative study. Artificial Intelligence, 201:81–105, 2013.

[2] P. Auer, R. Ortner, A boosting approach to multiple instance learning, in: Proc. of European Conference on Computer Vision, 2004, pp. 63–74.

[3] Z.-H. Zhou and M.-L. Zhang. Neural networks for multi instance learning. In ICIIT, 2002

[4] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. "Solving the multiple-instance problem with axis-parallel rectangles," Artificial Intelligence, vol.89, no.1-2, pp.31-71, 1997.

[5] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In NIPS, 2002.

[6] J. Ramon and L. De Raedt. Multi instance neural networks. In ICML workshop on attribute-value and relational learning, 2000.

[7] O. Maron and T. Lozano-Pérez. "A framework for multiple-instance learning," in: Advances in Neural Information Processing Systems 10, M. I. Jordan, M. J. Kearns, and S. A. Solla, Eds. Cambridge, MA: MIT Press, pp.570-576, 1998.

[8] Yang, C. and Lozano-Pérez, T.: Image database retrieval with multiple-instance learning techniques, In: Proceedings of the 16th International Conference on Data Engineering, pp. 233-243, San Diego, CA, 2000

[9] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. Signal Processing Magazine, IEEE, 29(6):82–97, 2012.

[10] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, and Y. LeCun. Learning convolutional feature hierarchies for visual recognition. In NIPS, 2010

[11] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In ICML, 2009.

[12] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In CVPR, 2014

[13] Y. Xu, T. Mo, Q. Feng, P. Zhong, M. Lai, E. I. Chang, et al. Deep learning of feature representation with multiple instance learning for medical image analysis. In ICASSP, 2014

[14] H. O. Song, Y. J. Lee, S. Jegelka, and T. Darrell. Weaklysupervised discovery of visual pattern configurations. In NIPS, 2014

[15] J. Wang and J.-D. Zucker. Solving the multiple-instance problem: a lazy learning approach, Proceedings 17th International Conference on Machine Learning, pp. 1119-1125, 2000

[16] Y. Chevaleyre, J. D. Zucker, Solving multiple-instance and multiple-part learning problems with decision trees and rule sets. Application to the mutagenesis problem. In: Stroulia, E., Matwin, S. (eds.): Lecture Notes in Artificial Intelligence, Vol. 2056. Springer, Berlin (2001) 204-214.

[17] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding. http://caffe.berkeleyvision.org/, 2013.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012

[19] A. Bosch, X. Muñoz, and R. Martí, “Which is the best way to organize/classify images by content?” Image and Vision Computing, vol. 25, no. 6, pp. 778–791, 2007

[20] Y. Rubner, C. Tomasi, and L. Guibas. The Earth Mover’s distance as a metric for image retrieval. International Journal of Computer Vision, 40(2):99–121, 2000.

[21] O. Chapelle, P. Haffner, V. Vapnik, Support vector machines for histogram-based image classification, IEEE Transactions on Neural Networks 10 (5) (1999) 1055–1064.

[22] Mercer, J. (1909), "Functions of positive and negative type and their connection with the theory of integral equations", Philosophical Transactions of the Royal Society A 209(441–458): 415– 446, doi:10.1098/rsta.1909.0016

[23] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[24] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, M-C. Hsu, PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In Proc. 2001 Int. Conf. Data Engineering (ICDE'01), Heidelberg, Germany, pp. 215–224, 2001

[25] Yelp Kaggle Database. https://www.kaggle.com/c/yelp-restaurant-photo-classification