

Cust_type prediction

- The objective is to develop predictive model to detect whether the customer will make default or not based on the data

Dataset information:

Data Dictionary:

Files:

- cust_type_prediction - Sample Users.csv

Data Fields:

- UserId
- bank_balance_m1
- bank_balance_m2
- total_withdrawals_m1
- total_withdrawals_m2
- total_deposit_m1
- total_deposit_m2
- expenses
- disposable_income_m1
- disposable_income_m2
- total_withdrawals_m3
- num_credit_card
- total_deposit_m3
- bank_balance_m3
- disposable_income_m3
- bank_balance_m4
- bank_balance_m5
- obligations_m1
- bank_balance_m6
- obligations_m2
- num_loans
- obligations_m3
- car_ownership
- bike_ownership
- cust_type

The class variable is cust_type, which is coded as 0 for good(No defaults) and 1 for bad(defaulted at least once)

Dimensions of Dataset

```
> dim(data)
```

```
[1] 2000 25
```

More information about variables in data:

```
> str(data)
```

```
'data.frame': 2000 obs. of 25 variables:
```

```
$ UserId : Factor w/ 2000 levels "u1","u10","u100",...: 1 1112 1224 1335 1446 1557 1668 1779 1890 2 ...
```

```
$ bank_balance_m1 : num NA 0 0 443 11447 ...
```

```
$ bank_balance_m2 : num 0 50 0 9343 25967 ...
```

```

$ total_withdrawals_m1: num NA 4948 49095 5200 20178 ...
$ total_withdrawals_m2: num 850 8579 16928 8795 15748 ...
$ total_deposit_m1 : num NA 4150 36868 NA 6150 ...
$ total_deposit_m2 : num 1000 7718 6644 11702 39368 ...
$ expenses : num 0 1063 62020 0 14281 ...
$ disposable_income_m1: num 0 -748 -12227 4143 11939 ...
$ disposable_income_m2: num 150 -461 -10284 2907 23620 ...
$ total_withdrawals_m3: num NA 2000 NA NA NA ...
$ num_credit_card : int 0 1 1 1 2 1 0 0 0 ...
$ total_deposit_m3 : num NA 500 NA NA NA ...
$ bank_balance_m3 : num NA 400 NA NA NA NA 0 0 NA 0 ...
$ disposable_income_m3: num 0 -1500 0 0 0 ...
$ bank_balance_m4 : num NA NA NA NA NA NA 0 NA NA 0 ...
$ bank_balance_m5 : num NA NA NA NA NA ...
$ obligations_m1 : int NA NA NA NA NA NA NA NA NA NA ...
$ bank_balance_m6 : num NA NA NA NA NA ...
$ obligations_m2 : num NA NA NA NA NA NA NA NA NA NA ...
$ num_loans : int 0 0 0 0 0 0 0 0 0 ...
$ obligations_m3 : num NA NA NA NA NA NA NA NA NA NA ...
$ car_ownership : int 0 0 0 0 0 0 0 0 0 ...
$ bike_ownership : int 0 0 0 0 0 0 0 0 0 ...
$ cust_type : int 1 1 1 1 1 1 1 0 0 ...

```

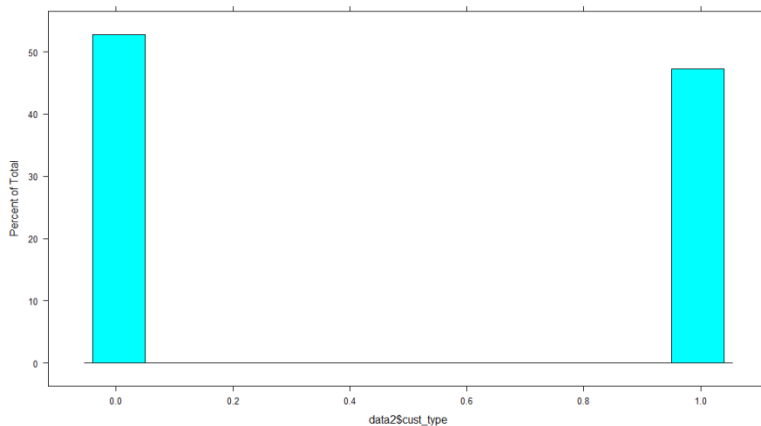
There are no string variable all the variables are numeric, some variable are categorical such as num_credit_card, obligations_m1, num_loans, car_ownership and bike_ownership.

