

1. INTRODUCTION

1.1 PROBLEM STATEMENT

Manual detection of credit card fraud is a tough job for banks and credit card merchants. Classification of transactions into fraud and genuine can be done with less effort by using data mining techniques. But selection of an efficient data mining algorithm needs a good understanding of algorithms and also what accuracy can be attained by using them.

Our goal is to implement Logistic Regression, Decision Trees, Random Forest, Naive Bayes and Neural Network algorithms and make a comparative analysis of these algorithms to determine which can be used by credit card companies.

1.2 MOTIVATION

Banks incur huge losses every year due to fraud transactions. While the exact amount of losses due to fraudulent activities on cards is unknown, various research analyst reports concur that the figure for year 2019 probably exceeds \$6.5 billion. Further, as the overall e-commerce volumes continue to grow the projected figure for losses to internet merchants in the US alone is expected to be in the range of \$5–15 billion by the year 2030.

1.3 CONTRIBUTION

This project helps banks and credit card merchants in deciding the efficient algorithm to be implemented for reducing their loss due to fraudulent transactions. We have used a data-set containing 2.8 million records of credit card transactions and implemented the fore-mentioned algorithms, concluding with the performance metrics of each algorithm.

1.4 RESEARCH METHODOLOGY

Credit card fraud can be defined as “Unauthorized account activity by a person for which the account was not intended. Operationally, this is an event for which action can be taken to stop the abuse in progress and incorporate risk management practices to protect against similar actions in the future”. In simple terms, Credit Card Fraud is defined as when an individual uses another individual’s credit card for personal reasons while the owner of the card and the card issuer are not aware of the fact that the card is being used.

Gathering data: Kaggle is a platform for predictive modelling and analytics competitions in which statisticians and data miners compete to produce the best models for predicting and describing the data sets uploaded by companies and users.

We have used a data set of credit card transactions from Kaggle.

Contents in the data set :

The data set contains a total of 2,84,808 credit card transactions of a croatian bank data set. It considers fraud transactions as the “positive class” and genuine ones as the “negative class” .

The data set is highly imbalanced, it has about 0.172% of fraud transactions and the rest are genuine transactions.

So we have done oversampling to balance the data set, which resulted in 60% of fraud transactions and 40% genuine ones.

❖ What is Imbalanced Classification ?

Imbalanced classification is a supervised learning problem where one class outnumbers other class by a large proportion. This problem is faced more frequently in binary classification problems than multi-level classification problems.

To overcome this problem we use Sampling techniques like Under Sampling, Over Sampling, etc. We have used Over Sampling technique to balance our dataset.

Oversampling

This method works with minority class. It replicates the observations from minority class to balance the data. It is also known as 'Up-sampling'.

2. LITERATURE SURVEY

2.1 Introduction to Credit Card Fraud

Credit Card Fraud is one of the biggest threats to business establishments today. However, to combat the fraud effectively, it is important to first understand the mechanisms of executing a fraud. Credit card fraudsters employ a large number of ways to commit fraud. In simple terms, Credit Card Fraud is defined as “when an individual uses another individuals’ credit card for personal reasons while the owner of the card and the card issuer are not aware of the fact that the card is being used”. Further, the individual using the card has no connection with the cardholder or issuer, and has no intention of either contacting the owner of the card or making repayments for the purchases made. Credit card frauds are committed in the following ways:

- An act of criminal deception (mislead with intent) by use of unauthorized account and/or personal information.
- Illegal or unauthorized use of account for personal gain
- Misrepresentation of account information to obtain goods and/or services.

2.2 Recent Developments in Fraud Detection

● Simple rule systems

Simple rule systems involve the creation of ‘if...then’ criteria to filter incoming authorizations/transactions. Rule-based systems rely on a set of expert rules designed to identify specific types of high-risk transactions.

● **Risk scoring technologies**

Risk scoring tools are based on statistical models designed to recognize fraudulent transactions, based on a number of indicators derived from the transaction characteristics. Typically, these tools generate a numeric score indicating the likelihood of a transaction being fraudulent: the higher the score, the more suspicious the order. Risk scoring systems provide one of the most effective fraud prevention tools available.

● **Neural network technologies**

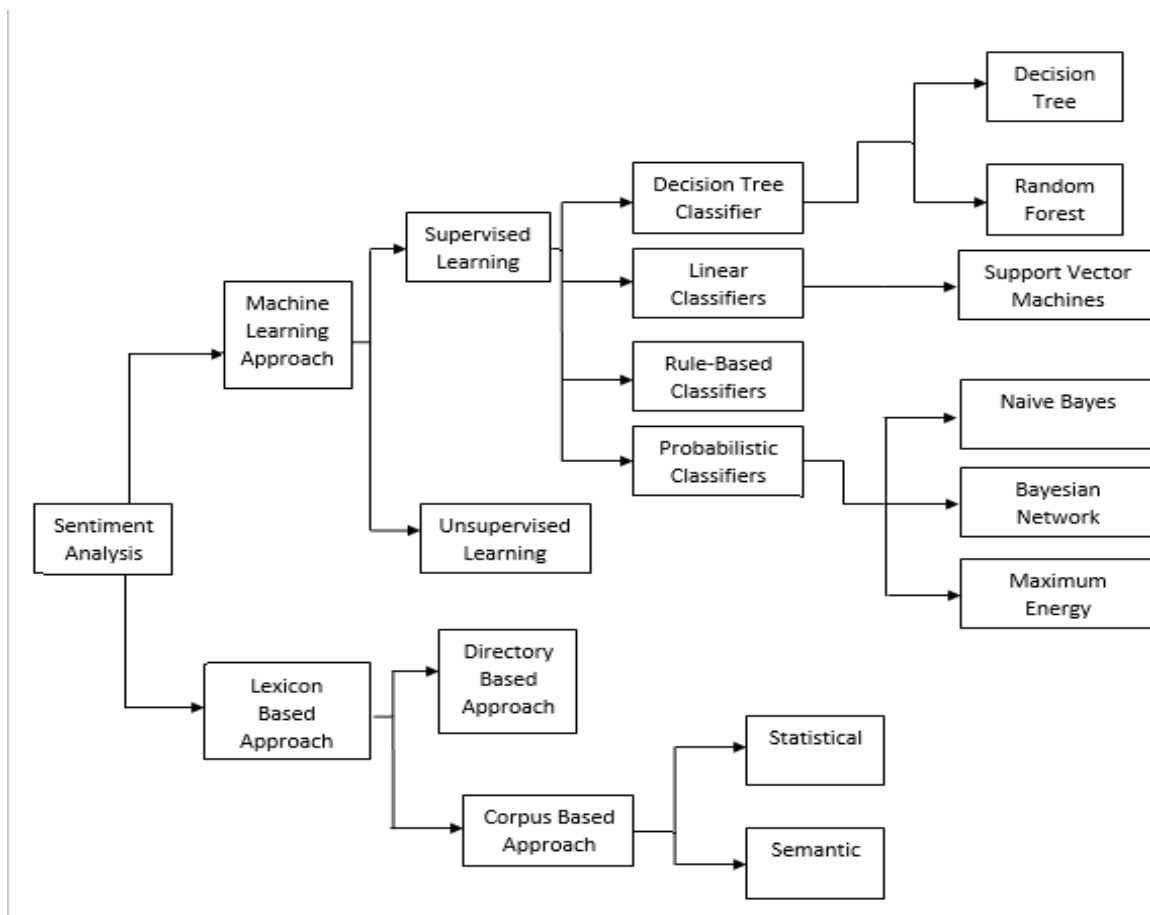
Neural networks are an extension of risk scoring techniques. They are based on the ‘statistical knowledge’ contained in extensive databases of historical transactions, and fraudulent ones in particular. These neural network models are basically ‘trained’ by using examples of both legitimate and fraudulent transactions and are able to correlate and weigh various fraud indicators (e.g., unusual transaction amount, card history, etc) to the occurrence of fraud. A neural network is a computerized system that sorts data logically by performing the following tasks:

- Identifies cardholder’s buying and fraudulent activity patterns.
- Processes data by trial and elimination (excluding data that is not relevant to the pattern).
- Finds relationships in the patterns and current transaction data.

2.3 MACHINE LEARNING

Two definitions of Machine Learning are offered. Arthur Samuel described it as: "the field of study that gives computers the ability to learn without being explicitly programmed." This is an older, informal definition.

Tom Mitchell provides a more modern definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ."



In general, any machine learning problem can be assigned to one of two broad classifications:

- a. Supervised learning
- b. Unsupervised learning.

SUPERVISED LEARNING

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories

UNSUPERVISED LEARNING

Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables. We can derive this structure by clustering the data based on relationships among the variables in the data. With unsupervised learning there is no feedback based on the prediction results.

CLASSIFICATION

Classification is a data mining function that assigns items in a collection to target categories or classes. The goal of classification is to accurately predict the target class for each case in the data. For example, a classification model could be used to identify loan applicants as low, medium, or high credit risks.

A classification task begins with a data set in which the class assignments are known. For example, a classification model that predicts credit risk could be developed based on observed data for many loan applicants over a period of time. In addition to the historical credit rating, the data might track employment history, home ownership or rental, years of residence, number and type of investments, and so on. Credit rating would be the target, the other attributes would be the predictors, and the data for each customer would constitute a case.

Classifications are discrete and do not imply order. Continuous, floating-point values would indicate a numerical, rather than a categorical, target. A predictive model with a numerical target uses a regression algorithm, not a classification algorithm.

The simplest type of classification problem is binary classification. In binary classification, the target attribute has only two possible values: for example, high credit rating or low credit rating. Multi class targets have more than two values: for example, low, medium, high, or unknown credit rating.

In the model build (training) process, a classification algorithm finds relationships between the values of the predictors and the values of the target. Different classification algorithms use different techniques for finding relationships. These relationships are summarized in a model, which can then be applied to a different data set in which the class assignments are unknown.

Classification models are tested by comparing the predicted values to known target values in a set of test data. The historical data for a classification project is typically divided into two data sets: one for building the model; the other for testing the model.

Scoring a classification model results in class assignments and probabilities for each case. For example, a model that classifies customers as low, medium, or high value would also predict the probability of each classification for each customer.

2.4 ALGORITHMS IMPLEMENTED

- Naive Bayes
- Decision Trees
- Neural Networks
- Random Forest
- Logistic Regression

2.5 LOGISTIC REGRESSION

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. To represent binary / categorical outcome, we use dummy variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as dependent variable. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function.

LOGISTIC REGRESSION WORKING

$$y = e^{(b_0 + b_1 * x)} / (1 + e^{(b_0 + b_1 * x)})$$

Where y is the predicted output, b_0 is the bias or intercept term and b_1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. To represent binary / categorical outcome, we use dummy variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as dependent variable. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Logistic Regression is part of a larger class of algorithms known as Generalized Linear Model (glm). In 1972, Nelder and Wedderburn proposed this model with an effort to provide a means of using linear regression to the problems which were not directly suited for application of linear regression.

To start with logistic regression, I'll first write the simple linear regression equation with dependent variable enclosed in a link function:

$$g(y) = \beta_0 + \beta(x)$$

where $g()$ is the link function y is the outcome variable x is the dependent variable This function is established using two things: Probability of Success(p) and Probability of Failure($1-p$). p should meet following criteria: 1. It must always be positive (since $p \geq 0$) 2. It must always be less than equals to 1 (since $p \leq 1$) So based on the first criteria by applying exp, it will make sure the value is always ≥ 0

$$p = \exp(\beta_0 + \beta(x)) = e^{(\beta_0 + \beta(x))}$$

For the second criteria we divide the same exp by adding 1 to it so the value will be ≤ 1

$$p = e^{(\beta_0 + \beta(x))} / e^{(\beta_0 + \beta(x))} + 1$$

This is the logit function used in Logistic Regression. The cost function quantifies the error, as it compares the model's response with the correct answer. $J(\theta) = -1/m * (\sum y_i \log(h\theta(x_i)) + (1-y_i) \log(1-h\theta(x_i)))$ where $h\theta(x_i)$ is the logit function

y_i is the outcome variable Gradient descent is a learning algorithm, that given the value of the cost function finds new values for the parameters that result in a smaller error. Descent comes from minimizing the error. The gradient part refers to how the algorithm updates the model's parameters at each training step. The update is based on the gradient of the cost function w.r.t to each parameter. The gradient is the first partial derivative of the loss function w.r.t. to a parameter.

2.6 NAIVE BAYES

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods.

