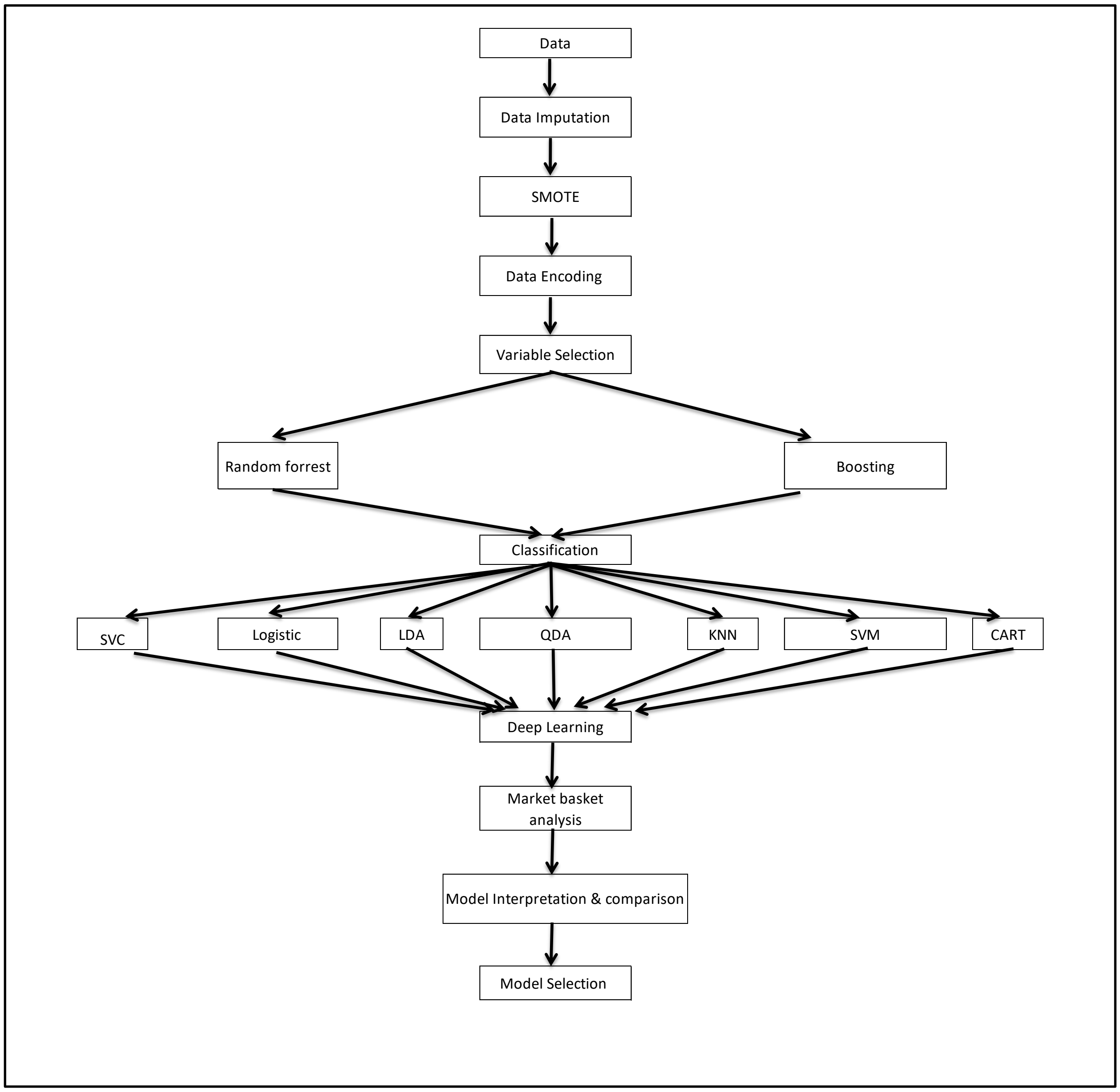
PROJECT APPROACH AND NEW TECHNIQUE DETAILS

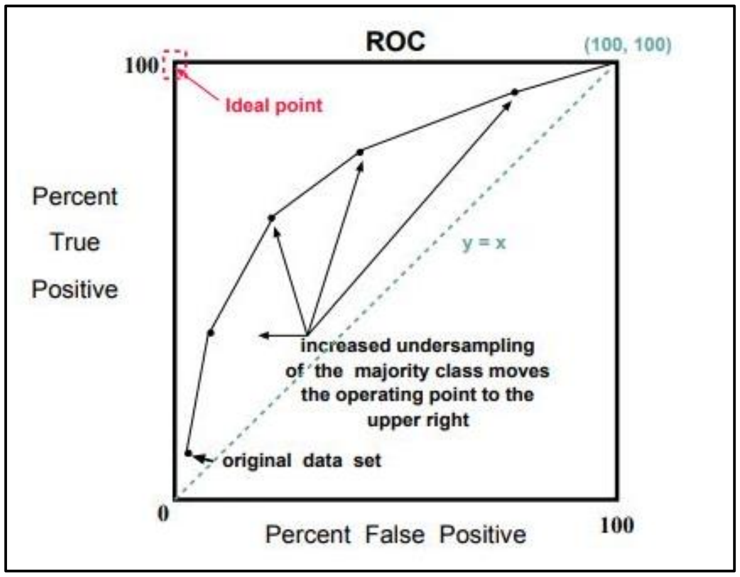
Project Approach:



New Techniques:

Synthetic Minority Over Sampling Technique (SMOTE)

This technique is used when there is an imbalance in the response of the classification. A data set is imbalanced when the classification categories are not approximately equally represented. To rectify this, either the dominant class can be under-sampled or the minority class can be over-sampled by creating synthetic samples rather than sampling with replacement, using the data. Under-sampling is shown to improve the sensitivity of the classifier to a minority class. A combination of both under-sampling the majority class and over-sampling the minority class can achieve better classification performance in the ROC space. [1]



The minority class is over-sampled by taking each minority class sample and introducing synthetic examples considering *k* minority class nearest neighbors. Depending upon the amount of over-sampling required, neighbors from the *k* nearest neighbors are randomly chosen. If *k=5* and the percent of over-sampling = 200%, then only two neighbors from the five nearest neighbors are chosen and one sample is generated in the direction of each.

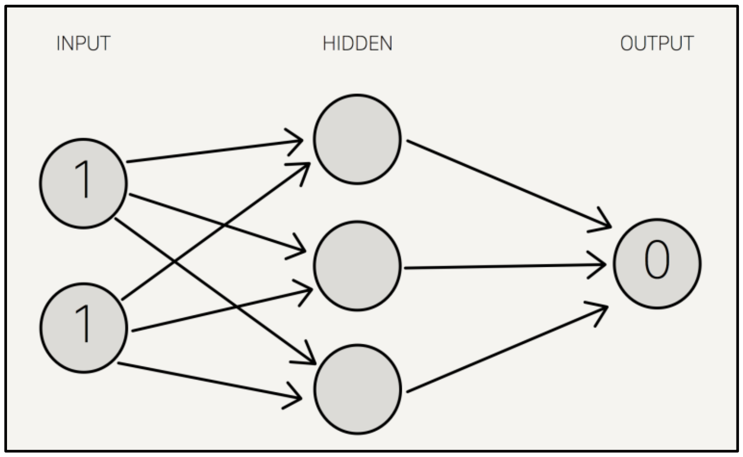
In case of under-sampling the majority class, this is done by removing samples from the majority class until the minority class becomes a percentage of the majority class. Due to this, the minority class will have a larger presence in the data set. For example, if we under-sample the majority class at 300%, it means that the modified data set will contain thrice as many elements from the minority class compared to the majority class. If the minority class had 50 samples and the majority class had 200 samples and we under-sample majority at 200%, the majority class would end up having 25 samples. When a combination of under-sampling and over-sampling is used, the bias towards the majority class is reversed and is equally split between the majority and minority class.

Neural Networks and Deep Learning

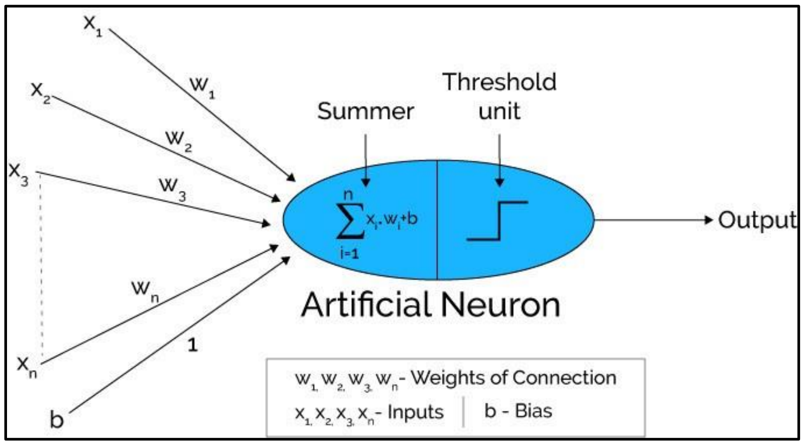
Neural networks can be defined as "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.” [2]

Artificial neural networks(ANN) are computing systems modeled after the neuron structure of the Human Cerebral System. A complex ANN might have thousands of processor units, whereas the Human Cerebral System has billions of interconnected neurons with a drastic increase in magnitude of their overall interaction and emergent behavior. Neural networks are typically structured in layers. The layers may perform different kinds of transformations on the inputs. The layers consist of several interconnected 'nodes' which have a 'transfer function'. Input variables are presented to the network via the 'input layer', which interacts to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' which gives the output of the neural network model. [3]

Neural networks have an exceptional ability to derive meaning from complex or imprecise data, and detect patterns that are too complicated to be fathomed by humans or even other computing techniques. Advantages of neural networks include their ability to detect nonlinear relationships and all possible interactions between variables implicitly, and the availability of multiple training algorithms which can be applied. ANN always takes place according to the situation of the data while techniques like regression and classification require data to be in a certain way. This is a major reason why neural networks are preferred today over other techniques.



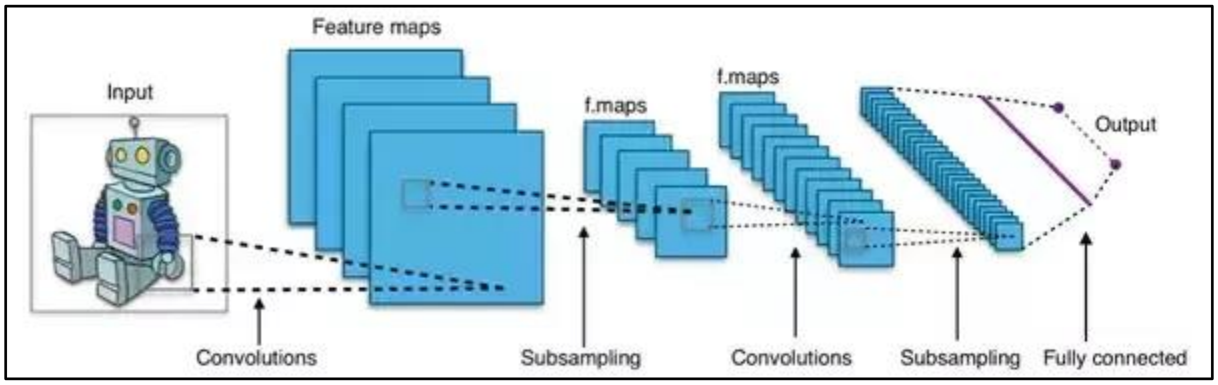
In the figure given below, it can be observed that x’s are the inputs with the w’s being the weights of each input. Each input is multiplied by its corresponding weights, which is the information used by the neural network to solve the problem. The summer sums up all the weights and bias is added if weights are non-zero or to scale up the response. The threshold unit or the activation function is then used to obtain the desired output from the sum. Different types of activation functions can be used according to the application.


x_1
x_2
x_3
x_n
b
w_1
w_2
w_3
w_n
1
Summer
Threshold unit
$\sum_{i=1}^{n} x_i \cdot w_i + b$
Output
Artificial Neuron
w_1, w_2, w_3, w_n - Weights of Connection
x_1, x_2, x_3, x_n - Inputs
b - Bias

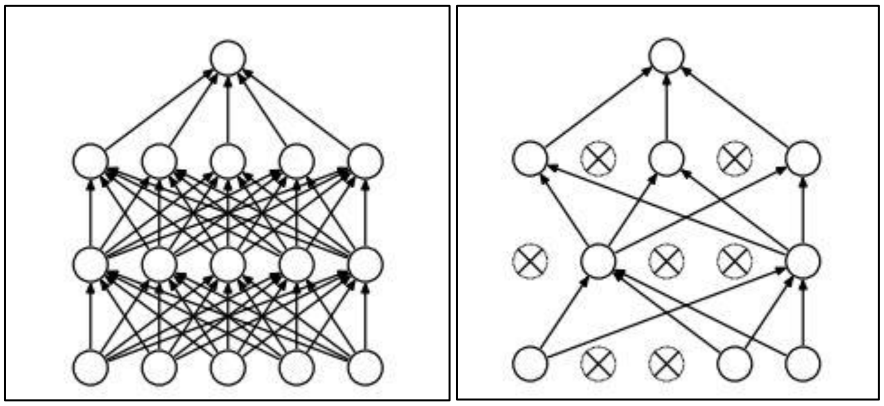
One type of activation function is the Rectifier or ReLU(Rectified Linear Unit). It's widely used in neural networks today and contrary to the name, it is a nonlinear activation function i.e it can easily back propagate the errors and have multiple layers of neurons being activated by the ReLU function. A major advantage over other activation functions is that it does not activate all the neurons at the same time. Only a few neurons are activated making the network less complicated and intricate and hence reducing the number of computations. This is the reason why the ReLU is widely used in deep learning neural networks.