Knowing more about class variable

```

> #Histogram of cust_type
> histogram(cust_type)

```



```

> #Proportion table
> prop.table(table(data$cust_type))

```

```

      0      1
0.5155 0.4845

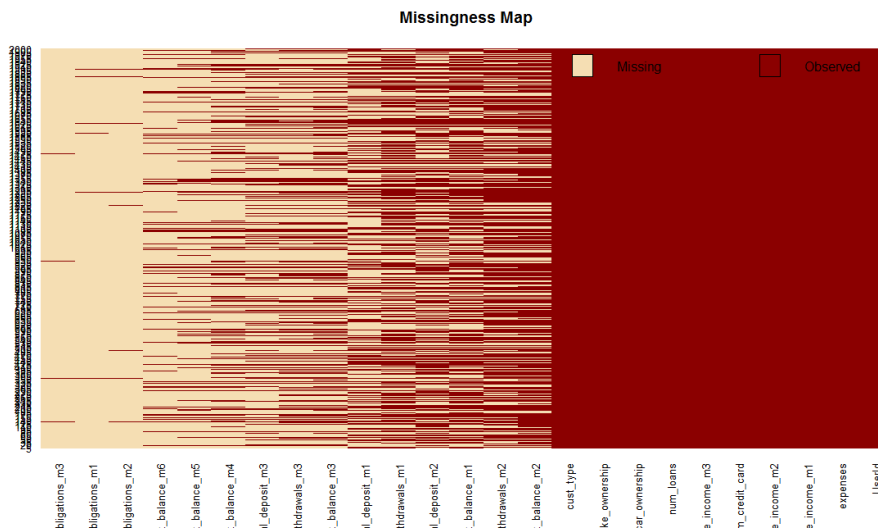
```

In the class variable is 1031 observations lie in 0 class and 969 lie in 1 class thus the ratio is 0.5155:0.4842. The classes are pretty balanced.

Data Preparation procedure:

1) Missing value treatment:

a) Missing values map:



The data set contains lot of missing values in some variables. We will remove those variables having missing values greater than 80% and we will have to fill missing values.

```
> z=0
> for(i in 1:25)
+ {
+   z[i] <-length(which(is.na(data[,i])==TRUE))
+ }
> z
[1] 0 838 580 920 707 1142 899 0 0 0 1319 0 1393 1234 0 1470
[17] 1628 1980 1716 1970 0 1983 0 0 0

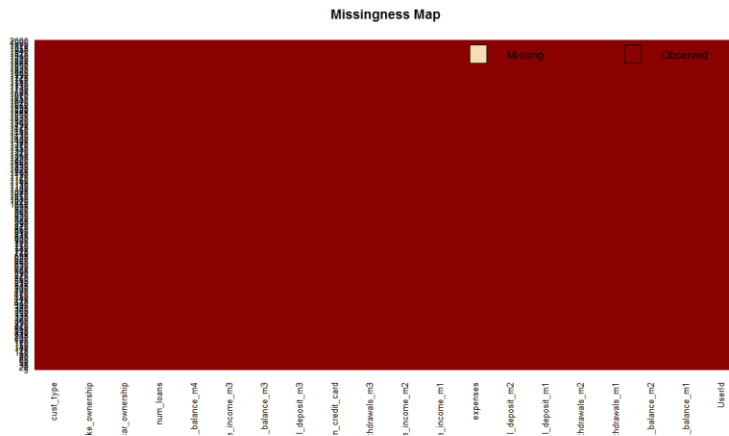
> #The variable numbers having missing values greater than 80%
> max_miss <- which(z>=0.8*nrow(data))
> data1 <- data[,-max_miss]
```

Now We use K-Nearest Neighbors approach to impute missing values. *What kNN imputation does in simpler terms is as follows: For every observation to be imputed, it identifies 'k' closest observations based on the euclidean distance and computes the weighted average (weighted based on distance) of these 'k' obs. The advantage is that you could impute all the missing values in all variables with one call to the function.*

```
> #Imputation of missing values using KNN imputation
> library(DMwR)
> data1[, names(data1)] <- data1
> data1[,] <- knnImputation(data1[,], k=5)
```

After imputation of the missing values we check whether all the missing values are filled using Missing map becomes

