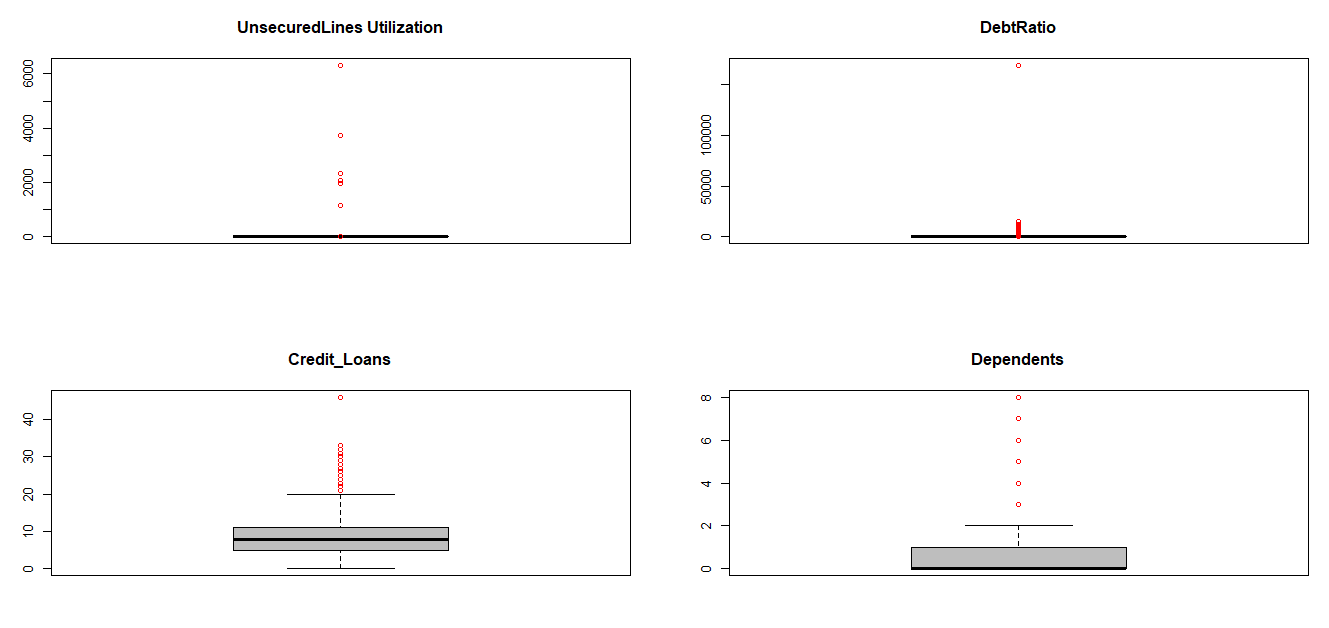
**FRA- Group 5- Solution**

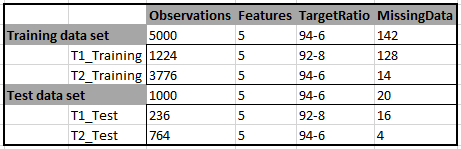
**Data Set**

* Skewed towards class – 0. Target variable ratio is 94:6 in training data and test data (approximately). So we aimed to handle this by SMOTE or Borderline SMOTE
* Outliers are very highly scaled in UnsecuredLines and DebtRatio. As we are doing classification and most of the algorithms works on the length. This huge point will impact the model. So we aimed at splitting the data into 2 type T1(with outliers) and T2(without outliers). We have considered outliers as the ratios that are more than 1 for UnsecuredLines and DebtRatio. We have ignored Dependents and Credit\_Loans outliers while splitting the data as by above conditions mostly these outliers too fell in T1.

Below plot is the original Train data set outlier representations.



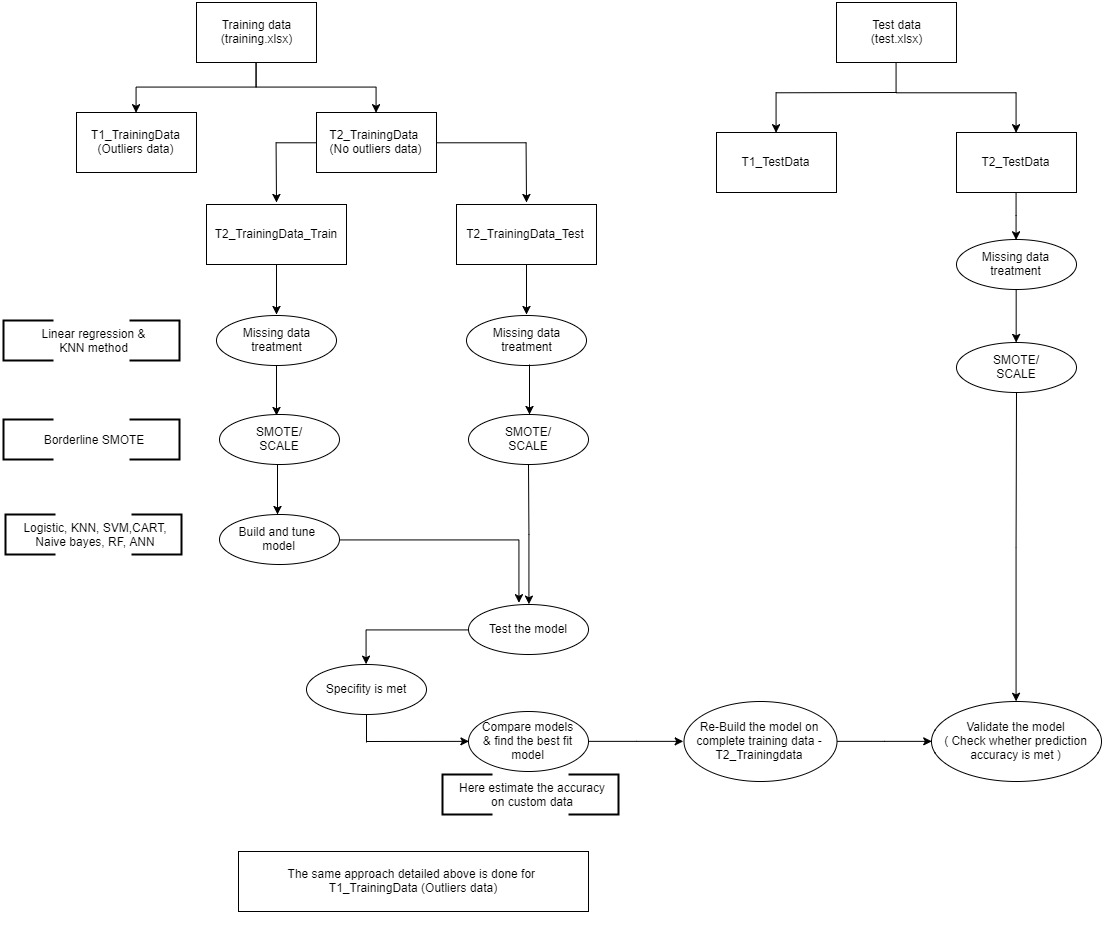
* Data sets measurements after splitting based on outliers, target ratio and missing values (Only Dependents feature is having missing data).



**Methods & Sequence followed**

* Missing Values: These are handled using KNN. We have tried KNN and Linear regression as KNN has given various classes and Linear regression tend to predict all of them to a same value.
* Skewness Handling: This is taken care by board line SMOTE. We have tried SMOTE also but SMOTE gave us points away from the cluster thus over simplified the majority region too.
* Scaling: We have applied normalized scaling to all the independent variables as the models are based on distance length.
* T1\_TrainingData(outliers) is split into 75:25 T1\_TrainingData\_Train and T1\_TrainingData\_Test. We have built our models on T1\_TrainingData\_Train and tested fitness of the model against T1\_TrainingData\_Test. Only stable models are moved to test finally on T1\_TestData (part of original test data provided). Similar process is followed on T2 data sets.
* Accuracy: We have considered Specificity as the measure of accuracy while selecting or testing the stability across the models.

Below diagram is the flow chart representing the process we followed.



**Non Outlier Models and Specificities**

Tried to explain the output for Train split on T1\_Training data. The same process has been followed for Test data split of T1\_Training data and same approach has been done on T2\_Training data (separate scaling, B-SMOTE parameters). Mostly encapsulated the code in functions for reusability, entire code has been provided at the end of the document.

**Data Wrangling**

* Data format handling –

traindata = Data\_Building(traindata)

Original data formats:

DLQs Utlz\_UnsecLines DebtRatio Credit\_Loans

"numeric" "numeric" "numeric" "numeric"

Dependents UUL\_flag DR\_flag

"character" "numeric" "numeric"

Final data formats:

DLQs Utlz\_UnsecLines DebtRatio Credit\_Loans

"factor" "numeric" "numeric" "numeric"

Dependents UUL\_flag DR\_flag

"numeric" "factor" "factor"

* Missing data handling –

T2\_traindata\_Train = Missing\_data\_handling(T2\_traindata\_Train)

Variables VIF

1 Utlz\_UnsecLines 1.069444778

2 DebtRatio 1.171752612

3 Credit\_Loans 1.187102216

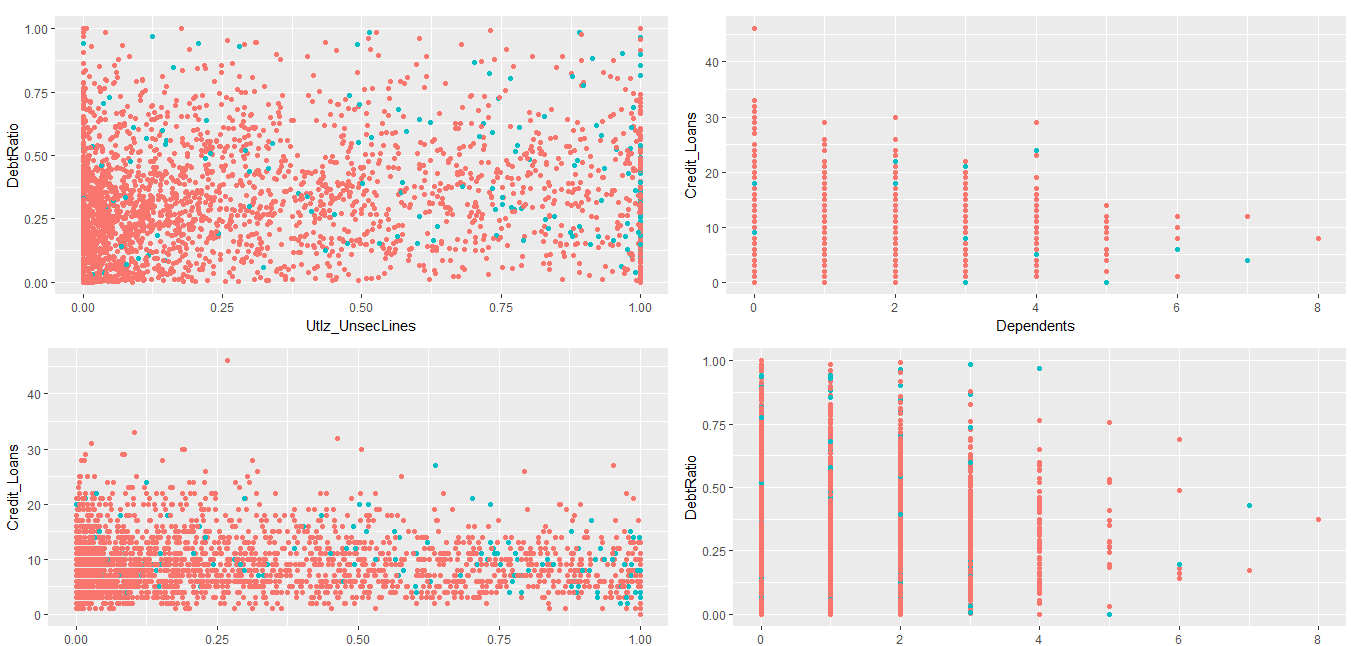
KNN Accuracy: 0.5

Linear regression Accuracy: 0.2

Missing data in each column after handling:

