CRM ANALYTICS: CUSTOMER CHURN, APPETENCY AND UPSELLING PREDICTION FOR A MOBILE NETWORK OPERATOR

ISEN_613: Engineering Data Analysis

Team: Homo Bayesians

Jay Shah -20%

Samarth Zala -20%

Sandipan Paul -20%

Ritvik Dhupkar -20

Harish Chelappa -20%

Instructor: Dr. Satish Bukkapatnam

Industrial & Systems Engineering department,

Texas A&M University, College Station.

INDEX

- Appendix to be provided through other submission link.

EXECUTIVE SUMMARY

Importance

With the evolution CRM adoption has become essential for every service/goods providing firms. In order to yield maximum profits, to serve the needs of their customer in a better way, and to provide the customers every time with the compelling deals data analytics complements CRM in a unique way. The efficient market hypothesis suggests that practically it is impossible to predict the customer's behaviours in B2B or B2C firms. But also, there are number of firms who are doing this very efficiently through the myriad information of past customers' data and through the sophisticated use of data analytics.

Since people cannot predict what will happen next, forecasting what will occur mostly has been one challenging and concerned issues in many areas especially in CRM prediction. But, whenever the predictions with less bias is produced it will enable the firms to make good amount of profits. As the emergence of new advanced predicting techniques such as artificial neural networks (ANN), time series, fuzzy time series models and etc. that has enabled the firms to make excellent predictions with minimal supervision and assumptions. The main constraints in conventional techniques are number of assumptions are required to be made by analyst, dependent and independent variables and a threshold value to make predictions isn't easy to be found. For the reasons above, and in order to overcome above constraints in this report we have analysed the power of Artificial neural networks and Multilevel classification techniques and has benchmarked them to evaluate the performance of conventional classification techniques such as logistic regression, KNN, random forest, boosting, bagging, to name a few.

Objectives

As has been mentioned earlier, objective of this project is to try and predict one of the most influential terms for any firm to grow exponentially- customers' churn, appetency and upselling tendency. Basically in this three class classification problem, we have set our objectives to build a model which should be sensitive enough to the new upcoming data and also it should predict/mimic a result as close as possible. And to achieve this we have used both known and unknown techniques.

Literature review

Paper 1

Topic: Logistic Model Trees with AUC Split Criterion for the KDD Cup 2009 Small Challenge

Authors: Patrick Doetsch, Christian Buck, Pavlo Golik, Niklas Hoppe, Michael Kramp, Johannes Laudenberg, Christian Oberd¨orfer, Pascal Steingrube, Jens Forster.

In this work, they have described approach to the "Small Challenge" of the KDD cup 2009. They suggested a criterion based on the number of missing values to select a suitable imputation method for each feature. Logistic Model Trees (LMT) are extended with a split criterion optimizing the Area under the ROC Curve (AUC), which was the requested evaluation criterion. With LMT they achieved the best result for the "Small Challenge" without making use of additional data from other feature sets, resulting in an AUC score of 0.8081. Also AUC optimizing model combination has been presented that scored only slightly worse with an AUC score of 0.8074.

Paper 2

Topic: Design and Analysis of the KDD Cup 2009

Authors: Authors: Isabelle Guyon, Vincent Lemaire & Marc Boull, Gideon Dror , David Vogel

The primary idea of this paper was to impute the dataset properly. In most of the works, the missing values have been replaced with mean/mode which may lead to severe problems in linear classifier. In this work, the dataset has been imputed using principal components. Classification has been done primarily by using an ensemble of tree. Other methods like boosting, bagging and heterogeneous ensembles built by forward model selection have also been used.

Literature review and gap

After reviewing number of excellent reports and articles on this problem, the main innovation has been drawn to review all the techniques that we have learn as a part of our ISEN 613 curriculum also, some of the new techniques. With this we wish to evaluate the relative performance of various techniques which gives us the fair estimate about which method gives us the better results especially for this kind of problems.