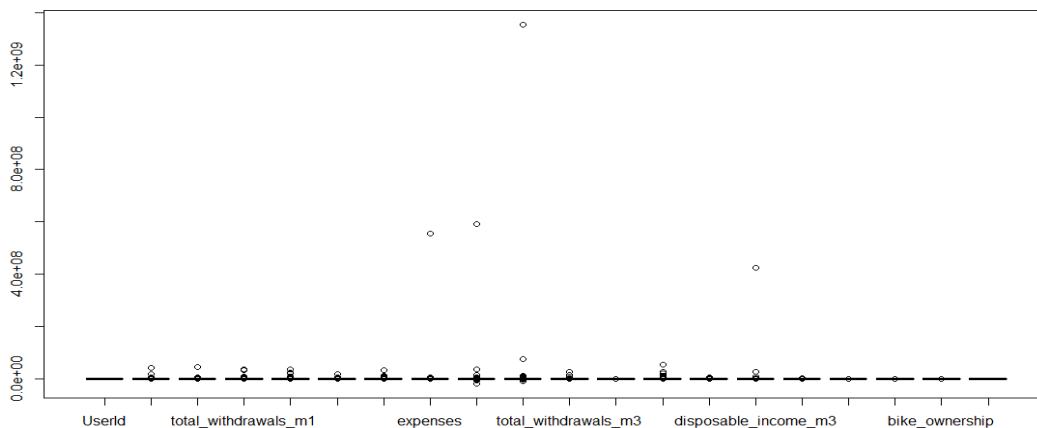
```
> missmap(data1)
```



The above missing map shows us that there are no missing values in our new data. Thus we proceed to deal with outliers now.

2) Outlier detection:

```
> #finding outliers in data
> boxplot(data1)
```



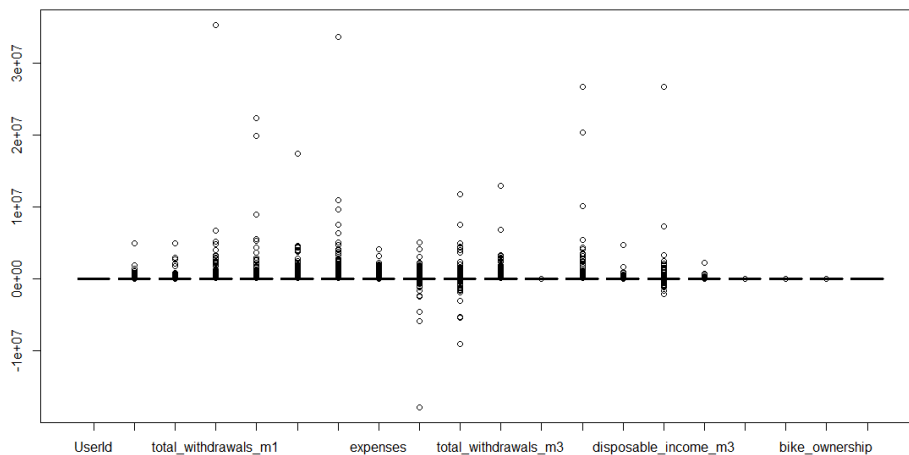
Here some of variable are having too much extreme values hence we remove these outliers iteratively and then use the Mahalanobis distance for multivariate outliers.

```
> #From the above boxplot we see that variables "expenses","disposable_income_m1","disposable_income_m2","disposable_income_m3"
> #have extreme values hence we need to Remove those
> z <- max(data1[,15])
> out1 <- which(data1[,15]==z)
```

```

> data1 <- data1[-out1,]
> #boxplot(data1)
> z <- max(data1[,10])
> out2 <- which(data1[,10]==z)
> data1 <- data1[-out2,]
> #boxplot(data1)
> z <- max(data1[,9])
> out3 <- which(data1[,9]==z)
> data1 <- data1[-out3,]
> data2 <- data1[, -1]
> #Now we have deleted extreme observations and want to focus on
> #removing the outliers from particular observation
> boxplot(data1)