A very simple rule for classifying a record into one of m classes, ignoring all predictor information ($X_1; X_2; \dots; X_p$) that we may have, is to classify the record as a member of the majority class. **NAIVE BAYES** The Naive Bayes classifier is a more sophisticated method than the naive rule. The main idea is to integrate the information given in a set of predictors into the naive rule to obtain more accurate classifications. In other words, the probability of a record belonging to a certain class is now evaluated not only based on the prevalence of that class but also on the additional information that is given on that record in terms of its X information. In contrast to the other classifiers discussed here, Naive Bayes works only with predictors that are categorical. Numerical predictors must be binned and converted to categorical variables before the Naive Bayes classifier can use them. **BAYES THEOREM** The naive Bayes method is based on conditional probabilities, and in

particular on a fundamental theorem in probability theory, called Bayes Theorem. This clever theorem provides the probability of a prior event, given that a certain subsequent event has occurred. A conditional probability of event A given event B (denoted by $P(A/B)$) represents the chances of event A occurring only under the scenario that event B occurs.

$$p(c/x)=(p(x/c)*p(c))/p(x)$$

BAYES_THEOREM

It works on conditional probability. Conditional probability is the probability that something will happen, given that something else has already occurred. Using the conditional probability, we can calculate the probability of an event using its prior knowledge.

Below is the formula for calculating the conditional probability.

$$P(H|E)=P(E|H)*P(H)/P(E)$$

Where

- $P(H)$ is the probability of hypothesis H being true. This is known as the prior probability.
- $P(E)$ is the probability of the evidence(regardless of the hypothesis).
- $P(E|H)$ is the probability of the evidence given that hypothesis is true.
- $P(H|E)$ is the probability of the hypothesis given that the evidence is there.

NAÏVE BAYES CLASSIFIER

Naive Bayes is a kind of classifier which uses the Bayes Theorem. The class with the highest probability is considered as the most likely class. This is also known as **Maximum A Posteriori (MAP)**.

$$\begin{aligned} MAP(H) \\ &= \max(P(H|E)) \\ &= \max((P(E|H) * P(H)) / P(E)) \\ &= \max(P(E|H) * P(H)) \end{aligned}$$

$P(E)$ is evidence probability, and it is used to normalize the result. It remains same so, removing it won't affect. Naive Bayes classifier assumes that all the features are unrelated to each other. Presence or absence of a feature does not influence the presence or absence of any other feature.

TYPES OF NAÏVE BAYES

1. Gaussian Naive Bayes

When attribute values are continuous, an assumption is made that the values associated with each class are distributed according to Gaussian i.e., Normal Distribution. If in our data, an attribute say “x” contains continuous data. We first segment the data by the class and then compute mean μ_y each class.

2. Multinomial Naive Bayes

Multinomial Naive Bayes is preferred to use on data that is multinomially distributed. It is one of the standard classic algorithms. Which is used in text categorization (classification). Each event in text classification represents the occurrence of a word in a document.

3. Bernoulli Naive Bayes

Bernoulli Naive Bayes is used on the data that is distributed according to multivariate Bernoulli distributions i.e., multiple features can be there, but each

one is assumed to be a binary-valued (Bernoulli, Boolean) variable. So, it requires features to be binary valued.

2.7 DECISION TREE ALGORITHM

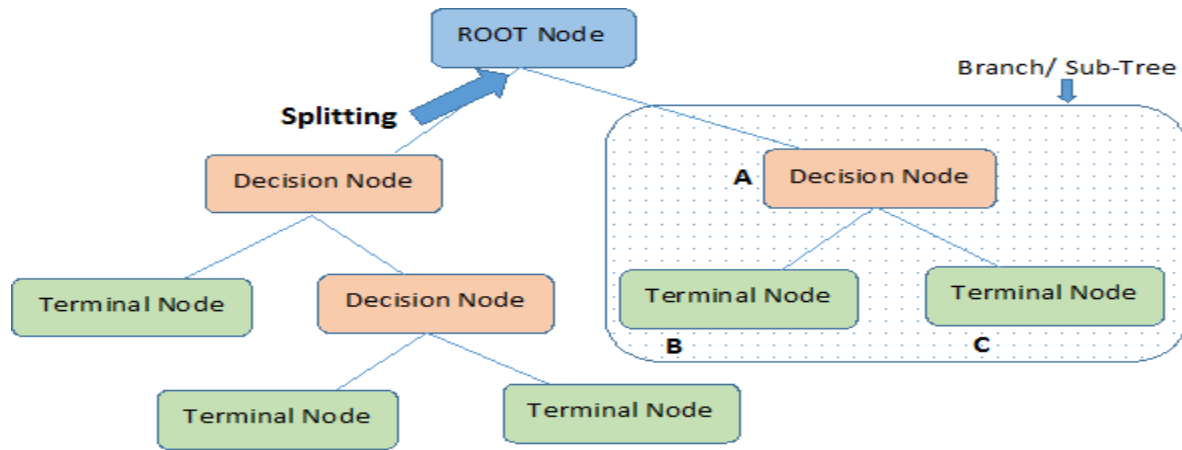
Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

TYPES OF DECISION TREE

1. **Categorical Variable Decision Tree:** Decision Tree which has categorical target variable then it called as categorical variable decision tree.
2. **Continuous Variable Decision Tree:** Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree

TERMINOLOGY OF DECISION TREE :

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
6. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
7. **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.



Note:- A is parent node of B and C.

WORKING OF DECISION TREE

Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that purity of the node increases with respect to the target variable. Decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

1. **Gini Index**
2. **Information Gain**
3. **Chi Square**
4. **Reduction of Variance**

2.8 NEURAL NETWORKS:

Neural Networks (NN), also called as Artificial Neural Network is named after its artificial representation of working of a human being's *nervous system*.

Neural Network is divided into layer of 3 types:

Input Layer: The training observations are fed through these neurons

Hidden Layers: These are the intermediate layers between input and output which help the Neural Network learn the complicated relationships involved in data.

Output Layer: The final output is extracted from previous two layers. For Example: In case of a classification problem with 5 classes, the output later will have 5 neurons.

The different components are:

x1, x2,..., xN: Inputs to the neuron. These can either be the actual observations from input layer or an intermediate value from one of the hidden layers.

x0: Bias unit. This is a constant value added to the input of the activation function. It works similar to an intercept term and typically has +1 value.

w0,w1, w2,...,wN: Weights on each input. Note that even bias unit has a weight.

a: Output of the neuron which is calculated as:

$$a = f\left(\sum_{i=0}^N w_i x_i\right)$$

2.9 RANDOM FOREST

Random forest is a tree-based algorithm which involves building several trees (decision trees), then combining their output to improve generalization ability of the model. The method of combining trees is known as an ensemble method. Ensembling is nothing but a combination of weak learners (individual trees) to produce a strong learner. Random Forest can be used to solve regression and

classification problems. In regression problems, the dependent variable is continuous. In classification problems, the dependent variable is categorical.

WORKING OF RANDOM FOREST

First, it uses the Bagging (Bootstrap Aggregating) algorithm to create random samples. Given a data set D1 (n rows and p columns), it creates a new data set (D2) by sampling n cases at random with replacement from the original data. About 1/3 of the rows from D1 are left out, known as Out of Bag (OOB) samples.

Then, the model trains on D2. OOB sample is used to determine unbiased estimate of the error. Out of p columns, $P \ll p$ columns are selected at each node in the data set. The P columns are selected at random. Usually, the default choice of P is $p/3$ for regression tree and P is \sqrt{p} for classification tree. Unlike a tree, no pruning takes place in random forest i.e, each tree is grown fully. In decision trees, pruning is a method to avoid overfitting. Pruning means selecting a subtree that leads to the lowest test error rate. We can use cross validation to determine the test error rate of a subtree. Several trees are grown and the final prediction is obtained by averaging or voting.

BAGGING VS RANDOM FOREST

It creates randomized samples of the data set (just like random forest) and grows trees on a different sample of the original data. The remaining 1/3 of the sample is used to estimate unbiased OOB error. It considers all the features at a node (for splitting). Once the trees are fully grown, it uses averaging or voting to combine the resultant predictions.

The main difference random forest and bagging is that random forest considers only a subset of predictors at a split. This results in trees with different predictors at top split, thereby resulting in decorrelated trees and more reliable average output.

3. SYSTEM REQUIREMENT SPECIFICATION

3.1 SOFTWARES USED IN THIS PROJECT:

Programming Language:	R
Operating System:	Windows, Linux, Unix, OSX
Front End:	R, HTML
Tool:	R-Studio

3.2 HARDWARE SPECIFICATION

Processor:	Intel Multi Core processor
RAM:	512 MB or above
Hard disk:	100 GB or above

3.2.1 USER INTERFACE:

The work of the user is to choose an algorithm

3.2.2 HARDWARE INTERFACE:

The performance metrics of chosen algorithm are displayed on the monitor screen.

3.2.3 SOFTWARE INTERFACE:

R is a programming language which supports machine learning algorithms for predicting accuracy and obtaining accuracy of algorithms

4. SOFTWARE METHODOLOGIES

4.1 INTRODUCTION TO R LANGUAGE

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, etc) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

4.1.1. R Environment

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. It includes

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,

- graphical facilities for data analysis and display either on-screen or on hard copy, and
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

R is designed around a true computer language, and it allows users to add additional functionality by defining new functions. Much of the system is itself written in the R dialect of S, 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 code to manipulate R objects directly.

Many users think of R as a statistics system. We prefer to think of it of an environment within which statistical techniques are implemented. R can be extended (easily) via packages. There are about eight packages supplied with the R distribution and many more are available through the CRAN family of Internet sites covering a very wide range of modern statistics.

4.1.2. CRAN Packages

The capabilities of R are extended through user-created *packages*, which allow specialized statistical techniques, graphical devices, import/export capabilities, reporting tools, etc. These packages are developed primarily in R, and sometimes in Java, C, C++, and Fortran. The R packaging system is also used by researchers to create compendia to organise research data, code and report files in a systematic way for sharing and public archiving.

A core set of packages is included with the installation of R, with more than 11,000 additional packages (as of July 2017) available at the Comprehensive R Archive Network (CRAN), Bioconductor, Omegahat, GitHub, and other repositories.

The "Task Views" page (subject list) on the CRAN website 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.

Other R package resources include Crantastic, a community site for rating and reviewing all CRAN packages, and R-Forge, a central platform for the collaborative development of R packages, R-related software, and projects. R-Forge also hosts many unpublished beta packages, and development versions of CRAN packages.

4.1.3. Advantages Of R

The R language has diversified application in the software development companies such as in gaming, web frameworks and applications, language development, prototyping, graphic design applications, etc. This provides the language a higher plethora over other programming languages used in the industry.

- **Extensive Support Libraries**

It provides large standard libraries that include the areas like string operations, Internet, web service tools, operating system interfaces and protocols. Most of the highly used programming tasks are already scripted into it that limits the length of the codes to be written in R.

- **Improved Programmer's Productivity**

The language has extensive support libraries and clean object-oriented designs that increase two to ten fold of programmer's productivity while using the languages like Java, VB, Perl, C, C++ and C#.

- **Productivity**

With its strong process integration features, unit testing framework and enhanced control capabilities contribute towards the increased speed for most applications and productivity of applications. It is a great option for building scalable multi-protocol network applications.

- **Data handling Capabilities**

Good data handling capabilities and options for parallel computation.