Key implications with respect to objectives set

In this type project, the main objective is set on predicting True Positive Rate as high as possible on test data. Basically for this, every classification techniques will predict the probabilities for each class and we can set the prediction threshold to classify the potential customer to be inclined towards churn, appetency and upselling.

INTRODUCTION

Importance of the Customer relationship management

C-R-M stands for Customer Relationship Management. At its simple definition CRM allows firms to manage business relationships and data related to them. CRM is a key element of modern marketing strategies. Any business basically start with a foundation of a great patron relationship. And with the firm's growth this relationship management becomes more sophisticated. You need to share myriad of information across various teams within your organization. This is where the CRM can serve as a ***vital nerve center*** to manage the many connections that happen in a growing business. CRM gives everyone across the business, sales, marketing, a better way to manage the customer relationships and interactions that drive success.

The most practical way to build knowledge about customers is to predict the score as per number of variables and through these scores, customers' tendency to churn, appetency or upselling is determined. And accordingly the scores produced by system is used to build/personalise the relationship with customers. Here, the main concern is to have rapid and robust detection of the important predictors which are essential for defining above mentioned relationships. And key implementation is to have the complete automation of the whole process. A system exploits the set of large hundreds predictors and thousands of instances and comes up with the best subset of predictors and instances which will best define the customer tendency. The KDD cup 2009 challenge was to beat the in-house classifier developed by the Orange- mobile network service providing company. [1]

With mainly focussed towards and in a view to get the better insights into a CRM system, the problem mainly revolves around predicting the three prime terms, which are described as follows:

1. Churn rate
2. Appetency
3. Upselling

1. Churn rate:

It is defined as the ratio of the subscribers who discontinue their service with the provider to the total number of subscribers. In a nutshell, churn is the customer turnover. Churn basically occurs when the supplier of service or goods doesn't fulfil the customer's perceived expectations. When any firm's new customer enrolment rate is lesser than churn rate, then it can be said to make less sales and ultimately less profits in the future. And customer churn can be labeled as one of the most prime component of CRM. [2]

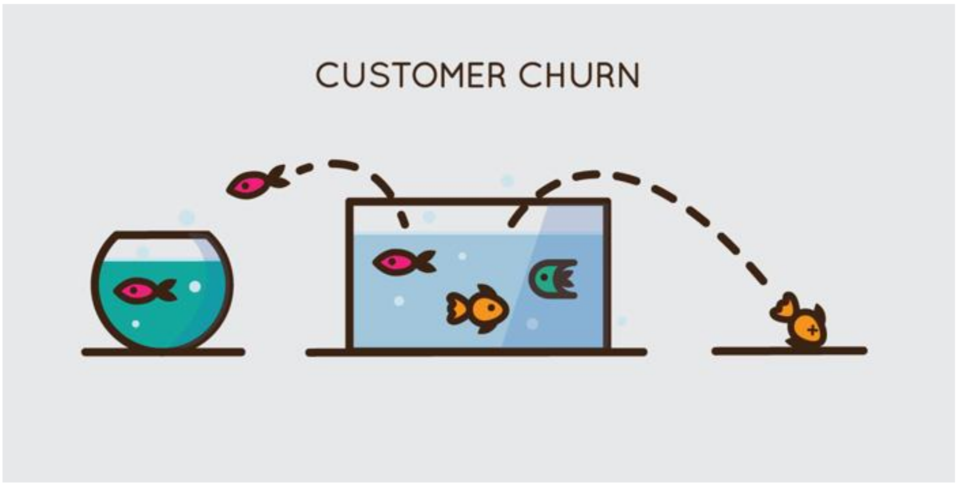


Figure 1. Customer Churn

2. Appetency: As the word explains, customer's inclination to buy a service or good from the provider is called an appetency.
3. Upselling: Selling the customer a more expensive version of the product is the key to profitability for seller and are an advantage to the customer. Fundamentally, it is a win-win situation. Customer's expectation is better met and your profit is increased. However, this is the only case when the policy is correct. And that is where the CRM comes into the play. One more advantage of a successful upselling is that it can improve customer relationship and cement customer relationship.

