MINOR PROJECT

A COMBINED APPROACH OF SMOT DATA BALANCING WITH DATA MINING ALGORITHMS FOR DETECTING CREDIT CARD FRAUD

Submitted in partial fulfilment of the
Requirement for the award of
Degree of Bachelor of Technology
In
Computer Science and Engineering
(Batch - 2014-2018)

Under the Supervision of
Ms. Indu Singh
(Assistant Professor, CSE)

By:

Ajit Singh Kushwaha (2K14/CO/006)

Bablu Singh (2K14/CO/020)

Vaibhav Kumar Kashyap (2K14/CO/135)

Vivek (2K14/CO/144)

To:

Department of Computer Science and Engineering
Delhi Technological University
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road, Delhi-110042

Declaration

I hereby certify that the work which is presented in the Minor Project entitled "**A COMBINED APPROACH OF SMOT DATA BALANCING WITH DATA MINING ALGORITHMS FOR DETECTING CREDIT CARD FRAUD**" in fulfilment of the requirement for the award of Degree of Bachelor of Technology and submitted to the Department of Computer Engineering, Delhi Technological University (Formerly Delhi College of Engineering), New Delhi is an authentic record of my own, carried out during a period from January 2017 to May 2017, under the supervision of Ms. Indu Singh (Assistant Professor, CSE Department). The matter presented in this report has not been submitted by me for the award of any other degree of this or any other Institute/University.

Ajit Singh Kushwaha (2K14/CO/006)

Bablu Singh (2K14/CO/020)

Vaibhav Kumar Kashyap (2K14/CO/135)

Vivek (2K14/CO/144)

Supervisor Certificate

This is to certify that AJIT SINGH KUSHWAHA(2K14/CO/006), BABLU SINGH (2K14/CO/020), VAIBHAV KASHYAP(2K14/CO/135) and VIVEK (2K14/CO/144) the bonafide students of Bachelor of Technology in Computer Engineering of Delhi Technological University (Formerly Delhi College of Engineering), New Delhi of 2014-2018 batch has completed their minor project entitled "**A COMBINED APPROACH OF SMOTE DATA BALANCING WITH DATA MINING ALGORITHMS FOR DETECTING CREDIT CARD FRAUD**" under the supervision of Ms. Indu Singh (Assistant Professor, CSE Department). It is further certified that the work done in this dissertation is a result of candidate's own efforts.

I wish them all success in their life.

Date :

Ms. Indu Singh

Assistant Professor

CSE Department

Delhi Technological University

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Bawana Road, Delhi - 110042

Abstract

Credit card fraud is a serious and growing problem. While predictive models for credit card fraud detection are in active use in practice, reported studies on the use of data mining approaches for credit card fraud detection are relatively few, possibly due to the lack of available data for research. This paper evaluates two advanced data mining approaches, support vector machines and random forests, together with the well-known logistic regression, as part of an attempt to better detect (and thus control and prosecute) credit card fraud. The study is based on real-life data of transactions from an international credit card operation. Billions of dollars are lost annually due to credit card fraud. The 10th annual online fraud report by Cyber Source shows that although the percentage loss of revenues has been a steady 1.4% of online payments for the last three years (2006 to 2008), the actual amount has gone up due to growth in online sales. The estimated loss due to online fraud is $4 billion for 2008, an increase of 11% on the 2007 loss of $3.6 billion. With the growth in credit card transactions, as a share of the payment system, there has also been an increase in credit card fraud, and 70% of U.S. consumers are noted to be significantly concerned about identity fraud. Additionally, credit card fraud has broader ramifications, as such fraud helps fund organized crime, international narcotics trafficking, and even terrorist financing. Over the years, along with the evolution of fraud detection methods, perpetrators of fraud have also been evolving their fraud practices to avoid detection. Therefore, credit card fraud detection methods need constant innovation. In this study, we evaluate two advanced data mining approaches, support vector machines and random forests, together with the well-known logistic regression, as part of an attempt to better detect (and thus control and prosecute) credit card fraud. The study is based on real-life data of transactions from an international credit card operation.

Index

List of Figures

Chapter 1
Introduction

Credit cards are considered as a “nice target of fraud” since in a very short time attacker can get lots of money without much risk and most of the time the fraud is discovered after few days. To commit the credit card fraud either offline or online, fraudsters are looking for sensitive information such as credit card number, bank account and social security member. IN case of offline payment to perform the fraudulent transactions, an attacker has to steal the credit card itself, while in case of online payment, the fraudsters should be steal customer identity. Credit card fraud is a significant issue and has considerable cost for banks card issuer companies. Thus, with this massive problem in transaction system, banks take credit card fraud very seriously, and have highly sophisticated security system to monitor transactions and detect the fraud as quickly as possible once it is committed.

A secured and trusted banking payment system requires high speed verification and authentication mechanisms that let legitimate users easy conduct their business, while flagging and detecting suspicious transaction attempts by others. Fraud detection has become a vital activity in order to decrease the impact of fraudulent transaction on service directory, costs and reputation of the company.

According to Kount, one of the top five fraud detection consultant revealed by topcrditcardprocessors.com in the month of august 2013, 40% of the total financial fraud is related to credit card and loss of amount due to credit card fraud worldwide is $5.55 billion. There are various methods used for fraud detection each of them try to increase the detection rate while keeping false alarm rate at minimum. Different methods have been used for fraud detection including K- Nearest Neighbour(KNN), Support Vector Machine etc. Statistical fraud detection methods have been divided into two broad categories; supervised and unsupervised. In supervised fraud detection methods, models are estimated based on the sample of fraudulent and legitimate transaction, to classify new transaction as fraudulent or legitimate. While in supervised fraud detection, outliers or unusual transaction are identified as potential case of fraudulent transaction. Both these fraud detection methods predict the probability of fraud in any given transaction.

Credit card fraud is increasing considerably with the development of modern technology and the global superhighways of communication. Credit card fraud costs consumers and the financial company billions of dollars annually, and fraudsters continuously try to find new rules and tactical to commit illegal actions. Thus, fraud detection system has become essential for banks and financial institution, to minimize their losses. However, there is a lack of published literature on credit card fraud detection technique, due to the unavailable credit card transaction dataset for researchers. The most commonly techniques used construct the fraud detection model. The performance evaluation is performed on real life credit card truncation dataset to demonstrate the benefit of the bagging ensemble algorithm.

Fraud detection method are Support Vector Machines(SVM), K-Nearest Neighbour algorithm(KNN). These techniques can be used alone or in collaboration using ensemble or meta learning techniques to build classifiers. But amongst all existing method, ensemble learning methods are identified as populate and common method, not because of its quite straightforward implementation, but also due to its exceptional predictive performance on practical problems. In this year paper we trained various data mining techniques used in credit card fraud detection and evaluate each methodology based on certain design criteria. After several trial and comparisons; we introduced the bagging classifier based on decision tree, as the best classifier to construct the fraud detection model. The performance evaluation is performed on real life credit card transaction dataset to demonstrate the benefit of the bagging ensemble algorithm.

Chapter 2
Data Mining

Overview

Data mining, *the extraction of hidden predictive information from large databases*, is a powerful new technology with great potential to help companies focus on the most important information in their data warehouses. Data mining tools predict future trends and behaviours, allowing businesses to make proactive, knowledge-driven decisions. The automated, prospective analyses offered by data mining move beyond the analyses of past events provided by retrospective tools typical of decision support systems. Data mining tools can answer business questions that traditionally were too time consuming to resolve. They scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations.

Most companies already collect and refine massive quantities of data. Data mining techniques can be implemented rapidly on existing software and hardware platforms to enhance the value of existing information resources, and can be integrated with new products and systems as they are brought on-line. When implemented on high performance client/server or parallel processing computers, data mining tools can analyse massive databases to deliver answers to questions such as, "Which clients are most likely to respond to my next promotional mailing, and why?"

This white paper provides an introduction to the basic technologies of data mining. Examples of profitable applications illustrate its relevance to today's business environment as well as a basic description of how data warehouse architectures can evolve to deliver the value of data mining to end users.

The Foundations of Data Mining

Data mining techniques are the result of a long process of research and product development. This evolution began when business data was first stored on computers, continued with improvements in data access, and more recently, generated technologies that allow users to navigate through their data in real time. Data mining takes this evolutionary process beyond retrospective data access and navigation to prospective and proactive information delivery.

Data mining is ready for application in the business community because it is supported by three technologies that are now sufficiently mature:

- Massive data collection
- Powerful multiprocessor computers
- Data mining algorithms

Commercial databases are growing at unprecedented rates. A recent META Group survey of data warehouse projects found that 19% of respondents are beyond the 50 gigabyte level, while 59% expect to be there by second quarter of 1996.1 In some industries, such as retail, these numbers can be much larger. The accompanying need for improved computational engines can now be met in a cost-effective manner with parallel multiprocessor computer technology. Data mining algorithms embody techniques that have existed for at least 10 years, but have only recently been implemented as mature, reliable, understandable tools that consistently outperform older statistical methods.

In the evolution from business data to business information, each new step has built upon the previous one. For example, dynamic data access is critical for drill-through in data navigation applications, and the ability to store large databases is critical to data mining. From the user's point of view, the four steps listed in Table 1 were revolutionary because they allowed new business questions to be answered accurately and quickly.

The core components of data mining technology have been under development for decades, in research areas such as statistics, artificial intelligence, and machine learning. Today, the maturity of these techniques, coupled with high-performance relational database engines and broad data integration efforts, make these technologies practical for current data warehouse environments.

Evolutionary Step	Business Question	Enabling Technologies	Product Providers	Characteristics
Data Collection (1960s)	"What was my total revenue in the last five years?"	Computers, tapes, disks	IBM, CDC	Retrospective, static data delivery
Data Access (1980s)	"What were unit sales in New England last March?"	Relational databases (RDBMS), Structured Query Language (SQL), ODBC	Oracle, Sybase, Informix, IBM, Microsoft	Retrospective, dynamic data delivery at record level
Data Warehousing & Decision Support (1990s)	"What were unit sales in New England last March? Drill down to Boston."	On-line analytic processing (OLAP), multidimensional databases, data warehouses	Pilot, Comshare, Arbor, Cognos, Microstrategy	Retrospective, dynamic data delivery at multiple levels
Data Mining (Emerging Today)	"What's likely to happen to Boston unit sales next month? Why?"	Advanced algorithms, multiprocessor computers, massive databases	Pilot, Lockheed, IBM, SGI, numerous startups (nascent industry)	Prospective, proactive information delivery

Table 1. Steps in the Evolution of Data Mining.

The Scope of Data Mining

Data mining derives its name from the similarities between searching for valuable business information in a large database — for example, finding linked products in gigabytes of store scanner data — and mining a mountain for a vein of valuable ore. Both processes require either sifting through an immense amount of material, or intelligently probing it to find exactly where the value resides. Given databases of sufficient size and quality, data mining technology can generate new business opportunities by providing these capabilities:

- ***Automated prediction of trends and behaviours***. Data mining automates the process of finding predictive information in large databases. Questions that traditionally required extensive hands-on analysis can now be answered directly from the data — quickly. A typical example of a predictive problem

is targeted marketing. Data mining uses data on past promotional mailings to identify the targets most likely to maximize return on investment in future mailings. Other predictive problems include forecasting bankruptcy and other forms of default, and identifying segments of a population likely to respond similarly to given events.

- ***Automated discovery of previously unknown patterns .*** Data mining tools sweep through databases and identify previously hidden patterns in one step. An example of pattern discovery is the analysis of retail sales data to identify seemingly unrelated products that are often purchased together. Other pattern discovery problems include detecting fraudulent credit card transactions and identifying anomalous data that could represent data entry keying errors.

Data mining techniques can yield the benefits of automation on existing software and hardware platforms, and can be implemented on new systems as existing platforms are upgraded and new products developed. When data mining tools are implemented on high performance parallel processing systems, they can analyze massive databases in minutes. Faster processing means that users can automatically experiment with more models to understand complex data. High speed makes it practical for users to analyze huge quantities of data. Larger databases, in turn, yield improved predictions.

Databases can be larger in both depth and breadth:

- ***More columns.*** Analysts must often limit the number of variables they examine when doing hands-on analysis due to time constraints. Yet variables that are discarded because they seem unimportant may carry information about unknown patterns. High performance data mining allows users to explore the full depth of a database, without preselecting a subset of variables.

- ***More rows.*** Larger samples yield lower estimation errors and variance, and allow users to make inferences about small but important segments of a population.

A recent Gartner Group Advanced Technology Research Note listed data mining and artificial intelligence at the top of the five key technology areas that "will clearly have a major impact across a wide range of industries within the next 3 to 5 years."2 Gartner also listed parallel architectures and data mining as two of the top 10 new technologies in which companies will invest during the next 5 years. According to a recent Gartner HPC Research Note, "With the rapid advance in data capture, transmission and storage, large-systems users will increasingly need to implement new and innovative ways to mine the after-market value of their vast stores of detail data, employing MPP [massively parallel processing] systems to create new sources of business advantage (0.9 probability)."3

The most commonly used techniques in data mining are:

- ***Artificial neural networks :*** Non-linear predictive models that learn through training and resemble biological neural networks in structure.

- ***Decision trees:*** Tree-shaped structures that represent sets of decisions. These decisions generate rules for the classification of a dataset. Specific decision tree methods include Classification and Regression Trees (CART) and Chi Square Automatic Interaction Detection (CHAID) .

- ***Genetic algorithms:*** Optimization techniques that use processes such as genetic combination, mutation, and natural selection in a design based on the concepts of evolution.

- ***Nearest neighbor method:*** A technique that classifies each record in a dataset based on a combination of the classes of the k record(s) most similar to it in a historical dataset (where k ³ 1). Sometimes called the k-nearest neighbor technique.

- ***Rule induction:*** The extraction of useful if-then rules from data based on statistical significance.

Many of these technologies have been in use for more than a decade in specialized analysis tools that work with relatively small volumes of data. These capabilities are now evolving to integrate directly with industry-standard data warehouse and OLAP platforms. The appendix to this white paper provides a glossary of data mining terms.

How Data Mining Works ?

How exactly is data mining able to tell you important things that you didn't know or what is going to happen next? The technique that is used to perform these feats in data mining is called modeling. Modeling is simply the act of building a model in one situation where you know the answer and then applying it to another situation that you don't. For instance, if you were looking for a sunken Spanish galleon on the high seas the first thing you might do is to research the times when Spanish treasure had been found by others in the past. You might note that these ships often tend to be found off the coast of Bermuda and that there are certain characteristics to the ocean currents, and certain routes that have likely been taken by the ship’s captains in that era. You note these similarities and build a model that includes the characteristics that are common to the locations of these sunken treasures. With these models in hand you sail off looking for treasure where your model indicates it most likely might be given a similar situation in the past. Hopefully, if you've got a good model, you find your treasure.

This act of model building is thus something that people have been doing for a long time, certainly before the advent of computers or data mining technology. What happens on computers, however, is not much different than the way people build

models. Computers are loaded up with lots of information about a variety of situations where an answer is known and then the data mining software on the computer must run through that data and distill the characteristics of the data that should go into the model. Once the model is built it can then be used in similar situations where you don't know the answer. For example, say that you are the director of marketing for a telecommunications company and you'd like to acquire some new long distance phone customers. You could just randomly go out and mail coupons to the general population - just as you could randomly sail the seas looking for sunken treasure. In neither case would you achieve the results you desired and of course you have the opportunity to do much better than random - you could use your business experience stored in your database to build a model.

As the marketing director you have access to a lot of information about all of your customers: their age, sex, credit history and long distance calling usage. The good news is that you also have a lot of information about your prospective customers: their age, sex, credit history etc. Your problem is that you don't know the long distance calling usage of these prospects (since they are most likely now customers of your competition). You'd like to concentrate on those prospects who have large amounts of long distance usage. You can accomplish this by building a model. Table 2 illustrates the data used for building a model for new customer prospecting in a data warehouse.

	Customers	Prospects
General information (e.g. demographic data)	Known	Known
Proprietary information (e.g. customer transactions)	Known	Target

Table 2 - Data Mining for Prospecting

The goal in prospecting is to make some calculated guesses about the information in the lower right hand quadrant based on the model that we build going from Customer General Information to Customer Proprietary Information. For instance, a simple model for a telecommunications company might be:

98% of my customers who make more than $60,000/year spend more than $80/month on long distance

This model could then be applied to the prospect data to try to tell something about the proprietary information that this telecommunications company does not

currently have access to. With this model in hand new customers can be selectively targeted.

Test marketing is an excellent source of data for this kind of modeling. Mining the results of a test market representing a broad but relatively small sample of prospects can provide a foundation for identifying good prospects in the overall market. Table 3 shows another common scenario for building models: predict what is going to happen in the future.

	Yesterday	Today	Tomorrow
Static information and current plans (e.g. demographic data, marketing plans)	Known	Known	Known
Dynamic information (e.g. customer transactions)	Known	Known	Target

Table 3 - Data Mining for Predictions

If someone told you that he had a model that could predict customer usage how would you know if he really had a good model? The first thing you might try would be to ask him to apply his model to your customer base - where you already knew the answer. With data mining, the best way to accomplish this is by setting aside some of your data in a vault to isolate it from the mining process. Once the mining is complete, the results can be tested against the data held in the vault to confirm the model's validity. If the model works, its observations should hold for the vaulted data.

An Architecture for Data Mining

To best apply these advanced techniques, they must be fully integrated with a data warehouse as well as flexible interactive business analysis tools. Many data mining tools currently operate outside of the warehouse, requiring extra steps for extracting, importing, and analyzing the data. Furthermore, when new insights require operational implementation, integration with the warehouse simplifies the application of results from data mining. The resulting analytic data warehouse can be applied to improve business processes throughout the organization, in areas such as promotional campaign management, fraud detection, new product rollout, and so on. Figure 1 illustrates an architecture for advanced analysis in a large data warehouse.

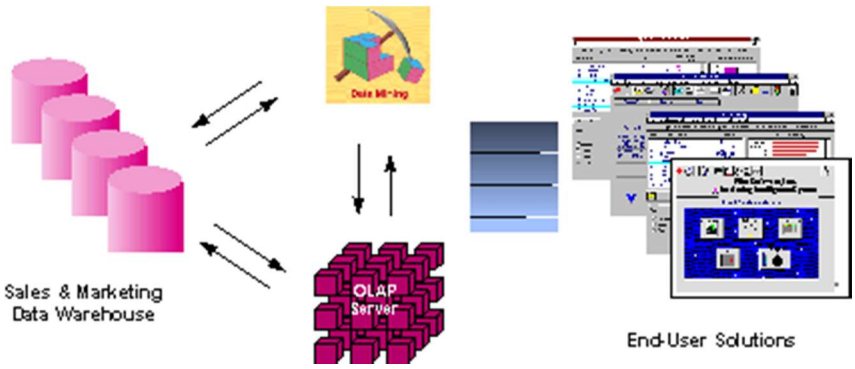


Figure 1 - Integrated Data Mining Architecture

The ideal starting point is a data warehouse containing a combination of internal data tracking all customer contact coupled with external market data about competitor activity. Background information on potential customers also provides an excellent basis for prospecting. This warehouse can be implemented in a variety of relational database systems: Sybase, Oracle, Redbrick, and so on, and should be optimized for flexible and fast data access.

An OLAP (On-Line Analytical Processing) server enables a more sophisticated end-user business model to be applied when navigating the data warehouse. The multidimensional structures allow the user to analyze the data as they want to view their business – summarizing by product line, region, and other key perspectives of their business. The Data Mining Server must be integrated with the data warehouse and the OLAP server to embed ROI-focused business analysis directly into this infrastructure. An advanced, process-centric metadata template defines the data mining objectives for specific business issues like campaign management, prospecting, and promotion optimization. Integration with the data warehouse enables operational decisions to be directly implemented and tracked. As the warehouse grows with new decisions and results, the organization can continually mine the best practices and apply them to future decisions.

This design represents a fundamental shift from conventional decision support systems. Rather than simply delivering data to the end user through query and reporting software, the Advanced Analysis Server applies users' business models directly to the warehouse and returns a proactive analysis of the most relevant information. These results enhance the metadata in the OLAP Server by providing a dynamic metadata layer that represents a distilled view of the data. Reporting, visualization, and other analysis tools can then be applied to plan future actions and confirm the impact of those plans.

Profitable Applications

A wide range of companies have deployed successful applications of data mining. While early adopters of this technology have tended to be in information-intensive industries such as financial services and direct mail marketing, the technology is applicable to any company looking to leverage a large data warehouse to better manage their customer relationships. Two critical factors for success with data mining are: a large, well-integrated data warehouse and a well-defined understanding of the business process within which data mining is to be applied (such as customer prospecting, retention, campaign management, and so on).

Some successful application areas include:

- A pharmaceutical company can analyze its recent sales force activity and their results to improve targeting of high-value physicians and determine which marketing activities will have the greatest impact in the next few months. The data needs to include competitor market activity as well as information about the local health care systems. The results can be distributed to the sales force via a wide-area network that enables the representatives to review the recommendations from the perspective of the key attributes in the decision process. The ongoing, dynamic analysis of the data warehouse allows best practices from throughout the organization to be applied in specific sales situations.
- A credit card company can leverage its vast warehouse of customer transaction data to identify customers most likely to be interested in a new credit product. Using a small test mailing, the attributes of customers with an affinity for the product can be identified. Recent projects have indicated more than a 20-fold decrease in costs for targeted mailing campaigns over conventional approaches.
- A diversified transportation company with a large direct sales force can apply data mining to identify the best prospects for its services. Using data mining to analyze its own customer experience, this company can build a unique segmentation identifying the attributes of high-value prospects. Applying this segmentation to a general business database such as those provided by Dun & Bradstreet can yield a prioritized list of prospects by region.
- A large consumer package goods company can apply data mining to improve its sales process to retailers. Data from consumer panels, shipments, and competitor activity can be applied to understand the reasons for brand and store switching. Through this analysis, the manufacturer can select promotional strategies that best reach their target customer segments.

Each of these examples have a clear common ground. They leverage the knowledge about customers implicit in a data warehouse to reduce costs and improve the value of customer relationships. These organizations can now focus their efforts on the most important (profitable) customers and prospects, and design targeted marketing strategies to best reach them.

Conclusion

Comprehensive data warehouses that integrate operational data with customer, supplier, and market information have resulted in an explosion of information. Competition requires timely and sophisticated analysis on an integrated view of the data. However, there is a growing gap between more powerful storage and retrieval systems and the users' ability to effectively analyze and act on the information they contain. Both relational and OLAP technologies have tremendous capabilities for navigating massive data warehouses, but brute force navigation of data is not enough. A new technological leap is needed to structure and prioritize information for specific end-user problems. The data mining tools can make this leap. Quantifiable business benefits have been proven through the integration of data mining with current information systems, and new products are on the horizon that will bring this integration to an even wider audience of users.

Chapter 3

Data Pre-processing

Introduction

Data pre-processing is an often neglected but important step in the data mining process. The phrase "Garbage In, Garbage Out" is particularly applicable to data mining and machine learning. Data gathering methods are often loosely controlled, resulting in out-of- range values (e.g., Income: -100), impossible data combinations (e.g., Gender: Male, Pregnant: Yes), missing values, etc. Analysing data that has not been carefully screened for such problems can produce misleading results. Thus, the representation and quality of data is first and foremost before running an analysis.

If there is much irrelevant and redundant information present or noisy and unreliable data, then knowledge discovery during the training phase is more difficult. Data preparation and filtering steps can take considerable amount of processing time. Data pre-processing includes cleaning, normalization, transformation, feature extraction and selection, etc. The product of data pre-processing is the final training set.

Data Pre-processing Models

Raw data is highly susceptible to noise, missing values, and inconsistency. The quality of data affects the data mining results. In order to help improve the quality of the data and, consequently, of the mining results raw data is pre-processed so as to improve the efficiency and ease of the mining process. Data pre-processing is one of the most critical steps in a data mining process which deals with the preparation and transformation of the initial dataset.

Data pre-processing methods are divided into following categories:

- Data Cleaning
- Data Integration
- Data Transformation
- Data Reduction

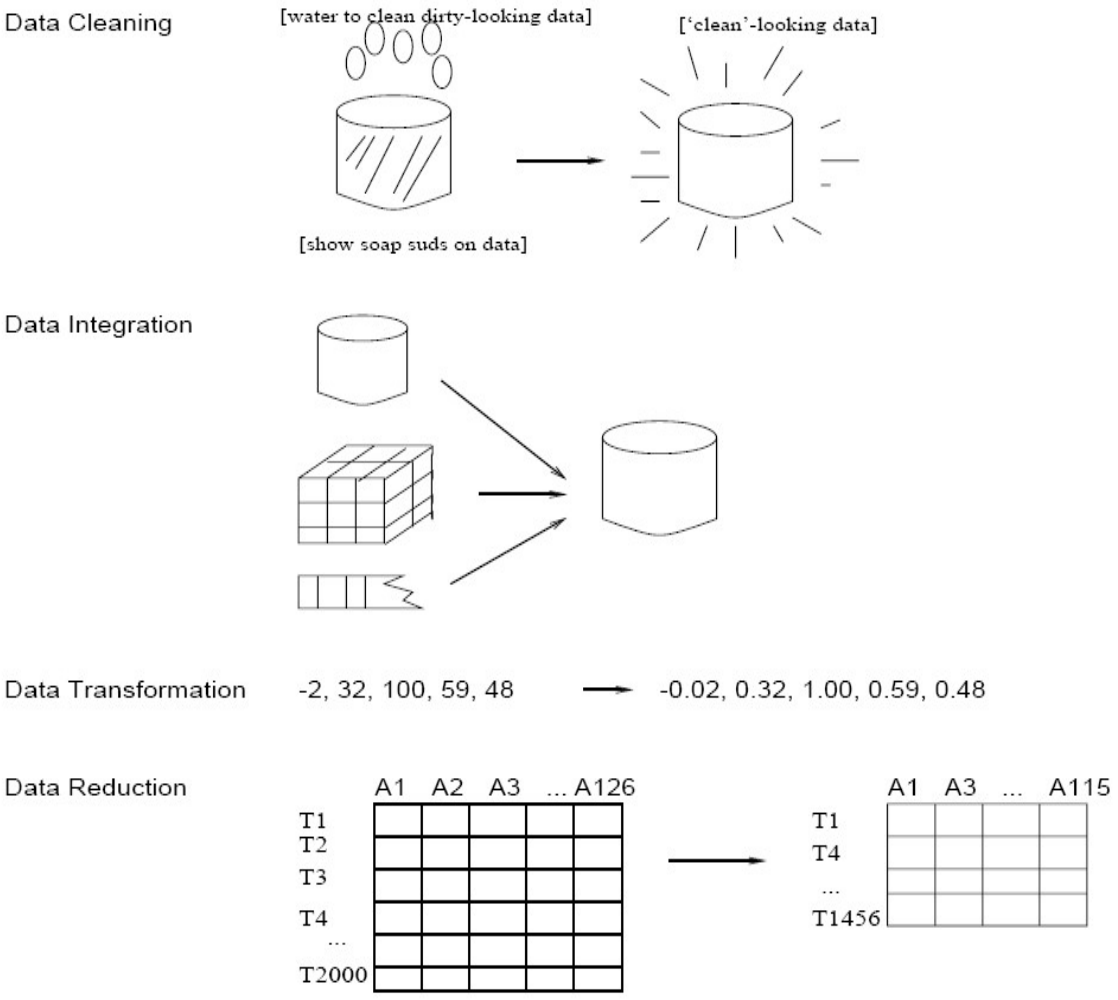


Figure 2: Forms of Data Pre-processing

Data Cleaning

Data that is to be analyse by data mining techniques can be incomplete (lacking attribute values or certain attributes of interest, or containing only aggregate data), noisy (containing errors, or outlier values which 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. Data can be noisy, having incorrect attribute values, owing to the following. 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. Duplicate tuples also require data cleaning. Data cleaning routines work to “clean" the data by filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies. Dirty data can cause confusion for the mining procedure. Although most mining routines have some procedures for dealing with incomplete or noisy data, they are not always robust. Instead, they may concentrate on avoiding over fitting the data to the function being modelled. Therefore, a useful pre-processing step is to run your data through some data cleaning routines.

Missing Values

If it is noted that there are many tuples that have no recorded value for several attributes, then the missing values can be filled in for the attribute by various methods described below:

1. **Ignore the tuple:** This is usually done when the class label is missing (assuming the mining task involves classification or description). This method is not very effective, unless the tuple contains several attributes with missing values. It is especially poor when the percentage of missing values per attribute varies considerably.

2. **Fill in the missing value manually:** In general, this approach is time-consuming and may not be feasible given a large data set with many missing values.

3. **Use a global constant to fill in the missing value :** Replace all missing attribute values by the same constant, such as a label like \Unknown", or -∞. If missing values are replaced by, say, \Unknown", then the mining program may mistakenly think that they form an interesting concept, since they all have a value in common | that of \Unknown". Hence, although this method is simple, it is not recommended.

4. **Use the attribute mean to fill in the missing value**

5. **Use the attribute mean for all samples belonging to the same class as the given tuple.**

6. **Use the most probable value to fill in the missing value: This may be determined with inference-based tools using a Bayesian formalism or decision tree induction.**

Methods 3 to 6 bias the data. The filled-in value may not be correct. 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.

Noisy Data

Noise is a random error or variance in a measured variable. Given a numeric attribute such as, say, price, how can the data be “smoothed" to remove the noise? The following data smoothing techniques describes this.

1. **Binning methods:** Binning methods smooth a sorted data value by consulting the

   \neighbourhood", or 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 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.

2. **Clustering:** Outliers may be detected by clustering, where similar values are organized into groups or \clusters".

3. **Combined computer and human inspection :** Outliers may be identified through a combination of computer and human inspection. In one application, for example, an information-theoretic measure was used to help identify outlier patterns in a handwritten character database for classification. The measure's value reflected the

   \surprise" content of the predicted character label with respect to the known label.

   Outlier patterns may be informative (e.g., identifying useful data exceptions, such as different versions of the characters \0" or \7"), or \garbage" (e.g., mislabelled characters). Patterns whose surprise content is above a threshold are output to a list. A human can then sort through the patterns in the list to identify the actual garbage ones. This is much faster than having to manually search through the entire database. The garbage patterns can then be removed from the (training) database.

4. **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 variables, so that one variable can be used to predict the other. Multiple linear regression is an extension of linear regression, where more than two variables are involved and the data are fit to a multidimensional surface. Using regression to find a mathematical equation to fit the data helps smooth out the noise.

Inconsistent data

There may be inconsistencies in the data recorded for some transactions. Some data inconsistencies may be corrected manually using external references. For example, errors made at data entry may be corrected by performing a paper trace. This may be coupled with routines designed to help correct the inconsistent use of codes. Knowledge engineering tools may also be used to detect the violation of known data constraints. For example, known functional dependencies between attributes can be used to find values contradicting the functional constraints.