```

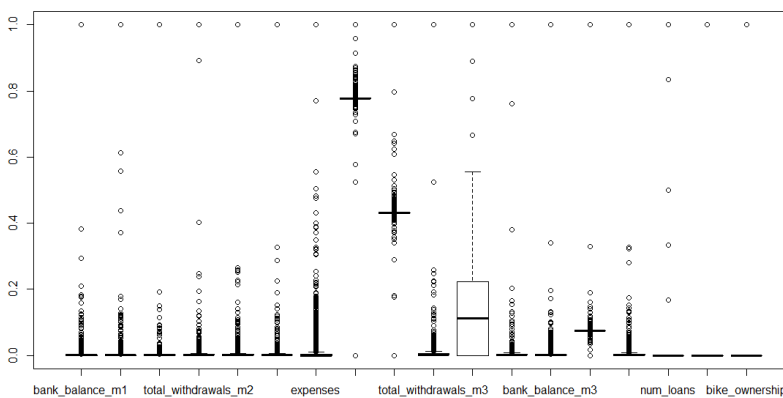


The above data yet shows variation hence we use min max normalization for scaling the data and then we will use Mahalanobis distance to remove outliers from the data.

```

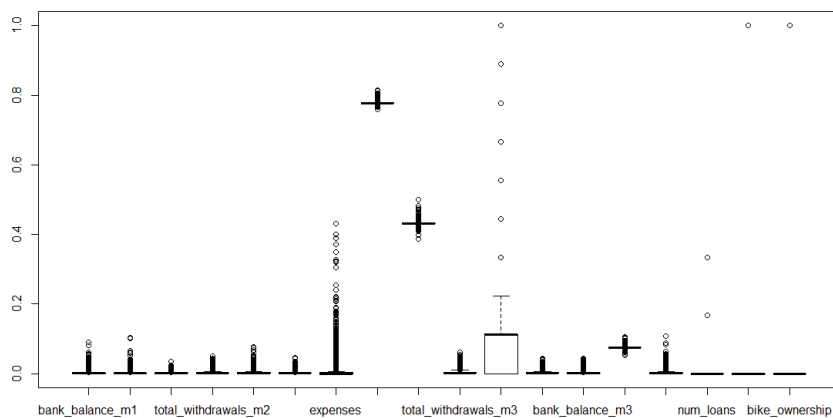
> for(i in 1:18)
+ {
+   max <- max(data2[,i])
+   min <- min(data2[,i])
+   data2[,i] <- (data2[,i] - min)/(max-min)
+ }
> boxplot(data2)

```



The data is scaled now thus we are ready to use mahalanobis distance for outlier detection and removal

```
> #Mahalanobis distance
> mat <- cov(data2[, -19])
> mah <- mahalanobis(data2[, -19], colMeans(data2[, 1:18]), mat)
> mean <- mean(mah)
> stdev <- sqrt(var(mah))
> Ucl <- mean + 3*stdev
> Lcl <- mean - 3*stdev
> z <- which(mah >= Ucl)
> z1 <- which(mah <= Lcl)
> data2 <- data2[-c(z, z1), ]
> boxplot(data2[, -19])
```

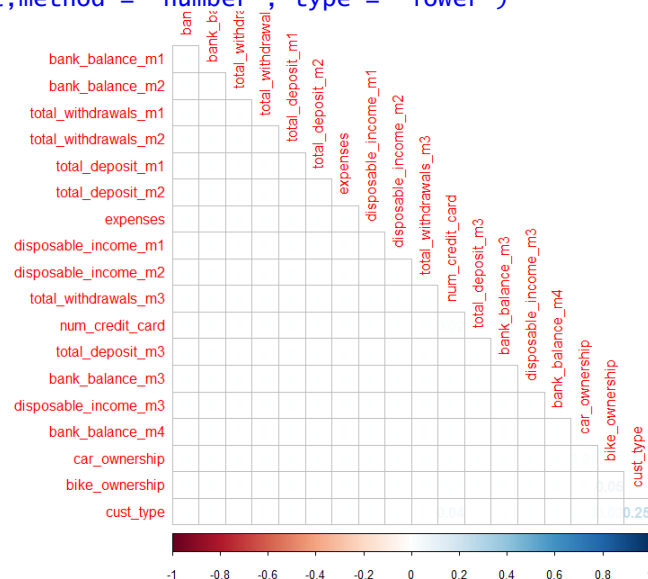


After removing outliers using Mahalanobis distance iteratively we have most of the outliers are removed from the data. We will proceed on this data for variable selection.

3. Variable selection:

We will check the correlation between independent variables.

```
> #install.packages("corrplot")
> library(corrplot)
> mat <- cov(data2[, -16])
> corrplot(mat, method = "number", type = "lower")
```



From the above correlation plot we can see that the variables are having less correlation hence we will try to find if there is multicollinearity.

Multicollinearity: for measuring multicollinearity we use the variance inflation factor(VIF).

If VIF happens to be greater than 5 then we say that the multicollinearity is present in data.

VIF is calculated by regression one of independent variable on the remaining ones. The formula for VIF is

$$VIF = \frac{1}{1 - R_i^2}$$

```
> #Checking for multicollinearity
> library(car)
> fit <- lm(cust_type ~ ., data = data2)
> vif1 <- vif(fit)
> vif1
```

bank_balance_m1	bank_balance_m2	total_withdrawals_m1	total_withdrawals_m2
1.201472	2.392639	9.737771	17.632206
total_deposit_m1	total_deposit_m2	expenses	disposable_income_m1
14.595037	27.723238	1.512882	7.074932
disposable_income_m2	total_withdrawals_m3	num_credit_card	total_deposit_m3

14.593322	4.317302	1.951774	5.563913
bank_balance_m3	disposable_income_m3	bank_balance_m4	num_loans
1.665718	2.710362	1.345812	1.016006
car_ownership	bike_ownership		
1.107891	1.133118		