0000000

* We have a 4 features in the dataset so visualize the data we have plotted 4 2D scatter graphs. Below is the approach plot for T2\_traindata\_Train



* SMOTE method (to handle skewness) below are the plot and Target variable results

T2\_traindata\_Train\_SMOTEd = SMOTE\_fitting(T2\_traindata\_Train,600,300)

Original data ratio:

0 1

0.94 0.06

SMOTEd data ratio:

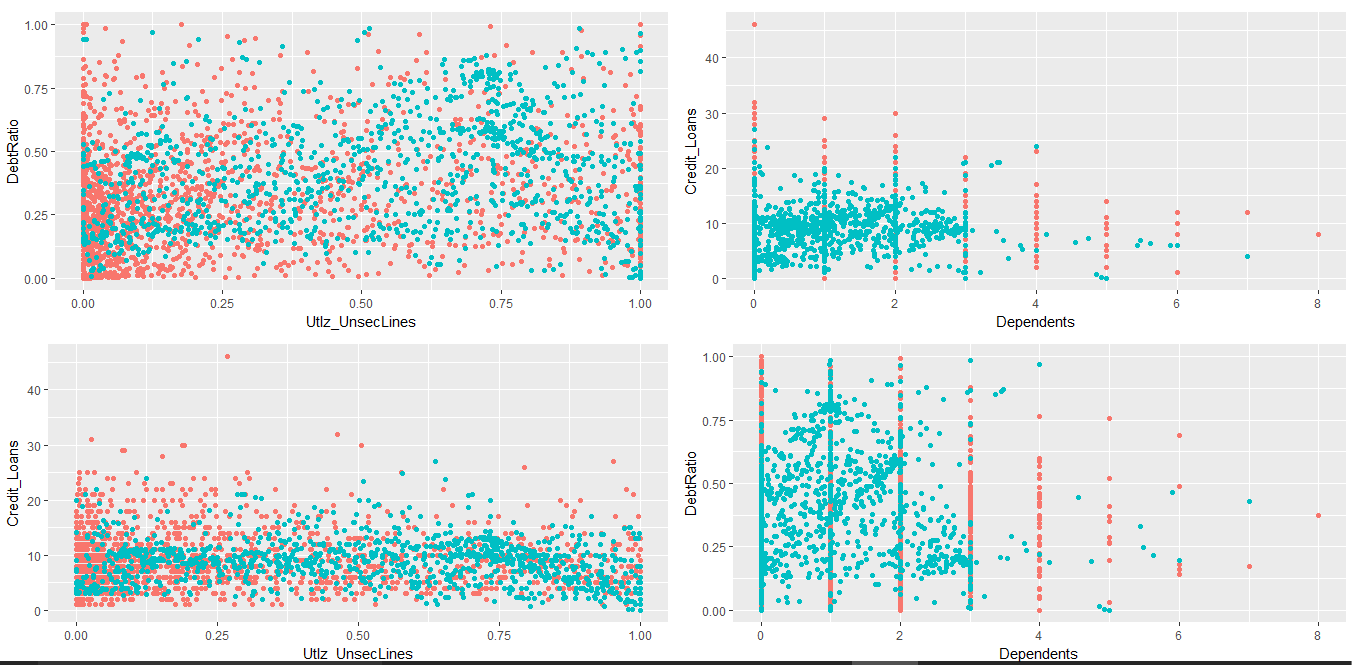
0 1

0.72 0.28

SMOTEd data split

0 1

2988 1162



The problem we faced is that it has over fitted the regions of Majority class. So we have tried borderline SMOTE.

* Borderline SMOTE –

T2\_traindata\_Train\_BS = Boderline\_SMOTE\_fitting(T2\_traindata\_Train,25)

[1] "Borderline-SMOTE done"

Boarderline SMOTE data Target variable ratio:

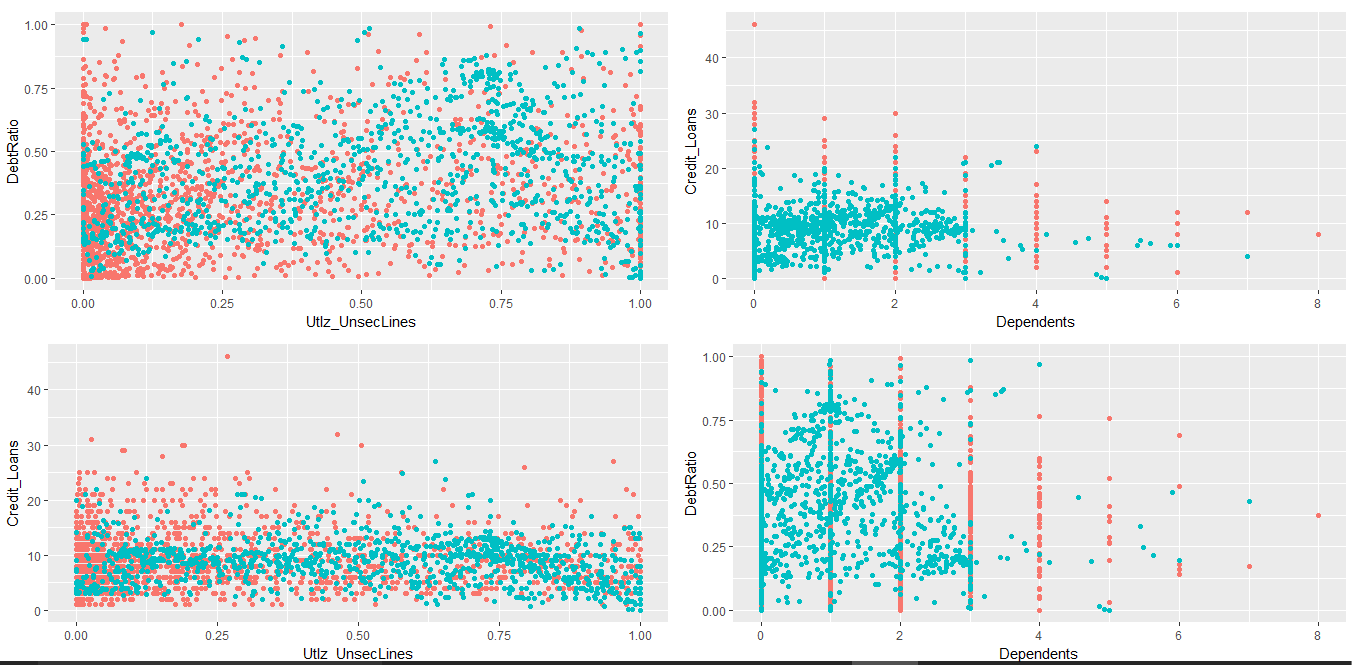
1 2

0.7 0.3

Original data set Target ratio:

0 1

0.94 0.06



Looks much better than SMOTE. Only problem is that Dependents and CreditLoans are integer numbers but due to the random point pick inside the neighbouring bounds of similar class records. This has happened.

* Scaling:

Scaling <- function(data){

+ data\_scaled = data

+ data\_scaled[-1] = scale(data\_scaled[-1])

+ return(data\_scaled)

+ }

> T2\_traindata\_Test\_BS\_Scaled = Scaling(T2\_traindata\_Test\_BS)

> T2\_traindata\_Train\_BS\_Scaled = Scaling(T2\_traindata\_Train\_BS)

This data is being used in modelling.

**Data Modelling:**

* Logistic Regression –

Results on T2\_traindata\_Train\_BS\_Scaled data.

lrtest(T2\_LR) # overall test is significant

Likelihood ratio test

Model 1: DLQs ~ Utlz\_UnsecLines + DebtRatio + Credit\_Loans + Dependents

Model 2: DLQs ~ 1

#Df LogLik Df Chisq Pr(>Chisq)

1 5 -1455.9295

2 1 -2290.8083 -4 1669.7575 < 0.000000000000000222 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> pR2(T2\_LR) # 35 - very good McFadden R2

llh llhNull G2 McFadden

-1455.9295218061 -2290.8082805713 1669.7575175304 0.3644472415

r2ML r2CU

0.3576819315 0.5086614376

> LR\_Speci\_Train = CM[4]/(CM[4]+CM[2])

> round(LR\_Speci\_Train\*100,2)

[1] 78.14

The results on T2\_traindata\_Train\_BS\_Scaled data is having a Specificity of 78.14 % and MC Fadden R2 of 35 indicating a very good model and overall test is significance.

Following is the result of K-fold on same T2\_traindata\_Train\_BS\_Scaled.

LR\_Speci\_KF = mean(as.numeric(cv))

> round(LR\_Speci\_KF\*100,2)

[1] 77.78

The results are more or less the same. So testing on the model test data, T2\_traindata\_Test\_BS\_Scaled.

prob\_pred = predict(T2\_LR, type = 'response', newdata = T2\_traindata\_Test\_BS\_Scaled[-1])

> y\_pred = ifelse(prob\_pred > 0.55, 1, 0)

> CM = table(T2\_traindata\_Test\_BS\_Scaled[,1],y\_pred)

> LR\_Speci\_Test = CM[4]/(CM[4]+CM[2])

> round(LR\_Speci\_Test\*100,2)

[1] 86.29

There is increase of 10% specificity on model test data, so concluded this as “under fitted”. Hence not choosing to test on original test data.

* KNN Classification –

Used Caret packet to tune the K value for number of neighbours.

caret\_tune = train(form = DLQs~ ., data = T2\_traindata\_Train\_BS\_Scaled, method = 'knn')

> caret\_tune

k-Nearest Neighbors

3772 samples

4 predictor

2 classes: '0', '1'

No pre-processing

Resampling: Bootstrapped (25 reps)

Summary of sample sizes: 3772, 3772, 3772, 3772, 3772, 3772, ...

