# 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) |

# 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.

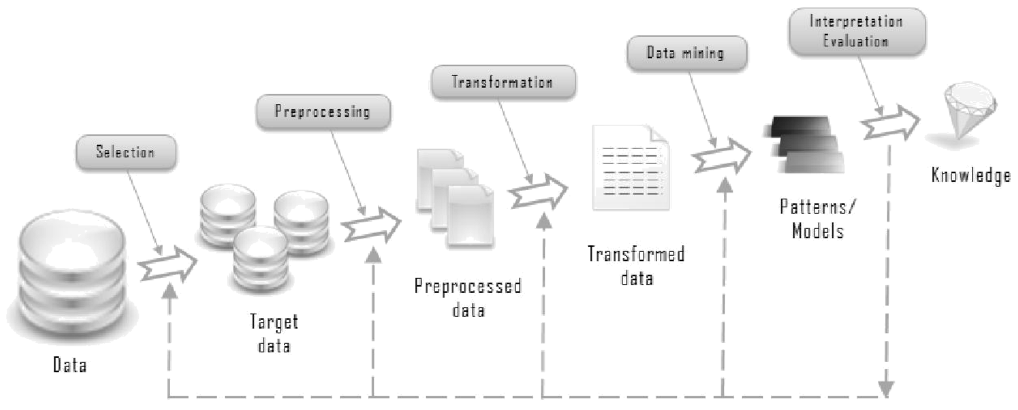


*Figure 2.1: Data mining as a step in the process of knowledge discovery.*[1]

Knowledge discovery as a process is depicted in Figure 2.1 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)

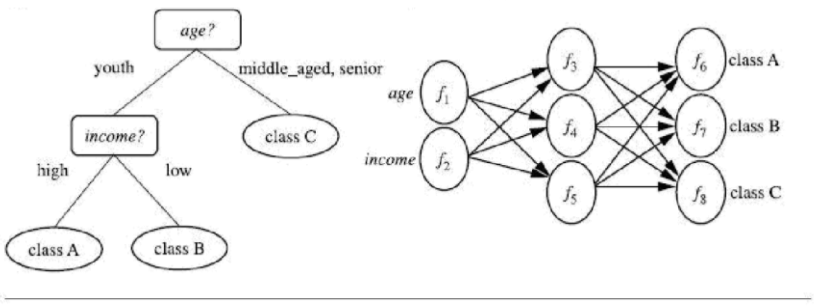


*Figure 2.2: A classification model can be represented in various forms, such as (a) IF-THEN rules, (b) a decision tree, or a (c) neural network.”*

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.

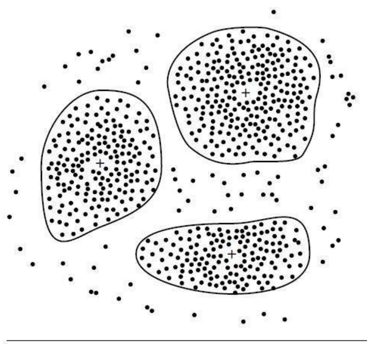


*Figure 2.3: A 2-D plot, showing three data clusters. Each cluster "center" is marked with a "+"*

## 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,

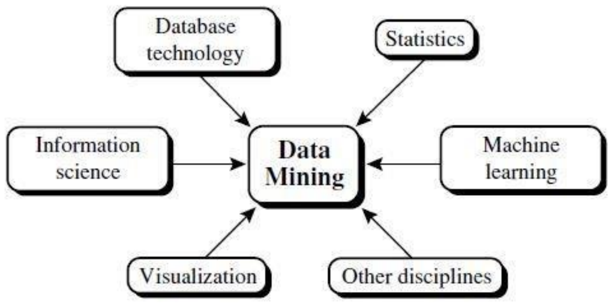
(2) valid on new or test data With some degree of certainty, (3) potentially useful, and (4) novel. An interesting pattern represents knowledge. Several objective measures of pattern interestingness exist.

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.

*Figure 2.4 Data mining as a confluence of multiple disciplines*

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.

# 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:

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

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.

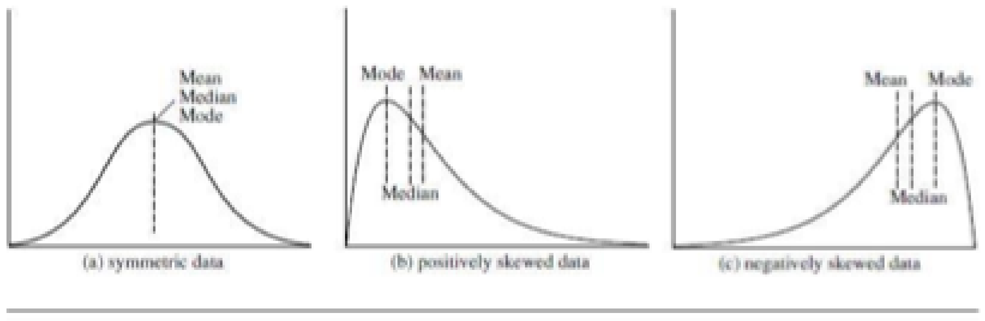


*Figure3.1: Mean, median, and mode of symmetric versus positively and negatively skewed data*

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.

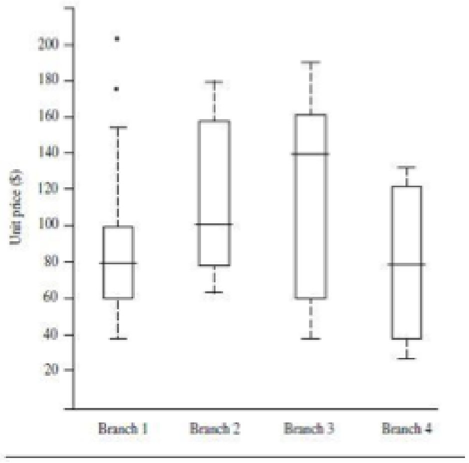


*Figure 3.2: Example of box-plot showing the unit price data for items sold at four branches of All Electronics during a given time period*

## 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 $\bar{x}$ 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.

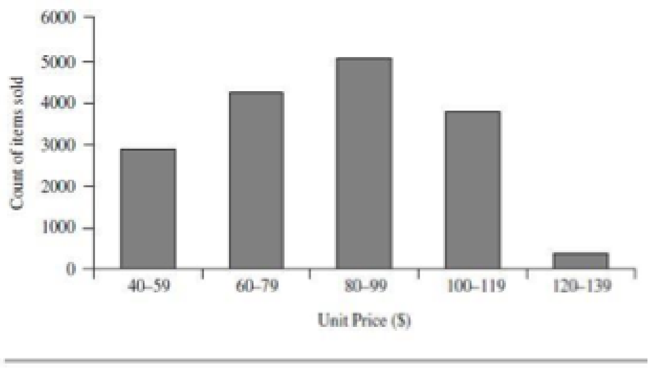


*Figure 3.3: Example of histogram*

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.