A standard neural network consists of many simple, inter-connected processors each producing a sequence of activations. After the input is been given to the neural network, the following neurons get activated through weighted connections from previously active neurons. Deep Learning or credit assignment is about obtaining those weights that make the Neural network exhibit the desired behavior, such as in applications like image recognition. Depending on the problem, such behavior may require long causal chains of computational stages or many hidden layers, where each layer the combined activation of the network. Deep Learning is about accurately assigning credit across many such stages. The learning algorithms used automatically learn the weights and biases from the training data [4]

A type of deep learning neural network called Convolution Neural Networks (CNN) significantly enhances the capabilities of the feed forward network by inserting convolution layers, majorly used for image recognition. The layers consist of filters which have a small receptive field that traverse through the input volume. During the forward pass, each filter is convolved across dimensions of the input, computing the dot product between the entries of the filter and the input, and producing a 2-dimensional activation map of that filter. Feature maps in between can be thought of as hidden layers where the parts of the input are mapped on them. This process is done repeatedly through sub sampling and convolutions until it results in a fully connected network. As a result, the CNN learns filters that activate when it detects some specific type of feature in the input. This can then be used to classify and train image inputs. [5]



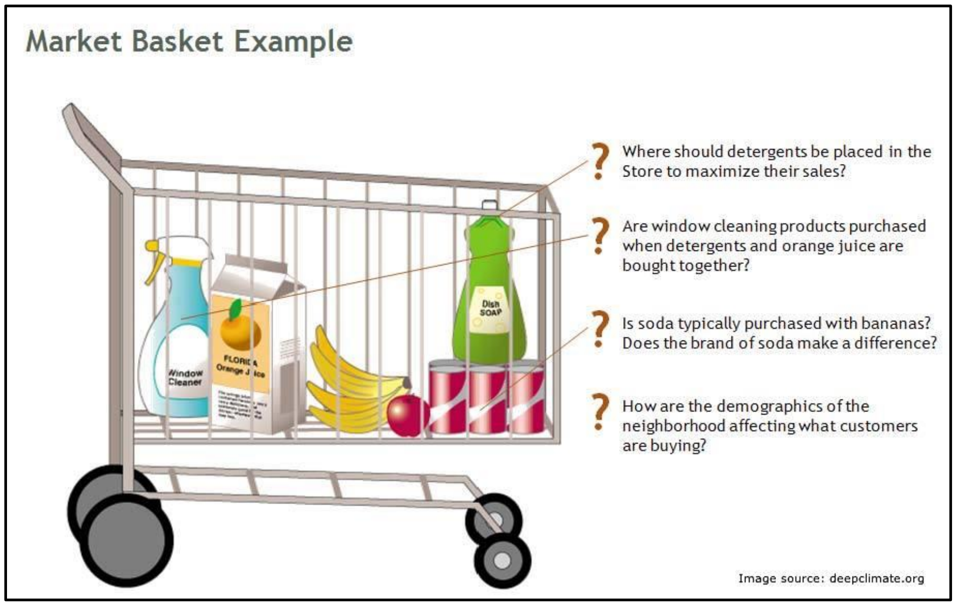
A major advantage in deep learning neural networks compared to standard neural networks is its ability to avoid overfitting using Dropout. The main idea is to randomly drop units along with their connections from the neural network during training. This provides a way of approximately combining many different neural network architectures efficiently. Hence it prevents the units from co-adapting too much. The resulting "thinned" networks can then be aggregated to improve the performance of the model. The below figure shows before and after dropout. [6]



Market Basket Analysis

Market Basket Analysis is performed to identify the association rules among a set of existing predictors and responses. This method is primarily used by retailers to identify association between items to have a better product placement policy.

MBA defines the relationship between certain parameters which may have an influence on the response value. For instance, big retailers such as Walmart in a famous study found out that shoppers especially parents who purchase diapers tend to buy beer more, hence Walmart opted for a strategy such that it placed baby-diaper section near to the alcoholic beverages section and the results were very propitious as there was a huge increment in the sale of beers.


Market Basket Example
Where should detergents be placed in the Store to maximize their sales?
Are window cleaning products purchased when detergents and orange juice are bought together?
Is soda typically purchased with bananas? Does the brand of soda make a difference?
How are the demographics of the neighborhood affecting what customers are buying?
Window Cleaner
Florida Orange Juice
Dish Soap
Image source: deepclimate.org

MBA also find its application in medical field as many researchers have used it to extrapolate information regarding diseases like diabetes and what association it has with factors like Age, Gender, and occupation of the personal [7]. One of the important result of the research was that individual might suffer from diseases like ophthalmic, neurological and peripheral circulatory complications if they also had type 2 diabetes mellitus and had no occupation. As can be seen there are numerous application of this technique in the real-world scenario.

MBA essentially uses concepts of probability to determine the co-occurrence and association among the predictor space. In the output we get four major parameters [8]: -

1) Rules: - A rule is an implication set i.e. X ---> Y it represents association of X (L.H.S) and Y (R.H.S).
2) Support: - It measures the frequency of occurrence of relationship between X and Y for a given dataset. Mathematically, $\frac{Number\ of\ transactions\ with\ both\ X\ and\ Y}{Total\ number\ of\ transactions} = P(X\Omega Y)$,where P represents the probability.
3) Confidence: - It is the probability of occurrence of event Y given event X already happened. (Bayes Theorem).

$\frac{Number\ of\ transactions\ with\ both\ X\ and\ Y}{Total\ number\ of\ transactions\ with\ \ X} = \frac{P(X\Omega Y)}{P(X)}$, where P represents the probability.

4) Lift: - It measures how the co-occurrence of A and B will exceeds the expected probability of A and B occurring simultaneously, had they been independent (lift=1).
$\frac{Number\ of\ transactions\ with\ both\ X\ and\ Y}{Total\ number\ of\ transactions\ with\ \ X} = \frac{P(X\Omega Y)}{P(X).P(Y)}$, where P represents the probability.

For our analysis we will be sorting all the rules i.e. rule having the maximum lift value because it is the strongest parameter which helps in clubbing all the predictors which have higher co-occurrence implying they are very much dependent on each other. Although directionality is lost for the rule i.e. lift is similar for X ---> Y and Y ---> X but for our analysis directionality is not relatively important. [9]

IMPLEMENTATION

Synthetic Minority Oversampling Technique

Loading the library 'Data Mining with R' for performing SMOTE.

```
set.seed(3)
```

```
library(DMwR)
```

Before performing SMOTE, we look at the existing situation of Attrition in the data set.

```
table(IBM$Attrition)
```

```
##   No  Yes
## 1233  237
```

We observe that the data set is highly imbalanced and needs to be altered.

Performing SMOTE on the existing data set, we use percentage over and percentage under to identify how much over and under sampling needs to be done. *K* = 3 pertains to the number of neighbors that are taken into consideration while determining the class of the sample.

```
IBM=SMOTE(Attrition~., data=IBM, perc.over = 200, perc.under = 200, k=3)
```

```
table(IBM$Attrition)
```

```
##  No Yes
## 948 711
```

As can be seen, the responses have been simultaneously being under and over sampled, keeping the initial ratios of 'No to Yes' intact. This would hence now be an ideal data set to work on.

We first split the data into training and test in the ratio of 80:20 respectively.

```
#####Train and test#####
set.seed(1)
train = sample (1: nrow(IBM), nrow(IBM)*.80)
IBM.train=IBM[train,]
IBM.test=IBM[-train,]
```

Data Cleaning

After applying SMOTE, we were supposed to convert every predictor into category such that they can be used for Market Basket Analysis and other classification techniques. It was essential for us to order the variables which we thought should have an order associated with it. The following table will explain easily how the numeric values are being assigned a categorical value.

Sr. No	Variable	Level of order						Ordered
		0	1	2	3	4	5	
1	EnvironmentSatisfaction	---	Not satisfied	Moderately satisfied	Satisfied	Very satisfied	---	Y
2	JobSatisfaction	---	Not satisfied	Moderately satisfied	Satisfied	Very satisfied	---	Y
3	WorklifeBalance	---	Unmanaged	partly managed	Managed	Perfectly managed	---	Y
4	JobInvolvement	---	No involvement	Very less involvement	Moderate Involvement	Full Involvement	---	Y
5	Education	---	High school	Associate degree	Bachelors	Masters	Post doctorate	Y
6	RelationshipSatisfaction	---	Not satisfied	Moderately satisfied	Satisfied	Very satisfied	---	Y
7	PerformanceRating	---	Poor	Fair	Average	Good	---	Y
8	Job level	---	Seniority level 0	Seniority level 1	Seniority level 2	Seniority level 3	Seniority level 4	Y
9	Attrition	No	Yes	---	---	---	---	N
10	BusinessTravel	---	Non-Travel	Travel Rarely	Travel Frequently	---	---	Y
11	Department	---	Human Resources	Research & Development	Sales	---	---	N
12	DailyRate	Low	Fair	Moderate	High	---	---	Y
13	YearsInCurrentRole	Low	Fair	Moderate	High	---	---	Y
14	MaritalStatus	---	Divorced	Married	Single	---	---	N
15	Overtime	No	Yes	---	---	---	---	N
16	Gender	Male	Female	---	---	---	---	N
17	Age	18-30	31-36	37-40	41-61	---	---	Y
18	DistanceFromHome	1-3	4-8	9-14	15-30	---	---	Y

For other predictors similar strategy was followed and implemented.

```
####Data Cleaning####

#Factorizing string columns

IBM$Attrition <-  factor(IBM$Attrition, levels= c("No","Yes"),labels = c("0",
"1"))

IBM$BusinessTravel <-  factor(IBM$BusinessTravel, levels = c("Non-Travel",
"Travel_Rarely","Travel_Frequently"), labels=c("1","2","3"),ordered="TRUE")

IBM$Department <-  factor(IBM$Department, levels = c("Human Resources",
"Research & Development","Sales"), labels=c("1","2","3"),ordered="FALSE")

IBM$EducationField <-  factor(IBM$EducationField, levels = c("Human
Resources", "Life Sciences","Marketing","Medical","Other","Technical
Degree"), labels=c("1","2","3","4","5","6"),ordered="FALSE")

IBM$JobRole <-  factor(IBM$JobRole, levels = c("Healthcare Representative",
"Human Resources","Laboratory Technician","Manager","Manufacturing
Director","Research Director","Research Scientist","Sales Executive","Sales
Representative"),
labels=c("1","2","3","4","5","6","7","8","9"),ordered="FALSE")

IBM$MaritalStatus <-  factor(IBM$MaritalStatus, levels = c("Divorced",
"Married","Single"), labels=c("1","2","3"),ordered="FALSE")

IBM$OverTime <-  factor(IBM$OverTime, levels = c("No", "Yes"),
labels=c("0","1") ,ordered="FALSE")

IBM$Gender<- factor(IBM$Gender, levels=  c("Male","Female"),
labels=c("0","1"), ordered="FALSE")
```

Here we removed redundant columns which had the same values for every row: -

```
##Removing redundant rows

IBM$EmployeeCount <- NULL
```

```
IBM$Over18 <- NULL

IBM$StandardHours <- NULL

IBM$EmployeeNumber<-NULL


row.has.na <- apply(IBM, 1, function(x){any(is.na(x))})

sum(row.has.na)


str(IBM)


##Converting Numerical Values into factors for MBA and NN

IBM$ï..Age=cut(IBM$ï..Age,breaks = c(18, 30, 36, 40, 61), labels = c("0",
"1", "2", "3"),right = FALSE, ordered_result = "TRUE")

is.factor(IBM$ï..Age)


summary(IBM$DistanceFromHome)

IBM$DistanceFromHome=cut(IBM$DistanceFromHome,breaks = c(1, 3, 8, 14, 30),
labels = c("0", "1", "2", "3"),right = FALSE, ordered_result = "TRUE")

is.factor(IBM$DistanceFromHome)


summary(IBM$DailyRate)

IBM$DailyRate=cut(IBM$DailyRate,breaks = c(102, 452, 811.6, 1158.1, 1500),
labels = c("0", "1", "2", "3"),right = FALSE, ordered_result = "TRUE")

is.factor(IBM$DailyRate)


summary(IBM$HourlyRate)

IBM$HourlyRate=cut(IBM$HourlyRate,breaks = c(30, 51.19, 67.33, 83.6, 101),
labels = c("0", "1", "2", "3"),right = FALSE, ordered_result = "TRUE")


summary(IBM$MonthlyIncome)

IBM$MonthlyIncome=cut(IBM$MonthlyIncome,breaks = c(1009, 2598, 4014 ,6634 ,
20000), labels = c("0", "1", "2", "3"),right = FALSE, ordered_result =
"TRUE")
```

```
summary(IBM$MonthlyRate)

IBM$MonthlyRate=cut(IBM$MonthlyRate,breaks = c(2094, 7739,12858 , 19179 ,
27000), labels = c("0", "1", "2", "3"),right = FALSE, ordered_result =
"TRUE")


summary(IBM$NumCompaniesWorked)

IBM$NumCompaniesWorked=cut(IBM$NumCompaniesWorked,breaks = c(0,1,2.9 ,5 ,10),
labels = c("0", "1", "2", "3"),right = FALSE, ordered_result = "TRUE")


summary(IBM$PercentSalaryHike)

IBM$PercentSalaryHike=cut(IBM$PercentSalaryHike,breaks = c(11, 13,14.27 ,18
,26), labels = c("0", "1", "2", "3"),right = FALSE, ordered_result = "TRUE")


summary(IBM$YearsWithCurrManager)

IBM$YearsWithCurrManager=cut(IBM$YearsWithCurrManager,breaks = c(0, 1.1,2.7
,6.9 ,18), labels = c("0", "1", "2", "3"),right = FALSE, ordered_result =
"TRUE")


summary(IBM$YearsSinceLastPromotion)

IBM$YearsSinceLastPromotion=cut(IBM$YearsSinceLastPromotion,breaks = c(0,
1,3,16), labels = c("0", "1", "2"),right = FALSE, ordered_result = "TRUE")


summary(IBM$YearsInCurrentRole)

IBM$YearsInCurrentRole=cut(IBM$YearsInCurrentRole,breaks = c(0, 1.45,2.25 ,7
,19), labels = c("0", "1", "2", "3"),right = FALSE, ordered_result = "TRUE")


summary(IBM$YearsAtCompany)

IBM$YearsAtCompany=cut(IBM$YearsAtCompany,breaks = c(0, 2,5 ,8.27 ,41),
labels = c("0", "1", "2", "3"),right = FALSE, ordered_result = "TRUE")


summary(IBM$TotalWorkingYears)
```

```
IBM$TotalWorkingYears=cut(IBM$TotalWorkingYears,breaks = c(0, 6,8.565
,12.145,41), labels = c("0", "1", "2", "3"),right = FALSE, ordered_result =
"TRUE")
```

```
summary(IBM$TrainingTimesLastYear)
```

```
IBM$TrainingTimesLastYear=cut(IBM$TrainingTimesLastYear,breaks = c(0,
2,3.42,7), labels = c("0", "1", "2"),right = FALSE, ordered_result = "TRUE")
```

Random Forest

Code and output:

```
####randomForest####
library(randomForest)
```

```
set.seed(3)
#for(ntree in seq(100,500,50)){
#  for(j in c(4,5,6,7,8)){
#    rf.IBM=randomForest(Attrition~., data=IBM, subset=train, ntree=ntree, mt
ry=j, importance=TRUE )
#    print(rf.IBM)
#    yhat.rf=predict(rf.IBM, IBM.test)
#    yhat.rf
#    print(mean(yhat.rf==IBM.test$Attrition))
#  }
#}
```

```
#The best accuracy was found for ntree=200 & mtry=4
```

```
set.seed(3)
rf.IBM=randomForest(Attrition~., data=IBM, subset=train,ntree=200, mtry=7, im
portance=TRUE)
rf.IBM
```