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, as in data warehousing. These sources may include multiple databases, data cubes, or flat files. There are a number of issues to consider during data integration. Schema integration can be tricky. How can like real world entities from multiple data sources be 'matched up'? This is referred to as the entity identification

problem. For example, how can the data analyst or the computer be sure that customer id in one database, and cust_number in another refer to the same entity? Databases and data warehouses typically have metadata - that is, data about the data. Such metadata can be used to help avoid errors in schema integration. Redundancy is another important issue. An attribute may be redundant if it can be \derived" from another table, such as annual revenue. Inconsistencies in attribute or dimension naming can also cause redundancies in the resulting data set.

Data Transformation

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

1. 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 to 1.0.

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

3. 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.

4. 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 county. Similarly, values for numeric attributes,like age, may be mapped to higher level concepts, like young, middle-aged, and senior.

Data Reduction

Complex data analysis and mining on huge amounts of data may take a very long time, making such analysis impractical or infeasible. Data reduction techniques have been helpful in analyzing reduced representation of the dataset without compromising the integrity of the original data and yet producing the quality knowledge. The concept of data reduction is commonly understood as either reducing the volume or reducing the dimensions (number of attributes). There are a number of methods that have facilitated in analyzing a reduced volume or dimension of data and yet yield useful knowledge. Certain partition based methods work on partition of data tuples.That is, mining on the reduced data set should be more efficient yet produce the same (or almost the same) analytical results

Strategies for data reduction include -

1. Data cube aggregation, where aggregation operations are applied to the data in the construction of a data cube.

2. Dimension reduction, where irrelevant, weakly relevant, or redundant attributes or dimensions may be detected and removed.

3. Data compression, where encoding mechanisms are used to reduce the data set size. The methods used for data compression are wavelet transform and Principal Component Analysis.

4. Numerosity reduction, where the data are replaced or estimated by alternative, smaller data representations such as parametric models (which need store only the model parameters instead of the actual data e.g. regression and log-linear models), or nonparametric methods such as clustering, sampling, and the use of histograms.

5. Discretization and concept hierarchy generation, where raw data values for attributes are replaced by ranges or higher conceptual levels. Concept hierarchies allow the mining of data at multiple levels of abstraction, and are a powerful tool for data mining.

Conclusion

Data preparation is an important issue for both data warehousing and data mining, as real- world data tends to be incomplete, noisy, and inconsistent. Data preparation includes data cleaning, data integration, data transformation, and data reduction. Data cleaning routines can be used to fill in missing values, smooth noisy data, identify outliers, and correct data inconsistencies. Data integration combines data from multiples sources to form a coherent data store. Metadata, correlation analysis, data conflict detection, and the resolution of semantic heterogeneity contribute towards smooth data integration. Data transformation routines conform the data into appropriate forms for mining. For example, attribute data may be normalized so as to fall between a small range, such as 0 to 1.0. Data reduction techniques such as data cube aggregation, dimension reduction, data compression, numerosity reduction, and discretization can be used to obtain a reduced representation of the data, while minimizing the loss of information content. Concept hierarchies organize the values of attributes or dimensions into gradual levels of abstraction. They are a form a discretization that is particularly useful in multilevel mining. Automatic generation of concept hierarchies for categoric data may be based on the number of distinct values of the attributes defining the hierarchy. For numeric data, techniques such as data segmentation by partition rules, histogram analysis, and clustering analysis can be used. Although several methods of data preparation have been developed, data preparation remains an active and important area of research.

Chapter 4
Classification of Algorithms

4.1 Classification Problem

In machine learning, classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations whose category membership is known.
Classification is a supervised learning problem i.e. problems in which correctly classified tuples are given and algorithm learns from these tuples. An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm, that maps input data to a category.

Binary and multiclass classification

Classification can be thought of as two separate problems – binary classification and multiclass classification. In binary classification, a better understood task, only two classes are involved, whereas multiclass classification involves assigning an object to one of several classes. Since many classification methods have been developed specifically for binary classification, multiclass classification often requires the combined use of multiple binary classifiers.

Comparing classification methods

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

1) **Accuracy:** The accuracy of a classifier refers to its ability to correctly predict the lable of unlabelled or previously unseen data or tuple. Higher the accuracy better will be the algorithm.

2) **Speed:** Speed of a classifier refers to how much classification time does the classification algorithm requires to complete the classification procedure.

3) **Robustness:** Robustness of a classifier is its ability to make correct classifications when the given data is noisy and has outliers.

4) **Scalability:** Scalability of a classifier is its ability to classify efficiently when large amounts of data is given

5) **Interpretability:** This refers to the level of understanding and insight that is provided by the classifier.

4.2 K - Nearest Neighbours

K Nearest Neighbours (KNN from now on) is one of those algorithms that are very simple to understand but works incredibly well in practice. Also it is surprisingly versatile and its applications range from vision to proteins to computational geometry to graphs and so on. Most people learn the algorithm and do not use it much which is a pity as a clever use of KNN can make things very simple.

Introduction

KNN is an *non parametric lazy learning* algorithm. That is a pretty concise statement. When you say a technique is non parametric, it means that it does not make any assumptions on the underlying data distribution. This is pretty useful as in the real world, most of the practical data does not obey the typical theoretical assumptions made (eg Gaussian mixtures, linearly separable etc). Non parametric algorithms like KNN come to the rescue here.
It is also a lazy algorithm. What this means is that it does not use the training data points to do any *generalization*. In other words, there is *no explicit training phase* or it is very minimal. This means the training phase is pretty fast . Lack of generalization means that KNN keeps all the training data. More exactly, all the training data is needed during the testing phase. (Well this is an exaggeration, but not far from truth). This is in contrast to other techniques like SVM where you can discard all non support vectors without any problem. Most of the lazy algorithms – especially KNN – makes decision based on the entire training data set (in the best case a subset of them).

The dichotomy is pretty obvious here – There is a non existent or minimal training phase but a costly testing phase. The cost is in terms of both time and memory. More time might be needed as in the worst case, all data points might take point in decision. More memory is needed as we need to store all training data.

KNN for Density Estimation

Although classification remains the primary application of KNN, we can use it to do density estimation also. Since KNN is non parametric, it can do estimation for arbitrary distributions. The idea is very similar to use of **Parzen window** . Instead of using hypercube and kernel functions, here we do the estimation as follows – For estimating the density at a point x, place a hypercube centered at x and keep increasing its size till k neighbors are captured. Now estimate the density using the formula,

$$p(x) = \frac{k/n}{V}$$

Where n is the total number of V is the volume of the hypercube. Notice that the numerator is essentially a constant and the density is influenced by the volume. The intuition is this : Lets say density at x is very high. Now, we can find k points near x very quickly . These points are also very close to x (by definition of high density). This means the volume of hypercube is small and the resultant density is high. Lets say the density around x is very low. Then the volume of the hypercube needed to encompass k nearest neighbors is large and consequently, the ratio is low.

The volume performs a job similar to the bandwidth parameter in kernel density estimation. In fact , KNN is one of common methods to estimate the bandwidth (eg adaptive mean shift) .

KNN for Classsification

Lets see how to use KNN for classification. In this case, we are given some data points for training and also a new unlabelled data for testing. Our aim is to find the class label for the new point. The algorithm has different behavior based on k.

Case 1: k = 1 or Nearest Neighbor Rule

This is the simplest scenario. Let x be the point to be labeled . Find the point closest to x . Let it be y. Now nearest neighbor rule asks to assign the label of y to x. This seems too simplistic and some times even counter intuitive. If you feel

that this procedure will result a huge error , you are right – but there is a catch. This reasoning holds only when the number of data points is not very large.

If the number of data points is very large, then there is a very high chance that label of x and y are same. An example might help – Lets say you have a (potentially) biased coin. You toss it for 1 million time and you have got head 900,000 times. Then most likely your next call will be head. We can use a similar argument here.

Let me try an informal argument here - Assume all points are in a D dimensional plane . The number of points is reasonably large. This means that the density of the plane at any point is fairly high. In other words , within any subspace there is adequate number of points. Consider a point x in the subspace which also has a lot of neighbors. Now let y be the nearest neighbor. If x and y are sufficiently close, then we can assume that probability that x and y belong to same class is fairly same – Then by decision theory, x and y have the same class.

The book "Pattern Classification" by Duda and Hart has an excellent discussion about this Nearest Neighbor rule. One of their striking results is to obtain a fairly tight error bound to the Nearest Neighbor rule. The bound is

$$P^* \leq P \leq P^*\left(2 - \frac{c}{c-1}P^*\right)$$

Where P^* is the Bayes error rate, c is the number of classes and P is the error rate of Nearest Neighbor. The result is indeed very striking (atleast to me) because it says that if the number of points is fairly large then the error rate of Nearest Neighbor is less that twice the Bayes error rate. Pretty cool for a simple algorithm like KNN. Do read the book for all the juicy details.

Case 2: k = K or k-Nearest Neighbor Rule

This is a straightforward extension of 1NN. Basically what we do is that we try to find the k nearest neighbor and do a majority voting. Typically k is odd when the number of classes is 2. Lets say k = 5 and there are 3 instances of C1 and 2 instances of C2. In this case , KNN says that new point has to labeled as C1 as it forms the majority. We follow a similar argument when there are multiple classes.

One of the straight forward extension is not to give 1 vote to all the neighbors. A very common thing to do is *weighted kNN* where each point has a weight which is typically calculated using its distance. For eg under inverse distance weighting, each point has a weight equal to the inverse of its distance to the point to be classified. This means that neighboring points have a higher vote than the farther points.

It is quite obvious that the accuracy *might* increase when you increase k but the computation cost also increases.

Some Basic Observations

1. If we assume that the points are d-dimensional, then the straight forward implementation of finding k Nearest Neighbor takes O(dn) time.

2. We can think of KNN in two ways – One way is that KNN tries to estimate the posterior probability of the point to be labeled (and apply bayesian decision theory based on the posterior probability). An alternate way is that KNN calculates the decision surface (either implicitly or explicitly) and then uses it to decide on the class of the new points.

3. There are many possible ways to apply weights for KNN – One popular example is the Shephard's method.

4. Even though the naive method takes O(dn) time, it is very hard to do better unless we make some other assumptions. There are some efficient data structures like **KD-Tree** which can reduce the time complexity but they do it at the cost of increased training time and complexity.

5. In KNN, k is usually chosen as an odd number if the number of classes is 2.

6. Choice of k is very critical – A small value of k means that noise will have a higher influence on the result. A large value make it computationally expensive and kinda defeats the basic philosophy behind KNN (that points that are near might have similar densities or classes) .A simple approach to select k is set $k = \sqrt{n}$

7. There are some interesting data structures and algorithms when you apply KNN on graphs – See **Euclidean minimum spanning tree** and **Nearest neighbor graph** .

8. There are also some nice techniques like condensing, search tree and partial distance that try to reduce the time taken to find the k nearest neighbor.

Applications of KNN

KNN is a versatile algorithm and is used in a huge number of fields. Let us take a look at few uncommon and non trivial applications.

1. Nearest Neighbor based Content Retrieval

This is one the fascinating applications of KNN – Basically we can use it in Computer Vision for many cases – You can consider handwriting detection as a rudimentary nearest neighbor problem. The problem becomes more fascinating if the content is a video – given a video find the video closest to the query from the database – Although this looks abstract, it has lot of practical applications – Eg : Consider **ASL** (American Sign Language) . Here the communication is done using hand gestures.
So lets say if we want to prepare a dictionary for ASL so that user can query it doing a gesture. Now the problem reduces to find the (possibly k) closest gesture(s) stored in the database and show to user. In its heart it is nothing but a KNN problem. One of the professors from my dept , Vassilis Athitsos , does research in this interesting topic – See **Nearest Neighbor Retrieval and Classification** for more details.

2. Gene Expression

This is another cool area where many a time, KNN performs better than other state of the art techniques . In fact a combination of KNN-SVM is one of the most popular techniques there. This is a huge topic on its own and hence I will refrain from talking much more about it.

3. Protein-Protein interaction and 3D structure prediction

Graph based KNN is used in protein interaction prediction. Similarly KNN is used in structure prediction.

4.3 Support Vector Machine

Introduction

Mastering machine learning algorithms isn't a myth at all. Most of the beginners start by learning regression. It is simple to learn and use, but does that solve our purpose? Of course not! Because, you can do so much more than just Regression!

Think of machine learning algorithms as an armory packed with axes, sword, blades, bow, dagger etc. You have various tools, but you ought to learn to use them at the right time. As an analogy, think of 'Regression' as a sword capable of slicing and dicing data efficiently, but incapable of dealing with highly complex data. On the contrary, 'Support Vector Machines' is like a sharp knife – it works on smaller datasets, but on them, it can be much more stronger and powerful in building models.

What is Support Vector Machine?

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot).

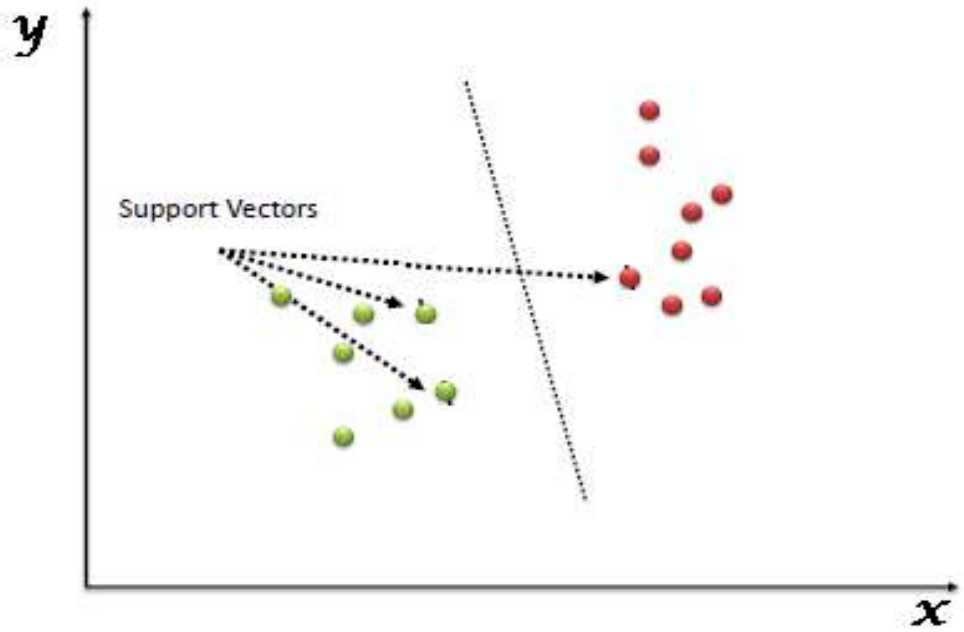


Figure 3: SVM hyperplane

Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

How does it work?

Above, we got accustomed to the process of segregating the two classes with a hyper-plane. Now let's see "How can we identify the right hyper-plane?".

- ***Identify the right hyper-plane (Scenario-1):*** Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star.

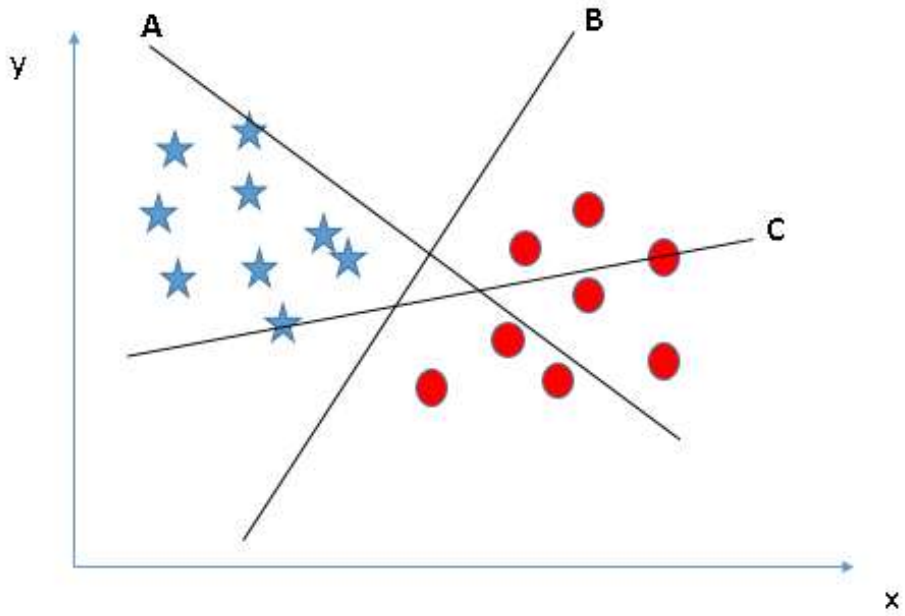


Figure 4: hyperplane identification

You need to remember a thumb rule to identify the right hyper-plane: "Select the hyper-plane which segregates the two classes better". In this scenario, hyper-plane "B" has excellently performed this job.

- ***Identify the right hyper-plane (Scenario-2):*** Here, we have three hyper-planes (A, B and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?

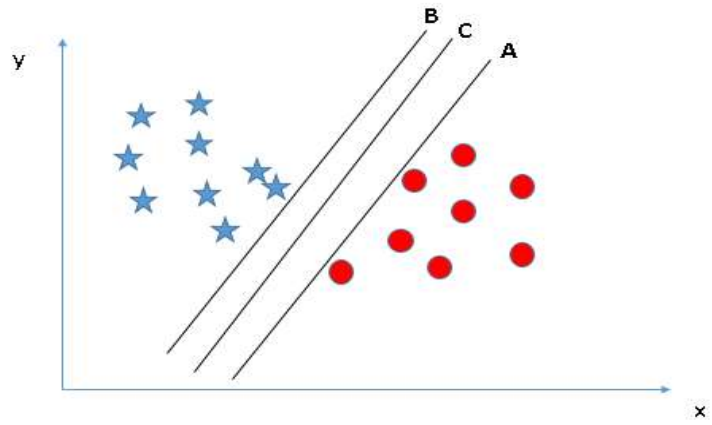


Figure 5: hyperplane segrated

Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**.

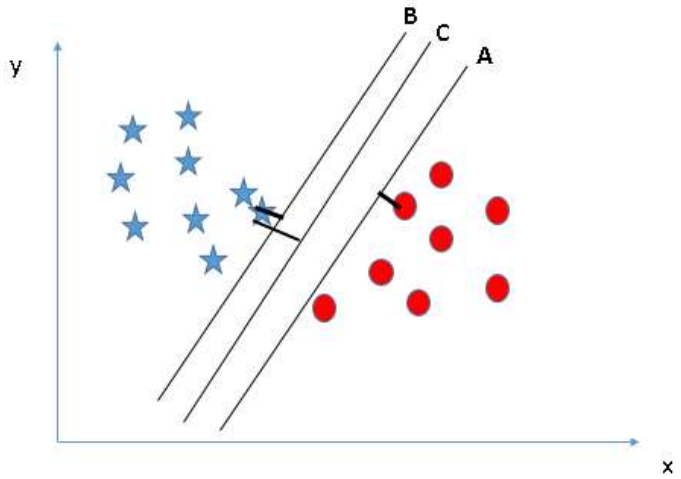


Figure 6: hyperplane with distances

Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

- ***Identify the right hyper-plane (Scenario-3):***

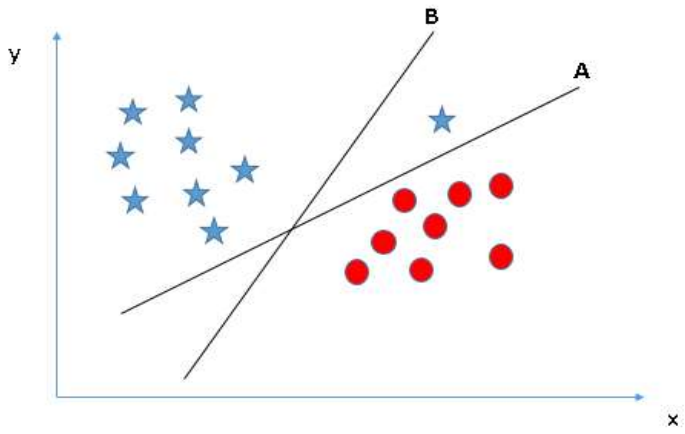


Figure 7: right hyperplane

Some of you may have selected the hyper-plane **B** as it has higher margin compared to **A.** But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**

- **Can we classify two classes (Scenario-4)?**

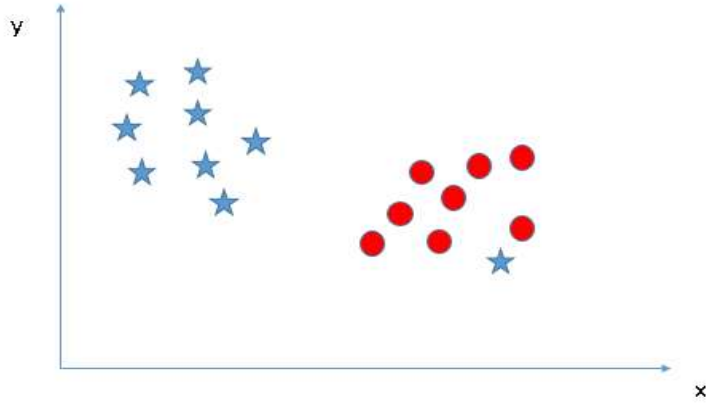


Figure 8: hyperplane classification

As I have already mentioned, one star at other end is like an outlier for star class. SVM has a feature to ignore outliers and find the hyper-plane that has maximum margin. Hence, we can say, SVM is robust to outliers.

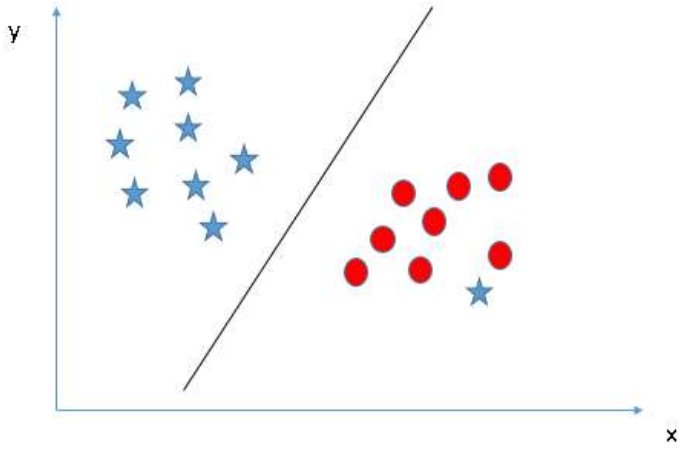


Figure 9: SVM with robust identifier

- **Find the hyper-plane to segregate to classes (Scenario-5):** In the scenario below, we can't have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.

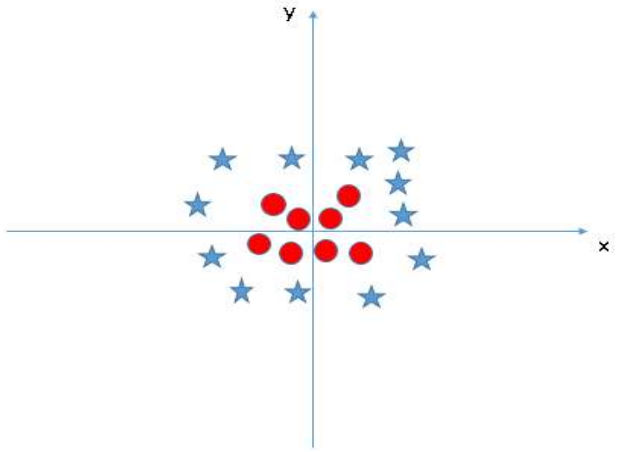


Figure 10: hyper-plane to segregate to classes

SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature z=x^2+y^2. Now, let's plot the data points on axis x and z:

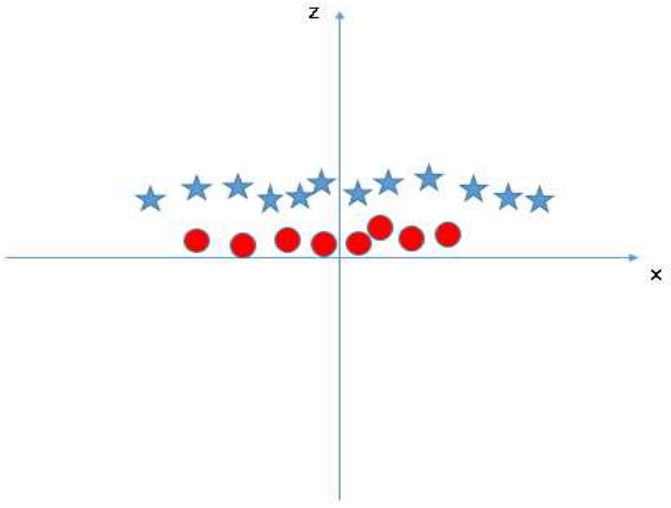


Figure 11: hyper-plane to segregate to classes

In above plot, points to consider are:

- All values for z would be positive always because z is the squared sum of both x and y
- In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.

In SVM, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, SVM has a technique called the **kernel trick**. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs you've defined.

When we look at the hyper-plane in original input space it looks like a circle:

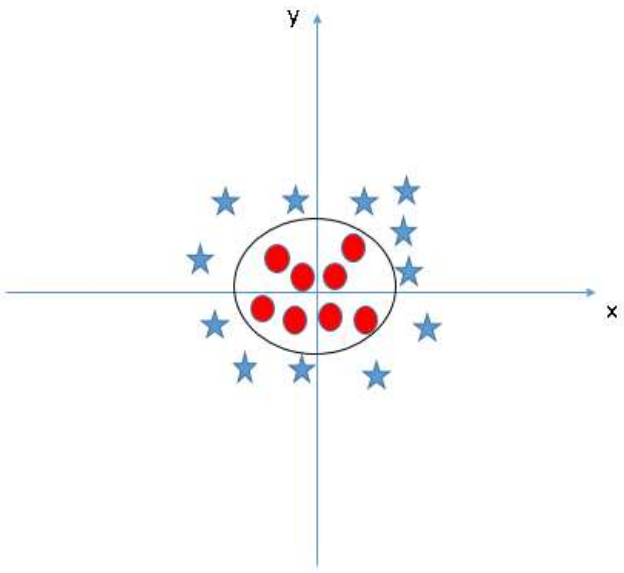


Figure 12: hyper-plane in original input space

Pros and Cons associated with SVM

- **Pros:**
  - It works really well with clear margin of separation
  - It is effective in high dimensional spaces.
  - It is effective in cases where number of dimensions is greater than the number of samples.
  - It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- **Cons:**
  - It doesn't perform well, when we have large data set because the required training time is higher
  - It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping
  - SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is related SVC method of Python scikit-learn library.

4.4 Random Forest

Introduction

This section gives a brief overview of random forests and some comments about the features of the method.

Overview

We assume that the user knows about the construction of single classification trees. Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is grown as follows:

1. If the number of cases in the training set is N, sample N cases at random - but *with replacement*, from the original data. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

In the original paper on random forests, it was shown that the forest error rate depends on two things:

- The *correlation* between any two trees in the forest. Increasing the correlation increases the forest error rate.
- The *strength* of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

Reducing m reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of m - usually quite wide. Using the oob error rate (see below) a value of m in the range can quickly be found. This is the only adjustable parameter to which random forests is somewhat sensitive.

Features of Random Forests

- It is unexcelled in accuracy among current algorithms.
- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets.
- Generated forests can be saved for future use on other data.
- Prototypes are computed that give information about the relation between the variables and the classification.

- It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.

Remarks

Random forests does not overfit. You can run as many trees as you want. It is fast. Running on a data set with 50,000 cases and 100 variables, it produced 100 trees in 11 minutes on a 800Mhz machine. For large data sets the major memory requirement is the storage of the data itself, and three integer arrays with the same dimensions as the data. If proximities are calculated, storage requirements grow as the number of cases times the number of trees.

How random forests work?

To understand and use the various options, further information about how they are computed is useful. Most of the options depend on two data objects generated by random forests.

*W*hen the training set for the current tree is drawn by sampling with replacement, about one-third of the cases are left out of the sample. This **oob (out-of-bag) data** is used to get a running unbiased estimate of the classification error as trees are added to the forest. It is also used to get estimates of variable importance.

*A*fter each tree is built, all of the data are run down the tree, and **proximities** are computed for each pair of cases. If two cases occupy the same terminal node, their proximity is increased by one. At the end of the run, the proximities are normalized by dividing by the number of trees. Proximities are used in replacing missing data, locating outliers, and producing illuminating low-dimensional views of the data.

Random forest is like bootstrapping algorithm with Decision tree (CART) model. Say, we have 1000 observation in the complete population with 10 variables. Random forest tries to build multiple CART model with different sample and different initial variables. For instance, it will take a random sample of 100 observation and 5 randomly chosen initial variables to build a CART model. It will repeat the process (say) 10 times and then make a final prediction on each

observation. Final prediction is a function of each prediction. This final prediction can simply be the mean of each prediction.

Conclusion

Random forest gives much more accurate predictions when compared to simple CART/CHAID or regression models in many scenarios. These cases generally have high number of predictive variables and huge sample size. This is because it captures the variance of several input variables at the same time and enables high number of observations to participate in the prediction. In some of the coming articles, we will talk more about the algorithm in more detail and talk about how to build a simple random forest on R.

4.5 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.

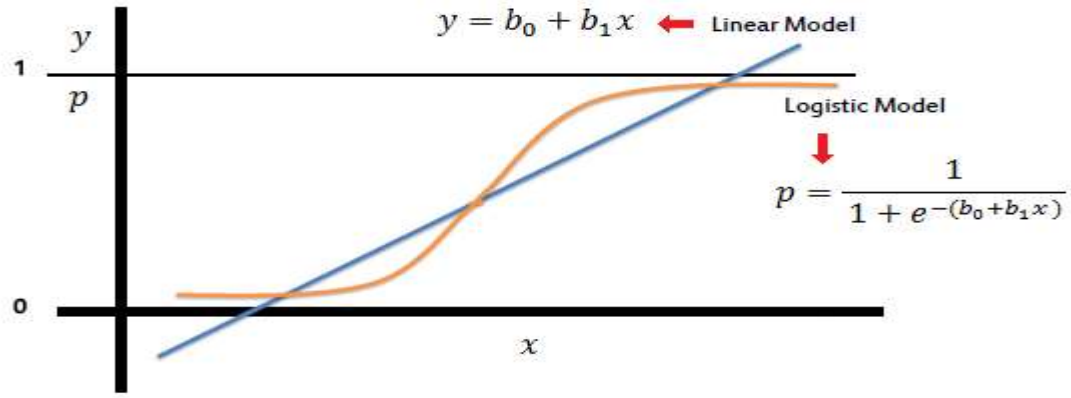


Figure 13: Logistic Curve

In the logistic regression the constant (b_0) 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.