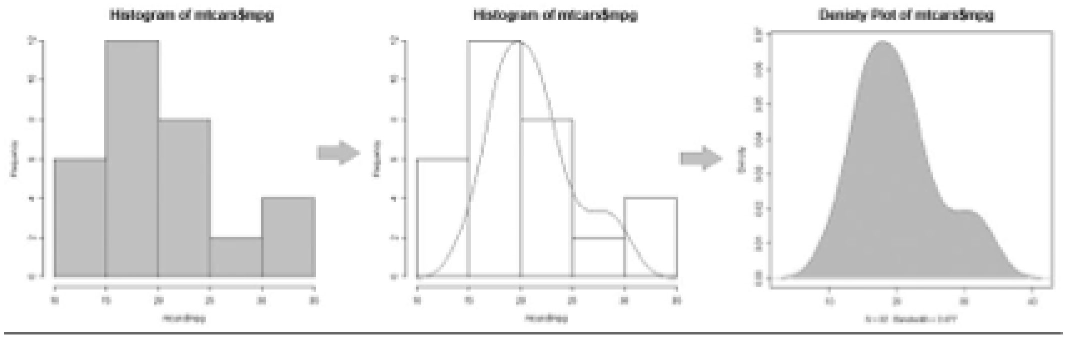


*Figure 3.4: Transformation of histogram into kernel density plot*

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 lOO$f_i$ % 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.

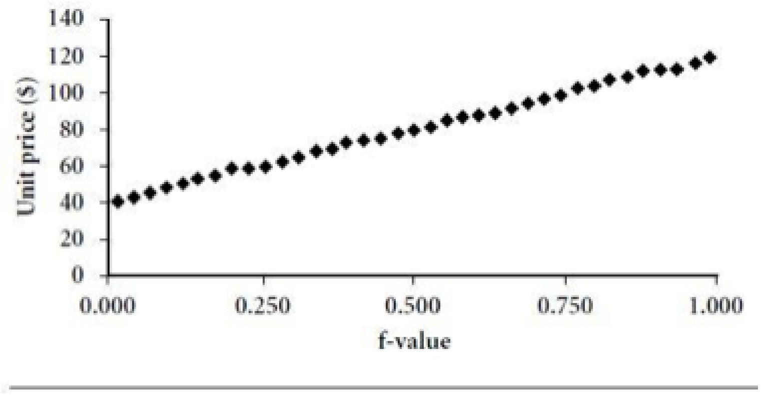


*Figure 3.5: Example of quantile plot*

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 $y_1, \ldots, y_M$ 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.

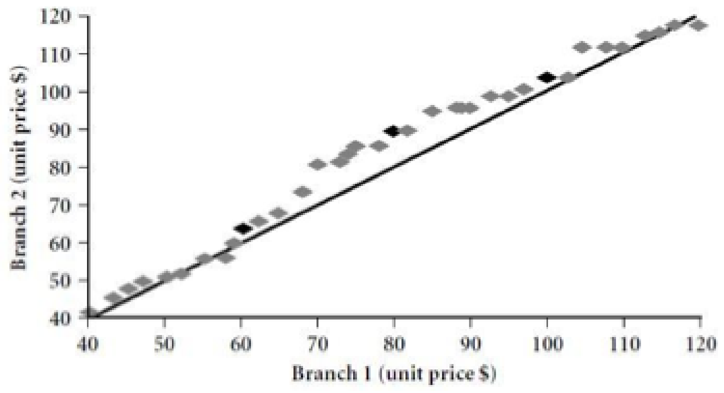
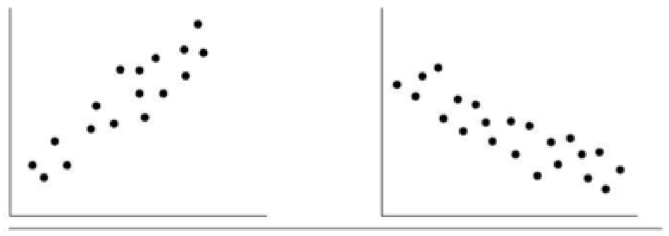


*Figure 3.7: Example of scatter plot.*

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.

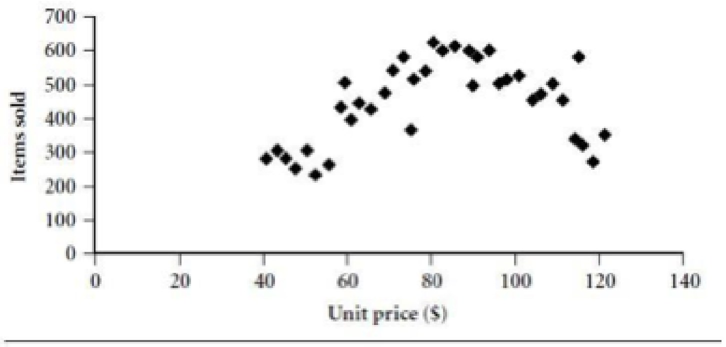


*Figure 3.8: Scatter plots can be used to find (a) positive or (b) negative correlations between attributes*

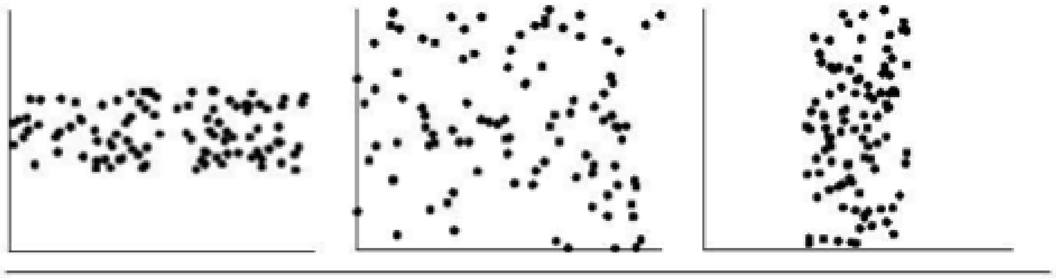


*Figure 3.9: Three cases where there is no observed correlation between the two plotted attributes in each of the data sets*

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.”

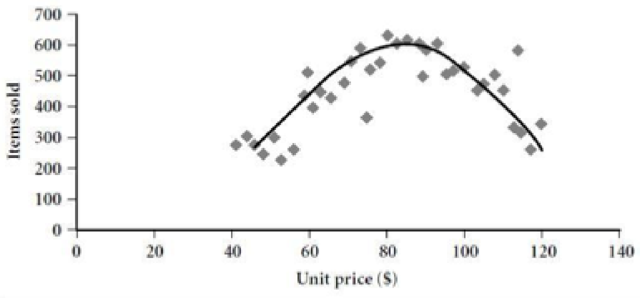


*Figure 3.10: Example of 10655 plot*

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.

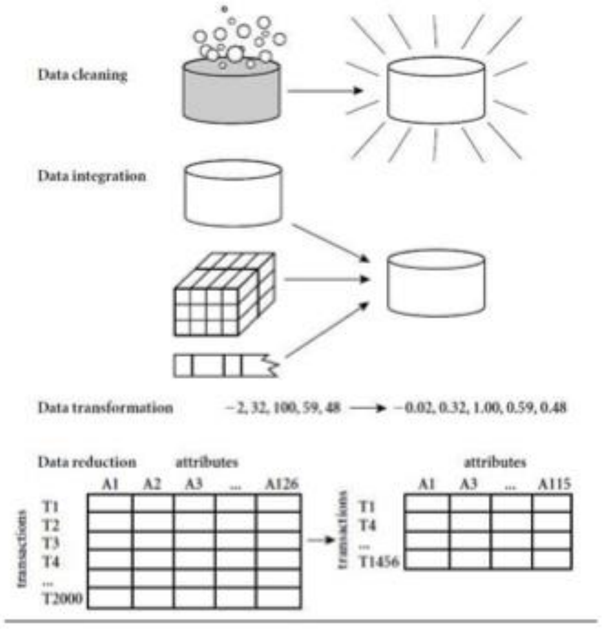


*Figure 3.11: Forms of Data Preprocessing*

### 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) removing the instances,

(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,

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

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.