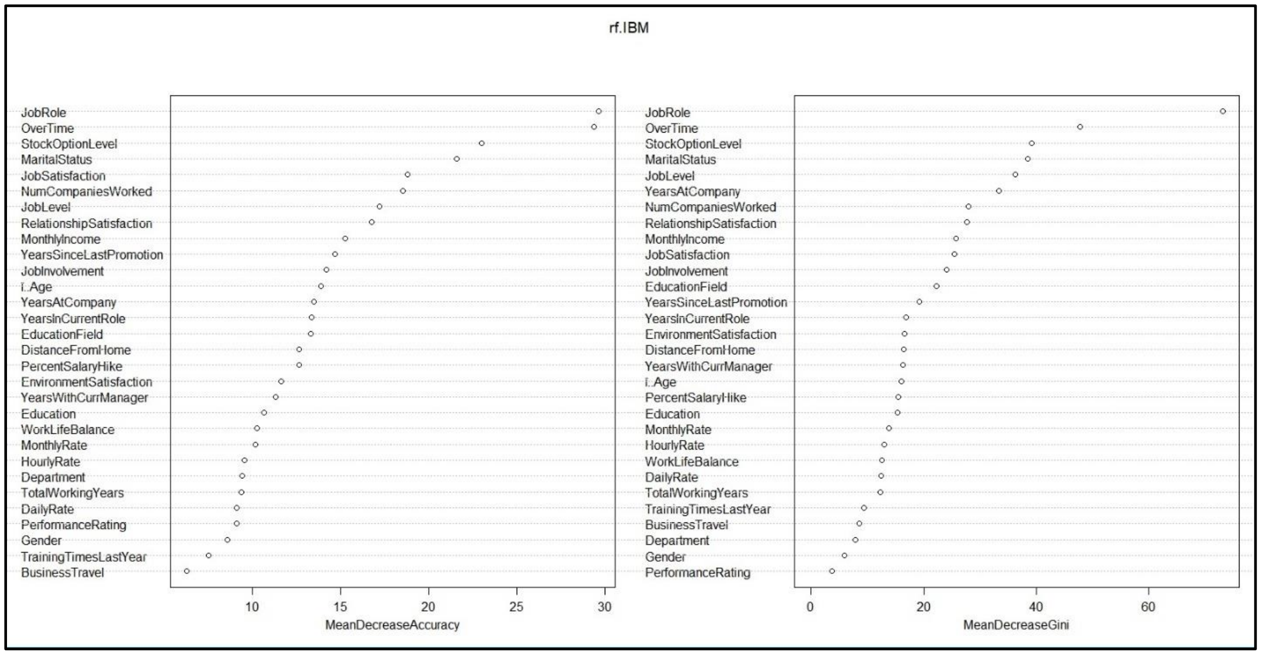
```
##
## Call:
##  randomForest(formula = Attrition ~ ., data = IBM, ntree = 200,      mtry
= 7, importance = TRUE, subset = train)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 11%
## Confusion matrix:
```

```
##     0   1 class.error
## 0 711  33  0.04435484
## 1 113 470  0.19382504
```

```
importance(rf.IBM)
```

```
##                                  0         1 MeanDecreaseAccuracy
## ï..Age                   13.078324  9.474914            16.391809
## BusinessTravel            7.177082  5.889172             8.642107
## DailyRate                 8.033488  2.468394             7.560102
## Department                8.016530  2.236445             8.670679
## DistanceFromHome         10.307770  5.659848            10.165196
## Education                 9.267599  2.906585             8.295980
## EducationField           13.157026  4.161863            13.658541
## EnvironmentSatisfaction  11.300660  4.123485            11.378714
## Gender                    9.043055  4.123324             8.617396
## HourlyRate                8.929711  2.379006             8.091165
## JobInvolvement           13.481268  5.495626            13.415916
## JobLevel                 16.148093  3.871528            16.719002
## JobRole                  23.398354 10.032952            25.401782
## JobSatisfaction          14.253639 11.581867            16.693482
## MaritalStatus            18.481887  6.588110            19.921616
## MonthlyIncome            11.339949  6.682183            13.657926
## MonthlyRate               9.787664  4.930814             8.815346
## NumCompaniesWorked       15.056911 13.236431            17.643740
## OverTime                 25.457990 17.308915            25.016356
## PercentSalaryHike         9.856931  7.187511            11.248331
## PerformanceRating         7.659445  1.554322             6.757568
## RelationshipSatisfaction 18.728072  6.952294            16.862484
## StockOptionLevel         19.953615  2.787205            21.598935
## TotalWorkingYears         8.832722  4.727253             9.446250
## TrainingTimesLastYear     8.302879  4.566150             9.073342
## WorkLifeBalance           8.965266  5.415502             9.205977
## YearsAtCompany           11.084868 10.289341            12.538799
## YearsInCurrentRole        9.287482  8.373779            11.454887
## YearsSinceLastPromotion  12.173607 10.288920            14.712402
## YearsWithCurrManager     10.311860  7.553776            11.760163
## Age                       8.906527  6.656687            11.227015
##                          MeanDecreaseGini
## ï..Age                          31.271344
## BusinessTravel                   8.092097
## DailyRate                       11.306744
## Department                       6.941694
## DistanceFromHome                14.804855
## Education                       15.304246
## EducationField                  20.045096
## EnvironmentSatisfaction         15.731941
## Gender                           5.876942
## HourlyRate                      11.464429
## JobInvolvement                  21.731242
## JobLevel                         38.162796
## JobRole                          73.963662
## JobSatisfaction                  23.272094
## MaritalStatus                    40.153921
## MonthlyIncome                    22.655645
## MonthlyRate                      13.550113
## NumCompaniesWorked               24.306009
## OverTime                         45.267752
## PercentSalaryHike                14.037872
## PerformanceRating                 3.116780
## RelationshipSatisfaction         26.972844
## StockOptionLevel                 37.685798
## TotalWorkingYears                11.893602
## TrainingTimesLastYear             8.605262
## WorkLifeBalance                  11.999092
## YearsAtCompany                   28.746205
## YearsInCurrentRole               18.204411
## YearsSinceLastPromotion          20.012939
## YearsWithCurrManager             16.609820
## Age                              11.920803
```

```
varImpPlot(rf.IBM)
```



```
yhat=predict(rf.IBM, IBM.test)
mean(yhat!=IBM.test$Attrition)
```

```
## [1] 0.09638554 #The mean error rate is 0.0963
```

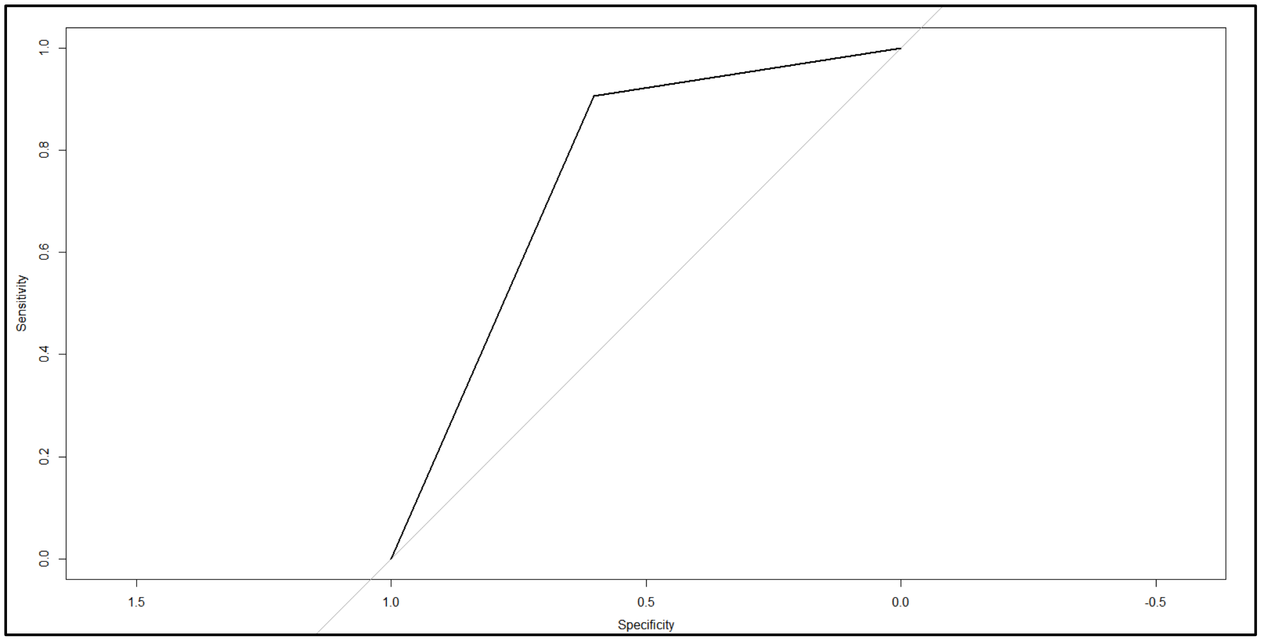
```
#Results of Random Forest suggest that JobRole, OverTime,
StockOptionLevel, MaritalStatus are important predictors
```

The mean error rate is 0.0963. The important variables suggested by Random forest are JobRole, OverTime, StockOptionLevel and MaritalStatus. Therefore, we will be using these variables as predictors for all the other models.

ROC for Random Forest

Code and Output:

```
#####ROC for RF####
yhat.rf=predict(rf.IBM, IBM.test)
a= as.numeric(yhat.rf)
roc(IBM.test$Attrition, a, plot = TRUE)
```



```
##
## Call:
## roc.default(response = IBM.test$Attrition, predictor = a, plot = TRUE)
##
## Data: a in 204 controls (IBM.test$Attrition 0) < 128 cases (IBM.test$Attrition 1).
## Area under the curve: 0.8939
```

The area under the curve is 0.8939 which suggests a very good model.

Therefore for the Random Forest the important predictors are JobRole, OverTime, StockOptionLevel, MaritalStatus are important predictors with a mean error of 9.63% and AUC 89.39%.

Boosting

Code and Output:

```
#####GBM#####
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.4.2
```

```
## Loading required package: survival
```

```
## Loading required package: splines
```

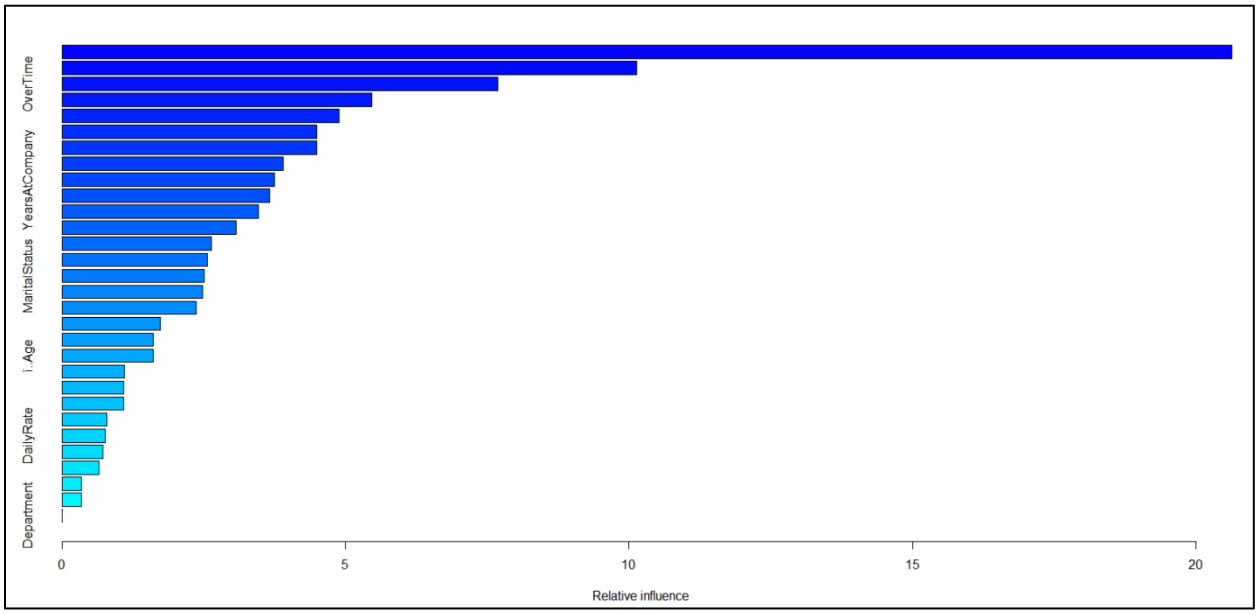
```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.3
```

```
set.seed (2)
boost.IBM=gbm(Attrition~., data=IBM.train, n.trees=1000, shrinkage = .2, inte
raction.depth =1, distribution = "multinomial" )
boost.IBM
```

```
## gbm(formula = Attrition ~ ., distribution = "multinomial", data = IBM.trai
n,
##     n.trees = 1000, interaction.depth = 1, shrinkage = 0.2)
## A gradient boosted model with multinomial loss function.
## 1000 iterations were performed.
## There were 30 predictors of which 29 had non-zero influence.
```

```
summary(boost.IBM)
```



```
##                                                var    rel.inf
## JobRole                                    JobRole 20.6362868
## StockOptionLevel                  StockOptionLevel 10.1326066
## OverTime                                  OverTime  7.6892275
## EducationField                      EducationField  5.4657703
## JobLevel                                       JobLevel  4.8826523
## JobInvolvement                           JobInvolvement  4.4901947
## NumCompaniesWorked                   NumCompaniesWorked  4.4848137
## YearsSinceLastPromotion         YearsSinceLastPromotion  3.9015171
## YearsAtCompany                           YearsAtCompany  3.7512460
## JobSatisfaction                         JobSatisfaction  3.6591946
## Education                                     Education  3.4619129
## EnvironmentSatisfaction         EnvironmentSatisfaction  3.0653900
## WorkLifeBalance                         WorkLifeBalance  2.6394233
## DistanceFromHome                       DistanceFromHome  2.5682039
## MaritalStatus                             MaritalStatus  2.5039588
## PercentSalaryHike                     PercentSalaryHike  2.4846367
## RelationshipSatisfaction RelationshipSatisfaction  2.3686227
## BusinessTravel                           BusinessTravel  1.7376504
## TrainingTimesLastYear             TrainingTimesLastYear  1.6085470
## ï..Age                                           ï..Age  1.6029072
## MonthlyRate                                 MonthlyRate  1.0977768
## YearsWithCurrManager               YearsWithCurrManager  1.0922936
## YearsInCurrentRole                   YearsInCurrentRole  1.0787259
## Gender                                           Gender  0.7882322
## DailyRate                                     DailyRate  0.7627077
## HourlyRate                                   HourlyRate  0.7177395
## MonthlyIncome                             MonthlyIncome  0.6427319
## PerformanceRating                     PerformanceRating  0.3429611
## TotalWorkingYears                     TotalWorkingYears  0.3420690
## Department                                   Department  0.0000000
```

```
#Results of boosting suggest JobRole, EducationField, StockOptionLevel
& Overtime are the most influencing predictors
```

The results of Boosting suggest that JobRole, EducationField, StockOptionLevel and OverTime are the most influencing predictors.

