German Credit Data Scoring Using R

Business and Data Analytics Project report

Master of Technology
in
Information Technology

By
Ojha Manish Kumar
14MCMB06
Under Supervision of
Prof.V.Ravi

School of Computer and Information Sciences
University of Hyderabad,
Hyderabad - 500046, India

November, 2015

Table of Contents

1 Abstract **4**

2 Introduction **4**

3 Problem definition **5**

4 About the data **5**

5 Data Preprocessing **6**

5.1 Statistical Analysis 6

5.1.1 Chi squared test 6

5.1.2 T-test: statistical test 6

5.1.3 Principal Component Analysis 8

5.2 Data Transformation 8

5.2.1 Recoding 8

5.2.2 Rescaling 9

5.2.3 Over sampling 9

6 Methodology **10**

6.1 Logistic Regression 10

6.2 Decision Tree 15

6.3 Support Vector Machine 23

6.4 Neural Network 27

6.4.1 PCA with Neural network 31

6.5 k Nearest Neighbour 32

6.6 Random Forest 36

7 Result and Discussions **41**

8 Conclusion **42**

List of Figures

1 PCA using eigen value 9
2 ROC Curve-Logistic Regression 12
3 Sensitivity/Specificity-Logistic Regression 13
4 Decision Tree Plot 20
5 ROC Curve-Decision Tree 21
6 Decision Tree - Sensitivity/Specificity Plot 21
7 Linearly separable set of 2D-points which belong to one of two classes 24
8 ROC Curve - Support Vector Machine 25
9 ROC Curve - Neural Networks 30
10 ROC Curve - K Nearest Neighbour 34
11 Random Forest - Variable Importance 38
12 Random Forest - ROC Curve 39
13 ROC Curve comparison of models 41

List of Tables

1 Chi squared statistics 7
2 t test 8
3 Comparison of models 42
4 Performance measures 42
5 Performance of measures with 10 FCV 43

1 Abstract

Credit scoring uses quantitative measures of the performance and characteristics of past loans to predict the future performance of loans with similar characteristics. The objective of credit scoring is to help credit providers quantify and manage the financial risk involved in providing credit so that they can make better lending decisions quickly and more objectively. As a result, various kinds of credit scoring models are established to evaluate the customers' credit rank.

A credit scoring system should be able to classify customers as good credit those who are expected to repay on time and as bad credit those who are expected to fail. A major problem for banks is how to determine the bad credit, because bad credit may cause serious problems in the future. This leads to loss in bank capital, lower bank revenues and subsequently increases bank losses, which can lead to insolvency or bankruptcy. The categorisation of good and bad credit is of fundamental importance, and is indeed the objective of a credit scoring model. Classification models for credit scoring are used to categorize new applicants as either accepted or rejected with respect to these characteristics. In some cases the final selection of the characteristics was based on the statistical analysis used, i.e. logistic regression, neural network etc.

This study illustrates the use of data mining techniques to construct credit scoring models. Also, it illustrates the comparison of credit scoring models to give a superior final model. The report also highlights each data mining approach using R language.

2 Introduction

Credit scoring is one of the applications for predictive modeling, to predict whether or not credit extended to an applicant will likely result in profit or losses for the lending institution. For instance when a bank provides money to an individual, and expects to be paid back in time with interest commensurate with the risk of default. When a bank grants loan to a new customer, bank uses techniques on the large sample of previous customers with their application details and subsequent credit history available. Applying techniques results in connection between the characteristics of the customers.

Banks use credit risk modeling in order to measure the amount of credit risk which they are exposed to. The most commonly used technique for this purpose is logistic regression. In our study, we applied different techniques like support vector machines, nearest neighbor, decision trees on data to classify the borrowers as good or bad. So that the borrowers which are classified as bad are not granted any credit.

For the experiments, we used the German credit data set which was available in the UCI Repository. The data set consists of 20 attributes (7 numerical and 13 categorical) and there are totally 1000 instances (300 bad and 700 good cases). It was produced by Strathclyde University and is associated with several academic work.

In Section 3 we discussed about the risks that are faced by credit scoring model, to grant loan to a new customer. Section 4 consists of dataset that we are used. Section 5 consists of preprocessing steps that are performed on dataset. Statistical analysis is Basic processing of data,including computing statistical quantities, smoothing, testing, and visualizing, gives a first level of analysis.

Under data preprocessing we applied chi-square test, statistical-test and principle component analysis on german credit dataset. Data transformation, in this preprocessing step, the data transformed or consolidated so that the resulting mining process may be more efficient, and the patterns found may be easier to understand. Dealing with categorical data poses some limitations. For example, if data contained too many categories there would be a need to combine several categories into one. Recode a categorical variable with character values into a variable with numeric values is done by *as.numeric* funcion which is available in R.

In Section 6, we applied various models on german credit dataset. Our aim is to perform models such as decision tree,support vector machines,logistics regression,k-nearest neighbour and random forest. Analysis of each model on the basis of quality and ROC. We compared each model accuracy and finalised which model is giving better accuracy on given dataset.

3 Problem definition

To develop a credit scoring model to predict the credit risk of applicants as bad risk(default) and good risk, which will help credit providers decide whether to grant loan to customers or not. The associated task for this problem is classification, and the German Credit Data set(source::UCI Machine Learning Repository) [2] is using.

4 About the data

To meet with the objective of the analysis, ie, from credit providers perspective, to minimize loss they needs a decision rule regarding who to give approval of the loan and who not to. German Credit Classification dataset obtained from the UCI(University of California,Irvine)Machine Learning Repository [2], was used in this study. The number of examples in this dataset is sufficient and its values for each attribute are complete or available. The detailed description of attributes included in Appendix.

The number of examples in the dataset is 1000.The dataset is classified into two classes:good and bad class. The good class has 700 examples whereas the bad one has 300. The dataset has 20 attributes, Seven of the attributes are of continuous(numerical) types, while the other 13 are of categorical types. The summary of data is given below:

```
dataset<-read.csv("/home/freestyler/BDA_project/Data/german.csv")
summary(dataset)

##  check_status    duration     history      purpose        credit
##  A11:274      Min.   : 4.0   A30: 40   A43    :280   Min.   :  250
##  A12:269      1st Qu.:12.0   A31: 49   A40    :234   1st Qu.: 1366
##  A13: 63      Median :18.0   A32:530   A42    :181   Median : 2320
##  A14:394      Mean   :20.9   A33: 88   A41    :103   Mean   : 3271
##               3rd Qu.:24.0   A34:293   A49    : 97   3rd Qu.: 3972
##               Max.   :72.0             A46    : 50   Max.   :18424
##                                        (Other): 55
##  bonds     jobex         rate        s_status  guarantor    residence
##  A61:603   A71: 62   Min.   :1.000   A91: 50   A101:907   Min.   :1.000
##  A62:103   A72:172   1st Qu.:2.000   A92:310   A102: 41   1st Qu.:2.000
##  A63: 63   A73:339   Median :3.000   A93:548   A103: 52   Median :3.000
##  A64: 48   A74:174   Mean   :2.973   A94: 92              Mean   :2.845
##  A65:183   A75:253   3rd Qu.:4.000                        3rd Qu.:4.000
##                      Max.   :4.000                        Max.   :4.000
##
##  property        age         install     house       nocredit
##  A121:282   Min.   :19.00   A141:139   A151:179   Min.   :1.000
##  A122:232   1st Qu.:27.00   A142: 47   A152:713   1st Qu.:1.000
##  A123:332   Median :33.00   A143:814   A153:108   Median :1.000
##  A124:154   Mean   :35.55                         Mean   :1.407
##             3rd Qu.:42.00                         3rd Qu.:2.000
##             Max.   :75.00                         Max.   :4.000
##
##    job         liable         ph         nri       credibility
##  A171: 22   Min.   :1.000   A191:596   A201:963   Min.   :1.0
##  A172:200   1st Qu.:1.000   A192:404   A202: 37   1st Qu.:1.0
##  A173:630   Median :1.000                         Median :1.0
##  A174:148   Mean   :1.155                         Mean   :1.3
##             3rd Qu.:1.000                         3rd Qu.:2.0
##             Max.   :2.000                         Max.   :2.0
```

```
##
```

The data may not be tidy and we may have to preprocess the data before our analysis can be done. We will discuss how we prepared the data in the following section.

5 Data Preprocessing

5.1 Statistical Analysis

5.1.1 Chi squared test

Before recoding categorical to numerical data, we performed chi squared statistics in original dataset. Since the number of attributes in this problem is not very high, it is possible to look into the dependency of the response (Credibility) on each of them individually. The test is applied when you have two categorical variables from a single dataset. It is used to determine whether there is a significant association between the two variables. Here we are analysing the relation between each attribute with the target variable. Chi squared test is applied for all categorical variables, Below code snippet shows result for one attribute :

```
library("MASS")
data<-read.csv("/home/freestyler/BDA_project/Data/german.csv")
tbl=table(data$check_status,data$credibility)
tbl

##
##         1   2
##   A11 139 135
##   A12 164 105
##   A13  49  14
##   A14 348  46

chisq.test(tbl)

##
##  Pearson's Chi-squared test
##
## data:  tbl
## X-squared = 123.7209, df = 3, p-value < 2.2e-16
```

If the chi square value is large, it tells us that there may be something causing a significant change. A significantly large value will allow us to reject the null hypothesis, which is defined as the prediction that there is no interaction between variables . Basically, if there is a big enough difference between the scores, then we can say something significant happened. If the scores are too close, then we have to conclude that they are basically the same. Overall chi squared result is shown in Table [1].

5.1.2 T-test: statistical test

A t-test is commonly used to determine whether the mean of a population significantly differs from a specific value or from the mean of another population. If the calculated t value exceeds we say that the means are significantly different at that level of probability. Statistical significance is determined by the size of the difference between the group averages, the sample size, and the standard deviations of the groups. For practical purposes statistical significance suggests that the two larger populations from which we sample are “actually“ different. The t-value will be positive if the first mean is larger than the second and negative if it is smaller. T test is applied for interval and normal data [3] here, Below code snippet shows result for one attribute :

Table 1: Chi squared statistics

Attribute name	Chi square value	p- value
check_status	123.72	2.20E-16
history	61.691	1.279e-12
bonds	36.099	0.0000002761
purpose	33.356	0.0001157
property	23.72	2.86E-05
jobex	18.368	0.001045
house	18.2	0.0001117
install	12.839	0.001629
s_status	9.6052	0.02224
guarantor	6.6454	0.03606
nri	5.8216	0.01583
job	1.8852	0.5966
ph	1.1726	0.2789

```
dataset <- read.csv("/home/freestyler/BDA_project/Data/german_test.csv")
library(fBasics, quietly=TRUE)
locationTest(na.omit(dataset[dataset[["credibility"]] == "1", "duration"]),
             na.omit(dataset[dataset[["credibility"]] == "2", "duration"]))

##
## Title:
##  t Test
##
## Test Results:
##   PARAMETER:
##     x Observations: 700
##     y Observations: 300
##     mu: 0
##   SAMPLE ESTIMATES:
##     Mean of x: 19.2071
##     Mean of y: 24.86
##     Var  of x: 122.7567
##     Var  of y: 176.4285
##   STATISTIC:
##                 T: -6.4696
##     T | Equal Var: -6.9523
##   P VALUE:
##     Alternative Two-Sided: 2.404e-10
##     Alternative      Less: 1.202e-10
##     Alternative   Greater: 1
##     Alternative Two-Sided | Equal Var: 6.488e-12
##     Alternative      Less | Equal Var: 3.244e-12
##     Alternative   Greater | Equal Var: 1
##   CONFIDENCE INTERVAL:
##     Two-Sided: -7.3697, -3.936
##          Less: -Inf, -4.2129
##       Greater: -7.0928, Inf
##     Two-Sided | Equal Var: -7.2484, -4.0573
##          Less | Equal Var: -Inf, -4.3142
##       Greater | Equal Var: -6.9915, Inf
```

Table 2: t test

Attribute name	Mean X	Mean Y	p- value
duration	19.2071	24.86	2.404e-10
credit	2985.4571	3938.1267	0.00002478
age	36.2243	33.9633	0.003788
rate	2.92	3.0967	0.02034
nocredit	1.4243	1.3667	0.1416
liable	1.1557	1.1533	0.924
residence	2.8429	2.85	0.925

```
##
## Description:
##  Mon Nov  2 19:41:15 2015
```

Once you compute the t-value you have to look it up in a table of significance to test whether the ratio is large enough to say that the difference between the groups is not likely to have been a chance finding. Overall t-test result(for numerical variables) is shown in Table[2] where mean X is the mean of creditworthy group and mean Y is the mean of non creditworthy group, p-value is associated with t-statistics, and it is the probability of rejecting a true null hypothesis or the probability associated with a false positive. Analysing the below table we found that *liable* and *residence* gives less significance.