We will remove variable having highest VIF first and then again compute the VIF for the remaining variable. The procedure will be continued unless all the variables will have VIF less than 5.

```
> # we remove total_deposit_m2 variable first and again find vif
```

```
> data2 <- data2[,-6]
```

```
> fit <- lm(cust_type~., data = data2)
```

```
> vif1 <- vif(fit)
```

```
> vif1
```

bank_balance_m1	bank_balance_m2	total_withdrawals_m1	total_withdrawals_m2
1.201358	2.379691	9.322372	1.824622
total_deposit_m1	expenses	disposable_income_m1	disposable_income_m2
14.289212	1.512863	6.934624	1.299531
total_withdrawals_m3	num_credit_card	total_deposit_m3	bank_balance_m3
4.317219	1.947521	5.528729	1.442646
disposable_income_m3	bank_balance_m4	num_loans	car_ownership
2.674289	1.338649	1.015785	1.105203
bike_ownership			
1.124787			

Step2

```
> # in second step we remove variable total_deposit_m1 as it has maximum VIF
```

```
> data2 <- data2[,-5]
```

```
> fit <- lm(cust_type~., data = data2)
```

```
> vif1 <- vif(fit)
```

```
> vif1
```

bank_balance_m1	bank_balance_m2	total_withdrawals_m1	total_withdrawals_m2
1.191588	1.317553	1.662328	1.796219
expenses	disposable_income_m1	disposable_income_m2	total_withdrawals_m3
1.512347	1.375624	1.299513	4.307993
num_credit_card	total_deposit_m3	bank_balance_m3	disposable_income_m3
1.918788	5.507582	1.439647	2.665121
bank_balance_m4	num_loans	car_ownership	bike_ownership
1.320138	1.015758	1.089453	1.122008

Step3

```
> #now just one variable is having multicollinearity greater than 5.
```

```
> #hence we remove that one variable also
```

```
> data2 <- data2[,-10]
```

```
> fit <- lm(cust_type~., data = data2)
```

```
> vif1 <- vif(fit)
```

```
> vif1
```

bank_balance_m1	bank_balance_m2	total_withdrawals_m1	total_withdrawals_m2
1.190894	1.308903	1.587289	1.795419
expenses	disposable_income_m1	disposable_income_m2	total_withdrawals_m3
1.509906	1.369821	1.299177	1.778413
num_credit_card	bank_balance_m3	disposable_income_m3	bank_balance_m4

1.903667	1.439590	1.128030	1.294915
num_loans	car_ownership	bike_ownership	
1.014648	1.088428	1.113677	

The VIF of all variables in newly obtained data is less than 5. Thus multicollinearity is removed from data and we are ready to build models.

4. Data Partition:

Before building model we will partition the model in two parts as train and test data. We will use 70% of data to train the model and 30% of data to evaluate or test the model.

```
> #Splitting data
> n<-nrow(data2)
> s<-sample(n,n*0.7,replace = F)
> train <- data2[s,]
> test <- data2[-s,]
> train_data <- train[,-16]
> train_class <- as.factor(train[,16])
> test_data <- test[,-16]
> test_class <- test[,16]
```

5. Model building on training data

Logistic Regression: For building model we use logistic regression classification for classification.

Model Building:

```
> #Logistic regression model fitted on
> model <- glm(cust_type~.,data = train,family = binomial(link = "logit"))
> summary(model)
```

```
call:
glm(formula = cust_type ~ ., family = binomial(link = "logit"),
    data = train)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-3.5101	-0.6584	-0.6107	0.7723	2.1916

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	7.0275	21.3898	0.329	0.742499
bank_balance_m1	-38.0566	15.2435	-2.497	0.012540 *
bank_balance_m2	-16.1212	14.3895	-1.120	0.262569
total_withdrawals_m1	-5.7761	37.5048	-0.154	0.877602

```

total_withdrawals_m2 -3.2104    26.2841   -0.122  0.902787
expenses            21.6236     8.1957    2.638  0.008329 **
disposable_income_m1 -24.6884   29.5878   -0.834  0.404049
disposable_income_m2  10.7602   21.4035    0.503  0.615155
total_withdrawals_m3  66.0341   18.5043    3.569  0.000359 ***
num_credit_card      10.7805    0.9627   11.198  < 2e-16 ***
bank_balance_m3     -43.7654   27.1321   -1.613  0.106734
disposable_income_m3  81.5274   46.5362    1.752  0.079789 .
bank_balance_m4      22.4479   10.3687    2.165  0.030390 *
num_loans            8.2438    3.8983    2.115  0.034454 *
car_ownership        14.2009   502.7248    0.028  0.977464
bike_ownership        0.6515    0.4183    1.557  0.119382
---

```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