Logistic Regression

```
####Logistic####
set.seed(1)
glm.fit=glm(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus ,data=I
BM ,family=binomial, subset = train)
glm.probs=predict(glm.fit ,IBM.test, type ="response")
a=nrow(IBM.test)
a
```

```
## [1] 332
```

```
glm.pred=rep(0 , a)
```

```
glm.pred[glm.probs>.5]="1"
table(glm.pred ,IBM.test$Attrition)
```

```
##
## glm.pred   0   1
##        0 177  38
##        1  27  90
```

```
mean(glm.pred!= IBM.test$Attrition )
```

```
## [1] 0.1957831 #The mean error rate is 0.1957
```

The logistic model gives a mean error rate of 0.1957.

Cross-Validation of Logistic Regression

```
#####CV for Logistic####
library(boot)
```

```
##
## Attaching package: 'boot'
```

```
## The following object is masked from 'package:lattice':
##
##     melanoma
```

```
set.seed(15)
cv.error=rep(0 ,10)
for (i in 1:10){
  glm.fit=glm(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus  ,dat
a=IBM ,family=binomial)
  cv.error[i]=cv.glm(IBM ,glm.fit ,K=10)$delta[1]
}
cv.error
```

```
##  [1] 0.1481327 0.1474964 0.1476920 0.1471595 0.1478829 0.1482471 0.1487169
##  [8] 0.1482536 0.1477867 0.1484321
```

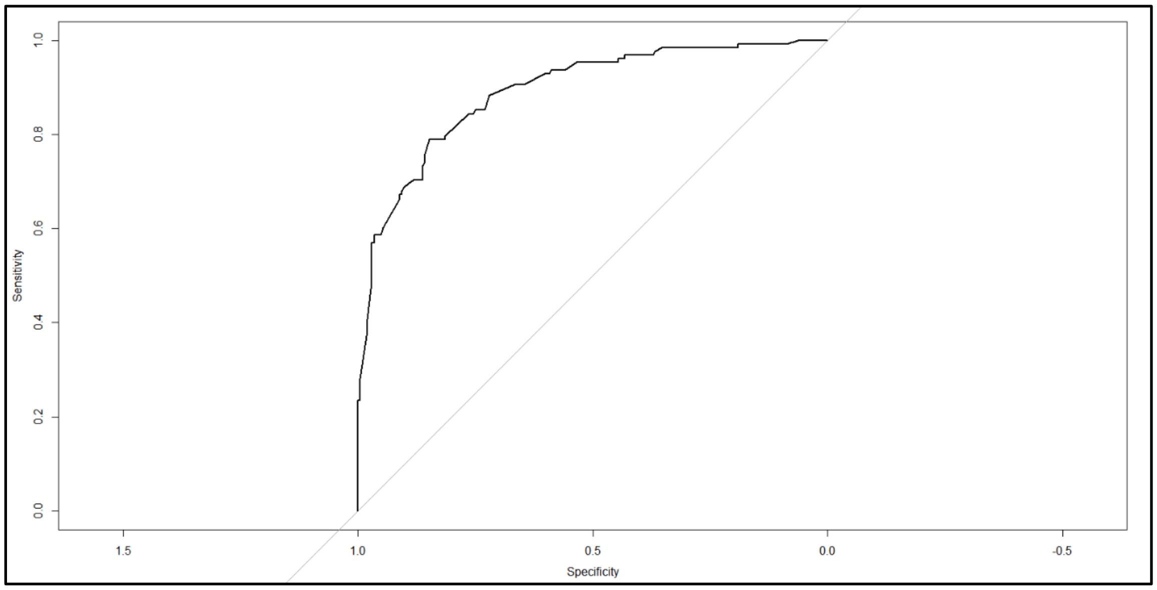
The cross-validation of the logistic regression shows a consistent error rate of 0.147

ROC of Logistic Regression

```
#####ROC for Logistic#####
library(pROC)
glm.fit=glm(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus  ,data=
IBM ,family=binomial, subset = train)
glm.probs=predict(glm.fit,IBM.test, type ="response")
q = roc(IBM.test$Attrition, glm.probs)
auc(q)
```

```
## Area under the curve: 0.8915
```

```
plot.roc(q)
```



The area under the ROC is 0.8915.

The results of logistic regression shows promising results with mean error rate 19.57%, CV-error of 14% and AUC 89.15%. The ROC show very good Specificity and Sensitivity.

Linear Discriminant Analysis

```
####LDA####
library(MASS)
set.seed(6)
lda.IBM=lda(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus  ,data=
IBM , subset=train)
#lda.IBM
lda.pred=predict(lda.IBM , IBM.test )
lda.class=lda.pred$class
table(lda.class , IBM.test$Attrition)
```

```
##
## lda.class   0   1
##         0 177  38
##         1  27  90
```

```
mean(lda.class!=IBM.test$Attrition)
```

```
## [1] 0.1957831
```

The mean error rate for this model is 0.1957

Cross-Validation for LDA

```
#####CV for LDA#####
cv.lda=lda(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus  ,data=I
BM , CV=TRUE,method="moment")
table(cv.lda$class, IBM$Attrition)
```

```
##
##       0   1
##   0 798 209
##   1 150 502
```

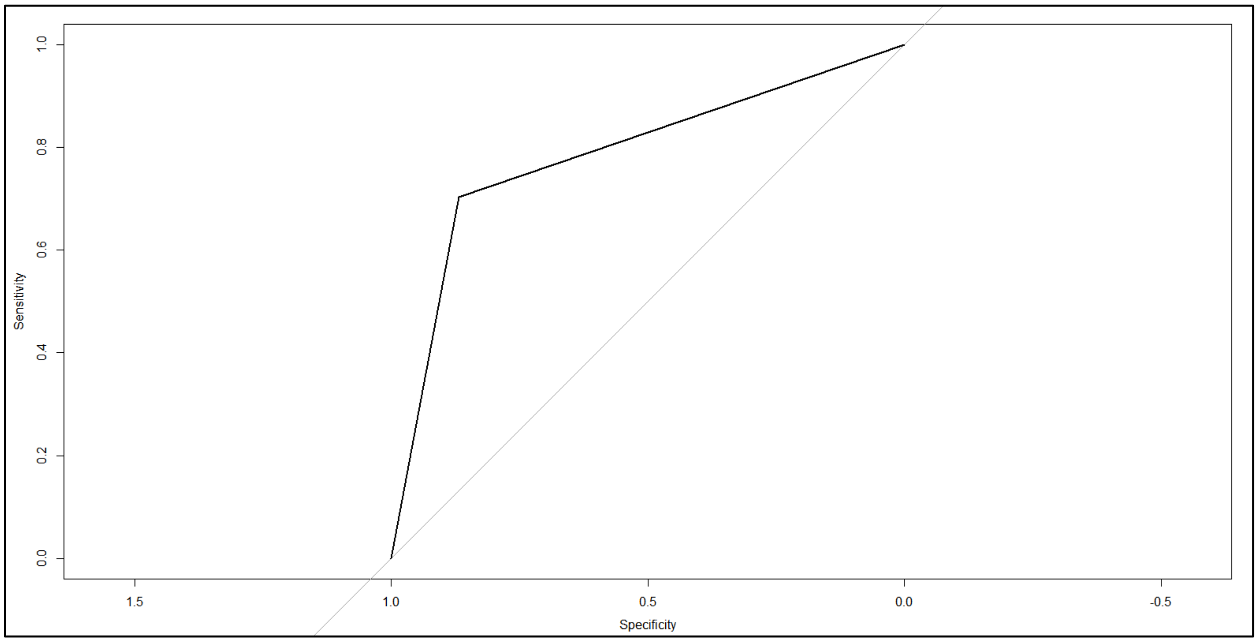
```
mean(cv.lda$class!=IBM$Attrition)
```

```
## [1] 0.2163954
```

The cross-validation error rate is 0.2163 which is very close to the testing error rate.

ROC for LDA

```
####ROC for LDA####
a=as.numeric(lda.class)
roccurve.lda = roc(IBM.test$Attrition,a)
plot(roccurve.lda, colorize = TRUE)
```



```
auc(roccurve.lda)
```

```
## Area under the curve: 0.7854
```

The mean error rate is same as that of logistic regression i.e. 19.57% but there is a rise of about 21.63% in the CV-error as compared to logistic. Also, the AUC is reduced to 78.54%. Logistic per forms better as it does not make any assumption about the normality of the distribution of the d ata.

Quadratic Discriminant Analysis

```
####QDA####
qda.IBM=qda(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus ,data=I
BM ,subset=train)
#qda.IBM
qda.pred=predict(qda.IBM , IBM.test )
qda.class=qda.pred$class
table(qda.class , IBM.test$Attrition)
```

```
##
## qda.class   0   1
##         0 123  12
##         1  81 116
```

```
mean(qda.class!=IBM.test$Attrition)
```

```
## [1] 0.2801205
```

The mean error rate here is 0.2801

Cross-validation for QDA

```
#####CV for QDA#####
cv.qda=qda(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus  ,data=I
BM , CV=TRUE,method="moment")
table(cv.qda$class, IBM$Attrition)#Calculate this manually
```

```
##
##       0   1
##   0 593  98
##   1 355 613
```

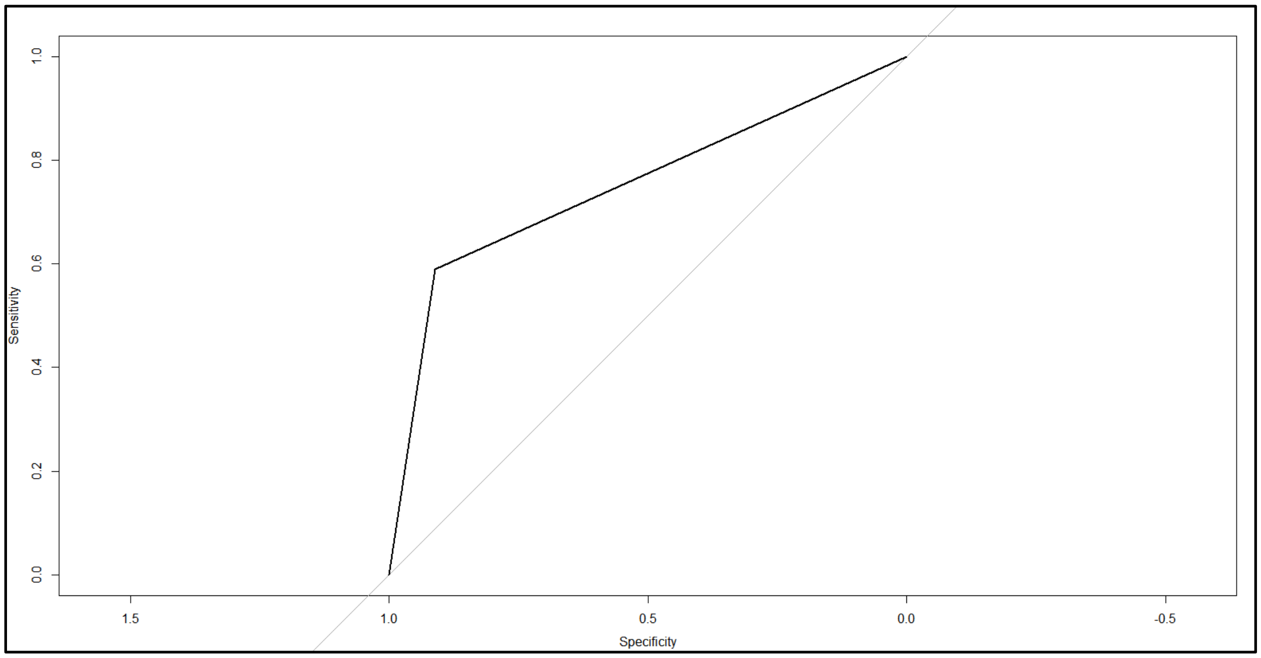
```
mean(cv.qda$class!=IBM$Attrition) )#Classification Error
```

```
## [1] 0.2730561
```

The CV-error is 27.3%.

ROC for QDA

```
####ROC for QDA####
library(pROC)
a=as.numeric(qda.class)
b=as.numeric(IBM.test$Attrition)
roccurve.qda=roc(a,b)
plot(roccurve.qda, colorize = TRUE)
```



```
auc(roccurve.qda)
```

```
## Area under the curve: 0.75
```

The QDA has a mean error rate of 19.57. The CV-error is 28.01% and the AUC is 75%. We can observe that as the flexibility of the model increases the performance is reducing.

KNN

```
####KNN####
library(class)
train.IBM=cbind(IBM.train$Overtime,IBM.train$MaritalStatus,IBM$JobRole,IBM$St
ockOptionLevel)
```

```
## Warning in cbind(IBM.train$Overtime, IBM.train$MaritalStatus, IBM
## $JobRole, : number of rows of result is not a multiple of vector length
## (arg 2)
```

```
test.IBM=cbind(IBM.train$Overtime,IBM.train$MaritalStatus,IBM$JobRole,IBM$Sto
ckOptionLevel)
```

```
## Warning in cbind(IBM.train$Overtime, IBM.train$MaritalStatus, IBM
## $JobRole, : number of rows of result is not a multiple of vector length
## (arg 2)
```

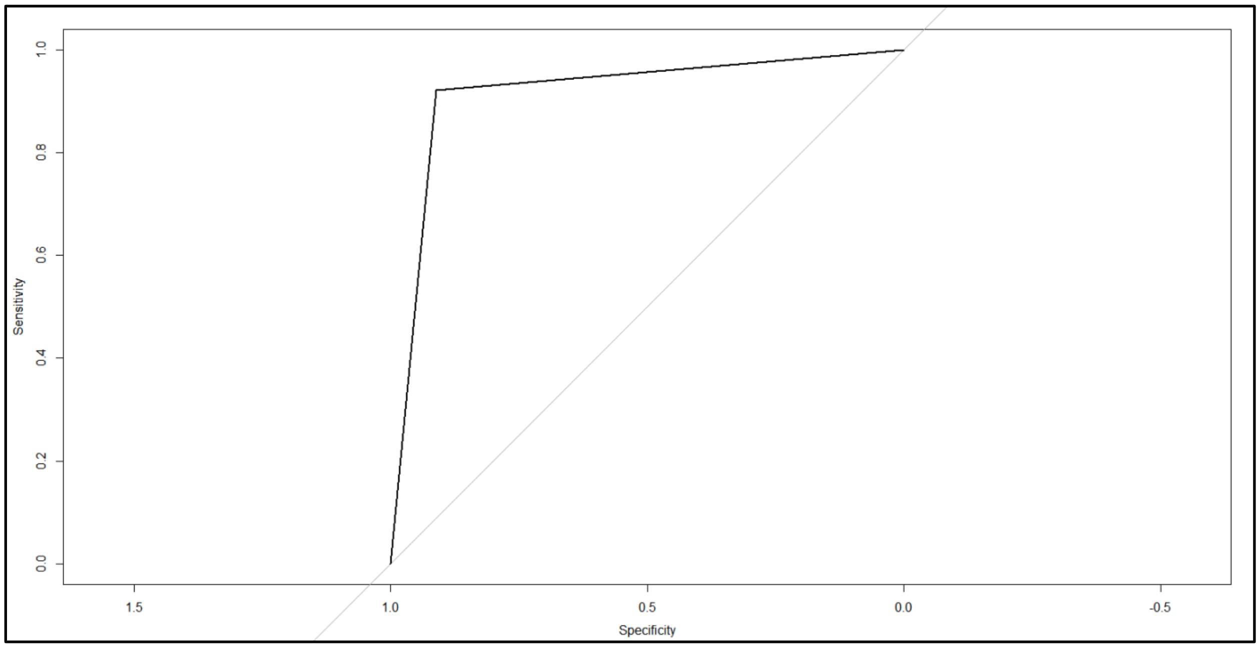
```
train.Attrition=IBM.train$Attrition
set.seed(8)
for(k in 1:10){
  knn.pred=knn(IBM.train, IBM.test, train.Attrition ,k=k)
  table(knn.pred, IBM.test$Attrition)
  print(k)
  print(mean(knn.pred==IBM.test$Attrition)) #Classification Accuracy
}
```

```
## [1] 1
## [1] 0.9216867
## [1] 2
## [1] 0.8885542
## [1] 3
## [1] 0.8825301
## [1] 4
## [1] 0.8524096
## [1] 5
## [1] 0.8524096
## [1] 6
## [1] 0.8524096
## [1] 7
## [1] 0.8493976
## [1] 8
## [1] 0.8584337
## [1] 9
## [1] 0.8554217
## [1] 10
## [1] 0.8463855
```

```
#Best model is at k=1 with 92.16% accuracy
```