4.2 R-Studio

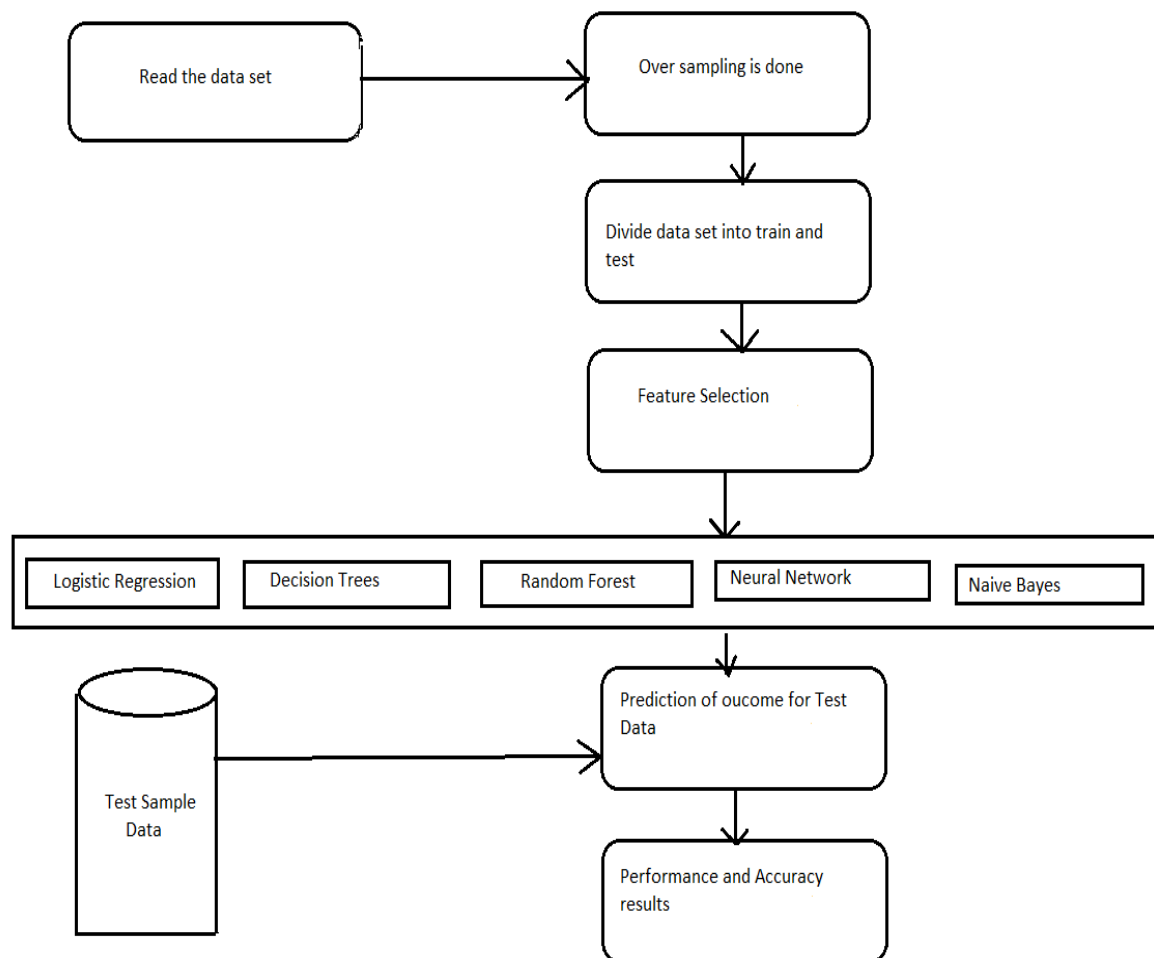
R-Studio is a free and open-source integrated development environment for R, a programming language for statistical computing and graphics. R-Studio was founded by JJ Allaire, creator of the programming language ColdFusion. It is available in two editions: R-Studio Desktop, where the program is run locally as a regular desktop application; and R-Studio Server, which allows accessing R-Studio using a web browser while it is running on a remote Linux server. Prepackaged distributions of R-Studio Desktop are available for Windows, macOS, and Linux.

R-Studio is available in open source and commercial editions and runs on the desktop or in a browser connected to R-Studio Server. It is written in the C++ programming language and uses the Qt framework for its graphical user interface.

5.PROPOSED SYSTEM

In the proposed system, we are trying to make a comparison among the machine learning algorithms Logistic Regression, Decision Trees, Random Forest, Naive Bayes and K-Nearest Neighbour, to determine which algorithm gives suits best and can be adapted by credit card merchants for identifying fraud transactions.

5.1 ARCHITECTURE



5.2 PROCESSING STEPS

1. Random Sampling is done on the data set to make it balanced.
2. Data set is divided into Train and Test data sets.
3. Required features are selected.
4. The train data set is given to our model for training.
5. Prediction is done using this model, on the test data set.
6. Accuracy and other performance metrics are calculated.
7. Receiver Operating Characteristic curve is plotted and results are viewed.

6. SYSTEM DESIGN

6.1 UML DIAGRAMS

Unified Modelling Language(UML) provides vocabulary and the results for combining words in that vocabulary for the purpose of communication.

A modelling language is language whose vocabulary and rules flows on the conceptual and physical representation of a system. A modelling language such as uml is a standard language for software blue prints.

Unified Modelling Language is used for visualizing, specifying, constructing and documenting. The software intensive artifacts of a system.

UML diagram are classified into two categories:

1. Structural or static
2. Dynamic or behavioural

Structural Model contains

Classes, object, use case, component and deployment.

Behavioural Model contains:

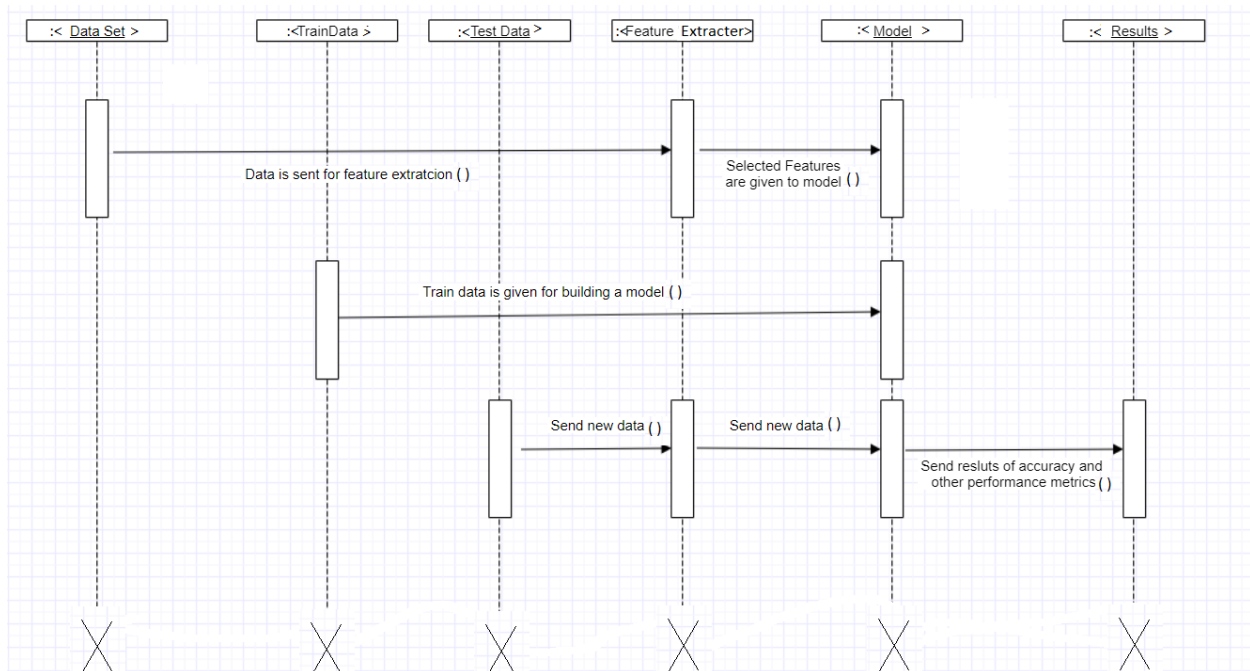
Collaboration, State chart and activity.

6.1.1 Sequential Diagram for finding Accuracy:

Interaction diagram is called is sequence diagram. Interaction diagram describes patterns of communication among a set of interaction objects. An object interacts with another object by sending messages, arguments may be passed along with a message and they are found to be parameters of executing methods in the receiving objects.

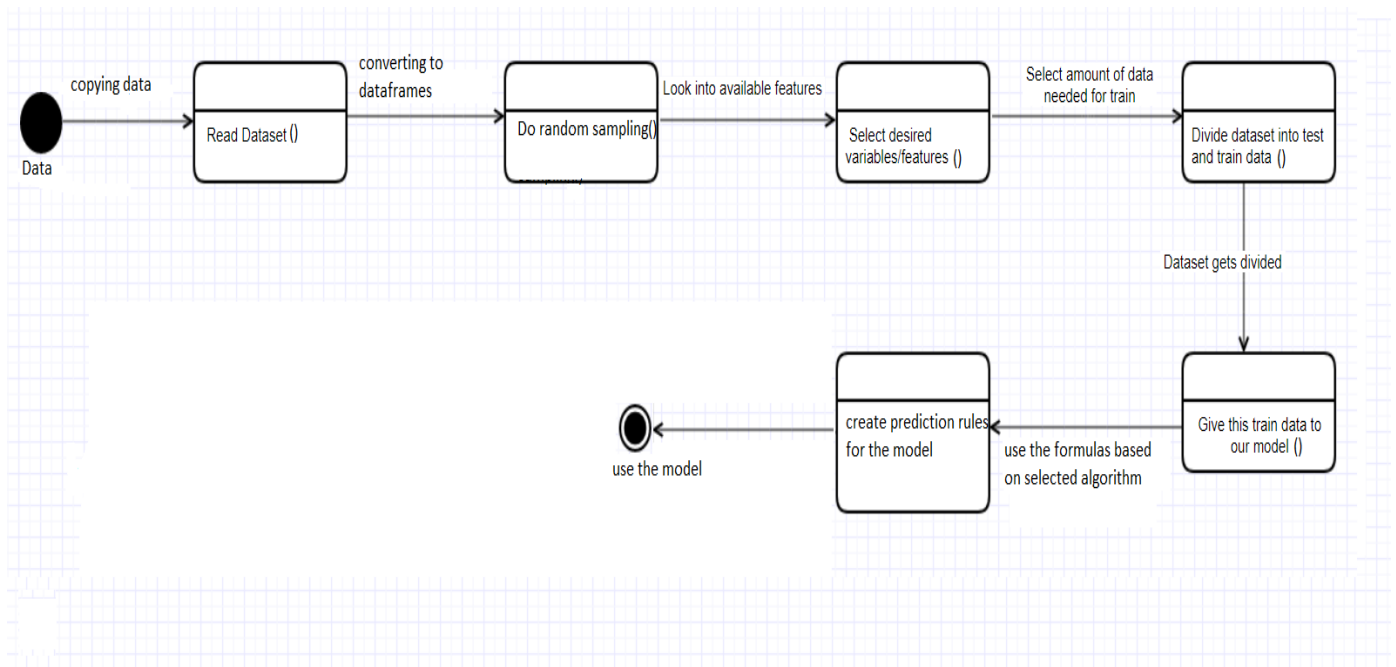
In this Diagram, the Objects are:

- 1.Data set
- 2.Train Data
- 3.Test Data
- 4.Feature Extracter
- 5.Model
- 6.Results



6.1.2 State Chart Diagram for finding Accuracy:

It describes the dynamic nature of the diagram how the objects changes its state. It has various States and Transitions.



7. IMPLEMENTATION

7.1 Code for User Interface:

```
library(shiny)
shinyUI(fluidPage(
  includeHTML("header.html"),
  br(),
  includeHTML("slideshow.html"),
  br(),
  tabsetPanel(
    tabPanel("Upload the FILE",fileInput("file","Upload the file"),uiOutput("tb1")),
    tabPanel("Select the Alogorithm",selectInput("algos", "Machine Learning Algorithms", c("Select an Algorithm","LOGISTIC REGRESSION", "DECISION TREES", "RANDOM FOREST", "NAIVE BAYES")),uiOutput("tb2")),
    tabPanel("testing",uiOutput("tst1"))
  ),
  includeHTML("bottom.html")
))
```

Code for Server:

```
library(shiny)
library(e1071)
# use the below options code if you wish to increase the file input limit, in this
# example file input limit is increased from 5MB to 9MB
options(shiny.maxRequestSize = 1000*1024^2)

shinyServer(function(input,output){
```

This reactive function will take the inputs from UI.R and use them for read.table() to read the data from the file. It returns the dataset in the form of a dataframe.

file\$datapath -> gives the path of the file

```
data <- reactive({  
  file1 <- input$file  
  if(is.null(file1)){return()}  
  read.csv(file=file1$datapath)  
})
```

```
test1<-reactive({  
  data<-data()  
  index<-sample(nrow(data),nrow(data)*.7,replace = TRUE)  
  train<-data[index,]  
  test<-data[-index,]  
  return(test)  
})
```

this reactive output contains the summary of the dataset and display the summary in table format

```
output$filedf <- renderTable({  
  if(is.null(data())){return ()}  
  input$file  
})
```

this reactive output contains the summary of the dataset and display the summary in table format

```
output$sum <- renderTable({  
  if(is.null(data())){return ()}  
  summary(data())  
})
```

```
# This reactive output contains the dataset and display the dataset in table format
output$table <- renderTable({
  if(is.null(data())){return ()}
  data()
})
```

```
datasetInput <- renderText(input$algos)
```

the following renderUI is used to dynamically generate the tabsets when the file is loaded. Until the file is loaded, app will not show the tabset.

```
output$tb1 <- renderUI({
  if(is.null(data()))
    h5("")
  else
    tabsetPanel(tabPanel("About file", tableOutput("filedf")),tabPanel("Data",
tableOutput("table")),tabPanel("Summary", tableOutput("sum")))
})
#algotlist<-c("LOGISTIC REGRESSION", "DESISION TREES", "RANDOM
FOREST", "NAIVE BAYES")
```

```
output$tb2 <- renderUI({
  algoname<-datasetInput()
  var<-variables()
  if(algoname == "LOGISTIC REGRESSION")
  {

    includeHTML("logistic.html")
```

```
}  
else if(algoname == "DECISION TREES")  
{  
  includeHTML("decision_tree.html")  
}  
else if(algoname=="RANDOM FOREST")  
{  
  includeHTML("random.html")  
}  
else if(algoname=="NAIVE BAYES")  
{  
  includeHTML("naive.html")  
}  
})
```