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

Finally, taking the natural log of both sides, we can write the equation in terms of log-odds (logit) which is a linear function of the predictors. The coefficient (b_1) is the amount the logit (log-odds) changes with a one unit change in x.

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

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

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

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

$$\boldsymbol{\beta^1 = \beta^0 + [X^T W X]^{-1}.X^T(y - \mu)}$$

$\boldsymbol{\beta}$ *is a vector of the logistic regression coefficients.*

$\boldsymbol{W}$ *is a square matrix of order N with elements* $n_i\pi_i(1-\pi_i)$ *on the diagonal and zeros everywhere else.*

$\boldsymbol{\mu}$ *is a vector of length N with elements* $\mu_i = n_i\pi_i$.

A **pseudo R^2** value is also available to indicate the adequacy of the regression model. **Likelihood ratio test** is a test of the significance of the difference between the likelihood ratio for the baseline model minus the likelihood ratio for a reduced model. This difference is called "model chi-square". **Wald test** is used to test the statistical significance of each coefficient (*b*) in the model (i.e., predictors contribution).

Pseudo R^2

There are several measures intended to mimic the R^2 analysis to evaluate the goodness-of-fit of logistic models, but they cannot be interpreted as one would interpret an R^2 and different pseudo R^2 can arrive at very different values. Here we discuss three pseudo R^2measures.

Pseudo R^2	Equation	Description
Efron's	$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - p_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$	'*p*' is the logistic model predicted probability. The model residuals are squared, summed, and divided by the total variability in the dependent variable.
McFadden's	$R^2 = 1 - \frac{LL_{full\ model}}{LL_{intercept}}$	The ratio of the log-likelihoods suggests the level of improvement over the intercept model offered by the full model.
Count	$R^2 = \frac{\#\ Corrects}{Total\ Count}$	The number of records correctly predicted, given a cutoff point of .5 divided by the total count of cases. This is equal to the accuracy of a classification model.

Likelihood Ratio Test

The likelihood ratio test provides the means for comparing the likelihood of the data under one model (e.g., full model) against the likelihood of the data under another, more restricted model (e.g., intercept model).

$$LL = \sum_{i=1}^{n} y_i ln(p_i) + (1 - y_i) ln(1 - p_i)$$

where '*p*' is the logistic model predicted probability. The next step is to calculate the difference between these two log-likelihoods.

$$2(LL_1 - LL_2)$$

The difference between two likelihoods is multiplied by a factor of 2 in order to be assessed for statistical significance using standard significance levels (Chi^2 test). The

degrees of freedom for the test will equal the difference in the number of parameters being estimated under the models (e.g., full and intercept).

Wald test

A Wald test is used to evaluate the statistical significance of each coefficient (*b*) in the model.

$$W_j = \frac{b_j}{SE_{b_j}}$$

where *W* is the Wald's statistic with a normal distribution (like Z-test), *b* is the coefficient and *SE* is its standard error. The *W* value is then squared, yielding a Wald statistic with a chi-square distribution.

$$SE_j = sqrt(diag(\boldsymbol{H^{-1}})_j)$$

$$\boldsymbol{H} = [\boldsymbol{X^T W X}]$$

Predictors Contributions

The Wald test is usually used to assess the significance of prediction of each predictor. Another indicator of contribution of a predictor is *exp*(*b*) or **odds-ratio** of coefficient which is the amount the logit (log-odds) changes, with a one unit change in the predictor (*x*).

4.6 SMOTE

Description

This function handles unbalanced classification problems using the SMOTE method. Namely, it can generate a new "SMOTE" data set that addresses the class unbalance problem. Alternatively, it can also run a classification algorithm on this new data set and return the resulting model.

Usage

```
SMOTE(form, data, perc.over = 200, k = 5, perc.under = 200,
    learner = NULL, ...)
```

Arguments

form	A formula describing the prediction problem
data	A data frame containing the original (unbalanced) data set
perc.over	A number that drives the decision of how many extra cases from the minority class are generated (known as over-sampling).
k	A number indicating the number of nearest neighbours that are used to generate the new examples of the minority class.
perc.under	A number that drives the decision of how many extra cases from the majority classes are selected for each case generated from the minority class (known as under-sampling)
learner	Optionally you may specify a string with the name of a function that implements a classification algorithm that will be applied to the resulting SMOTEd data set (defaults to NULL).
...	In case you specify a learner (parameter learner) you can indicate further arguments that will be used when calling this learner.

Algorithm *SMOTE*(T, N, k)
Input: Number of minority class samples T; Amount of SMOTE $N\%$; Number of nearest neighbors k
Output: $(N/100) * T$ synthetic minority class samples

1. (∗ *If N is less than 100%, randomize the minority class samples as only a random percent of them will be SMOTEd.* ∗)
2. **if** $N < 100$
3. **then** Randomize the T minority class samples
4. $T = (N/100) * T$
5. $N = 100$
6. **endif**
7. $N = (int)(N/100)$ (∗ *The amount of SMOTE is assumed to be in integral multiples of 100.* ∗)
8. $k =$ Number of nearest neighbors
9. $numattrs =$ Number of attributes
10. $Sample[\,][\,]$: array for original minority class samples
11. $newindex$: keeps a count of number of synthetic samples generated, initialized to 0
12. $Synthetic[\,][\,]$: array for synthetic samples
(∗ *Compute k nearest neighbors for each minority class sample only.* ∗)
13. **for** $i \leftarrow 1$ **to** T
14. Compute k nearest neighbors for i, and save the indices in the $nnarray$
15. Populate(N, i, $nnarray$)
16. **endfor**

Populate(N, i, $nnarray$) (∗ *Function to generate the synthetic samples.* ∗)

17. **while** $N \neq 0$
18. Choose a random number between 1 and k, call it nn. This step chooses one of the k nearest neighbors of i.
19. **for** $attr \leftarrow 1$ **to** $numattrs$
20. Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][$
21. Compute: $gap =$ random number between 0 and 1
22. $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
23. **endfor**
24. $newindex{+}{+}$
25. $N = N - 1$
26. **endwhile**
27. **return** (∗ *End of Populate.* ∗)

End of Pseudo-Code.

Details

Unbalanced classification problems cause problems to many learning algorithms. These problems are characterized by the uneven proportion of cases that are available for each class of the problem.

SMOTE (Chawla et. al. 2002) is a well-known algorithm to fight this problem. The general idea of this method is to artificially generate new examples of the minority class using the nearest neighbors of these cases. Furthermore, the majority class examples are also under-sampled, leading to a more balanced dataset.

The parameters perc.over and perc.under control the amount of over-sampling of the minority class and under-sampling of the majority classes, respectively. perc.over will tipically be a number above 100. With this type of values, for each case in the orginal data set belonging to the minority class, perc.over/100 new examples of that class will be created. If perc.over is a value below 100 than a single case will be generated for a randomly selected proportion (given by perc.over/100) of the cases belonging to the minority class on the original data set. The parameter perc.under controls the proportion of cases of the majority class that will be randomly selected for the final "balanced" data set. This proportion is calculated with respect to the number of newly generated minority class cases. For instance, if 200 new examples were generated for the minority class, a value of perc.under of 100 will randomly select exactly 200 cases belonging to the majority classes from the original data set to belong to the final data set. Values above 100 will select more examples from the majority classes.

The parameter k controls the way the new examples are created. For each currently existing minority class example X new examples will be created (this is controlled by the parameter perc.over as mentioned above). These examples will be generated by using the information from the k nearest neighbours of each example of the minority class. The parameter k controls how many of these neighbours are used.

The function can also be used to obtain directely the classification model from the resulting balanced data set. This can be done by including the name of the R function that implements the classifier in the parameter learner. You may also include other parameters that will be forward to this learning function. If the learner parameter is not NULL (the default) the returning value of the

function will be the learned model and not the balanced data set. The function that learns the model should have as first parameter the formula describing the classification problem and in the second argument the training set.

Value

The function can return two different types of values depending on the value of the parameter learner. If this parameter is NULL (the default), the function will return a data frame with the new data set resulting from the application of the SMOTE algorithm. Otherwise the function will return the classification model obtained by the learner specified in the parameter learner.

Chapter 5
R Programming and the IDE used

R Programming

R is a programming language and software environment for statistical computing and graphics. The R language is widely used among statisticians and data miners for developing statistical software and data analysis.[8][9] Polls, surveys of data miners, and studies of scholarly literature databases show that R's popularity has increased substantially in recent years.[10][11][12][13]

R is an implementation of the S programming language combined with lexical scoping semantics inspired by Scheme.[14] S was created by John Chambers while at Bell Labs and R was created by Ross Ihaka and Robert Gentleman[15] at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team, of which Chambers is a member. R is named partly after the first names of the first two R authors and partly as a play on the name of S.[16] There are some important differences, but much of the code written for S runs unaltered.

R is a GNU project.[17][18] The source code for the R software environment is written primarily in C, Fortran, and R.[19] R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems. R uses a command line interface; there are also several graphical front-ends for it.

Statistical Features

R and its libraries implement a wide variety of statistical and graphical techniques, including linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, and others. R is easily extensible through functions and extensions, and the R community is noted for its active contributions in terms of packages. Many of R's standard functions are written in R itself, which makes it easy for users to follow the algorithmic choices made. For computationally intensive tasks, C, C++, and Fortran code can be linked and called at run time. Advanced users can write C, C++,[20] Java,[21] .NET [22][23][24] or Python code to manipulate R objects directly.

R is highly extensible through the use of user-submitted packages for specific functions or specific areas of study. Due to its S heritage, R has stronger object-oriented programming facilities than most statistical computing languages. Extending R is also eased by its lexical scoping rules.[25] Another strength of R is static graphics, which can produce publication-quality graphs, including mathematical symbols. Dynamic and interactive graphics are available through additional packages. R has its own LaTeX-like documentation format, which is used to supply comprehensive documentation, both on-line in a number of formats and in hard copy.[26]

Packages

The capabilities of R are extended through user-created packages, which allow specialized statistical techniques, graphical devices (ggplot2), import/export capabilities, reporting tools (knitr, Sweave), etc. These packages are developed primarily in R, and sometimes in Java, C, C++ and Fortran. A core set of packages is included with the installation of R, with more than 5,800 additional packages and 120,000 functions (as of June 2014) available at the Comprehensive R Archive Network (CRAN), Bioconductor, Omegahat, GitHub and other repositories.[12][27]

The "Task Views" page (subject list) on the CRAN website[28] lists a wide range of tasks (in fields such as Finance, Genetics, High Performance Computing, Machine Learning, Medical Imaging, Social Sciences and Spatial Statistics) to which R has been applied and for which packages are available. R has also been identified by the FDA as suitable for interpreting data from clinical research.[29]

Editors and IDEs

Text editors and Integrated development environments (IDEs) with some support for R include: ConTEXT, Eclipse (StatET),[30] Emacs (Emacs Speaks Statistics), LyX (modules for knitr and Sweave), Vim, jEdit,[31] Kate,[32] Revolution R Enterprise DevelopR (part of Revolution R Enterprise),[33] RStudio,[34] Sublime Text, TextMate, WinEdt (R Package RWinEdt), Tinn-R and Notepad++.[35]

RStudio

RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, and debugging and workspace management.

RStudio is available in open source and commercial editions and runs on the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server or RStudio Server Pro (Debian/Ubuntu, RedHat/CentOS, and SUSE Linux). Its features include:

- Access RStudio locally
- Syntax highlighting, code completion, and smart indentation
- Execute R code directly from the source editor
- Quickly jump to function definitions
- Easily manage multiple working directories using projects
- Integrated R help and documentation
- Interactive debugger to diagnose and fix errors quickly
- Extensive package development tools
- All of the features of open source

IDE and Interface used -

- R – 3.4.0 ; https://www.r-project.org/about.html ; https://cran.r-project.org/
- RStudio – 1.0.143 ; https://www.rstudio.com/

R Packages used -

- caret - https://cran.r-project.org/web/packages/caret/index.html
- ggplot2 - https://cran.r-project.org/web/packages/ggplot2/index.html
- DMwR - https://cran.r-project.org/web/packages/DMwR/index.html\
- ROSE - https://cran.r-project.org/web/packages/ROSE/index.html
- e1071 - https://cran.r-project.org/web/packages/e1071/index.html
- randomForest - https://cran.r-project.org/web/packages/randomForest/
- plotly - https://cran.r-project.org/web/packages/plotly/
- stats - https://stat.ethz.ch/R-manual/R-devel/library/stats/html/00Index.html
- cluster - https://cran.r-project.org/web/packages/cluster/
- caTools - https://cran.r-project.org/web/packages/caTools/index.html
- data.table - https://cran.r-project.org/web/packages/data.table/index.html

Chapter 6
Dataset Description

The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, they cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-senstive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification.

The dataset has been collected and analysed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. More details on current and past projects on related topics are available on http://mlg.ulb.ac.be/BruFence and http://mlg.ulb.ac.be/ARTML

Number of Instances: 284,807

Number of Fraud cases: 492

Number of Attributes: 34

Source: https://www.kaggle.com/dalpozz/creditcardfraud

Chapter 7
Implementation

Implementation : Credit Card Fraud Detection

```
library(data.table)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(ggplot2)
library(DMwR)
```

```
## Warning: package 'DMwR' was built under R version 3.3.3
```

```
## Loading required package: grid
```

```
library(ROSE)
```

```
## Warning: package 'ROSE' was built under R version 3.3.3
```

```
## Loaded ROSE 0.0-3
```

```
library(e1071)
library(class)
library(gmodels)
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(plotly)
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

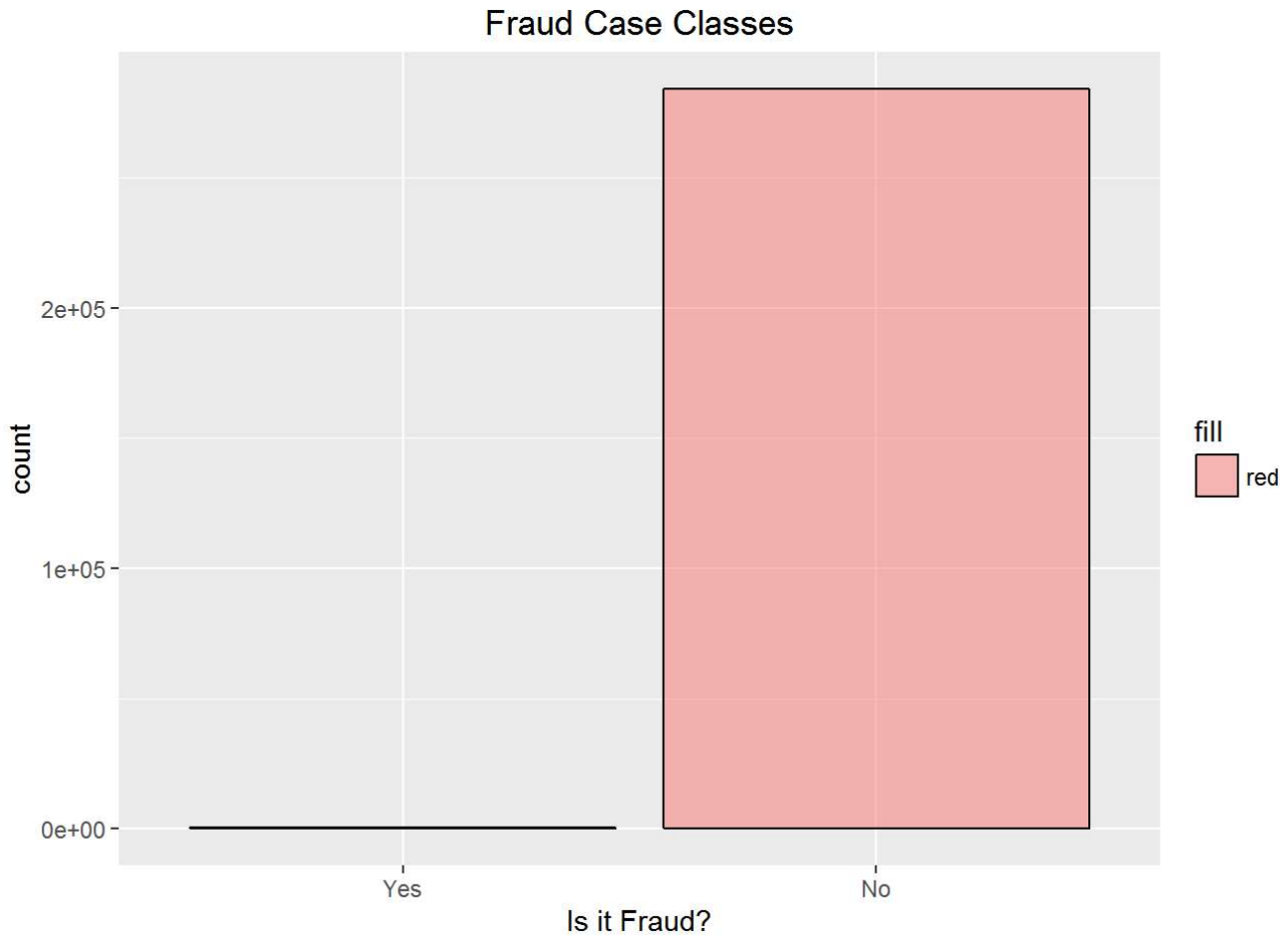
```
library(stats)
library(caTools)
```

```
## Warning: package 'caTools' was built under R version 3.3.2
```

```
library(C50)
library(gmodels)
library(cluster)
library(fpc)
###  Importing the dataset.
creditcard <- read.csv("creditcard.csv")
### Get the feel of the data
creditcard$Class <- factor(creditcard$Class, levels = c("1", "0"), labels = c("Yes", "No"))
creditcard <- creditcard[sample(nrow(creditcard)),]
table(creditcard$Class)
```

```
##
##    Yes     No
##    492 284315
```

```
#Plot the data
ggplot(creditcard,aes(x = creditcard$Class, fill="red")) +
  geom_bar(position = "dodge", alpha = 0.5, col ="black") +
  scale_x_discrete( name = "Is it Fraud?") +
  scale_y_continuous() +
  ggtitle("Fraud Case Classes") +
  theme(plot.title = element_text(hjust = 0.5))
```




```
### Dividing data into fraud and non fraud classes
fraudData = creditcard[creditcard$Class=="Yes",]
NofraudData = creditcard[creditcard$Class=="No",]

#############################################################################
### New dataset with 25% fraud data
fraud25 = fraudData[1:123,]
dataWith25Fraud = rbind(NofraudData,fraud25)

#Randomize the data for 10% fraud
dataFor25 <- dataWith25Fraud[sample(nrow(dataWith25Fraud)),]
table(dataFor25$Class)
```

```
##
##    Yes     No
##    123 284315
```

```
###removing unbalanced classification problemand making data balanced
set.seed(4356)
smote_data <- SMOTE(Class ~ ., data  = dataFor25, perc.over = 300, perc.under = 150, k=5)
new.data <- sample(2, nrow(smote_data), replace = TRUE, prob = c(0.7, 0.3))
trainSplit2 <-smote_data[new.data==1,]
testSplit2 <-smote_data[new.data==2,]
table(trainSplit2$Class)
```

```
##
## Yes  No
## 346 386
```

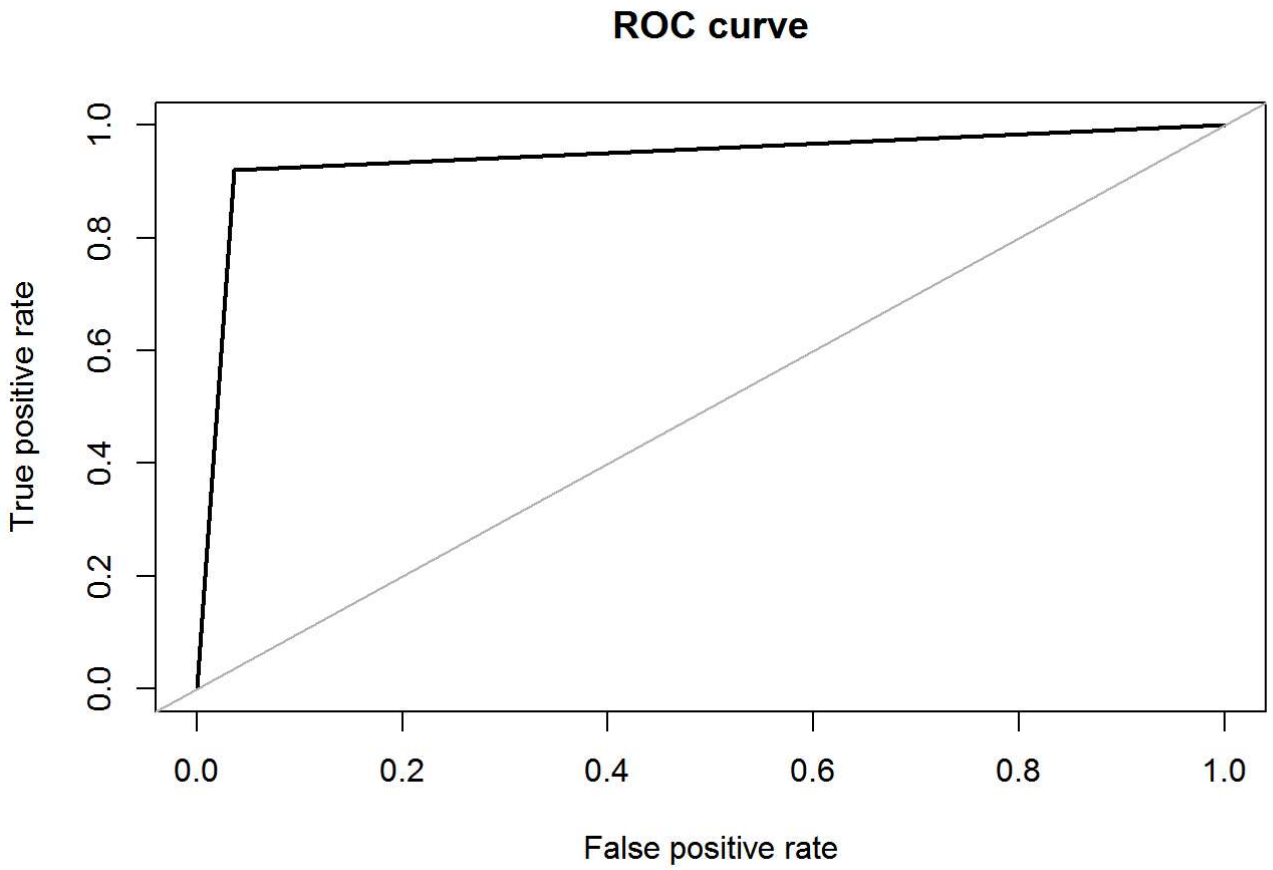
```
table(testSplit2$Class)
```

```
##
## Yes  No
## 146 167
```

```
### Applying SVM algorithm for 25% fraud data
svm.model <- svm(Class ~ ., data = trainSplit2, kernel = "radial", cost = 1, gamma = 0.1)
svm.predict <- predict(svm.model, testSplit2)
confusionMatrix(testSplit2$Class, svm.predict)
```

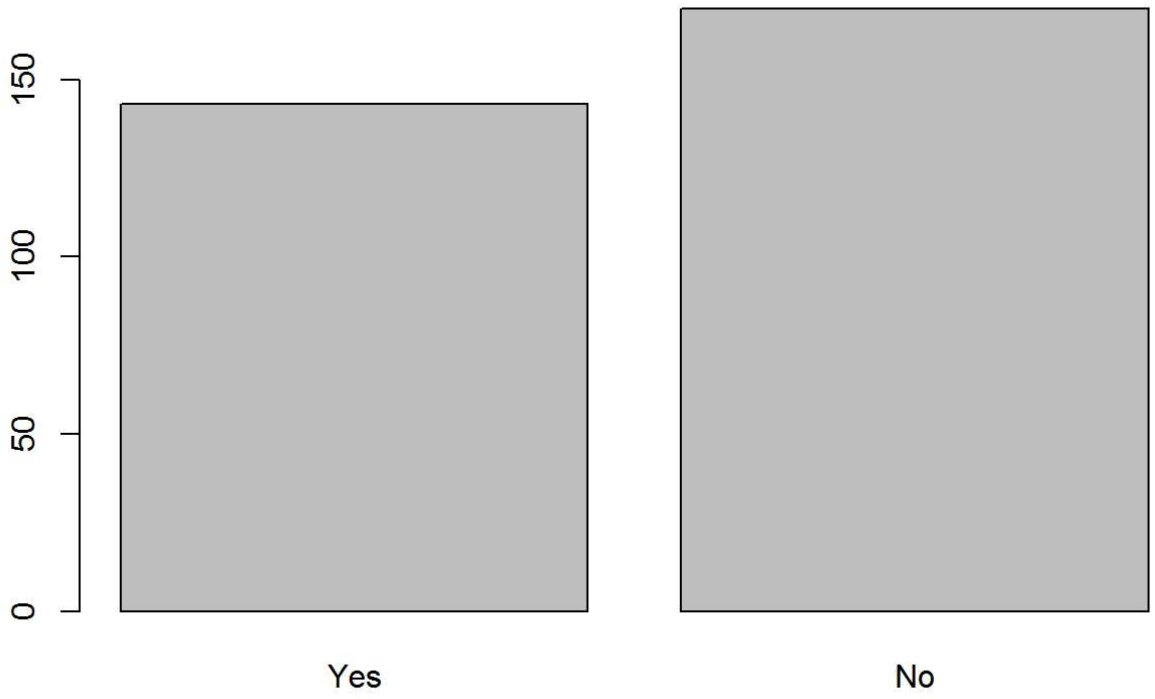
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 132  14
##        No    5 162
##
##                Accuracy : 0.9393
##                  95% CI : (0.9068, 0.9631)
##     No Information Rate : 0.5623
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8776
##  Mcnemar's Test P-Value : 0.06646
##
##             Sensitivity : 0.9635
##             Specificity : 0.9205
##          Pos Pred Value : 0.9041
##          Neg Pred Value : 0.9701
##              Prevalence : 0.4377
##          Detection Rate : 0.4217
##    Detection Prevalence : 0.4665
##       Balanced Accuracy : 0.9420
##
##        'Positive' Class : Yes
##
```

```
roc.curve(svm.predict, testSplit2$Class, plotit = T)
```




```
## Area under the curve (AUC): 0.942
```

```
## Applying KNN Algorithm
knn.model <- knn(train = trainSplit2[,1:30],
    test = testSplit2[,1:30],
    cl = trainSplit2$Class)
plot(knn.model)
```



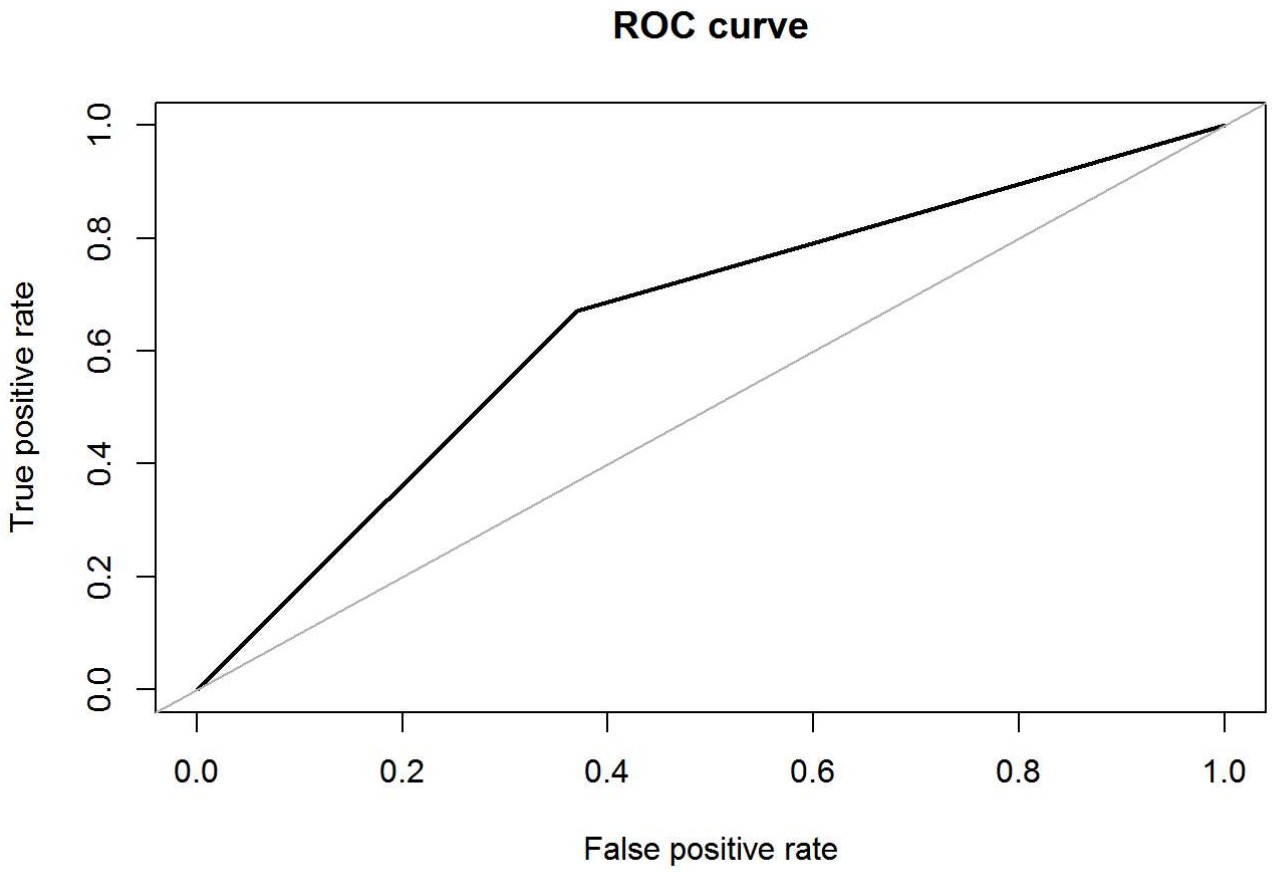

```
table(knn.model, testSplit2$Class)
```

```
##
## knn.model Yes  No
##       Yes  90  53
##       No   56 114
```

```
confusionMatrix(knn.model, testSplit2[,31])
```