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,

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, $\sigma_A$ and $\sigma_B$ 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.

# 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.

## 4.3 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.

### 4.3.1 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.

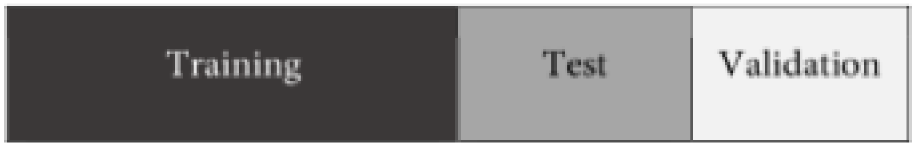


*Figure 4.1: Training, test and validation set*

## 4.4 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.

## 4.5 Linear Regression

Linear regression attempts to model the relationship between two or more variables by fitting a linear equation to observed data. One variable is considered to be an dependent variable, and the others are considered to be explanatory variables.

For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.

Consider the data obtained from a chemical process where the yield of the process is thought to be related to the reaction temperature (see the table below).

| Observation Number | Temperature ($x_i$) | Yield ($y_i$) |
|---|---|---|
| 1 | 50 | 122 |
| 2 | 53 | 118 |
| 3 | 54 | 128 |
| 4 | 55 | 121 |
| 5 | 56 | 125 |
| 6 | 59 | 136 |
| 7 | 62 | 144 |
| 8 | 65 | 142 |
| 9 | 67 | 149 |
| 10 | 71 | 161 |
| 11 | 72 | 167 |
| 12 | 74 | 168 |
| 13 | 75 | 162 |
| 14 | 76 | 171 |
| 15 | 79 | 175 |
| 16 | 80 | 182 |
| 17 | 82 | 180 |
| 18 | 85 | 183 |
| 19 | 87 | 188 |
| 20 | 90 | 200 |
| 21 | 93 | 194 |
| 22 | 94 | 206 |
| 23 | 95 | 207 |
| 24 | 97 | 210 |
| 25 | 100 | 219 |

*Figure 4.2 Sample data for Chemical Process*

And a scatter plot can be obtained as shown in the following figure. In the scatter plot yield, $y_i$ is plotted for different temperature values, $x_i$.

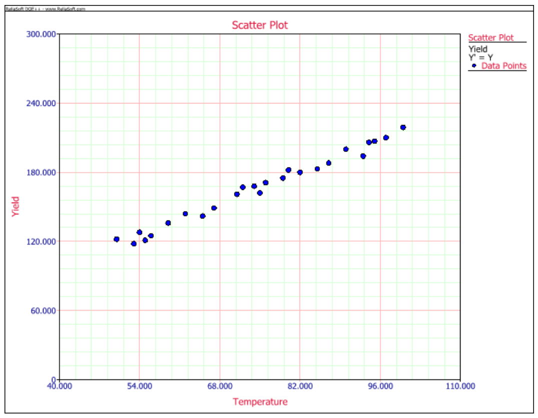


*Figure 4.3: Scatter plot of Temperature vs Yield*

It is clear that no line can be found to pass through all points of the plot. Thus no functional relation exists between the two variables x and Y. However, the scatter plot does give an indication that a straight line may exist such that all the points on the plot are scattered randomly around this line. A statistical relation is said to exist in this case.

The statistical relation between x and Y may be expressed as follows:

$$Y = \beta_0 + \beta_1 x + \epsilon$$

The above equation is the linear regression model that can be used to explain the relation between x and Y that is seen on the scatter plot above. In this model, the mean value of Y(abbreviated as E(Y)) is assumed to follow the linear relation:

$$E(Y) = \beta_0 + \beta_1 x$$

The actual values of Y(which are observed as yield from the chemical process from time to time and are random in nature) are assumed to be the sum of the mean value, E(Y), and a random error term, ε:

$$\begin{aligned} Y &= E(Y) + \epsilon \\ &= \beta_0 + \beta_1 x + \epsilon \end{aligned}$$

The regression model here is called a simple linear regression model because there is just one independent variable, x, in the model. In regression models, the independent variables are also referred to as regressors or predictor variables. The dependent variable, Y, is also referred to as the response. The slope, $\beta_1$, and the intercept, $\beta_0$, of the line are called regression coefficients. The slope, $\beta_1$, can be interpreted as the change in the mean value of Y for a unit change in x.

The random error term, ε, is assumed to follow the normal distribution with a mean of 0 and variance of $\sigma^2$. Since Y is the sum of this random term and the mean value, E(Y), which is a constant, the variance of Y at any given value of x is also $\sigma^2$. Therefore, at any given value of x, say $x_i$, the dependent variable Y follows a normal distribution with a mean of $\beta_0+\beta_1 Xi$ and a standard deviation of σ.

A linear regression model that contains more than one predictor variable is called a multiple linear regression model. The following model is a multiple linear regression model with two predictor variables, $x_1$ and $x_2$.

$$Y = 30 + 5x_1 + 7x_2 + \epsilon$$

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

The model is linear because it is linear in the parameters $\beta0$, $\beta1$ and $\beta2$. The model describes a plane in the three-dimensional space of Y, $x_1$ and $x_2$. The parameter $\beta0$ is the intercept of this plane. Parameters $\beta1$ and $\beta2$ are referred to as partial regression coefficients. Parameter $\beta1$ represents the change in the mean response corresponding to a unit change in $x_1$ when $x_2$ is held constant. Parameter $\beta2$ represents the change in the mean response corresponding to a unit change in $x_2$ when $x_1$ is held constant. Consider the following example of a multiple linear regression model with two predictor variables, $x_1$ and $x_2$:
This regression model is a first order multiple linear regression model. This is because the maximum power of the variables in the model is 1.
Also shown $\varepsilon$ is an observed data point and the corresponding random error, . The true regression model is usually never known (and therefore the values of the random error terms corresponding to observed data points remain unknown).

The regression plane and contour plot for this model are shown in the following two figures, respectively.

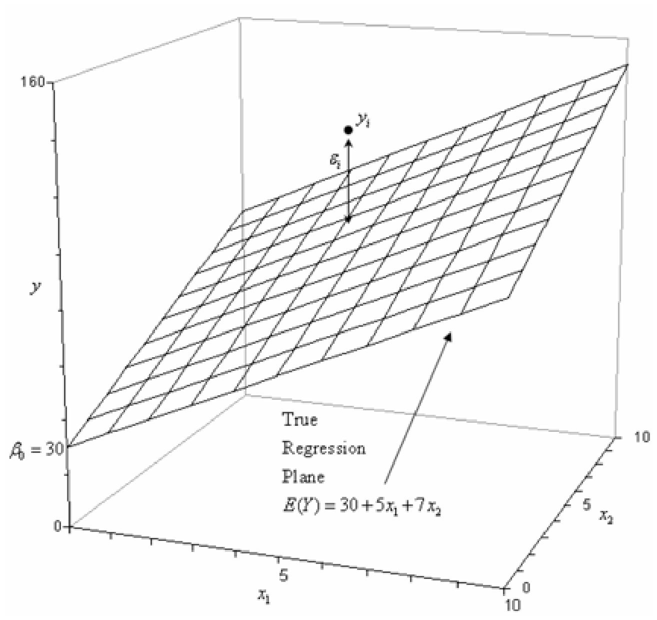


*Figure 4.4 Regression plane and contour plot for regression model*

## 4.6 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.

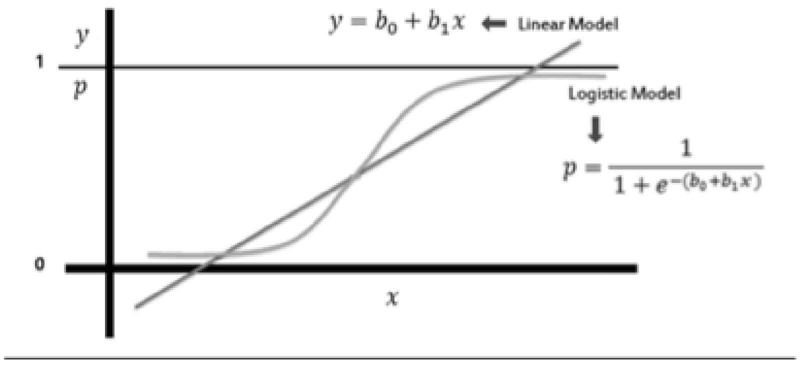
### 4.6.1 Logistic Regression

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)
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 4.5: 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.