7.2 ACCURACY FOR NAÏVE BAYES

```
library(data.table)  
require(e1071)  
data<-fread("E:\\AFINAL_PROJECT\\R\\creditcard.csv\\creditcard.csv")  
data<-as.data.frame(data)  
data$Class<-as.factor(data$Class)  
  
#ROSE  
library(ROSE)
```


CREDIT CARD FRAUD DETECTION

```
data1<-ROSE(Class ~ ., data = data, seed = 1)
#dividing test and train
index<-sample(nrow(data1$data),nrow(data1$data)*.7)
train<-data1$data[index,]
test<-data1$data[-index,]
#naive bayes
library(class)
data1naive <- naiveBayes(Class~V17+V12+V14+V10+V11, data = train)
data1naivepred <- predict(data1naive,test)
#naive bayes
data1naive2<-
naiveBayes(Class~V17+V12+V14+V10+V11+V16+V18+V9+V4+V7, data =
train)
data1naive2pred <- predict(data1naive2,test)
#naive bayes
data1naive3 <- naiveBayes(Class~., data = train)
data1naive3pred <- predict(data1naive3,test)

A<-function(data,ref)
{
  C<-0
  C1<-0
  C2<-0
  C3<-0
```

```

for(i in 1:length(data)){
  if(data[i]==ref[i]&ref[i]==0)
    {C=C+1
  }else if(data[i]==ref[i]&ref[i]==1){
    C1<-C1+1
  }else if(data[i]!=ref[i]&ref[i]==0){
    C2<-C2+1
  }else{ C3<-C3+1 }
}
M1<-matrix(c(C,C3,C2,C1),byrow = T,ncol = 2,nrow = 2)
rownames(M1)<-c("0","1")
colnames(M1)<-c("0","1")
cat("Pred Ref \n")
print(M1)
cat("\n")
sensitivity<-C/(C+C2)
specificity<-C1/(C1+C3)
accuracy<-(C1+C)/(C1+C2+C3+C)
balanced_accuracy<-(sensitivity+specificity)/2
cat("sensitivity = ", sensitivity, "\n")
cat("specificity = ", specificity , "\n")
cat("accuracy = ", accuracy, "\n")
cat("balanced_accuracy = ", balanced_accuracy, "\n")
}

```

7.3 ACCURACY FOR NEURAL NETWORK

```
#read data
library(data.table)
data<-fread("E:\\AFINAL_PROJECT\\R\\creditcard.csv\\creditcard.csv")
data<-as.data.frame(data)
data$Class<-as.factor(data$Class)

library(ROSE)
data1<-ROSE(Class ~ ., data = data, seed = 1)

data1$Class<-as.factor(data1$Class)
index<-sample(nrow(data1$data),nrow(data1$data)*.7)
train<-data1$data[index,]
test<-data1$data[-index,]

library(h2o)
h2o.init(nthreads = 2)
htrain<-as.h2o(train)
htest<-as.h2o(test)
x<-2:29
y<-31

model<-h2o.deeplearning(x,y,training_frame = htrain,
                        activation = "Rectifier",hidden = c(200),
                        epochs = 10,distribution = "bernoulli" )
pr<-predict(model,htest)
pr1<-as.vector(pr[,1])
A(pr1,test$Class)

x<-c(17,12,14,10,11)
y<-31

model1<-h2o.deeplearning(x,y,training_frame = htrain,
                        activation = "Rectifier",hidden = c(100),
```

CREDIT CARD FRAUD DETECTION

```
epochs = 10,distribution = "bernoulli" )
pr1<-predict(model1,htest)
pr2<-as.vector(pr1[,1])
A(pr2,test$Class)

x<-c(17,12,14,10,11,16,18,9,4,7)
y<-31

model2<-h2o.deeplearning(x,y,training_frame = htrain,
                           activation = "Rectifier",hidden = c(100),
                           epochs = 10,distribution = "bernoulli" )
pr2<-predict(model2,htest)
pr3<-as.vector(pr2[,1])
A(pr3,test$Class)

A<-function(data,ref)
{
  C<-0
  C1<-0
  C2<-0
  C3<-0
  for(i in 1:length(data)){
    if(data[i]==ref[i]&ref[i]==0)
    { C=C+1
    }else if(data[i]==ref[i]&ref[i]==1){
      C1<-C1+1
    }else if(data[i]!=ref[i]&ref[i]==0){
      C2<-C2+1
    }else{ C3<-C3+1 }
  }
  M1<-matrix(c(C,C3,C2,C1),byrow = T,ncol = 2,nrow = 2)
  rownames(M1)<-c("0","1")
  colnames(M1)<-c("0","1")
  cat("Pred Ref \n")
  print(M1)
  cat("\n")
}
```

```
sensitivity<-(C/(C+C2))
specificity<-(C1/(C1+C3))
accuracy<-(C1+C)/(C1+C2+C3+C)
balanced_accuracy<-(sensitivity+specificity)/2
cat("sensitivity = ", sensitivity, "\n")
cat("specificity = ", specificity, "\n")
cat("accuracy = ", accuracy, "\n")
cat("balanced_accuracy = ", balanced_accuracy, "\n")

}
```

7.4 ACCURACY FOR LOGISTIC REGRESSION

```
#read data
library(data.table)
data<-fread("E:\\AFINAL_PROJECT\\R\\creditcard.csv\\creditcard.csv")
data<-as.data.frame(data)
data$Class<-as.factor(data$Class)
#ROSE
library(ROSE)
data1<-ROSE(Class ~ ., data = data, seed = 1)

#divide into test and train
index<-sample(nrow(data1$data),nrow(data1$data)*.7)
train<-data1$data[index,]
test<-data1$data[-index,]
```

CREDIT CARD FRAUD DETECTION

```
#logistic regression model
data1logistic<-glm(formula = Class~V17+V12+V14+V10+V11,family =
binomial(link="logit"),data = train)
data1logisticpred<-predict(logistic[[1]],test,type = "response")
pr<-ifelse(pred>0.5,1,0)
data1logisticpred<-ifelse(data1logisticpred>0.4,1,0)
A(pr,test$Class)
A(data1logisticpred,test$Class)

#
data1logistic2<-glm(formula = Class~V17+V12+V14+V10+V11+V16+V18+V9+V4+V7,family =
binomial(link="logit"),data = train)
data1logistic2pred<-predict(logistic[[2]],test,type = "response")
pr10<-ifelse(pred10>0.5,1,0)
data1logistic2pred<-ifelse(data1logistic2pred>0.4,1,0)
A(pr10,test$Class)
A(pr410,test$Class)

#
data1logistic3<-glm(formula = Class~.,family = binomial(link="logit"),data =
train)
data1logistic3pred<-predict(logistic[[3]],test,type = "response")
pra<-ifelse(preda>0.5,1,0)
```

CREDIT CARD FRAUD DETECTION

```
data1logistic3pred<-ifelse(data1logistic3pred>0.4,1,0)
A(pra,test$Class)
A(pr4a,test$Class)
A<-function(data,ref)
{
  C<-0
  C1<-0
  C2<-0
  C3<-0
  for(i in 1:length(data)){
    if(data[i]==ref[i]&ref[i]==0)
      {C=C+1
    }else if(data[i]==ref[i]&ref[i]==1){
      C1<-C1+1
    }else if(data[i]!=ref[i]&ref[i]==0){
      C2<-C2+1
    }else{ C3<-C3+1 }
  }
  M1<-matrix(c(C,C3,C2,C1),byrow = T,ncol = 2,nrow = 2)
  rownames(M1)<-c("0","1")
  colnames(M1)<-c("0","1")
  cat("Pred Ref \n")
  print(M1)
  cat("\n")
}
```

```
sensitivity<-C/(C+C2)
specificity<-C1/(C1+C3)
accuracy<-(C1+C)/(C1+C2+C3+C)
balanced_accuracy<-(sensitivity+specificity)/2
cat("sensitivity = ", sensitivity, "\n")
cat("specificity = ", specificity , "\n")
cat("accuracy = ", accuracy, "\n")
cat("balanced_accuracy = ", balanced_accuracy, "\n")

}
```

7.5 ACCURACY FOR DECISION TREE

```
library(data.table)
data<-fread("E:\\AFINAL_PROJECT\\R\\creditcard.csv\\creditcard.csv")
data<-as.data.frame(data)
data$Class<-as.factor(data$Class)

#ROSE
library(ROSE)
data1<-ROSE(Class ~ ., data = data, seed = 1)

#divide into test and train
index<-sample(nrow(data1$data),nrow(data1$data)*.7)
train<-data1$data[index,]
test<-data1$data[-index,]

#decision tree model
```


CREDIT CARD FRAUD DETECTION

```
library(rpart)
data1decision<-rpart(Class~V17+V12+V14+V10+V11,data = train)
data1decisionpred<-predict(data1decision,test,type = "class")
A(pred1,test$Class)

#decision tree model
data1decision2<-
rpart(Class~V17+V12+V14+V10+V11+V16+V18+V9+V4+V7,data = train)
data1decision2pred<-predict(data1decision2,test,type = "class")
A(pred1,test$Class)

#decision tree model
data1decision3<-rpart(Class~.,data = train)
data1decision3pred<-predict(data1decision3,test,type = "class")
A(pred1,test$Class)

A<-function(data,ref)
{
  C<-0
  C1<-0
  C2<-0
  C3<-0
  for(i in 1:length(data)){
    if(data[i]==ref[i]&ref[i]==0)
    {C=C+1
    }else if(data[i]==ref[i]&ref[i]==1){
      C1<-C1+1
    }else if(data[i]!=ref[i]&ref[i]==0){
      C2<-C2+1
    }else{ C3<-C3+1 }
  }
  M1<-matrix(c(C,C3,C2,C1),byrow = T,ncol = 2,nrow = 2)
  rownames(M1)<-c("0","1")
  colnames(M1)<-c("0","1")
  cat("Pred Ref \n")
  print(M1)
  cat("\n")
}
```

```
sensitivity<-C/(C+C2)
specificity<-C1/(C1+C3)
accuracy<-(C1+C)/(C1+C2+C3+C)
balanced_accuracy<-(sensitivity+specificity)/2
cat("sensitivity = ", sensitivity, "\n")
cat("specificity = ", specificity , "\n")
cat("accuracy = ", accuracy, "\n")
cat("balanced_accuracy = ", balanced_accuracy, "\n")

}
```

7.6 ACCURACY FOR RANDOM FOREST

```
library(data.table)

data<-fread("E:\\AFINAL_PROJECT\\R\\creditcard.csv\\creditcard.csv")
data<-as.data.frame(data)
data$Class<-as.factor(data$Class)
#ROSE
library(ROSE)
data1<-ROSE(Class ~ ., data = data, seed = 1)

#divide into test and train
index<-sample(nrow(data1$data),nrow(data1$data)*.7)
train<-data1$data[index,]
test<-data1$data[-index,]

library(ranger)
```

CREDIT CARD FRAUD DETECTION

```
data1random<-ranger(Class~V17+V12+V14+V10+V11,data = train,num.trees =  
500,importance = "impurity")
```

```
data1randompred<-predict(data1random,test)
```

```
data1random2<-
```

```
ranger(Class~V17+V12+V14+V10+V11+V16+V18+V9+V4+V7,data =  
train,num.trees = 500,importance = "impurity")
```

```
data1randompred2<-predict(data1random2,test)
```

```
data1random3<-ranger(Class~.,data = train,num.trees = 500,importance =  
"impurity")
```

```
data1randompred3<-predict(data1random3,test)
```