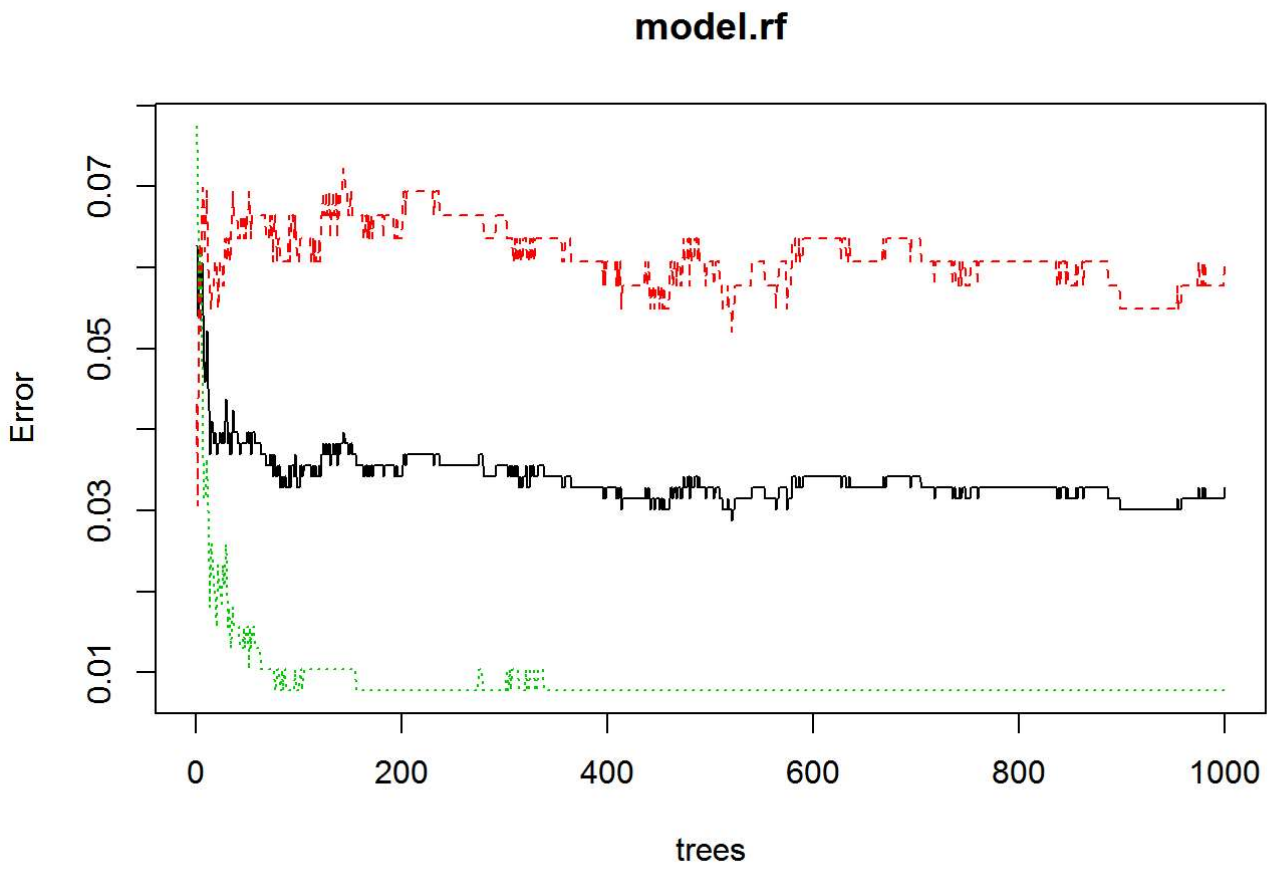
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes  90  53
##        No   56 114
##
##                Accuracy : 0.6518
##                  95% CI : (0.5961, 0.7045)
##     No Information Rate : 0.5335
##     P-Value [Acc > NIR] : 1.49e-05
##
##                   Kappa : 0.2995
##  Mcnemar's Test P-Value : 0.8481
##
##             Sensitivity : 0.6164
##             Specificity : 0.6826
##          Pos Pred Value : 0.6294
##          Neg Pred Value : 0.6706
##              Prevalence : 0.4665
##          Detection Rate : 0.2875
##    Detection Prevalence : 0.4569
##       Balanced Accuracy : 0.6495
##
##        'Positive' Class : Yes
##
```

```
roc.curve(knn.model, testSplit2$Class, plotit = T)
```



```
## Area under the curve (AUC): 0.650
```

```
### Applying Random Forest algorithm for 25% fraud data
model.rf <- randomForest(Class ~ ., data =trainSplit2 , ntree = 1000, importance = TRUE)
plot(model.rf)
```



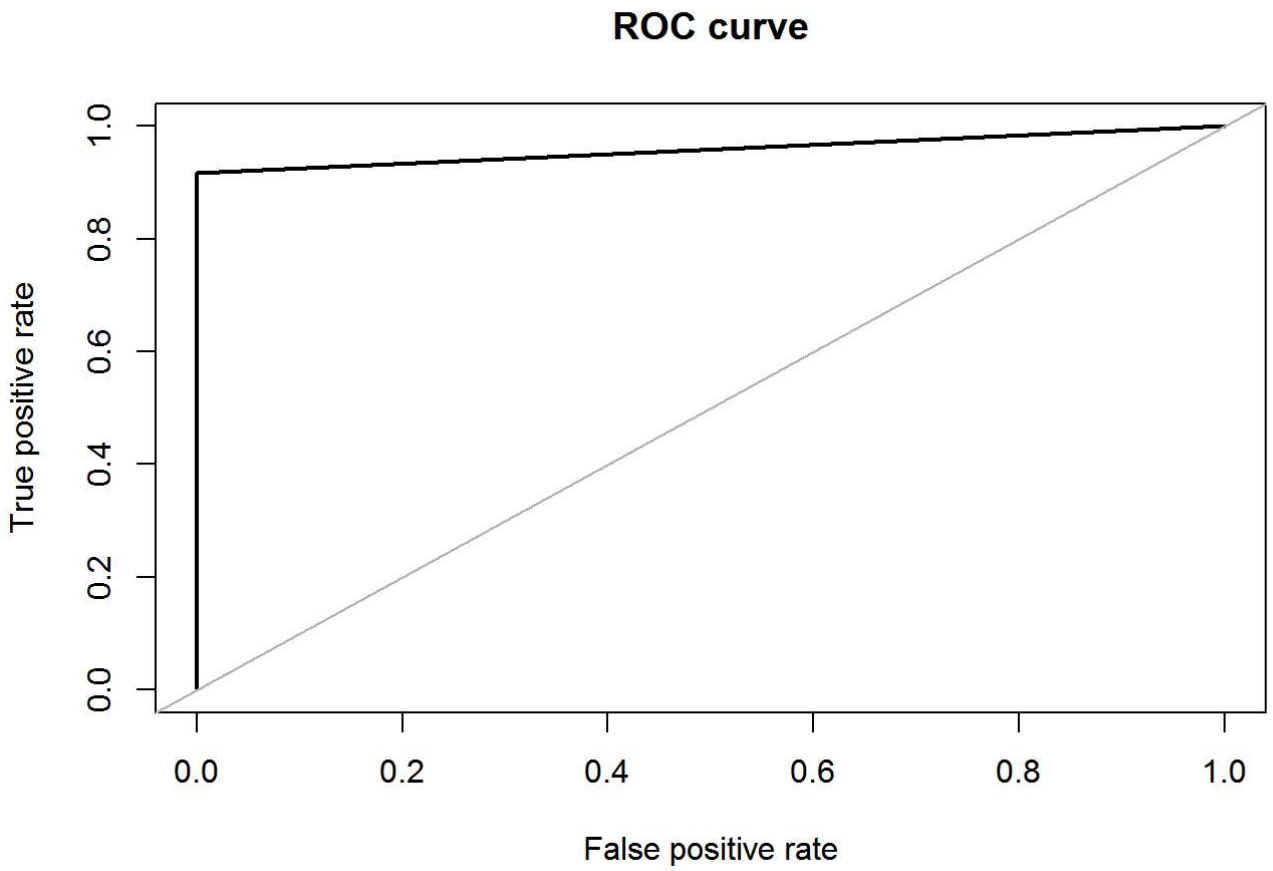

```
cv.tree.pred2 <- predict(model.rf, testSplit2)

# Making table of Confusion matrix
CrossTable(cv.tree.pred2, testSplit2$Class,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
           dnn = c('actual class', 'predicted class'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  313
##
##
##              | predicted class
## actual class |       Yes |        No | Row Total |
## -------------|-----------|-----------|-----------|
##          Yes |       131 |         0 |       131 |
##              |     0.419 |     0.000 |           |
## -------------|-----------|-----------|-----------|
##           No |        15 |       167 |       182 |
##              |     0.048 |     0.534 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       146 |       167 |       313 |
## -------------|-----------|-----------|-----------|
##
##
```

```
confusionMatrix(cv.tree.pred2, testSplit2$Class)
```

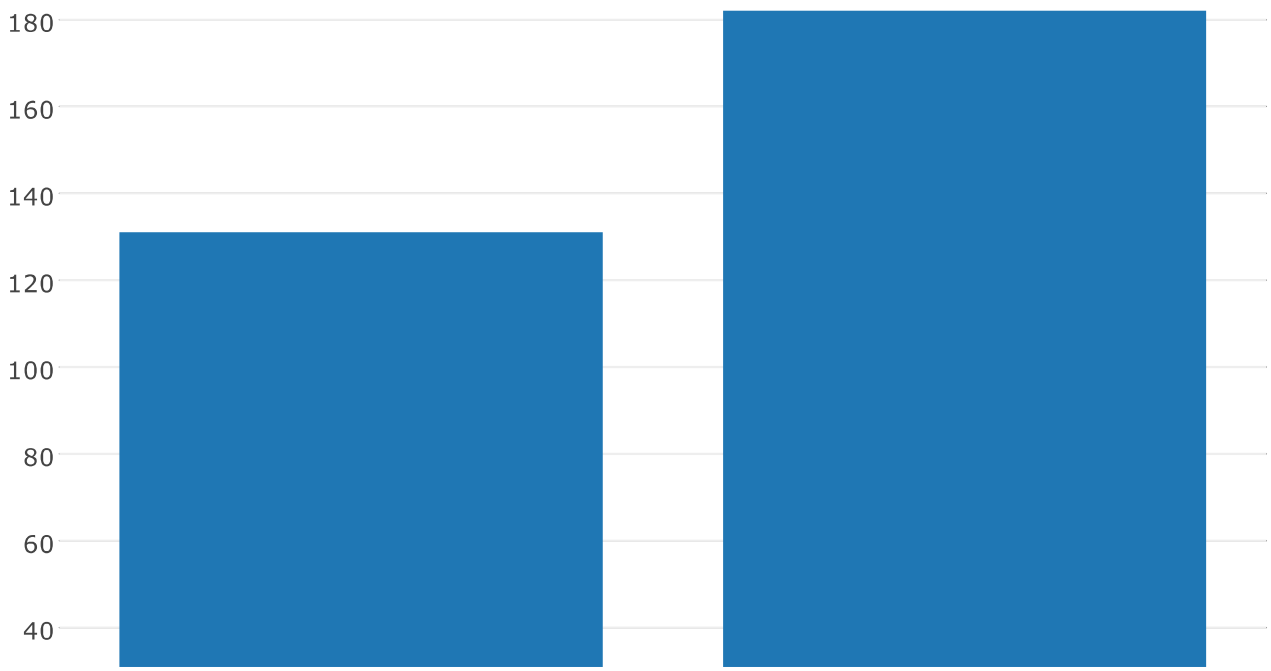
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 131   0
##        No   15 167
##
##                Accuracy : 0.9521
##                  95% CI : (0.9222, 0.9729)
##     No Information Rate : 0.5335
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9031
##  Mcnemar's Test P-Value : 0.0003006
##
##             Sensitivity : 0.8973
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9176
##              Prevalence : 0.4665
##          Detection Rate : 0.4185
##    Detection Prevalence : 0.4185
##       Balanced Accuracy : 0.9486
##
##        'Positive' Class : Yes
##
```

```
### ploting results
roc.curve(cv.tree.pred2, testSplit2$Class, plotit = T)
```




```
## Area under the curve (AUC): 0.959
```

```
x<-rchisq(1000,5,0)
plot_ly(x=cv.tree.pred2,type = 'histogram')
```




```
### Applying Logistic Regression Algorithm for 25% fraud data
logist <- glm(Class ~ ., data = trainSplit2, family = "binomial")
```

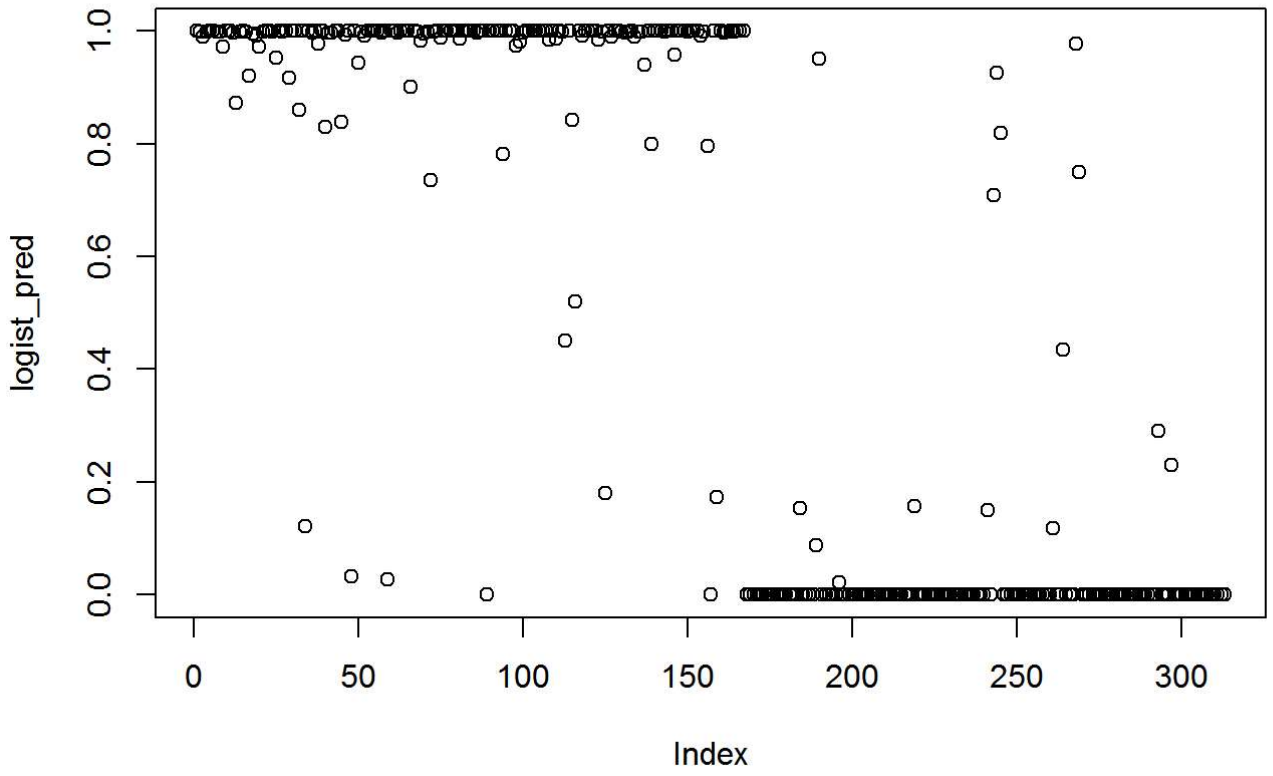
```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(logist)
```

```
##
## Call:
## glm(formula = Class ~ ., family = "binomial", data = trainSplit2)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.96293   0.00000   0.00000   0.01889   3.02022
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.368e+01  9.313e+01   0.362    0.718
## Time         1.960e-05  1.599e-05   1.226    0.220
## V1           1.244e+01  3.560e+01   0.349    0.727
## V2          -7.519e+01  1.383e+02  -0.544    0.587
## V3           6.193e+01  7.513e+01   0.824    0.410
## V4          -4.168e+01  5.145e+01  -0.810    0.418
## V5           2.845e+01  3.711e+01   0.766    0.443
## V6           3.646e+01  6.112e+01   0.597    0.551
## V7           1.412e+02  2.299e+02   0.614    0.539
## V8          -2.451e+01  6.237e+01  -0.393    0.694
## V9           5.302e+01  7.594e+01   0.698    0.485
## V10          1.204e+02  1.717e+02   0.701    0.483
## V11         -9.061e+01  1.333e+02  -0.680    0.497
## V12          1.648e+02  2.391e+02   0.689    0.491
## V13          2.064e+00  8.756e+00   0.236    0.814
## V14          1.749e+02  2.575e+02   0.679    0.497
## V15          6.362e+00  9.844e+00   0.646    0.518
## V16          1.562e+02  2.298e+02   0.680    0.497
## V17          2.782e+02  4.072e+02   0.683    0.495
## V18          1.036e+02  1.551e+02   0.668    0.504
## V19         -4.000e+01  6.171e+01  -0.648    0.517
## V20          8.462e+00  5.497e+01   0.154    0.878
## V21         -1.122e+01  6.325e+01  -0.177    0.859
## V22         -1.573e+01  3.733e+01  -0.421    0.673
## V23         -3.286e+01  8.132e+01  -0.404    0.686
## V24          4.328e+00  7.944e+00   0.545    0.586
## V25         -1.928e+01  3.678e+01  -0.524    0.600
## V26         -1.129e+00  9.347e+00  -0.121    0.904
## V27         -2.481e+01  3.112e+01  -0.797    0.425
## V28         -3.478e+01  9.784e+01  -0.355    0.722
## Amount      -3.910e-01  9.375e-01  -0.417    0.677
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1012.581  on 731  degrees of freedom
## Residual deviance:   77.153  on 701  degrees of freedom
## AIC: 139.15
##
## Number of Fisher Scoring iterations: 25
```

```
logist_pred <- predict(logist, newdata=testSplit2,type = "response")
plot(logist_pred)
```

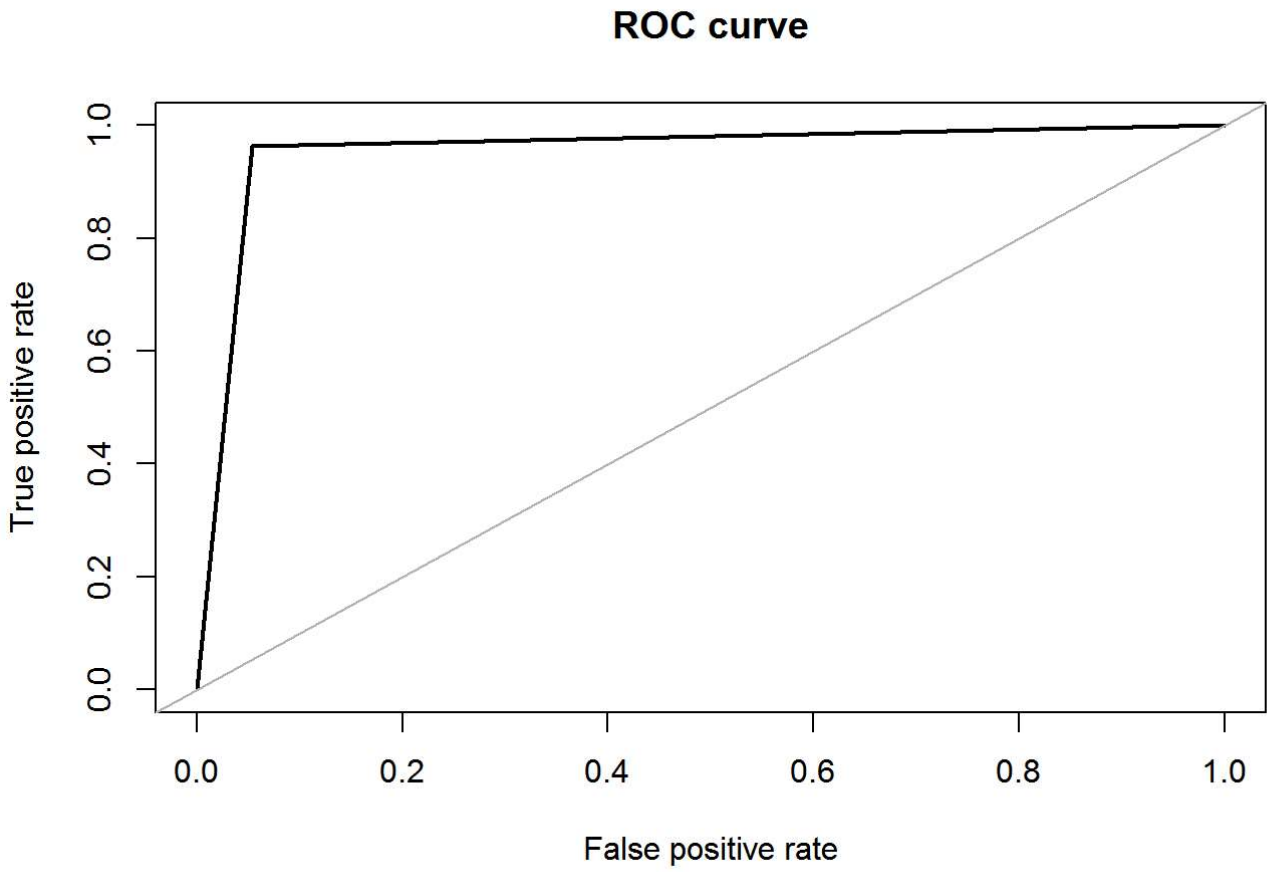



```
pred=rep("Yes",length(logist_pred))
pred[logist_pred > 0.5] = "No"
confusionMatrix(testSplit2$Class, pred)
```

```
## Warning in confusionMatrix.default(testSplit2$Class, pred): Levels are not
## in the same order for reference and data. Refactoring data to match.
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  159   8
##        Yes   6 140
##
##                Accuracy : 0.9553
##                  95% CI : (0.9261, 0.9753)
##     No Information Rate : 0.5272
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9102
##  Mcnemar's Test P-Value : 0.7893
##
##             Sensitivity : 0.9636
##             Specificity : 0.9459
##          Pos Pred Value : 0.9521
##          Neg Pred Value : 0.9589
##              Prevalence : 0.5272
##          Detection Rate : 0.5080
##    Detection Prevalence : 0.5335
##       Balanced Accuracy : 0.9548
##
##        'Positive' Class : No
##
```

```
roc.curve(pred, testSplit2$Class, plotit = T)
```



```
## Area under the curve (AUC): 0.955
```

```
#############################################################################
### New dataset with 50% fraud data
fraud50 = fraudData[1:246,]
dataWith50Fraud = rbind(NofraudData,fraud50)

#Randomize the data for 50% fraud
dataFor50 <- dataWith50Fraud[sample(nrow(dataWith50Fraud)),]
table(dataFor50$Class)
```

```
##
##    Yes     No
##    246 284315
```

```
###removing unbalanced classification problemand making data balanced
set.seed(4356)
smote_data <- SMOTE(Class ~ ., data  = dataFor50, perc.over = 300, perc.under = 150, k=5)
new.data <- sample(2, nrow(smote_data), replace = TRUE, prob = c(0.7, 0.3))
trainSplit2 <-smote_data[new.data==1,]
testSplit2 <-smote_data[new.data==2,]
table(trainSplit2$Class)
```

```
##
## Yes  No
## 682 772
```

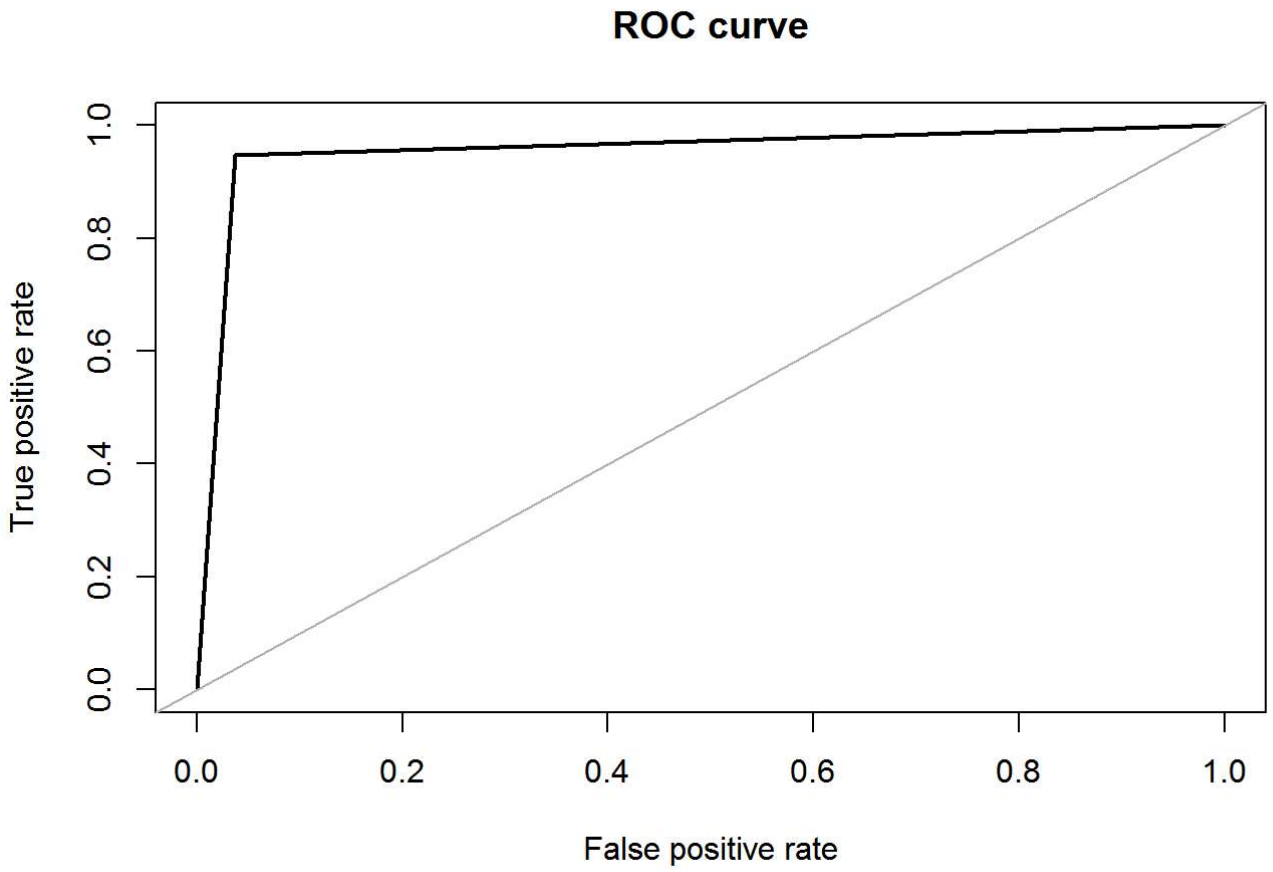
```
table(testSplit2$Class)
```

```
##
## Yes  No
## 302 335
```

```
### Applying SVM algorithm for 50% fraud data
svm.model <- svm(Class ~ ., data = trainSplit2, kernel = "radial", cost = 1, gamma = 0.1)
svm.predict <- predict(svm.model, testSplit2)
confusionMatrix(testSplit2$Class, svm.predict)
```

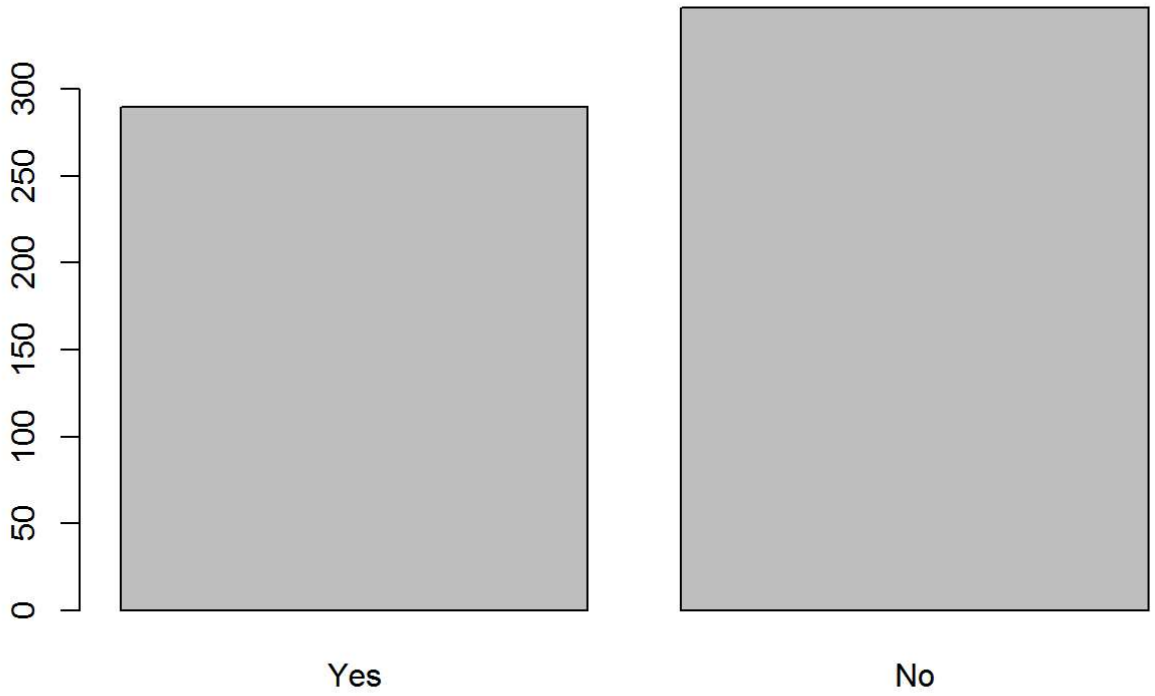
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 284  18
##        No   11 324
##
##                Accuracy : 0.9545
##                  95% CI : (0.9353, 0.9693)
##     No Information Rate : 0.5369
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9086
##  Mcnemar's Test P-Value : 0.2652
##
##             Sensitivity : 0.9627
##             Specificity : 0.9474
##          Pos Pred Value : 0.9404
##          Neg Pred Value : 0.9672
##              Prevalence : 0.4631
##          Detection Rate : 0.4458
##    Detection Prevalence : 0.4741
##       Balanced Accuracy : 0.9550
##
##        'Positive' Class : Yes
##
```