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

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

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

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.

$$p = \frac{1}{1 + e^{-(b_0+b_1x_1+x_2b_2+\cdots+b_px_p)}}$$

## 4.7 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).

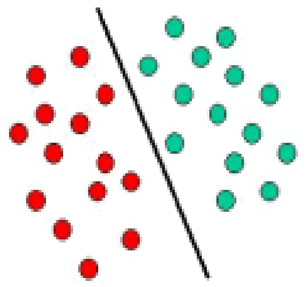


*Figure 4.6: Example of Linear Classifier*

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.

### 4.7.1 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 group of points for $y_i = 1$ which, which is 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 $\vec{w}$

### 4.7.2 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: , for . $k(\vec{x_i}, \vec{x_j}) = \exp(-\gamma \|\vec{x_i} - \vec{x_j}\|^2)$ $\gamma > 0$
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 $c < 0$

### 4.7.3 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.

### 4.7.4 Evaluation of binary classifiers

#### 4.7.4.1 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).

#### 4.7.4.2 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.

#### 4.7.4.3 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.

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

## 4.8 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?

### 4.8.1 ROC Curves

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

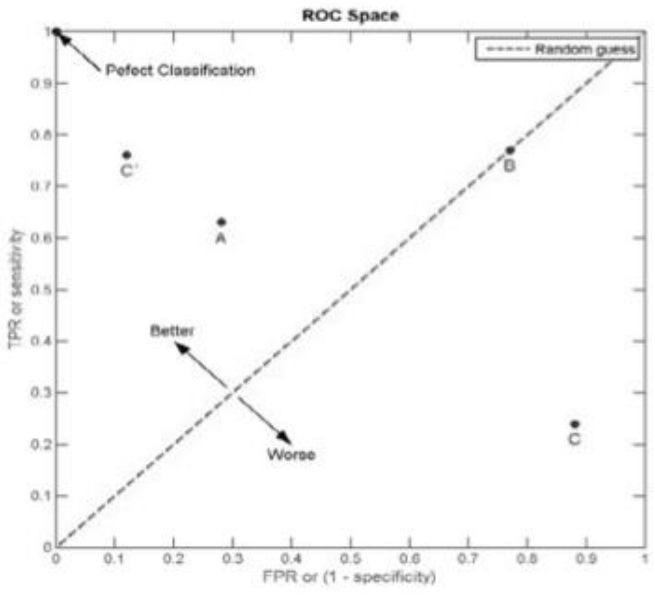


*Figure 4.7 ROC Space*

*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 back propagation 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 4.8 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.

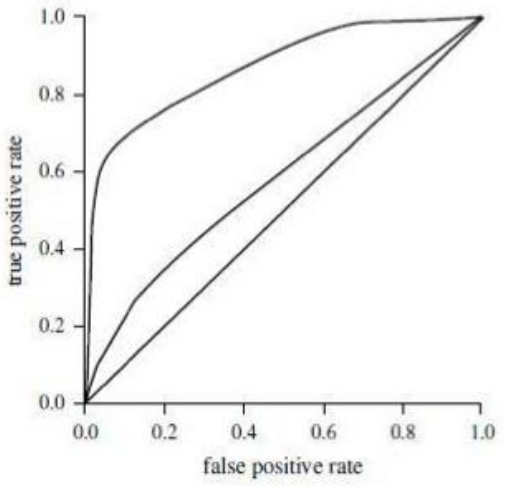


*Figure 4.8: The ROC curves of two classification models.*

# PRACTICAL IMPLEMENTATION

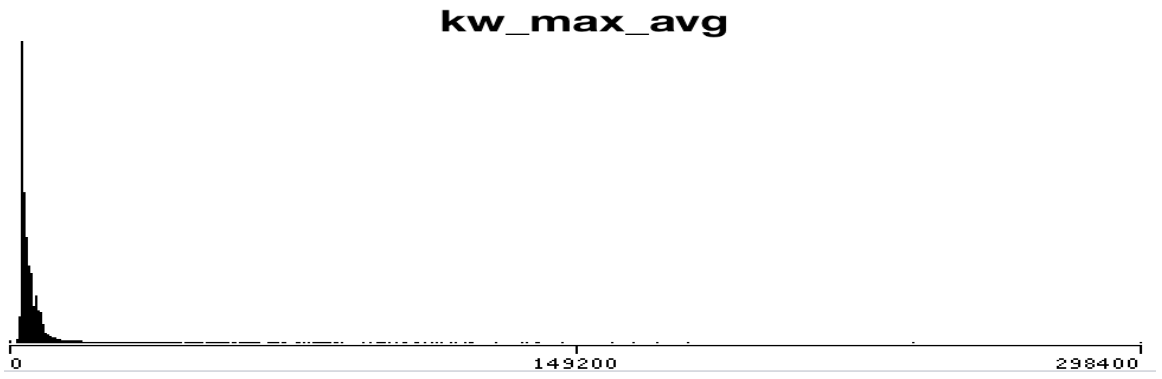
## 5.1.Dataset Analysis

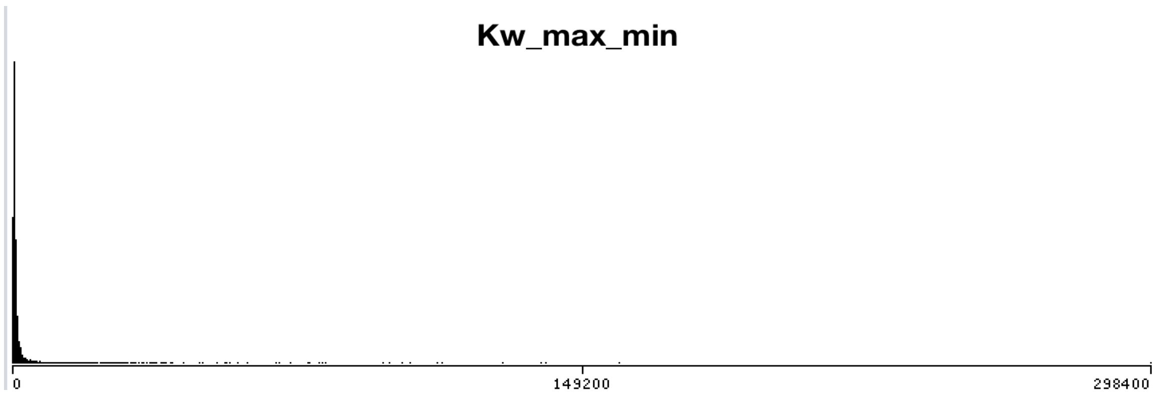
The initial Dataset consisted of 39,644 observations with 61 features collected over a 2 years period from Jan 2013 - Jan 2015 from mashable website.
After extensive analysis it was discovered that the data of numerous anomalies, for instance:

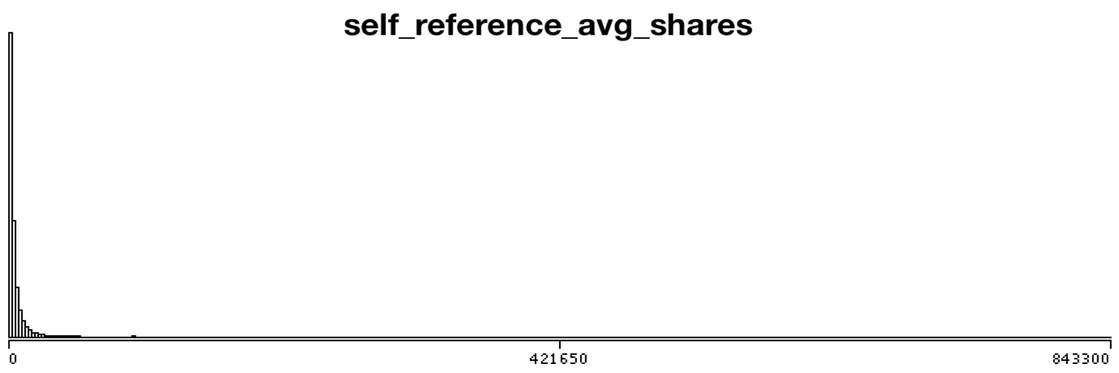
| Data Set | Website |
|---|---|
| 843,330 shares<br>12 videos<br>128 videos | 792 shares<br>0 videos<br>12 videos |

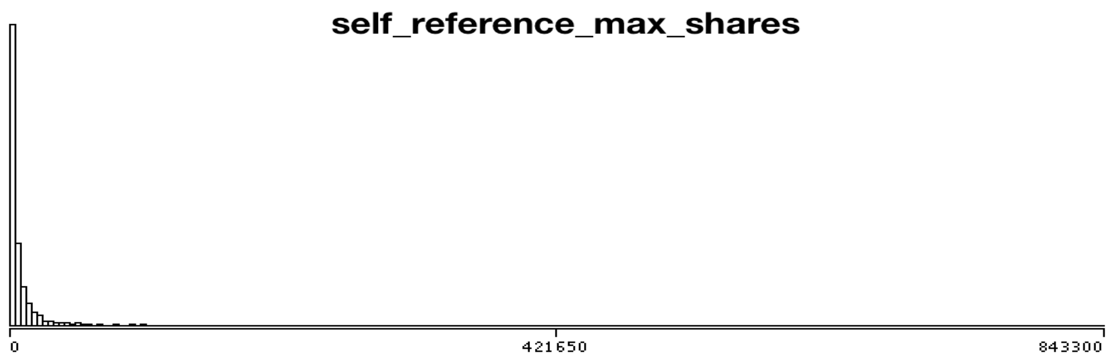
Fig.5.1: Comparison of Actual vs Recorded Data for an article(leaked: More Low Cost iPhone photos)

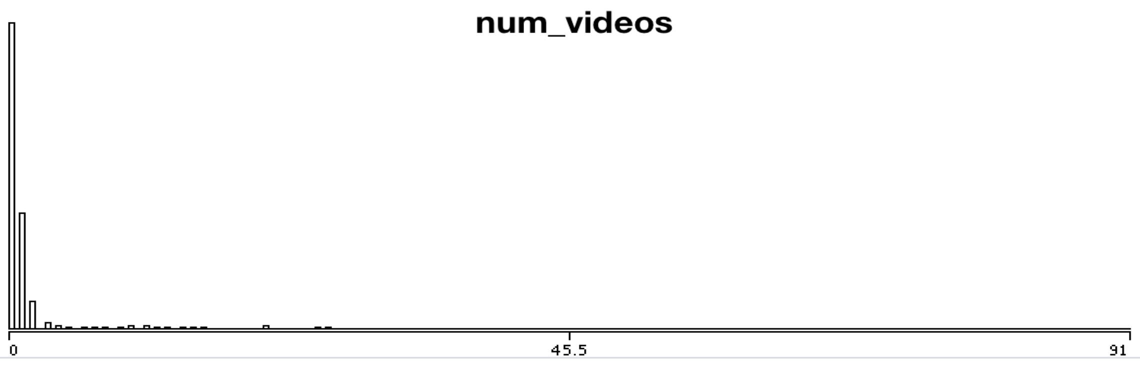
Frequency Distribution histogram are presented for some attributes showing outlier observations:

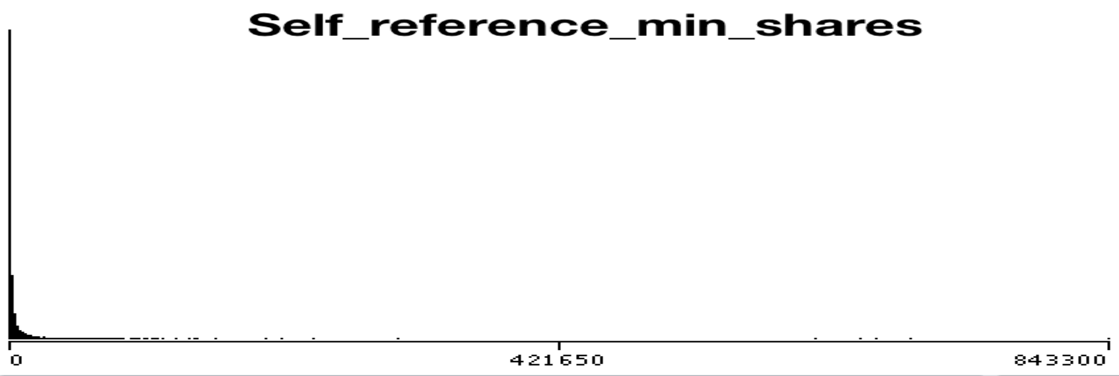



Kw_max_min
0
149200
298400


self_reference_avg_shares
0
421650
843300


self_reference_max_shares
0
421650
843300


num_videos
0
45.5
91



2

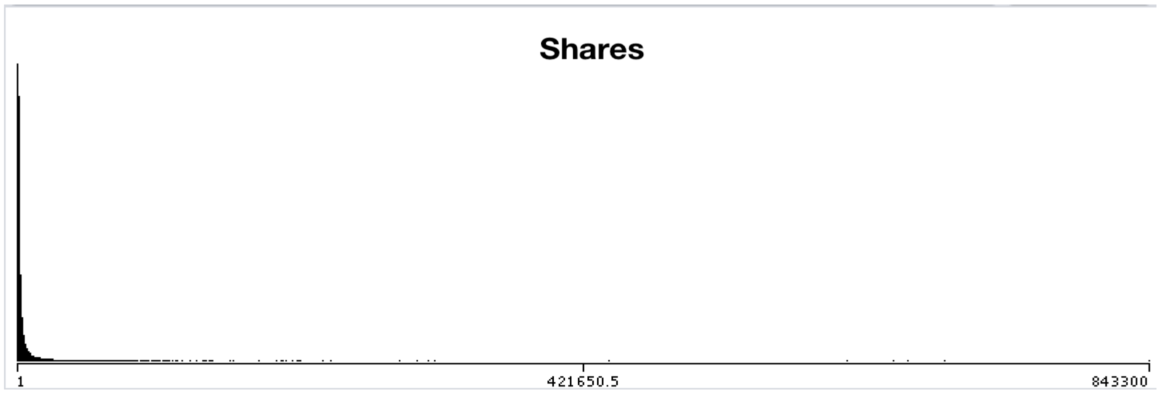


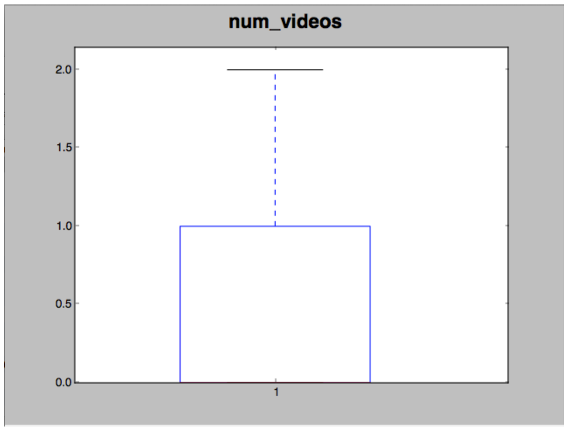
Figure 5.2 : Frequency Distribution histogram showing outlier observations

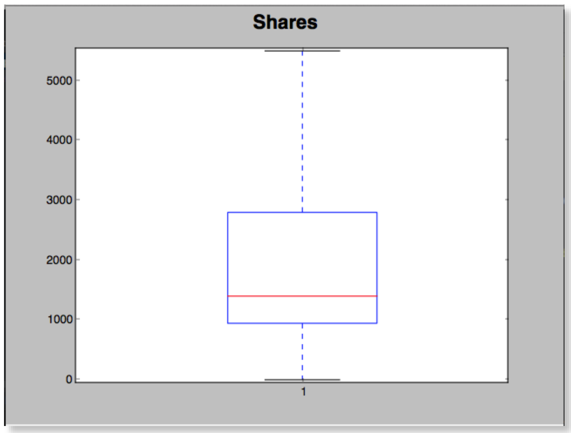
## 5.2 Data Preprocessing

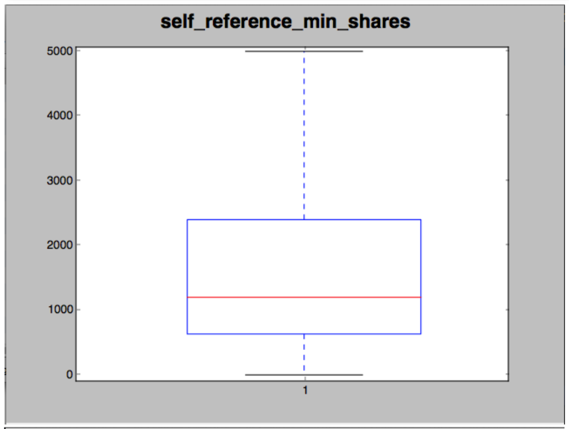
### Outlier Reduction

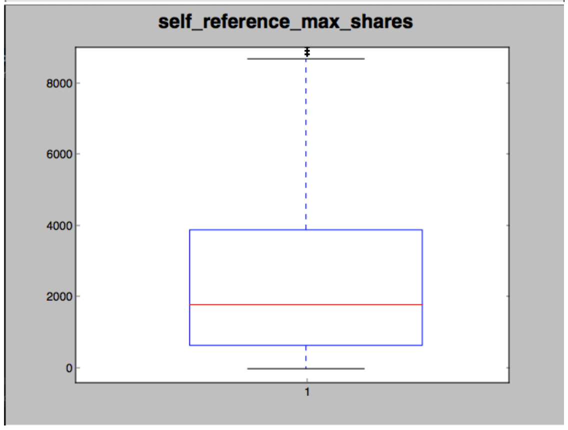
A **Boxplot** is a convenient way of graphically depicting groups of numerical data through their quartiles. Hence the aforementioned erroneous attributes were box plotted and observations containing outliers were removed.

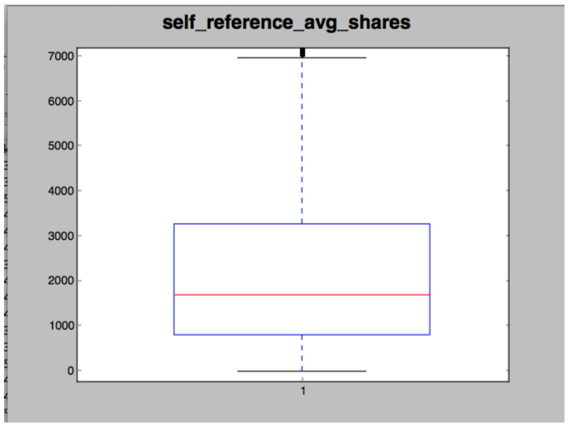
The dataset observations were reduced from 39,644 to 21,105 after removing outliers using boxplotting. The boxplots are as follows:




Shares
5000
4000
3000
2000
1000
0
1


self_reference_min_shares
5000
4000
3000
2000
1000
0
1


self_reference_max_shares
8000
6000
4000
2000
0
1


self_reference_avg_shares
7000
6000
5000
4000
3000
2000
1000
0
1

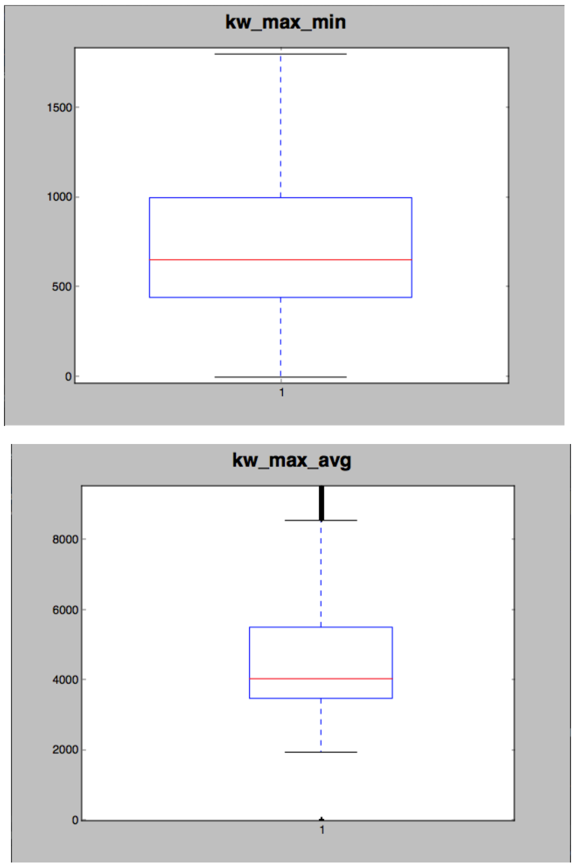
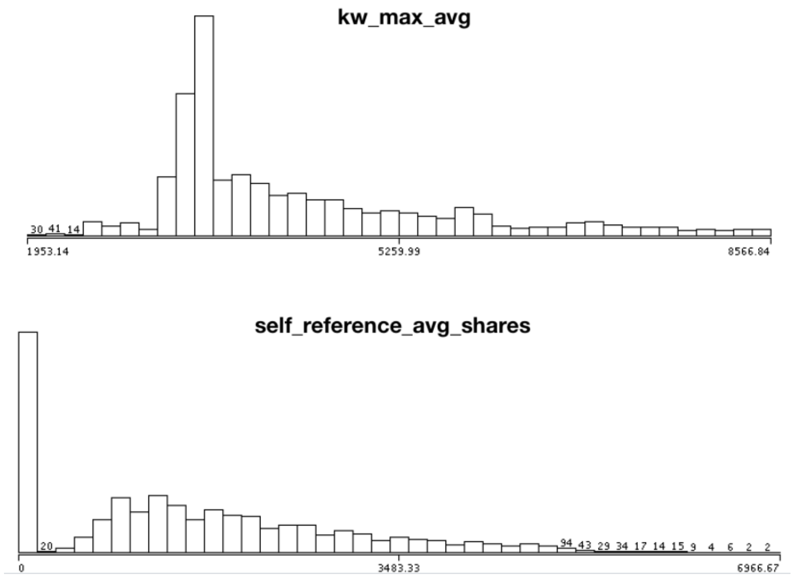