5.1.3 Principal Component Analysis

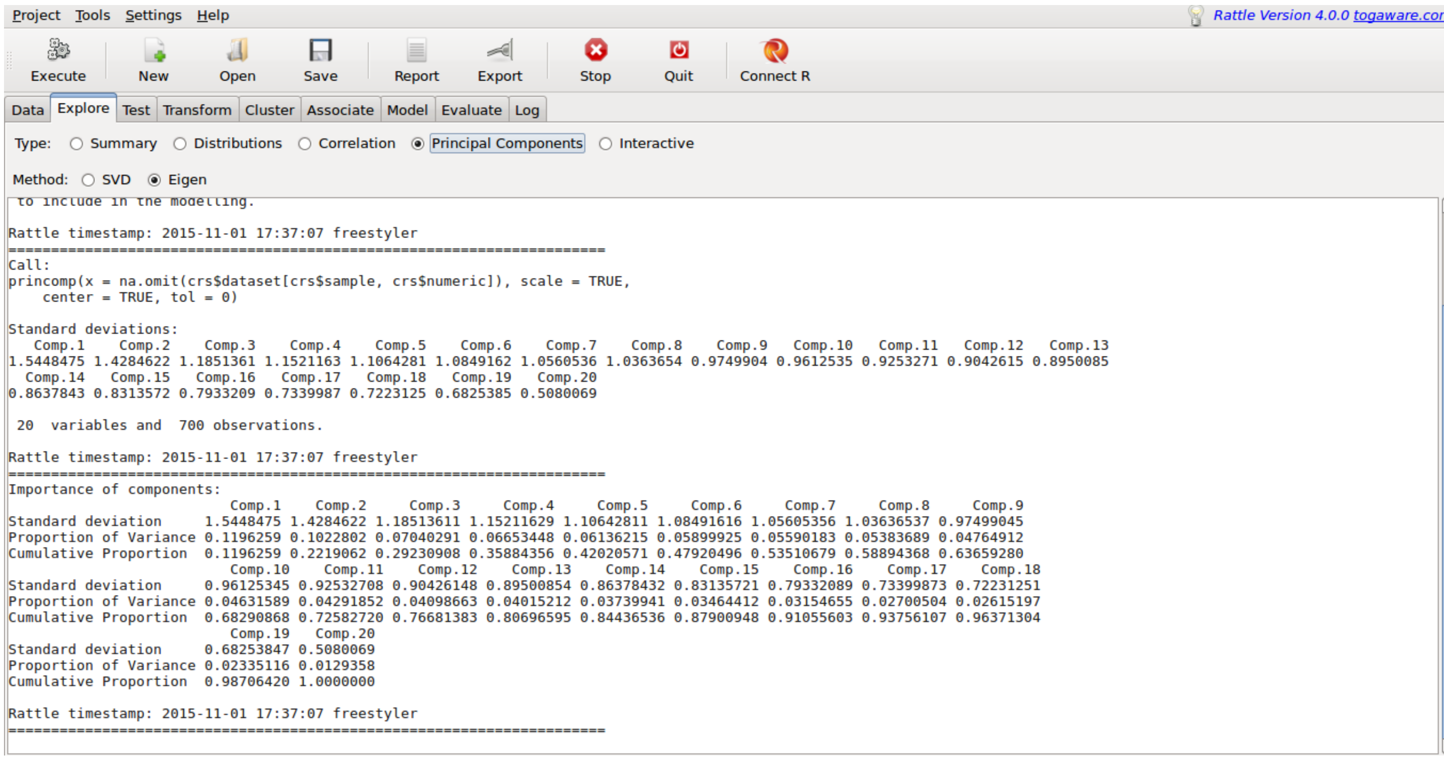
Principal Components Analysis can provide insights into the importance of variables in explaining the variation found within a dataset. A principal component is a numeric linear combination of the values of other variables in the dataset that captures maximal variation in the data. The approach to derive principal components is to determine the eigenvalues of the covariance matrix[5]. The importants of components can be analysed from rattle shown in below figure[1]. From this we can analyse the cumulative proportion of components and we can include the components into our models based on the threshold value we chose.

5.2 Data Transformation

5.2.1 Recoding

After examine the whole data, it is found that there is no missing values for all attributes. Out of 20 attributes, Seven of the attributes are of continuous(numerical) types, while the other 13 are of categorical types. Using *rattle* package 13 categorical type attributes Recoded as numeric.

```
dataset <- read.csv("/home/freestyler/BDA_project/Data/german.csv")
 dataset[["TNM_check_status"]] <- as.numeric( dataset[["check_status"]])
 dataset[["TNM_history"]] <- as.numeric( dataset[["history"]])
 dataset[["TNM_purpose"]] <- as.numeric( dataset[["purpose"]])
 dataset[["TNM_bonds"]] <- as.numeric( dataset[["bonds"]])
 dataset[["TNM_jobex"]] <- as.numeric( dataset[["jobex"]])
 dataset[["TNM_s_status"]] <- as.numeric( dataset[["s_status"]])
 dataset[["TNM_guarantor"]] <- as.numeric( dataset[["guarantor"]])
 dataset[["TNM_property"]] <- as.numeric( dataset[["property"]])
 dataset[["TNM_install"]] <- as.numeric( dataset[["install"]])
 dataset[["TNM_house"]] <- as.numeric( dataset[["house"]])
 dataset[["TNM_job"]] <- as.numeric( dataset[["job"]])
 dataset[["TNM_ph"]] <- as.numeric( dataset[["ph"]])
 dataset[["TNM_nri"]] <- as.numeric( dataset[["nri"]])
```


Project Tools Settings Help
Rattle Version 4.0.0 togaware.com
Execute
New
Open
Save
Report
Export
Stop
Quit
Connect R
Data Explore Test Transform Cluster Associate Model Evaluate Log

Type: ○ Summary ○ Distributions ○ Correlation [✓ Principal Components] ○ Interactive

Method: ○ SVD [✓ Eigen]

```
 to include in the modelling.

Rattle timestamp: 2015-11-01 17:37:07 freestyler
======================================================================
Call:
princomp(x = na.omit(crs$dataset[crs$sample, crs$numeric]), scale = TRUE,
    center = TRUE, tol = 0)

Standard deviations:
   Comp.1    Comp.2    Comp.3    Comp.4    Comp.5    Comp.6    Comp.7    Comp.8    Comp.9   Comp.10   Comp.11   Comp.12   Comp.13
1.5448475 1.4284622 1.1851361 1.1521163 1.1064281 1.0849162 1.0560536 1.0363654 0.9749904 0.9612535 0.9253271 0.9042615 0.8950085
  Comp.14   Comp.15   Comp.16   Comp.17   Comp.18   Comp.19   Comp.20
0.8637843 0.8313572 0.7933209 0.7339987 0.7223125 0.6825385 0.5080069

 20  variables and  700 observations.

Rattle timestamp: 2015-11-01 17:37:07 freestyler
======================================================================
Importance of components:
                          Comp.1    Comp.2     Comp.3     Comp.4     Comp.5     Comp.6     Comp.7     Comp.8     Comp.9
Standard deviation     1.5448475 1.4284622 1.18513611 1.15211629 1.10642811 1.08491616 1.05605356 1.03636537 0.97499045
Proportion of Variance 0.1196259 0.1022802 0.07040291 0.06653448 0.06136215 0.05899925 0.05590183 0.05383689 0.04764912
Cumulative Proportion  0.1196259 0.2219062 0.29230908 0.35884356 0.42020571 0.47920496 0.53510679 0.58894368 0.63659280
                          Comp.10    Comp.11    Comp.12    Comp.13    Comp.14    Comp.15    Comp.16    Comp.17    Comp.18
Standard deviation     0.96125345 0.92532708 0.90426148 0.89500854 0.86378432 0.83135721 0.79332089 0.73399873 0.72231251
Proportion of Variance 0.04631589 0.04291852 0.04098663 0.04015212 0.03739941 0.03464412 0.03154655 0.02700504 0.02615197
Cumulative Proportion  0.68290868 0.72582720 0.76681383 0.80696595 0.84436536 0.87900948 0.91055603 0.93756107 0.96371304
                          Comp.19   Comp.20
Standard deviation     0.68253847 0.5080069
Proportion of Variance 0.02335116 0.0129358
Cumulative Proportion  0.98706420 1.0000000

Rattle timestamp: 2015-11-01 17:37:07 freestyler
======================================================================
```

Figure 1: PCA using eigen value

5.2.2 Rescaling

Using *reshape* library the outcome values (1 = Good, 2 = Bad) rescaled to (0 = Good, 1 = Bad).

```r
library(reshape, quietly=TRUE)
dataset[["R01_credibility"]] <- dataset[["credibility"]]
dataset[["R01_credibility"]] <- ( dataset[["credibility"]] - 1.000000)/abs(2.000000 - 1.000000)
```

5.2.3 Over sampling

As we know that the dataset is classified into two classes:good and bad class. The good class has 700 examples whereas the bad one has 300. Learning from data sets that contain very few instances of the minority (or interesting) class usually produces biased classifiers that have a higher predictive accuracy over the majority class(es), but poorer predictive accuracy over the minority class.

```r
dataset<-read.csv("/home/freestyler/BDA_project/Data/outfile.csv")
df<- data.frame(dataset)
table(df$R01_credibility)

##
##   0   1
## 700 300
```

Using the *unbalanced* package available for R, which implements some of most well-known techniques and propose a racing algorithm to select adaptively the most appropriate strategy for a given unbalanced task. Here we are using *ubOver* [1] function for oversampling, the function replicates randomly some instances from the minority class in order to obtain a final dataset with the same number of instances from the two classes.

```
library(unbalanced)
n <-ncol(train)
out <- ytrain
input <- xtrain

table(ytrain)

## ytrain
##   0   1
## 485 215

outdata<- ubOver(X=input, Y= out)
```

```
train_dataset<-read.csv("/home/freestyler/BDA_project/Data/overSampled_data.csv")
df<- data.frame(train_dataset)
table(train_dataset$R01_credibility)

##
##   0   1
## 485 485
```

By analysing the result from above methods, two attributes has been deleted. We assume that the attributes *liable* and *residence* are less significant to contribute for loan sanction.

6 Methodology

This section will include the methods we used for our analysis. The techniques we studied for this classification problem are Logistic Regression, Decision Tree, Support Vector Machine, Neural network, Random forest available with *rattle* package. Other than that we experimented k nearest neighbour and PCA- Neural network .

6.1 Logistic Regression

Logistic regression models are quite useful for classifying new cases into one of two outcome categories ("success" or "failure"). The estimated logistic model, applied to new cases of a test (evaluation) data set, provides predictions of success probabilities. With a certain cutoff on the predicted success probabilities, the logistic regression provides a rule for classifying new cases. One can use the actual realizations of the cases in the test (evaluation) data set to investigate whether the logistic regression (or any other classification method) is in fact capable of identifying the actual outcomes.

The logistic regression model links the predictor variables to probabilities through the equation:

$$p = f(\alpha + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_k X_k) = \frac{exp(\alpha + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_k X_k)}{1 + exp(\alpha + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_k X_k)} \tag{1}$$

We randomly select 800 of the 1000 cases for the training set, and put the remaining 200 cases to the test set .The R program and its output for the model is listed below.

```
library(rattle)
dataset<-read.csv("/home/freestyler/BDA_project/Data/outfile.csv")
set.seed(42)
nobs <- nrow(dataset) # 1000 observations
```

```
sample <- train <- sample(nrow(dataset), 0.8*nobs) # 800 observations
validate <- NULL
test <- setdiff(setdiff(seq_len(nrow(dataset)), train), validate) # 200 observations
```

The regression model built. The code below estimates a logistic regression model using the glm (generalized linear model) function.

```
glm <- glm(R01_credibility ~ .,
               data=dataset[train, c(input, target)],
               family=binomial(link="logit"))

print(summary(glm))

##
## Call:
## glm(formula = R01_credibility ~ ., family = binomial(link = "logit"),
##     data = dataset[train, c(input, target)])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9202  -0.7486  -0.4362   0.7863   2.4630
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)       5.373e+00  1.124e+00   4.780 1.76e-06 ***
## duration          2.853e-02  9.914e-03   2.878 0.004000 **
## credit            5.210e-05  4.466e-05   1.167 0.243310
## rate              2.451e-01  9.297e-02   2.636 0.008390 **
## age              -1.037e-02  8.889e-03  -1.166 0.243500
## nocredit          2.765e-01  1.760e-01   1.571 0.116141
## TNM_check_status -5.992e-01  7.818e-02  -7.664 1.80e-14 ***
## TNM_history      -4.278e-01  9.663e-02  -4.427 9.55e-06 ***
## TNM_purpose      -3.784e-02  3.433e-02  -1.102 0.270388
## TNM_bonds        -2.234e-01  6.714e-02  -3.327 0.000877 ***
## TNM_jobex        -1.637e-01  7.835e-02  -2.089 0.036686 *
## TNM_s_status     -1.996e-01  1.251e-01  -1.595 0.110604
## TNM_guarantor    -5.074e-01  2.016e-01  -2.516 0.011866 *
## TNM_property      1.817e-01  1.000e-01   1.817 0.069292 .
## TNM_install      -3.320e-01  1.270e-01  -2.614 0.008955 **
## TNM_house        -2.212e-01  1.847e-01  -1.198 0.230959
## TNM_job          -9.282e-02  1.505e-01  -0.617 0.537296
## TNM_ph           -2.969e-01  2.085e-01  -1.424 0.154394
## TNM_nri          -1.044e+00  6.230e-01  -1.676 0.093835 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 985.71  on 799  degrees of freedom
## Residual deviance: 766.61  on 781  degrees of freedom
## AIC: 804.61
##
## Number of Fisher Scoring iterations: 5
```