```
roc.curve(svm.predict, testSplit2$Class, plotit = T)
```



```
## Area under the curve (AUC): 0.955
```

```
## Applying KNN Algorithm
knn.model <- knn(train = trainSplit2[,1:30],
    test = testSplit2[,1:30],
    cl = trainSplit2$Class)
plot(knn.model)
```



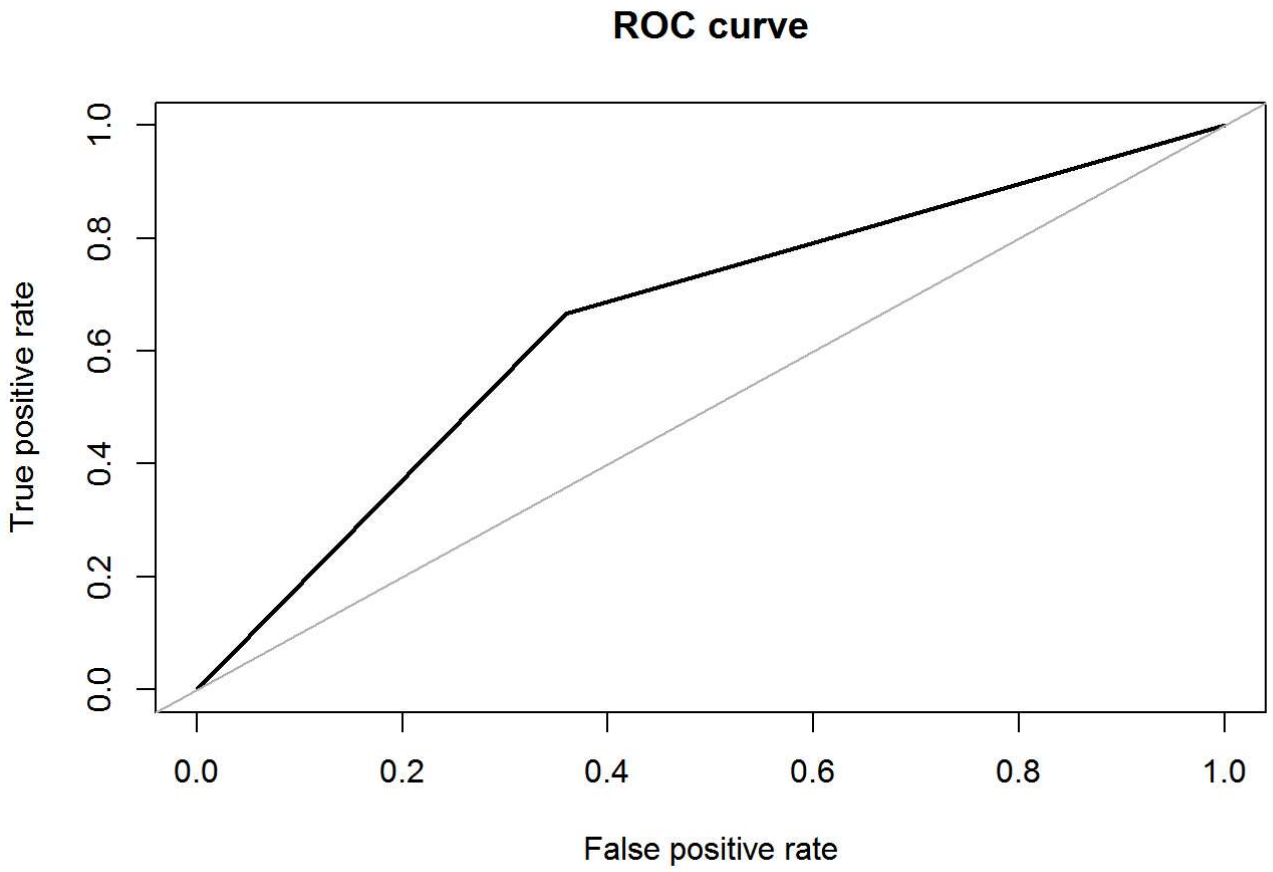

```
table(knn.model, testSplit2$Class)
```

```
##
## knn.model Yes  No
##       Yes 186 104
##       No  116 231
```

```
confusionMatrix(knn.model, testSplit2[,31])
```

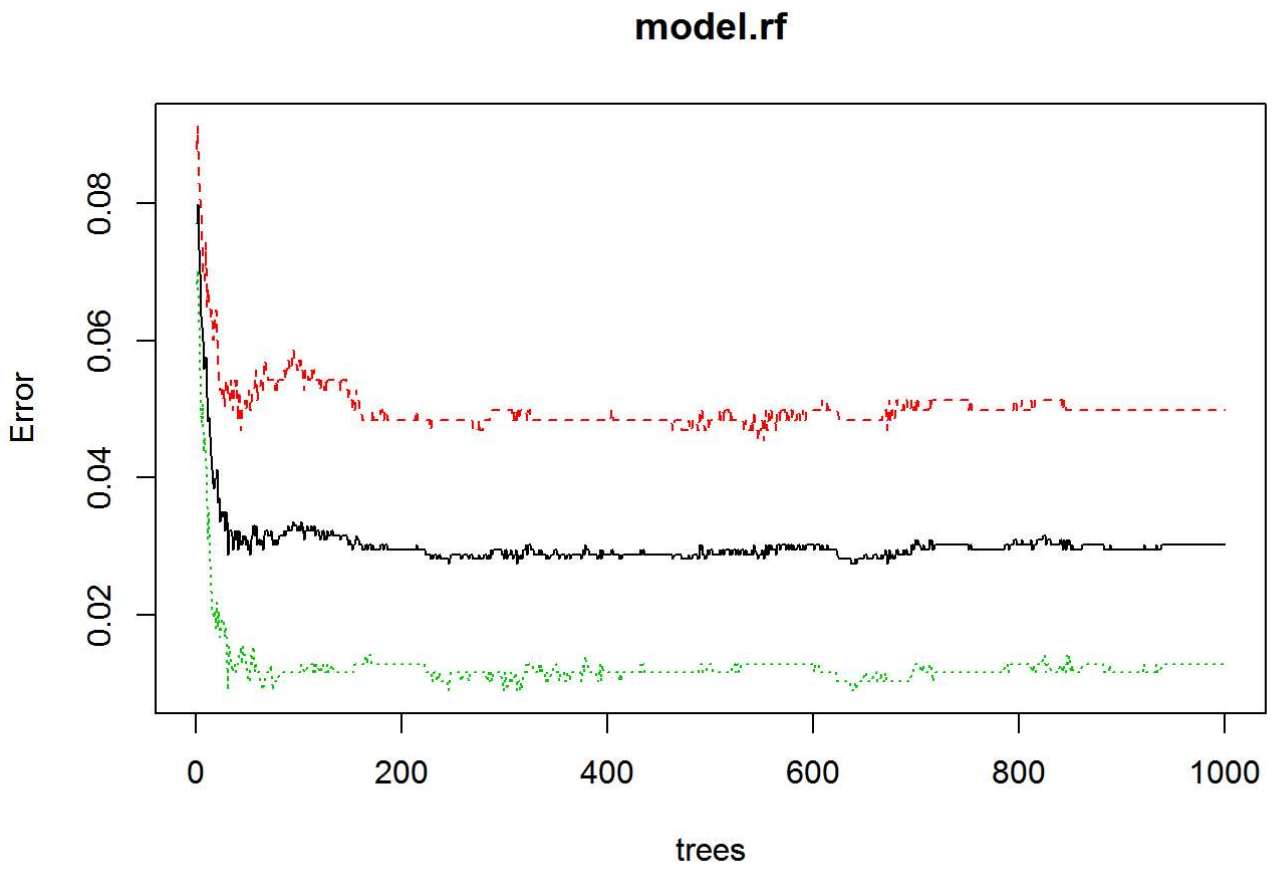
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 186 104
##        No  116 231
##
##                Accuracy : 0.6546
##                  95% CI : (0.6163, 0.6916)
##     No Information Rate : 0.5259
##     P-Value [Acc > NIR] : 3.252e-11
##
##                   Kappa : 0.306
##  Mcnemar's Test P-Value : 0.4583
##
##             Sensitivity : 0.6159
##             Specificity : 0.6896
##          Pos Pred Value : 0.6414
##          Neg Pred Value : 0.6657
##              Prevalence : 0.4741
##          Detection Rate : 0.2920
##    Detection Prevalence : 0.4553
##       Balanced Accuracy : 0.6527
##
##        'Positive' Class : Yes
##
```

```
roc.curve(knn.model, testSplit2$Class, plotit = T)
```



```
## Area under the curve (AUC): 0.654
```

```
### Applying Random Forest algorithm for 50% fraud data
model.rf <- randomForest(Class ~ ., data =trainSplit2 , ntree = 1000, importance = TRUE)
plot(model.rf)
```



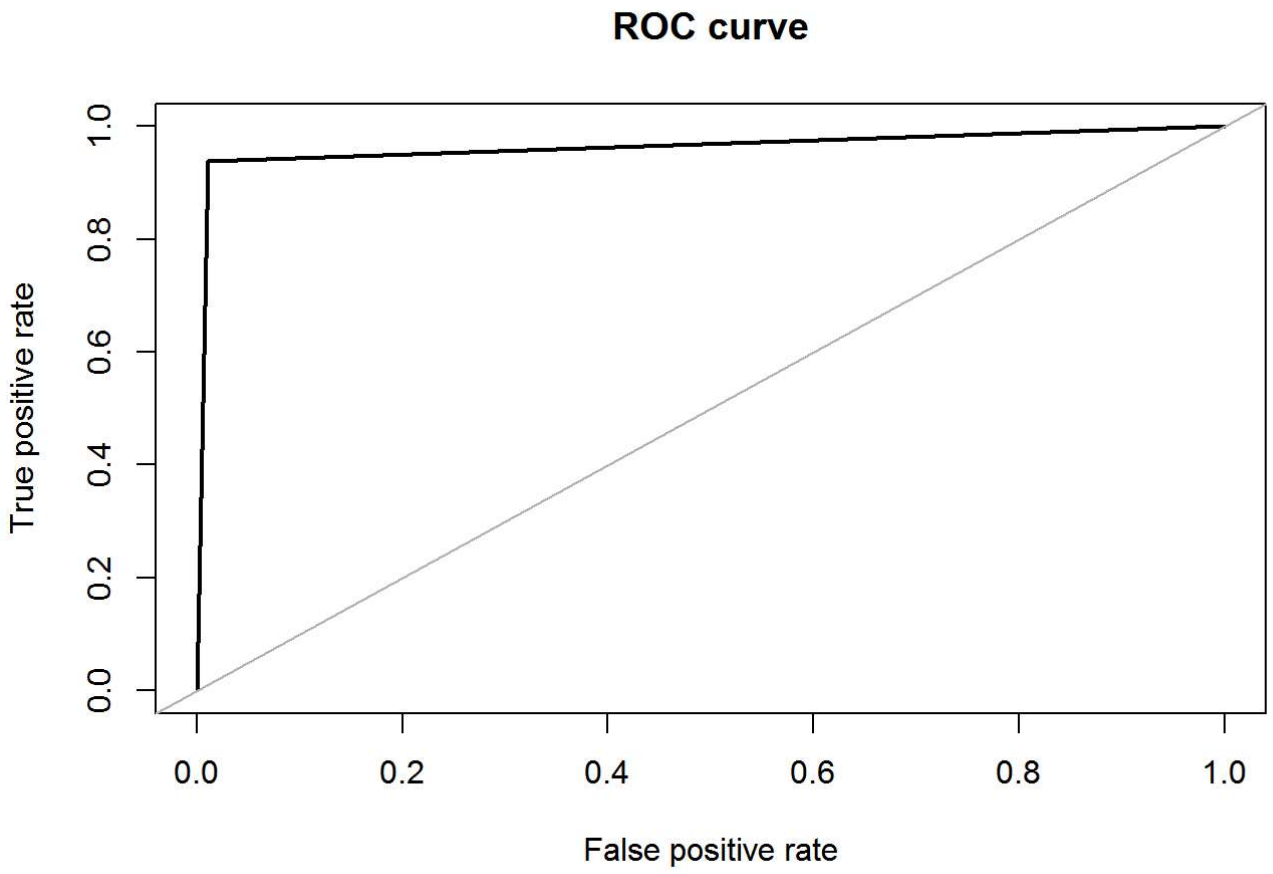

```
cv.tree.pred2 <- predict(model.rf, testSplit2)

# Making table of Confusion matrix
CrossTable(cv.tree.pred2, testSplit2$Class,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
           dnn = c('actual class', 'predicted class'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  637
##
##
##              | predicted class
## actual class |       Yes |        No | Row Total |
## -------------|-----------|-----------|-----------|
##          Yes |       280 |         3 |       283 |
##              |     0.440 |     0.005 |           |
## -------------|-----------|-----------|-----------|
##           No |        22 |       332 |       354 |
##              |     0.035 |     0.521 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       302 |       335 |       637 |
## -------------|-----------|-----------|-----------|
##
##
```

```
confusionMatrix(cv.tree.pred2, testSplit2$Class)
```

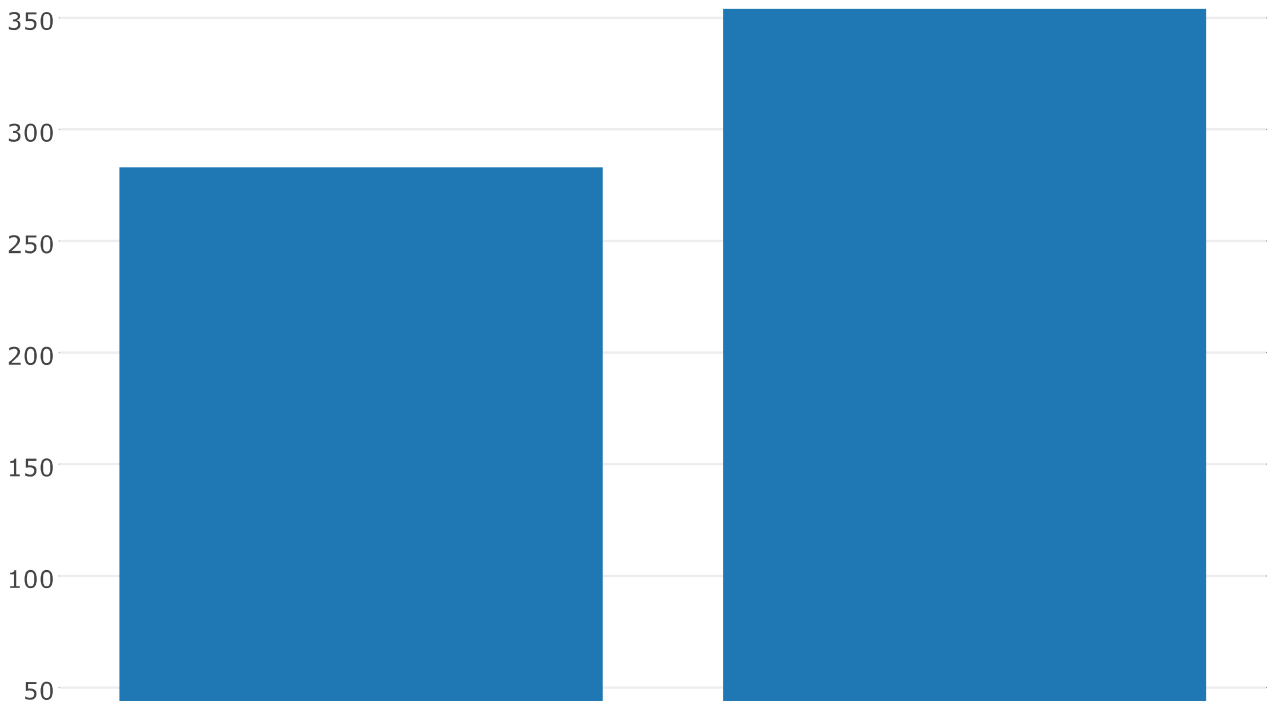
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 280   3
##        No   22 332
##
##                Accuracy : 0.9608
##                  95% CI : (0.9426, 0.9744)
##     No Information Rate : 0.5259
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9211
##  Mcnemar's Test P-Value : 0.0003182
##
##             Sensitivity : 0.9272
##             Specificity : 0.9910
##          Pos Pred Value : 0.9894
##          Neg Pred Value : 0.9379
##              Prevalence : 0.4741
##          Detection Rate : 0.4396
##    Detection Prevalence : 0.4443
##       Balanced Accuracy : 0.9591
##
##        'Positive' Class : Yes
##
```

```
roc.curve(cv.tree.pred2, testSplit2$Class, plotit = T)
```




```
## Area under the curve (AUC): 0.964
```

```
x<-rchisq(1000,5,0)
plot_ly(x=cv.tree.pred2,type = 'histogram')
```



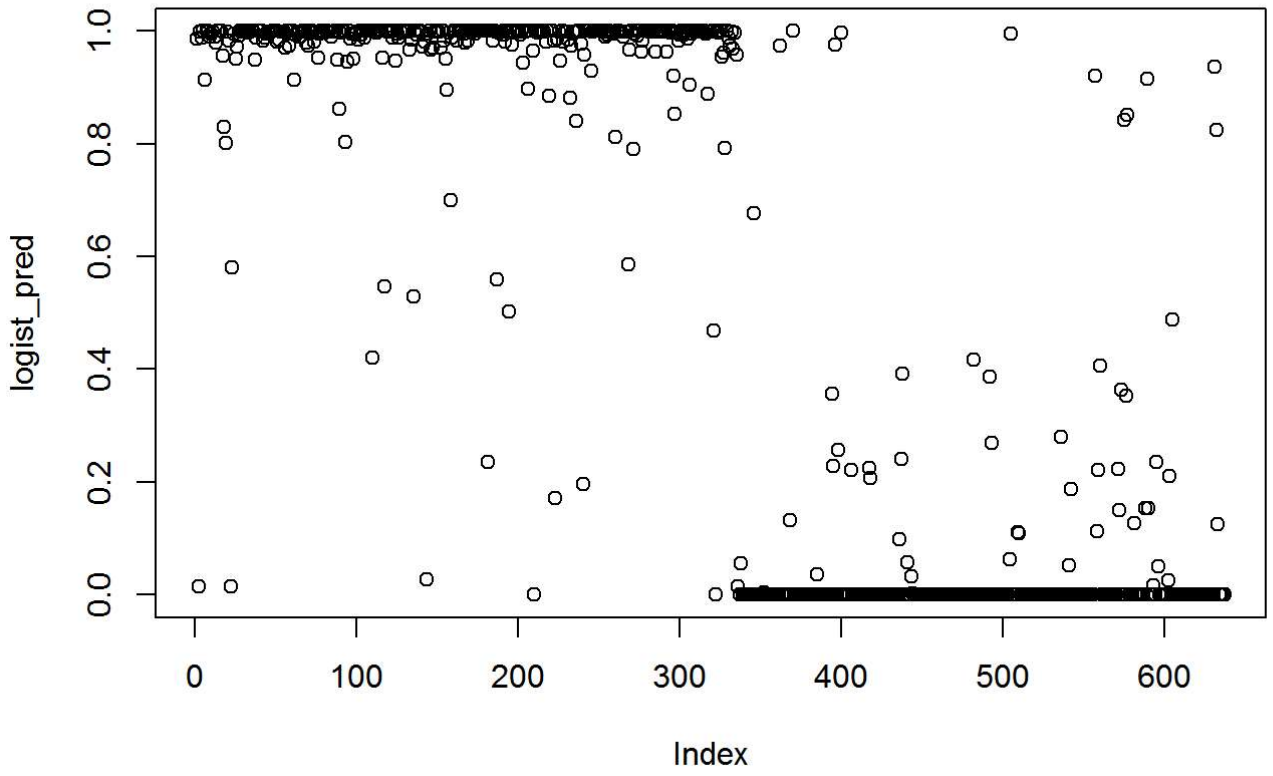

```
### Applying Logistic Regression Algorithm for 50% fraud data
logist <- glm(Class ~ ., data = trainSplit2, family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(logist)
```

```
##
## Call:
## glm(formula = Class ~ ., family = "binomial", data = trainSplit2)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.96702   0.00000   0.00000   0.05639   2.99552
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.935e+01  8.519e+01   0.462  0.64412
## Time         4.322e-05  1.069e-05   4.041 5.32e-05 ***
## V1           1.328e+01  1.934e+01   0.686  0.49245
## V2          -9.543e+01  1.257e+02  -0.759  0.44766
## V3           7.865e+01  5.023e+01   1.566  0.11740
## V4          -5.247e+01  4.030e+01  -1.302  0.19294
## V5           3.194e+01  9.969e+00   3.204  0.00136 **
## V6           4.568e+01  5.717e+01   0.799  0.42427
## V7           1.783e+02  1.972e+02   0.904  0.36601
## V8          -3.569e+01  3.359e+01  -1.062  0.28810
## V9           6.577e+01  6.051e+01   1.087  0.27707
## V10          1.525e+02  1.391e+02   1.096  0.27323
## V11         -1.165e+02  1.178e+02  -0.990  0.32232
## V12          2.107e+02  2.115e+02   0.996  0.31915
## V13          2.256e+00  5.561e+00   0.406  0.68492
## V14          2.253e+02  2.307e+02   0.977  0.32879
## V15          6.782e+00  8.228e+00   0.824  0.40981
## V16          2.004e+02  2.036e+02   0.984  0.32500
## V17          3.570e+02  3.574e+02   0.999  0.31787
## V18          1.358e+02  1.366e+02   0.994  0.32025
## V19         -5.114e+01  5.654e+01  -0.905  0.36567
## V20          1.215e+01  3.828e+01   0.317  0.75093
## V21         -1.792e+01  9.703e+00  -1.847  0.06470 .
## V22         -1.663e+01  2.500e+01  -0.665  0.50579
## V23         -4.205e+01  7.549e+01  -0.557  0.57753
## V24          5.465e+00  7.228e+00   0.756  0.44956
## V25         -2.234e+01  3.437e+01  -0.650  0.51569
## V26         -3.689e+00  8.678e+00  -0.425  0.67075
## V27         -2.954e+01  2.718e+01  -1.087  0.27714
## V28         -5.311e+01  9.139e+01  -0.581  0.56114
## Amount      -5.067e-01  8.720e-01  -0.581  0.56118
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2010.10  on 1453  degrees of freedom
## Residual deviance:  171.25  on 1423  degrees of freedom
## AIC: 233.25
##
## Number of Fisher Scoring iterations: 25
```

```
logist_pred <- predict(logist, newdata=testSplit2,type = "response")
plot(logist_pred)
```

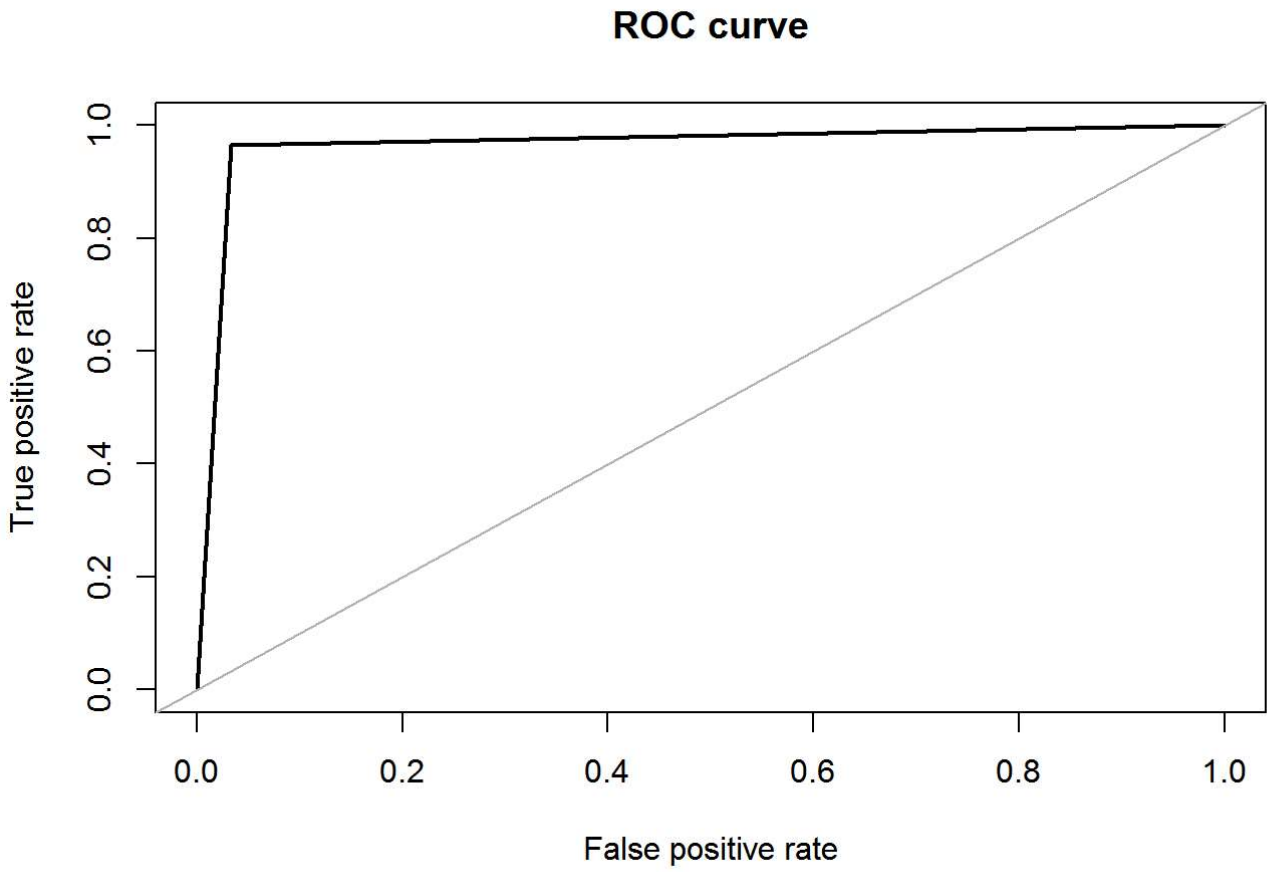



```
pred=rep("Yes",length(logist_pred))
pred[logist_pred > 0.5] = "No"
confusionMatrix(testSplit2$Class, pred)
```

```
## Warning in confusionMatrix.default(testSplit2$Class, pred): Levels are not
## in the same order for reference and data. Refactoring data to match.
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  325  10
##        Yes  12 290
##
##                Accuracy : 0.9655
##                  95% CI : (0.9482, 0.9782)
##     No Information Rate : 0.529
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9307
##  Mcnemar's Test P-Value : 0.8312
##
##             Sensitivity : 0.9644
##             Specificity : 0.9667
##          Pos Pred Value : 0.9701
##          Neg Pred Value : 0.9603
##              Prevalence : 0.5290
##          Detection Rate : 0.5102
##    Detection Prevalence : 0.5259
##       Balanced Accuracy : 0.9655
##
##        'Positive' Class : No
##
```

```
roc.curve(pred, testSplit2$Class, plotit = T)
```



```
## Area under the curve (AUC): 0.966
```

```
#############################################################################
### New dataset with 75% fraud data
fraud75 = fraudData[1:369,]
dataWith75Fraud = rbind(NofraudData,fraud75)

#Randomize the data for 75% fraud
dataFor75 <- dataWith75Fraud[sample(nrow(dataWith75Fraud)),]
table(dataFor75$Class)
```

```
##
##    Yes     No
##    369 284315
```

```
###removing unbalanced classification problemand making data balanced
set.seed(4356)
smote_data <- SMOTE(Class ~ ., data  = dataFor75, perc.over = 300, perc.under = 150, k=5)
new.data <- sample(2, nrow(smote_data), replace = TRUE, prob = c(0.7, 0.3))
trainSplit2 <-smote_data[new.data==1,]
testSplit2 <-smote_data[new.data==2,]
table(trainSplit2$Class)
```

```
##
##  Yes   No
## 1026 1173
```

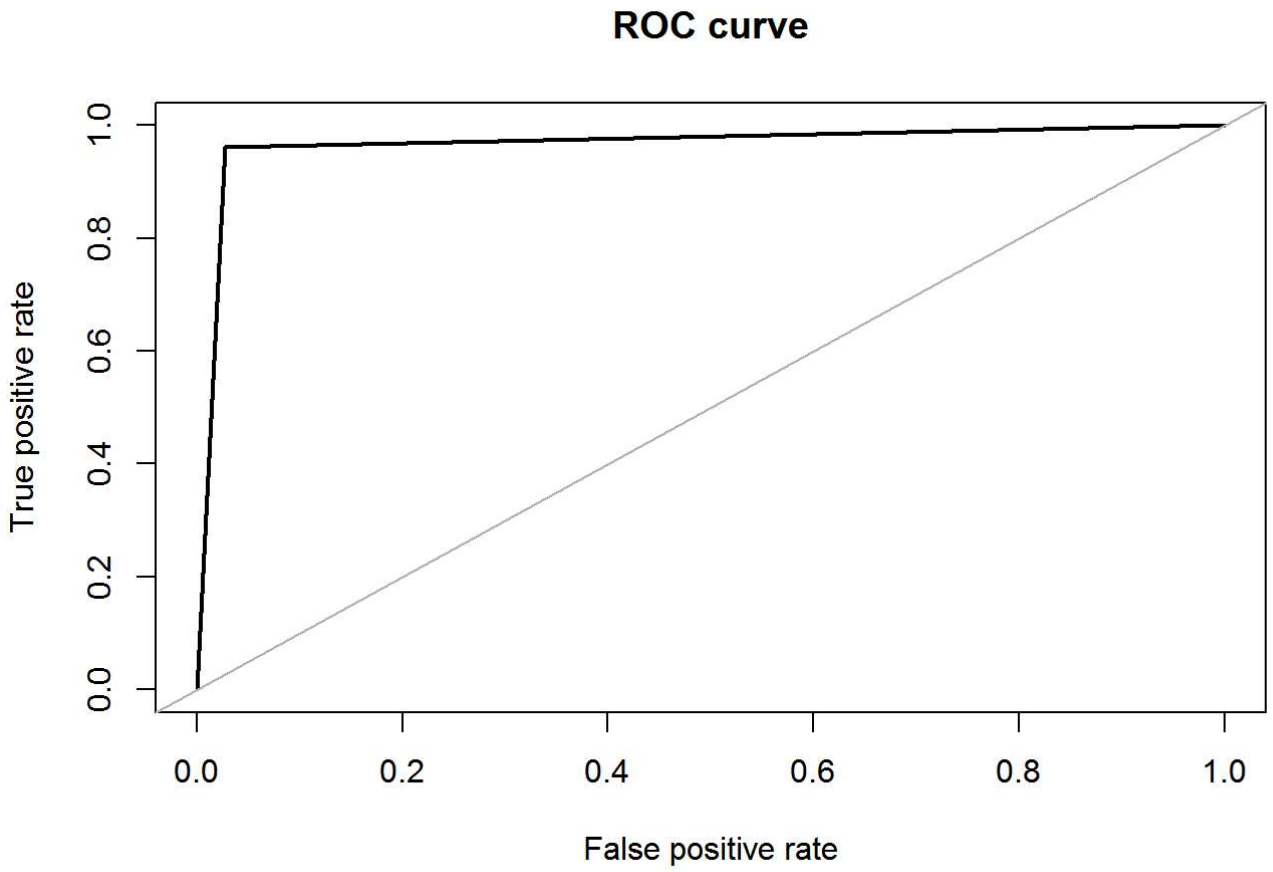
```
table(testSplit2$Class)
```

