MyBank's Personal Finance

Machine Learning Application

Author: Amit KULKARNI

Table Contents

Project Objective: Data pertaining to **Mybank** with regards to 20000 customers is provided. The bank is interested in floating a personal loan offer with 10% interest rates for these targeted customers. We are required to perform 3 machine learning algorithms i.e.

i. CART
ii. Random Forest
iii. Neural Networks

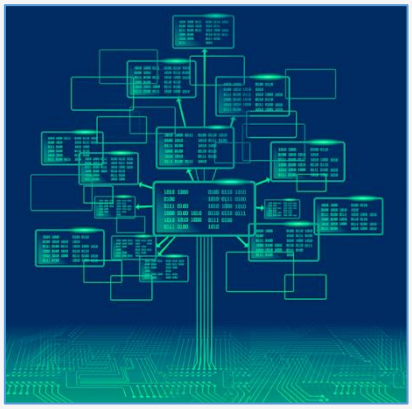
It is also expected to compare the 3 models, perform model performance evaluation and to ensure that it is not overfitting.

1. Assumptions: There are no specific assumptions made in the explanation of the solutions
2. Dataset: The provided data is in the Excel Object attached below


PL_XSELL

3. **Exploratory Data Analysis**
   3.1. **Environmental Set Up and Data Import**: The provided dataset will be analyzed in R STUDIO by importing it using the *read.csv* function. Various data mining techniques like CART, Random Forest and Neural Network will be used to test the predicted class and the models' goodness fit will be tested and also a comparison between the techniques will be made
   3.2. Install packages and invoke libraries: Variety of packages used and invoked libraries.
   3.2.1. CaTools
   3.2.2. Rpart
   3.2.3. Rpartplot
   3.2.4. Randomforest
   3.2.5. Nerualnet
   3.2.6. Data.table
   3.2.7. Scales
   3.2.8. ROCR
   3.2.9. Ineq

4. CLASSIFICATION AND REGRESSION TREE (CART) TECHNIQUE:



Dataset with respect to MYBANK customers who have responded to a survey with regards to a personal loan scheme @10% is been provided. The data will be split 80% and 20% for train and test respectively.

CART decision tree technique will be used to predict the customers' interest.

Following steps have been followed to develop the model.

i. Calling the data and splitting it in 80 and 20 percent for train and test data.

ii. Minsplit and minbucket have been defined at 100 and 10 accordingly. The test to perform the optimum Minsplit and minbucket not performed

```
setwd("C:/Users/Amit Kulkarni/Documents/R Programming/Machine Learning/data mining/Project 4")
CARTdata= read.csv("PL_XSELL.csv", header=TRUE)
library(caTools)
set.seed(3000)
split = sample.split(CARTdata$TARGET, SplitRatio = 0.80)
CARTtrain= subset(CARTdata, split==T)
CARTtest= subset(CARTdata, split==F)
CARTtrain.final=subset(CARTtrain, select=-c(1,11))
CARTtest.final=subset(CARTtest, select=-c(1,11))
```

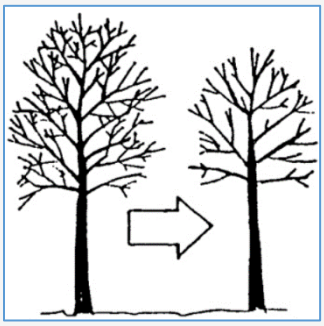
or considered since the tree will be pruned later. The model is built with the below parameters

a. Minsplit 100
b. Minbucket 10
c. Cp = 0
d. Xval/Cross Validation = 10

iii. The output is rambling and there are too many nodes that makes the output difficult to interpret and thus pruning is an efficient idea to ensure that the tree is reduced to a meaningful output.

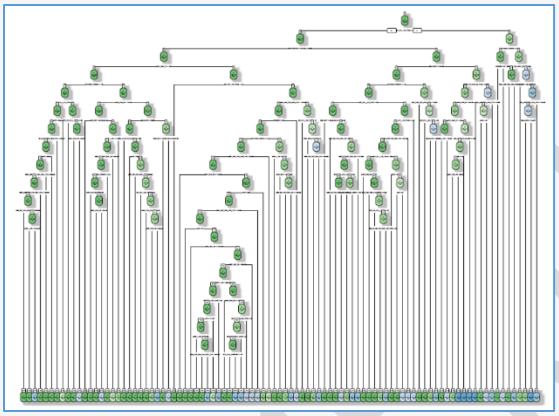
```
library(rpart)
library(rpart.plot)
setwd("C:/Users/Amit Kulkarni/Documents/R Programming/Machine Learning/data mining/Project 4")
r.ctrl=rpart.control(minsplit = 100, minbucket = 10, cp=0, xval=10)

CART_Train_Model = rpart(formula = TARGET~., data=CARTtrain.final , method="class", control = r.ctrl )
```