Figure 2 Upselling

Importance of this solution can be given as below:

- Needs of the customer is better identified by understanding their requirements.
- Increased sales/profit through better prediction of the needs based on the historic trends.
- Cross selling/Upselling of the products by highlighting and suggesting alternatives and enhancements.
- Potential customer identification.
- This can lead to better marketing of products.

The firms which are most influenced by this solutions are the B2C and B2B firms, by adopting the power of data analytics and CRM, the following advantages come along:

- Enhanced customer satisfaction
- Increased overall efficiencies of the organization.
- Improved profitability by emphasizing more on potential customers.

Facts& figures about potential of CRM & data analytics:

- A forecast from Gartner anticipates that the customer relationship management market will be worth $27.5 billion worldwide in 2015. Rising to $37 billion by just 2017.
- We can see that CRM software leads this technology investment, with 49% of businesses planning to increase spending.

- We live in a world where 80% of consumers will research your product online before making a purchase, 87% will use their mobile device to shop and will use several channels to interact with your brand.
- In order to engage potential customers across multiple devices in multiple channels, you need data. And with CRM software, you are able to create a 360 degree overview of each person you connect with.[3]

Specifics of the problem:

- The dataset used will be the KDD cup – 2009 customer relationship dataset.
- The small dataset from French telecom company Orange provided an opportunity to predict a specific customer's inclination to switch the provider (churn), buy new products and services (appetency) or buy upgrades proposed to them (upselling).
- The KDD Cup 2009 dataset uses marketing databases from the Telecom company Orange to predict the propensity of customers to switch provider (churn), buy new products or services (appetency), or buy upgrades or add-ons proposed to them to make the sale more profitable (up-selling)
- This is a classification problem with three classification labels:- churn, appetency and upselling
- The dataset has 15,000 training. It also has a large number of missing values. It has 190 numerical variables and 40 categorical.

Objectives

- It is an opportunity to prove that we can deal with a very large database, including heterogeneous noisy data (numerical and categorical variables), and unbalanced class distributions.
- The task is to estimate the churn, appetency and up-selling probability of customers, hence there are three target values to be predicted. A large number of variables (230) is made available for prediction.
- The main objective of the problem is to make good predictions of the target variables. The prediction of each target variable is thought of as a separate classification problem. The results of classification, obtained by thresholding the prediction score, may be represented in a confusion matrix, where TP (true positive), FN (false negative), TN

(true negative) and FP (false positive) represent the number of examples falling into each possible outcome.

- Also, the classification methods learnt in the class will be applied along with two novel techniques i.e.
    1. Artificial Neural Network (ANN)
    2. Multi label classification

<u>Scope of the work</u>:

In this project due to so many missing values major task is going to be the data cleaning and data imputations. Apart from that we are going to apply following methods:

1. Random Forest
2. Bagging
3. Boosting
4. Support vector machine
5. Logistic regression
6. TREE
7. K-Nearest Neighbours
8. Artificial Neural Networks (ANN)
9. Multi label classification