CREDIT CARD FRAUD DETECTION

8.OUTPUT SCREENSHOTS

8.1 DATA SET

creditcard - Microsoft Excel																									
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V			
1	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20				
2	0	-1.3583	-0.3776	2.536347	1.70335	-0.33832	0.462388	0.235995	0.898888	0.383387	0.980954	-0.5518	-0.4576	-0.95335	-0.33117	1.48277	-0.4304	0.207971	0.622793	0.400993	0.253				
3	0	-1.191857	-0.266354	0.16668	0.426354	0.080918	-0.06296	-0.0308	0.05309	-0.25561	-0.16897	0.611727	0.665255	0.489695	-0.16127	0.635558	0.481917	-0.1548	-0.18108	-0.18578	-0.08				
4	1	-1.33825	-1.34816	1.773209	0.37576	-0.2632	1.809495	0.791403	1.547876	-1.53405	0.207843	0.624501	0.666094	0.717255	-0.16395	2.340865	-2.89008	1.509965	-0.12128	-2.26388	0.52				
5	1	-0.96627	-0.18521	1.767991	-0.80329	-0.01011	1.247201	0.233679	0.47418	-1.30301	-0.05495	-0.22849	0.576228	0.507557	0.26793	-0.61582	-1.05965	-0.68209	1.96575	-1.21264	-0.29				
6	2	-1.15421	0.677577	1.548718	0.40354	-0.40715	0.093821	0.592941	-0.27953	0.817739	0.753874	-0.82284	0.530296	1.340352	-1.11967	0.375223	0.40345	-0.21701	-0.03815	0.803487	0.408				
7	2	-0.42097	0.966521	1.331309	-0.16825	0.420987	-0.02971	0.476201	0.260114	-0.56867	-0.17551	1.351267	0.150894	-0.50809	-0.14713	0.517817	0.801726	-0.05413	0.266854	-0.03119	0.584				
8	4	1.329856	0.341004	0.045371	1.202613	0.331803	0.272708	-0.00516	0.011213	0.46496	-0.09925	-1.41891	-0.15303	-0.75506	0.387737	0.850544	0.44335	0.803823	0.61129	-0.04358	-0.21				
9	7	-0.648277	1.457965	1.07418	-0.4932	0.940938	0.420318	1.120831	-0.80388	0.815375	1.200176	-0.619817	0.294174	1.57984	-1.32387	0.888553	0.03913	-1.22213	0.55822	0.44595	-0.15				
10	7	0.89429	0.296357	-0.11115	0.27353	2.665993	1.721818	0.370340	0.851084	0.35005	-0.41043	0.70512	-0.11040	-0.28825	0.874335	-0.32878	0.21008	-0.49577	0.118765	0.570328	0.352				
11	9	-0.13826	1.155991	1.045867	0.222159	0.499384	-0.24878	0.851581	0.665109	-0.71871	-0.36485	1.017814	0.83839	1.008854	0.54512	0.550219	0.399518	0.54898	0.876479	0.451721	0.201				
12	10	1.449544	-1.17834	0.51336	-1.37567	-1.97138	0.52915	-1.42324	0.64846	-1.72941	1.626459	1.299844	0.67544	0.51379	0.09505	0.23093	0.331967	0.253415	0.854344	0.221337	0.38				
13	10	0.685536	0.856095	-0.4538	0.094832	2.525588	1.417027	0.47045	0.582657	0.50809	0.895755	-0.25112	0.53814	-0.09685	0.868832	0.62908	0.17885	-0.80998	0.10985	0.30364	0.125				
14	10	1.249999	-1.22394	0.36393	-1.2340	-1.40542	0.75323	-0.6894	0.22348	-2.09401	1.327529	0.227868	0.34868	1.205417	0.31763	0.725675	0.01561	0.877938	0.34779	0.68393	-0.10				
15	11	1.069178	0.242755	0.828811	0.710251	-0.1708	0.117568	-0.09472	0.115380	-0.21798	0.86623	-0.77984	0.121817	-0.01038	-0.11385	-0.05554	0.15999	0.124835	-0.5886	-0.98293	-0.1				
16	12	-2.75015	-0.32777	1.84515	1.781475	0.13655	0.807598	-0.42291	1.90713	0.752713	1.133067	0.844515	0.792844	0.73488	0.406798	0.30709	0.15567	0.776202	2.221868	1.58					
17	12	-0.75245	0.89548	2.057101	-1.86884	1.15839	-0.07285	-0.6858	0.001401	-0.81817	0.762751	-0.70798	-0.75051	0.047627	-1.0866	1.308154	1.666114	0.77927	-0.51999	0.45515	0.261				
18	12	-1.333121	-0.0403	1.297332	1.200911	-0.736	0.280665	0.58808	0.18938	0.782333	-0.26796	-0.40313	0.526708	0.70838	-0.46485	0.354574	0.24663	0.09023	0.35920	0.57584	-0.11				
19	14	-0.43491	0.010966	0.94891	-0.70722	0.915476	-0.12367	0.797652	0.807867	-0.86527	-0.75798	0.128898	0.27739	0.25824	-0.7919	-0.18842	0.381174	-0.90871	0.688837	0.62548	-0.08				
20	14	-0.40126	-0.40215	1.180305	1.796239	1.69708	-1.76341	-1.53974	0.390842	1.23309	0.340371	0.91723	0.97017	0.28637	-0.47913	0.52861	0.472094	-0.73548	0.875081	-0.40687	-2.15				
21	15	1.487836	-0.02915	0.454795	-1.48001	-1.55545	-0.73896	-1.08866	-0.05411	-1.97888	1.63616	0.575547	0.63265	-0.81896	0.050811	-0.08298	0.16863	0.304281	0.538102	0.05475	-0.38				
22	16	0.694802	-1.36332	1.029221	0.834335	-1.15921	1.399599	-0.87855	0.44529	-0.44405	0.568521	1.015511	1.280329	0.42988	-0.37265	-0.80798	-2.04456	0.515863	0.825847	-1.30943	-0.13				
23	17	0.968186	0.426881	-0.17948	2.489005	1.27656	1.086818	0.33713	0.521901	-1.19151	0.748798	1.69031	0.80674	-0.91847	0.881709	0.710911	-0.60221	0.830884	-1.71718	-0.02761	-0.26				
24	18	1.186816	0.50012	-0.0875	2.351595	0.420804	0.889474	0.341347	0.158803	-0.80316	0.522375	0.384788	-0.51158	-2.10535	1.13887	0.803875	0.484428	-0.464468	-0.098837	-0.1866	-0.36				
25	18	-0.41781	0.277866	1.85471	-0.0916	-1.31819	-0.13812	-0.94838	-1.81795	1.558271	0.40788	-0.5832	0.524931	-0.05118	0.081791	1.555204	-1.78689	0.811511	0.438421	2.171807	-0.21				

8.2 FEATURE EXTRACTION

The screenshot displays the RStudio environment with the following components:

- Source Editor:** Contains R code for loading data, creating a model, and sorting variables by importance.


```

2 data<-read.csv("E:\\AFINAL\\PROJECT\\K\\creditcard.csv\\creditcard.csv")
3 data<-as.data.frame(data)
4 data$class<-as.factor(data$class)
5
6 #ROSE
7
8 library(ROSE)
9 #data1<-ROSE(class~,data,N=1000,p=.5)
10 data1<-ROSE(class ~., data = data, seed = 1)
11
12 #method 1
13 model<-glm(formula = class~,family = binomial(link="logit"),data = data)
14 summary(model)
15 model<-step(model)
16 #method 2
17 library(ranger)
18 ranmodel<-ranger(class~,data = data1,num.trees = 500,importance = "impurity")
19 ranmodel$variable.importance
20 ?sort
21 sort(ranmodel$variable.importance,decreasing = TRUE)
22 summary(model)
23

```
- Environment Pane:** Shows the Global Environment with variables like V25, V26, V27, V28, Amount, and Class. It also lists objects: model (Large glm), model1 (Large rpart), pr1n (List of 5), and ranmodel (Large ranger).
- Console:** Displays the output of the `sort` function, showing a table of variable importance scores.


```

9.968304 10.564336
> sort(ranmodel$variable.importance,decreasing = TRUE)
      V17      V14      V12      V10      V11      V16      V9
164.029451 121.807926 118.057523 79.364017 65.408374 63.393512 38.823539
      V4      V7      V18      V26      V21      V3      V20
29.003653 28.201060 28.136872 18.794212 16.436814 15.693290 14.907155
      V6      V2      V1      Time      V19      V5      V27
14.709166 14.561307 13.775636 12.354899 12.002657 11.982758 11.960617
      V15      V8      Amount      V22      V13      V28      V24
11.423967 11.381498 10.564336 10.183004 10.145619 9.968304 9.543693
      V25      V23
8.640444 7.479812
>

```
- Viewer Pane:** Shows the R documentation for the `sort` function, titled "Sorting or Ordering Vectors".

Description

Sort (or order) a vector or factor (partially) into ascending or descending order. For ordering along more than one variable, e.g., for sorting data frames, see [order](#).

Usage

```
sort(x, decreasing = FALSE, ...)
```

Default S3 method:

8.3 ACCURACY FOR DIFFERENT CLASSIFIERS

8.3.1 Accuracy for Logistic Regression

8.3.1.1 Accuracy for Logistic Regression using Five variables

The screenshot displays the RStudio environment with the following components:

- Source Editor:** Contains R code for fitting a logistic regression model and predicting probabilities.


```

33 #
34 logistic<-glm(formula = class~v17+v12+v14+v10+v11,family = binomial(link="logit"),data = train)
35 pred<-predict(logistic,test,type = "response")
36 pr<-ifelse(pred>0.5,1,0)
37 pr4<-ifelse(pred>0.4,1,0)
38 A(pr,test$class)
39 A(pr4,test$class)
40 <
      
```
- Console:** Shows the execution output, including the model fit, predicted probabilities, and performance metrics.


```

> logistic<-glm(formula = class~v17+v12+v14+v10+v11,family = binomial(link="logit"),data = train)
> pred<-predict(logistic,test,type = "response")
> pr<-ifelse(pred>0.5,1,0)
> pr4<-ifelse(pred>0.4,1,0)
> A(pr,test$class)
Pred Ref
  0      1
0 42262 6120
1  379 36682

sensitivity = 0.9911118
specificity = 0.857016
accuracy = 0.9239376
balanced_accuracy = 0.9240639
> A(pr4,test$class)
Pred Ref
  0      1
0 41751 5320
1  890 37482

sensitivity = 0.9791281
specificity = 0.8757067
accuracy = 0.92732
balanced_accuracy = 0.9274174
      
```
- Environment:** Lists the objects in the global environment:
 - `logistic10`: Large glm (30 elements, 167.5 Mb)
 - `test`: 85443 obs. of 31 variables
 - `train`: 199364 obs. of 31 variables
 - `index`: Large integer (199364 elements, 778.8 kb)
 - `pr`: Large numeric (85443 elements, 5.2 Mb)
 - `pr10`: Large numeric (85443 elements, 5.2 Mb)
 - `pr4`: Large numeric (85443 elements, 5.2 Mb)
 - `pr410`: Large numeric (85443 elements, 5.2 Mb)
 - `pred`: Large numeric (85443 elements, 5.2 Mb)
- Files Plots Packages Help Viewer:** Displays the RStudio help viewer with search results for "R".

8.3.1.2 Accuracy for Logistic Regression using Ten variables

The screenshot displays the RStudio environment with the following components:

- Source Editor:** Contains R code for fitting a logistic regression model and evaluating its performance. The code includes:


```

38 A(pr,test$class)
39 A(pr4,test$class)
40 #
41 logistic10<-glm(formula = Class~V17+V12+V14+V10+V11+V16+V18+V9+V4+V7,family = binomial, data = test)
42 pred10<-predict(logistic10,test,type = "response")
43 pr10<-ifelse(pred10>0.5,1,0)
44 pr410<-ifelse(pred10>0.4,1,0)
45 A(pr10,test$class)
46 A(pr410,test$class)
47
46:20 (Top Level)
      
```
- Console:** Shows the output of the R code, including the predicted values for the test set and the performance metrics:


```

> pred10<-predict(logistic10,test,type = "response")
> pr10<-ifelse(pred10>0.5,1,0)
> pr410<-ifelse(pred10>0.4,1,0)
> A(pr10,test$class)
Pred Ref
  0      1
0 41954 5171
1  687 37631

sensitivity =  0.9838887
specificity =  0.8791879
accuracy =  0.9314397
balanced_accuracy =  0.9315383
> A(pr410,test$class)
Pred Ref
  0      1
0 41193 4483
1 1448 38319

sensitivity =  0.9660421
specificity =  0.8952619
accuracy =  0.9305853
balanced_accuracy =  0.930652
      
```
- Environment:** Lists the objects in the global environment:

Object	Class	Size
test	data.frame	85443 obs. of 31 variables
train	data.frame	199364 obs. of 31 variables
index	Large integer	(199364 elements, 778.8 kb)
pr	Large numeric	(85443 elements, 5.2 Mb)
pr10	Large numeric	(85443 elements, 5.2 Mb)
pr4	Large numeric	(85443 elements, 5.2 Mb)
pr410	Large numeric	(85443 elements, 5.2 Mb)
pred	Large numeric	(85443 elements, 5.2 Mb)
pred10	Large numeric	(85443 elements, 5.2 Mb)
- Files, Plots, Packages, Help, Viewer:** The bottom right pane shows the RStudio logo and a list of vignettes and help pages related to the 'car' package.