PRUNING THE TREE

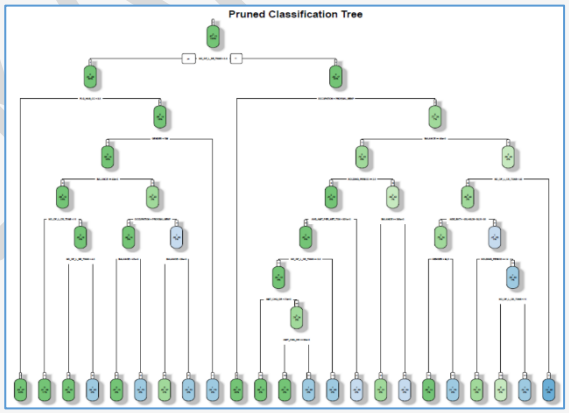


Pruning the decision tree is necessary where the decision tree has grown into manifolds and is difficult to be analyzed till the terminal node. This also is done to decide if the nodes need to split further to reduce the errors. This is also called the substitution error.

Pruning provides analysis with regards to the details of where the last nodes can be terminated. An illustration of the original decision tree until last leaf nodes in comparison to the same decision tree pruned is depicted below. (These are done on the provided dataset of MYBANK)



The original decision tree i.e. the tree before pruning



The decision tree after pruning done

To perform the pruning we need to calculate the CP or the substitution error. It is done as shown in the below codes

```
library(RColorBrewer)
printcp(CART_Train_Model)
```

The output is as below

	CP	nsplit	rel error	xerror	xstd
1	0.00298507	0	1.00000	1.00000	0.020857
2	0.00248756	8	0.96418	0.97960	0.020673
3	0.00199005	15	0.94428	0.97711	0.020651
4	0.00186567	16	0.94229	0.98259	0.020700
5	0.00167910	22	0.92985	0.98209	0.020696
6	0.00116086	30	0.91642	0.98955	0.020763
7	0.00049751	54	0.87463	0.99154	0.020781
8	0.00037313	58	0.87264	0.99701	0.020830
9	0.00029851	74	0.86667	0.99851	0.020844
10	0.00016584	79	0.86517	1.00299	0.020884
11	0.00012438	88	0.86368	1.00199	0.020875
12	0.00000000	92	0.86318	0.99900	0.020848

Identifying the CP error: Compare the xerror at every increase of nsplit and it can be observed that it increase at every increase in the nsplit. However after a certain level then xerror increases meaning that the nsplit from here will not provide optimization. The corresponding CP value of **0.00167910** will be considered for pruning. Below code is used to arrive at the pruning

```
pruning_Tree<- prune(CART_Train_Model, cp=0.0017, "CP" )
```

The visualization can be done as per the image depicted above ***(in pruning section)***

The model is now built and will have to undergo model measuring evaluation. Following are the methods used to do the model performance evaluation

1. Rank Order evaluation
2. KS
3. Gini Coefficient
4. Classification error (with function)

1. Rank Ordering (Decile) In using this method, the data needs to be assigned with a predicted class and probability. After the probability is assigned then the decile has to be defined so that a certain probability or class value range falls within the specified decile. Later the dataset is ranked to arrive at top order deciles. If the deciling.

Deciling step

```
CARTtrain.final$predict.class = predict(pruning_Tree, CARTtrain.final, type = "class")
CARTtrain.final$predict.score = predict(pruning_Tree, CARTtrain.final, type = "prob")
```

```
#Deciling

library(scales)
library(data.table)
decile=function(x) {
  deciles=vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10]=quantile(x,i, na.rm=T)
  }
  return(
  ifelse(x<deciles[1],1,
  ifelse(x<deciles[2],2,
  ifelse(x<deciles[3],3,
  ifelse(x<deciles[4],4,
  ifelse(x<deciles[5],5,
  ifelse(x<deciles[6],6,
  ifelse(x<deciles[7],7,
  ifelse(x<deciles[8],8,
  ifelse(x<deciles[9],9,10
  ))))))))))
}

CARTtrain.final$deciles <- decile(CARTtrain.final$predict.score[,2])
```

Ranking the assigned deciles

```
#Model Measurement using the Rank Ordering method

library(formattable)
tmp_DT_CART = data.table(CARTtrain.final)
rank<-tmp_DT_CART[,list(
  cnt=length(TARGET),
  cnt_resp=sum(TARGET),
  cnt_non_resp=sum(TARGET==0)),
  by=deciles][order(-deciles)]
rank$rrate = round(rank$cnt_resp/rank$cnt,4);
rank$cum_resp = cumsum(rank$cnt_resp)
rank$cum_non_resp = cumsum(rank$cnt_non_resp)
rank$cum_rel_resp = round(rank$cum_resp/sum(rank$cnt_resp),4);
rank$cum_non_rel_resp = round(rank$cum_non_resp/sum(rank$cnt_non_resp),4);
rank$ks = abs(rank$cum_rel_resp - rank$cum_non_rel_resp)*100;
rank$rrate=percent(rank$rrate)
rank$cum_rel_resp = percent(rank$cum_rel_resp)
rank$cum_rel_non_resp = percent(rank$cum_non_rel_resp)
View(rank)
```