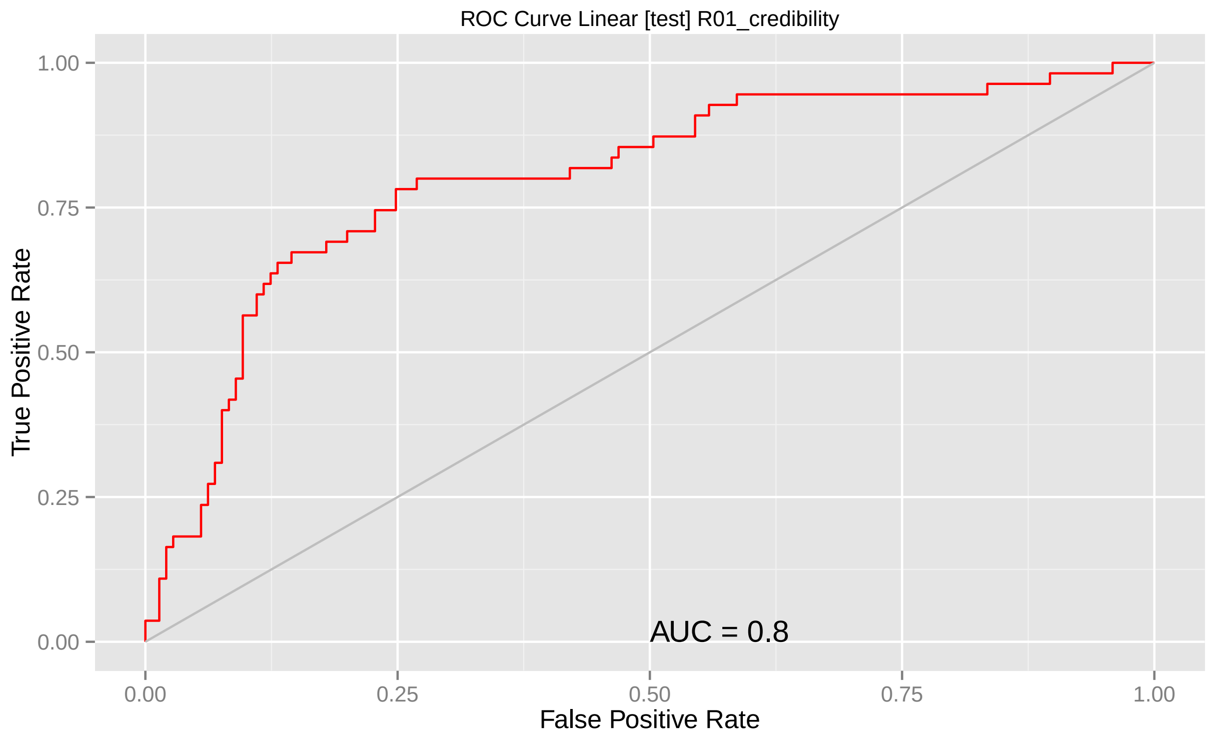


Figure 2: ROC Curve-Logistic Regression

When developing models for prediction, the most critical metric regards how well the model does in predicting the target variable on out of sample observations. This process involves using the model estimates to predict values on the test set. Afterwards, we compared the predicted target variable versus the observed values for each observation.

```
library(ROCR)
library(ggplot2, quietly=TRUE)

pr <- predict(glm, type="response", newdata=dataset[test, c(input, target)])
```

The ROC curve is plotted Figure [2] and the Area Under Curve obtained is 0.8.

```
plot(p)
```

The error matrix for the model generated. Below code produce the confusion(error) matrix showing counts.

```
pr <- as.vector(ifelse(predict(glm, type="response",
                               newdata=dataset[test, c(input, target)]) > 0.5, "1", "0"))

table(dataset[test, c(input, target)]$R01_credibility, pr,
      dnn=c("Actual", "Predicted"))

##         Predicted
## Actual    0    1
```

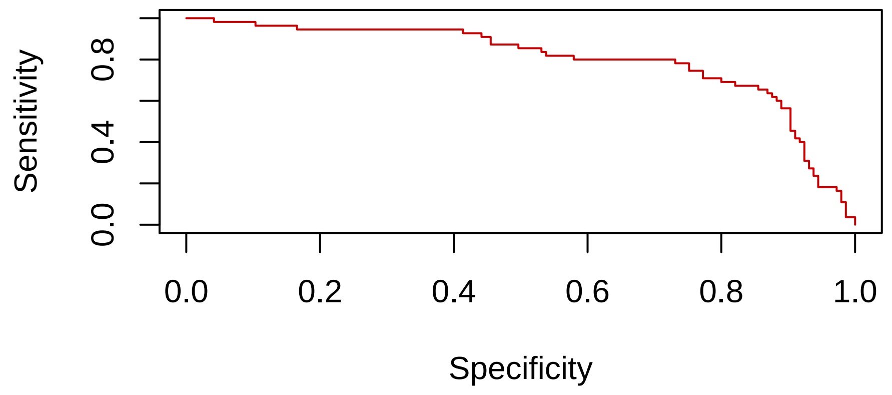


Figure 3: Sensitivity/Specificity-Logistic Regression

```
##       0 131  14
##       1  27  28
```

And the overall error percentage is :

```
overall(table(pr, dataset[test, c(input, target)]$R01_credibility,
              dnn=c("Predicted", "Actual")))

## 0.205
```

The error matrix showing the proportion is :

```
per <- pcme(dataset[test, c(input, target)]$R01_credibility, pr)
round(per, 2)

##        Predicted
## Actual     0    1 Error
##       0 0.66 0.07  0.10
##       1 0.14 0.14  0.49
```

From above error matrix we will get sensitivity and specificity proportion as (1-error) from first row and second row respectively. The plot in Figure [2] gives sensitivity(true positive rate) against specificity(true negative rate) :

```
plot(performance(pred, "sens", "spec"), col="#CC0000FF", lty=1, add=FALSE)
```

Thus the whole performance of Logistic Regression model is analysed by the measures discussed above, and it is giving an accuracy of 79.5%.

The k-fold cross validation method involves splitting the dataset into k-subsets. For each subset is held out while the model is trained on all other subsets. This process is completed until accuracy is determine for each instance in the dataset, and an overall accuracy estimate is provided. We observe that we have the same number of examples in each fold. We applied 10-fold cross validation on 1000 samples of the german credit dataset using the package *survival*.

```
library(survival)
dataset <- read.csv("/home/freestyler/BDA_project/Data/outfile.csv")
n<-nrow(dataset)
K<-10
divide<- n %/% K
set.seed(5)
unirand<-runif(n)
rang<-rank(unirand)
bloc<-(rang - 1)%/%divide +1
bloc<-as.factor(bloc)
print(summary(bloc))

##   1   2   3   4   5   6   7   8   9  10
## 100 100 100 100 100 100 100 100 100 100
```

We can repeat now the learning process and the test process. We collect each value in a vector. Printing each sub-sample's sensitivity ,specificity and accuracy :

```
all.sense<-numeric(0)
all.spese<-numeric(0)
all.acc1<-numeric(0)
for(k in 1:K){
glm <- glm(R01_credibility ~., data = dataset[bloc!=k,], family=binomial(link="logit"))
pred <- as.vector(ifelse(predict(glm,newdata = dataset[bloc==k,],type = 'response')
                         > 0.5, "1", "0"))
#confusion matrix for each partition
mc<-table(dataset$R01_credibility[bloc==k],pred)
err<-1.0 - (mc[1,1]+mc[2,2])/sum(mc)
acc1<-1-(err)
a<-mc[1,1]
b<-mc[1,1]+mc[1,2]
sensitivity<-a/b
c<-mc[2,2]
d<-mc[2,2]+mc[2,1]
specificity<-c/d
#function combines vector, matrix or data frame by rows.
all.sense<-rbind(all.sense,sensitivity)
all.spese<-rbind(all.spese,specificity)
#all.err <- rbind(all.err,err)
all.acc1<-rbind(all.acc1,acc1)
}
```

The sensitivity, specificity and accuracy in each fold is listed below

```
print(all.sense)

##                  [,1]
## sensitivity 0.8750000
## sensitivity 0.9253731
## sensitivity 0.8611111
## sensitivity 0.9189189
## sensitivity 0.8591549
## sensitivity 0.9402985
## sensitivity 0.9242424
## sensitivity 0.8333333
## sensitivity 0.8205128
## sensitivity 0.8805970

print(all.spese)

##                   [,1]
## specificity 0.5000000
## specificity 0.6363636
## specificity 0.5000000
## specificity 0.3076923
## specificity 0.6206897
## specificity 0.4545455
## specificity 0.4411765
## specificity 0.3823529
## specificity 0.3636364
## specificity 0.4848485

#print(all.err)
print(all.acc1)

##      [,1]
## acc1 0.77
## acc1 0.83
## acc1 0.76
## acc1 0.76
## acc1 0.79
## acc1 0.78
## acc1 0.76
## acc1 0.68
## acc1 0.72
## acc1 0.75
```

Because we have the same number of examples in each fold, we can compute unweighted mean. This is the average of each sub-sample's sensitivity:

```
## [1] 0.8838542
```

This is the average of each sub-sample's specificity:

```
## [1] 0.4691305
```

This is the cross validation accuracy estimation:

```
## [1] 0.76
```

6.2 Decision Tree

Anything that we can express in terms of only two outcomes can be handled as a binary classification problem. A decision tree model is one of the most common data mining models. It is popular because the resulting model is easy to understand. The algorithms use a recursive partitioning approach. The traditional algorithm is implemented in the *rpart* package. It is comparable to CART and ID3/C4.

Classification and regression trees are known under their acronym CART. If the response is categorical then it is about classification trees ;If the response is continuous then it is about regression trees . Regression trees try to predict a (numeric) mean response at the leaves of the tree, such as the expected amount of rain in inches, or the expected default rate on loans. Classification trees try to predict the class probabilities at the leaves, such as the probability that there will be rain, the preference for one of five different types of movie genres,or here the probability of defaulting on a loan.

Tree construction uses a recursive partitioning of the data set; the approach is also referred to as divide-and-conquer partitioning. At each stage, the data set is split into two nonoverlapping, smaller data sets. The objective of a split is to increase the homogeneity of the resulting smaller data sets with respect to the target variable. We continually divide the data set by creating node splits of smaller and smaller data sets[4].

We randomly select 800 of the 1000 cases for the training set, and put the remaining 200 cases to the test set like in previous model. The R program and its output for the model is listed below.

Decision tree model built using *rpart* function.

```
library(rpart, quietly=TRUE)
library(rpart.plot)
set.seed(42)

rpart <- rpart(R01_credibility ~ .,
                   data=dataset[train, c(input, target)],
                   method="class",
                   parms=list(split="information"),
                   control=rpart.control(usesurrogate=0,
                                         maxsurrogate=0))
```

The textual view of the Decision tree is :

```
## n= 800
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##    1) root 800 245 0 (0.69375000 0.30625000)
##      2) TNM_check_status>=2.5 364  48 0 (0.86813187 0.13186813) *
##      3) TNM_check_status< 2.5 436 197 0 (0.54816514 0.45183486)
##        6) duration< 22.5 253  91 0 (0.64031621 0.35968379)
##         12) TNM_history>=2.5 228  72 0 (0.68421053 0.31578947)
##           24) credit< 624 11   0 0 (1.00000000 0.00000000) *
##           25) credit>=624 217  72 0 (0.66820276 0.33179724)
##             50) TNM_guarantor>=2.5 23   2 0 (0.91304348 0.08695652) *
##             51) TNM_guarantor< 2.5 194  70 0 (0.63917526 0.36082474)
##              102) credit>=904.5 173  55 0 (0.68208092 0.31791908)
##                204) TNM_s_status>=2.5 103  25 0 (0.75728155 0.24271845) *
##                205) TNM_s_status< 2.5 70  30 0 (0.57142857 0.42857143)
##                  410) TNM_bonds>=2.5 16   3 0 (0.81250000 0.18750000) *
##                  411) TNM_bonds< 2.5 54  27 0 (0.50000000 0.50000000)
##                    822) credit>=1387.5 35  13 0 (0.62857143 0.37142857)
##                     1644) TNM_ph< 1.5 24   5 0 (0.79166667 0.20833333) *
##                     1645) TNM_ph>=1.5 11   3 1 (0.27272727 0.72727273) *
##                    823) credit< 1387.5 19   5 1 (0.26315789 0.73684211) *
##              103) credit< 904.5 21   6 1 (0.28571429 0.71428571) *
##         13) TNM_history< 2.5 25   6 1 (0.24000000 0.76000000) *
##        7) duration>=22.5 183  77 1 (0.42076503 0.57923497)
##         14) TNM_purpose>=1.5 149  70 1 (0.46979866 0.53020134)
##           28) TNM_bonds>=3.5 19   4 0 (0.78947368 0.21052632) *
##           29) TNM_bonds< 3.5 130  55 1 (0.42307692 0.57692308)
##             58) duration< 46.5 108  52 1 (0.48148148 0.51851852)
##              116) TNM_purpose< 3.5 23   5 0 (0.78260870 0.21739130) *
##              117) TNM_purpose>=3.5 85  34 1 (0.40000000 0.60000000) *
##             59) duration>=46.5 22   3 1 (0.13636364 0.86363636) *
##         15) TNM_purpose< 1.5 34   7 1 (0.20588235 0.79411765) *
##
## Classification tree:
## rpart(formula = R01_credibility ~ ., data = dataset[train, c(input,
##     target)], method = "class", parms = list(split = "information"),
##     control = rpart.control(usesurrogate = 0, maxsurrogate = 0))
##
## Variables actually used in tree construction:
## [1] credit           duration         TNM_bonds        TNM_check_status
## [5] TNM_guarantor    TNM_history      TNM_ph           TNM_purpose
## [9] TNM_s_status
##
## Root node error: 245/800 = 0.30625
##
## n= 800
##
##         CP nsplit rel error  xerror     xstd
## 1 0.059184      0   1.00000 1.00000 0.053213
## 2 0.053061      2   0.88163 1.02857 0.053627
## 3 0.022449      3   0.82857 0.94286 0.052318
## 4 0.012245      7   0.73061 0.83265 0.050318
## 5 0.010000     14   0.63673 0.79592 0.049566
```