8.3.1.3 Accuracy for Logistic Regression using all variables

The screenshot displays the RStudio environment with the following components:

- Source Editor:** Contains R code for fitting a logistic regression model and predicting probabilities. The code is as follows:


```

46 A(pr410,test$class)
47 #
48 logistica<-glm(formula = class~.,family = binomial(link="logit"),data = train)
49 preda<-predict(logistica,test,type = "response")
50 pra<-ifelse(preda>0.5,1,0)
51 pr4a<-ifelse(preda>0.4,1,0)
52 A(pra,test$class)
53 A(pr4a,test$class)
54
54:1 (Top Level)
      
```
- Console:** Shows the output of the model fit and predictions. The output includes:


```

glm.fit: fitted probabilities numerically 0 or 1 occurred
> preda<-predict(logistica,test,type = "response")
> pra<-ifelse(preda>0.5,1,0)
> pr4a<-ifelse(preda>0.4,1,0)
> A(pra,test$class)
Pred Ref
  0    1
0 41853 4863
1  788 37939

sensitivity =  0.9815201
specificity =  0.8863838
accuracy    =  0.9338623
balanced_accuracy =  0.933952
> A(pr4a,test$class)
Pred Ref
  0    1
0 41156 4192
1 1485 38610

sensitivity =  0.9651744
specificity =  0.9020607
accuracy    =  0.933558
balanced_accuracy =  0.9336175
>
      
```
- Environment:** Lists the objects in the global environment:
 - `test`: 85443 obs. of 31 variables
 - `train`: 199364 obs. of 31 variables
 - `index`: Large integer (199364 elements, 778.8 kb)
 - `pr`: Large numeric (85443 elements, 5.2 Mb)
 - `pr10`: Large numeric (85443 elements, 5.2 Mb)
 - `pr4`: Large numeric (85443 elements, 5.2 Mb)
 - `pr410`: Large numeric (85443 elements, 5.2 Mb)
 - `pr4a`: Large numeric (85443 elements, 5.2 Mb)
 - `pra`: Large numeric (85443 elements, 5.2 Mb)
- Files, Plots, Packages, Help, Viewer:** The Help pane is active, showing vignettes and help pages for the `caret` package.
 - Vignettes:**
 - [caret::caret](#): A Short Introduction to the caret Package (PDF, source, R code)
 - [http::secrets](#): Managing secrets (HTML, source, R code)
 - Help pages:**
 - [car::car-internal.Rd](#): Internal Objects for the car package
 - [caret::caret-internal](#): Internal Functions

8.3.2 Accuracy for Random Forest

8.3.2.1 Accuracy for Random Forest using Five variables

The screenshot shows the RStudio interface with a script editor, console, and environment pane. The script defines a function to train and evaluate a Random Forest model using five variables (V12, V14, V10, V11, V16). The console output shows the model's predictions and performance metrics.

```

33
34
35
36
37
38 random<-ranger(class~V12+V14+V10+V11,data = train,num.trees = 500,importance = "impuri
39 prrf<-predict(random,test)
40 A(prrf$predictions,test$class)
41
42 random10<-ranger(class~V12+V14+V10+V11+V16+V18+V9+V4+V7,data = train,num.trees = 500,i
43 prrf10<-predict(random10,test)
44 A(prrf10$predictions,test$class)
45
46 randoma<-ranger(class~,data = train,num.trees = 500,importance = "impurity")
47 prrfa<-predict(randoma,test)
48 A(prrfa$predictions,test$class)
49
50
51 #####
52 A<-function(data,ref)
53 {
54   C<-0
55   for(i in 1:nrow(data))
56     C=C+A(predict(ranger(class~V12+V14+V10+V11,data[i,]),test$class))
57 }
58
59 > A(prrf$predictions,test$class)
60 Pred Ref
61 0      1
62 0 42201 625
63 1  440 42177
64
65 sensitivity = 0.9896813
66 specificity = 0.9853979
67 accuracy = 0.9875356
68 balanced_accuracy = 0.9875396
69
70

```

Environment pane:

Object	Class	Size
prrf	List of 5	
random	Large ranger (14 elements, 74.6 Mb)	
test	85443 obs. of 31 variables	
train	199364 obs. of 31 variables	
index	Large integer (199364 elements, 778.8 kb)	
pr	Large numeric (85443 elements, 5.2 Mb)	
pr10	Large numeric (85443 elements, 5.2 Mb)	
pr4	Large numeric (85443 elements, 5.2 Mb)	
pr410	Large numeric (85443 elements, 5.2 Mb)	

Files pane:

Object	Description
num.trees	Number of trees.
mtry	Number of variables to possibly split at in each node. Default is the (rounded down) square root of the number variables.
importance	Variable importance mode, one of 'none', 'impurity', 'permutation'. The 'impurity' measure is the Gini index for classification and the variance of the responses for regression. For survival, only 'permutation' is available.
write forest	Save ranger. forest object, required for prediction. Set to FALSE to reduce memory usage if no prediction intended.
probability	Grow a probability forest as in Maile et al.

8.3.2.2 Accuracy for Random Forest using Ten variables

The screenshot shows the RStudio interface with a script editor, console, and environment pane. The script defines a function to train and evaluate a Random Forest model using ten variables (V12, V14, V10, V11, V16, V18, V9, V4, V7, V1). The console output shows the model's predictions and performance metrics.

```

37
38 random<-ranger(class~V12+V14+V10+V11+V16+V18+V9+V4+V7,data = train,num.trees = 500,i
39 prrf<-predict(random,test)
40 A(prrf$predictions,test$class)
41
42 random10<-ranger(class~V12+V14+V10+V11+V16+V18+V9+V4+V7,data = train,num.trees = 500,i
43 prrf10<-predict(random10,test)
44 A(prrf10$predictions,test$class)
45
46 randoma<-ranger(class~,data = train,num.trees = 500,importance = "impurity")
47 prrfa<-predict(randoma,test)
48 A(prrfa$predictions,test$class)
49
50
51 #####
52 A<-function(data,ref)
53 {
54   C<-0
55   for(i in 1:nrow(data))
56     C=C+A(predict(ranger(class~V12+V14+V10+V11+V16+V18+V9+V4+V7,data[i,]),test$class))
57 }
58
59 > A(prrf$predictions,test$class)
60 Pred Ref
61 0      1
62 0 42371 183
63 1  270 42619
64
65 sensitivity = 0.9936681
66 specificity = 0.9957245
67 accuracy = 0.9946982
68 balanced_accuracy = 0.9946963
69
70
71 > randoma<-ranger(class~,data = train,num.trees = 500,importance = "impurity")
72 growing trees.. Progress: 3%. Estimated remaining time: 19 minutes, 21 seconds.
73

```

Environment pane:

Object	Class	Size
prrf	List of 5	
random	Large ranger (14 elements, 74.6 Mb)	
test	85443 obs. of 31 variables	
train	199364 obs. of 31 variables	
index	Large integer (199364 elements, 778.8 kb)	
pr	Large numeric (85443 elements, 5.2 Mb)	
pr10	Large numeric (85443 elements, 5.2 Mb)	
pr4	Large numeric (85443 elements, 5.2 Mb)	
pr410	Large numeric (85443 elements, 5.2 Mb)	

Files pane:

Object	Description
num.trees	Number of trees.
mtry	Number of variables to possibly split at in each node. Default is the (rounded down) square root of the number variables.
importance	Variable importance mode, one of 'none', 'impurity', 'permutation'. The 'impurity' measure is the Gini index for classification and the variance of the responses for regression. For survival, only 'permutation' is available.
write forest	Save ranger. forest object, required for prediction. Set to FALSE to reduce memory usage if no prediction intended.
probability	Grow a probability forest as in Maile et al.

8.3.2.3 Accuracy for Random Forest using all variables

The screenshot shows the RStudio environment with the following components:

- Source Editor:** Contains R code for training and testing a Random Forest model using the `ranger` package. The code includes comments and function calls like `random<-ranger`, `prrf<-predict`, and `A(prrf$predictions,test$class)`.
- Console:** Displays the progress of growing trees (from 68% to 98%) and the final accuracy metrics:


```
sensitivity = 0.9972093
specificity = 0.9992757
accuracy = 0.9982444
balanced_accuracy = 0.9982425
```
- Environment:** Lists the objects in the global environment:
 - `prrf`: List of 5
 - `prrf10`: List of 5
 - `prrfa`: List of 5
 - `random`: Large ranger (14 elements, 74.6 Mb)
 - `random10`: Large ranger (14 elements, 45 Mb)
 - `randoma`: Large ranger (14 elements, 39.9 Mb)
 - `test`: 85443 obs. of 31 variables
 - `train`: 199364 obs. of 31 variables
 - `index`: Large integer (199364 elements, 778.8 Kb)
- Viewer:** Shows the documentation for the `ranger` package, including parameters like `num.trees`, `mtry`, `importance`, `write.forest`, and `probability`.

8.3.3 Accuracy for Decision Trees

8.3.3.1 Accuracy for Decision Trees using Five variables

The screenshot shows the RStudio interface with the following components:

- Source Editor:** Contains R code for training and testing a decision tree model using five variables (V17, V12, V14, V10, V11).
- Environment:** Lists objects in the global environment, including `data1decision`, `h1test`, `h1train`, `model1`, `model11`, `model12`, `pr`, `pr2`, `test`, and `train`.
- Console:** Displays the execution of the code, showing the creation of the decision tree model and the resulting accuracy metrics.

R Code in Source Editor:

```
11 index<-sample(nrow(data1$train),nrow(data1$train)*.7)
12 train<-data1$train[index,]
13 test<-data1$train[-index,]
14
15 #decision tree model
16 library(rpart)
17 data1decision<-rpart(Class~V17+V12+V14+V10+V11,data = train)
18 data1decisionpred<-predict(data1decision,test,type = "class")
19 A(data1decisionpred,test$class)
20
21 #decision tree model
22 data1decision2<-rpart(Class~V17+V12+V14+V10+V11+V16+V18+V9+V4+V7,data = train)
23 data1decision2pred<-predict(data1decision2,test,type = "class")
24 A(pred1,test$class)
```

Console Output:

```
> #decision tree model
> library(rpart)
> data1decision<-rpart(Class~V17+V12+V14+V10+V11,data = train)
> data1decisionpred<-predict(data1decision,test,type = "class")
> A(pred1,test$class)
Error in A(pred1, test$class) : object 'pred1' not found
> A(data1decisionpred,test$class)
Pred Ref
0 0 1
0 40892 1376
1 1597 41578

sensitivity = 0.9624138
specificity = 0.9679657
accuracy = 0.9652049
balanced_accuracy = 0.9651898
>
```

8.3.3.2 Accuracy for Decision Trees using Ten variables

The screenshot shows the RStudio interface with the following components:

- Source Editor:** Contains R code for training and testing a decision tree model using 10 variables.
- Environment Pane:** Lists objects in the Global Environment, including `data1decision2`, `h1test`, `h1train`, `model1`, `model11`, `model12`, `pr`, `pr2`, `test`, and `train`.
- Console:** Displays the output of the R code, including model performance metrics and a confusion matrix.

R Code (Source Editor):

```
16 library(rpart)
17 data1decision<-rpart(Class~V17+V12+V14+V10+V11,data = train)
18 data1decisionpred<-predict(data1decision,test,type = "class")
19 A(data1decisionpred,test$class)
20
21 #decision tree model
22 data1decision2<-rpart(Class~V17+V12+V14+V10+V11+V16+V18+V9+V4+V7,data = train)
23 data1decision2pred<-predict(data1decision2,test,type = "class")
24 A(data1decision2pred,test$class)
25
26 #decision tree model
27 data1decision3<-rpart(Class~.,data = train)
28 data1decision3pred<-predict(data1decision3,test,type = "class")
29 A(pred1,test$class)
```