The rank table

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_non_rel_resp	ks	cum_rel_non_resp
1	10	6117	1171	4946	19.14%	1171	4946	58.26%	0.3535	22.91	35.35%
2	7	555	81	474	14.59%	1252	5420	62.29%	0.3874	23.55	38.74%
3	6	2627	273	2354	10.39%	1525	7774	75.87%	0.5557	20.30	55.57%
4	5	6701	485	6216	7.24%	2010	13990	100.00%	1.0000	0.00	100.00%

Interpretation of the rank table

1. Deciling and ranking provides the decision makers to identify the probable area of focus where the required efforts need to be channelized. *Example; if a budget has to be spent across geographical areas like, AsiaPAC, EMEA, Europe, Americas etc. deciling can help decision makers focus where can the capital be infused more so that the returns can be maximized*
2. A well spread deciles ranking talk about an efficient model designed and the technique has worked successful
3. The decile with the highest KS value which is 23.55 is the most efficient decile

2. KS/ auc/ classification error coding

```
#KS calculation
library(ROCR)
pred<-prediction(CARTtrain.final$predict.score[,2], CARTtrain.final$TARGET)
perf<-performance(pred, "tpr", "fpr")
plot(perf)
KS<-max(attr(perf, 'y.values')[[1]] - attr(perf, 'x.values')[[1]])


#Area Under Curve

auc<- performance(pred, "auc");
auc<- as.numeric(auc@y.values)


## Gini Cooefficient

library(ineq)
gini=ineq(CARTtrain.final$predict.score[,2], type = "Gini")



## classification Error
with(CARTtrain.final, table(TARGET, predict.class))
(129+1740)/16000
KS
auc
gini
```

```
      predict.class
TARGET     0     1
     0 13861   129
     1  1740   270
> (129+1740)/16000
[1] 0.1168125
> KS
[1] 0.2370595
> auc
[1] 0.6561471
> gini
[1] 0.2730623
```

A. KS value for a model to be good can be around-0.4-0.5. if the KS value is greater than 0.6 or 0.7 the model is assumed to be overfit model
B. Classification error is at ~ 11% error rate meaning 89% of the time the model does a correct prediction
C. For Area Under the curve the value of 0.656 is fairly good for the model

The above model is run on the train data and a similar model has been run on the test data with the below values

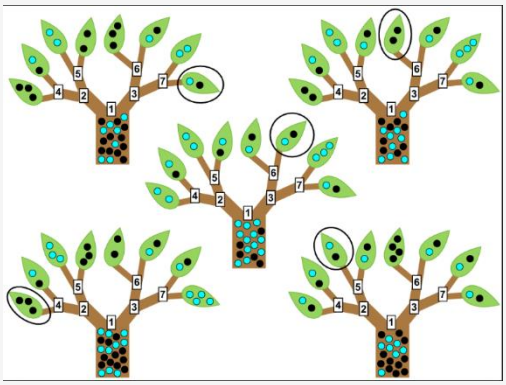
1. Rank Order

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_non_rel_resp	ks	cum_rel_non_resp
1	10	1603	297	1306	18.53%	297	1306	59.16%	0.3734	21.82	37.34%
2	6	2397	205	2192	8.55%	502	3498	100.00%	1.0000	0.00	100.00%

2. KS, auc and gini

```
      predict.class
TARGET     0     1
     0 13988     2
     1   187  1823
> KS
[1] 0.9979328
> auc
[1] 0.9999706
> gini
[1] 0.7619822
```

5. RANDOM FOREST TECHNIQUE:



It is an ensemble technique which means it will run multiple trees with random sampling and without replacement. It is a black box technique meaning that it cannot be visualized. It functions in the background and a final confusion matrix is arrived at. Since there are numerous trees developed hence the name.

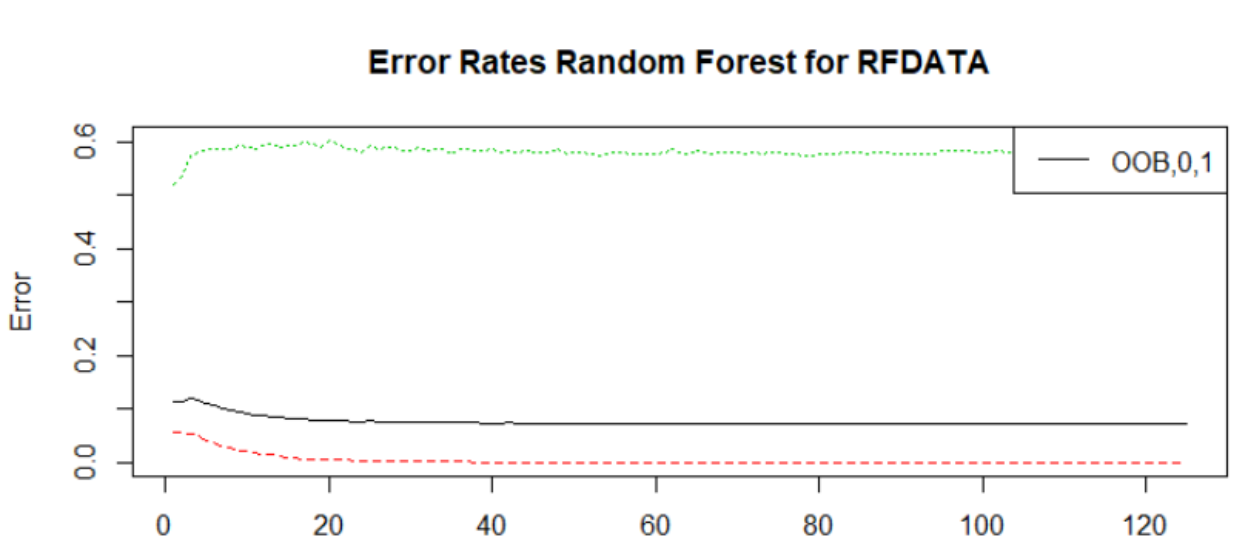
Performing the Random Forest Technique on the Mybank dataset done below