```
##
## Yes  No
## 450 487
```

```
### Applying SVM algorithm for 75% fraud data
svm.model <- svm(Class ~ ., data = trainSplit2, kernel = "radial", cost = 1, gamma = 0.1)
svm.predict <- predict(svm.model, testSplit2)
confusionMatrix(testSplit2$Class, svm.predict)
```

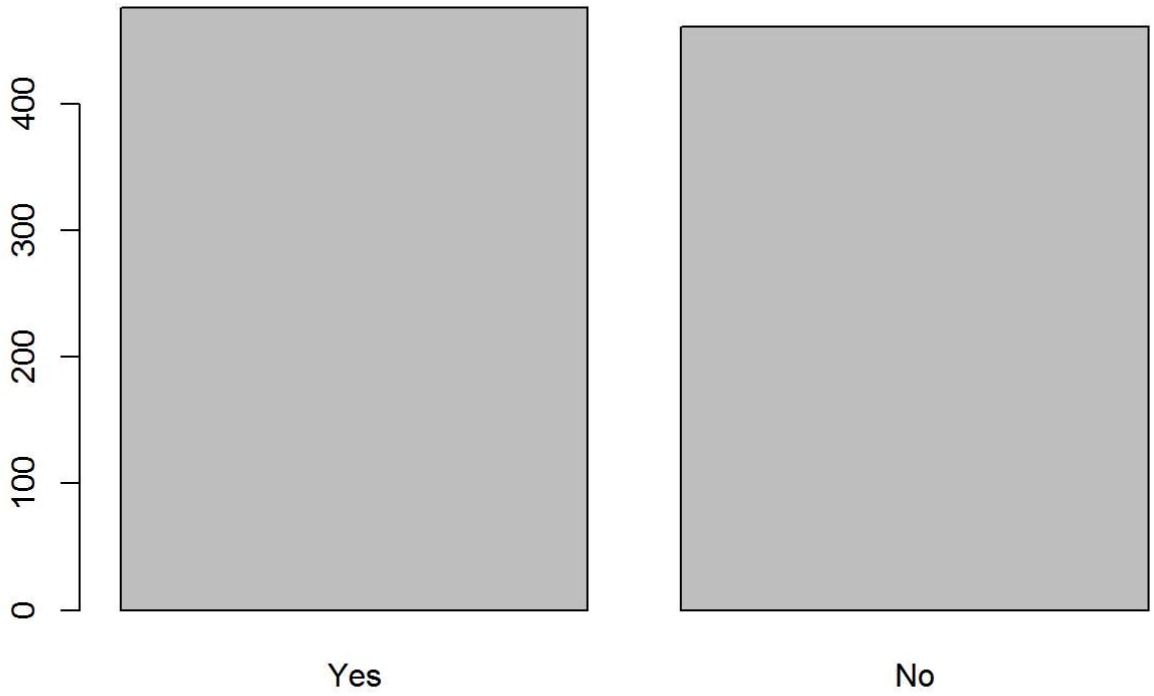
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 431  19
##        No   12 475
##
##                Accuracy : 0.9669
##                  95% CI : (0.9534, 0.9774)
##     No Information Rate : 0.5272
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9337
##  Mcnemar's Test P-Value : 0.2812
##
##             Sensitivity : 0.9729
##             Specificity : 0.9615
##          Pos Pred Value : 0.9578
##          Neg Pred Value : 0.9754
##              Prevalence : 0.4728
##          Detection Rate : 0.4600
##    Detection Prevalence : 0.4803
##       Balanced Accuracy : 0.9672
##
##        'Positive' Class : Yes
##
```

```
roc.curve(svm.predict, testSplit2$Class, plotit = T)
```



```
## Area under the curve (AUC): 0.967
```

```
## Applying KNN Algorithm
knn.model <- knn(train = trainSplit2[,1:30],
    test = testSplit2[,1:30],
    cl = trainSplit2$Class)
plot(knn.model)
```



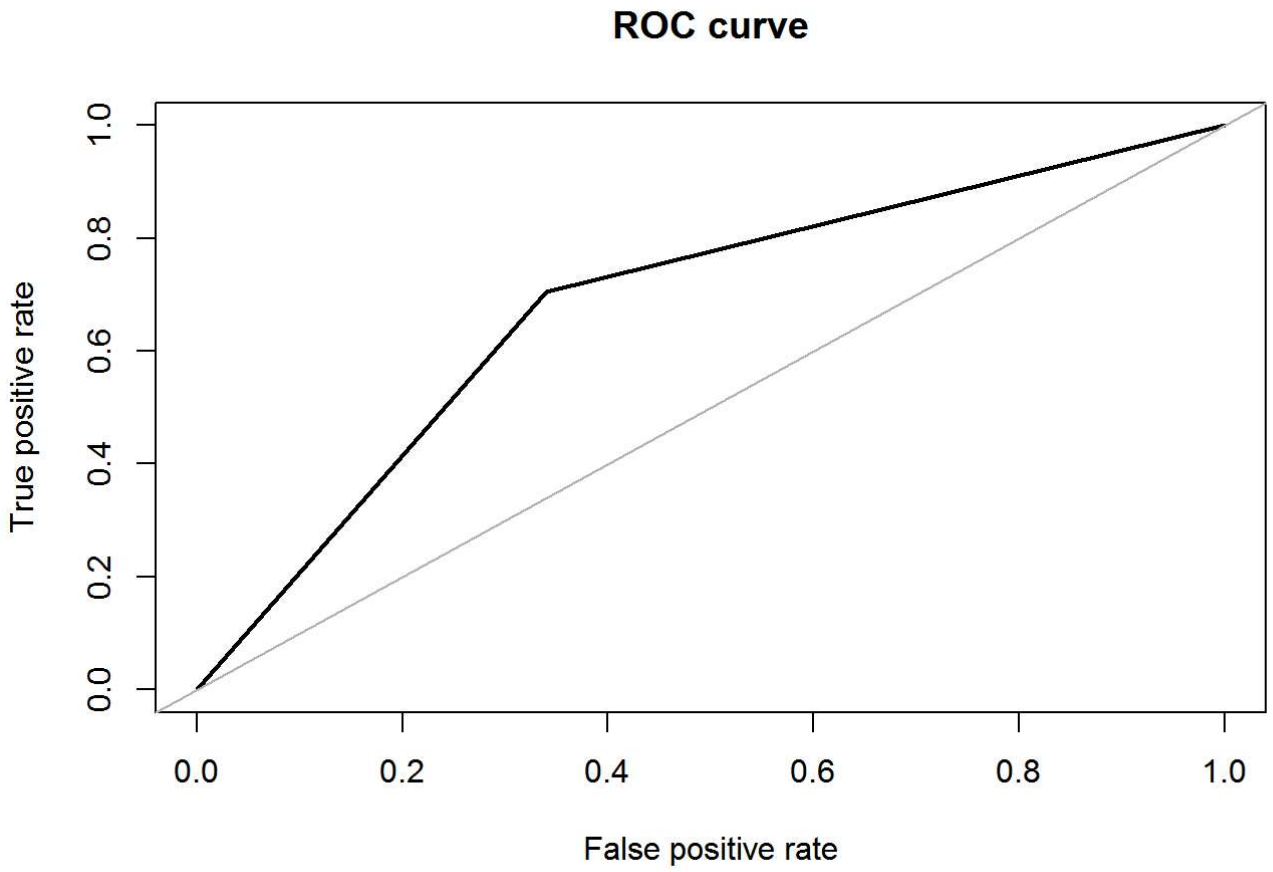

```
table(knn.model, testSplit2$Class)
```

```
##
## knn.model Yes  No
##       Yes 314 162
##       No  136 325
```

```
confusionMatrix(knn.model, testSplit2[,31])
```

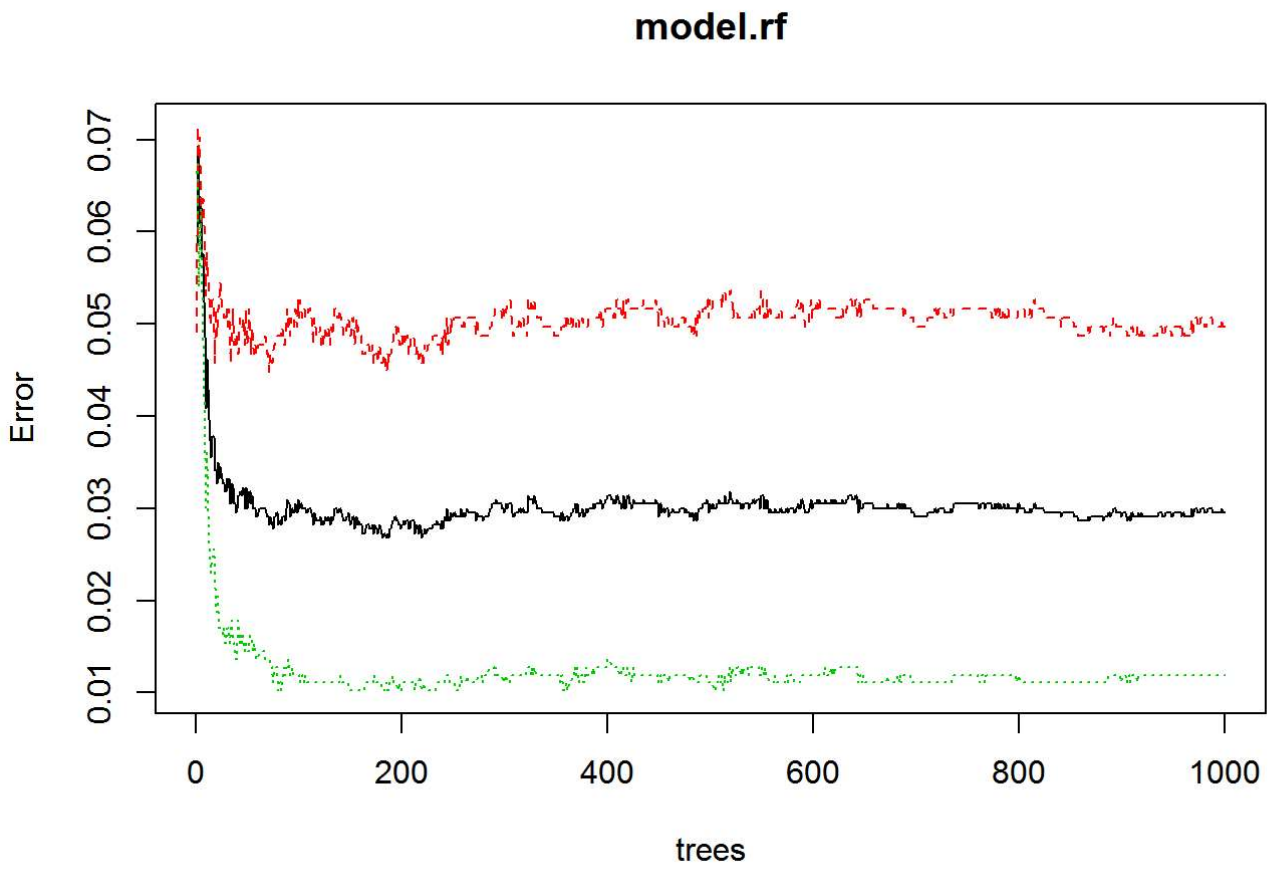
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 314 162
##        No  136 325
##
##                Accuracy : 0.682
##                  95% CI : (0.6511, 0.7117)
##     No Information Rate : 0.5197
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.3643
##  Mcnemar's Test P-Value : 0.1476
##
##             Sensitivity : 0.6978
##             Specificity : 0.6674
##          Pos Pred Value : 0.6597
##          Neg Pred Value : 0.7050
##              Prevalence : 0.4803
##          Detection Rate : 0.3351
##    Detection Prevalence : 0.5080
##       Balanced Accuracy : 0.6826
##
##        'Positive' Class : Yes
##
```

```
roc.curve(knn.model, testSplit2$Class, plotit = T)
```



```
## Area under the curve (AUC): 0.682
```

```
### Applying Random Forest algorithm for 75% fraud data
model.rf <- randomForest(Class ~ ., data =trainSplit2 , ntree = 1000, importance = TRUE)
plot(model.rf)
```



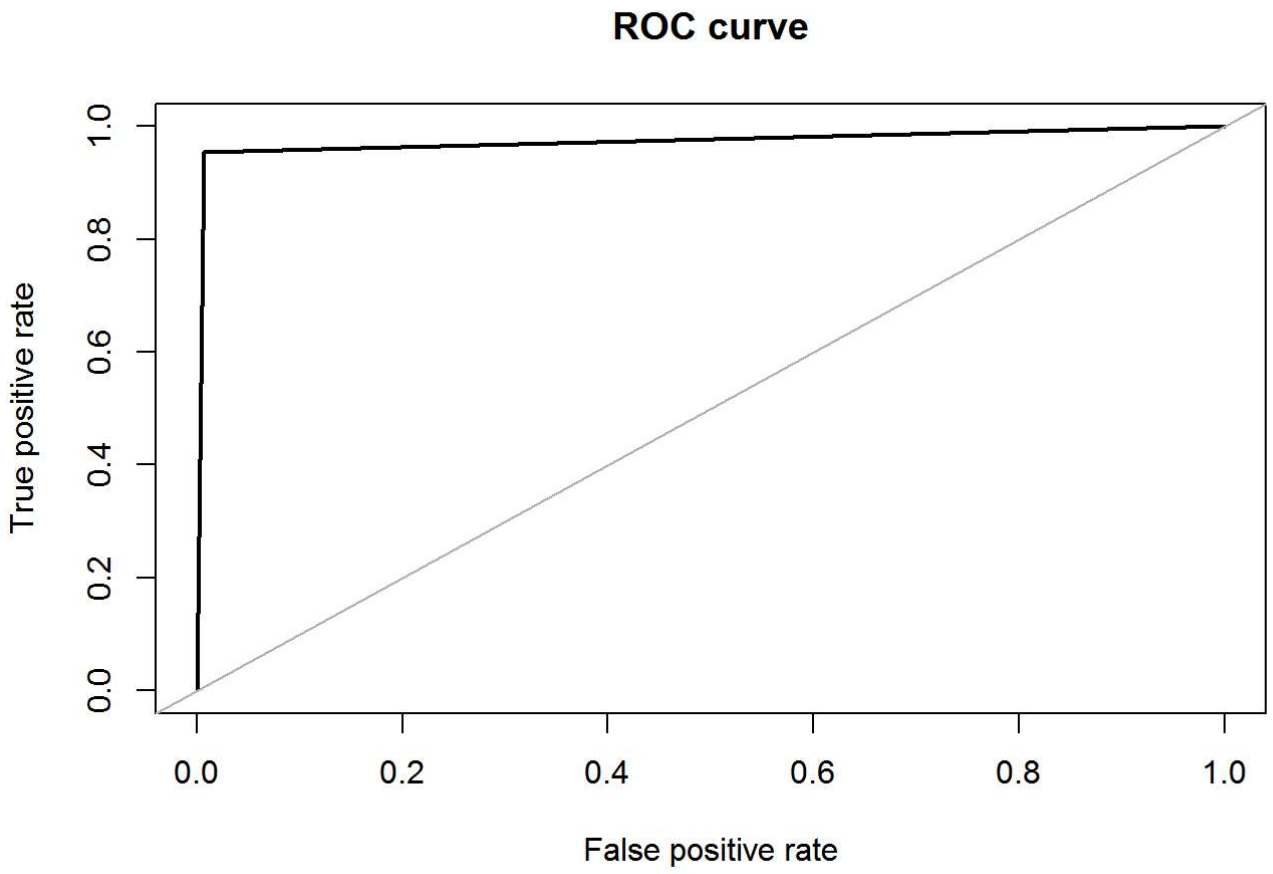

```
cv.tree.pred2 <- predict(model.rf, testSplit2)

# Making table of Confusion matrix
CrossTable(cv.tree.pred2, testSplit2$Class,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
           dnn = c('actual class', 'predicted class'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  937
##
##
##              | predicted class
## actual class |       Yes |        No | Row Total |
## -------------|-----------|-----------|-----------|
##          Yes |       427 |         3 |       430 |
##              |     0.456 |     0.003 |           |
## -------------|-----------|-----------|-----------|
##           No |        23 |       484 |       507 |
##              |     0.025 |     0.517 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       450 |       487 |       937 |
## -------------|-----------|-----------|-----------|
##
##
```

```
confusionMatrix(cv.tree.pred2, testSplit2$Class)
```

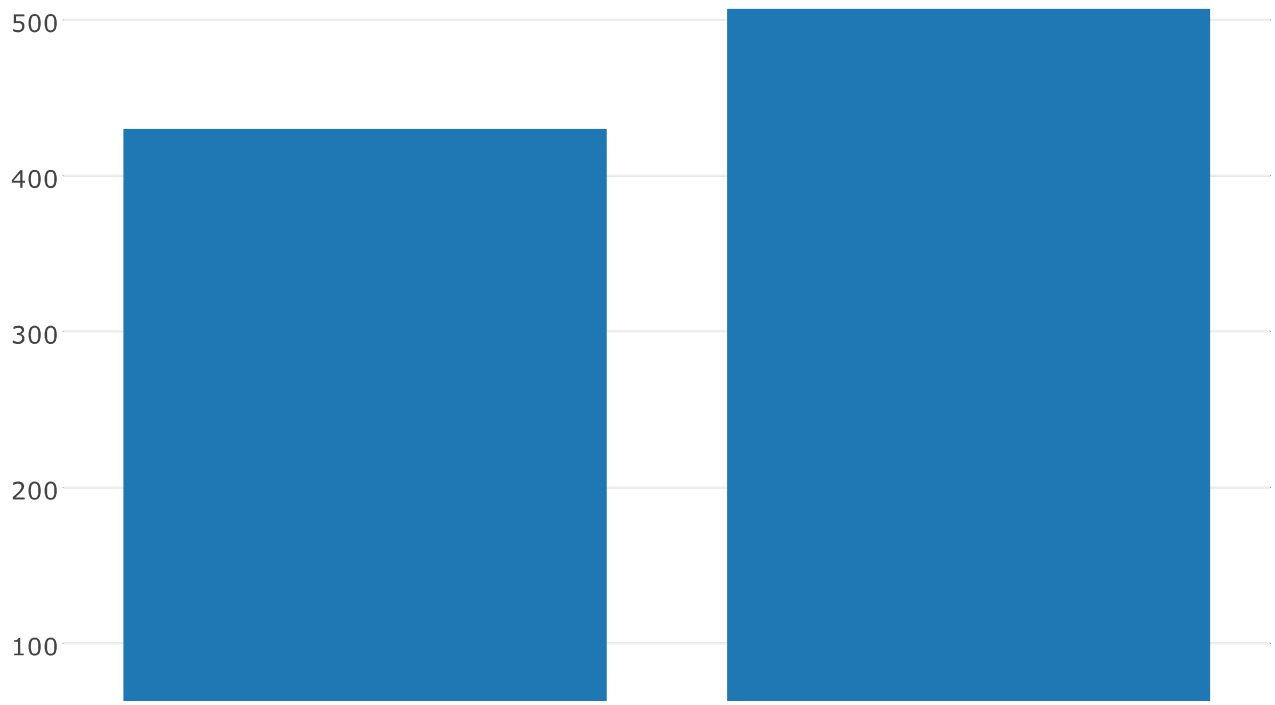
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 427   3
##        No   23 484
##
##                Accuracy : 0.9723
##                  95% CI : (0.9596, 0.9818)
##     No Information Rate : 0.5197
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9443
##  Mcnemar's Test P-Value : 0.0001944
##
##             Sensitivity : 0.9489
##             Specificity : 0.9938
##          Pos Pred Value : 0.9930
##          Neg Pred Value : 0.9546
##              Prevalence : 0.4803
##          Detection Rate : 0.4557
##    Detection Prevalence : 0.4589
##       Balanced Accuracy : 0.9714
##
##        'Positive' Class : Yes
##
```

```
roc.curve(cv.tree.pred2, testSplit2$Class, plotit = T)
```




```
## Area under the curve (AUC): 0.974
```

```
x<-rchisq(1000,5,0)
plot_ly(x=cv.tree.pred2,type = 'histogram')
```



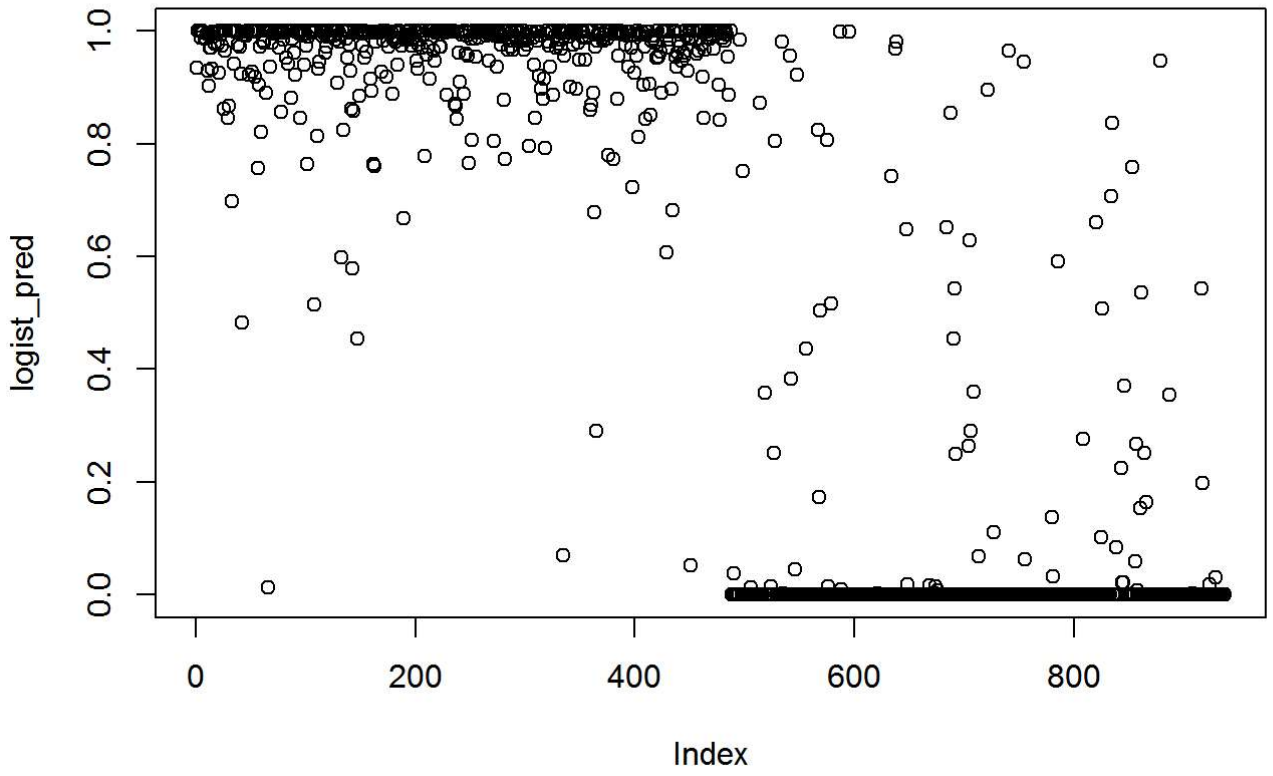

```
### Applying Logistic Regression Algorithm for 75% fraud data
logist <- glm(Class ~ ., data = trainSplit2, family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(logist)
```

```
##
## Call:
## glm(formula = Class ~ ., family = "binomial", data = trainSplit2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6113   0.0000   0.0000   0.1607   3.0528
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.515e+01  2.701e+01   0.931 0.351820
## Time         1.015e-05  4.314e-06   2.354 0.018594 *
## V1           7.164e+00  6.644e+00   1.078 0.280949
## V2          -5.338e+01  3.983e+01  -1.340 0.180183
## V3           4.507e+01  1.681e+01   2.681 0.007337 **
## V4          -3.114e+01  1.318e+01  -2.362 0.018155 *
## V5           1.847e+01  5.132e+00   3.599 0.000319 ***
## V6           2.537e+01  1.812e+01   1.400 0.161523
## V7           1.014e+02  6.275e+01   1.616 0.106192
## V8          -1.917e+01  1.074e+01  -1.785 0.074200 .
## V9           3.830e+01  1.945e+01   1.969 0.048993 *
## V10          8.962e+01  4.477e+01   2.002 0.045296 *
## V11         -6.810e+01  3.767e+01  -1.808 0.070625 .
## V12          1.227e+02  6.765e+01   1.814 0.069634 .
## V13          1.480e+00  1.768e+00   0.837 0.402575
## V14          1.312e+02  7.371e+01   1.779 0.075185 .
## V15          3.495e+00  2.612e+00   1.338 0.180820
## V16          1.158e+02  6.508e+01   1.780 0.075051 .
## V17          2.077e+02  1.144e+02   1.816 0.069324 .
## V18          7.892e+01  4.367e+01   1.807 0.070737 .
## V19         -2.973e+01  1.801e+01  -1.651 0.098723 .
## V20          6.410e+00  1.225e+01   0.523 0.600936
## V21         -1.082e+01  3.448e+00  -3.138 0.001699 **
## V22         -9.449e+00  7.923e+00  -1.193 0.233020
## V23         -2.372e+01  2.392e+01  -0.991 0.321507
## V24          2.870e+00  2.291e+00   1.253 0.210274
## V25         -1.291e+01  1.089e+01  -1.186 0.235554
## V26         -2.388e+00  2.760e+00  -0.865 0.386868
## V27         -1.735e+01  8.843e+00  -1.962 0.049756 *
## V28         -3.276e+01  2.893e+01  -1.132 0.257567
## Amount      -2.825e-01  2.763e-01  -1.022 0.306585
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3038.63  on 2198  degrees of freedom
## Residual deviance:  431.95  on 2168  degrees of freedom
## AIC: 493.95
##
## Number of Fisher Scoring iterations: 25
```

```
logist_pred <- predict(logist, newdata=testSplit2,type = "response")
plot(logist_pred)
```

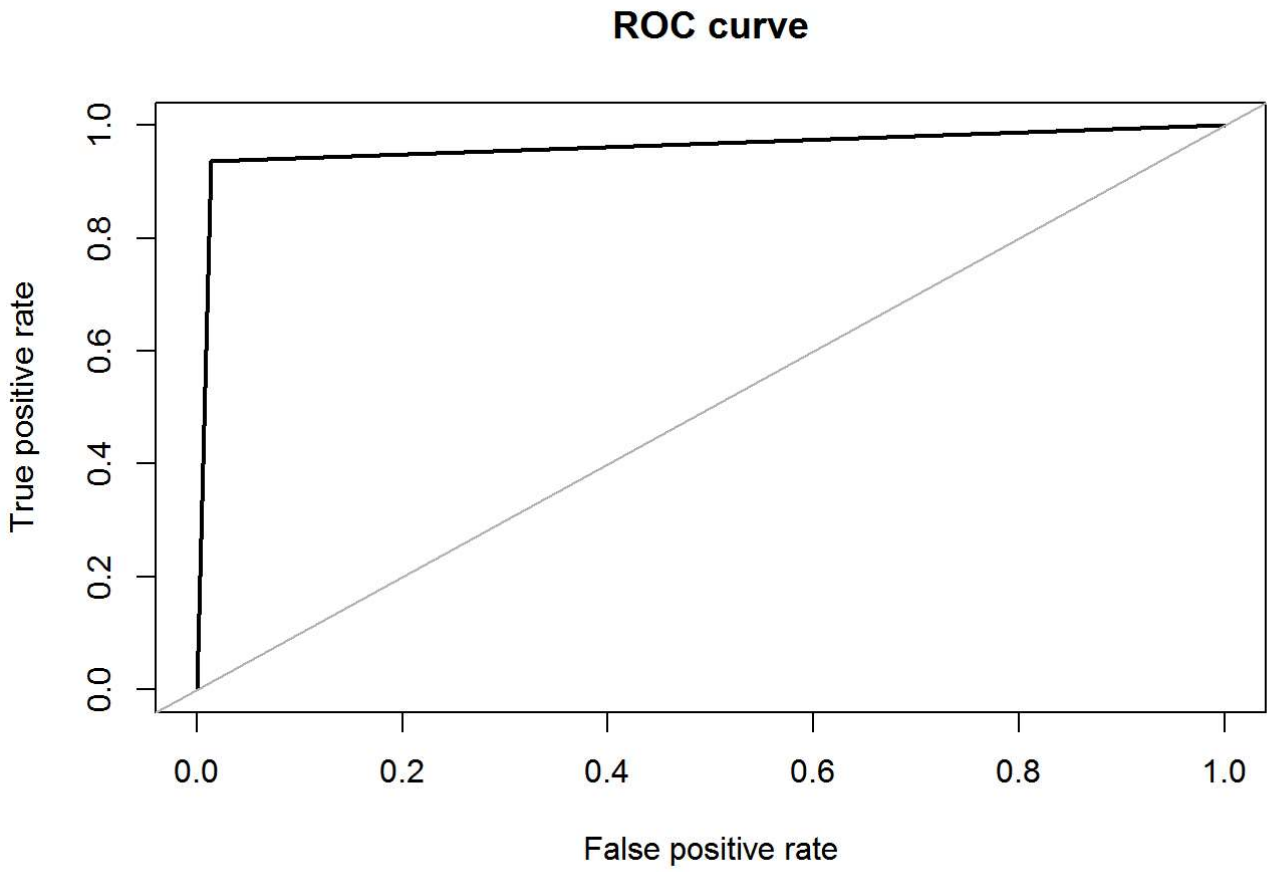



```
pred=rep("Yes",length(logist_pred))
pred[logist_pred > 0.5] = "No"
confusionMatrix(testSplit2$Class, pred)
```

```
## Warning in confusionMatrix.default(testSplit2$Class, pred): Levels are not
## in the same order for reference and data. Refactoring data to match.
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  481   6
##        Yes  33 417
##
##                Accuracy : 0.9584
##                  95% CI : (0.9435, 0.9702)
##     No Information Rate : 0.5486
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9164
##  Mcnemar's Test P-Value : 3.136e-05
##
##             Sensitivity : 0.9358
##             Specificity : 0.9858
##          Pos Pred Value : 0.9877
##          Neg Pred Value : 0.9267
##              Prevalence : 0.5486
##          Detection Rate : 0.5133
##    Detection Prevalence : 0.5197
##       Balanced Accuracy : 0.9608
##
##        'Positive' Class : No
##
```

```
roc.curve(pred, testSplit2$Class, plotit = T)
```



```
## Area under the curve (AUC): 0.961
```

```
#############################################################################
### New dataset with 100% fraud data
fraud100 = fraudData[1:492,]

dataWith100Fraud = rbind(NofraudData,fraud100)

#Randomize the data for 100% fraud
dataFor100 <- dataWith100Fraud[sample(nrow(dataWith100Fraud)),]
table(dataFor100$Class)
```

```
##
##    Yes     No
##    492 284315
```

```
###removing unbalanced classification problemand making data balanced
set.seed(4356)
smote_data <- SMOTE(Class ~ ., data  = dataFor100, perc.over = 300, perc.under = 150, k=5)
new.data <- sample(2, nrow(smote_data), replace = TRUE, prob = c(0.7, 0.3))
trainSplit2 <-smote_data[new.data==1,]
testSplit2 <-smote_data[new.data==2,]
table(trainSplit2$Class)
```

```
##
##  Yes   No
## 1360 1560
```