```

Null deviance: 1821.3 on 1317 degrees of freedom
Residual deviance: 1295.3 on 1302 degrees of freedom
AIC: 1327.3

```

Number of Fisher Scoring iterations: 15

Predicting values for test data:

```

> pred<- predict(model,test_data)#type = "class"
> pred1 <- ifelse(pred>=0.5,1,0)

```

Confusion matrix:

```

> confusionMatrix(test_class, pred1 , mode = "everything")

```

Confusion Matrix and Statistics

```

      Reference
Prediction  0   1
0  287   25
1  127  126

```

```

      Accuracy : 0.731
      95% CI   : (0.6924, 0.7671)
No Information Rate : 0.7327
P-Value [Acc > NIR] : 0.5595

```

```

      Kappa : 0.4345
McNemar's Test P-Value : 2.565e-16

```

```

      Sensitivity : 0.6932
      Specificity : 0.8344
Pos Pred Value : 0.9199
Neg Pred Value : 0.4980
Precision : 0.9199
Recall : 0.6932
F1 : 0.7906

```

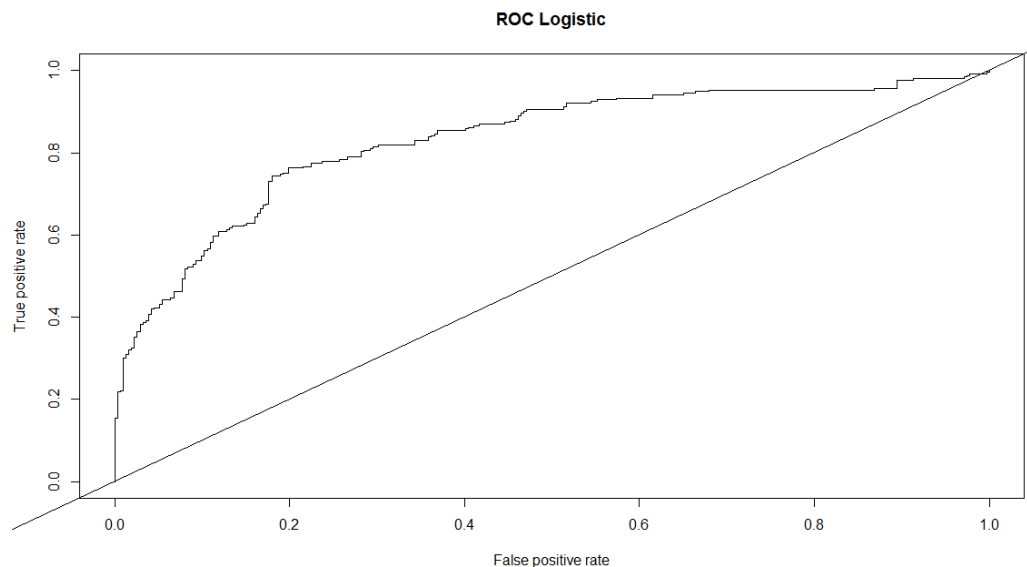
```
Prevalence : 0.7327
Detection Rate : 0.5080
Detection Prevalence : 0.5522
Balanced Accuracy : 0.7638
```

```
'Positive' class : 0
```

Accuracy: By using logistic regression classifier we get a accuracy of 73.1% and Recall is 69.32%.

ROC Curve:

```
> #ROC curve
> library(ROCR)#
> pred1=prediction(predict(model,test_data),test$cust_type)
> perf1 <- performance(pred1,"tpr","fpr")
> plot(perf1,main="ROC")
> abline(0,1)
```

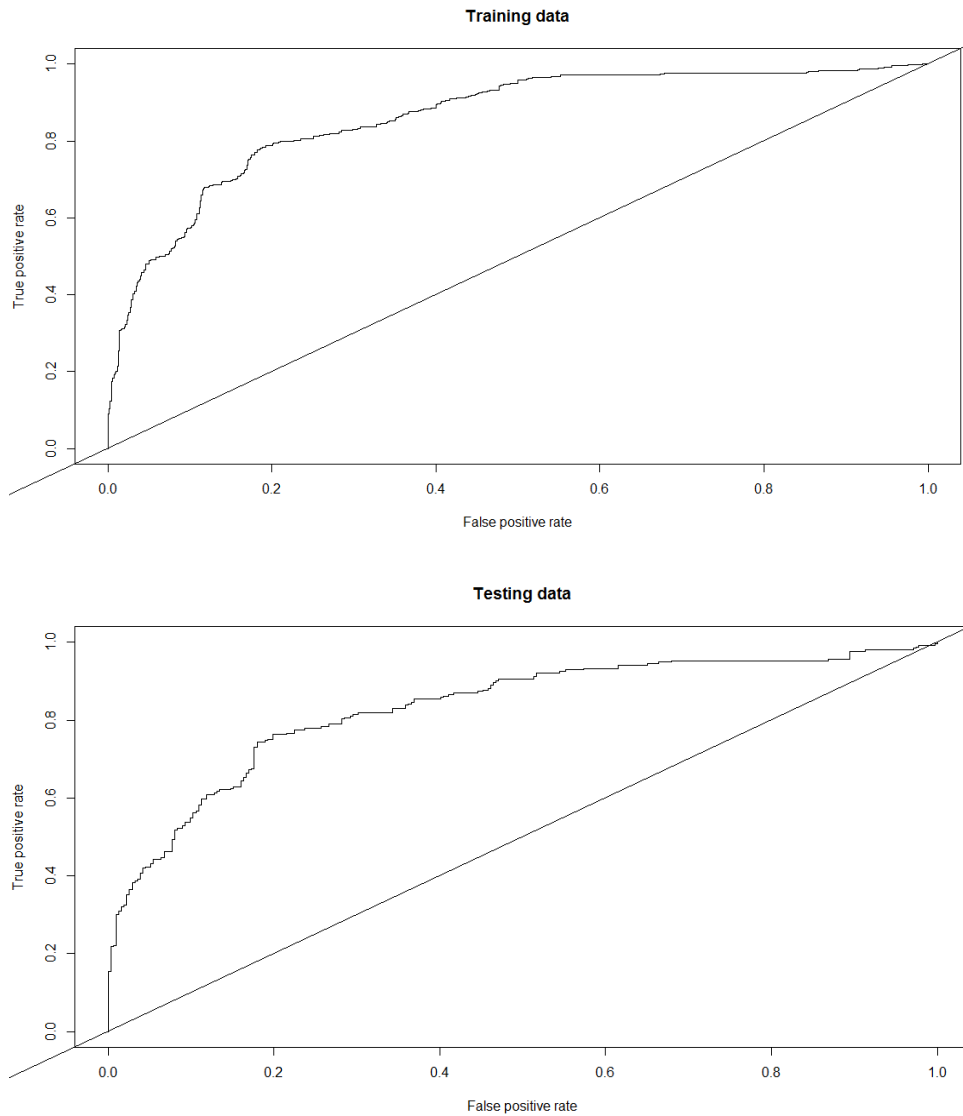


Build Model on same data to see results are stable or not.

```
> #train and test data set with pred prob variables
> train_pred = predict(model,data=train_data)
> test_pred<-predict(model,test_data)

> train_pred1 <- prediction(train_pred, train$cust_type)
> train_perf1 <- performance(train_pred1, measure = "tpr", x.measure = "fpr")
> plot(train_perf1,main="Training data")
> abline(0,1)
```

```
> val_pred <- prediction(test_pred, test$cust_type)
> val_perf <- performance(val_pred, measure = "tpr", x.measure = "fpr")
> plot(val_perf, main="Testing data")
> abline(0,1)
```



Here lift is good for train and test data, we can see the results are stable for both train and test data hence there is no problem of over fitting.

6) Other Classifiers:

Knn Classifier

```
library(class)
pred <- knn(train_data, test_data, train_class, k=1,l=0)
confusionMatrix(test_class,pred,mode = "everything")
```

#SVM

```
library(e1071)
model <- svm(cust_type~.,data=train)
pred<- predict(model,test_data,type = "Class")
pred1 <- ifelse(pred>=0.5,1,0)
library(caret)
confusionMatrix(test_class, pred1 , mode = "everything")
pred1=prediction(predict(model,test_data),test$cust_type)
perf1 <- performance(pred1,"tpr","fpr")
plot(perf1,main="ROC")
abline(0,1)
```

#Decision tree using rpart package

```
library(rpart)
model <- rpart(cust_type~., data = train)
pred<- predict(model,test_data)#type = "Class"
pred1 <- ifelse(pred>=0.5,1,0)
confusionMatrix(test_class, pred1 , mode = "everything")
library(rpart.plot)
rpart.plot(model)
pred1=prediction(predict(model,test_data),test$cust_type)
perf1 <- performance(pred1,"tpr","fpr")
plot(perf1,main="ROC")
abline(0,1)
```

#Neural net

```
library(nnet)
model<-nnet(cust_type~.,data<-train,size=40)
```

```

pred<-predict(model,test_data)
pred1<-ifelse(pred>0.56,1,0)
confusionMatrix(test_class,pred1,mode = "everything")
pred1=prediction(predict(model,test_data),test$cust_type)
perf1 <- performance(pred1,"tpr","fpr")
plot(perf1,main="ROC")
abline(0,1)

```