Setting the seed and splitting the data is same as mentioned in the CART technique

Later invoke the Random Forest library and run the below code

```
#Random Forest

library(randomForest)
setwd("C:/Users/Amit Kulkarni/Documents/R Programming/Machine Learning/data mining/Project 4")
RF<- randomForest(as.factor(TARGET)~., data=RFtrain.final,
                  ntree=125, mtry = 3, nodesize = 10, importance= TRUE)
print(RF)
plot(RF, main="")
legend("topright", c("OOB,0,1"), text.col = 1:6, lty = 1:3, col=1:3)
title(main="Error Rates Random Forest for RFDATA")
```

The output is arrived at as shown below



The above graph depicts that the error rate is reduced at a certain increase of trees but later stabilize even if the tree numbers are increased. Thus an optimum tree numbers can be identified. To do the same the model can be tuned using the TUNERF function.

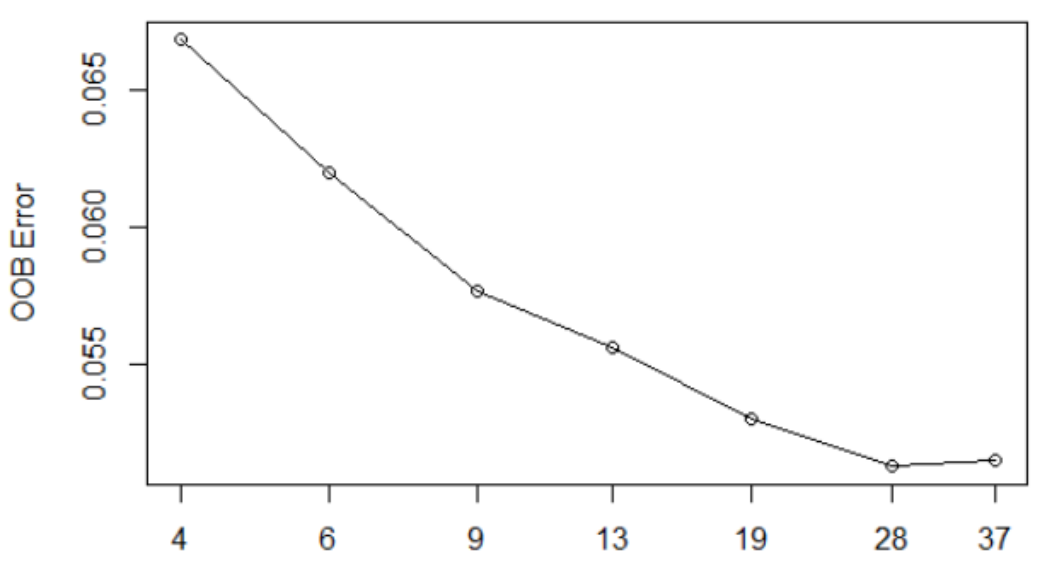
Important variables also can be identified using the err.rate function as shown below.