Figure 5.3 Boxplot of Original data attributes

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



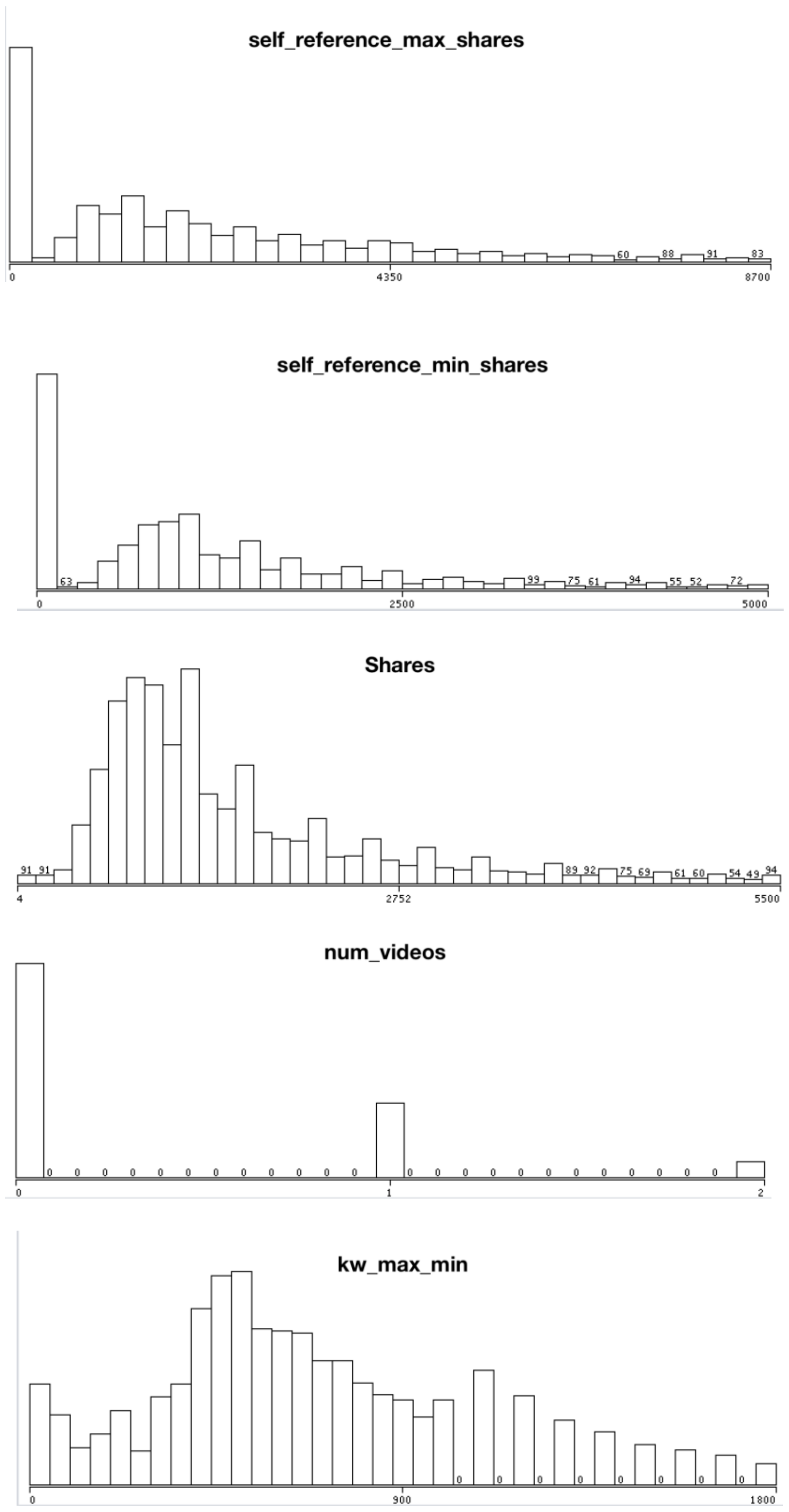


Fig 5.4: Frequency Distribution after Box Plot

## 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 5.5: 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

# 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:

- $\alpha = 0.2$

- number of iterations = 8000

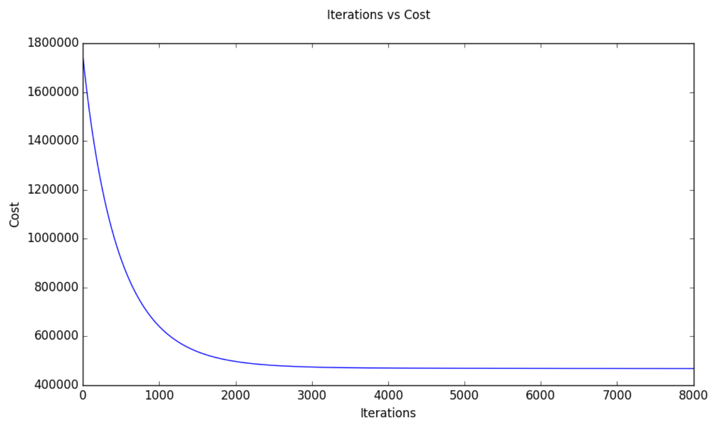


Fig 6.1: Minimization of Cost Function in Gradient Descent with Feature Selection

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

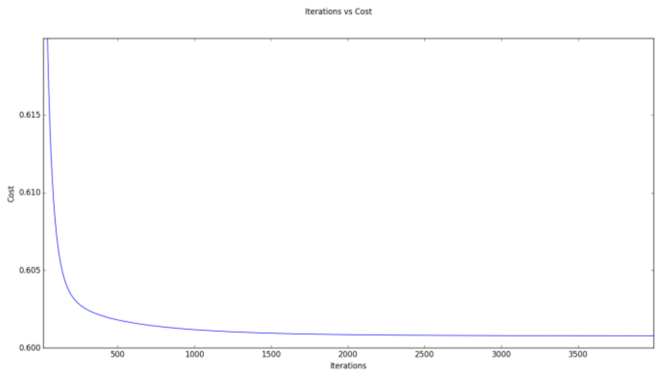


Fig 6.2: Minimization of Cost Function in Gradient Descent without Feature Selection

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:

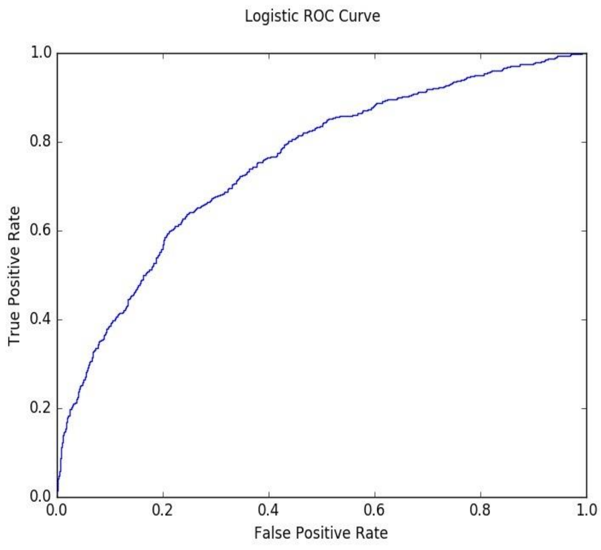


Fig 6.3: ROC Curve for Logistic Classifier without Feature Selection

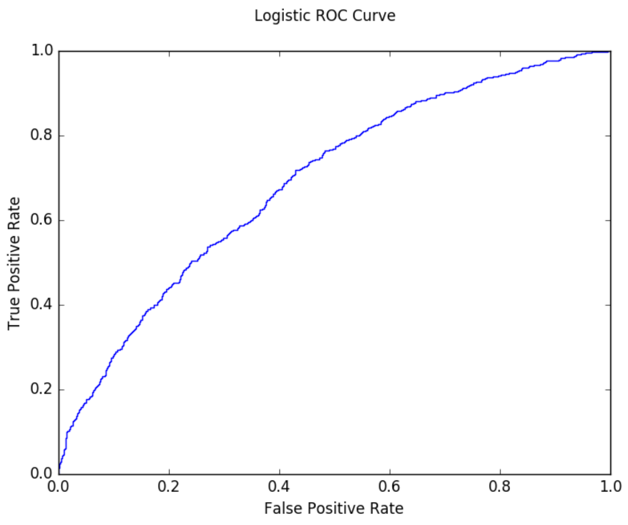


Fig 6.4: ROC Curve for Logistic Classifier with Feature Selection

1. Without Feature Selection

| | |
|---|---|
| Mean Absolute Error | 0.615069 |

Confusion Matrix

| | |
|---|---|
| Correctly Classified Instances: | **65.3556366393 %** |
| Incorrectly Classified Instances: | 34.6443633607 % |
| Mean Absolute Error: | 0.837596492743 |
| TP Rate/Recall: | 0.197193410616 |

| | |
|---|---|
| FP Rate: | 0.0567090176635 |
| Precision: | 0.688245315162 |
| AUC: | 0.697801976868 |

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

2. With feature Selection:

| | |
|---|---|
| Mean Absolute Error | 0.909651 |

Confusion Matrix

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

| | |
|---|---|
| Correctly Classified Instances: | **66.9146566839** % |
| Incorrectly Classified Instances: | 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:

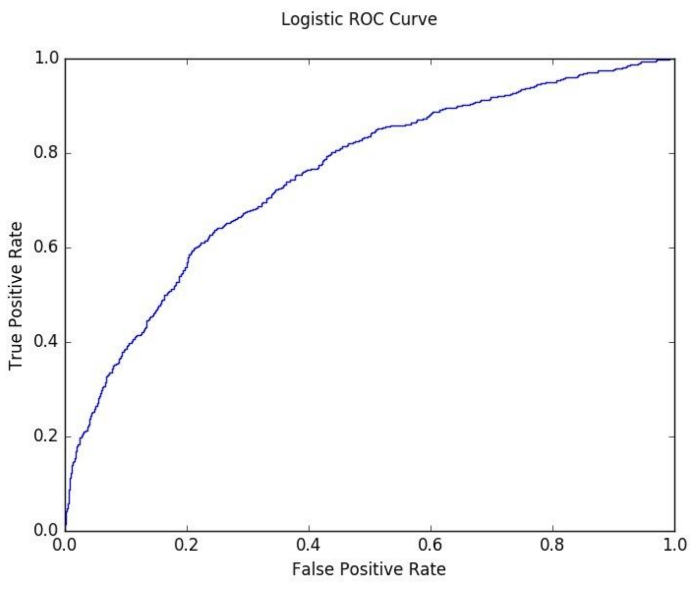


*Fig 6.5: ROC Curve for SVM Classifier with Feature Selection*

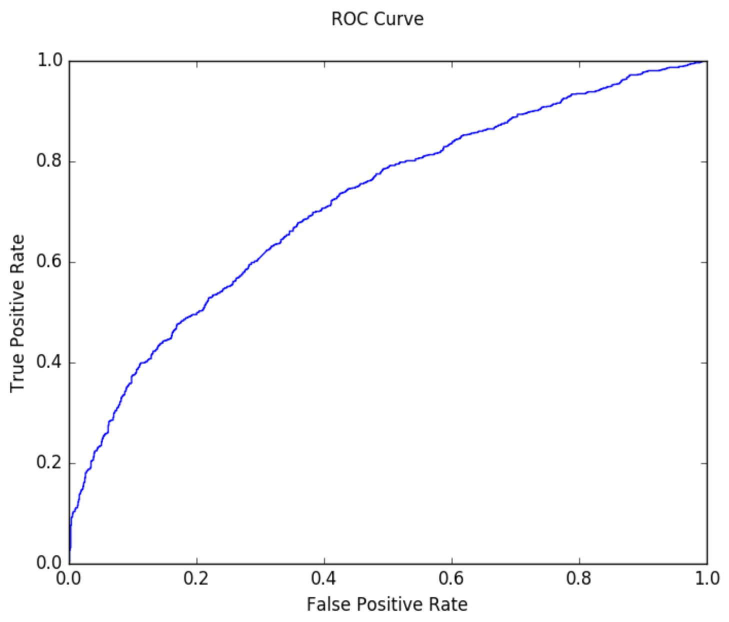


*Fig 6.6 : ROC Curve for SVM Classifier without Feature Selection*

1. Without Reduced Attributes

| | |
|---|---|
| Mean Absolute Error | 0.568637 |

Confusion Matrix

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

| | |
|---|---|
| Correctly Classified Instances | **67.4216935981 %** |
| Incorrectly Classified Instances | 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 |

Confusion Matrix

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

| | |
|---|---|
| Correctly Classified Instances: | **69.3029427096 %** |
| Incorrectly Classified Instances: | 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.

# RESULT and CONCLUSION

## 7.1 Results of Linear Regression

### 7.1.1 Without Feature Selection

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

### 7.1.2 Without 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

## 7.2 Results of Logistics Regression

### 7.2.1 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 |

### 7.2.2 Without Feature Selection

| | |
|---|---|
| Mean Absolute Error | **0.909651** |
| 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

## 7.3 Results of Support Vector Classification

### 7.3.1 Without Feature Selection

| | |
|---|---|
| Mean Absolute Error | 0.568637 |
| 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 |

### 7.3.2 Without Feature Selection

| | |
|---|---|
| Mean Absolute Error | 0.601971 |
| 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

## 7.4 Conclusion

The prediction of online news articles was done on the basis of various parameters like the category of news articles, the number of links in the article, title subjectivity etc, using the predictive models of machine learning, such as Linear Regression, Logistics Regression and Support Vector Machine. The script was implemented in Python. On the basis of these models and their corresponding predictions, we will be able to determine the the themes or genres of content that is trending at that point in time. This would help establish the best time to post an article, type of objectivity liked by the public. Also, this would be able to help news agencies modify their schedule in such a way that they can attract maximum viewership.

# 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.

## 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.