And the list of the rules obtained from tree is :

```
asRules(rpart)

##
##  Rule number: 59 [R01_credibility=1 cover=22 (3%) prob=0.86]
##    TNM_check_status< 2.5
##    duration>=22.5
##    TNM_purpose>=1.5
##    TNM_bonds< 3.5
##    duration>=46.5
##
##  Rule number: 15 [R01_credibility=1 cover=34 (4%) prob=0.79]
##    TNM_check_status< 2.5
##    duration>=22.5
##    TNM_purpose< 1.5
##
##  Rule number: 13 [R01_credibility=1 cover=25 (3%) prob=0.76]
##    TNM_check_status< 2.5
##    duration< 22.5
##    TNM_history< 2.5
##
##  Rule number: 823 [R01_credibility=1 cover=19 (2%) prob=0.74]
##    TNM_check_status< 2.5
##    duration< 22.5
##    TNM_history>=2.5
##    credit>=624
##    TNM_guarantor< 2.5
##    credit>=904.5
##    TNM_s_status< 2.5
##    TNM_bonds< 2.5
##    credit< 1388
##
##  Rule number: 1645 [R01_credibility=1 cover=11 (1%) prob=0.73]
##    TNM_check_status< 2.5
##    duration< 22.5
##    TNM_history>=2.5
##    credit>=624
##    TNM_guarantor< 2.5
##    credit>=904.5
##    TNM_s_status< 2.5
##    TNM_bonds< 2.5
##    credit>=1388
##    TNM_ph>=1.5
##
##  Rule number: 103 [R01_credibility=1 cover=21 (3%) prob=0.71]
##    TNM_check_status< 2.5
##    duration< 22.5
##    TNM_history>=2.5
##    credit>=624
##    TNM_guarantor< 2.5
##    credit< 904.5
##
##  Rule number: 117 [R01_credibility=1 cover=85 (11%) prob=0.60]
##    TNM_check_status< 2.5
##    duration>=22.5
##    TNM_purpose>=1.5
##    TNM_bonds< 3.5
##    duration< 46.5
##    TNM_purpose>=3.5
##
##  Rule number: 204 [R01_credibility=0 cover=103 (13%) prob=0.24]
##    TNM_check_status< 2.5
##    duration< 22.5
##    TNM_history>=2.5
##    credit>=624
##    TNM_guarantor< 2.5
##    credit>=904.5
##    TNM_s_status>=2.5
##
##  Rule number: 116 [R01_credibility=0 cover=23 (3%) prob=0.22]
##    TNM_check_status< 2.5
##    duration>=22.5
##    TNM_purpose>=1.5
##    TNM_bonds< 3.5
##    duration< 46.5
##    TNM_purpose< 3.5
##
##  Rule number: 28 [R01_credibility=0 cover=19 (2%) prob=0.21]
##    TNM_check_status< 2.5
##    duration>=22.5
##    TNM_purpose>=1.5
##    TNM_bonds>=3.5
##
##  Rule number: 1644 [R01_credibility=0 cover=24 (3%) prob=0.21]
##    TNM_check_status< 2.5
##    duration< 22.5
##    TNM_history>=2.5
##    credit>=624
##    TNM_guarantor< 2.5
##    credit>=904.5
##    TNM_s_status< 2.5
##    TNM_bonds< 2.5
##    credit>=1388
##    TNM_ph< 1.5
##
##  Rule number: 410 [R01_credibility=0 cover=16 (2%) prob=0.19]
##    TNM_check_status< 2.5
##    duration< 22.5
##    TNM_history>=2.5
##    credit>=624
##    TNM_guarantor< 2.5
##    credit>=904.5
##    TNM_s_status< 2.5
##    TNM_bonds>=2.5
##
##  Rule number: 2 [R01_credibility=0 cover=364 (46%) prob=0.13]
##    TNM_check_status>=2.5
##
##  Rule number: 50 [R01_credibility=0 cover=23 (3%) prob=0.09]
##    TNM_check_status< 2.5
##    duration< 22.5
##    TNM_history>=2.5
##    credit>=624
##    TNM_guarantor>=2.5
##
##  Rule number: 24 [R01_credibility=0 cover=11 (1%) prob=0.00]
##    TNM_check_status< 2.5
##    duration< 22.5
##    TNM_history>=2.5
##    credit< 624
```

The plot of the Decision tree is shown in Figure [4]:

```
prp(rpart)
```

Model Performance : For evaluating the model performance, we generated confusion matrix for the Decision tree model. First generated the response from the model:

```
pr <- predict(rpart, newdata=dataset[test, c(input, target)], type="class")
```

And the confusion matrix showing count is :

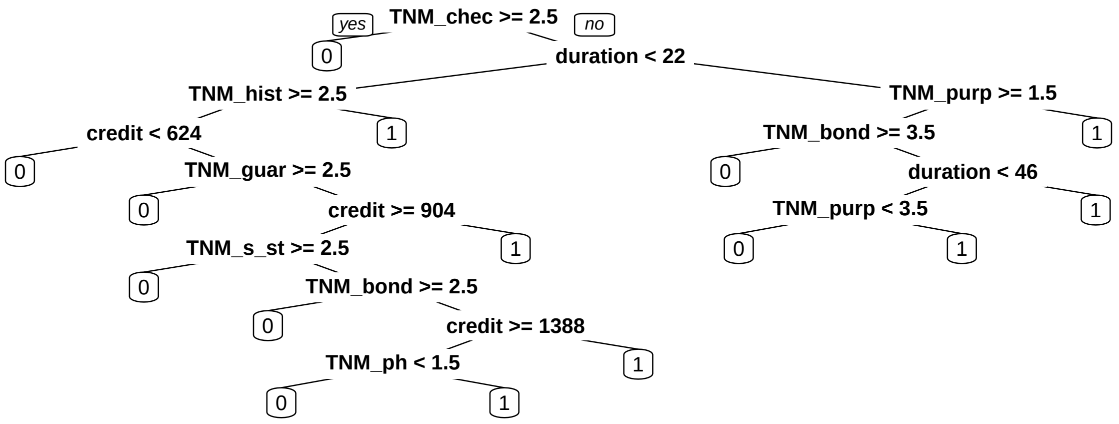


Figure 4: Decision Tree Plot

```
table(dataset[test, c(input, target)]$R01_credibility, pr,
      dnn=c("Actual", "Predicted"))

##       Predicted
## Actual   0   1
##      0 124  21
##      1  23  32
```

Confusion matrix showing the proportions:

```
pcme(dataset[test, c(input, target)]$R01_credibility, pr)

##       Predicted
## Actual    0    1 Error
##      0 0.62 0.10  0.14
##      1 0.12 0.16  0.42
```

From above error matrix we will get sensitivity and specificity proportion as (1-error) from first row and second row respectively. The overall error percentage is:

```
overall(table(pr, dataset[test, c(input, target)]$R01_credibility,
              dnn=c("Predicted", "Actual")))

## 0.22
```

ROC curve for Decision tree model is shown in Figure [5]. The line right on the diagonal serves like a boundary between the good and bad models. The curve approaching to the top left hints a good fitting model.

```
print(p)
```

And the specificity/sensitivity plot is shown in Figure [6].

```
plot(performance(pred, "sens", "spec"), col="#CC0000FF", lty=1, add=FALSE)
```

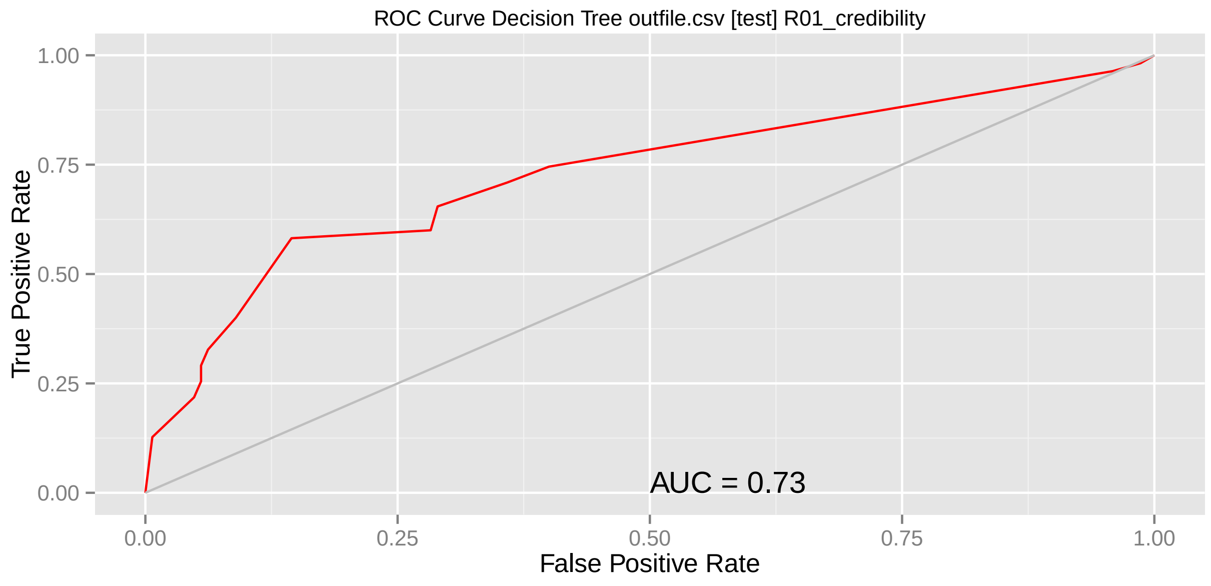


Figure 5: ROC Curve-Decision Tree

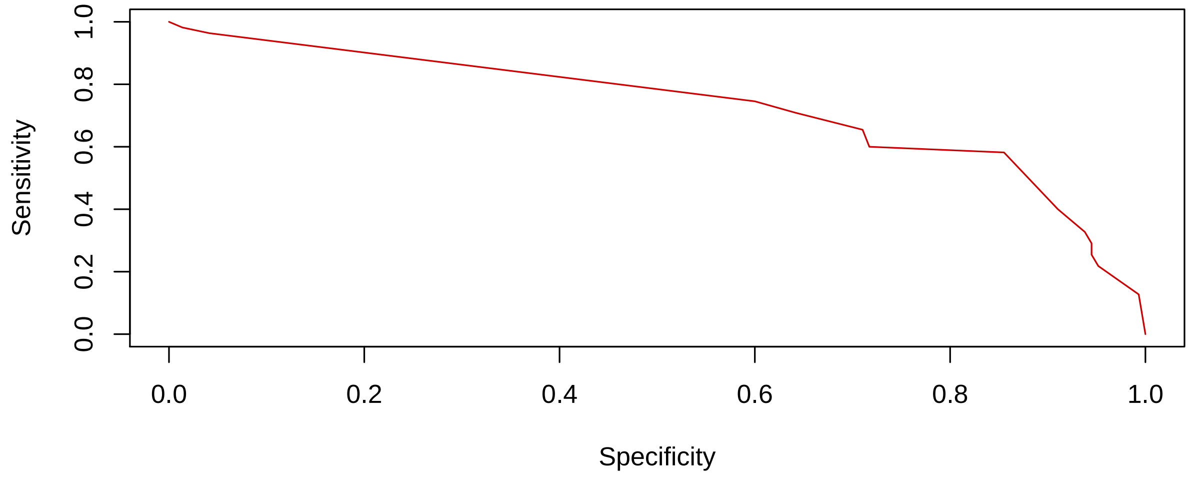


Figure 6: Decision Tree - Sensitivity/Specificity Plot

Thus the overall performance of Decision tree is analysed from above discussed measures. It is giving an accuracy of 78%.

Cross validation is a resampling approach which enables to obtain a more honest error rate estimate or average accuracy of the tree computed on the whole dataset. The cross validation consists to randomly split the data in K folds. We reiterate the following process, by turning the sub-samples: learning the model on (K-1) folds, computing performance measures on the fold number K. We observe that we have the same number of examples in each fold. We applied 10-fold cross validation on 1000 samples of the german credit dataset.

```
library(rpart)
dataset <- read.csv("/home/freestyler/BDA_project/Data/outfile.csv")
n<-nrow(dataset)
K<-10
divide<- n %/% K
set.seed(5)
#The runif() function can be used to simulate n independent uniform random variables.
unirand<-runif(n)
#rank returns the lowest order of position, returns the sample ranks of the values in a vector.
rang<-rank(unirand)
bloc<-(rang - 1)%/%divide +1
bloc<-as.factor(bloc)
print(summary(bloc))

##   1   2   3   4   5   6   7   8   9  10
## 100 100 100 100 100 100 100 100 100 100
```

We can repeat now the learning process and the test process. We collect each error rate,sensitivity, specificity in a vector.Printing Each sub-sample's of sensitivity ,specificity and accuracy:

```
##                  [,1]
## sensitivity 0.9305556
## sensitivity 0.8656716
## sensitivity 0.9027778
## sensitivity 0.9189189
## sensitivity 0.9014085
## sensitivity 0.9104478
## sensitivity 0.9090909
## sensitivity 0.8181818
## sensitivity 0.8589744
## sensitivity 0.8208955
##                  [,1]
## specificity 0.2500000
## specificity 0.3939394
## specificity 0.4642857
## specificity 0.3076923
## specificity 0.4482759
## specificity 0.4242424
## specificity 0.2941176
## specificity 0.4705882
## specificity 0.2272727
## specificity 0.4545455
##      [,1]
## acc1 0.74
## acc1 0.71
## acc1 0.78
## acc1 0.76
## acc1 0.77
## acc1 0.75
## acc1 0.70
## acc1 0.70
## acc1 0.72
## acc1 0.70
```

Because we have the same number of examples in each fold, we can compute unweighted mean. This is the average of each sub-sample's sensitivity:

```
## [1] 0.8836923
```

This is the average of each sub-sample's specificity:

```
## [1] 0.373496
```

This is the cross validation accuracy estimation:

```
## [1] 0.733
```

6.3 Support Vector Machine

Support Vector Machines are an excellent tool for classification. The goal of a support vector machine is to find the optimal separating hyperplane which maximizes the margin of the training data. SVM is a classification algorithm, this will use to predict if something belongs to a particular class or not. Support Vector Machines was worked out for linear two-class classification with margin, where margin means the minimal distance from the separating hyperplane to the closest data points. SVM learning machine seeks for an optimal separating hyperplane, where the margin is maximal. An important and unique feature of this approach is that the solution is based only on those data points, which are at the margin. These points are called support vectors. For instance in Figure [7] we can see that there exists multiple lines that offer a solution to the problem. A line is bad if it passes too close to the points because it will be noise sensitive and it will not generalize correctly. The operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples.