	0	1	MeanDecreaseAccuracy	MeanDecreaseGini
LEN_OF_RLTN_IN_MNTH	10.46	16.75	15.95	118.40
AGE_BKT	14.65	10.67	14.90	107.28
AMT_L_DR	13.89	6.41	13.27	125.33
BALANCE	10.63	13.85	12.89	142.64
AVG_AMT_PER_CSH_WDL_TXN	9.76	6.60	11.44	107.26
OCCUPATION	9.81	10.38	10.69	83.06
AMT_BR_CSH_WDL_DR	7.45	12.52	9.76	108.63
TOT_NO_OF_L_TXNS	8.19	6.75	9.62	116.36
AGE	7.89	9.74	9.28	83.45
SCR	8.82	8.24	8.89	145.21
NO_OF_L_CR_TXNS	6.44	8.09	8.87	106.35
HOLDING_PERIOD	8.13	8.06	8.86	120.55
FLG_HAS_CC	7.35	7.72	8.12	35.57
AMT_NET_DR	7.00	7.28	8.06	74.49
AVG_AMT_PER_NET_TXN	6.79	5.73	8.03	74.37
AMT_ATM_DR	6.57	6.01	7.85	93.40
AMT_CHQ_DR	6.11	4.66	7.49	91.04
GENDER	6.88	6.69	7.35	22.42
FLG_HAS_OLD_LOAN	6.28	5.95	7.23	16.52
NO_OF_BR_CSH_WDL_DR_TXNS	6.68	6.98	7.04	52.95
FLG_HAS_NOMINEE	5.12	6.16	7.04	14.24
AVG_AMT_PER_MOB_TXN	5.75	5.59	6.87	41.96
NO_OF_L_DR_TXNS	5.93	6.98	6.47	72.42
ACC_TYPE	4.42	3.25	5.91	18.07
AVG_AMT_PER_CHQ_TXN	4.86	4.50	5.30	89.96
NO_OF_CHQ_DR_TXNS	4.62	5.54	5.13	41.01
FLG_HAS_ANY_CHGS	3.43	6.02	5.01	11.73
NO_OF_NET_DR_TXNS	4.30	4.14	4.85	27.26
AMT_MOB_DR	3.90	4.72	4.36	42.20
NO_OF_ATM_DR_TXNS	3.83	0.38	3.99	32.70
AVG_AMT_PER_ATM_TXN	3.19	4.45	3.82	97.03
NO_OF_OW_CHQ_BNC_TXNS	2.71	3.77	3.63	10.95
NO_OF_IW_CHQ_BNC_TXNS	2.40	2.92	3.30	7.61
AMT_MIN_BAL_NMC_CHGS	0.58	2.52	2.54	2.85
NO_OF_MOB_DR_TXNS	1.98	2.59	2.16	10.68
AMT_OTH_BK_ATM_USG_CHGS	1.26	0.62	1.26	1.41
random	0.06	-0.99	-0.64	64.10

Where the mean decrease gini is highest is the most important variable.

```
tRF<- tuneRF(x=RFtrain.final[, -c(1)],
             y=as.factor(RFtrain.final$TARGET),
             mtrystart=3,
             ntreeTry = 101,
             stepFactor = 1.5,
             improve=0.001,
             trace= TRUE,
             plot=TRUE,
             doBest = TRUE,
             nodesize=10,
             importance=TRUE,
)
```

```
mtry = 6  OOB error = 6.2%
Searching left ...
mtry = 4        OOB error = 6.68%
-0.07762097 0.001
Searching right ...
mtry = 9        OOB error = 5.77%
0.06955645 0.001
mtry = 13       OOB error = 5.56%
0.03575298 0.001
mtry = 19       OOB error = 5.31%
0.04606742 0.001
mtry = 28       OOB error = 5.13%
0.03297998 0.001
mtry = 37       OOB error = 5.16%
-0.004872107 0.001
>
```



The above graph shows and the tunerf function shows that the error rate at 28 trees is least and after that it starts to increase thus the Random Forest model can be run for 28 ntrees again

This model again can be tested using the Rank Order, KS, gini coefficient and classification error

The technique is the same as explained in CART above

The results are given below for the train data.

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_non_rel_resp	ks
1	10	1600	1600	0	100.00%	1600	0	79.60%	0.00%	79.60
2	9	1616	410	1206	25.37%	2010	1206	100.00%	8.62%	91.38
3	8	1602	0	1602	0.00%	2010	2808	100.00%	20.07%	79.93
4	7	1761	0	1761	0.00%	2010	4569	100.00%	32.66%	67.34
5	6	1673	0	1673	0.00%	2010	6242	100.00%	44.62%	55.38
6	5	1452	0	1452	0.00%	2010	7694	100.00%	55.00%	45.00
7	4	2058	0	2058	0.00%	2010	9752	100.00%	69.71%	30.29
8	3	1303	0	1303	0.00%	2010	11055	100.00%	79.02%	20.98
9	2	1771	0	1771	0.00%	2010	12826	100.00%	91.68%	8.32
10	1	1164	0	1164	0.00%	2010	13990	100.00%	100.00%	0.00

KS, auc, gini and classification error

```
      predict.class
TARGET     0     1
     0 13902    88
     1  1558   452
>
> KS
[1] 0.4768744
> auc
[1] 0.8192632
> gini
[1] 0.5129349
```

In all the 3 model measurements the train data performs well given the values above.