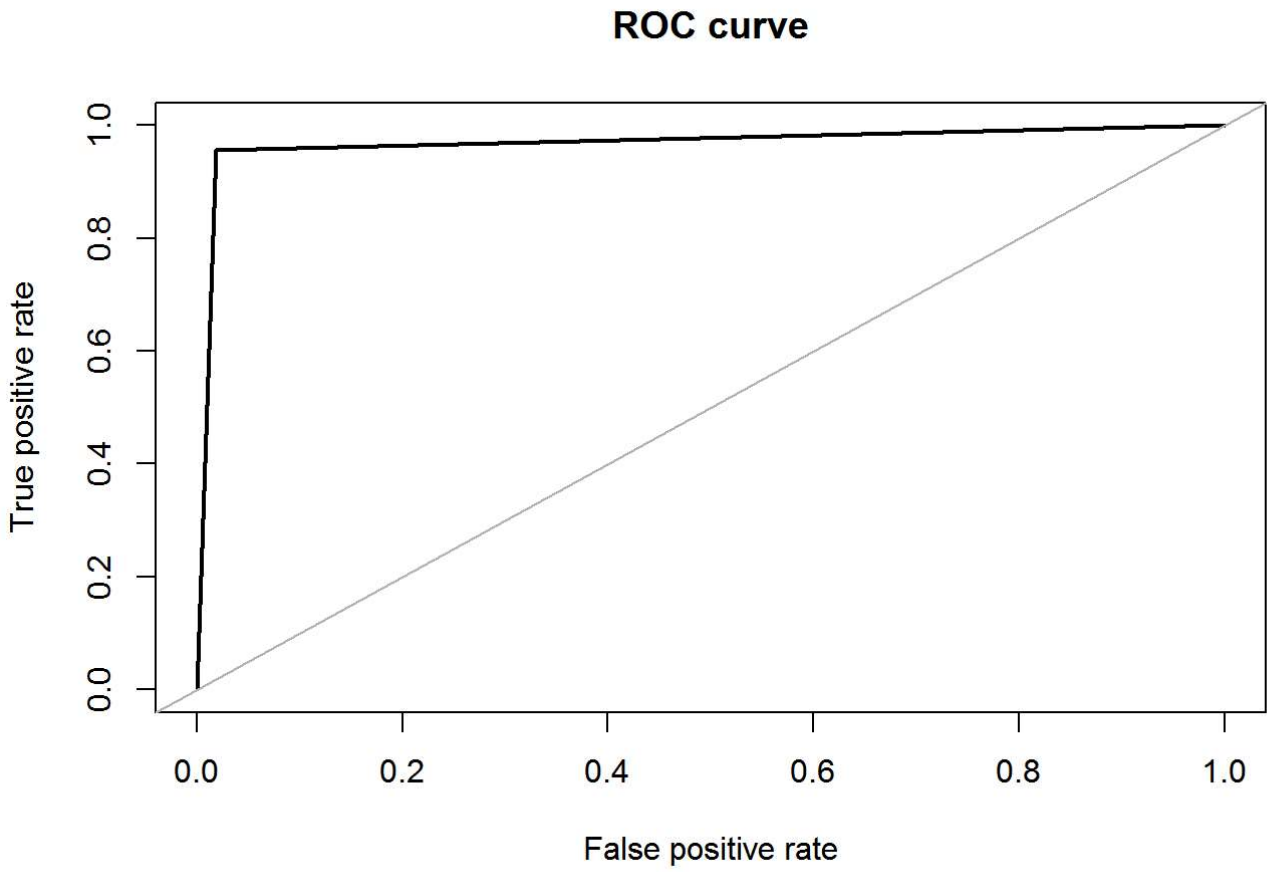
```
table(testSplit2$Class)
```

```
##
## Yes  No
## 608 654
```

```
### Applying SVM algorithm for 100% fraud data
svm.model <- svm(Class ~ ., data = trainSplit2, kernel = "radial", cost = 1, gamma = 0.1)
svm.predict <- predict(svm.model, testSplit2)
confusionMatrix(testSplit2$Class, svm.predict)
```

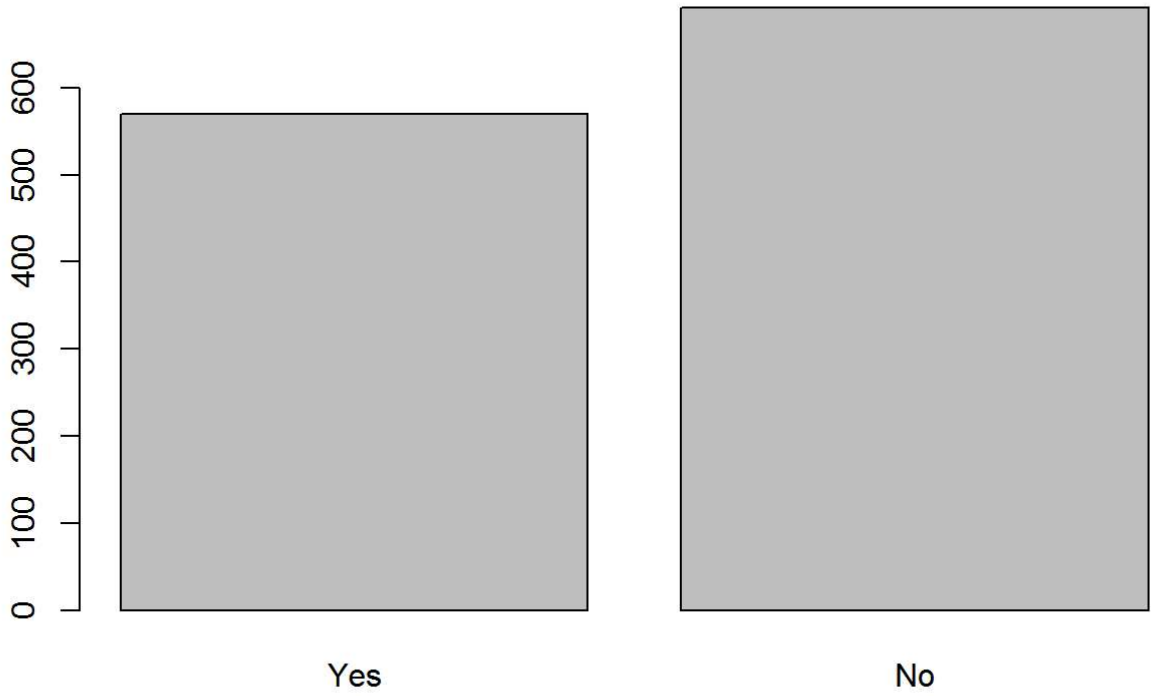
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 579  29
##        No   11 643
##
##                Accuracy : 0.9683
##                  95% CI : (0.9571, 0.9773)
##     No Information Rate : 0.5325
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9365
##  Mcnemar's Test P-Value : 0.00719
##
##             Sensitivity : 0.9814
##             Specificity : 0.9568
##          Pos Pred Value : 0.9523
##          Neg Pred Value : 0.9832
##              Prevalence : 0.4675
##          Detection Rate : 0.4588
##    Detection Prevalence : 0.4818
##       Balanced Accuracy : 0.9691
##
##        'Positive' Class : Yes
##
```

```
roc.curve(svm.predict, testSplit2$Class, plotit = T)
```



```
## Area under the curve (AUC): 0.969
```

```
## Applying KNN Algorithm
knn.model <- knn(train = trainSplit2[,1:30],
    test = testSplit2[,1:30],
    cl = trainSplit2$Class)
plot(knn.model)
```



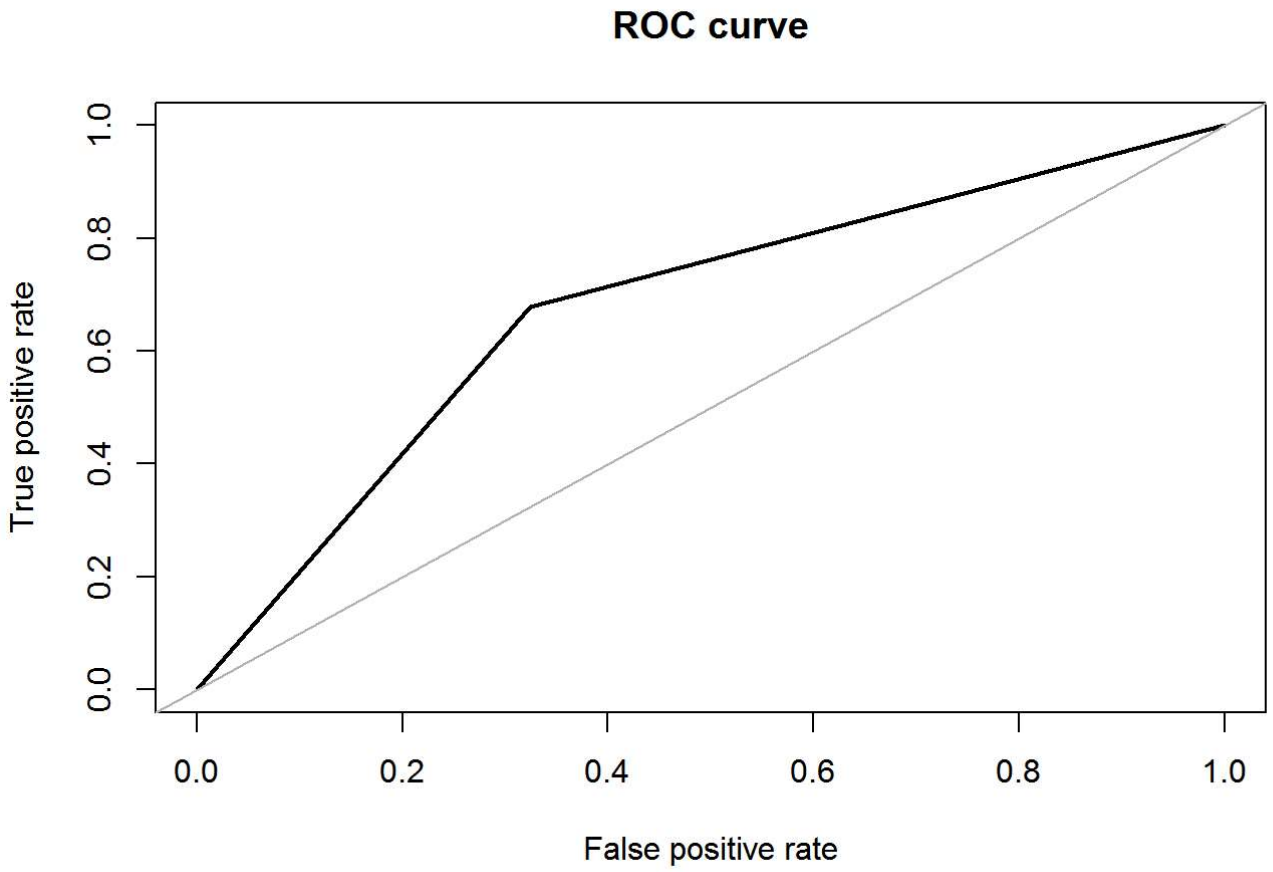

```
table(knn.model, testSplit2$Class)
```

```
##
## knn.model Yes  No
##       Yes 385 185
##       No  223 469
```

```
confusionMatrix(knn.model, testSplit2[,31])
```

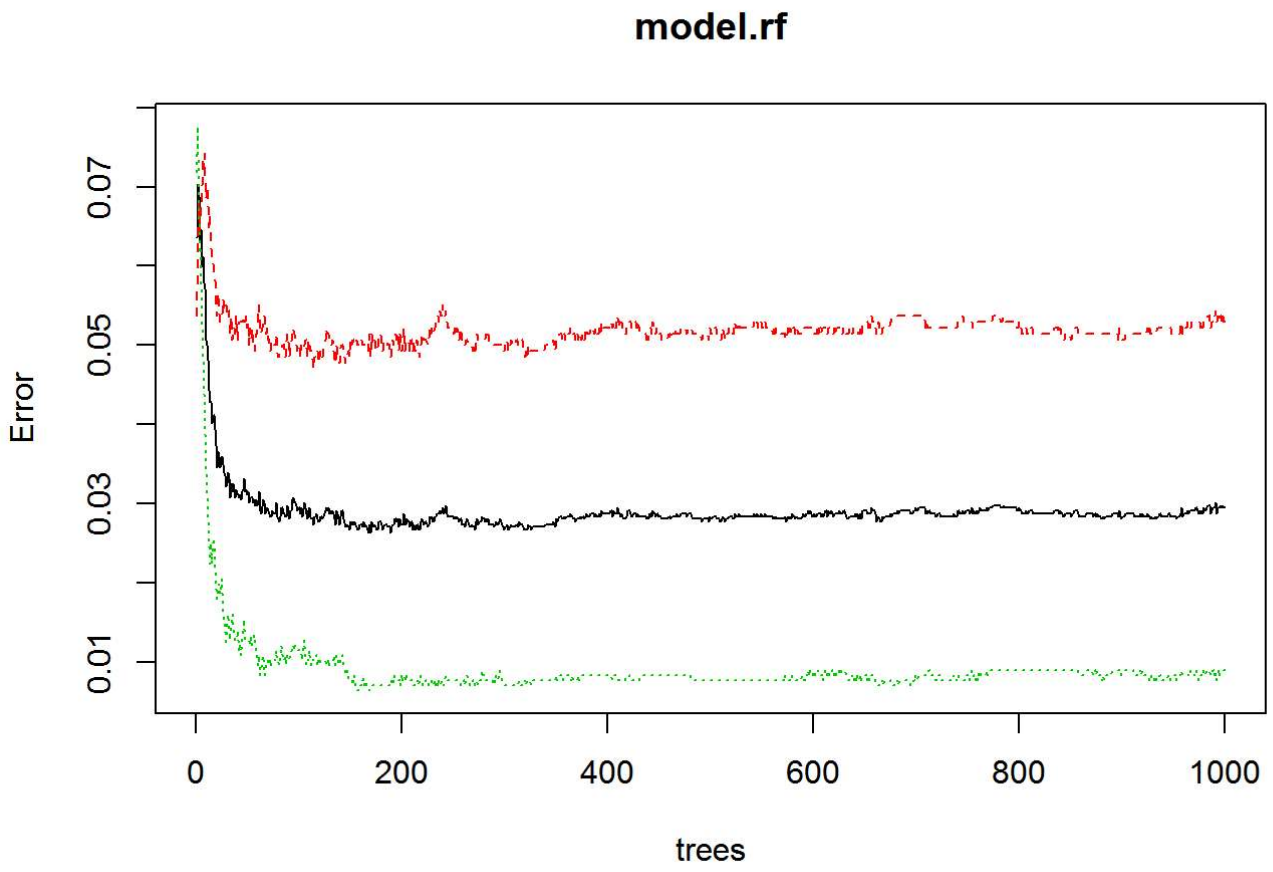
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 385 185
##        No  223 469
##
##                Accuracy : 0.6767
##                  95% CI : (0.6501, 0.7025)
##     No Information Rate : 0.5182
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.3511
##  Mcnemar's Test P-Value : 0.06699
##
##             Sensitivity : 0.6332
##             Specificity : 0.7171
##          Pos Pred Value : 0.6754
##          Neg Pred Value : 0.6777
##              Prevalence : 0.4818
##          Detection Rate : 0.3051
##    Detection Prevalence : 0.4517
##       Balanced Accuracy : 0.6752
##
##        'Positive' Class : Yes
##
```

```
roc.curve(knn.model, testSplit2$Class, plotit = T)
```



```
## Area under the curve (AUC): 0.677
```

```
### Applying Random Forest algorithm for 100% fraud data
model.rf <- randomForest(Class ~ ., data =trainSplit2 , ntree = 1000, importance = TRUE)
plot(model.rf)
```



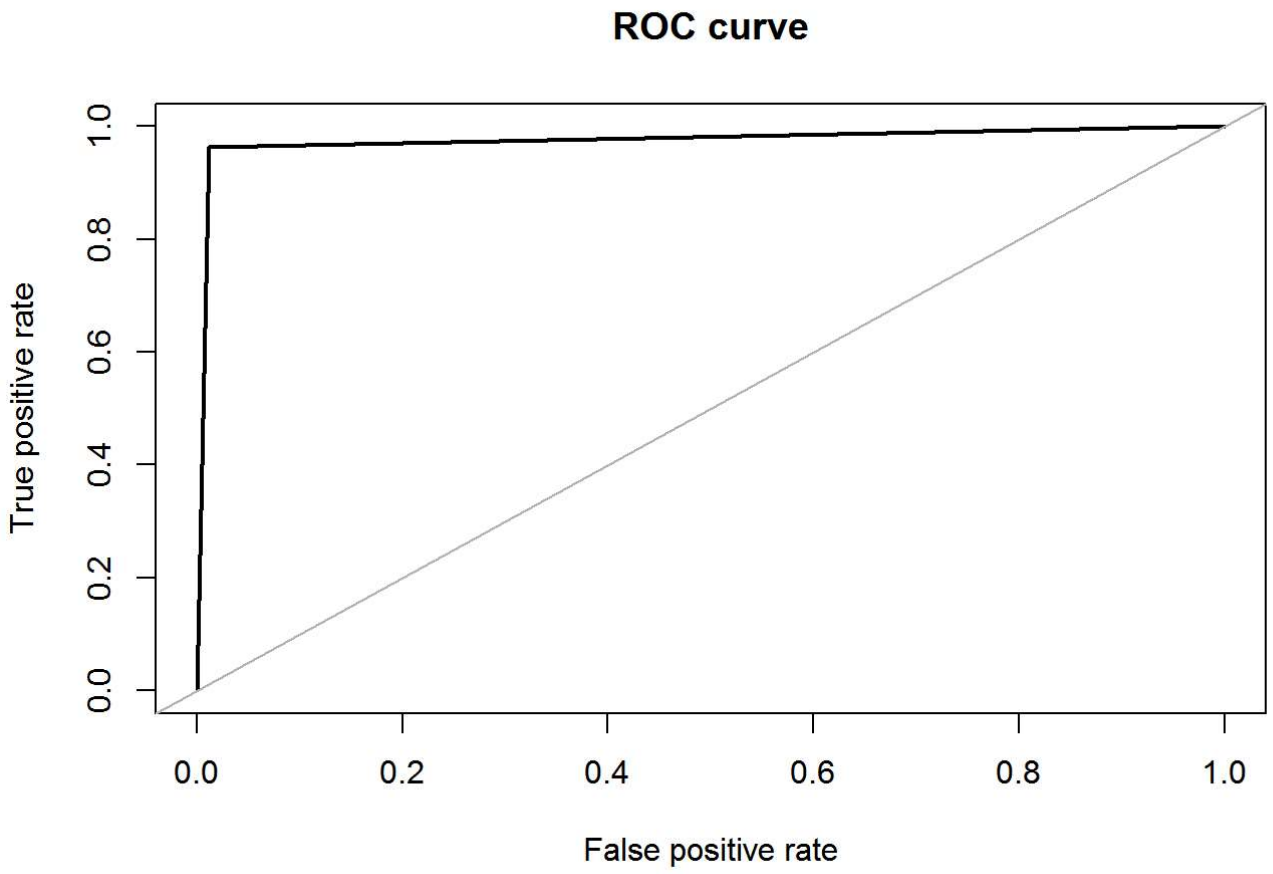

```
cv.tree.pred2 <- predict(model.rf, testSplit2)

# Making table of Confusion matrix
CrossTable(cv.tree.pred2, testSplit2$Class,
           prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
           dnn = c('actual class', 'predicted class'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  1262
##
##
##              | predicted class
## actual class |       Yes |        No | Row Total |
## -------------|-----------|-----------|-----------|
##          Yes |       584 |         7 |       591 |
##              |     0.463 |     0.006 |           |
## -------------|-----------|-----------|-----------|
##           No |        24 |       647 |       671 |
##              |     0.019 |     0.513 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       608 |       654 |      1262 |
## -------------|-----------|-----------|-----------|
##
##
```

```
confusionMatrix(cv.tree.pred2, testSplit2$Class)
```

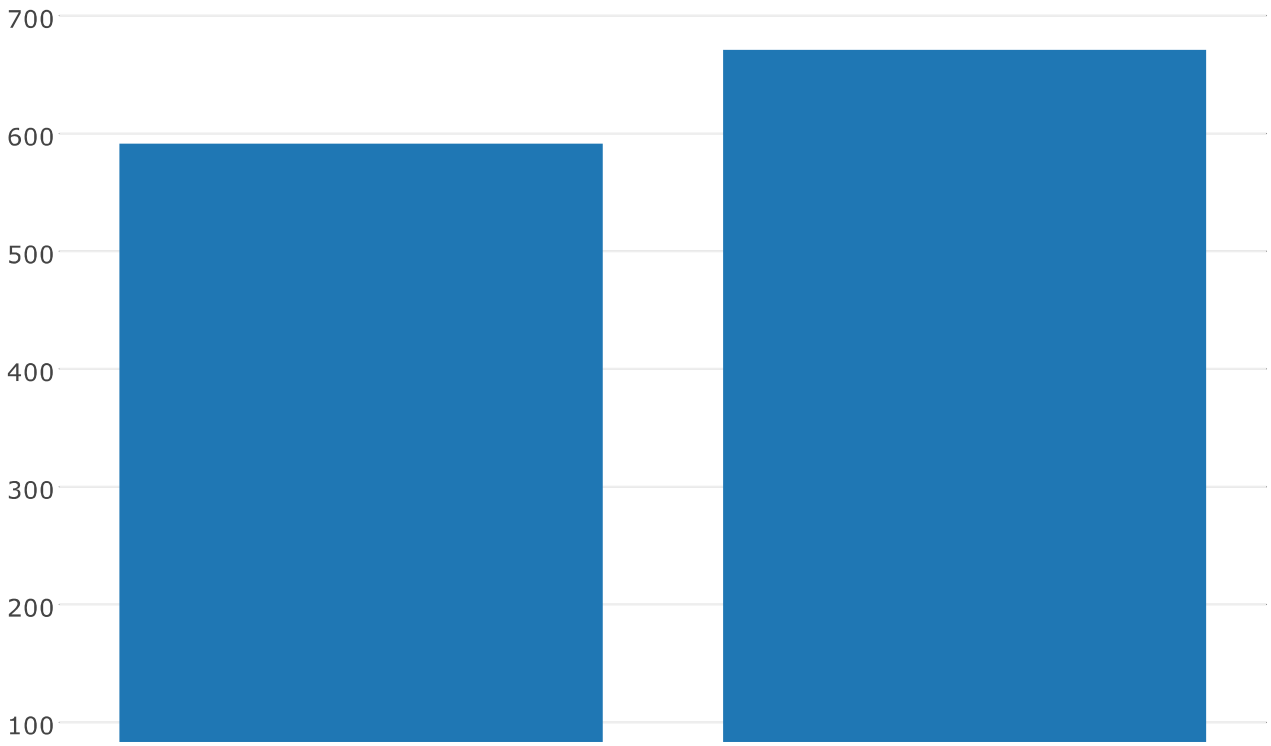
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes 584   7
##        No   24 647
##
##                Accuracy : 0.9754
##                  95% CI : (0.9653, 0.9833)
##     No Information Rate : 0.5182
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9508
##  Mcnemar's Test P-Value : 0.004057
##
##             Sensitivity : 0.9605
##             Specificity : 0.9893
##          Pos Pred Value : 0.9882
##          Neg Pred Value : 0.9642
##              Prevalence : 0.4818
##          Detection Rate : 0.4628
##    Detection Prevalence : 0.4683
##       Balanced Accuracy : 0.9749
##
##        'Positive' Class : Yes
##
```

```
roc.curve(cv.tree.pred2, testSplit2$Class, plotit = T)
```




```
## Area under the curve (AUC): 0.976
```

```
x<-rchisq(1000,5,0)
plot_ly(x=cv.tree.pred2,type = 'histogram')
```



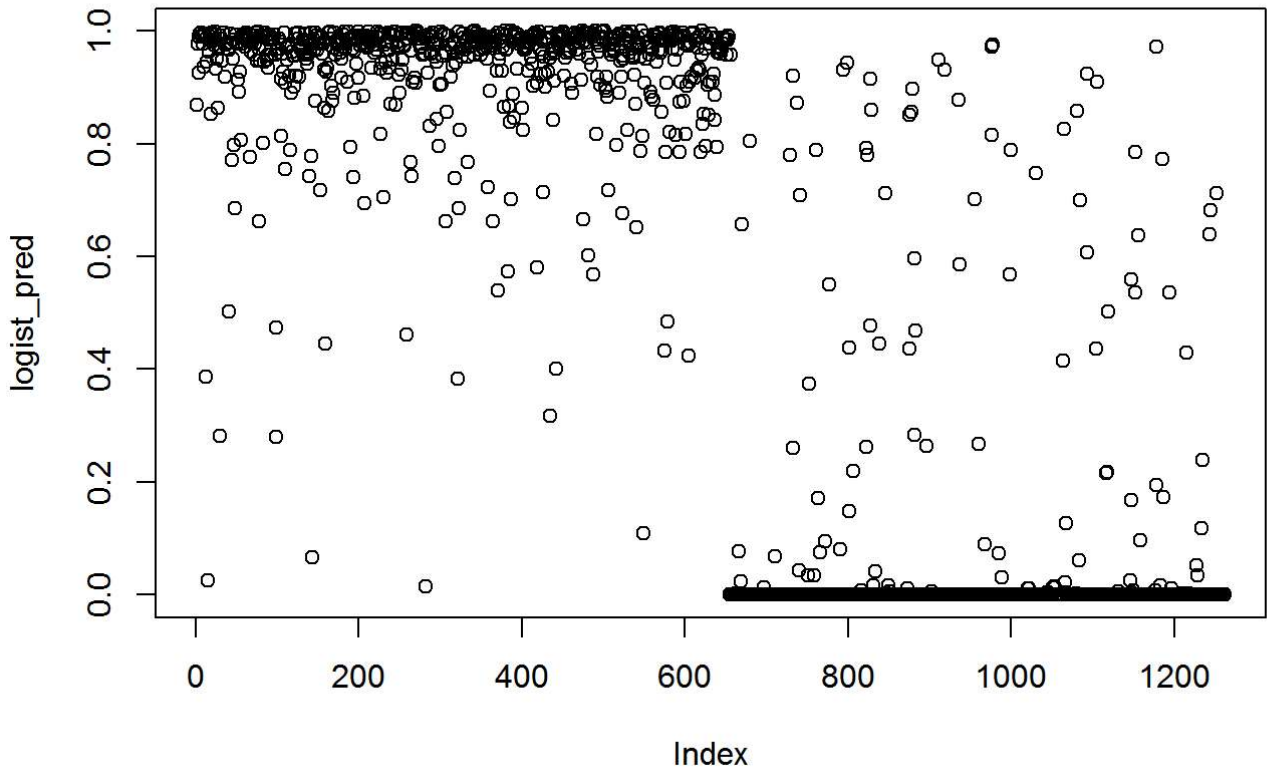

```
### Applying Logistic Regression Algorithm for 100% fraud data
logist <- glm(Class ~ ., data = trainSplit2, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(logist)
```

```
##
## Call:
## glm(formula = Class ~ ., family = "binomial", data = trainSplit2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1263  -0.0002   0.0758   0.2410   6.7890
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.014e+00  3.686e-01   8.179 2.86e-16 ***
## Time         8.538e-06  3.227e-06   2.646 0.008146 **
## V1          -2.193e-01  9.681e-02  -2.265 0.023519 *
## V2          -1.267e-01  2.085e-01  -0.608 0.543189
## V3           5.379e-02  9.909e-02   0.543 0.587224
## V4          -7.667e-01  9.896e-02  -7.748 9.37e-15 ***
## V5          -2.678e-01  1.304e-01  -2.054 0.039987 *
## V6           2.812e-01  1.261e-01   2.230 0.025773 *
## V7          -2.300e-02  1.544e-01  -0.149 0.881616
## V8           4.041e-01  1.043e-01   3.875 0.000107 ***
## V9           1.914e-01  1.434e-01   1.334 0.182217
## V10          3.864e-01  1.943e-01   1.988 0.046770 *
## V11         -3.108e-01  1.145e-01  -2.715 0.006618 **
## V12          8.151e-01  1.401e-01   5.819 5.92e-09 ***
## V13          3.726e-01  9.637e-02   3.866 0.000111 ***
## V14          1.050e+00  1.451e-01   7.237 4.58e-13 ***
## V15          4.615e-02  1.098e-01   0.420 0.674227
## V16          1.932e-01  1.570e-01   1.231 0.218441
## V17          2.124e-01  1.701e-01   1.249 0.211704
## V18          2.920e-02  1.634e-01   0.179 0.858154
## V19         -2.369e-01  1.262e-01  -1.877 0.060470 .
## V20          3.399e-01  2.352e-01   1.445 0.148519
## V21         -9.448e-02  1.187e-01  -0.796 0.426045
## V22         -5.936e-01  1.655e-01  -3.586 0.000336 ***
## V23          1.945e-01  1.629e-01   1.194 0.232509
## V24          1.060e-01  1.993e-01   0.532 0.594905
## V25          1.918e-02  2.377e-01   0.081 0.935698
## V26          7.133e-01  2.558e-01   2.789 0.005295 **
## V27          1.495e-01  3.095e-01   0.483 0.629016
## V28         -2.511e-01  4.531e-01  -0.554 0.579483
## Amount      -2.797e-03  1.819e-03  -1.538 0.124080
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4034.3  on 2919  degrees of freedom
## Residual deviance:  764.1  on 2889  degrees of freedom
## AIC: 826.1
##
## Number of Fisher Scoring iterations: 12
```