Notation of hyperplane is:

$$f(x) = \beta_0 + \beta^T X. \tag{2}$$

where β is known as the weight vector and β_0 as the bias.

The optimal hyperplane can be represented in an infinite number of different ways by scaling of β and β_0. As a matter of convention, among all the possible representations of the hyperplane, the one chosen is:

$$|\beta_0 + \beta^T X| = 1. \tag{3}$$

Where x symbolizes the training examples closest to the hyperplane.Geometry that gives the distance between a point x and a hyperplane (β, β_0):

$$distance = \frac{\beta_0 + \beta^T X|}{||\beta||} \tag{4}$$

Training set that is closest to the hyperplane is called support vectors. This representation is known as the canonical hyperplane.For the canonical hyperplane, the numerator is equal to one and the distance to the support vectors is:

$$distance_{supportvectors} = \frac{\beta_0 + \beta^T X|}{||\beta||} = \frac{1}{||\beta||}. \tag{5}$$

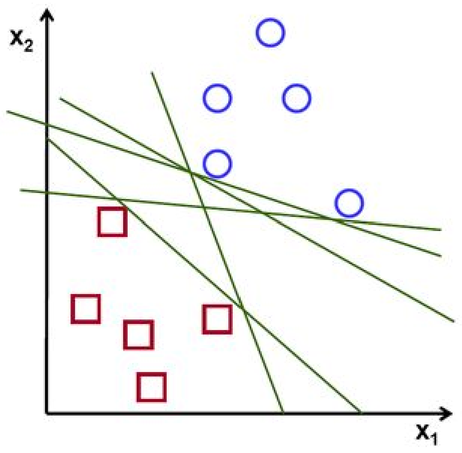


Figure 7: Linearly separable set of 2D-points which belong to one of two classes

Margin M, is twice the distance to the closest examples:

$$M = \frac{2}{||\beta||} \tag{6}$$

The problem of maximizing M is equivalent to the problem of minimizing a function L(β) subject to some constraints. The constraints model the requirement for the hyperplane to classify correctly all the training examples x$_i$.

$$min_{\beta,\beta_0} L(\beta) = \frac{1}{2}||\beta^2||(\beta^T x_i + \beta_0) >= 1\forall_i. \tag{7}$$

So as mentioned earlier the objective is to find the classifier with the best margin of separation. SVM can be implemented with R using the command ksvm() in the R package *kernlab*.

```
library(kernlab, quietly=TRUE)
ksvm <- ksvm(as.factor(R01_credibility) ~ .,
                 data=dataset[train,c(input, target)],
                 kernel="rbfdot",
                 prob.model=TRUE)
```

```
ksvm

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0361190834923943
##
## Number of Support Vectors : 492
##
## Objective Function Value : -380.6035
## Training error : 0.175
## Probability model included.
```

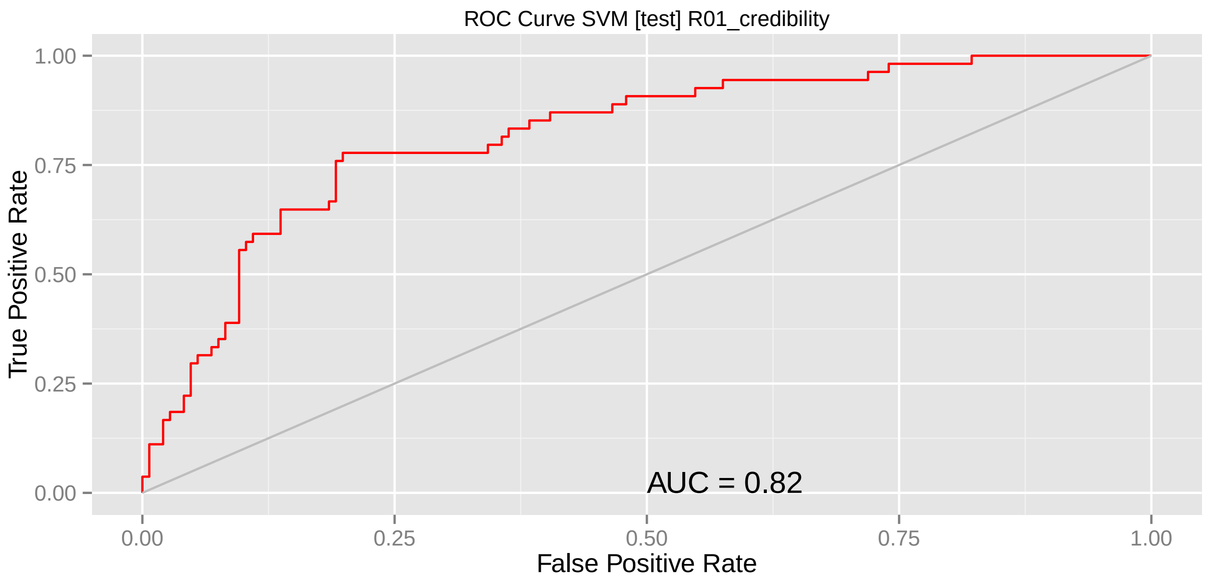


Figure 8: ROC Curve - Support Vector Machine

Then using predict function and test data generate the predicted values. ROC Curve for this model is shown in Figure [8].

```
library(ROCR)
library(ggplot2, quietly=TRUE)
pr <- predict(ksvm, newdata=na.omit(dataset[test, c(input, target)]),
            type="probabilities")[,2]
```

```
print(p)
```

The confusion matrix showing the counts and error proportion is given below:

```
pr <- predict(ksvm, newdata=na.omit(dataset[test, c(input, target)]))
table(dataset[test, c(input, target)]$R01_credibility, pr,
      dnn=c("Actual", "Predicted"))

##       Predicted
## Actual   0   1
##      0 132  14
##      1  27  27
```

```
##       Predicted
## Actual    0    1 Error
##      0 0.66 0.07   0.1
##      1 0.14 0.14   0.5
```

Thus the Support vector machine model giving an accuracy of 79.5%, sensitivity proportion 0.9 but specificity proportion 0.5 only.

We observe that we have the same number of examples in each fold like in previous models we are applying 10 fold cross validation. We can repeat now the learning process and the test process. We collect each measure in a vector.

```r
for(k in 1:K){
ksvm1 <- ksvm(as.factor(R01_credibility) ~., data = dataset[bloc!=k,],probability=TRUE)
pred <- predict(ksvm1,newdata = na.omit(dataset[bloc==k,]))
#confusion matrix for each partition
mc<-table(dataset$R01_credibility[bloc==k],pred)
err<-1.0 - (mc[1,1]+mc[2,2])/sum(mc)
acc1<-1-(err)
a<-mc[1,1]
b<-mc[1,1]+mc[1,2]
sensitivity<-a/b
c<-mc[2,2]
d<-mc[2,2]+mc[2,1]
specificity<-c/d
all.sense<-rbind(all.sense,sensitivity)
all.spese<-rbind(all.spese,specificity)
all.acc1<-rbind(all.acc1,acc1)
}
```

```
##                  [,1]
## sensitivity 0.8888889
## sensitivity 0.9402985
## sensitivity 0.9166667
## sensitivity 0.9324324
## sensitivity 0.9154930
## sensitivity 0.9402985
## sensitivity 0.9242424
## sensitivity 0.8787879
## sensitivity 0.8717949
## sensitivity 0.8805970
##                  [,1]
## specificity 0.3928571
## specificity 0.4545455
## specificity 0.4642857
## specificity 0.3846154
## specificity 0.5862069
## specificity 0.3939394
## specificity 0.3529412
## specificity 0.4705882
## specificity 0.2727273
## specificity 0.3939394
##      [,1]
## acc1 0.75
## acc1 0.78
## acc1 0.79
## acc1 0.79
## acc1 0.82
## acc1 0.76
## acc1 0.73
## acc1 0.74
## acc1 0.74
## acc1 0.72
```

Because we have the same number of examples in each fold, we can compute unweighted mean. This is the average of each sub-sample's sensitivity:

```
## [1] 0.90895
```

This is the average of each sub-sample's specificity:

```
## [1] 0.4166646
```

This is the cross validation accuracy estimation:

```
## [1] 0.762
```

6.4 Neural Network

Neural networks comprise yet another group of nonlinear procedures for prediction and classification. Neural networks are computational analogs of the processes that describe the working of neurons. Neural nets consist of connected input and output layers, each containing numerous units, where the connections among the units of the layers have weights associated with them. A multilayer neural network consists of an input layer of distinct units (usually input units represent the attributes of the case that needs to be classified), one or more hidden layers, and an output layer with several units that represent the class that needs to be predicted. Each layer is made up of several units. The inputs from the units of the first layer are weighted and fed simultaneously to a second layer of neuron-like units, known as the hidden layer. Each of its units takes as its input a weighted sum of the outputs of the previous layer and applies a nonlinear activation function to determine its output. The activation function is usually a logistic function that transforms the output to a number that is between 0 and 1; for this reason, it is often referred to as the squashing function. The output units may already be the actual output of the system (in which case we talk about a single hidden layer), or the outputs of the first hidden layer can be fed as inputs to another hidden layer (in which case we have two hidden layers), and so on if additional hidden layers are involved.

One needs to decide on the number of hidden layers and the number of units in the various layers, as well as the weights that connect the inputs and outputs of these layers. Neural nets are very general, and because of the nonlinear activation functions neural nets are able to approximate most nonlinear functional relationships very well. Of course, the weights in these systems have to be estimated by training the system on actual data. The iterative method of back-propagation can be used to determine the empirical weights that lead to the best fit on a given training sample. For that, one needs to specify a learning parameter that controls the speed of convergence of this iterative estimation method.

One disadvantage of neural nets is that the approximating models relating inputs and outputs are purely "black box" models, and they provide very little insight into what these models really do. Also, the user of neural nets must make many modeling assumptions, such as the number of hidden layers and the number of units in each hidden layer, and usually there is little guidance on how to do this. It takes considerable experience to find the most appropriate representation. The package *nnet* is used here[7].

The Neural network model using nnet package is build:

```r
library(nnet, quietly=TRUE)

set.seed(199)
nnet <- nnet(as.factor(R01_credibility) ~ .,
                 data=dataset[sample,c(input, target)],
                 size=7, skip=TRUE, MaxNWts=10000, trace=FALSE, maxit=100)
```