Similarly the test data gives the below results

Rank Order

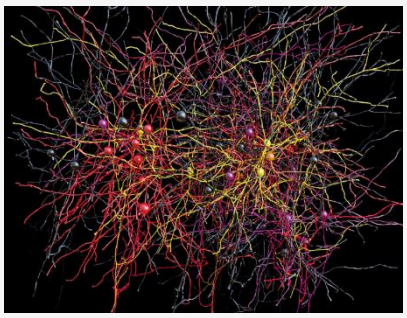
	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_non_rel_resp	ks
1	10	400	400	0	100.00%	400	0	79.68%	0.00%	79.68
2	9	408	102	306	25.00%	502	306	100.00%	8.75%	91.25
3	8	397	0	397	0.00%	502	703	100.00%	20.10%	79.90
4	7	397	0	397	0.00%	502	1100	100.00%	31.45%	68.55
5	6	431	0	431	0.00%	502	1531	100.00%	43.77%	56.23
6	5	410	0	410	0.00%	502	1941	100.00%	55.49%	44.51
7	4	423	0	423	0.00%	502	2364	100.00%	67.58%	32.42
8	3	435	0	435	0.00%	502	2799	100.00%	80.02%	19.98
9	2	404	0	404	0.00%	502	3203	100.00%	91.57%	8.43
10	1	295	0	295	0.00%	502	3498	100.00%	100.00%	0.00

KS, auc, gini and classification error

```
      predict.class
TARGET    0    1
     0 3498    0
     1  126  376
>
> KS
[1] 0.9951492
> auc
[1] 0.9999496
> gini
[1] 0.6585258
```

The model is an overfit in the test data.

6. ARTIFICIAL NEURAL NETWORK TECHNIQUE:



Artificial neural network operates like the biological neural network and are connected through artificial neurons known as synaptic. Neural Network is also a supervised learning technique where a class is provided to us and then a predicted class is identified.

The given data set will have a predicted class using the neural network as shown below

Splitting the data is done as given below

```
NNdata=read.csv("PL_XSELL.csv", header=TRUE)
s= sample(c(1:20000), size = 16000)
data.train= NNdata[s,]
data.train.final= subset(data.train, select=-c(1,11))
data.test=NNdata[-s,]
data.test.final= subset(data.test, select=-c(1,11))
str(data.train.final)
```

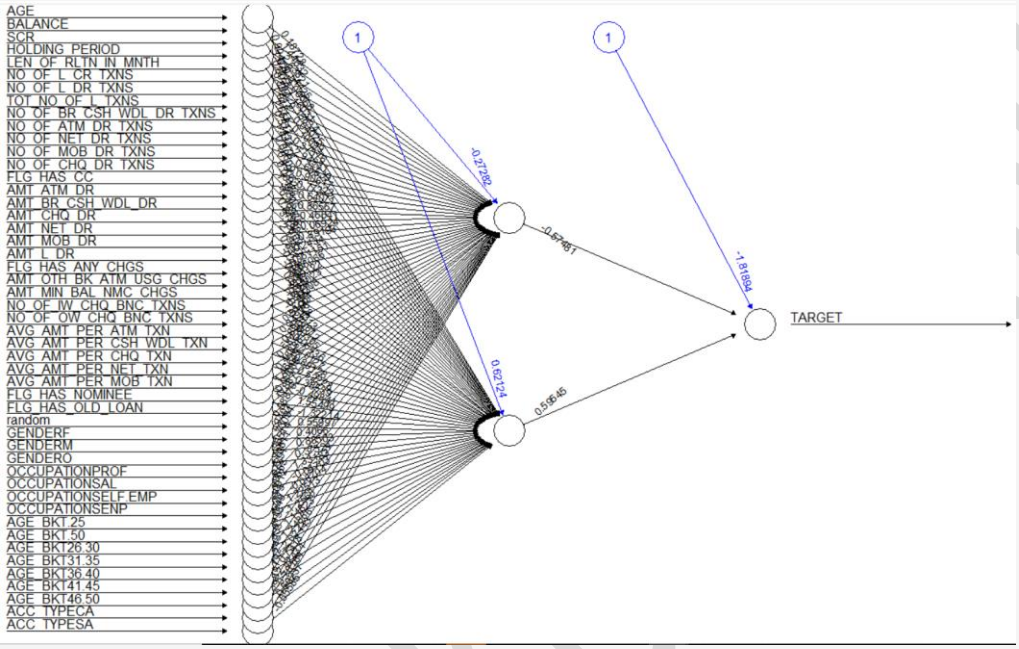
```
'data.frame':   16000 obs. of  38 variables:
 $ TARGET                : int  0 0 0 0 0 0 0 0 0 0 ...
 $ AGE                   : int  25 55 51 40 43 50 29 30 39 43 ...
 $ GENDER                : Factor w/ 3 levels "F","M","O": 2 1 2 1 2 2 2 2
2 1 ...
 $ BALANCE               : num  1514478 133879 2264 231238 77964 ...
 $ OCCUPATION            : Factor w/ 4 levels "PROF","SAL","SELF-EMP",..: 2
 4 4 1 1 1 4 2 1 4 ...
 $ AGE_BKT               : Factor w/ 7 levels "<25",">50","26-30",..: 1 2 2
 5 6 7 3 3 5 6 ...
 $ SCR                   : int  197 292 211 196 208 660 475 640 790 391 ...
 $ HOLDING_PERIOD        : int  18 18 19 22 20 9 6 30 13 15 ...
 $ ACC_TYPE              : Factor w/ 2 levels "CA","SA": 2 2 2 2 2 2 2 2 2
1 ...
```