ROC for KNN

```
#ROC FOR KNN####
knn.pred=knn(IBM.train, IBM.test, train.Attrition, k=1)
x=as.numeric(knn.pred)
roc(IBM.test$Attrition,x,plot=TRUE)
```



```
##
## Call:
## roc.default(response = IBM.test$Attrition, predictor = x, plot = TRUE)
##
## Data: x in 204 controls (IBM.test$Attrition 0) < 128 cases (IBM.test$Attri
tion 1).
## Area under the curve: 0.9168
```

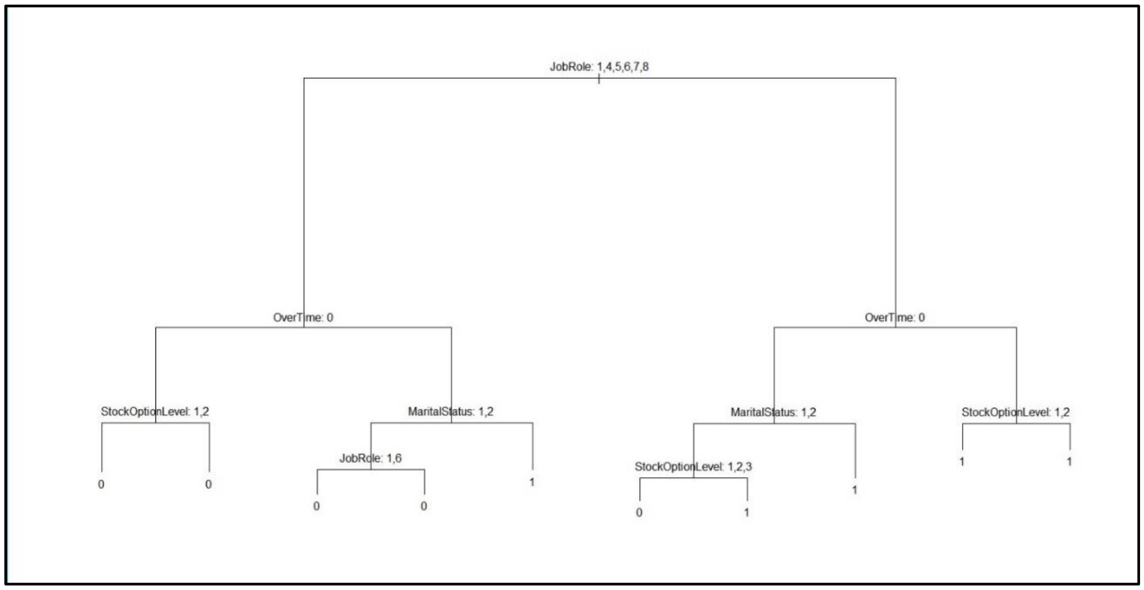
Here, KNN gives an accuracy of about 92.16% and AUC as 91.68 for k=1. When k=1, the boundary is highly flexible and follows the error. The results of KNN are not consistent with the theory. The reason is that our dataset consists of categorical and continuous variables which KNN is not able to handle.

Classification Tree

```
####Classification Tree####
library(tree)
```

```
set.seed(3)
tree.IBM=tree(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus, IBM,
subset = train)
summary(tree.IBM)
## Classification tree:
## tree(formula = Attrition ~ JobRole + StockOptionLevel + OverTime +
##     MaritalStatus, data = IBM, subset = train)
## Number of terminal nodes:  10
## Residual mean deviance:  0.926 = 1220 / 1317
## Misclassification error rate: 0.208 = 276 / 1327
```

```
plot(tree.IBM)
text(tree.IBM ,pretty =0)
```



The Tree has 10 nodes with JobRole, OverTime, MaritalStatus and StockOptionLevel as the prediction terms. The first split can be interpreted as,if the Job Role is of the type-1,4,5,6,7,8 than we track the left branch. Now, if Overtime is 0, track the left branch and follow the same approach for StockOptionLevel.

```
tree.pred=predict(tree.IBM ,IBM.test ,type ="class")
table(tree.pred ,IBM.test$Attrition)
```

```
##
## tree.pred   0   1
##         0 178  35
##         1  26  93
```

```
mean(tree.pred!=IBM.test$Attrition)
```

```
## [1] 0.1837349
```

The mean error rate is 18.37%

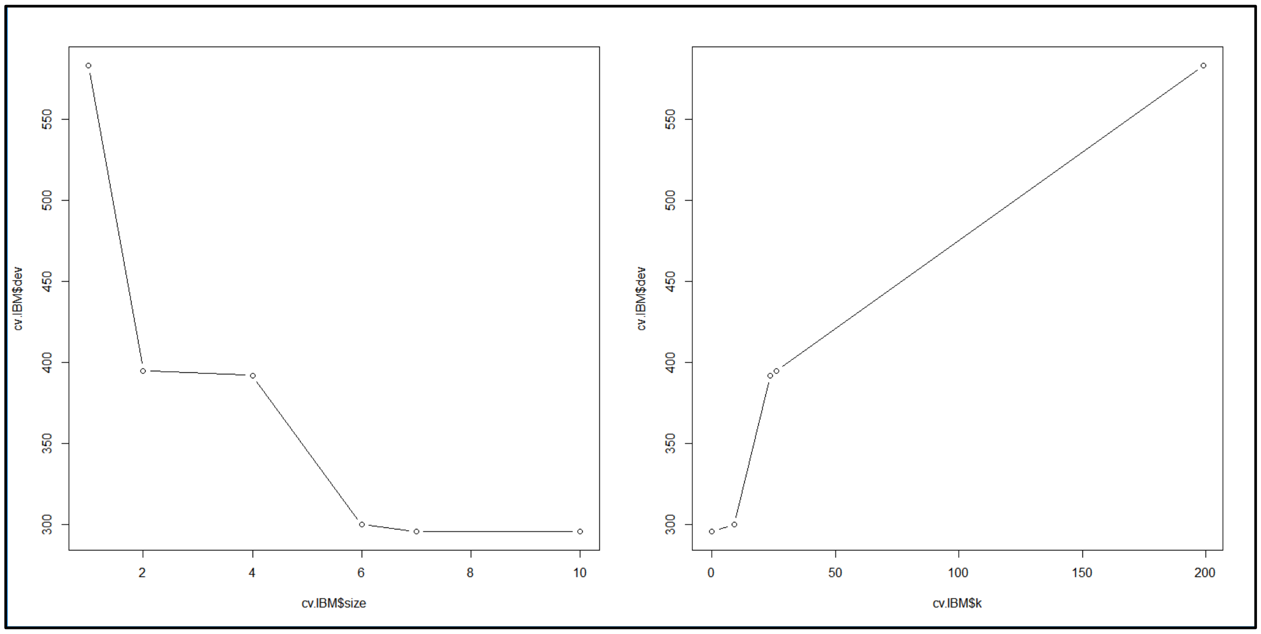
Cross-Validation and Pruning of Classification Tree

```
####CV for classification tree####
set.seed (3)
cv.IBM=cv.tree(tree.IBM ,FUN=prune.misclass )
cv.IBM
```

```
## $size
## [1] 10  7  6  4  2  1
##
## $dev
## [1] 296 296 300 392 395 583
##
## $k
## [1]  -Inf   0.0   9.0  23.5  26.0 199.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
#dev stands for classification error.

#THe lowest classification error rate(296) occurs for tree of size 10
& 7
par(mfrow =c(1,2))
plot(cv.IBM$size ,cv.IBM$dev ,type="b")
plot(cv.IBM$k ,cv.IBM$dev ,type="b")
```



We can see from the graphs that as the tree size increases, there is a drop in the classification error. As, k increases, classification error increases.

```
prune.IBM=prune.misclass(tree.IBM ,best =7)
tree.pred=predict(prune.IBM , IBM.test ,type="class")
table(tree.pred ,IBM.test$Attrition)
```

```
##
## tree.pred   0   1
##         0 178  35
##         1  26  93
```

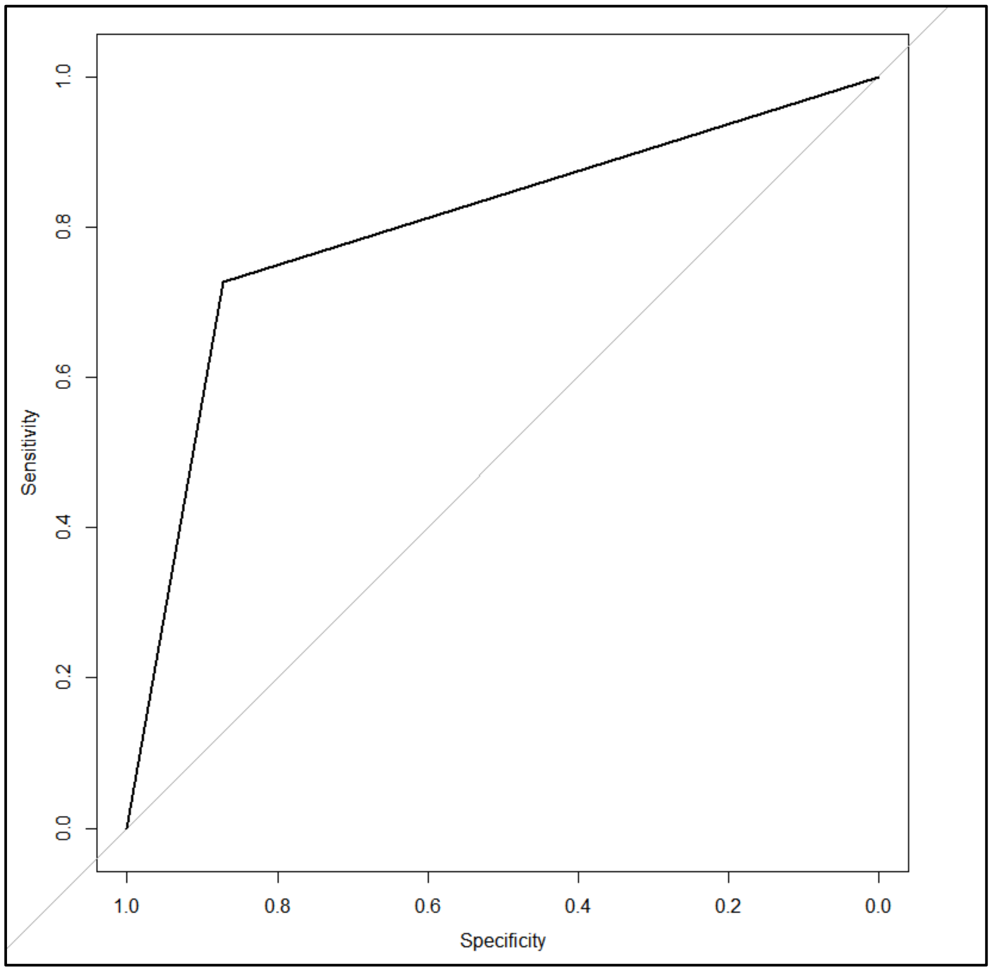
```
mean(tree.pred!=IBM.test$Attrition)
```

```
## [1] 0.1837349
```

ROC for Classification Tree

```
#####ROC for CART####
u = as.numeric(tree.pred)
roc(IBM.test$Attrition, u, plot = TRUE)
```

```
##
## Call:
## roc.default(response = IBM.test$Attrition, predictor = u, plot = TRUE)
##
## Data: u in 204 controls (IBM.test$Attrition 0) < 128 cases (IBM.test$Attri
tion 1).
## Area under the curve: 0.7996
```



Support Vector Classifier

```
####SVC####
library(e1071)
svmfit =svm(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus, data=I
BM.train , kernel="linear", cost =10,scale =TRUE )
tune.out=tune(svm ,Attrition~., data=IBM.train ,kernel ="linear", ranges =lis
t(cost=c(0.001 , 0.01, 0.1, 1,5,10,100) ))
bestmod =tune.out$best.model
summary(bestmod )
```

```
##
## Call:
## best.tune(method = svm, train.x = Attrition ~ ., data = IBM.train,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)),
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.01098901
##
## Number of Support Vectors:  396
##
##  ( 205 191 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
ypred=predict(bestmod, IBM.test)
table(predict=ypred, truth=IBM.test$Attrition)
```

```
##        truth
## predict   0   1
##       0 183  18
##       1  21 110
```

```
mean(ypred!=IBM.test$Attrition)
```

```
## [1] 0.1174699 #The mean error rate is 11.74%
```

#Best model has cost=0.1

CV for SVC

```
####CV for SVC####
tuned=tune.svm(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus, dat
a=IBM , kernel="linear", cost = 10^(-2:2), scale=TRUE)
summary(tuned)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.2175904
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-02 0.2332494 0.03463696
## 2 1e-01 0.2175904 0.01984765
## 3 1e+00 0.2175904 0.02141131
## 4 1e+01 0.2236145 0.02193227
## 5 1e+02 0.2236145 0.02193227
```

```
bestmod=tuned$best.model
summary(bestmod)
```