Resampling results across tuning parameters:

k Accuracy Kappa

5 0.9131763685 0.7943492544

7 0.9139152044 0.7961727342

9 0.9132985382 0.7952232623

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was k = 7.

> caret\_tune$bestTune # caret to tune for k value

k

2 7

With K as 7 below are the results for KNN fold validation

Knn\_Speci\_KF = mean(as.numeric(cv))

> Knn\_Speci\_KF

[1] 0.8924629987

Tested the same with model testing data

Knn\_Speci\_Test = CM[4]/(CM[4]+CM[2])

> Knn\_Speci\_Test

[1] 0.1838006231

There was a huge drop in specificity of around 70%. So called this as “Over fitted”, and not going to deploy this on original Test data.

* Support Vector Classification

We have used caret to find the optimal parameters for linear, polynomial, radial and sigmoid. Used these respective values and found the below specificities for k-fold validation on Training data.

[1] linear-kernal C-classification has K-fold specificity of 83.961

[1] polynomial-kernal C-classification has K-fold specificity of 5.446

[1] radial-kernal C-classification has K-fold specificity of 88.8

[1] sigmoid-kernal C-classification has K-fold specificity of 37.542

[1] linear-kernal nu-classification has K-fold specificity of 83.961

[1] polynomial-kernal nu-classification has K-fold specificity of 5.446

[1] radial-kernal nu-classification has K-fold specificity of 88.8

[1] sigmoid-kernal nu-classification has K-fold specificity of 37.542

We have test the best model with 88.8% specificity, Radial C-Classification on model test data and follow the following

y\_pred = predict(T2\_SVM, newdata = T2\_traindata\_Test\_BS\_Scaled[-1])

> CM = table(T2\_traindata\_Test\_BS\_Scaled[,1],y\_pred)

> SVM\_Speci\_Test = CM[4]/(CM[4]+CM[2])

> SVM\_Speci\_Test

[1] 0.08722741433

The model specificity has dropped to 8% I.e. 80% drop. As due to this Overfitting of this model. We didn’t implement this on original test data.

* Naïve Bayes Classification

On model training data below is the specificity results

T2\_NB = naiveBayes(x = T2\_traindata\_Train\_BS\_Scaled[-1],

+ y = T2\_traindata\_Train\_BS\_Scaled$DLQs)

> y\_pred = predict(T2\_NB, newdata = T2\_traindata\_Train\_BS\_Scaled[-1])

> CM = table(T2\_traindata\_Train\_BS\_Scaled[,1],y\_pred)

> NB\_Speci\_Train = CM[4]/(CM[4]+CM[2])

> NB\_Speci\_Train

[1] 0.8109318996

The K-fold test on model training data is also showing similar specificity.

NB\_Speci\_KF

[1] 81.004

On model test data also this showed similar specificity. So this is used to test on the original test data (Those results are in the later section).

* Decision Tree Classification – CART.

Caret package used to find out the best complexity parameter, which is helpful in post pruning the tree. For pre pruning minsplit of 225 (2.5% of overall training data population on which this is build) and xval of 7 for tree cross validation. These values are obtained after multiple runs, brute-force methods basing specificity as accuracy parameter.

caret\_tune$bestTune # CP - Tunning

cp

1 0.03315412186

Model performance on model train data

CART\_Speci\_Train = CM[4]/(CM[4]+CM[2])

> CART\_Speci\_Train

[1] 0.7535842294

Model performance on same model train data but with K-fold, remained the same.

CART\_Speci\_KF = round(mean(as.numeric(cv)),5)\*100

> CART\_Speci\_KF

[1] 78.585

So testing the specificity on model train data

CM = table(T2\_traindata\_Test\_BS\_Scaled[,1],y\_pred)

> CART\_Speci\_Test = CM[4]/(CM[4]+CM[2])

> CART\_Speci\_Test

[1] 0.3925233645

Specificity has dropped by 35% so due to “over fitting” we are not going to implement this model on test data.

* Random Forest

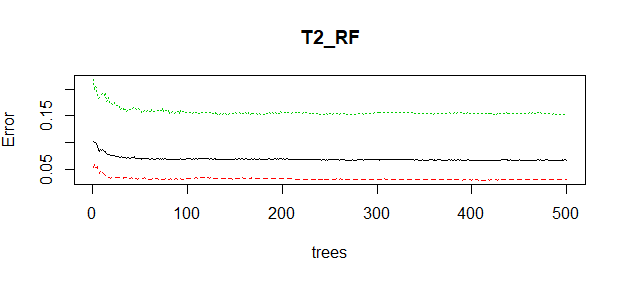
Caret package used to find optimal mtry parameter

caret\_tune$bestTune # mtry - Tunning

mtry

1 2

From build an initial model with 500 trees and optimal mtry of 2 and verified the OOB to find out the flattening point for optimal ntree.



Took 150 as the ntree count, and node size of 20 for final tree building. Following is the specificity on model training data.

RF\_Speci\_Train = CM[4]/(CM[4]+CM[2])

> RF\_Speci\_Train

[1] 0.8862007168

The same reaming on k-fold training model data

RF\_Speci\_KF = mean(as.numeric(cv))

> RF\_Speci\_KF

[1] 0.8396557272

Tested the model against the model test data

RF\_Speci\_Test = CM[4]/(CM[4]+CM[2]) # overfitted

> RF\_Speci\_Test

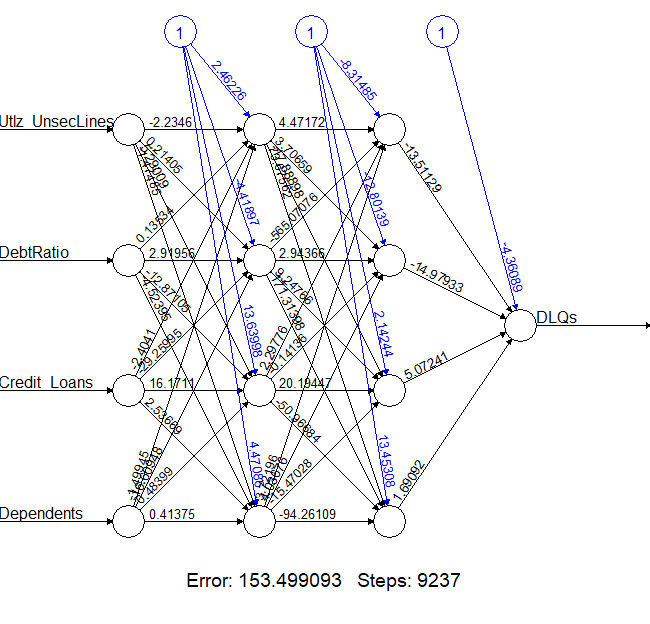
[1] 0.04984423676

The specificity has dropped by 40%, so due to overfitting this model is not used on testing the original test data.

* Artificial Neural Networks’

As there are 3K data observations we have chosen two-layer neural network, error function is Sum of Square errors, threshold improvement is 0.05 with max of 1 lakh iterations, and a nonlinear output as earlier out linear based logistic model has failed due to under fitting. The neurons in each layer is chosen as 4,4 this based on multiple runs, brute-force, started from root(#featurs) i.e. 2 and achieved 4,4 combination yielded the best results on training model dataset.

Below is the neural net



Training model test results on specificity is

ANN\_Speci\_Train = CM[4]/(CM[4]+CM[2])

> ANN\_Speci\_Train

[1] 0.8243727599

We check this with model test data

ANN\_Speci\_Test = CM[4]/(CM[4]+CM[2])

> ANN\_Speci\_Test # overfitted

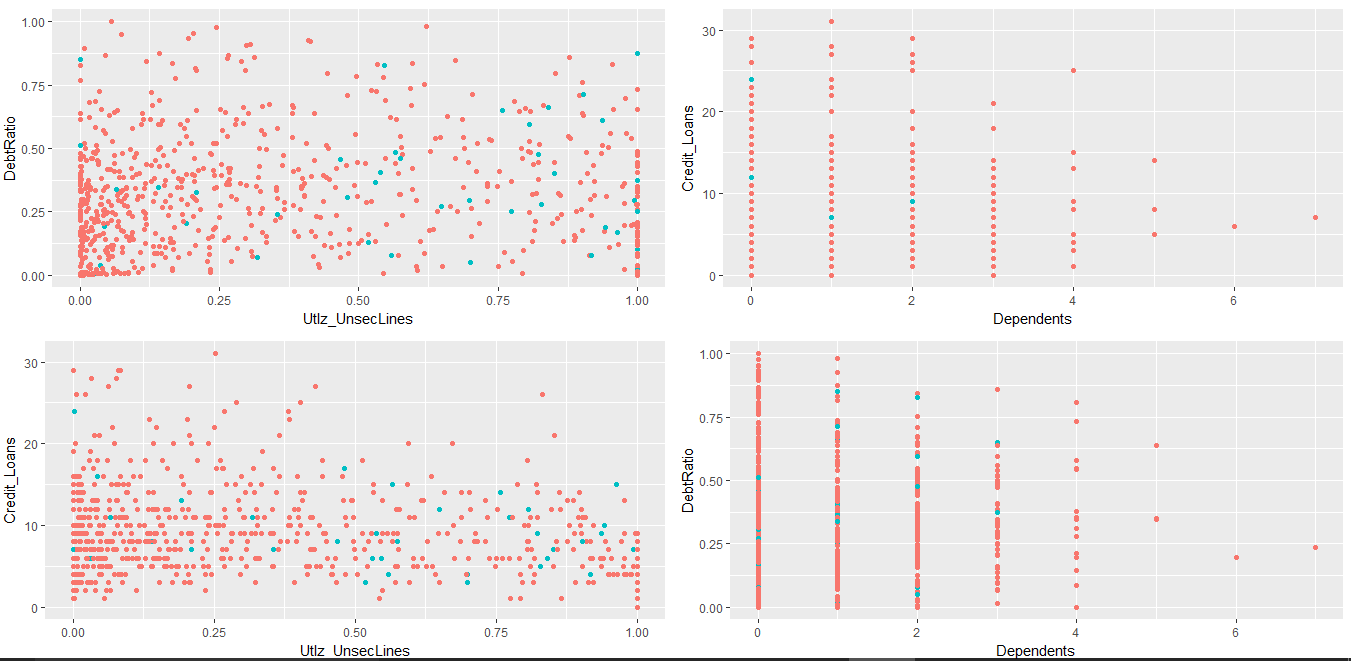
[1] 0.3831775701

There is a 40% drop in specificity saying the model is over fitted. So this is not moved to test on original test data.