There are factors that the neural networks doesn't work and all the variables need to be integers or numeric. Thus we need to convert these using the below set of codes

```
gen.matrix<- model.matrix(~GENDER - 1, data= data.train.final)
data.train.final<- data.frame(data.train.final, gen.matrix)
View(data.train.final)
occ.matrix<- model.matrix(~OCCUPATION - 1, data= data.train.final)
data.train.final<- data.frame(data.train.final, occ.matrix)
age.matrix<- model.matrix(~AGE_BKT - 1, data= data.train.final)
data.train.final<- data.frame(data.train.final, age.matrix)
acc.matrix<- model.matrix(~ACC_TYPE - 1, data= data.train.final)
data.train.final<- data.frame(data.train.final, acc.matrix)

data.train.final.nn<- subset(data.train.final, select=-c(3, 5, 6, 9))
View(data.train.final.nn)
```

Once the categorization is done the data can be run through the neural network model and the below visualization is built

```
nn1= neuralnet(formula = TARGET~.,
               data= data.train.final.nn,
               hidden = 2,
               err.fct= "sse",
               linear.output = FALSE,
               lifesign = "full",
               lifesign.step = 10,
               threshold = 0.01,
               stepmax= 2000)

plot(nn1)
```



The model will be tested using the rank order

```
decile=function(x) {
  deciles=vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10]=quantile(x,i, na.rm=T)
  }
  return(
    ifelse(x<deciles[1],1,
           ifelse(x<deciles[2],2,
                  ifelse(x<deciles[3],3,
                         ifelse(x<deciles[4],4,
                                ifelse(x<deciles[5],5,
                                       ifelse(x<deciles[6],6,
                                              ifelse(x<deciles[7],7,
                                                     ifelse(x<deciles[8],8,
                                                            ifelse(x<deciles[9],9,10
                                                            ))))))))))
}

data.train.final.nn$deciles<- decile(data.train.final.nn$prob)
```

The rank command is given below with viewing the created decile table

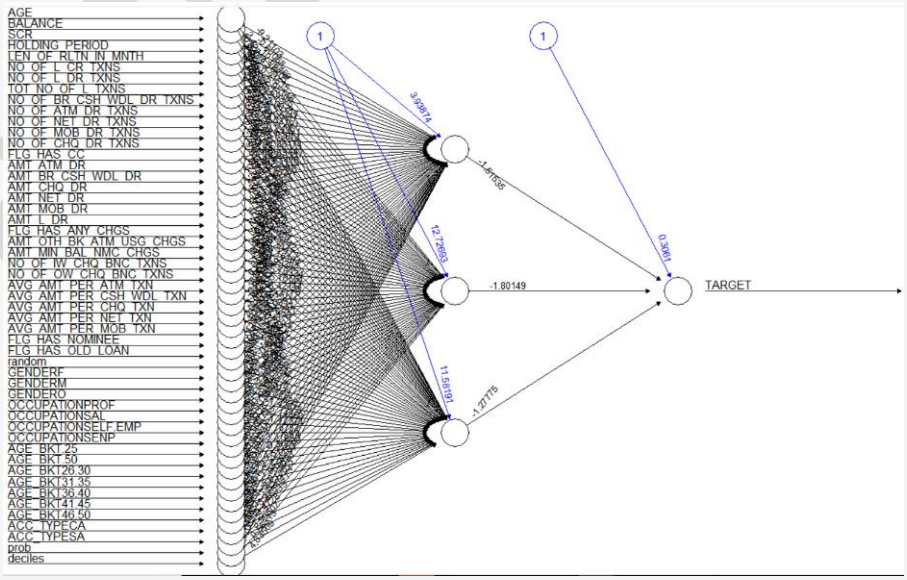
```
tmp_NNDT = data.table(data.train.final.nn)
rank<-tmp_NNDT[,list(
  cnt=length(TARGET),
  cnt_resp=sum(TARGET),
  cnt_non_resp=sum(TARGET==0)),
  by=deciles][order(deciles)]
rank$rrate = round(rank$cnt_resp/rank$cnt,4);
rank$cum_resp = cumsum(rank$cnt_resp)
rank$cum_non_resp = cumsum(rank$cnt_non_resp)
rank$cum_rel_resp = round(rank$cum_resp/sum(rank$cnt_resp),4);
rank$cum_non_rel_resp = round(rank$cum_non_resp/sum(rank$cnt_non_resp),4);
rank$ks = abs(rank$cum_rel_resp - rank$cum_non_rel_resp)*100;
rank$rrate=percent(rank$rrate)
rank$cum_rel_resp = percent(rank$cum_rel_resp)
rank$cum_rel_non_resp = percent(rank$cum_non_rel_resp)
View(rank)
```

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_non_rel_resp	ks	cum_rel_non_resp
1	3	4736	398	4338	8.40%	398	4338	19.67%	0.3104	11.37	31.04%
2	10	11264	1625	9639	14.43%	2023	13977	100.00%	1.0000	0.00	100.00%