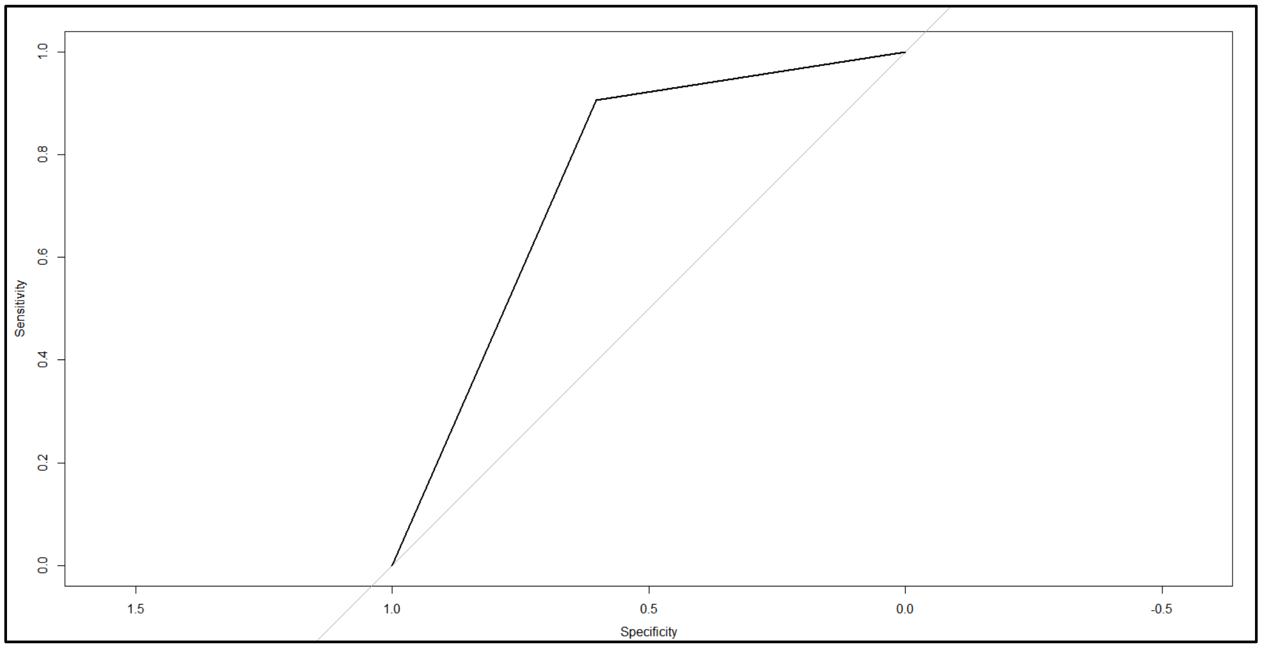
```
##
## Call:
## best.svm(x = Attrition ~ JobRole + StockOptionLevel + OverTime +
##     MaritalStatus, data = IBM, cost = 10^(-2:2), kernel = "linear",
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##       cost:  0.1
##       gamma:  0.06666667
##
## Number of Support Vectors:  895
##
##  ( 449 446 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

ROC for Support Vector Classifier

```
svm=svm(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus, data=IBM.t
rain , kernel="linear", cost =1, scale=TRUE)
svm.pred=predict(svm,IBM.test, decision.values = TRUE)
a=as.numeric(svm.pred)
q = roc(IBM.test$Attrition, a)
auc(q)
```

```
## Area under the curve: 0.7849
```

```
plot.roc(q)
```



We have taken "linear" kernel. The mean error rate is 11.74% but the CV-error results to be 21.75% for cost=0.1. The AUC is 78.49%.

Support Vector Machines

```
#####SVM#####
library(e1071)
svmfit=svm(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus, data=IB
M.train, kernel ="radial", gamma =1, cost=1, scale=TRUE)
#plot(svmfit , IBM.train)
summary(svmfit)
```

```
##
## Call:
## svm(formula = Attrition ~ JobRole + StockOptionLevel + OverTime +
##     MaritalStatus, data = IBM.train, kernel = "radial", gamma = 1,
##     cost = 1, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##
## Number of Support Vectors:  633
##
##  ( 329 304 )
##
##
## Number of Classes:  2
```

```
##
## Levels:
##  0 1
```

Cross-validation for SVM

```
#####CV for SVM####
tuned=tune.svm(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus, dat
a=IBM.train, kernel ="radial", gamma = 10^(-6:-1), cost = 10^(-2:2), scale=TR
UE)
summary(tuned)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma cost
##    0.1  100
##
## - best performance: 0.2155161
##
## - Detailed performance results:
##    gamma  cost     error dispersion
## 1  1e-06 1e-02 0.4393028 0.03277548
## 2  1e-05 1e-02 0.4393028 0.03277548
## 3  1e-04 1e-02 0.4393028 0.03277548
## 4  1e-03 1e-02 0.4393028 0.03277548
## 5  1e-02 1e-02 0.4393028 0.03277548
## 6  1e-01 1e-02 0.4393028 0.03277548
## 7  1e-06 1e-01 0.4393028 0.03277548
## 8  1e-05 1e-01 0.4393028 0.03277548
## 9  1e-04 1e-01 0.4393028 0.03277548
## 10 1e-03 1e-01 0.4393028 0.03277548
## 11 1e-02 1e-01 0.3736956 0.06880180
## 12 1e-01 1e-01 0.2403167 0.04403223
## 13 1e-06 1e+00 0.4393028 0.03277548
## 14 1e-05 1e+00 0.4393028 0.03277548
## 15 1e-04 1e+00 0.4393028 0.03277548
## 16 1e-03 1e+00 0.3518854 0.04530059
## 17 1e-02 1e+00 0.2403338 0.04351880
## 18 1e-01 1e+00 0.2200103 0.03701419
## 19 1e-06 1e+01 0.4393028 0.03277548
## 20 1e-05 1e+01 0.4393028 0.03277548
## 21 1e-04 1e+01 0.3518854 0.04530059
## 22 1e-03 1e+01 0.2433413 0.04676677
## 23 1e-02 1e+01 0.2192584 0.04157923
## 24 1e-01 1e+01 0.2237924 0.03181240
## 25 1e-06 1e+02 0.4393028 0.03277548
## 26 1e-05 1e+02 0.3518854 0.04530059
## 27 1e-04 1e+02 0.2433413 0.04676677
## 28 1e-03 1e+02 0.2185065 0.04075704
## 29 1e-02 1e+02 0.2200216 0.03704217
## 30 1e-01 1e+02 0.2155161 0.03113311
```

```
bestmod=tuned$best.model
summary(bestmod)
```

```
##
## Call:
## best.svm(x = Attrition ~ JobRole + StockOptionLevel + OverTime +
##     MaritalStatus, data = IBM.train, gamma = 10^(-6:-1), cost = 10^(-2:2),
##     kernel = "radial", scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  100
##       gamma:  0.1
##
## Number of Support Vectors:  598
##
##  ( 300 298 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
#best model at cost=100 and gamma=0.1
ypred=predict(bestmod, IBM.test)
table(predict=ypred, truth=IBM.test$Attrition)
```

```
##        truth
## predict   0   1
##       0 176  32
##       1  28  96
```

```
mean(ypred!=IBM.test$Attrition)
```

```
## [1] 0.1807229
```

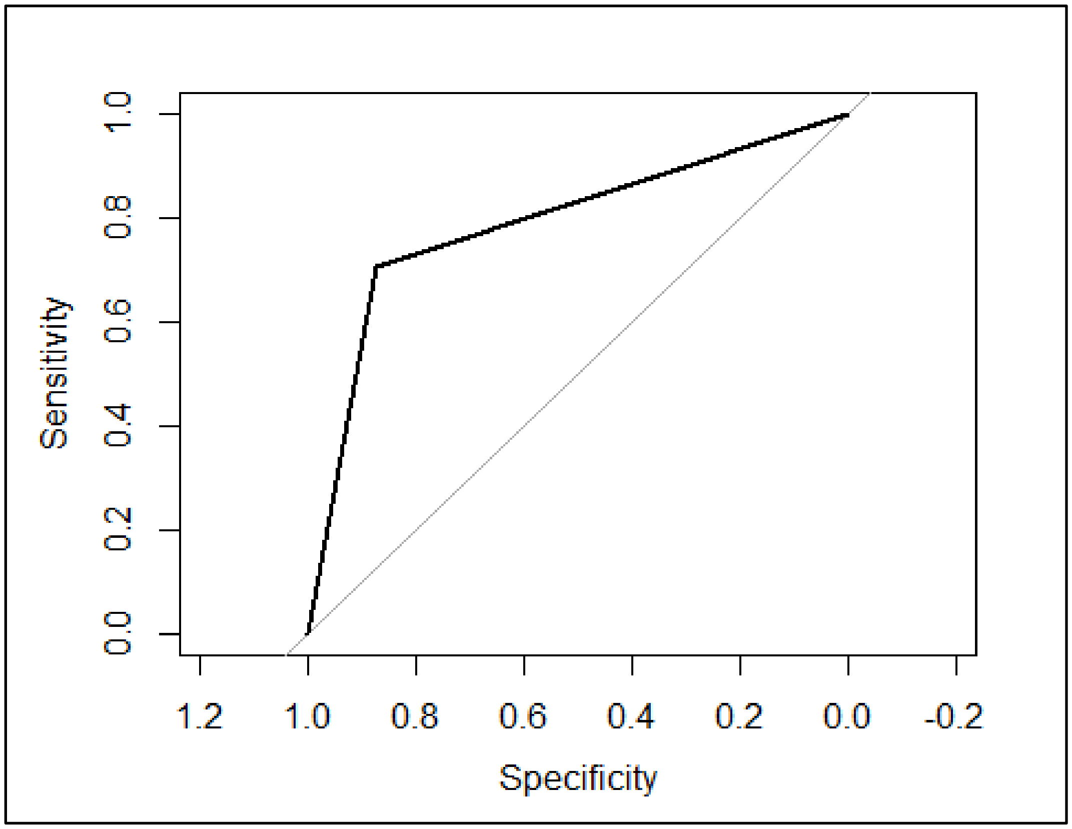
The mean error rate is 18.07% for SVM with kernel “Radial”. The CV-error is 21.55% for cost of 100 and gamma 0.1.

ROC for SVM

```
####ROC for SVM####
svm=svm(Attrition~JobRole+StockOptionLevel+OverTime+MaritalStatus, data=IBM.t
rain, kernel ="radial", gamma=0.1, cost =1, scale=TRUE)
svm.pred=predict(svm,IBM.test, decision.values = TRUE)
a=as.numeric(svm.pred)
q = roc(IBM.test$Attrition, a)
auc(q)
```

```
## Area under the curve: 0.7878
```

```
plot.roc(q)
```



The AUC is 78.78%. There is a drop in performance of SVM as compared to SVC. This again proof that as the boundary becomes flexible the performance drops.

Deep Learning

Start by loading the package ‘h2o’ and calling the server. [11]

```
library(h2o)
```

```
h2o.init(nthread=-1)
```

H2o requires the data to be imported from outside R. The data set is hence first written using write.csv and then imported.

```
data <- write.csv(IBM, file="IBM_New.csv”)
```

```
data <- h2o.importFile(path = normalizePath("C:/Users/Neehar/Desktop/TAMU/Sem
1/613/Project/IBM_New.csv"))
```

Observing the structure of data,

```
str(data)
```

```
##  - attr(*, "data")='data.frame': 10 obs. of  31 variables:
##   ..$ Attrition               : num  0 0 0 0 0 0 0 0 0 0
##   ..$ BusinessTravel          : num  2 2 2 2 2 2 2 2 2 3
##   ..$ DailyRate               : num  3 0 3 1 0 2 1 3 1 1
##   ..$ Department              : num  3 3 2 3 3 3 2 2 3 2
##   ..$ DistanceFromHome        : num  1 1 3 1 0 1 1 1 3 2
##   ..$ Education               : num  3 4 4 1 3 2 1 2 4 4
##   ..$ EducationField          : num  2 2 2 3 4 2 2 2 3 4
##   ..$ EnvironmentSatisfaction : num  3 2 1 3 1 2 1 4 3 2
##   ..$ Gender                  : num  0 0 0 0 1 1 0 0 0 1
##   ..$ HourlyRate              : num  1 0 2 3 2 3 3 1 2 3
##   ..$ JobInvolvement          : num  3 2 3 2 3 2 3 3 3 3
##   ..$ JobLevel                : num  3 1 2 2 2 2 5 2 2 3
##   ..$ JobRole                 : num  8 9 1 8 8 8 6 3 8 6
##   ..$ JobSatisfaction         : num  4 3 3 1 3 4 2 2 4 3
##   ..$ MaritalStatus           : num  2 2 2 2 3 3 1 3 3 3
##   ..$ MonthlyIncome           : num  3 0 2 2 2 2 3 2 2 3
##   ..$ MonthlyRate             : num  3 1 0 2 0 1 2 1 1 2
##   ..$ NumCompaniesWorked      : num  1 2 3 1 1 1 2 3 0 3
##   ..$ OverTime                : num  1 0 0 0 0 1 0 0 1 0
##   ..$ PercentSalaryHike       : num  1 2 1 0 2 0 1 2 3 2
##   ..$ PerformanceRating       : num  3 3 3 3 3 3 3 3 4 3
##   ..$ RelationshipSatisfaction: num  4 2 2 1 4 2 1 4 4 3
##   ..$ StockOptionLevel        : num  3 0 2 0 0 0 2 0 0 0
##   ..$ TotalWorkingYears       : num  2 3 2 0 0 0 3 3 2 3
##   ..$ TrainingTimesLastYear   : num  1 1 1 0 1 1 2 1 1 1
##   ..$ WorkLifeBalance         : num  3 4 2 4 3 3 2 4 1 2
##   ..$ YearsAtCompany          : num  2 2 1 2 2 2 3 3 3 3
##   ..$ YearsInCurrentRole      : num  2 3 1 1 2 2 2 3 3 3
##   ..$ YearsSinceLastPromotion : num  1 2 0 0 0 1 1 2 1 1
##   ..$ YearsWithCurrManager    : num  0 1 1 2 2 2 2 3 0 3
##   ..$ Age                     : num  1 1 2 0 1 0 3 2 3 3
```

We see that all the variables are numeric in nature including the response 'Attrition'. Hence, we convert Attrition to a factor, leaving the rest as it is taken care of by the h2o.deeplearning function. Categorical encoding can be done within h2o.deeplearning. One Hot Encoding is done as it transforms features

```
data$Attrition = h2o.asfactor(data$Attrition)
```

Splitting the data into train and test for developing the model to train and test with the test data set.

```
split <- h2o.splitFrame( data, c(0.8), seed = 1 )
train <- h2o.assign( split[[1]], "train" ) # 80%
test  <- h2o.assign( split[[2]], "test" )  # 20%
```

Now using the h2o.deeplearning function to develop the model and validating it simultaneously using *k* fold cross validation with number of folds being 5.