The result of the modelling is :

```
## A 18-7-1 network with 159 weights.
## Output: as.factor(R01_credibility).
## Sum of Squares Residuals: 41.3214.
```

```
print(summary(nnet))

## Neural Network build options: skip-layer connections; entropy fitting.
##
## In the following table:
##    b  represents the bias associated with a node
##    h1 represents hidden layer node 1
##    i1 represents input node 1 (i.e., input variable 1)
##    o  represents the output node
##
## Weights for node h1:
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1
##  -17.61    8.21   39.25   -3.06  -22.45    3.17    2.21   -1.37    1.08
##  i9->h1 i10->h1 i11->h1 i12->h1 i13->h1 i14->h1 i15->h1 i16->h1 i17->h1
##    0.21   11.62    5.64  -18.04    2.67  -17.93   -0.42    5.55   -3.50
## i18->h1
##  -10.84
##
## Weights for node h2:
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2
##    3.69   -3.82    7.44   -4.46    3.67    4.50    5.36    1.85   16.08
##  i9->h2 i10->h2 i11->h2 i12->h2 i13->h2 i14->h2 i15->h2 i16->h2 i17->h2
##    7.22    8.77   -9.37   -3.38   20.61   -0.76   20.55    0.54   17.97
## i18->h2
##   -4.14
##
## Weights for node h3:
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3
##    4.03   29.54    5.60    1.67  -14.70   10.29    9.72   14.08   -4.25
##  i9->h3 i10->h3 i11->h3 i12->h3 i13->h3 i14->h3 i15->h3 i16->h3 i17->h3
##   -4.83    0.89    0.09  -23.48   10.57   -7.15   -8.43   11.59    0.71
## i18->h3
##    2.54
##
## Weights for node h4:
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4
##   11.70    4.36   12.76   11.72    6.57   15.23    3.50   -7.24   39.54
##  i9->h4 i10->h4 i11->h4 i12->h4 i13->h4 i14->h4 i15->h4 i16->h4 i17->h4
##  -11.78   -7.63   -9.42   -3.56   -0.60  -14.00    5.71    8.61   12.65
## i18->h4
##   -8.92
##
## Weights for node h5:
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5  i8->h5
##    3.17   -0.82   23.11    8.37    2.56    5.12  -14.45   -0.15   31.72
##  i9->h5 i10->h5 i11->h5 i12->h5 i13->h5 i14->h5 i15->h5 i16->h5 i17->h5
##   -7.03  -10.73    1.39   -9.58    4.12  -42.00    7.66   11.44    6.00
## i18->h5
##   -7.19
##
## Weights for node h6:
##   b->h6  i1->h6  i2->h6  i3->h6  i4->h6  i5->h6  i6->h6  i7->h6  i8->h6
##   -8.26   -8.20    6.96   23.23   -2.50    6.83    7.10  -20.09   -7.52
##  i9->h6 i10->h6 i11->h6 i12->h6 i13->h6 i14->h6 i15->h6 i16->h6 i17->h6
##   20.61   -7.99    2.24   -7.59   20.47  -16.77    4.87  -16.66   13.21
## i18->h6
##  -11.23
##
## Weights for node h7:
##   b->h7  i1->h7  i2->h7  i3->h7  i4->h7  i5->h7  i6->h7  i7->h7  i8->h7
##   13.33  -37.31   30.45   -8.37   -7.88    9.32    3.32  -28.75    8.92
##  i9->h7 i10->h7 i11->h7 i12->h7 i13->h7 i14->h7 i15->h7 i16->h7 i17->h7
##   12.83   10.51  -15.35  -13.05    3.55   -1.41    7.22    4.05    9.33
## i18->h7
##   14.62
##
## Weights for node o:
##   b->o  h1->o  h2->o  h3->o  h4->o  h5->o  h6->o  h7->o  i1->o  i2->o
##   8.19  -7.13  -7.55  -7.37 -11.69  14.46   6.41  -8.55   1.85   2.14
##  i3->o  i4->o  i5->o  i6->o  i7->o  i8->o  i9->o i10->o i11->o i12->o
##  -1.33  -2.03   2.18  -0.26  -1.47   1.31  -1.39   1.64  -2.97  -2.59
## i13->o i14->o i15->o i16->o i17->o i18->o
##   1.85   0.16   0.34   1.75   0.56  -0.08
```

Now evaluating the model performance. At first take the response from the model then generate the Confusion matrix for the Neural Network model.

```
pr <- predict(nnet, newdata=dataset[test, c(input, target)], type="class")
table(dataset[test, c(input, target)]$R01_credibility, pr,
      dnn=c("Actual", "Predicted"))

##       Predicted
## Actual   0   1
##      0 110  35
##      1  27  28
```

```
per <- pcme(dataset[test, c(input, target)]$R01_credibility, pr)
round(per, 2)

##       Predicted
## Actual    0    1 Error
##      0 0.55 0.18  0.24
##      1 0.14 0.14  0.49
```

The ROC curve generated for the model is in Figure [9].

```
print(p)
```

The model was giving an accuracy percentage of 69 only.
The 10 fold cross validation with this model results:

```
##   1   2   3   4   5   6   7   8   9  10
## 100 100 100 100 100 100 100 100 100 100
```

Repeating the learning process and the test process. Then collect each sensitivity,specificity, and accuracy in a vector. Printing each sub-sample's sensitivity, specificity and accuracy:

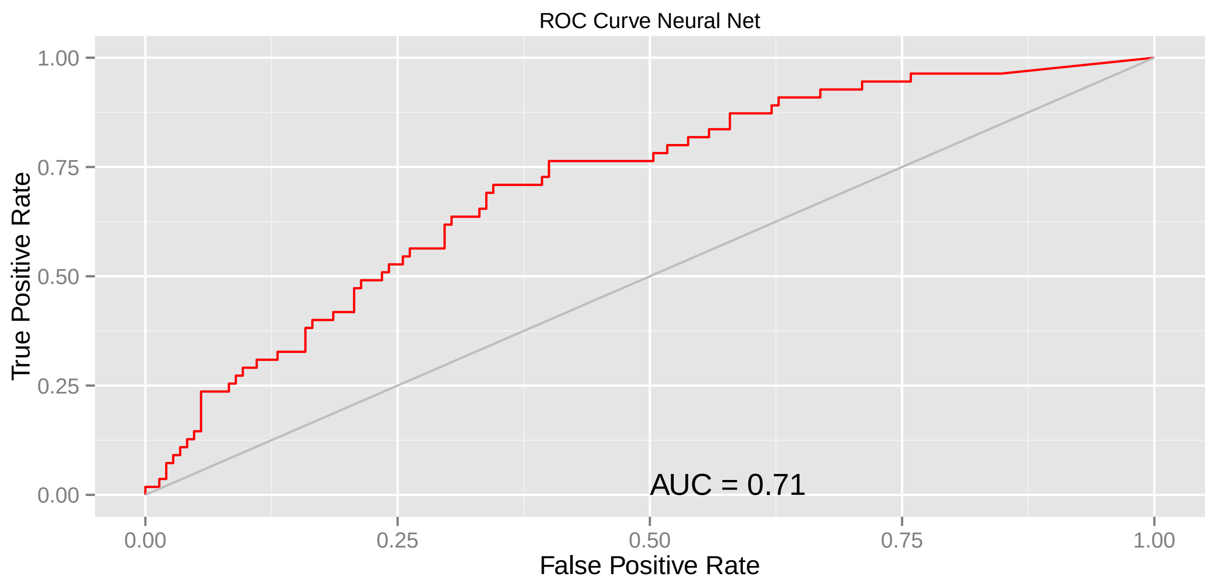


Figure 9: ROC Curve - Neural Networks

```
##                   [,1]
## sensitivity 0.8194444
## sensitivity 0.8059701
## sensitivity 0.8055556
## sensitivity 0.7702703
## sensitivity 0.7887324
## sensitivity 0.8507463
## sensitivity 0.7727273
## sensitivity 0.8030303
## sensitivity 0.7564103
## sensitivity 0.7761194
##                   [,1]
## specificity 0.4642857
## specificity 0.3939394
## specificity 0.3571429
## specificity 0.5000000
## specificity 0.5862069
## specificity 0.4545455
## specificity 0.5294118
## specificity 0.4117647
## specificity 0.4090909
## specificity 0.3636364
##      [,1]
## acc1 0.72
## acc1 0.67
## acc1 0.68
## acc1 0.70
## acc1 0.73
## acc1 0.72
## acc1 0.69
## acc1 0.67
## acc1 0.68
## acc1 0.64
```

Because we have the same number of examples in each fold, we can compute unweighted mean. This is the average of each sub-sample's sensitivity:

```
## [1] 0.7949006
```

This is the average of each sub-sample's specificity:

```
## [1] 0.4470024
```

This is the cross validation average accuracy:

```
## [1] 0.69
```

6.4.1 PCA with Neural network

In this model we are doing PCA in the dataset and use it in the neural network model. For that purpose library *caret* we are using.

```
dataset <- read.csv("/home/freestyler/outfile.csv", na.strings=c(".", "NA", "", "?"), strip.white=TRUE,
```

```
set.seed(crv$seed)
nobs <- nrow(dataset) # 1000 observations
sample <- train <- sample(nrow(dataset), 0.7*nobs) # 700 observations
validate <- sample(setdiff(seq_len(nrow(dataset)), train), 0.15*nobs) # 150 observations
test <- setdiff(setdiff(seq_len(nrow(dataset)), train), validate) # 150 observations
```

```
set.seed(42)
nobs <- nrow(dataset) # 1000 observations
train <- sample(nrow(dataset), 0.8*nobs) # 800 observations
validate <- NULL
test <- setdiff(setdiff(seq_len(nrow(dataset)), train), validate) # 200 observations
```

```
xtrain <- dataset[train,]
xnew <- dataset[-train,]
ytrain <- dataset$R01_credibility[train]
ynew <- dataset$R01_credibility[-train]
```

The function first will run principal component analysis on the data. The cumulative percentage of variance is computed for each principal component. The function uses the thresh argument to determine how many components must be retained to capture this amount of variance in the predictors. Here we used threshold value as 0.95, the model description is given below.

```
library(caret)
modelFit <- pcaNNet(xtrain[, 1:20], ytrain, thresh=0.95,size = 5, linout = TRUE, trace = FALSE)
modelFit
```

```
## Neural Network Model with PCA Pre-Processing
##
## Created from 800 samples and 20 variables
## PCA needed 18 components to capture 95 percent of the variance
##
## a 18-5-1 network with 101 weights
## options were - linear output units
```

The principal components are then used in a neural network model. When predicting samples, the new data are similarly transformed using the information from the PCA analysis on the training data and then predicted[8].

```
pr<-predict(modelFit, xnew[, 1:20])
```

```
table(ynew, pr,dnn=c("Actual", "Predicted"))

##        Predicted
## Actual   0   1
##      0 110  35
##      1  26  29
```

From the confusion matrix generated above we can conclude that with PCA -neural network model also not giving better performance, so we are ignoring this model.

6.5 k Nearest Neighbour

One useful method for making the classification relies on the k -nearest neighbor algorithm. The k -nearest neighbor (knn) algorithm classifies new objects according to the outcome of the closest object or the outcomes of several closest objects in the feature space of the training set. The k -nearest neighbor algorithm is among the simplest of all machine learning algorithms: an object is classified by a majority vote of its neighbors, with the new object being assigned to the class that is most common among its k nearest neighbors (k is a positive integer, and typically small) [4].

The neighbors are taken from a set of objects for which the correct classification is known. This can be thought of as the training set for the algorithm, although no explicit training step is required. The training samples are vectors in a multi-dimensional feature space, each with a specified class label. The training phase of the algorithm only consists of storing the feature vectors and class labels of the training samples. In the classification phase, k is a user-defined constant, and a new object with given features (sometimes also referred to as a query or test point) is classified by assigning to it the label that is most frequent among the k training samples nearest to that new object.

Usually, with continuous features (such as income in multiples of 1000, or age in years), the Euclidean distance is used as the distance metric. Assume that the 'd' features for case 1 are given by

$$x_{11}, x_{12}, ..., x_{1d} \tag{8}$$

and that the features for case 2 are given by

$$x_{21}, x_{22}, ..., x_{2d} \tag{9}$$

. Then the Euclidean distance between cases 1 and 2 is defined as

$$\sqrt{(x_{11} - x_{21})^2 + (x_{12} - x_{22})^2 + ... + (x_{1d} - x_{2d})^2} \tag{10}$$

. If there is only one feature, the Euclidean distance is the absolute value of the difference. It is usually recommended to standardize the feature variables if their units are quite different. Otherwise, more weight would be given to feature variables with larger units. For **standardization** we converted the data set to a new one with mean zero and standard deviation (stdev) one using *rattle* Type: Rescale, Normalize: Recenter from Transform option.

A drawback of the basic "majority voting" classification of the k -nearest neighbor algorithm is that classes with more frequent outcomes tend to dominate the classification of the new object. Because of their large numbers, they tend to show up more often among the k -nearest neighbors when neighbors are computed. So here we are doing oversampling as mentioned in data preprocessing part in training data and then applying algorithm.

```
set.seed(42)
nobs <- nrow(dataset) # 1000 observations
train <- sample(nrow(dataset), 0.8*nobs) # 800 observations
validate <- NULL
test <- setdiff(setdiff(seq_len(nrow(dataset)), train), validate) # 200 observations
```

```
xtrain <- dataset[train,]
xnew <- dataset[-train,]
ytrain <- dataset$R01_credibility[train]
ynew <- dataset$R01_credibility[-train]

train_dataset<-read.csv("/home/freestyler/new_sample.csv")
```

```
xtrain <- train_dataset
ytrain <- train_dataset$R01_credibility

table(ytrain)#Proportion of classes in training set

## ytrain
##   0   1
## 555 555

table(ynew)#proportion of classes in test set

## ynew
##   0   1
## 145  55
```

k nearest neighbour algorithm applied, after analysing with some values of k, we are getting better solution for k=7. The confusion matrix showing the counts are:

```
library(class)
nearest3 <- knn(train=xtrain, test=xnew, cl=ytrain, k=7)

table(dataset[test, c(input, target)]$R01_credibility, nearest3,
      dnn=c("Actual", "Predicted"))

##       Predicted
## Actual   0   1
##      0 119  26
##      1  18  37
```

The erro proportion matrix is:

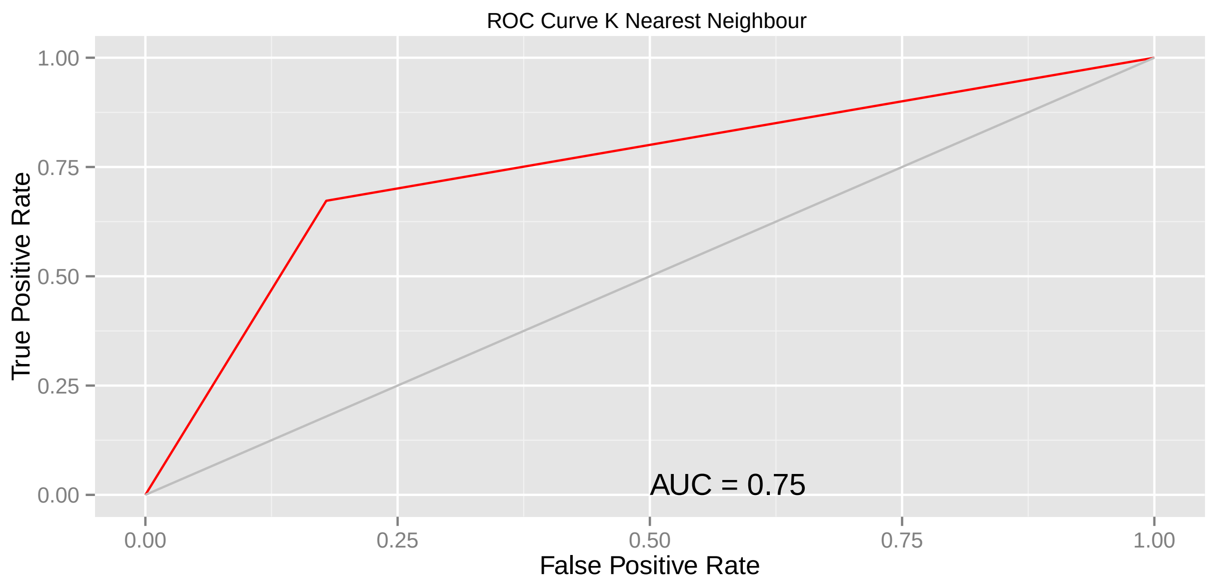


Figure 10: ROC Curve - K Nearest Neighbour

```
pcme(na.omit(dataset[test, c(input, target)])$R01_credibility, nearest3)

##       Predicted
## Actual    0    1 Error
##      0 0.60 0.13  0.18
##      1 0.09 0.18  0.33
```

An ROC (receiver operator characteristic) curve shown in Figure [10] compares the false positive rate to the true positive rate. We can access the trade off the number of observations that are incorrectly classified as positives against the number of observations that are correctly classified as positives.

```
print(p)
```

The Proportion of correct classification is:

```
pcorrn3=100*sum(ynew==nearest3)/100
pcorrn3

## [1] 157
```

We are applying Leave one out cross validation method here and the average accuracy obtained is 80.91%:

```
psum<- 0
pcorr=dim(15)
for (k in 1:15) {
  pred=knn.cv(dataset,cl=dataset$R01_credibility,k)
  pcorr[k]=100*sum(dataset$R01_credibility==pred)/1000
  psum<- psum+pcorr[k]
}
pcorr
```

```
##  [1] 76.6 77.4 80.3 79.0 82.0 81.4 82.4 82.1 82.6 82.0 81.8 81.2 81.2 81.3
## [15] 82.3

psum/15

## [1] 80.90667
```

Thus all the above mentioned measures help to analyse the model well, and it was giving an accuracy of 78 %, with specificity proportion of 0.67 and sensitivity of 0.82.
10 fold cross validation also applied in this model, each sub-sample's sensitivity ,specificity and accuracy is computed:

```
##   1   2   3   4   5   6   7   8   9  10
## 100 100 100 100 100 100 100 100 100 100
##                  [,1]
## sensitivity 0.9722222
## sensitivity 0.9850746
## sensitivity 0.9027778
## sensitivity 0.9594595
## sensitivity 0.9859155
## sensitivity 0.9552239
## sensitivity 0.9545455
## sensitivity 0.9696970
## sensitivity 0.9358974
## sensitivity 0.9850746
##                  [,1]
## specificity 0.4285714
## specificity 0.5757576
## specificity 0.6071429
## specificity 0.3076923
## specificity 0.6206897
## specificity 0.4848485
## specificity 0.5882353
## specificity 0.5882353
## specificity 0.2272727
## specificity 0.5151515
##      [,1]
## acc1 0.82
## acc1 0.85
## acc1 0.82
## acc1 0.79
## acc1 0.88
## acc1 0.80
## acc1 0.83
## acc1 0.84
## acc1 0.78
## acc1 0.83
```

The average sensitivity is:

```
sens.cv<-mean(all.sense)
print(sens.cv)

## [1] 0.9605888
```

The average specificity is :

```
spec.cv<-mean(all.spese)
print(spec.cv)

## [1] 0.4943597
```

And the average accuracy using 10 fold cross validation is :

```
acc1.cv<-mean(all.acc1)
print(acc1.cv)

## [1] 0.824
```

6.6 Random Forest

A random forest is an ensemble (i.e., a collection) of un-pruned decision trees. Ensemble models are often robust to variance and bias. Random forests are often used when we have large training datasets and particularly a very large number of input variables (hundreds or even thousands of input variables). The algorithm is efficient with respect to a large number of variables since it repeatedly subsets the variables available. We can make use of this model to view the relative importance of each variable also. A random forest model is typically made up of tens or hundreds of decision trees [5].

The *randomForest* package provides the 'randomForest' function. The code snippet building random forest model is given below:

```
library(randomForest, quietly=TRUE)
set.seed(crv$seed)
rf <- randomForest::randomForest(as.factor(R01_credibility) ~ .,
                                 data=dataset[sample,c(input, target)],
                                 ntree=500,
                                 mtry=4,
                                 importance=TRUE,
                                 na.action=randomForest::na.roughfix,
                                 replace=FALSE)
```

The textual output of random forest model is:

```
print(rf)

##
## Call:
##  randomForest(formula = as.factor(R01_credibility) ~ ., data = dataset[sample,      c(input, target)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 24.12%
## Confusion matrix:
##     0   1 class.error
## 0 494  61   0.1099099
## 1 132 113   0.5387755
```

The *pROC* package implements various AUC functions.

```
# Calculate the Area Under the Curve (AUC).

pROC::roc(rf$y, as.numeric(rf$predicted))

##
## Call:
## roc.default(response = rf$y, predictor = as.numeric(rf$predicted))
##
## Data: as.numeric(rf$predicted) in 555 controls (rf$y 0) < 245 cases (rf$y 1).
## Area under the curve: 0.6757

# Calculate the AUC Confidence Interval.

pROC::ci.auc(rf$y, as.numeric(rf$predicted))

## 95% CI: 0.6418-0.7095 (DeLong)
```

From this model we will get the importance of variables:

```
rn <- round(randomForest::importance(rf), 2)
rn[order(rn[,3], decreasing=TRUE),]

##                       0     1 MeanDecreaseAccuracy MeanDecreaseGini
## TNM_check_status 17.94 33.19                31.84            25.38
## duration         17.04  7.83                19.05            22.11
## TNM_history      10.94 11.81                15.44            14.53
## credit           12.64  3.34                12.72            29.98
## TNM_guarantor    12.03  3.45                11.65             4.40
## TNM_bonds         0.66 10.54                 6.56             9.96
## TNM_purpose       4.86  2.92                 5.74            14.58
## TNM_install       6.68 -0.03                 5.64             6.68
## TNM_property      8.34 -2.28                 5.52            10.16
## age               4.60  2.55                 5.31            23.18
## nocredit          4.33 -0.64                 3.25             4.59
## TNM_ph            2.39  2.21                 3.17             4.57
## TNM_nri           1.86  2.31                 2.84             1.10
## TNM_jobex         1.10  3.06                 2.77            11.22
## TNM_s_status      1.24  2.84                 2.77             8.48
## TNM_job           3.12 -0.32                 2.49             7.36
## rate              2.73 -0.33                 2.00             9.32
## TNM_house         3.41 -2.36                 1.55             5.48
```

Random Forest gives less importance to the ignored variables *liable* and *residance* by running in original dataset, which makes the assumption made in preprocessing part strong. Figure [11] shows the relative importance of the variables:

```
randomForest::varImpPlot(rf, main="")
title(main="Variable Importance Random Forest",
      sub=paste("Rattle", format(Sys.time(), "%Y-%b-%d %H:%M:%S"), Sys.info()["user"]))
```

Gini importance [6] : The variable importance metric refers to the Gini used for asserting model performance which is not closely related to AUC. Every time a split of a node is made on variable m the gini impurity criterion for the two descendent nodes is less than the parent node. Adding up the gini decreases for each individual variable over all trees in the forest gives a fast variable importance that is often very

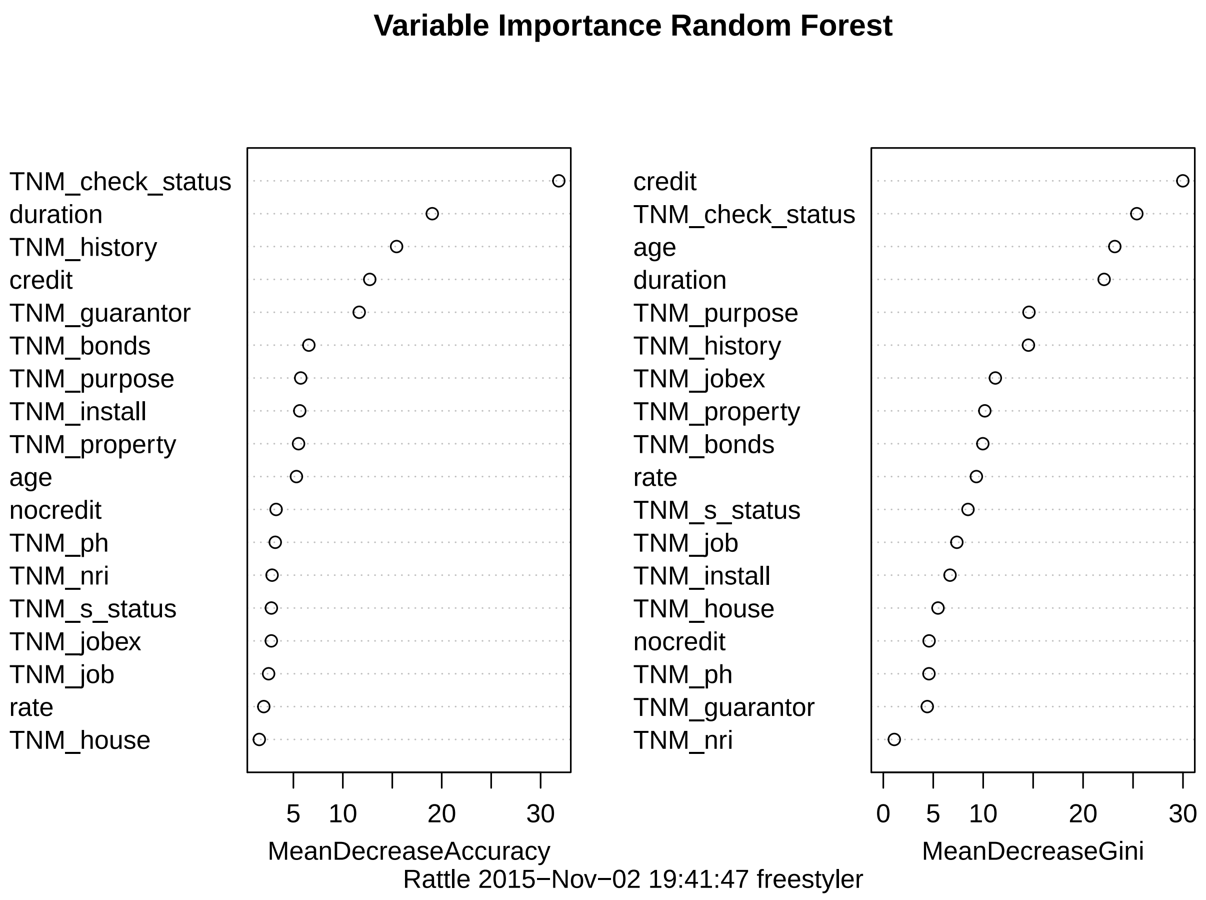


Figure 11: Random Forest - Variable Importance

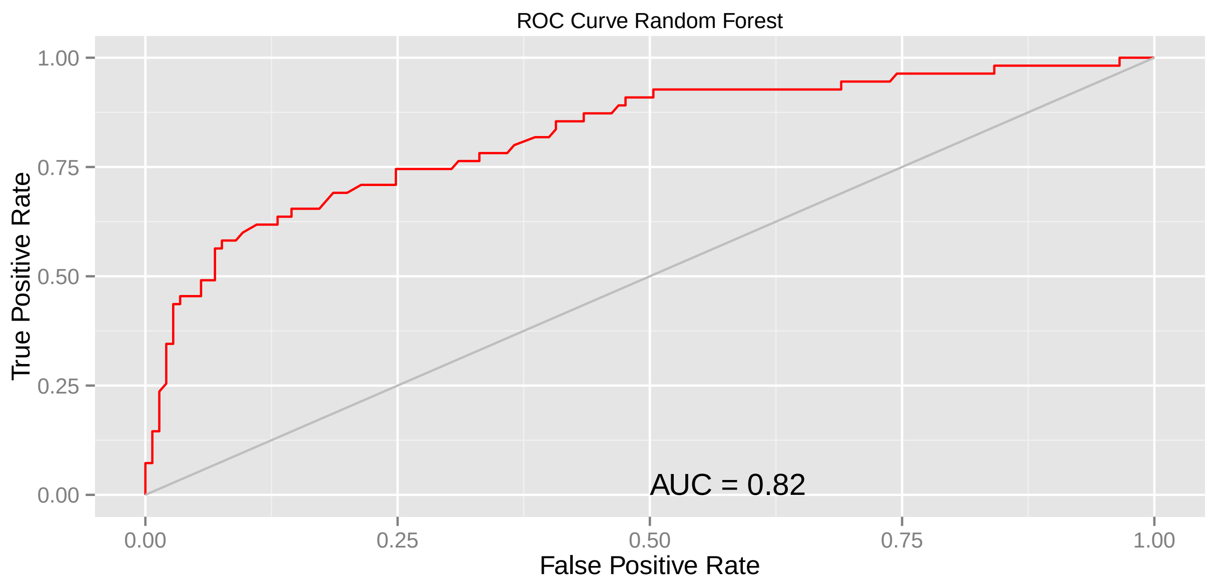


Figure 12: Random Forest - ROC Curve

consistent with the permutation importance measure.
Each tree can be displayed using below script ,where 1 is the tree number :

```
printRandomForests(rf, 1)
```