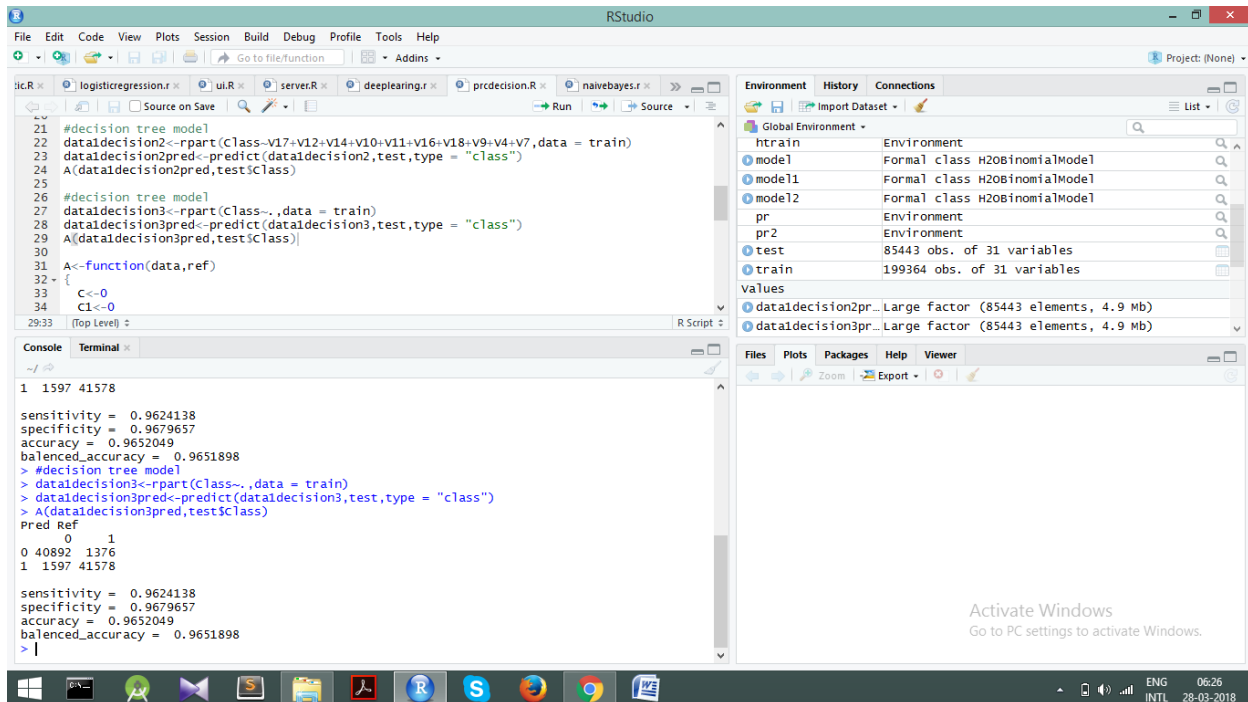
Console Output:

```
1 1597 41578

sensitivity = 0.9624138
specificity = 0.9679657
accuracy = 0.9652049
balanced_accuracy = 0.9651898
> #decision tree model
> data1decision2<-rpart(Class~V17+V12+V14+V10+V11+V16+V18+V9+V4+V7,data = train)
> data1decision2pred<-predict(data1decision2,test,type = "class")
> A(data1decision2pred,test$class)
Pred Ref
0 0 40892 1376
1 1597 41578

sensitivity = 0.9624138
specificity = 0.9679657
accuracy = 0.9652049
balanced_accuracy = 0.9651898
> |
```

8.3.3.3 Accuracy for Decision Trees using all variables



8.3.4 Accuracy for Naive Bayes

8.3.4.1 Accuracy for Naive Bayes using Five variables

The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains R code for training and testing a Naive Bayes model using five variables (V17, V12, V14, V10, V11).
- Environment:** Lists objects in the global environment, including 'model1', 'model2', 'pr', 'pr2', 'test', and 'train'.
- Console:** Shows the output of the R script, including model performance metrics and a confusion matrix.

Console Output:

```
specificity = 0.9679657
accuracy = 0.9652049
balanced_accuracy = 0.9651898
> require(e1071)
Loading required package: e1071
warning message:
package 'e1071' was built under R version 3.4.3
> data1naive <- naiveBayes(Class~V17+V12+V14+V10+V11, data = train)
> data1naivepred <- predict(data1naive,test)
> A(data1naivepred,test$class)
Pred Ref
0 41867 1069
1 622 41865

sensitivity = 0.9853609
specificity = 0.9751129
accuracy = 0.980209
balanced_accuracy = 0.9802369
> |
```

Environment:

Object	Class	Size
model1	Formal class H2OBinomialModel	
model2	Formal class H2OBinomialModel	
pr	Environment	
pr2	Environment	
test	85443 obs. of 31 variables	
train	199364 obs. of 31 variables	

Values:

- data1decision2pr... Large factor (85443 elements, 4.9 Mb)
- data1decision3pr... Large factor (85443 elements, 4.9 Mb)
- data1decisionpred Large factor (85443 elements, 4.9 Mb)
- data1naivepred Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1...

8.3.4.2 Accuracy for Naive Bayes using Ten variables

The screenshot displays the RStudio interface with a script editor, console, and environment pane. The script defines three Naive Bayes models using different variable sets. The console shows the output for the first model, including sensitivity, specificity, accuracy, and balanced accuracy metrics, as well as a confusion matrix and predicted values for the test set.

```

14 #naive bayes
15 library(class)
16 data1naive <- naiveBayes(Class~V17+V12+V14+V10+V11, data = train)
17 data1naivepred <- predict(data1naive,test)
18 A(data1naivepred,test$class)
19 #naive bayes
20 data1naive2 <- naiveBayes(Class~V17+V12+V14+V10+V11+V16+V18+V9+V4+V7, data = train)
21 data1naive2pred <- predict(data1naive2,test)
22 A(data1naive2pred,test$class)
23 #naive bayes
24 data1naive3 <- naiveBayes(Class~., data = train)
25 data1naive3pred <- predict(data1naive3,test)
26 A(data1naive3pred,test$class)
27
23:1 (Top Level) R Script

```

Console Output:

```

1 622 41885

sensitivity = 0.9853609
specificity = 0.9751129
accuracy = 0.980209
balanced_accuracy = 0.9802369
> #naive bayes
> data1naive2 <- naiveBayes(Class~V17+V12+V14+V10+V11+V16+V18+V9+V4+V7, data = train)
> data1naive2pred <- predict(data1naive2,test)
> A(data1naive2pred,test$class)
Pred Ref
0 1
0 42038 500
1 451 42454

sensitivity = 0.9893855
specificity = 0.9883596
accuracy = 0.9888698
balanced_accuracy = 0.9888726
>

```

Environment Pane:

Object	Class	Attributes
model1	Formal class H2OBinomialModel	
model2	Formal class H2OBinomialModel	
pr	Environment	
pr2	Environment	
test	85443 obs. of 31 variables	
train	199364 obs. of 31 variables	
data1decision2pr...	Large factor (85443 elements, 4.9 Mb)	
data1decision3pr...	Large factor (85443 elements, 4.9 Mb)	
data1decisionpred	Large factor (85443 elements, 4.9 Mb)	
data1naive2pred	Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1...	

8.3.4.3 Accuracy for Naive Bayes using all variables

The screenshot displays the RStudio interface with the following components:

- Script Editor:** Contains R code for training and testing a Naive Bayes model using all variables (V1-V31). The code includes:


```

18 A(data1naivepred,test$class)
19 #naive bayes
20 data1naive2 <- naiveBayes(class~V17+V12+V14+V10+V11+V16+V18+V9+V4+V7, data = train)
21 data1naive2pred <- predict(data1naive2,test)
22 A(data1naive2pred,test$class)
23 #naive bayes
24 data1naive3 <- naiveBayes(class~., data = train)
25 data1naive3pred <- predict(data1naive3,test)
26 A(data1naive3pred,test$class)
27
28 A<-function(data,ref)
29 {
30   C<-0
31   CI<-0
32 }
28:1 A(data, ref)
      
```
- Environment:** Lists objects in the global environment:
 - `model2`: Formal class H2ObinomialModel
 - `pr`: Environment
 - `pr2`: Environment
 - `test`: 85443 obs. of 31 variables
 - `train`: 199364 obs. of 31 variables
 - `data1decision2pr`: Large factor (85443 elements, 4.9 Mb)
 - `data1decision3pr`: Large factor (85443 elements, 4.9 Mb)
 - `data1decisionpred`: Large factor (85443 elements, 4.9 Mb)
 - `data1naive2pred`: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1...
 - `data1naive3pred`: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1...
- Console:** Shows the output of the model evaluation:


```

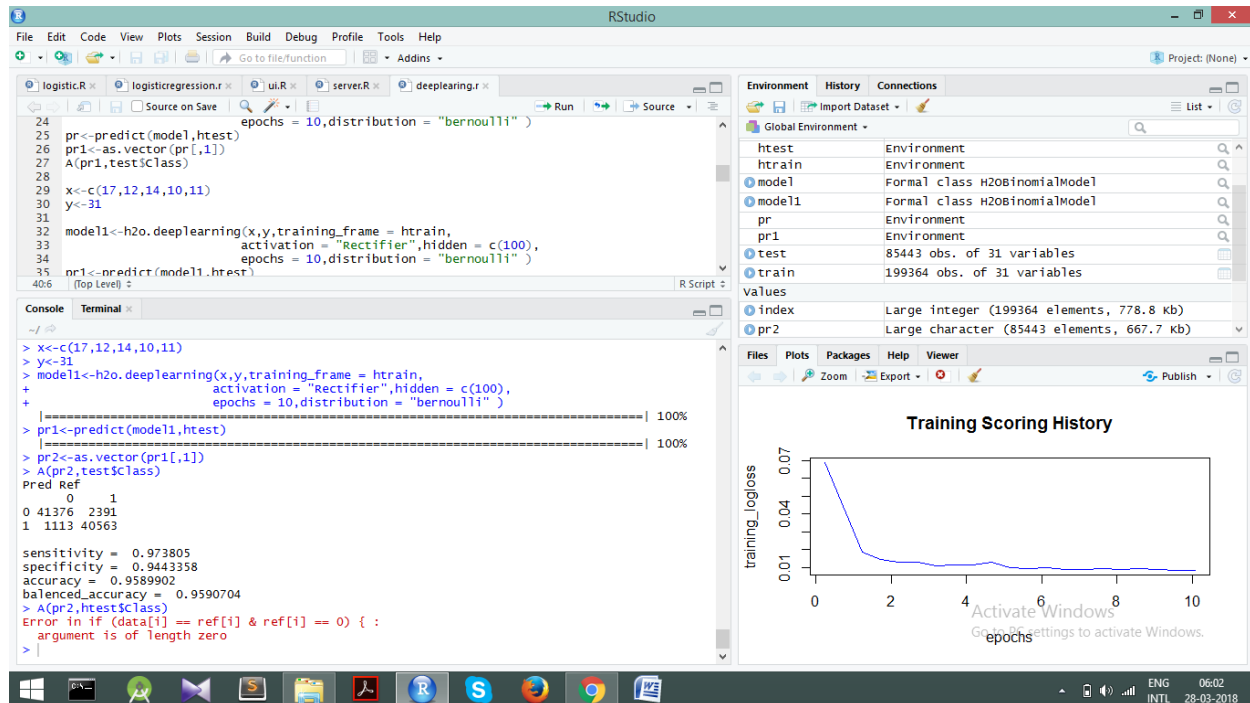
1 451 42454

sensitivity = 0.9893855
specificity = 0.9883596
accuracy = 0.9888698
balanced_accuracy = 0.9888726
> #naive bayes
> data1naive3 <- naiveBayes(class~., data = train)
> data1naive3pred <- predict(data1naive3,test)
> A(data1naive3pred,test$class)
Pred Ref
0 41577 128
1 912 42826

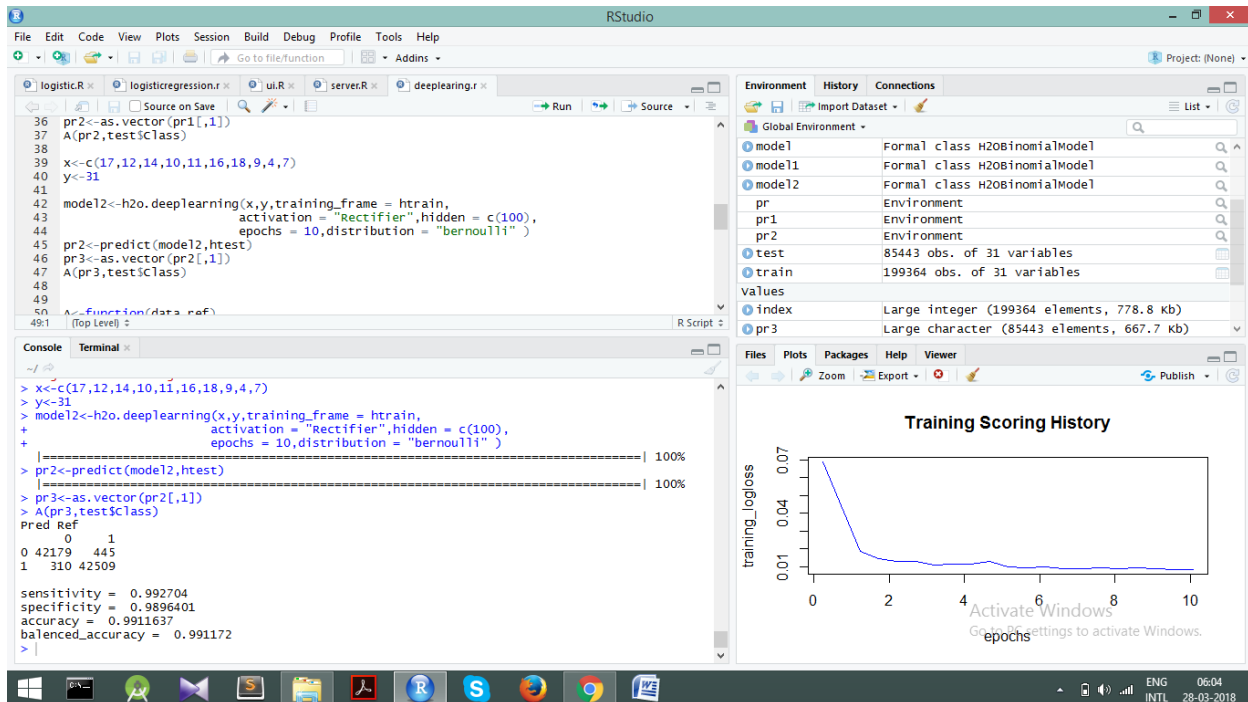
sensitivity = 0.9785356
specificity = 0.9970201
accuracy = 0.9878281
balanced_accuracy = 0.9877778
      
```


8.3.5 Accuracy for Neural Networks

8.3.5.1 Accuracy for Neural Networks using 5 variables



8.3.5.2 Accuracy for Neural Networks using 10 variables



8.3.5.3 Accuracy for Neural Networks using all variables

The screenshot shows the RStudio interface with the following components:

- Script Editor:** Contains R code for loading the `h2o` library, initializing the H2O environment, training a deep learning model, and evaluating its performance.
- Environment:** Lists objects in the global environment, including `model`, `model1`, `model2`, `pr`, `pr2`, `test`, `train`, `index`, `pr1`, and `pr3`.
- Console:** Displays the output of the R script, showing the model's predictions and performance metrics.