```
model <- h2o.deeplearning( x=2:31, y=1, train, distribution = "bernoulli",
                                activation="Rectifier", nfolds=5, seed = 3,
                             hidden=c(60,30),   input_dropout_ratio=0.15,
                             categorical_encoding="OneHotInternal",
                             epochs=100,  variable_importances = TRUE )
```

The activation function used here is Rectifier(ReLU) due to its advantages and versatility. Distribution used is Bernoulli as it's a classification problem with response being in two categories. Two hidden layers are used in this model with the first layer having 60 nodes while the second layer has 30. The dropout ratio is taken as 15% of the inputs to reduce overfitting and produce a simpler model. A higher value was not chosen as there aren't many rows in the training data to compensate for the missing neurons. 100 iterations are run for this model. Categorical encoding is One Hot Internal i.e it generates a Boolean column for each category. Only one of the columns can take the value 1 while others have 0. This generally works for many machine learning algorithms. [12]

Model output:

```
model
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: deeplearning
## Model ID:  DeepLearning_model_R_1513011486164_345
## Status of Neuron Layers: predicting Attrition, 2-class classification,
bernoulli distribution, CrossEntropy loss, 3,752 weights/biases, 54.8 KB,
146,520 training samples, mini-batch size 1

##   layer units      type dropout       l1       l2 mean_rate rate_rms
## 1     1    30     Input 15.00 %
## 2     2    60 Rectifier  0.00 % 0.000000 0.000000  0.001682 0.001055
## 3     3    30 Rectifier  0.00 % 0.000000 0.000000  0.014872 0.060835
## 4     4     2   Softmax          0.000000 0.000000  0.001748 0.000955
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1
## 2 0.000000   -0.006219   0.175576  0.434055 0.099914
## 3 0.000000   -0.007754   0.167238  0.983764 0.077313
## 4 0.000000    0.219735   0.982086  0.007399 0.033492
##
## H2OBinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## MSE:  0.05199304
## RMSE:  0.2280198
## LogLoss:  0.1769449
## Mean Per-Class Error:  0.05933144
## AUC:  0.9874129
## Gini:  0.9748258
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal
threshold:
##          0   1    Error      Rate
## 0      725  27 0.035904   =27/752
## 1       48 532 0.082759   =48/580
## Totals 773 559 0.056306  =75/1332
```

Interpreting the output:

The h2o.deeplearning() framework automatically identifies the loss function and final activation function to use for classification. The loss function used here is of cross entropy or log loss function i.e the loss increases as the predicted probability diverges from the true value.

The Mean per class error calculated for the training model is 0.0593 as seen and the confusion matrix is observed. There are 75 misclassifications out of 1332 training observations.

Rest of the output:

```
## H2OBinomialMetrics: deeplearning
## ** Reported on cross-validation data. **
## ** 5-fold cross-validation on training data (Metrics computed for combined
holdout predictions) **
##
## MSE:  0.2717899
## RMSE:  0.5213347
## LogLoss:  1.584544
## Mean Per-Class Error:  0.1884996
## AUC:  0.8806963
## Gini:  0.7613926
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal
threshold:
##          0   1    Error       Rate
## 0      580 172 0.228723   =172/752
## 1       86 494 0.148276    =86/580
## Totals 666 666 0.193694  =258/1332
```

As cross validation is done simultaneously by the h2o.deeplearning() framework on the training data set, the results of that are shown above. It's observed that the misclassification errors are 258 out of all the training data. The error rate = 0.188

Another benefit of using deep learning with h2o is its ability to identify important features with h2o.varimp() function.

```
head(h2o.varimp(model))

## Variable Importances:
##                   variable relative_importance scaled_importance percentage
## 1 EnvironmentSatisfaction            1.000000          1.000000   0.039966
## 2                JobLevel            0.953338          0.953338   0.038101
## 3               DailyRate            0.949478          0.949478   0.037947
## 4         JobSatisfaction            0.949167          0.949167   0.037934
## 5      YearsInCurrentRole            0.911106          0.911106   0.036413
## 6          BusinessTravel            0.882115          0.882115   0.035254
```

We see that environment satisfaction has the highest importance with Job level and Daily rate coming in second and third. These results are very pragmatic as the attrition of an employee is majorly affected by the work environment he/she is in, if it is conducive or discouraging. So is Job Level or the hierarchy of the position he/she is and the Daily Rate.

Next, we use this model to test with the test data and obtain the ROC curve to understand how it performs on new data.

```
predictions <- h2o.predict(model, newdata=test)
```

```
mean(predictions$predict != test$Attrition)
```

```
## [1] 0.1498471
```

For the test data, the misclassification error rate = 0.149

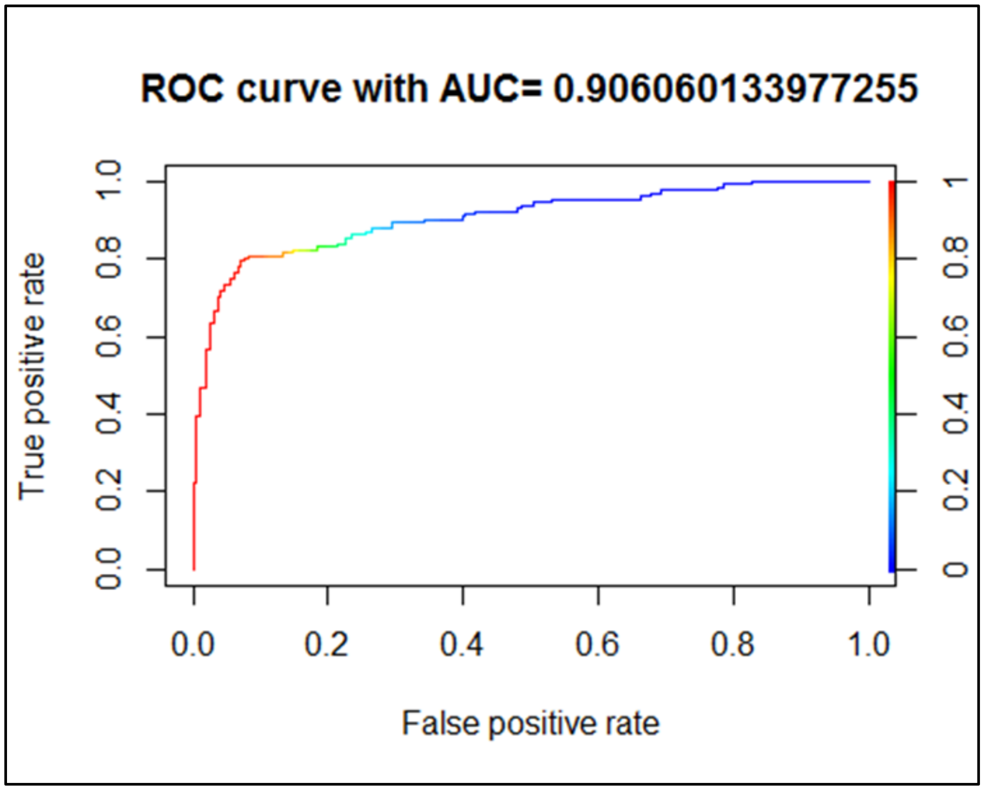
Obtaining the ROC curve:

```
pred=as.data.frame(predictions)
```

```
test_h2o = as.data.frame(test)
```

```
roc_auc <- function(probabilities,dataset)
  { #Command - roc_auc(probabilities,dataset)
  #probabilities are those obtained from predict function #dataset is the act
ual data (0s and 1s)
library(ROCR)
  pr=prediction(probabilities, dataset)
  prf=performance(pr,measure = "tpr", x.measure = "fpr")
  auc=performance(pr,measure = "auc")
  auc=auc@y.values[[1]]
  plot(prf,colorize=TRUE,main=paste("ROC curve with AUC=",auc)) }
```

```
roc_auc(pred[,3],test_h2o[,1])
```



As a designer, we want to know every single time any of our employee is unhappy. To ensure that whenever our employee is unhappy, our model performs true prediction hence our true positive rate should be as maximum as possible. For neural network algorithm although our mean square error is less compared to Random Forest, but AUC and TPR is maximum therefore it is advisable to use neural network prediction model to identify what all predictors should be kept in mind to avoid employee attrition.

Market Basket Analysis

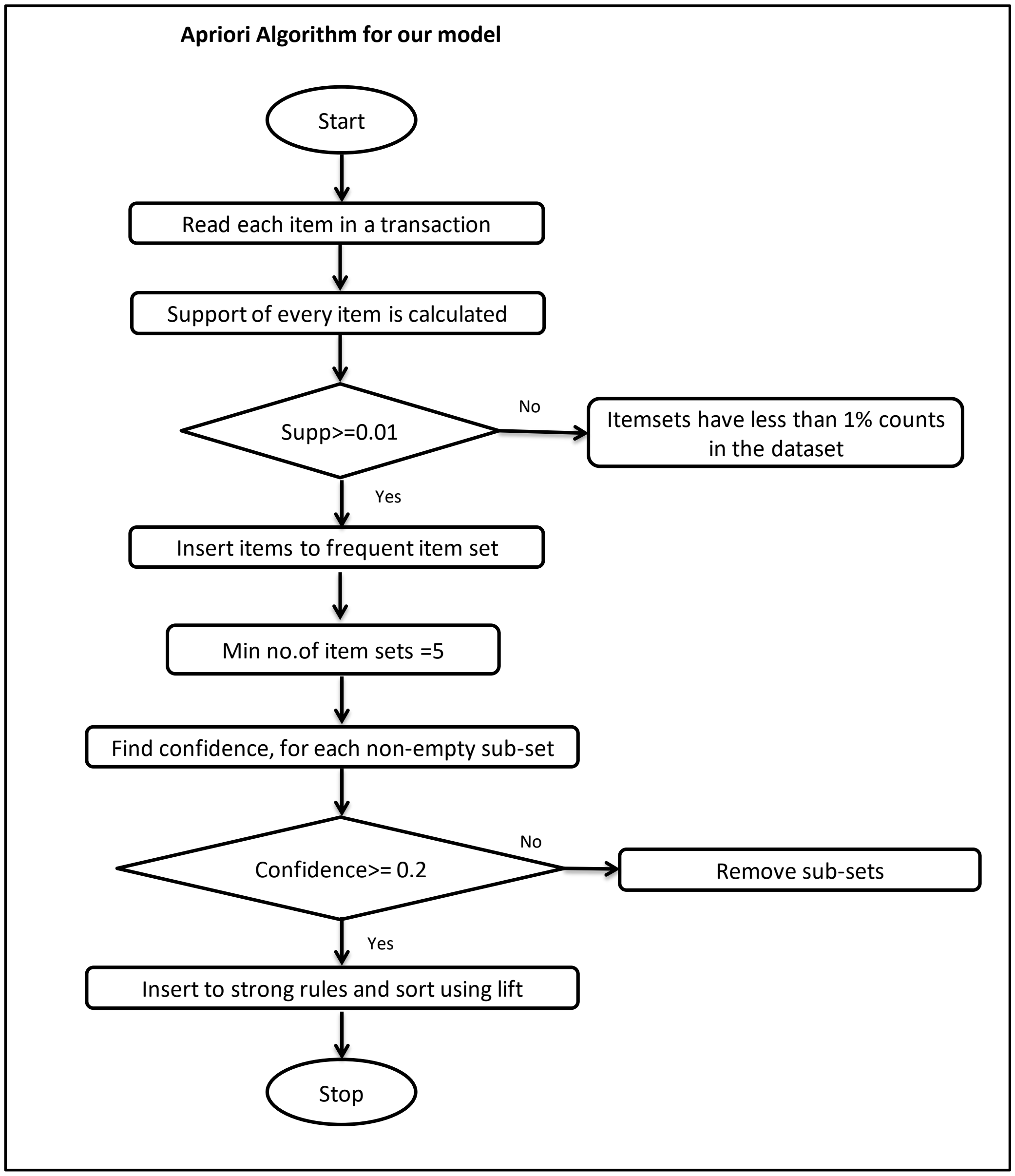
Introduce the library arules for the implementation of Apriori algorithm. It is used for frequent dataset mining using association rules.

```
library(arules)
```

All the predictors in the LHS were selected based on the result we got from neural network as it was the most accurate model in terms of AUC. We wanted to predict values of particular variables of the following itemset [13]: -

{EnvironmentSatisfaction, JobLevel, BusinessTravel, JobSatisfaction, YearsInCurrentRole, DailyRate}

Algorithm for the following is as explained with the help of flow chart.


Apriori Algorithm for our model
Start
Read each item in a transaction
Support of every item is calculated
Supp>=0.01
No
Itemsets have less than 1% counts in the dataset
Yes
Insert items to frequent item set
Min no.of item sets =5
Find confidence, for each non-empty sub-set
Confidence>= 0.2
No
Remove sub-sets
Yes
Insert to strong rules and sort using lift
Stop

```
R Code:-

rules <- apriori(IBM,parameter = list(minlen=5, supp=0.01,
conf=0.2),appearance =
list(rhs=c("Attrition=1"),default="none",lhs=c("EnvironmentSatisfaction=1","E
nvironmentSatisfaction=2","EnvironmentSatisfaction=3","EnvironmentSatisfactio
n=4","JobLevel=1","JobLevel=2","JobLevel=3","JobLevel=4","JobLevel=5","Busine
ssTravel=1","BusinessTravel=2","BusinessTravel=3",
"JobSatisfaction=1","JobSatisfaction=2","JobSatisfaction=3","JobSatisfaction=
4","YearsInCurrentRole=0","YearsInCurrentRole=1","YearsInCurrentRole=2","Year
sInCurrentRole=3",
"DailyRate=0","DailyRate=1","DailyRate=2","DailyRate=3")),control =
list(verbose=F))
```

Displaying top 5 rules among 152 printed in decreasing order for lift values: -

```
##       lhs                               rhs             support confidence
lift count
## [1]   {BusinessTravel=3,
##        DailyRate=1,
##        JobLevel=1,
##        YearsInCurrentRole=0}      => {Attrition=1} 0.01024714  1.0000000
2.3333333    17
## [2]   {BusinessTravel=2,
##        DailyRate=2,
##        JobLevel=1,
##        JobSatisfaction=1,
##        YearsInCurrentRole=2}      => {Attrition=1} 0.01265823  0.9545455
2.2272727    21
## [3]   {DailyRate=1,
##        EnvironmentSatisfaction=4,
##        JobLevel=1,
##        JobSatisfaction=3}         => {Attrition=1} 0.01205546  0.9523810
2.2222222    20
## [4]   {DailyRate=2,
##        EnvironmentSatisfaction=3,
##        JobLevel=1,
##        YearsInCurrentRole=2}      => {Attrition=1} 0.01928873  0.9411765
2.1960784    32
## [5]   {BusinessTravel=2,
##        DailyRate=2,
##        EnvironmentSatisfaction=3,
##        JobLevel=1,
##        YearsInCurrentRole=2}      => {Attrition=1} 0.01567209  0.9285714
2.1666667    26
```

Now we have a pretty good idea that for the defined itemset following values have the most influence and should be taken care by the HR department.

Sr. No	Variable	Level	Value
1	BusinessTravel	3	Travel Frequently
2	DailyRate	1	Fair
3	JobLevel	1	Seniority level = 0
4	YearsInCurrentRole	0	Low
Response	Attrition	1	Yes

With MBA we can also remove redundant rules, so if any there is any major rule and a sub rule having same value for lift parameter then it can be removed. Hence pruning was performed with the help of subset function to trim down the number of rules and further limit predictor space which leads to better interpretation. We can further inspect all the rules which are present in the superset and assign rules.pruned to the new set of rules.