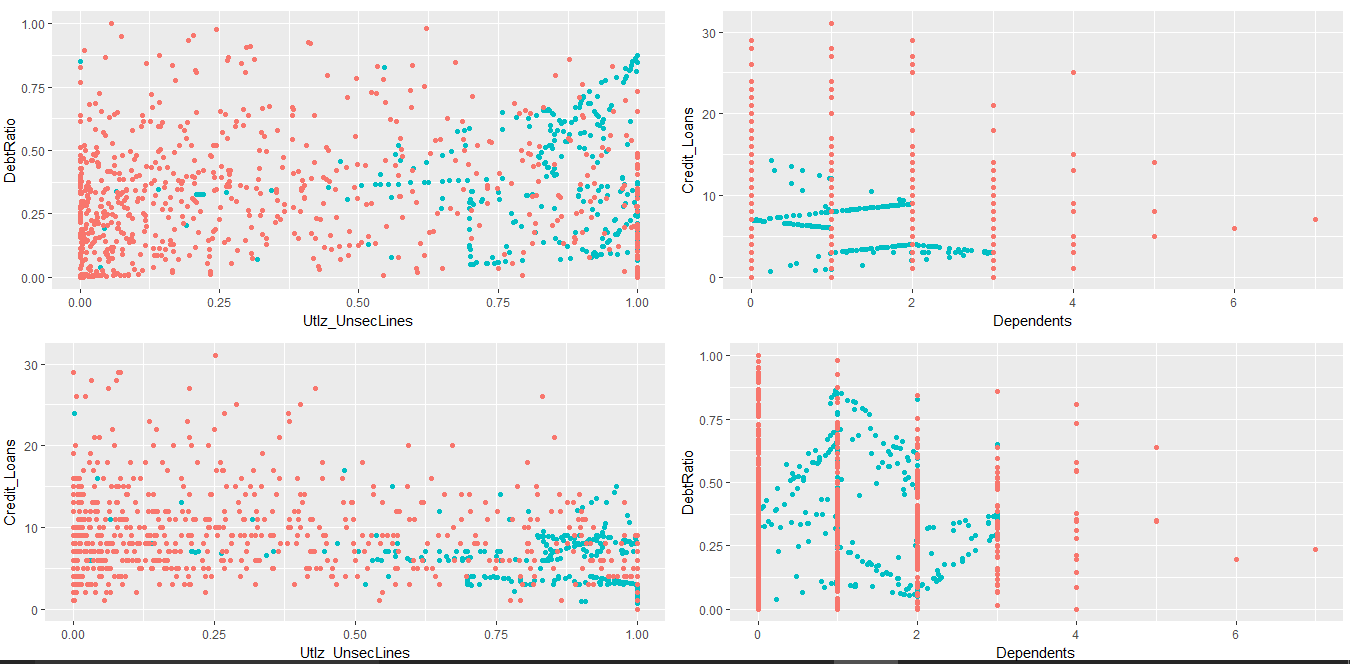
**Test data Preparation**

The test data is followed the same process like Missing data handling by KNN, border line SMOTE, Scaling. Below are few important results.

Original data:

****

Border line SMOTE data:



T2\_testdata\_BS = Boderline\_SMOTE\_fitting(T2\_testdata,14)

[1] "Borderline-SMOTE done"

Boarderline SMOTE data Target variable ratio:

1 2

0.72 0.28

Original data set Target ratio:

0 1

0.94 0.06

**Model Testing on Test data**

As out of all the models build Naïve Bayes Classification is the only stable model found so we have tested this on original test data.

Instead of taking the old model which was built on part of T2\_Training data, we have rebuilt the same model on complete T2\_Training data and used against the T2\_Test data.

T2\_traindata\_Complete\_BS\_Scaled = rbind(T2\_traindata\_Train\_BS\_Scaled,

+ T2\_traindata\_Test\_BS\_Scaled)

> T2\_traindata\_Complete\_BS\_Scaled = rbind(T2\_traindata\_Train\_BS\_Scaled,

+ T2\_traindata\_Test\_BS\_Scaled)

> T2\_NB = naiveBayes(x = T2\_traindata\_Complete\_BS\_Scaled[-1],

+ y = T2\_traindata\_Complete\_BS\_Scaled$DLQs)

> y\_pred = predict(T2\_NB, newdata = T2\_testdata\_BS\_Scaled[-1])

> CM = table(T2\_testdata\_BS\_Scaled[,1],y\_pred)

> NB\_Speci\_Hold = CM[4]/(CM[4]+CM[2])

> NB\_Speci\_Hold

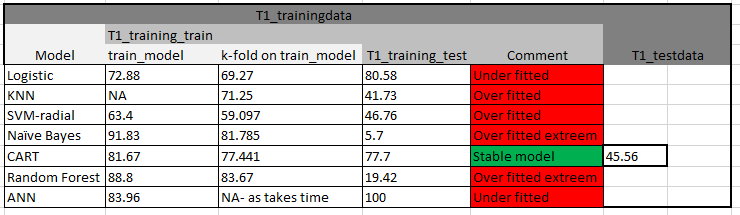
[1] 0.7933333333

Resulted in 79.33 % Specificity on Test data!!!

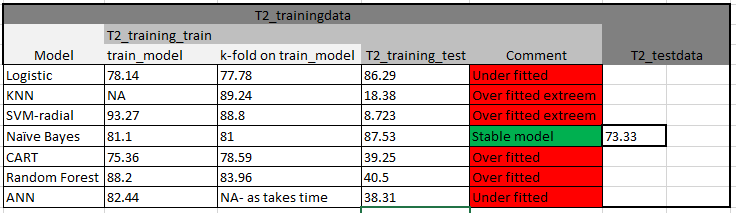
The similar approach has taken on T1\_data set too – (the one in the 1st step split due to outliers)

Below are the numerical results that we achieved on that data set.

**T1\_data – Outliers**



**T2\_data – Non Outliers**



**Conclusion:**

* CART can be deployed for outlier and Naïve Bayes model for non-outlier’s data.
* Breaking the outliers and modelling gives us the benefit of fit higher specificity based model.
* To improve the specify of the outlier model we can include some more feature variable based on the business, with limited knowledge on the domain we didn’t include any new features. As an extension we can try root(feature) and build model to improve the outlier specificity as these are high distance apart points root would reduce the distance and help in model improvement.