Console Output:


```

> x<-2:29
> y<-31
> model<-h2o.deeplearning(x,y,training_frame = htrain,
+                           activation = "Rectifier",hidden = c(200),
+                           epochs = 10,distribution = "bernoulli" )
> pr<-predict(model,htest)
> pr1<-as.vector(pr[,1])
> A(pr1,test$class)
Pred Ref
0      1
0 42399 37
1    90 42917

sensitivity = 0.9978818
specificity = 0.9991386
accuracy = 0.9985136
balanced_accuracy = 0.9985102

```


CREDIT CARD FRAUD DETECTION



INTRODUCTION

Data mining technique is one notable methods used in solving credit fraud detection problem. A lot of algorithms like Neural Networks, Random forest and many other advanced methods can help to detect the fraudulent records. In this project we aim to test methods of Logistic regression, Decision trees, Random forest, Neural networks models and ensemble model. Which includes a comparative analysis of the algorithms based on the Accuracy,Sensitivity,Specificity. Dataset of credit card transactions is sourced from European cardholders containing 284,807 transactions with 30 variables.The positive class (fraud cases) make up 0.172% of the transactions data. The dataset is highly unbalanced and skewed towards the positive class. So we have balanced the dataset using the oversampling technique (ROSE) in R. Dataset contains only numerical (continuous) input variables which are as a result of a Principal Component Analysis(PCA) feature selection transformation resulting to 28 principal components. Out of the 30 variables we have done the feature selection using the Random forest in R and applied all the techniques for first 5 ,10 and all variables. The work is implemented in R.

Upload the FILE

Select the Algorithm

Upload the file

Browse... No file selected

CREDIT CARD FRAUD DETECTION

Copyright © .

Upload the FILE

Select the Algorithm

Machine Learning Algorithms

DECISION TREES

Decision Tree

Decision tree is a type of supervised learning algorithm that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets based on most significant splitter in input variables.

Let's look at the basic terminology used with Decision trees:

Root Node: It represents entire population or sample and this further gets divided into two or more homogeneous sets.

Splitting: It is a process of dividing a node into two or more sub-nodes.

Decision Node: When a sub-node splits into further sub-nodes, then it is called decision node. Leaf/ Terminal Node: Nodes do not split is called Leaf or Terminal node.

Pruning: When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.

Branch / Sub-Tree: A sub section of entire tree is called branch or sub-tree.

Parent and Child Node: A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.

SPLITTING

Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. Decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

CREDIT CARD FRAUD DETECTION

The screenshot shows a web browser window displaying a Shiny application. The browser's address bar shows the URL `http://127.0.0.1:6565`. The application's title bar reads `E:/AFINAL_PROJECT/R/Project_working layouts/MyUi - Shiny`. The interface includes a sidebar with a button labeled `Upload the FILE` and a dropdown menu titled `Machine Learning Algorithms` with `LOGISTIC REGRESSION` selected. The main content area features a section header `Logistic Regression` followed by a detailed text explanation of the algorithm. The text describes it as a classification algorithm for binary outcomes, mentions its historical context (Generalized Linear Model, Nelder and Wedderburn, 1972), and provides the mathematical formulation of the link function. Below the text, the equation $g(y) = \beta_0 + \beta(x)$ is shown, along with definitions for $g()$, y , and x . It then lists two criteria for the probability p and presents the final equation $p = \exp(\beta_0 + \beta(x)) = e^{(\beta_0 + \beta(x))}$. The bottom of the image shows a Windows taskbar with various application icons and a system clock indicating 12:51 on 27-03-2018.

http://127.0.0.1:6565 | Open in Browser | Publish

Upload the FILE | Select the Algorithm

Machine Learning Algorithms

LOGISTIC REGRESSION

Logistic Regression

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. To represent binary / categorical outcome, we use dummy variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as dependent variable. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Logistic Regression is part of a larger class of algorithms known as Generalized Linear Model (glm). In 1972, Nelder and Wedderburn proposed this model with an effort to provide a means of using linear regression to the problems which were not directly suited for application of linear regression. To start with logistic regression, I'll first write the simple linear regression equation with dependent variable enclosed in a link function:

$$g(y) = \beta_0 + \beta(x)$$

where $g()$ is the link function
 y is the outcome variable
 x is the dependent variable

This function is established using two things: Probability of Success(p) and Probability of Failure($1-p$). p should meet following criteria:

1. It must always be positive (since $p \geq 0$)
2. It must always be less than equals to 1 (since $p \leq 1$)

So based on the first criteria by applying exp, it will make sure the value is always ≥ 0

$$p = \exp(\beta_0 + \beta(x)) = e^{(\beta_0 + \beta(x))}$$

9. TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and or/a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations does not fail in unacceptable manner. There are various types of test. Each test type addresses a specific requirement.

9.1 TYPES OF TESTS

UNIT TESTING

Unit testing involves the design of test cases that validate the internal program logic is functioning properly, and that program inputs procedure valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion o an individual unit before integration. This is structural testing, that relies on knowledge of its construction and is invasive. Unit test perform basic test at component level and and test a specific business process, application ,and/or system configuration .Unit test ensures that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfactory, as shown successfully by unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

FUNCTIONAL TEST

Functional tests provide systematic demonstrations that function tests are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centred on the following items:

Valid Input : identified classes of valid input must be accepted

Invalid Input : identified classes of invalid input must be rejected

Functions : identified function must be exercised

Output : identified classes of application outputs must be exercised

Procedures : interfacing systems or procedures must be invoked

Organization and preparation of function tests is focused on requirements, key functions or special test cases. In addition, systematic coverage pertaining to identify business process flow; data fields, predefined process, and successive process must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current test is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is configuration oriented system integration test.

White Box Testing

White Box Testing is a testing in which the software tester has and of the inner workings, structure and language of the software ,or at least its purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kind of tests, must be written from a definitive source document, such as specification or requirements document. It is a testing in which the software under test is treated, as black box you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

9.2 Cross Validation

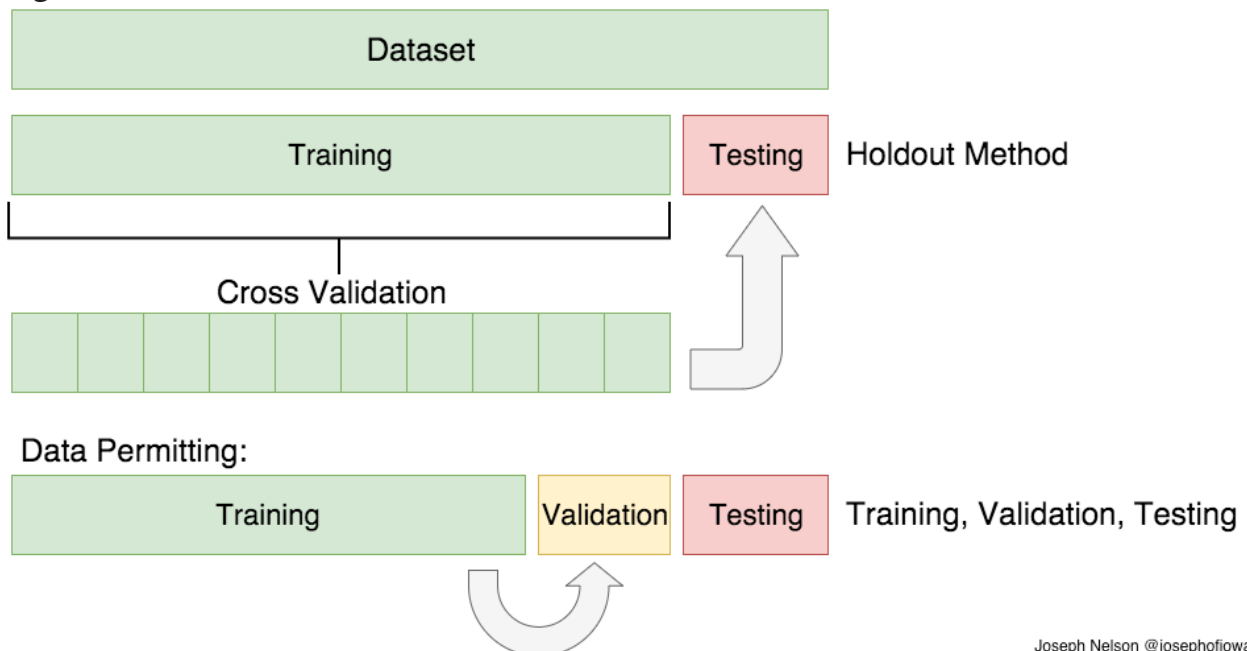
HOLD OUT METHOD

In the holdout method, we randomly assign data points to two sets d_0 and d_1 , usually called the training set and the test set, respectively. The size of each of the sets is arbitrary although typically the test set is smaller than the training set. We then train on d_0 and test on d_1 .

In typical cross-validation, multiple runs are aggregated together; in contrast, the holdout method, in isolation, involves a single run. While the holdout method can be framed as "the simplest kind of cross-validation", many sources instead classify holdout as a type of simple validation, rather than a simple or degenerate form of cross-validation.¹

Fig

10.2.1



9.3 TEST CASES:

Test case for sampling()

Step	Test Steps	Test Data	Expected Results	Actual Result	Status (Pass/Fail)
1.	Import the dataset	Complete dataset	Dataset records shown	All the transaction records are viewed	Pass
2.	Convert data into dataframes format	Complete Dataset	data to be converted into frames	Data has been converted into dataframes	Pass
3.	Do random oversampling using ROSE package		50% of the data is replicated	Sampled data is obtained	

Test case for datasetdivision()

Step	Test Steps	Test Data	Expected Results	Actual Result	Status (Pass/Fail)
1.	Decide on amount of data for test, train sets	Complete dataset			
2.	Give 70% data for training		70% data has to be randomly assigned to train	Data has been assigned	Pass
3.	Give remaining data for test		Rest of the data has to be given to test	Data has been assigned	Pass

CREDIT CARD FRAUD DETECTION

Test case for model creation()

Step	Test Steps	Test Data	Expected Results	Actual Result	Status (Pass/Fail)
1.	Assign train dataset to the model	train dataset			
2.	Choose algorithm and create the model	train dataset	No errors	Model has been created	Pass
3.					

Test case for accuracy()

Step	Test Steps	Test Data	Expected Results	Actual Result	Status (Pass/Fail)
1.	Make predictions for test dataset	Test dataset			
2.	Calculate accuracy	Test dataset	100% accuracy	Greater than 99% accuracy	Pass
3.	Check the confusion matrix	Test dataset			

10.CONCLUSION

In this, we implemented various algorithms and compared their accuracy. We performed Hold-Out method and concluded that “Neural Networks” is the best classifier with accuracy of almost 99 percentage.

The accuracies on different algorithms for the dataset are as follows -

Feature Selection	Logistic regression	Decision Trees	Random Forest	Naive Bayes	Neural Networks
For 5 variables	92.7	96.5	98.7	98.0	95.9
For 10 variables	93.0	96.7	99.4	98.8	99.1
For all variables	93.3	96.7	99.8	98.7	99.9

11.REFERENCES

WEBSITES:

- 1 <https://www.nltk.org/>
2. <https://www.r-studio.org/>
3. <https://www.analyticsvidhya.com/>
- 4.<https://stackoverflow.com>
5. <https://www.quora.com>
6. <https://machinelearningmastery.com/naive-bayes-for-machine-learning/>
7. <https://rpubs.com/aka7h/simple-sentiment-analysis>
8. <http://ieeexplore.ieee.org>
9. <https://www.rdocumentation.org>
10. <https://data-flair.training/blogs/>

BASEPAPER URL:

<https://www.google.com/url?q=https://ieeexplore.ieee.org/document/8123782/&sa=D&source=hangouts&ust=1523955780361000&usg=AFQjCNFdOOBehHlc6gr5DTe4ENSZyPgaXg>