```
R Code:-
```

```
superset.matrix <- is.subset(rules.sorted@lhs,y=NULL,sparse=FALSE)
superset <- rowSums(superset.matrix, na.rm=T) ==1
which(superset)
```

```
rules.pruned <- rules.sorted[superset]
inspect(rules.pruned)
```

```
Number of rules were reduced to 77 from 152.
```

Data Visualization

Introduce the library arulesViz.

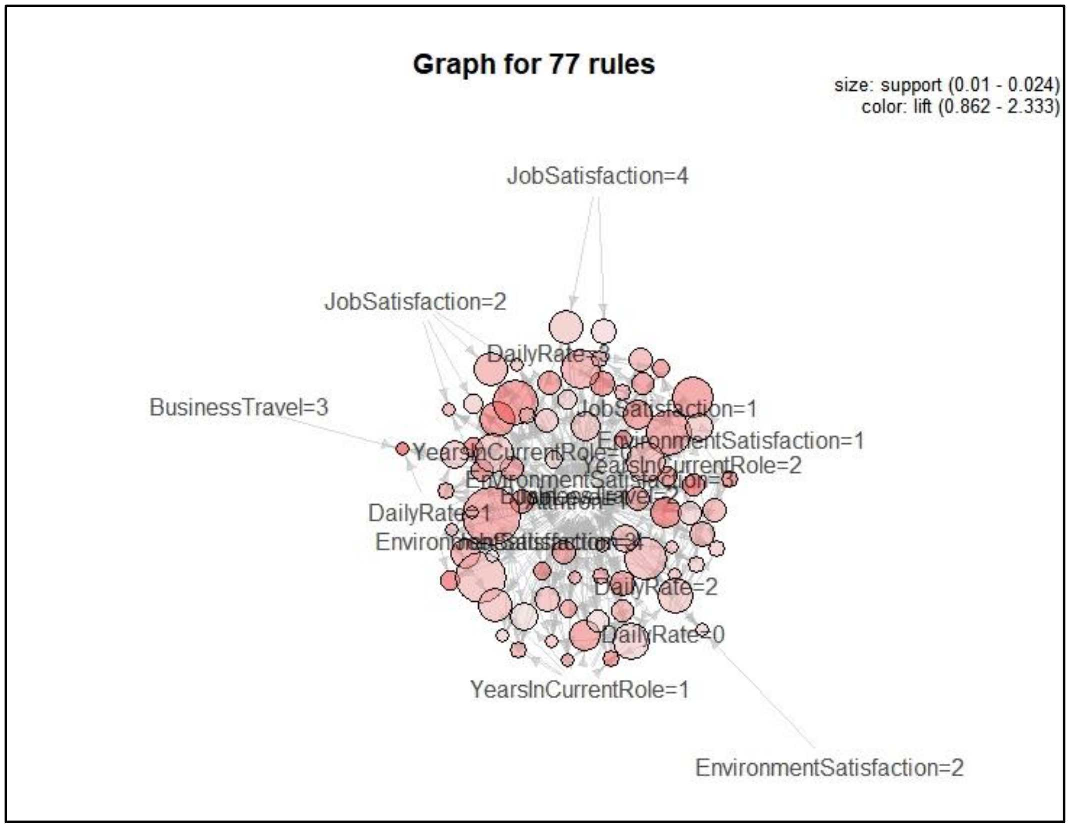
```
library(arulesViz)
```

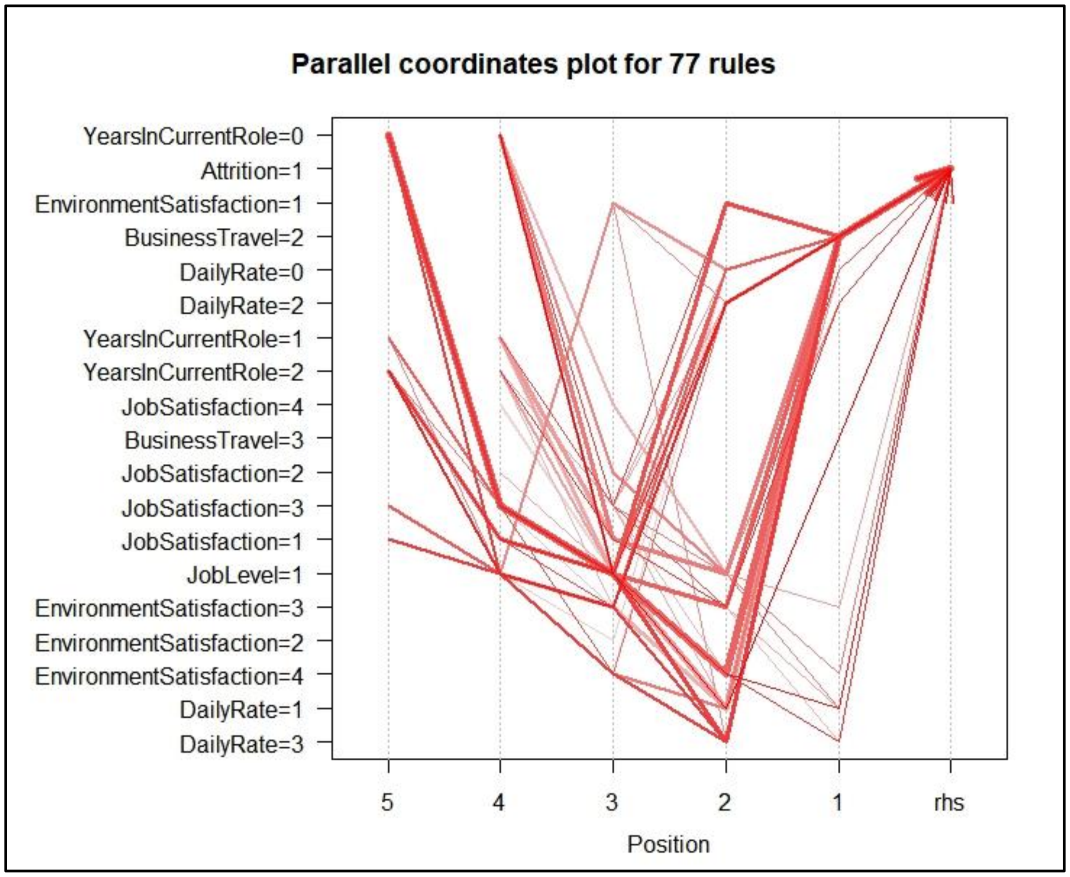
```
plot(rules.pruned, method="graph", interactive=TRUE, shading=NA)

plot(rules.pruned, method="paracoord", control=list(reorder=TRUE))
```

Plotting graphs for data visualization, following codes plot graph which differentiates among all the rules presented based on color and sizes, as rules having high support values have bigger sizes and rules having high lift are represented by a darker color.

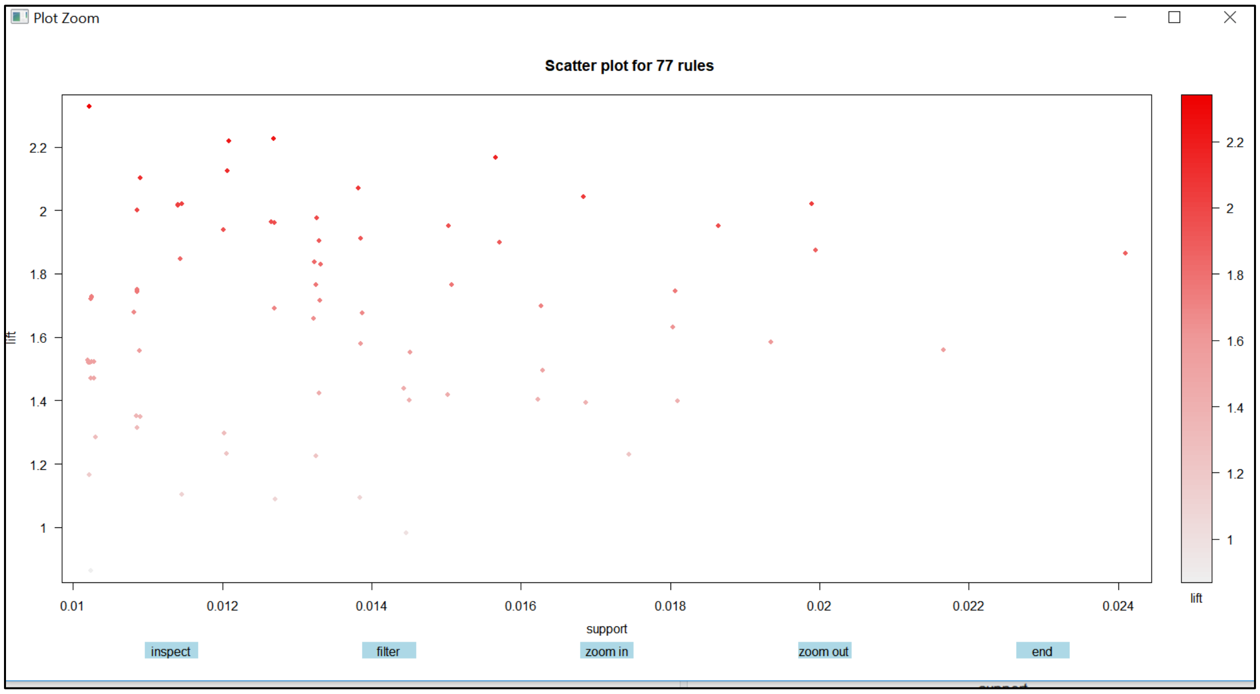
Parallel coordinates plots are designed to visualize multidimensional data where each dimension is displayed separately on the x-axis and the y-axis is shared. Each data point is represented by a line connecting the values for each dimension


Graph for 77 rules
size: support (0.01 - 0.024)
color: lift (0.862 - 2.333)
JobSatisfaction=4
JobSatisfaction=2
DailyRate=3
BusinessTravel=3
JobSatisfaction=1
EnvironmentSatisfaction=1
YearsInCurrentRole=0
YearsInCurrentRole=2
EnvironmentSatisfaction=3
DailyRate=1
DailyRate=2
DailyRate=0
YearsInCurrentRole=1
EnvironmentSatisfaction=2


Parallel coordinates plot for 77 rules
YearsInCurrentRole=0
Attrition=1
EnvironmentSatisfaction=1
BusinessTravel=2
DailyRate=0
DailyRate=2
YearsInCurrentRole=1
YearsInCurrentRole=2
JobSatisfaction=4
BusinessTravel=3
JobSatisfaction=2
JobSatisfaction=3
JobSatisfaction=1
JobLevel=1
EnvironmentSatisfaction=3
EnvironmentSatisfaction=2
EnvironmentSatisfaction=4
DailyRate=1
DailyRate=3
5
4
3
2
1
rhs
Position

We have also included an interactive plot in which you have to select a point on the graph and it provides you with corresponding values of LHS and RHS. For exploration, the scatter plot method offers interactive features for selecting and zooming. Interaction is activated using interactive=TRUE.

```
sel <- plot(rules.pruned, measure=c("support", "lift"), shading="lift",
interactive=TRUE)
```



COMPARISON

Here we have presented a table where we compare every method used, its result parameters and performance associated with other methods.

S.No	METHODOLOGY	RESULT PARAMETERS	PERFORMANCE
1	Random Forest	Mean error rate: 0.09638554 Important variables: are JobRole, OverTime, StockOptionLevel and MaritalStatus AUC:0.8939	The Mean error rate for Random Forest is very low and the AUC is high. Therefore, the results of RF are quite reliable and the results can be used for further alaysis. Initially the predictors for further classification techniques will be given by neural network.
2	Boosting	Important variables: JobRole, EducationField, StockOptionLevel and OverTime	The results of Boosting almost matches with the results of Random Forest.
3	Logistic Regression	Mean error rate: 0.1957831 CV-error: 0.1481327 AUC: 0.8915	Although the mean error rate matches for logistic regression and LDA, the CV error clearly show that logistic outperforms LDA. Also the AUC of logistic regression is high than LDA. The reason for such results is that logistic does not make any underlying assumptions of normality
4	LDA	Mean error rate: 0.1957831 CV-error: 0.2163954 AUC: 0.7854	
5	QDA	Mean error rate: 0.2801205 CV-error: 0.2730561 AUC: 0.75	The error rate and AUC is worse than LDA and logistic. This suggest that a flexible classification leads to high variance and thus contributes to increase in error.
6	KNN	Best model is at k=1 with 92.16% accuracy AUC: 0.9168	As we are using a dataset with a combination of continuous and categorical data, KNN resulted in very unstable and inflated accuracy. This is because KNN method only uses data which is numeric in nature
7	Classification Tree	Mean Error rate:0.1837349 The lowest CV-error is found for tree with 7 & 10 nodes and the error rate is 0.1837349 AUC:0.7996	The results of classification tree are comparable with logistic and another advantage provided by this method is ease of interpretation.

8	Support Vector Classifier	Mean error rate: 0.1174699 The best model has cost=0.1 CV-error:0.2175904 AUC: 0.7849	A linear kernel performs better than radial kernel based on the mean error rate. Suggesting again a less flexible boundary fits better than highly flexible boundary.
9	Support Vector Machines	Mean error rate: 0.1807229 The best model has cost=100 and gamma=0.1 with CV-error:0.2155161 AUC: 0.7878	
10	Deep Learning Neural Network	Classification Accuracy=0.85 AUC = 0.906 CV Error = 0.189	Although the classification accuracy is lower compared to others, it has higher AUC i.e. greater sensitivity which was essential in our project.

As can be seen from the table Deep learning outperforms every other method and hence should be used as a final model.

REFERENCES

[1] Nitesh V. Chawla et al. SMOTE: Synthetic Minority Over-Sampling Technique (2002), Journal of Artificial Intelligence Research 16 321–357

[2] Frauke Günther, Stefan Fritsch. neuralnet: Training of Neural Networks

[3] Christos Stergiou, Dimitrios Siganos. Neural Networks.

[4] Jurgen Schmidhuber. Deep Learning in Neural Networks: An Overview (2014). Technical Report IDSIA

[5] Adit Deshpande. A Beginner's Guide To Understanding Convolutional Neural Networks (2016).

[6] Nitish Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting (2015), Journal of Machine Learning Research

[7] Divya Jain, Sumanlata Gautam. Implementation of Apriori Algorithm in Health Care Sector: A Survey (2013), *International Journal of Computer Science and Communication Engineering*

[8] Selva Prabhakaran. Association Mining (Market Basket Analysis)

[9] Susan Li, A Gentle Introduction on Market Basket Analysis — Association Rules (2017)

[10] Gareth James et al. An Introduction to Statistical Learning with Applications in R

[11] The H2O.ai team. R Interface for H2O (2017)

[12] Ethen Liu. H2o Deep Learning (2016)

[13] David Smith. Association Rules and Market Basket Analysis with R (2015)