An ROC (receiver operator characteristic) curve shown in Figure [12] compares the false positive rate to the true positive rate. We can access the trade off the number of observations that are incorrectly classified as positives against the number of observations that are correctly classified as positives.

```
print(p)
```

The confusion matrix showing counts and proportions shown below:

```
pr <- predict(rf, newdata=na.omit(dataset[test, c(input, target)]))

table(na.omit(dataset[test, c(input, target)])$R01_credibility, pr,
      dnn=c("Actual", "Predicted"))

##       Predicted
## Actual   0   1
##      0 137   8
##      1  28  27
```

```
pcme(na.omit(dataset[test, c(input, target)])$R01_credibility, pr)

##       Predicted
## Actual    0    1 Error
##      0 0.68 0.04  0.06
##      1 0.14 0.14  0.51
```

From the above computations, this model can also be used to analyse the variable importance. Random Forest gives an accuracy of 82%, but specificity proportion 0.49 only.
The 10 fold cross validation computation as follows:

```
##   1   2   3   4   5   6   7   8   9  10
## 100 100 100 100 100 100 100 100 100 100
```

Each sub-sample's of sensitivity ,specificity and error rate:

```
##                  [,1]
## sensitivity 0.9166667
## sensitivity 0.9104478
## sensitivity 0.9027778
## sensitivity 0.9189189
## sensitivity 0.9577465
## sensitivity 0.9402985
## sensitivity 0.9090909
## sensitivity 0.8939394
## sensitivity 0.8974359
## sensitivity 0.9104478
##                  [,1]
## specificity 0.5000000
## specificity 0.4545455
## specificity 0.5000000
## specificity 0.3076923
## specificity 0.4482759
## specificity 0.3939394
## specificity 0.3823529
## specificity 0.4117647
## specificity 0.3181818
## specificity 0.4242424
##      [,1]
## acc1 0.80
## acc1 0.76
## acc1 0.79
## acc1 0.76
## acc1 0.81
## acc1 0.76
## acc1 0.73
## acc1 0.73
## acc1 0.77
## acc1 0.75
```

Because we have the same number of examples in each fold, we can compute unweighted mean. This is the average of sub-sample's sensitivity:

```
## [1] 0.915777
```

This is the average of sub-sample's specificity:

```
## [1] 0.4140995
```

This is the cross validation average accuracy proportion:

```
## [1] 0.766
```

7 Result and Discussions

In this section we are discussing about the results we found from exploratory data analysis onwards to each model's evaluation stage.

In data analysis stage we conducted chi squared statistics to categorical data but could not reach any conclusion that impact on the independent variable. In t-statistics p value of two varailbes named "liable", and "residence" were varying drastically hence we assumed that it could not contribute well in output prediction. In addition to that using random forest the obtained variable significance plot also shows those two variables as less significant ones'. The principal componenet analysis was done, and 18 components were giving above 95 percent of the variance. Hence we made PCA Neural network model and analysed its performance as mentioned in section 6.4.1.

The ROC graph is a technique for visualizing, organizing and selecting classifiers based on their performance. Comparison of two or more ROC curves is usually based on a comparison of the area measures. The ROC curves of all models is shown in Figure[13] and their area under curve value is showin Table[3].

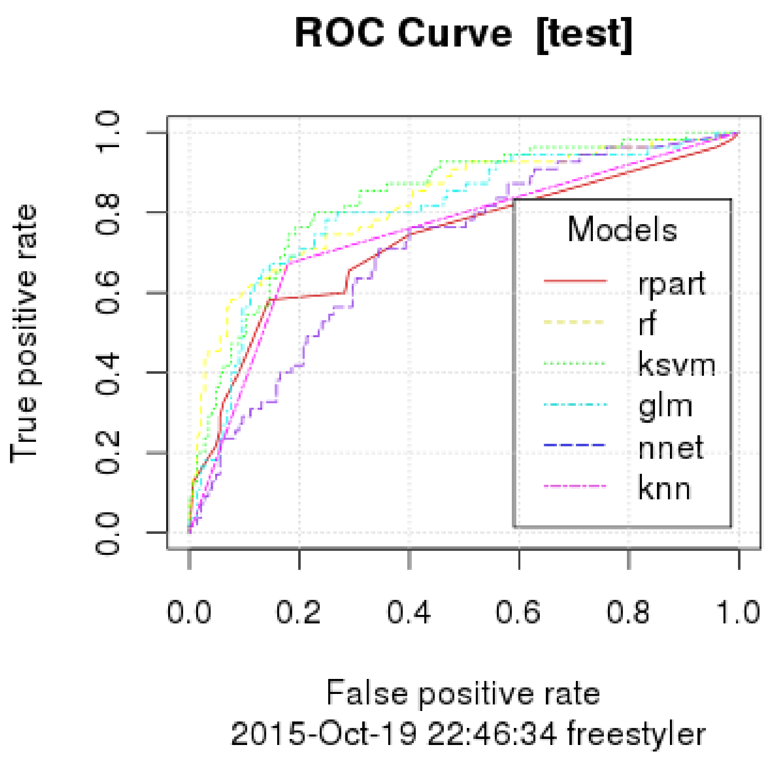


Figure 13: ROC Curve comparison of models

Table[4] shows each model's sensitivity, specificity and accuracy without 10 fold cross validation and Table[5] with 10 fold cross validation.

Table 3: Comparison of models

Model No	Model	Area Under Curve
1	Logistic Regression	0.8
2	Decision Tree	0.73
3	Support Vector Machine	0.82
4	Neural Networks	0.71
5	K Nearest Neighbour	0.75
6	Random Forest	0.82

Table 4: Performance measures

Model No	Model	Sensitivity	Specificity	Accuracy
1	Logistic Regression	0.9	0.51	0.795
2	Decision Tree	0.86	0.58	0.78
3	Support Vector Machine	0.90	0.50	0.795
4	Neural Networks	0.76	0.51	0.69
5	K Nearest Neighbour	0.82	0.673	0.78
6	Random Forest	0.945	0.491	0.82

8 Conclusion

This report illustrated the study of classification task in german credit dataset. Varoius data mining techniques applied to the dataset, after pre processing techniques. Each model analysed with its performance measures. In section 7 all the models compared with and without 10 fold cross validation as well with ROC curves. Based on the Area Under curve measure from Table[3] we can say that Logistic regression, Support Vector Machine and Random Forest have almost near AUC value and are better than other models.

The dataset information, [2] mentioned ”It is worse to class a customer as good when they are bad (5), than it is to class a customer as bad when they are good (1)”. Based on that while analysing the specificity of each model with out 10 FCV, k nearest neighbour giving more value, but among Logistic regression, Support Vector Machine and Random Forest, Logistic Regression has more specificity value with and without 10 FCV.

Based on ROC and accuracy Random Forest model performed well. We can ignore the models Neural network and Decision Tree based on their poor performance compared to other models.

Table 5: Performance of measures with 10 FCV

Model No	Model	Sensitivity	Specificity	Accuracy
1	Logistic Regression	0.88	0.46	0.76
2	Decision Tree	0.88	0.37	0.73
3	Support Vector Machine	0.90	0.42	0.76
4	Neural Networks	0.79	0.45	0.69
5	K Nearest Neighbour	0.96	0.49	0.82
6	Random Forest	0.91	0.41	0.77

References

[1] Andrea Dal Pozzolo, Olivier Caelen and Gianluca Bontempi, *unbalanced: Racing for Unbalanced Methods Selection*, R package version 2.0, http://cran.r-project.org/web/packages/unbalanced, 2015

[2] M. Lichman , *UCI Machine Learning Repository.* University of California, Irvine, School of Information and Computer Sciences http://archive.ics.uci.edu/ml 2013.

[3] Institute for Digital Research and Education *What statistical analysis should I use*, UCLA:Statistical Consulting Group. http://www.ats.ucla.edu/stat/mult_pkg/whatstat/., (accessed October 07, 2015)

[4] Johannes Ledolter, *DATA MINING AND BUSINESS ANALYTICS WITH R*, Department of Management Sciences, Tippie College of Business, University of Iowa. 2013.

[5] Graham J Williams, *Rattle: A Data Mining GUI for R*, The R Journal,Vol 1,Pages 45-55 http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf, Dec, 2009.

[6] Leo Breiman, Adele Cutler and Andy Liaw, Matthew Wiener, *Breiman and Cutler's Random Forests for Classification and Regression*, Version 4.6-12, Repository CRAN, http://www.stat.berkeley.edu/ breiman/RandomForests/, 2015.

[7] Brian Ripley ,William Venables, *Feed-Forward Neural Networks and Multinomial Log-Linear Models*, Package nnet,Version 7.3-11, Repository CRAN, https://cran.r-project.org/web/packages/nnet/, 2015.

[8] Brian Ripley ,William Venables, *Neural Networks with a Principal Component Step* , Package caret, Version: 6.0-24, http://www.inside-r.org/packages/cran/caret/docs/pcaNNet.default, (accessed October 31, 2015).

Appendix

Attribute Information:
Attribute 1: (qualitative)
Renamed as TNM_check_status
Recoded to numeric values [1,2,3,4] respectively.
Status of existing checking account
A11 : ... 0 DM
A12 : 0 $\leq$ ... 200 DM
A13 : ... $\geq$ 200 DM / salary assignments for at least 1 year
A14 : no checking account

Attribute 2: (numerical)
Duration in month

Attribute 3: (qualitative)
Renamed as $TNM\_history$
Recoded to numeric values [1,2,3,4,5] respectively.
Credit history
A30 : no credits taken/ all credits paid back duly
A31 : all credits at this bank paid back duly
A32 : existing credits paid back duly till now
A33 : delay in paying off in the past
A34 : critical account/ other credits existing (not at this bank)

Attribute 4: (qualitative)
Renamed as $TNM\_purpose$
Recoded to numeric values [1 to 11] respectively.
Purpose
A40 : car (new)
A41 : car (used)
A42 : furniture/equipment
A43 : radio/television
A44 : domestic appliances
A45 : repairs
A46 : education
A47 : (vacation - does not exist)
A48 : retraining
A49 : business
A410 : others

Attribute 5: (numerical)
Credit amount

Attibute 6: (qualitative)
Renamed as TNM_bonds
Recoded to numeric values [1 to 5] respectively.
Savings account/bonds
A61 : ... 100 DM A62 : 100 $\leq$ ... 500 DM
A63 : 500 $\leq$ ... 1000 DM
A64 : .. $\geq$ 1000 DM
A65 : unknown/ no savings account

Attribute 7: (qualitative)

Renamed as TNM_jobex
Recoded to numeric values [1 to 5] respectively.
Present employment since
A71 : unemployed
A72 : ... 1 year
A73 : 1 $\leq$ ... 4 years
A74 : 4 $\leq$ ... 7 years
A75 : .. $\geq$ 7 years

Attribute 8: (numerical)
Installment rate in percentage of disposable income

Attribute 9: (qualitative)
Renamed as TNM_s_status
Recoded to numeric values [1 to 5] respectively.
Personal status and sex
A91 : male : divorced/separated
A92 : female : divorced/separated/married
A93 : male : single
A94 : male : married/widowed
A95 : female : single

Attribute 10: (qualitative)
Renamed as $TNM\_guarantor$
Recoded to numeric values [1 to 3] respectively.
Other debtors / guarantors
A101 : none
A102 : co-applicant
A103 : guarantor

Attribute 11: (numerical)
Present residence since

Attribute 12: (qualitative)
Renamed as $TNM\_property$
Recoded to numeric values [1 to 4] respectively.
Property
A121 : real estate
A122 : if not A121 : building society savings agreement/ life insurance
A123 : if not A121/A122 : car or other, not in attribute 6
A124 : unknown / no property

Attribute 13: (numerical)
Age in years

Attribute 14: (qualitative)
Renamed as $TNM\_install$
Recoded to numeric values [1 to 3] respectively.
Other installment plans
A141 : bank
A142 : stores
A143 : none

Attribute 15: (qualitative)

Renamed as TNM_house
Recoded to numeric values [1 to 3] respectively.
Housing
A151 : rent
A152 : own
A153 : for free

Attribute 16: (numerical)
Number of existing credits at this bank

Attribute 17: (qualitative)
Renamed as TNM_job
Recoded to numeric values [1 to 4] respectively.
Job
A171 : unemployed/ unskilled - non-resident
A172 : unskilled - resident
A173 : skilled employee / official
A174 : management/ self-employed/
highly qualified employee/ officer

Attribute 18: (numerical)
Number of people being liable to provide maintenance for

Attribute 19: (qualitative)
Renamed as TNM_ph
Recoded to numeric values [1 - 2] respectively.
Telephone
A191 : none
A192 : yes, registered under the customers name

Attribute 20: (qualitative)
Renamed as TNM_nri
Recoded to numeric values [1 -2] respectively.
foreign worker
A201 : yes
A202 : no

Attribute 21: (Target variable)
Renamed as $R01\_credibility$
Recoded to numeric values [0-1] respectively.
Labels
0 : Good
1 : Bad