```

#Random Forest
library(randomForest)
model <- randomForest(cust_type~.,data = train)
summary(model)
pred<-predict(model, test_data)
pred1<-ifelse(pred>0.56,1,0)
confusionMatrix(test_class,pred1,mode = "everything")
pred1=prediction(predict(model,test_data),test$cust_type)
perf1 <- performance(pred1,"tpr","fpr")
plot(perf1,main="ROC")
abline(0,1)

```

Results:

1. Knn Classifier Accuracy = 0.8566
 Recall = 0.8714
 Kappa = 0.7102

2. SVM Classifier Accuracy = 0.7628
 Recall = 0.7557
 Kappa = 0.5139

3. Decision Tree Accuracy = 0.8549
 Recall = 0.9457
 Kappa = 0.7123

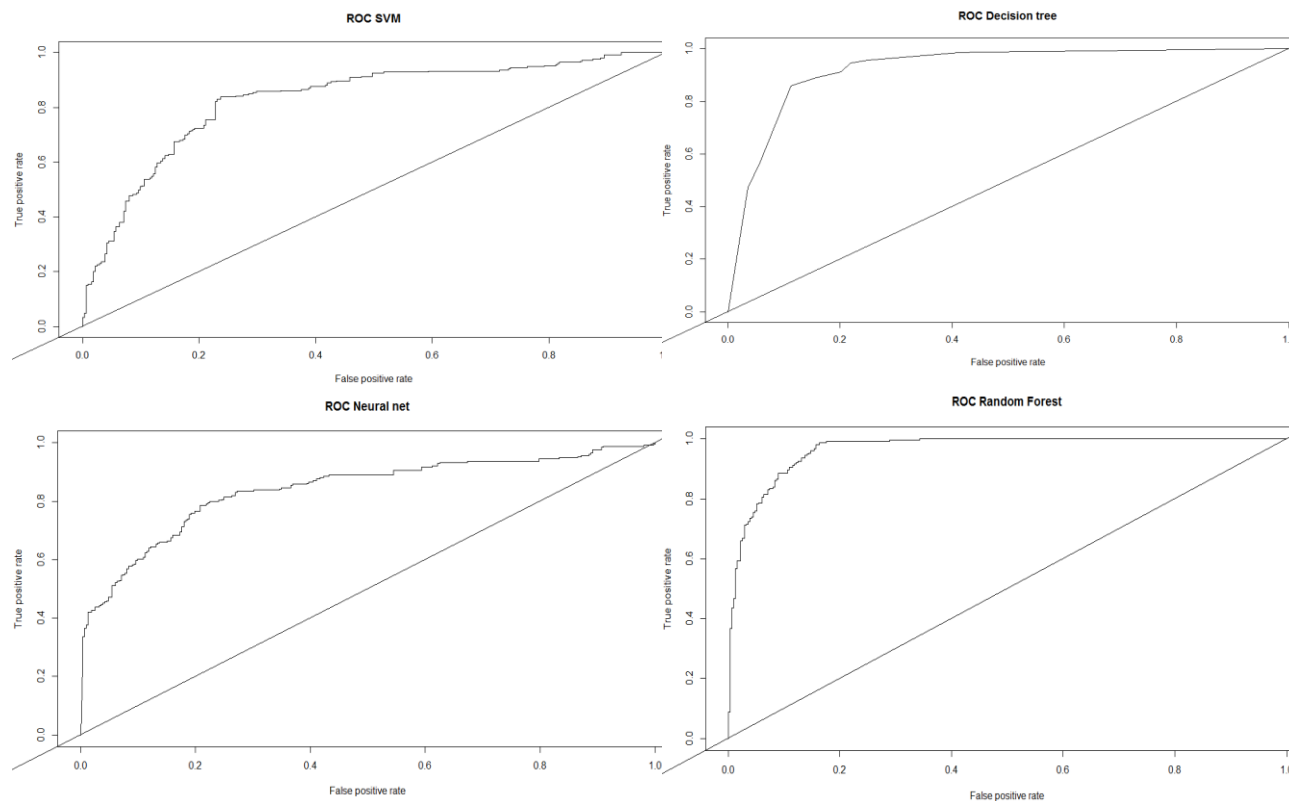
4. Neural net Accuracy = 0.7823
 Recall = 0.7739
 Kappa = 0.544

5. Random Forest Accuracy = 0.9009

Recall = 0.9238

Kappa = 0.7859

ROC Curves:



7) Major Findings:

- 1) The tree based classifier random forest and works better and Knn algorithm than any other classifier in classifying the defaults.
- 2) Using Random Forest classifier we can predict with maximum accuracy and recall that the customer will be default or not.