In the rank table it can be seen that the deciling has not been efficient and the purpose of neural network seems to be not accomplished. This could be due to non-scaling of the data. Scaling of data is required due to the fact that there are variables that do not have the same measuring units and the calculations between them can be incorrect. Thus the data needs to be scaled. It can be done as given below.

```
x= subset(data.train.final.nn, select =-c(1))
data.train.nn.scaled <- cbind(data.train.final.nn[1], x)
View(x)
test.final= scale(x)
test.final<- cbind(data.train.final.nn[1], test.final)
View(test.final)
```

After the data is scaled re-run the ANN model again and a new model is generated



Finally when the deciling is run we get the below RANK ORDER table

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_non_rel_resp	ks	cum_rel_non_resp
1	10	1600	665	935	41.56%	665	935	32.87%	0.0669	26.18	6.69%
2	9	1600	391	1209	24.44%	1056	2144	52.20%	0.1534	36.86	15.34%
3	8	1600	260	1340	16.25%	1316	3484	65.05%	0.2493	40.12	24.93%
4	7	1600	112	1488	7.00%	1428	4972	70.59%	0.3557	35.02	35.57%
5	6	1600	119	1481	7.44%	1547	6453	76.47%	0.4617	30.30	46.17%
6	5	1600	123	1477	7.69%	1670	7930	82.55%	0.5674	25.81	56.74%
7	4	1600	109	1491	6.81%	1779	9421	87.94%	0.6740	20.54	67.40%
8	3	1600	86	1514	5.38%	1865	10935	92.19%	0.7824	13.95	78.24%
9	2	1600	85	1515	5.31%	1950	12450	96.39%	0.8907	7.32	89.07%
10	1	1600	73	1527	4.56%	2023	13977	100.00%	1.0000	0.00	100.00%

This table is efficient and the deciling is in the ascending order with KS value being highest at 40.12 at 8th Decile.

Classification error

```
View(test.final)
test.final$class = ifelse(test.final$prob>0.2, 1, 0)
(with(test.final, table(TARGET, as.factor(class))))
```

```
TARGET     0     1
     0 11664  2313
     1   939  1084
```

KS, GINI and AUC

```
> KS
[1] 0.9951492
> auc
[1] 0.9999496
> gini
[1] 0.5349698
```

The model is overfit due to high KS and AUC values. With the gini co-efficient the model is a goodness fit

For the test data below are the model measures as given below

Same approach as used in the train data is done.

RANK ORDER

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_non_rel_resp	ks	cum_rel_non_resp
1	10	400	214	186	53.50%	214	186	44.40%	0.0529	39.11	5.29%
2	9	400	48	352	12.00%	262	538	54.36%	0.1529	39.07	15.29%
3	8	400	55	345	13.75%	317	883	65.77%	0.2510	40.67	25.10%
4	7	400	41	359	10.25%	358	1242	74.27%	0.3530	38.97	35.30%
5	6	400	25	375	6.25%	383	1617	79.46%	0.4596	33.50	45.96%
6	5	400	23	377	5.75%	406	1994	84.23%	0.5668	27.55	56.68%
7	4	400	18	382	4.50%	424	2376	87.97%	0.6754	20.43	67.54%
8	3	400	14	386	3.50%	438	2762	90.87%	0.7851	12.36	78.51%
9	2	400	26	374	6.50%	464	3136	96.27%	0.8914	7.13	89.14%
10	1	400	18	382	4.50%	482	3518	100.00%	1.0000	0.00	100.00%

It can be seen that the KS is 40.67 at decile 8. A much better table than the one before scaling

```
      class
TARGET    0    1
     0 3287  231
     1  267  215
>
> KS
[1] 0.9951492
> auc
[1] 0.9999496
> gini
[1] 0.6438691
```

The model is an overfit model as per all model measurement techniques.

Train data and test data are almost similar making the model a good fit but the model is overfit

A quick comparison between the various values under various techniques with train and test are given below.

Model measurement	CART Train	CART test	RF Train	RF Test	ANN Train	ANN Test
KS	0.24	0.99	0.48	0.99	0.99	0.99
AUC	0.66	0.99	0.82	0.99	0.99	0.99
Gini	0.27	0.76	0.51	0.66	0.53	0.64

The dataset model is a good fit under CART Train and RF Train. In other all measures including the test the model values are identical

Concluding remarks.

The various machine learning models used are extremely useful in performing any prediction analysis with regards to categorical or continuous variables. They provide us the required guidance in making the decisions that otherwise would seem judgmental and could risk the organization's finances and reputations.

Individually the models are explained above.

The given dataset has a lot of collinearity where the dimensions can be reduced and the model built can be made simpler with lesser variables. Due to scaling and converting the factors to numeric there is increase in the variables again. There are not thresholds provided to see if the model passes and this can be decided but the business given the criticality of the output expected. Here the illustration of 3 machine learning techniques are done.