**R-Code**

|  |
| --- |
| ## Process followed. Considered the Test data as future data to see how our best model with stands to unknown data.  ## The Train data is used to build, tune, compare the models and give the best to production (handle the test data)  ## So the train data is splited, handle the missing data, SMOTE it, SCALE it and then feature it, in the same order before building the models.  require(readxl) ## read Excel Files  require(dplyr) ## data manupulation  require(usdm) ## VIF  require(ggplot2) ## Visualization  require(caTools) ## split  require(class) ## KNN  require(DMwR) ## SMOTE  require(smotefamily) ## BoderLine SMOTE  require(caret) ## K-Fold, tuning  require(e1071) ## SVM  require(rpart) ## CART - Decision Tree  require(randomForest) ## RandomForest  require(neuralnet) ## ANN  require(gridExtra) ## Multiple plots in single pannel  # reading the data ----  excel\_sheets(path = 'training.xlsx')  traindata = read\_excel(path = 'training.xlsx', sheet = 'training')  excel\_sheets('test.xlsx')  testdata = read\_excel(path = 'test.xlsx', sheet = 'test')  # rename cols, new features, data type adjustments ----  Data\_Building <- function(data){  colnames(data) = c('RowID','DLQs','Utlz\_UnsecLines','DebtRatio',  'Credit\_Loans','Dependents')  data = data %>%  dplyr::select(DLQs,Utlz\_UnsecLines,DebtRatio,Credit\_Loans,Dependents)  data$UUL\_flag = ifelse(data$Utlz\_UnsecLines>1,1,0)  data$DR\_flag = ifelse(data$DebtRatio>1,1,0)  message('Original data formats:')  print(sapply(data,class))  data$Dependents = ifelse(data$Dependents =='NA',NA,data$Dependents)  data$Dependents = as.numeric(data$Dependents)  data[,c(1,6,7)] = data.frame(lapply(data[,c(1,6,7)],as.factor))  message('Final data formats:')  print(sapply(data,class))  return(data)  }  traindata = Data\_Building(traindata)  testdata = Data\_Building(testdata)  # Outliers Visualization  Boxplot\_outliers <- function(data){  par(mfrow=c(2,2))  B2 = boxplot(as.numeric(data$Utlz\_UnsecLines),main="UnsecuredLines Utilization",col="grey", pars=list(outcol="red"))  B3 = boxplot(as.numeric(data$DebtRatio), main = "DebtRatio",col="grey", pars=list(outcol="red"))  B4 = boxplot(as.numeric(data$Credit\_Loans),main = "Credit\_Loans",col="grey", pars=list(outcol="red"))  B5 = boxplot(as.numeric(data$Dependents), main = "Dependents",col="grey", pars=list(outcol="red"))  par(mfrow=c(1,1))  }  Boxplot\_outliers(traindata)  # seperating into two kinds outliers and normal data ----  T1\_traindata = subset(traindata, UUL\_flag == 1 | DR\_flag == 1)  T2\_traindata = subset(traindata, UUL\_flag == 0 & DR\_flag == 0)  T1\_testdata = subset(testdata, UUL\_flag == 1 | DR\_flag == 1)  T2\_testdata = subset(testdata, UUL\_flag == 0 & DR\_flag == 0)  ## T2 data modeling  # Missing data checking ----  Missing\_data\_Check <- function(data\_set){  NA\_Count = sapply(data\_set,function(y) sum(length(which(is.na(y)))))  Null\_Count = sapply(data\_set,function(y) sum(length(which(is.null(y)))))  Length0\_Count = sapply(data\_set,function(y) sum(length(which(length(y)==0))))  Empty\_Count = sapply(data\_set,function(y) sum(length(which(y==''))))  Total\_NonData = NA\_Count+Null\_Count+Length0\_Count+Empty\_Count  return( Total\_NonData )  }  Missing\_data\_Check(T2\_traindata)  # Splitting the training data (SET1: build and tune the model) ~ (SET2: test the model) ----  set.seed(123)  split = sample.split(T2\_traindata$Dependents, SplitRatio = 0.75)  T2\_traindata\_Train = subset(T2\_traindata, split == TRUE)  T2\_traindata\_Test = subset(T2\_traindata, split == FALSE)  # Missing data handling ----  Missing\_data\_handling <- function(data){  print(vif(data.frame(data[,c(2:4)])))  data\_C = subset(data,!is.na(Dependents))  data\_M = subset(data,is.na(Dependents))    set.seed(123)  split = sample.split(data\_C$Dependents, SplitRatio = 0.75)  data\_C\_Tr = subset(data\_C, split == TRUE)  data\_C\_Te = subset(data\_C, split == FALSE)  dependents = knn(train = scale(data\_C\_Tr[,c(2,3,4)]),  test = scale(data\_C\_Te[,c(2,3,4)]),  cl = as.factor(data\_C\_Tr$Dependents),  k = 9,  prob = F)  message(paste0('KNN Accuracy: ',round(length(which(data\_C\_Te$Dependents == dependents))/length(dependents),2)))  model = lm(Dependents~Utlz\_UnsecLines+DebtRatio+Credit\_Loans,  data=data\_C\_Tr)  dependents = round(predict(model,data\_C\_Te)) ## LR is not working as it gives all as 1  message(paste0('Linear regression Accuracy: ',round(length(which(data\_C\_Te$Dependents == dependents))/length(dependents),2)))  rm(list = c('data\_C\_Te','data\_C\_Tr'))  set.seed(1234)  dependents = knn(train = scale(data\_C[,c(2,3,4)]),  test = scale(data\_M[,c(2,3,4)]),  cl = as.factor(data\_C$Dependents),  k = 9,  prob = F)  data\_M$Dependents = dependents  data = rbind(data\_C,data\_M)  rm(list = c('data\_C','data\_M'))  message('Missing data in each column after handling:')  message(paste0(Missing\_data\_Check(data)))  data$Dependents = as.numeric(data$Dependents)  return(data)  }  T2\_traindata\_Test = Missing\_data\_handling(T2\_traindata\_Test)  T2\_traindata\_Train = Missing\_data\_handling(T2\_traindata\_Train)  # Target variable ratio check ----  Two\_D\_View <- function(data){  P1 = ggplot(data = data)+  geom\_point(aes(x = Utlz\_UnsecLines, y = DebtRatio,  color = DLQs),show.legend = F)  P2 = ggplot(data = data)+  geom\_point(aes(x = Dependents, y = Credit\_Loans,  color = DLQs),show.legend = F)  P3 = ggplot(data = data)+  geom\_point(aes(x = Utlz\_UnsecLines, y = Credit\_Loans,  color = DLQs),show.legend = F)  P4 = ggplot(data = data)+  geom\_point(aes(x = Dependents, y = DebtRatio,  color = DLQs),show.legend = F)  grid.arrange(P1, P2, P3,P4, ncol = 2, nrow = 2)  }  Target\_Ratio\_Check <- function(data){  data$Dependents = as.numeric(data$Dependents)  message('Target ratio split:')  print(table(data$DLQs))  message('Target ratio:')  print(round(table(data$DLQs)[1]/sum(table(data$DLQs)),2))  Two\_D\_View(data)  }  dev.off()  Target\_Ratio\_Check(T2\_traindata\_Test) #95:5  Target\_Ratio\_Check(T2\_traindata\_Train) #94:6  # SMOTE for treating imbalance data set ----  SMOTE\_fitting <- function(data,o,u){  data\_SMOTE = DMwR::SMOTE(DLQs~Utlz\_UnsecLines+DebtRatio+Credit\_Loans+Dependents,  as.data.frame(data),perc.over = o,perc.under = u)  message('Original data ratio:')  print(round(table(data$DLQs)/length(data$DLQs),2))  message('SMOTEd data ratio:')  print(table(data\_SMOTE$DLQs)/length(data\_SMOTE$DLQs))  message('SMOTEd data split')  print(table(data\_SMOTE$DLQs))  Two\_D\_View(data\_SMOTE)  return(data\_SMOTE)  }  Two\_D\_View(T2\_traindata\_Test)  T2\_traindata\_Test\_SMOTEd = SMOTE\_fitting(T2\_traindata\_Test,600,300)  Two\_D\_View(T2\_traindata\_Train)  T2\_traindata\_Train\_SMOTEd = SMOTE\_fitting(T2\_traindata\_Train,600,300)  # SMOTE has oversampled the major class area too - so trying borderline SMOTE ----  Boderline\_SMOTE\_fitting <- function(data,i){  set.seed(1234)  data\_SMOTE\_B = BLSMOTE(as.data.frame(data[2:5]),as.numeric(data$DLQs),  K=4,C=3,dupSize=i,method =c("type1"))  message('Boarderline SMOTE data Target variable ratio:')  print(round(table(data\_SMOTE\_B$data$class)/length(data\_SMOTE\_B$data$class),2))  message('Original data set Target ratio:')  print(round(table(data$DLQs)/length(data$DLQs),2))  data\_SMOTE\_BS = data\_SMOTE\_B$data  data\_SMOTE\_BS$DLQs = ifelse(data\_SMOTE\_BS$class == 1, 0, 1)  data\_SMOTE\_BS = data\_SMOTE\_BS[,c(6,1,2,3,4)]  data\_SMOTE\_BS$DLQs = as.factor(data\_SMOTE\_BS$DLQs)  Two\_D\_View(data\_SMOTE\_BS)  return(data\_SMOTE\_BS)  }  Two\_D\_View(T2\_traindata\_Test)  T2\_traindata\_Test\_BS = Boderline\_SMOTE\_fitting(T2\_traindata\_Test,25) #70:30  Two\_D\_View(T2\_traindata\_Train)  T2\_traindata\_Train\_BS = Boderline\_SMOTE\_fitting(T2\_traindata\_Train,25)  rm(list = c('T2\_traindata\_Test\_SMOTEd','T2\_traindata\_Train\_SMOTEd')) # Removing as SMOTE has overfitted the majored regions too  # Building a Scaled data set for classification models ----  Scaling <- function(data){  data\_scaled = data  data\_scaled[-1] = scale(data\_scaled[-1])  return(data\_scaled)  }  T2\_traindata\_Test\_BS\_Scaled = Scaling(T2\_traindata\_Test\_BS)  T2\_traindata\_Train\_BS\_Scaled = Scaling(T2\_traindata\_Train\_BS)  # Logistic regression -- Specificity: Train - 78.14 K-fold Train - 77.78 Test - 86.29 ----  T2\_LR = glm( formula = DLQs~.,  family = binomial,  data = T2\_traindata\_Train\_BS\_Scaled)  prob\_pred = predict(T2\_LR, type = 'response', newdata = T2\_traindata\_Train\_BS\_Scaled[-1])  y\_pred = ifelse(prob\_pred > 0.55, 1, 0)  CM = table(T2\_traindata\_Train\_BS\_Scaled[,1],y\_pred)  LR\_Speci\_Train = CM[4]/(CM[4]+CM[2])  round(LR\_Speci\_Train\*100,2)  require(lmtest)  lrtest(T2\_LR) # overall test i significant  require(pscl)  pR2(T2\_LR) # 35 - very good McFadden R2  set.seed(1234)  folds = createFolds(T2\_traindata\_Train\_BS\_Scaled$DLQs, k = 10)  cv = lapply(folds, function(x) {  training\_fold = T2\_traindata\_Train\_BS\_Scaled[-x, ]  test\_fold = T2\_traindata\_Train\_BS\_Scaled[x, ]  T2\_LR\_KF = glm( formula = DLQs~.,  family = binomial,  data = training\_fold)  prob\_pred = predict(T2\_LR\_KF, type = 'response', newdata = test\_fold[-1])  y\_pred = ifelse(prob\_pred > 0.55, 1, 0)  CM = table(test\_fold[,1],y\_pred)  temp = CM[4]/(CM[4]+CM[2])  return(temp)  })  LR\_Speci\_KF = mean(as.numeric(cv))  round(LR\_Speci\_KF\*100,2)  prob\_pred = predict(T2\_LR, type = 'response', newdata = T2\_traindata\_Test\_BS\_Scaled[-1])  y\_pred = ifelse(prob\_pred > 0.55, 1, 0)  CM = table(T2\_traindata\_Test\_BS\_Scaled[,1],y\_pred)  LR\_Speci\_Test = CM[4]/(CM[4]+CM[2])  round(LR\_Speci\_Test\*100,2) # underfitted  # KNN Classification -- Specificity:Train - xxxxx K-fold Train - 89.24 Test 18.38 ----  set.seed(1234)  caret\_tune = train(form = DLQs~ ., data = T2\_traindata\_Train\_BS\_Scaled, method = 'knn')  caret\_tune  caret\_tune$bestTune # caret to tune for k value  y\_pred = knn(train =T2\_traindata\_Train\_BS\_Scaled[,-1],  test =T2\_traindata\_Test\_BS\_Scaled[,-1],  cl = T2\_traindata\_Train\_BS\_Scaled[, 1],  k = 7,  prob = TRUE)  CM = table(T2\_traindata\_Test\_BS\_Scaled[,1],y\_pred)  Knn\_Speci\_Test = CM[4]/(CM[4]+CM[2])  Knn\_Speci\_Test  set.seed(1234)  folds = createFolds(T2\_traindata\_Train\_BS\_Scaled$DLQs, k = 10)  cv = lapply(folds, function(x) {  training\_fold = T2\_traindata\_Train\_BS\_Scaled[-x, ]  test\_fold = T2\_traindata\_Train\_BS\_Scaled[x, ]  y\_pred = knn(train =training\_fold[,-1],  test =test\_fold[,-1],  cl = training\_fold[, 1],  k = 7,  prob = TRUE)  CM = table(test\_fold[,1],y\_pred)  temp = CM[4]/(CM[4]+CM[2])  return(temp)  })  Knn\_Speci\_KF = mean(as.numeric(cv)) #overfitted  # SVM Classification -- Specificity:Train - 93.27 K-fold Train - 88.8 Test 8.723 ----  set.seed(1234)  caret\_tune = train(form = DLQs~ ., data = T2\_traindata\_Train\_BS\_Scaled, method = 'svmLinearWeights')  caret\_tune  caret\_tune$bestTune # caret to tune for cost and weight value - cost is 0.25  set.seed(1234)  tune\_svm\_kernal = tune(svm, DLQs~ ., data = T2\_traindata\_Train\_BS\_Scaled,  kernal = 'radial',  ranges = list(cost = c(0.1,0.4,0.8,1,3,5,10,50,100), # penalising factor for missclassification, high c => low bias, high viariance, default is 1  gamma = c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,2,4))) # smoothening the boundary shape sharpness, low gama => pointy bounday, low bias, high variance, default 1/dimensions  summary(tune\_svm\_kernal) # tuned parameters says cost 3 and gamma 4  set.seed(1234)  tune\_svm\_kernal = tune(svm, DLQs~ ., data = T2\_traindata\_Train\_BS\_Scaled,  kernal = 'sigmoid',  ranges = list(cost = c(0.1,0.4,0.8,1,3,5,10,50,100),  gamma = c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,2,4)))  summary(tune\_svm\_kernal) # tuned parameters says cost 3 and gamma 4  set.seed(1234)  tune\_svm\_kernal = tune(svm, DLQs~ ., data = T2\_traindata\_Train\_BS\_Scaled,  kernal = 'polynomial',  ranges = list(ccost = c(0.1,0.4,0.8,1,3,5,10,50,100),  gamma = c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,2,4),  degree = c(2,3,4,5,6)))  summary(tune\_svm\_kernal) # tuned parameters says cost 0.1 and gamma 4 and degree 2  for(svmType in c('C-classification','nu-classification')){  for(svmKernal in c('linear','polynomial','radial','sigmoid')){  set.seed(1234)  folds = createFolds(T2\_traindata\_Train\_BS\_Scaled$DLQs, k = 10)  cv = lapply(folds, function(x) {  training\_fold = T2\_traindata\_Train\_BS\_Scaled[-x, ]  test\_fold = T2\_traindata\_Train\_BS\_Scaled[x, ]  if(svmKernal == 'radial'){  T2\_SVM = svm(formula = DLQs ~ .,  data = training\_fold,  type = 'C-classification',  kernel = svmKernal, cost = 3,gamma = 4)  y\_pred = predict(T2\_SVM, newdata = test\_fold[-1])  }else if(svmKernal=='sigmoid'){  T2\_SVM = svm(formula = DLQs ~ .,  data = training\_fold,  type = 'C-classification',  kernel = svmKernal, cost = 3,gamma = 4)  y\_pred = predict(T2\_SVM, newdata = test\_fold[-1])  }else if(svmKernal=='polynomial'){  T2\_SVM = svm(formula = DLQs ~ .,  data = training\_fold,  type = 'C-classification',  kernel = svmKernal, cost = 0.1,gamma = 4 ,degre = 2)  y\_pred = predict(T2\_SVM, newdata = test\_fold[-1])  }else{  T2\_SVM = svm(formula = DLQs ~ .,  data = training\_fold,  type = 'C-classification',  kernel = svmKernal, cost =0.25)  y\_pred = predict(T2\_SVM, newdata = test\_fold[-1])  }  CM = table(test\_fold[,1],y\_pred)  temp = CM[4]/(CM[4]+CM[2])  return(temp)  })  specificity\_SVM = round(mean(as.numeric(cv)),5)\*100  print.noquote(paste0(svmKernal,'-kernal ',svmType,' has K-fold specificity of ',specificity\_SVM))  }  } # choose radial kernal with C-Classification as it has highest 88.8  # [1] linear-kernal C-classification has K-fold specificity of 83.961  # [1] polynomial-kernal C-classification has K-fold specificity of 5.446  # [1] radial-kernal C-classification has K-fold specificity of 88.8  # [1] sigmoid-kernal C-classification has K-fold specificity of 37.542  # [1] linear-kernal nu-classification has K-fold specificity of 83.961  # [1] polynomial-kernal nu-classification has K-fold specificity of 5.446  # [1] radial-kernal nu-classification has K-fold specificity of 88.8  # [1] sigmoid-kernal nu-classification has K-fold specificity of 37.542  T2\_SVM = svm(formula = DLQs ~ .,  data = T2\_traindata\_Train\_BS\_Scaled,  type = 'C-classification',  kernel = 'radial', cost= 3, gamma= 4)  y\_pred = predict(T2\_SVM, newdata = T2\_traindata\_Train\_BS\_Scaled[-1])  CM = table(T2\_traindata\_Train\_BS\_Scaled[,1],y\_pred)  SVM\_Speci\_Train = CM[4]/(CM[4]+CM[2])  y\_pred = predict(T2\_SVM, newdata = T2\_traindata\_Test\_BS\_Scaled[-1])  CM = table(T2\_traindata\_Test\_BS\_Scaled[,1],y\_pred)  SVM\_Speci\_Test = CM[4]/(CM[4]+CM[2]) # dropped in Test, overfitted, but will consider for Test set, as k-fold is close to train\_test  # Naive Bayes -- Specificity:Train - 81.1 K-fold Train - 81.0 Test 87.53 ----  set.seed(1234)  folds = createFolds(T2\_traindata\_Train\_BS\_Scaled$DLQs, k = 10)  cv = lapply(folds, function(x) {  training\_fold = T2\_traindata\_Train\_BS\_Scaled[-x, ]  test\_fold = T2\_traindata\_Train\_BS\_Scaled[x, ]  T2\_NB = naiveBayes(x = training\_fold[-1],  y = training\_fold$DLQs)  y\_pred = predict(T2\_NB, newdata = test\_fold[-1])  CM = table(test\_fold[,1],y\_pred)  temp = CM[4]/(CM[4]+CM[2])  return(temp)  })  NB\_Speci\_KF = round(mean(as.numeric(cv)),5)\*100  T2\_NB = naiveBayes(x = T2\_traindata\_Train\_BS\_Scaled[-1],  y = T2\_traindata\_Train\_BS\_Scaled$DLQs)  y\_pred = predict(T2\_NB, newdata = T2\_traindata\_Train\_BS\_Scaled[-1])  CM = table(T2\_traindata\_Train\_BS\_Scaled[,1],y\_pred)  NB\_Speci\_Train = CM[4]/(CM[4]+CM[2])  y\_pred = predict(T2\_NB, newdata = T2\_traindata\_Test\_BS\_Scaled[-1])  CM = table(T2\_traindata\_Test\_BS\_Scaled[,1],y\_pred)  NB\_Speci\_Test = CM[4]/(CM[4]+CM[2]) # moves to predict against test  # CART -- Specificity:Train - 75.36 K-fold Train - 78.59 Test 39.25 --------  caret\_tune = train(form = DLQs~ ., data = T2\_traindata\_Train\_BS\_Scaled, method = 'rpart')  caret\_tune  caret\_tune$bestTune # CP - Tunning  T2\_CART\_temp = rpart(formula = DLQs ~ .,  data = T2\_traindata\_Train\_BS\_Scaled,  method = "class",  minsplit= 225,  cp = 0,  xval = 7)  printcp(T2\_CART\_temp)  plotcp(T2\_CART\_temp)  T2\_CART = prune(T2\_CART\_temp, cp= 0.03315412186 ,"CP")  y\_pred = predict(T2\_CART, newdata = T2\_traindata\_Train\_BS\_Scaled[-1], type='class')  CM = table(T2\_traindata\_Train\_BS\_Scaled[,1],y\_pred)  CART\_Speci\_Train = CM[4]/(CM[4]+CM[2])  set.seed(1234)  folds = createFolds(T2\_traindata\_Train\_BS\_Scaled$DLQs, k = 10)  cv = lapply(folds, function(x) {  training\_fold = T2\_traindata\_Train\_BS\_Scaled[-x, ]  test\_fold = T2\_traindata\_Train\_BS\_Scaled[x, ]  T2\_CART\_temp = rpart(formula = DLQs ~ .,  data = training\_fold,  method = "class",  minsplit= 225,  cp = 0.05284974,  xval = 7)  y\_pred = predict(T2\_CART\_temp, newdata = test\_fold[-1], type='class')  CM = table(test\_fold[,1],y\_pred)  temp = CM[4]/(CM[4]+CM[2])  return(temp)  })  CART\_Speci\_KF = round(mean(as.numeric(cv)),5)\*100  y\_pred = predict(T2\_CART, newdata = T2\_traindata\_Test\_BS\_Scaled[-1],type='class')  CM = table(T2\_traindata\_Test\_BS\_Scaled[,1],y\_pred)  CART\_Speci\_Test = CM[4]/(CM[4]+CM[2]) # overfitted  # Random Forest -- Specificity:Train - 88.2 K-fold Train - 83.96 Test 40.5 ------  set.seed(1234)  T2\_RF = randomForest(DLQs ~ ., data = T2\_traindata\_Train\_BS\_Scaled,  ntree=500, mtry = 2, nodesize = 40,  importance=TRUE)  plot(T2\_RF) ## 150 tree from OOB  caret\_tune = train(form = DLQs~ ., data = T2\_traindata\_Train\_BS\_Scaled, method = 'rf')  caret\_tune  caret\_tune$bestTune # mtry - Tunning  set.seed(1234)  T2\_RF = tuneRF(x = T2\_traindata\_Train\_BS\_Scaled[,-1],  y=T2\_traindata\_Train\_BS\_Scaled$DLQs,  mtryStart = 2,  ntreeTry=150,  stepFactor = 1, ## 1st try 2 variables, next 4 , next 5 , next 6 MtryStart\*Stepfactor  improve = 0.001, ## delta OOB  trace=TRUE,  plot = TRUE,  doBest = TRUE,  nodesize = 20,  importance=TRUE  ) # random Forest tuning also lead to mtry = 2  y\_pred = predict(T2\_RF, newdata = T2\_traindata\_Train\_BS\_Scaled[-1], type='class')  CM = table(T2\_traindata\_Train\_BS\_Scaled[,1],y\_pred)  RF\_Speci\_Train = CM[4]/(CM[4]+CM[2])  set.seed(1234)  folds = createFolds(T2\_traindata\_Train\_BS\_Scaled$DLQs, k = 10)  cv = lapply(folds, function(x) {  training\_fold = T2\_traindata\_Train\_BS\_Scaled[-x, ]  test\_fold = T2\_traindata\_Train\_BS\_Scaled[x, ]  T2\_RF\_temp = randomForest(DLQs ~ ., data = training\_fold,  ntree=150, mtry = 2, nodesize = 40)  y\_pred = predict(T2\_RF\_temp, newdata = test\_fold[-1], type='class')  CM = table(test\_fold[,1],y\_pred)  temp = CM[4]/(CM[4]+CM[2])  return(temp)  })  RF\_Speci\_KF = round(mean(as.numeric(cv)),5)\*100  y\_pred = predict(T2\_RF, newdata = T2\_traindata\_Test\_BS\_Scaled[-1], type='class')  CM = table(T2\_traindata\_Test\_BS\_Scaled[,1],y\_pred)  RF\_Speci\_Test = CM[4]/(CM[4]+CM[2]) # overfitted  # ANN -- Specificity:Train - 82.44 K-fold Train - xxxxxx Test 38.31 ----  training\_set\_scaled\_ANN = T2\_traindata\_Train\_BS\_Scaled  training\_set\_scaled\_ANN$DLQs = as.numeric(as.character(training\_set\_scaled\_ANN$DLQs))  test\_set\_scaled\_ANN = T2\_traindata\_Test\_BS\_Scaled  test\_set\_scaled\_ANN$DLQs = as.numeric(as.character(test\_set\_scaled\_ANN$DLQs))  n = names(training\_set\_scaled\_ANN)  long\_formula = as.formula(paste("DLQs ~", paste(n[!n %in% "DLQs"], collapse = " + ")))  set.seed(123)  T2\_ANN = neuralnet(formula = long\_formula,  data = training\_set\_scaled\_ANN,  hidden = c(4,4),  err.fct = "sse",  linear.output = FALSE,  lifesign = "full",  lifesign.step = 1,  threshold = 0.05,  stepmax = 100000)  plot(T2\_ANN)  y\_pred = ifelse(T2\_ANN$net.result[[1]] >= 0.5,1,0)  CM = table(training\_set\_scaled\_ANN[,1],y\_pred)  ANN\_Speci\_Train = CM[4]/(CM[4]+CM[2])  ANN\_Speci\_Train  y\_pred = compute(T2\_ANN,test\_set\_scaled\_ANN[,-1])  y\_pred = ifelse(y\_pred$net.result >= 0.5,1,0)  CM = table(test\_set\_scaled\_ANN[,1],y\_pred)  ANN\_Speci\_Test = CM[4]/(CM[4]+CM[2])  ANN\_Speci\_Test # overfitted  # Test data prep LR: 57.67 KNN: 46 SVM:31 NB: 73.33 CART: 75.67 RF: 47.34 ANN: 58.667 ----  Missing\_data\_Check(T2\_testdata)  data\_C = subset(T2\_testdata,!is.na(Dependents))  data\_M = subset(T2\_testdata,is.na(Dependents))  dependents = knn(train = scale(data\_C[,c(2,3,4)]),  test = as.matrix(cbind(scale(data\_M[2]), data\_M[3], scale(data\_M[4]))),  cl = as.factor(data\_C$Dependents),  k = 9,  prob = F)  data\_M$Dependents = dependents  T2\_testdata = rbind(data\_C,data\_M)  rm(list = c('data\_C','data\_M'))  Missing\_data\_Check(T2\_testdata)  T2\_testdata$Dependents = as.numeric(T2\_testdata$Dependents)  Two\_D\_View(T2\_testdata)  set.seed(1234)  T2\_testdata\_BS = Boderline\_SMOTE\_fitting(T2\_testdata,14)  T2\_testdata\_BS\_Scaled = Scaling(T2\_testdata\_BS)  T2\_traindata\_Complete\_BS\_Scaled = rbind(T2\_traindata\_Train\_BS\_Scaled,  T2\_traindata\_Test\_BS\_Scaled)  T2\_NB = naiveBayes(x = T2\_traindata\_Complete\_BS\_Scaled[-1],  y = T2\_traindata\_Complete\_BS\_Scaled$DLQs)  y\_pred = predict(T2\_NB, newdata = T2\_testdata\_BS\_Scaled[-1])  CM = table(T2\_testdata\_BS\_Scaled[,1],y\_pred)  NB\_Speci\_Hold = CM[4]/(CM[4]+CM[2]) #79.33  # T1 data  # seperating into two kinds outliers and normal data ----  T1\_traindata = subset(traindata, UUL\_flag == 1 | DR\_flag == 1)  T1\_testdata = subset(testdata, UUL\_flag == 1 | DR\_flag == 1)  ## T1 data modeling  # Missing data checking ----  Missing\_data\_Check(T1\_traindata)  # Splitting the training data (SET1: build and tune the model) ~ (SET1: test the model) ----  set.seed(123)  split = sample.split(T1\_traindata$Dependents, SplitRatio = 0.75)  T1\_traindata\_Train = subset(T1\_traindata, split == TRUE)  T1\_traindata\_Test = subset(T1\_traindata, split == FALSE)  # Missing data handling ----  T1\_traindata\_Test = Missing\_data\_handling(T1\_traindata\_Test)  Missing\_data\_Check(T1\_traindata\_Test)  T1\_traindata\_Train = Missing\_data\_handling(T1\_traindata\_Train)  Missing\_data\_Check(T1\_traindata\_Train)  # Target variable ratio check ----  Target\_Ratio\_Check(T1\_traindata\_Test) #93:7  Target\_Ratio\_Check(T1\_traindata\_Train) #92:8  # SMOTE for treating imbalance data set ----  Two\_D\_View(T1\_traindata\_Test)  T1\_traindata\_Test\_SMOTEd = SMOTE\_fitting(T1\_traindata\_Test,600,300) #72:28  Two\_D\_View(T1\_traindata\_Train)  T1\_traindata\_Train\_SMOTEd = SMOTE\_fitting(T1\_traindata\_Train,600,300) #72:28  # SMOTE has oversampled the major class area too - so trying boundary SMOTE ----  Two\_D\_View(T1\_traindata\_Test)  T1\_traindata\_Test\_BS = Boderline\_SMOTE\_fitting(T1\_traindata\_Test,8) #73:27  Two\_D\_View(T1\_traindata\_Train)  T1\_traindata\_Train\_BS = Boderline\_SMOTE\_fitting(T1\_traindata\_Train,15) #71:29  rm(list = c('T1\_traindata\_Test\_SMOTEd','T1\_traindata\_Train\_SMOTEd')) # Removing as SMOTE has overfitted the majored regions too  # Building a Scaled data set for classification models ----  T1\_traindata\_Test\_BS\_Scaled = Scaling(T1\_traindata\_Test\_BS)  T1\_traindata\_Train\_BS\_Scaled = Scaling(T1\_traindata\_Train\_BS)  # Logistic regression -- Specificity: Train - 72.88 K-fold Train - 69.27 Test - 80.58 ----  T1\_LR = glm( formula = DLQs~.,  family = binomial,  data = T1\_traindata\_Train\_BS\_Scaled)  prob\_pred = predict(T1\_LR, type = 'response', newdata = T1\_traindata\_Train\_BS\_Scaled[-1])  y\_pred = ifelse(prob\_pred > 0.4, 1, 0)  CM = table(T1\_traindata\_Train\_BS\_Scaled[,1],y\_pred)  LR\_Speci\_Train = CM[4]/(CM[4]+CM[2])  round(LR\_Speci\_Train\*100,2)  LR\_Acc = (CM[4]+CM[1])/(CM[4]+CM[1]+CM[2]+CM[3])  LR\_Acc  require(lmtest)  lrtest(T1\_LR) # overall test i significant  require(pscl)  pR2(T1\_LR) # 18 - ok model by McFadden R2  set.seed(1234)  folds = createFolds(T1\_traindata\_Train\_BS\_Scaled$DLQs, k = 10)  cv = lapply(folds, function(x) {  training\_fold = T1\_traindata\_Train\_BS\_Scaled[-x, ]  test\_fold = T1\_traindata\_Train\_BS\_Scaled[x, ]  T1\_LR\_KF = glm( formula = DLQs~.,  family = binomial,  data = training\_fold)  prob\_pred = predict(T1\_LR\_KF, type = 'response', newdata = test\_fold[-1])  y\_pred = ifelse(prob\_pred > 0.4, 1, 0)  CM = table(test\_fold[,1],y\_pred)  temp = CM[4]/(CM[4]+CM[2])  return(temp)  })  LR\_Speci\_KF = mean(as.numeric(cv))  round(LR\_Speci\_KF\*100,2)  prob\_pred = predict(T1\_LR, type = 'response', newdata = T1\_traindata\_Test\_BS\_Scaled[-1])  y\_pred = ifelse(prob\_pred > 0.4, 1, 0)  CM = table(T1\_traindata\_Test\_BS\_Scaled[,1],y\_pred)  LR\_Speci\_Test = CM[4]/(CM[4]+CM[2])  round(LR\_Speci\_Test\*100,2)  # KNN Classification -- Specificity:Train - xxxxx K-fold Train - 71.25 Test 41.73 ----  caret\_tune = train(form = DLQs~ ., data = T1\_traindata\_Train\_BS\_Scaled, method = 'knn')  caret\_tune  caret\_tune$bestTune # caret to tune for k value  y\_pred = knn(train =T1\_traindata\_Train\_BS\_Scaled[,-1],  test =T1\_traindata\_Test\_BS\_Scaled[,-1],  cl = T1\_traindata\_Train\_BS\_Scaled[, 1],  k = 5,  prob = TRUE)  CM = table(T1\_traindata\_Test\_BS\_Scaled[,1],y\_pred)  Knn\_Speci\_Test = CM[4]/(CM[4]+CM[2])  Knn\_Speci\_Test  set.seed(1234)  folds = createFolds(T1\_traindata\_Train\_BS\_Scaled$DLQs, k = 10)  cv = lapply(folds, function(x) {  training\_fold = T1\_traindata\_Train\_BS\_Scaled[-x, ]  test\_fold = T1\_traindata\_Train\_BS\_Scaled[x, ]  y\_pred = knn(train =training\_fold[,-1],  test =test\_fold[,-1],  cl = training\_fold[, 1],  k = 5,  prob = TRUE)  CM = table(test\_fold[,1],y\_pred)  temp = CM[4]/(CM[4]+CM[2])  return(temp)  })  Knn\_Speci\_KF = mean(as.numeric(cv)) #overfitted  # SVM Classification -- Specificity:Train - 63.4 K-fold Train - 59.097 Test 46.76 ----  set.seed(1234)  caret\_tune = train(form = DLQs~ ., data = T1\_traindata\_Train\_BS\_Scaled, method = 'svmLinearWeights')  caret\_tune  caret\_tune$bestTune # caret to tune for cost and weight value - cost is 1 which is default  set.seed(1234)  tune\_svm\_kernal = tune(svm, DLQs~ ., data = T1\_traindata\_Train\_BS\_Scaled,  kernal = 'radial',  ranges = list(cost = c(0.1,0.4,0.8,1,3,5,10,50,100), # penalising factor for missclassification, high c => low bias, high viariance, default is 1  gamma = c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,2,4))) # smoothening the boundary shape sharpness, low gama => pointy bounday, low bias, high variance, default 1/dimensions  summary(tune\_svm\_kernal) # tuned parameters says cost 50 and gamma 0.9  set.seed(1234)  tune\_svm\_kernal = tune(svm, DLQs~ ., data = T1\_traindata\_Train\_BS\_Scaled,  kernal = 'sigmoid',  ranges = list(cost = c(0.1,0.4,0.8,1,3,5,10,50,100),  gamma = c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,2,4)))  summary(tune\_svm\_kernal) # tuned parameters says cost 50 and gamma 0.9  set.seed(1234)  tune\_svm\_kernal = tune(svm, DLQs~ ., data = T1\_traindata\_Train\_BS\_Scaled,  kernal = 'polynomial',  ranges = list(ccost = c(0.1,0.4,0.8,1,3,5,10,50,100),  gamma = c(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,2,4),  degree = c(2,3,4,5,6)))  summary(tune\_svm\_kernal) # tuned parameters says cost 0.1 and gamma 4 and degree 2  for(svmType in c('C-classification','nu-classification')){  for(svmKernal in c('linear','polynomial','radial','sigmoid')){  set.seed(1234)  folds = createFolds(T1\_traindata\_Train\_BS\_Scaled$DLQs, k = 10)  cv = lapply(folds, function(x) {  training\_fold = T1\_traindata\_Train\_BS\_Scaled[-x, ]  test\_fold = T1\_traindata\_Train\_BS\_Scaled[x, ]  if(svmKernal == 'radial'){  T1\_SVM = svm(formula = DLQs ~ .,  data = training\_fold,  type = 'C-classification',  kernel = svmKernal, cost = 50,gamma = 0.9)  y\_pred = predict(T1\_SVM, newdata = test\_fold[-1])  }else if(svmKernal=='sigmoid'){  T1\_SVM = svm(formula = DLQs ~ .,  data = training\_fold,  type = 'C-classification',  kernel = svmKernal, cost = 50 ,gamma = 0.9)  y\_pred = predict(T1\_SVM, newdata = test\_fold[-1])  }else if(svmKernal=='polynomial'){  T1\_SVM = svm(formula = DLQs ~ .,  data = training\_fold,  type = 'C-classification',  kernel = svmKernal, cost =0.1 ,gamma = 4 ,degre = 2)  y\_pred = predict(T1\_SVM, newdata = test\_fold[-1])  }else{  T1\_SVM = svm(formula = DLQs ~ .,  data = training\_fold,  type = 'C-classification',  kernel = svmKernal, cost =1)  y\_pred = predict(T1\_SVM, newdata = test\_fold[-1])  }  CM = table(test\_fold[,1],y\_pred)  temp = CM[4]/(CM[4]+CM[2])  return(temp)  })  specificity\_SVM = round(mean(as.numeric(cv)),5)\*100  print.noquote(paste0(svmKernal,'-kernal ',svmType,' has K-fold specificity of ',specificity\_SVM))  }  } # choose radial kernal with C-Classification as it has highest 59.097  T1\_SVM = svm(formula = DLQs ~ .,  data = T1\_traindata\_Train\_BS\_Scaled,  type = 'C-classification',  kernel = 'radial', cost= 50, gamma= 9)  y\_pred = predict(T1\_SVM, newdata = T1\_traindata\_Train\_BS\_Scaled[-1])  CM = table(T1\_traindata\_Train\_BS\_Scaled[,1],y\_pred)  SVM\_Speci\_Train = CM[4]/(CM[4]+CM[2])  y\_pred = predict(T1\_SVM, newdata = T1\_traindata\_Test\_BS\_Scaled[-1])  CM = table(T1\_traindata\_Test\_BS\_Scaled[,1],y\_pred)  SVM\_Speci\_Test = CM[4]/(CM[4]+CM[2]) # dropped in Test, overfitted, but will consider for Test set, as k-fold is close to train\_test  # Naive Bayes -- Specificity:Train - 91.83 K-fold Train - 81.785 Test 5.7 ----  T1\_NB = naiveBayes(x = T1\_traindata\_Train\_BS\_Scaled[-1],  y = T1\_traindata\_Train\_BS\_Scaled$DLQs)  y\_pred = predict(T1\_NB, newdata = T1\_traindata\_Train\_BS\_Scaled[-1])  CM = table(T1\_traindata\_Train\_BS\_Scaled[,1],y\_pred)  NB\_Speci\_Train = CM[4]/(CM[4]+CM[2])  set.seed(1234)  folds = createFolds(T1\_traindata\_Train\_BS\_Scaled$DLQs, k = 10)  cv = lapply(folds, function(x) {  training\_fold = T1\_traindata\_Train\_BS\_Scaled[-x, ]  test\_fold = T1\_traindata\_Train\_BS\_Scaled[x, ]  T1\_NB = naiveBayes(x = training\_fold[-1],  y = training\_fold$DLQs)  y\_pred = predict(T1\_NB, newdata = test\_fold[-1])  CM = table(test\_fold[,1],y\_pred)  temp = CM[4]/(CM[4]+CM[2])  return(temp)  })  NB\_Speci\_KF = round(mean(as.numeric(cv)),5)\*100  y\_pred = predict(T1\_NB, newdata = T1\_traindata\_Test\_BS\_Scaled[-1])  CM = table(T1\_traindata\_Test\_BS\_Scaled[,1],y\_pred)  NB\_Speci\_Test = CM[4]/(CM[4]+CM[2]) # overfitted  # CART -- Specificity:Train - 81.67 K-fold Train - 77.441 Test 77.7 --------  caret\_tune = train(form = DLQs~ ., data = T1\_traindata\_Train\_BS\_Scaled, method = 'rpart')  caret\_tune  caret\_tune$bestTune # CP - Tunning 0.06535947712  T1\_CART\_temp = rpart(formula = DLQs ~ .,  data = T1\_traindata\_Train\_BS\_Scaled,  method = "class",  minsplit= 25,  cp = 0,  xval = 7)  printcp(T1\_CART\_temp)  plotcp(T1\_CART\_temp)  T1\_CART = prune(T1\_CART\_temp, cp= 0.06535947712 ,"CP")  y\_pred = predict(T1\_CART, newdata = T1\_traindata\_Train\_BS\_Scaled[-1], type='class')  CM = table(T1\_traindata\_Train\_BS\_Scaled[,1],y\_pred)  CART\_Speci\_Train = CM[4]/(CM[4]+CM[2])  set.seed(1234)  folds = createFolds(T1\_traindata\_Train\_BS\_Scaled$DLQs, k = 10)  cv = lapply(folds, function(x) {  training\_fold = T1\_traindata\_Train\_BS\_Scaled[-x, ]  test\_fold = T1\_traindata\_Train\_BS\_Scaled[x, ]  T1\_CART\_temp = rpart(formula = DLQs ~ .,  data = training\_fold,  method = "class",  minsplit= 25,  cp = 0.06535947712,  xval = 7)  y\_pred = predict(T1\_CART\_temp, newdata = test\_fold[-1], type='class')  CM = table(test\_fold[,1],y\_pred)  temp = CM[4]/(CM[4]+CM[2])  return(temp)  })  CART\_Speci\_KF = round(mean(as.numeric(cv)),5)\*100  y\_pred = predict(T1\_CART, newdata = T1\_traindata\_Test\_BS\_Scaled[-1],type='class')  CM = table(T1\_traindata\_Test\_BS\_Scaled[,1],y\_pred)  CART\_Speci\_Test = CM[4]/(CM[4]+CM[2]) # moves to build the test solution  # Random Forest -- Specificity:Train - 88.8 K-fold Train - 83.67 Test 19.42 ------  set.seed(1234)  T1\_RF = randomForest(DLQs ~ ., data = T1\_traindata\_Train\_BS\_Scaled,  ntree=500, mtry = 2, nodesize = 20,  importance=TRUE)  plot(T1\_RF) ## 150 tree from OOB  caret\_tune = train(form = DLQs~ ., data = T1\_traindata\_Train\_BS\_Scaled, method = 'rf')  caret\_tune  caret\_tune$bestTune # mtry - Tunning  set.seed(1234)  T1\_RF = tuneRF(x = T1\_traindata\_Train\_BS\_Scaled[,-1],  y=T1\_traindata\_Train\_BS\_Scaled$DLQs,  mtryStart = 2,  ntreeTry=150,  stepFactor = 1, ## 1st try 2 variables, next 4 , next 5 , next 6 MtryStart\*Stepfactor  improve = 0.001, ## delta OOB  trace=TRUE,  plot = TRUE,  doBest = TRUE,  nodesize = 20,  importance=TRUE  ) # random Forest tuning also lead to mtry = 2  y\_pred = predict(T1\_RF, newdata = T1\_traindata\_Train\_BS\_Scaled[-1], type='class')  CM = table(T1\_traindata\_Train\_BS\_Scaled[,1],y\_pred)  RF\_Speci\_Train = CM[4]/(CM[4]+CM[2])  set.seed(1234)  folds = createFolds(T1\_traindata\_Train\_BS\_Scaled$DLQs, k = 10)  cv = lapply(folds, function(x) {  training\_fold = T1\_traindata\_Train\_BS\_Scaled[-x, ]  test\_fold = T1\_traindata\_Train\_BS\_Scaled[x, ]  T1\_RF\_temp = randomForest(DLQs ~ ., data = training\_fold,  ntree=150, mtry = 2, nodesize = 20)  y\_pred = predict(T1\_RF\_temp, newdata = test\_fold[-1], type='class')  CM = table(test\_fold[,1],y\_pred)  temp = CM[4]/(CM[4]+CM[2])  return(temp)  })  RF\_Speci\_KF = round(mean(as.numeric(cv)),5)\*100  y\_pred = predict(T1\_RF, newdata = T1\_traindata\_Test\_BS\_Scaled[-1], type='class')  CM = table(T1\_traindata\_Test\_BS\_Scaled[,1],y\_pred)  RF\_Speci\_Test = CM[4]/(CM[4]+CM[2]) # overfitted  # ANN -- Specificity:Train - 83.96 K-fold Train - xxxxxx Test 100 ----  training\_set\_scaled\_ANN = T1\_traindata\_Train\_BS\_Scaled  training\_set\_scaled\_ANN$DLQs = as.numeric(as.character(training\_set\_scaled\_ANN$DLQs))  test\_set\_scaled\_ANN = T1\_traindata\_Test\_BS\_Scaled  test\_set\_scaled\_ANN$DLQs = as.numeric(as.character(test\_set\_scaled\_ANN$DLQs))  n = names(training\_set\_scaled\_ANN)  long\_formula = as.formula(paste("DLQs ~", paste(n[!n %in% "DLQs"], collapse = " + ")))  set.seed(123)  T1\_ANN = neuralnet(formula = long\_formula,  data = training\_set\_scaled\_ANN,  hidden = c(4),  err.fct = "sse",  linear.output = FALSE,  lifesign = "full",  lifesign.step = 1,  threshold = 0.05,  stepmax = 100000)  plot(T1\_ANN)  y\_pred = ifelse(T1\_ANN$net.result[[1]] >= 0.5,1,0)  CM = table(training\_set\_scaled\_ANN[,1],y\_pred)  ANN\_Speci\_Train = CM[4]/(CM[4]+CM[2])  ANN\_Speci\_Train  y\_pred = compute(T1\_ANN,test\_set\_scaled\_ANN[,-1])  y\_pred = ifelse(y\_pred$net.result >= 0.5,1,0)  CM = table(test\_set\_scaled\_ANN[,1],y\_pred)  ANN\_Speci\_Test = CM[4]/(CM[4]+CM[2])  ANN\_Speci\_Test # underfitted  # Test data prep LR: 49.07 KNN: 71.29 SVM:3.7 NB: 39.81 CART: 25 RF: 67.59 ANN: 36.11 ----  Missing\_data\_Check(T1\_testdata)  data\_C = subset(T1\_testdata,!is.na(Dependents))  data\_M = subset(T1\_testdata,is.na(Dependents))  dependents = knn(train = scale(data\_C[,c(2,3,4)]),  test = as.matrix(cbind(scale(data\_M[2]), data\_M[3], scale(data\_M[4]))),  cl = as.factor(data\_C$Dependents),  k = 9,  prob = F)  data\_M$Dependents = dependents  T1\_testdata = rbind(data\_C,data\_M)  rm(list = c('data\_C','data\_M'))  Missing\_data\_Check(T1\_testdata)  T1\_testdata$Dependents = as.numeric(T1\_testdata$Dependents)  Two\_D\_View(T1\_testdata)  T1\_testdata\_BS = SMOTE\_fitting(T1\_testdata,500,300)  T1\_testdata\_BS = T1\_testdata\_BS[,c(1:5)]  T1\_testdata\_BS\_Scaled = Scaling(T1\_testdata\_BS)  T1\_traindata\_Complete\_BS\_Scaled = rbind(T1\_traindata\_Train\_BS\_Scaled,  T1\_traindata\_Test\_BS\_Scaled)  T1\_CART = rpart(formula = DLQs ~ .,  data = T1\_traindata\_Complete\_BS\_Scaled,  method = "class",  minsplit= 25,  cp = 0.06535947712,  xval = 7)  y\_pred = predict(T1\_CART, newdata = T1\_testdata\_BS\_Scaled[-1],type='class')  CM = table(T1\_testdata\_BS\_Scaled[,1],y\_pred)  CART\_Speci\_Hold = CM[4]/(CM[4]+CM[2]) |