```
logist_pred <- predict(logist, newdata=testSplit2,type = "response")
plot(logist_pred)
```

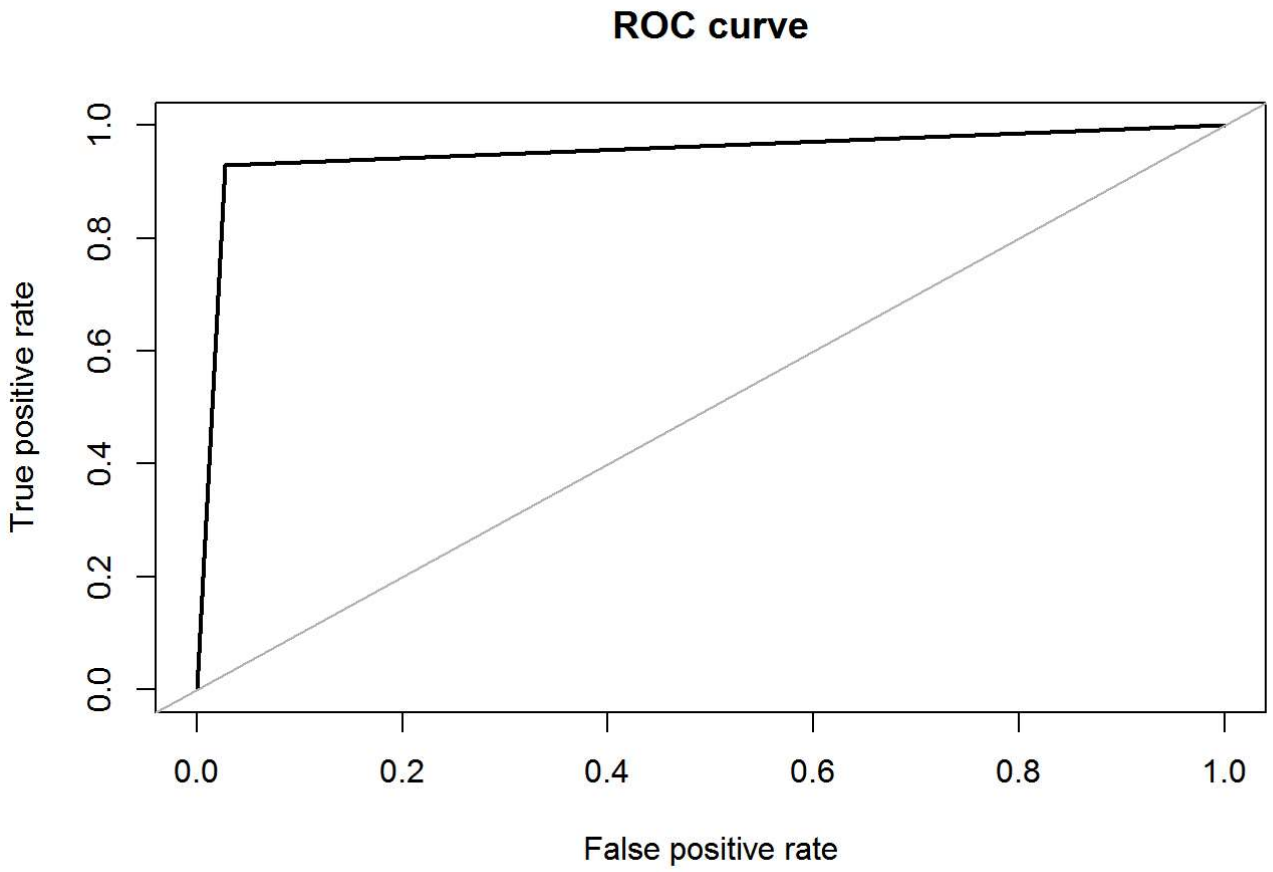



```
pred=rep("Yes",length(logist_pred))
pred[logist_pred > 0.5] = "No"
confusionMatrix(testSplit2$Class, pred)
```

```
## Warning in confusionMatrix.default(testSplit2$Class, pred): Levels are not
## in the same order for reference and data. Refactoring data to match.
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  638  16
##        Yes  48 560
##
##                Accuracy : 0.9493
##                  95% CI : (0.9357, 0.9607)
##     No Information Rate : 0.5436
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8983
##  Mcnemar's Test P-Value : 0.0001066
##
##             Sensitivity : 0.9300
##             Specificity : 0.9722
##          Pos Pred Value : 0.9755
##          Neg Pred Value : 0.9211
##              Prevalence : 0.5436
##          Detection Rate : 0.5055
##    Detection Prevalence : 0.5182
##       Balanced Accuracy : 0.9511
##
##        'Positive' Class : No
##
```

```
roc.curve(pred, testSplit2$Class, plotit = T)
```

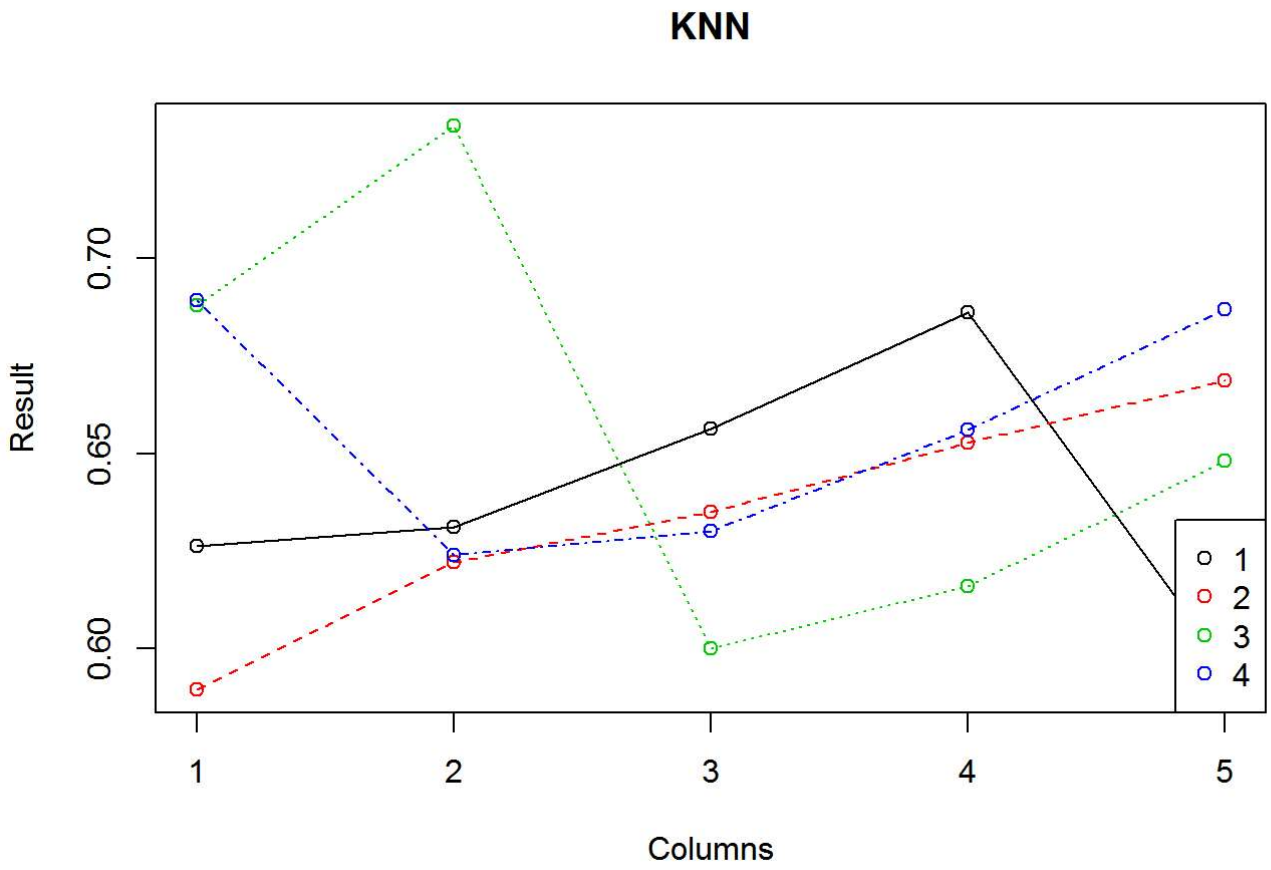


```
## Area under the curve (AUC): 0.951
```

```
#---------------------- RESULTS ----------------------#

#KNN Algorithm results for different fraud percent
mat <-
matrix(c(0.6262,0.5959,0.6527,0.6000,0.6240,0.6311,0.5894,0.6687,0.6159,0.6300,0.6564,0.6222,0.
6879,0.6481,0.6560,0.6862,0.6349,0.7339,0.6893,0.6870),ncol=5,byrow = TRUE)
colnames(mat) <- c(" Accuracy"," Sensitivity"," Specificity"," Precision","  AUC")
rownames(mat) <- c(" 25% Fraud"," 50% Fraud"," 75% Fraud","100% Fraud")
res <- as.table(mat)
res
```

```
##            Accuracy  Sensitivity  Specificity  Precision    AUC
##  25% Fraud    0.6262       0.5959       0.6527     0.6000 0.6240
##  50% Fraud    0.6311       0.5894       0.6687     0.6159 0.6300
##  75% Fraud    0.6564       0.6222       0.6879     0.6481 0.6560
## 100% Fraud    0.6862       0.6349       0.7339     0.6893 0.6870
```

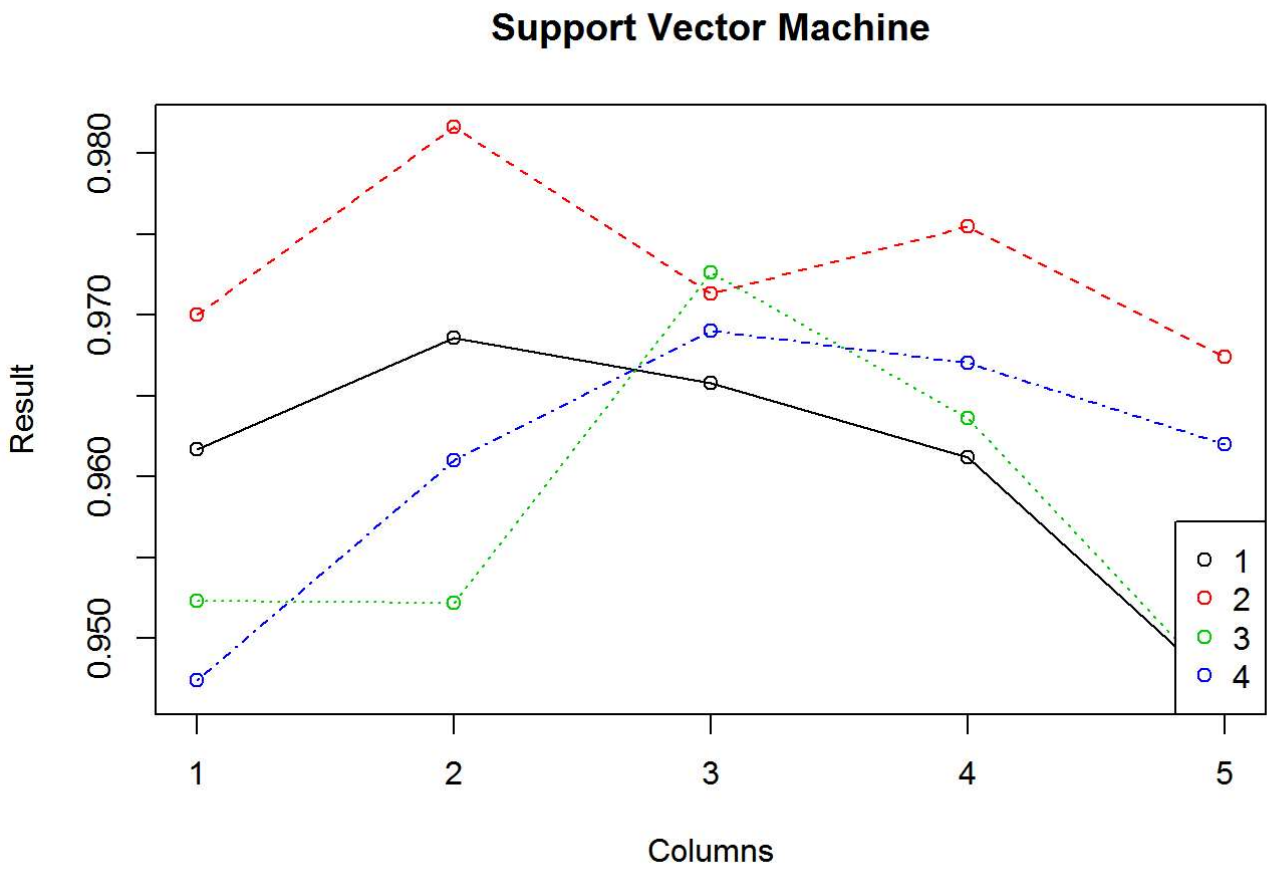
```
dat <- matrix(res,ncol=4) # make data
matplot(dat, type = c("o"),pch=1,col = 1:4,xlab = "Columns", ylab = "Result",
        main = "KNN") #plot
legend("bottomright", legend = 1:4, col=1:4, pch=1)
```



```
#SVM Algorithm results for different fraud percent
mat <-
matrix(c(0.9617,0.9467,0.9755,0.9726,0.9610,0.9686,0.9700,0.9674,0.9636,0.9690,0.9658,0.9816,0.
9523,0.9467,0.9670,0.9612,0.9713,0.9522,0.9474,0.9620),ncol=5,byrow = TRUE)
colnames(mat) <- c(" Accuracy"," Sensitivity"," Specificity"," Precision","  AUC")
rownames(mat) <- c(" 25% Fraud"," 50% Fraud"," 75% Fraud","100% Fraud")
res <- as.table(mat)
res
```

```
##             Accuracy  Sensitivity  Specificity  Precision    AUC
##  25% Fraud    0.9617       0.9467       0.9755     0.9726 0.9610
##  50% Fraud    0.9686       0.9700       0.9674     0.9636 0.9690
##  75% Fraud    0.9658       0.9816       0.9523     0.9467 0.9670
## 100% Fraud    0.9612       0.9713       0.9522     0.9474 0.9620
```

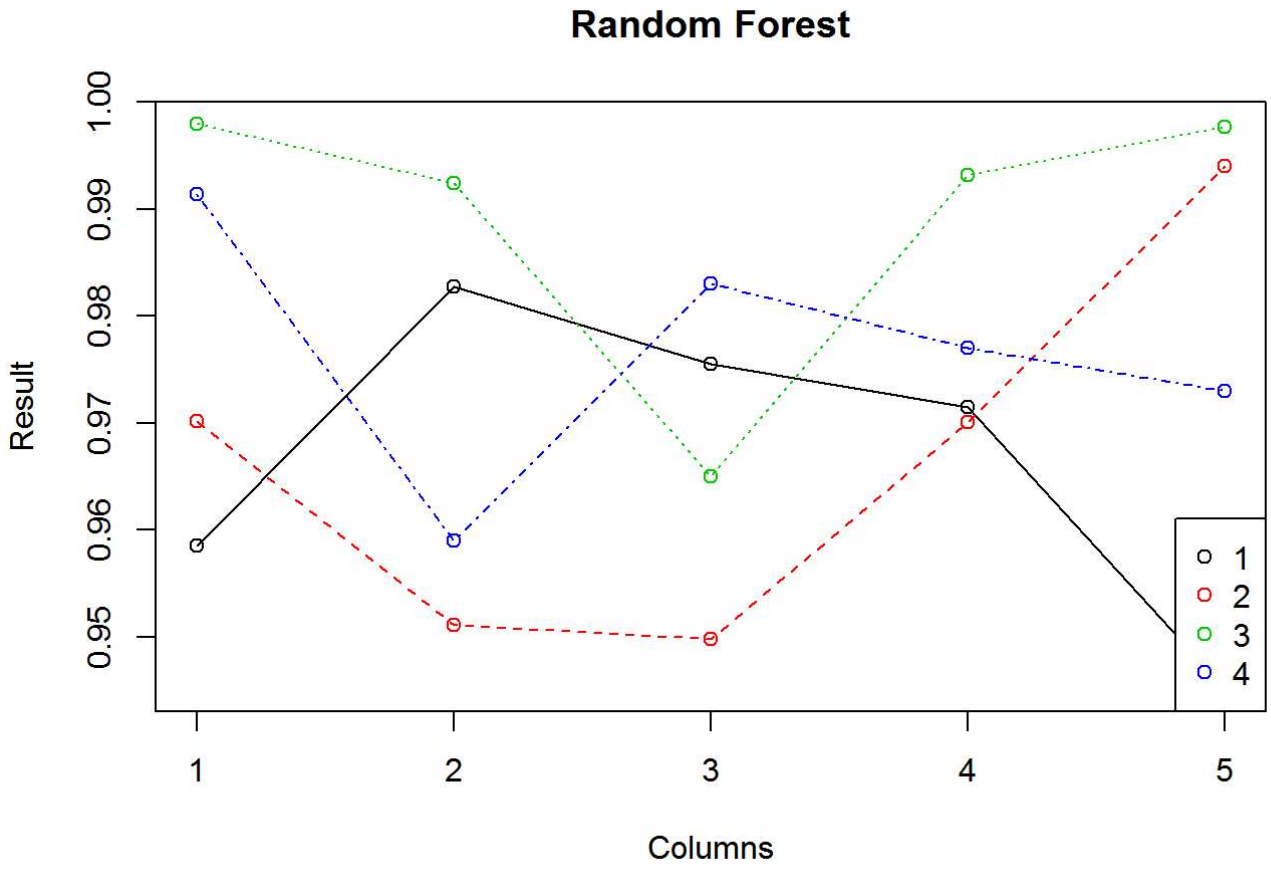
```
dat <- matrix(res,ncol=4) # make data
matplot(dat, type = c("o"),pch=1,col = 1:4,xlab = "Columns", ylab = "Result",
        main = "Support Vector Machine") #plot
legend("bottomright", legend = 1:4, col=1:4, pch=1)
```



```
#Random Forest Algorithm results for different fraud percent
mat <-
matrix(c(0.9585,0.9452,0.9701,0.9650,0.9590,0.9827,0.9702,0.9940,0.9932,0.9830,0.9755,0.9511,0.
9979,0.9977,0.9770,0.9715,0.9498,0.9924,0.9914,0.9730),ncol=5,byrow = TRUE)
colnames(mat) <- c(" Accuracy"," Sensitivity"," Specificity"," Precision","  AUC")
rownames(mat) <- c(" 25% Fraud"," 50% Fraud"," 75% Fraud","100% Fraud")
res <- as.table(mat)
res
```

```
##             Accuracy  Sensitivity  Specificity  Precision    AUC
##  25% Fraud    0.9585       0.9452       0.9701     0.9650 0.9590
##  50% Fraud    0.9827       0.9702       0.9940     0.9932 0.9830
##  75% Fraud    0.9755       0.9511       0.9979     0.9977 0.9770
## 100% Fraud    0.9715       0.9498       0.9924     0.9914 0.9730
```

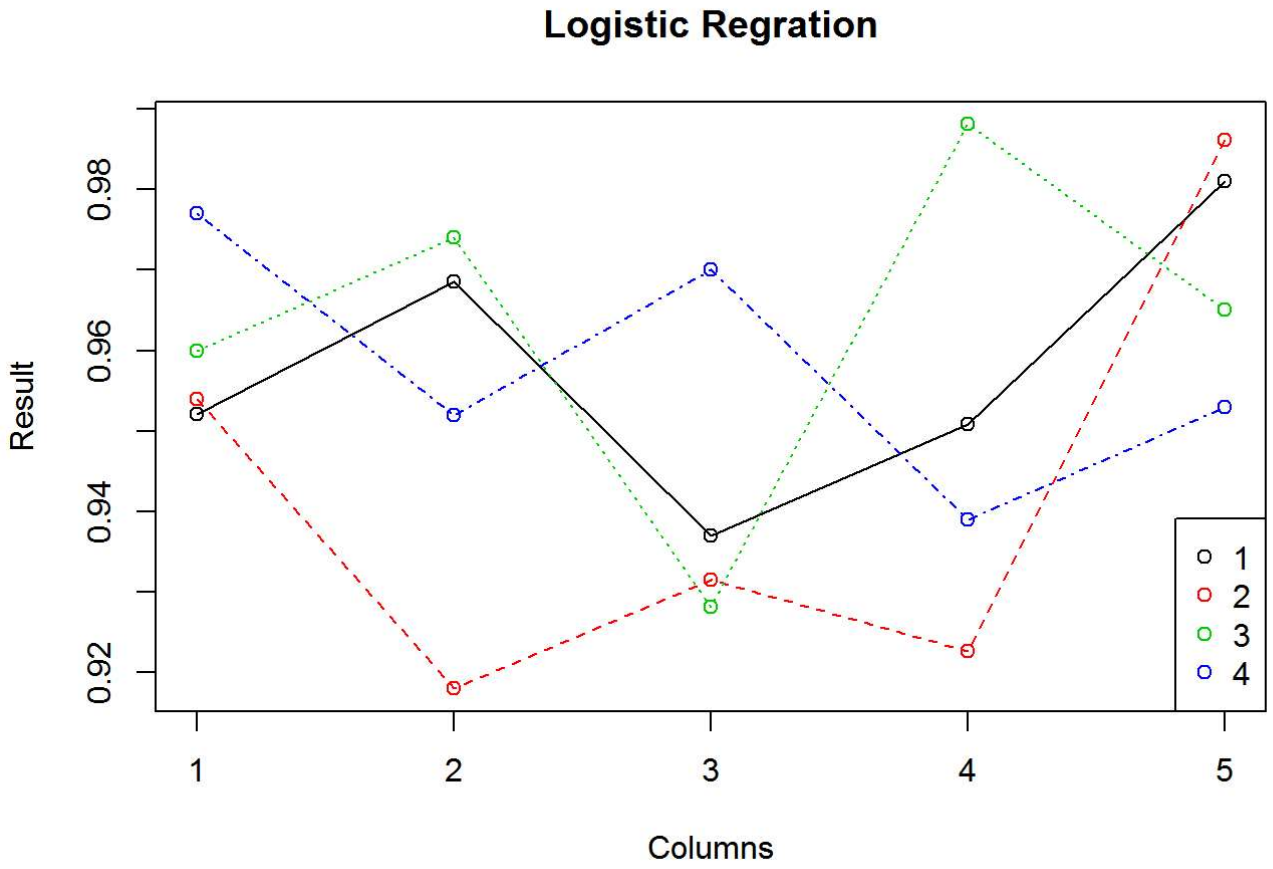
```
dat <- matrix(res,ncol=4) # make data
matplot(dat, type = c("o"),pch=1,col = 1:4,xlab = "Columns", ylab = "Result",
        main = "Random Forest") #plot
legend("bottomright", legend = 1:4, col=1:4, pch=1)
```



```
#Logistic Regression Algorithm results for different fraud percent
mat <-
matrix(c(0.9521,0.9810,0.9226,0.9281,0.9520,0.9686,0.9539,0.9862,0.9881,0.9700,0.9370,0.9180,0.
9600,0.9651,0.9390,0.9509,0.9315,0.9740,0.9771,0.9530),ncol=5,byrow = TRUE)
colnames(mat) <- c(" Accuracy"," Sensitivity"," Specificity"," Precision","  AUC")
rownames(mat) <- c(" 25% Fraud"," 50% Fraud"," 75% Fraud","100% Fraud")
res <- as.table(mat)
res
```

```
##             Accuracy  Sensitivity  Specificity  Precision    AUC
##  25% Fraud    0.9521       0.9810       0.9226     0.9281 0.9520
##  50% Fraud    0.9686       0.9539       0.9862     0.9881 0.9700
##  75% Fraud    0.9370       0.9180       0.9600     0.9651 0.9390
## 100% Fraud    0.9509       0.9315       0.9740     0.9771 0.9530
```

```
dat <- matrix(res,ncol=4) # make data
matplot(dat, type = c("o"),pch=1,col = 1:4,xlab = "Columns", ylab = "Result",
        main = "Logistic Regration") #plot
legend("bottomright", legend = 1:4, col=1:4, pch=1)
```



Chapter 8

Result and Conclusion

Result

As we have applied four different algorithms on the Credit card dataset taken from Kaggle. On Some calculations these Algorithms provides Accuracy, Precision, and Recall values of the Implementation process, Which are tabulated as :

KNN Algorithm

Fraud Rate	Accuracy	Precision	Sensitivity	Specificity	AUC
25%	0.6262	0.6000	0.5959	0.6527	0.6240
50%	0.6311	0.6159	0.5894	0.6687	0.6300
75%	0.6564	0.6481	0.6222	0.6879	0.6560
100%	0.6862	0.6893	0.6349	0.7339	0.6870

SVM Algorithm

Fraud Rate	Accuracy	Precision	Sensitivity	Specificity	AUC
25%	0.9617	0.9726	0.9467	0.9755	0.9610
50%	0.9686	0.9636	0.9700	0.9674	0.9690
75%	0.9658	0.9467	0.9816	0.9523	0.9670
100%	0.9612	0.9474	0.9713	0.9522	0.9620

Random Forest Algorithm

Fraud Rate	Accuracy	Precision	Sensitivity	Specificity	AUC
25%	0.9585	0.9650	0.9452	0.9701	0.9590
50%	0.9827	0.9932	0.9702	0.9940	0.9830
75%	0.9755	0.9977	0.9511	0.9979	0.9770
100%	0.9715	0.9914	0.9498	0.9924	0.9730

Logistic Regression Algorithm

Fraud Rate	Accuracy	Precision	Sensitivity	Specificity	AUC
25%	0.9521	0.9281	0.9810	0.9226	0.9520
50%	0.9686	0.9881	0.9539	0.9862	0.9700
75%	0.9370	0.9651	0.9180	0.9600	0.9390
100%	0.9509	0.9771	0.9315	0.9740	0.9530

Conclusion

Finally, as the results in table shows that highest accuracy in given by the Random forest algorithm when applied all algorithms with SMOTE data balancing technique. These results are formed on taking different fraud data rates in the dataset befor applying to the algorithms.

Chapter 9

Future Scope

- More Algorithms can be applied on the same dataset to get good results.

- Combination of two or more algorithms of data mining are very helpful in determining the best results. This combination of algorithms is known as Hybrid Algorithm.

- Also by making classifiers in present algorithms and applying new upcoming algorithms may be helpful for improvement.

- Choosing only those attributes which most affect the liver if slight change in quantity occurs in body. And applying Different approaches may improve the outcomes.

Chapter 10

References

1. Masoumeh Zareapoor, P.Shamsolmoali{2015}. Application of Credit Card Fraud Detection: Based on Bagging Ensemmble Classifier.International Conference on Intelligent Computing And Convergence(ICCC-2015).
2. S. Kotsiantis, D. Kanellopoulos, P. Pintelas (2006). Handling imbalanced datasets: A review. International Transactions on Computer Science and Engineering.
3. G.H. John, P. Langley (1995). Estimating continuous distributions in Bayesian classifiers. in: Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence, (1995); 338 — 345.
4. I3I R.J. Bolton, D.J. Hand (2001). Unsupervised profiling methods for fraud detection. In Conference on credit scoring and credit control, Edinburgh.
5. D. Kibler. D.W. Aha, M. Albeit (1989). Instance-based prediction of real-valued attributes. Computational Intelligent
6. P.K. Chan, W. Fan, A.L. Prodromidis. S.J. Stolfo (1999). Distributed Data Mining in Credit Card Fraud Detection. IEEE Intelligent Systems. pp 67—74.
7. C. Cortes, V. Vapnik (1995). Support vector networks. Machine Learning. 20:273—297.
8. T.M. Cover, P.E. Hart (1967). Nearest neighbor pattern classification. IEEE Trans. Information Theory, l3(1):21—27.
9. G. Potamilis (2013). Design and Implementation of a Fraud Detection Expert System using Ontology-Based Techniques. A dissertation submitted to the University of Manchester for the degree of Master of Science in the Faculty of Engineering and Physical Sciences.
10. E. David (2012). Bayesian inference-the future of online fraud protection. Computer Fraud & Security, 8-11.
11. S. Ghosh, D.L. Reilly. (1994). Credit Card Fraud Detection with a Neural-Network. In Proceedings of the International Conference on System Science, pages 621-630.
12. Jha. Sanjeev, G. Montserrat, J.C. Westland (2012). Employing transaction aggregation strategy to detect credit card fraud. Expert system with application, 39: 12650-12657.
13. L. Breiman - Random forests. Machine Learning. (2001). Vol (45); 5—32.
14. J. Piotr., AM. Niall. J.D. Hand, C. Whitrow, J. David (2008). Off the peg and bespoke classifiers for fraud detection. Computationa Statistics and Data Analysis, 52

15. L. Qibei & J. Chunhua (2011). Research on Credit Card Fraud Detection Model Based on Class Weighted Support Vector Machine. Journal of Convergence Information Technology, 6(1). 62-68.
16. S. Maes, K. Tuyls, B.Vanschoenwinkel, B.Manderick (1993). Credit card fraud detection using Bayesian and neural networks. In Proceedings for the First International NAISO Congress on Nettro Fuzzy Technologies, pages 261-270.
17. E.W.T. Ngai, H.Yong., Y.H.Wong, Y.Chen, X. Sun (2011). The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. Decision Support Systems, 50:559—569
18. M. Zareapoor, Seeja, K.R. & M. Alam, Afshar (2012). Analyzing Credit Card Fraud Detection Techniques Based On Certain Design Criteria. International Journal of Computer Application. 52(3):35-42.
19. E.Aleskerov,B.Freisleben, B.Rao, CARDWATCH: a neural network based database mining system for credit card fraud detection, in computational intelligence for financial engineering, Proceedings of the IEEE/IAFE, IEEE, Piscataway, NJ, 1998, pp. 220–226.
20. M. Artis, M. Ayuso, M. Guillen(2002), Detection of automobile insurance fraud with discrete choice models and misclassified claims, The Journal of Risk and Insurance 69 (3}325–340.
21. R.J. Bolton, D.J. Hand{2001}, Unsupervised profiling methods for fraud detection, Conference on Credit Scoring and Credit Control, Edinburgh.
22. R.J. Bolton, D.J. Hand (2002), Statistical fraud detection: a review, Statistical Science 17 (3) 235–249.
23. R. Brause, T. Langsdorf, M. Hepp {1999}, Neural data mining for credit card fraud detection, Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence.
24. C.R. Bollinger, M.H. David (1997), Modeling discrete choice with response error: food stamp participation, Journal of the American Statistical Association .
25. R. Caruana, N. Karampatziakis, A. Yessenalina{2008}, An Empirical Evaluation of Supervised Learning in High Dimensions, in: Proceedings of the 25th international Conference on Machine Learning Helsinki, Finland, July,.