PROJECT APPROACH

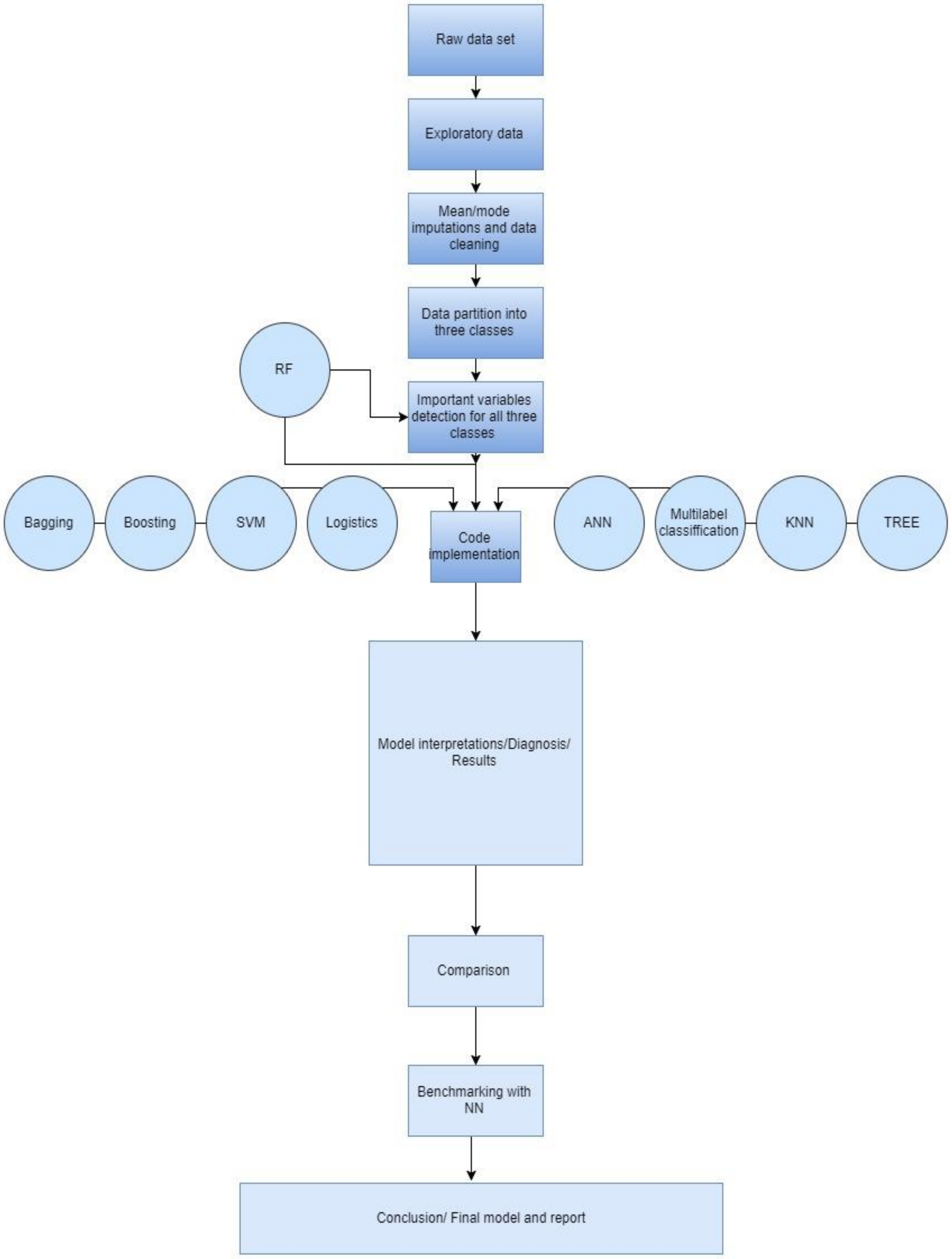

Raw data set
Exploratory data
Mean/mode imputations and data cleaning
Data partition into three classes
RF
Important variables detection for all three classes
Bagging
Boosting
SVM
Logistics
Code implementation
ANN
Multilabel classiffication
KNN
TREE
Model interpretations/Diagnosis/ Results
Comparison
Benchmarking with NN
Conclusion/ Final model and report

Figure 3

Description of the new techniques applied[4]

1. Artificial Neural Network

A feed forward neural network is also known as Deep Neural Network (DNN) or Multi-Layer Perceptron (MLP), is the most common type of neural network. There are various other type of neural network as well, such as Convolution Neural Network (CNN), and Recurrent Neural Network (RNN).

We can say that this function is underlying relationship between responses to predictors. What make it real brute force in terms of approximating underlying relationship is lies in its number of parameters? In neural network we will have large number of parameters that we can tweak to get some close approximation of some function. Training of neural network boils down to estimating these parameters. We train an artificial neural network by showing it millions of training examples and gradually adjusting the network parameters until it gives the classifications we want. [5]

The ANN attempts to recreate the computational mirror of the biological neural network. This system is based on depicting system of neurons to approximate the function. Here, in MLP the perceptron is type of neuron. It receives n input and gives output in binary way.

Perceptron

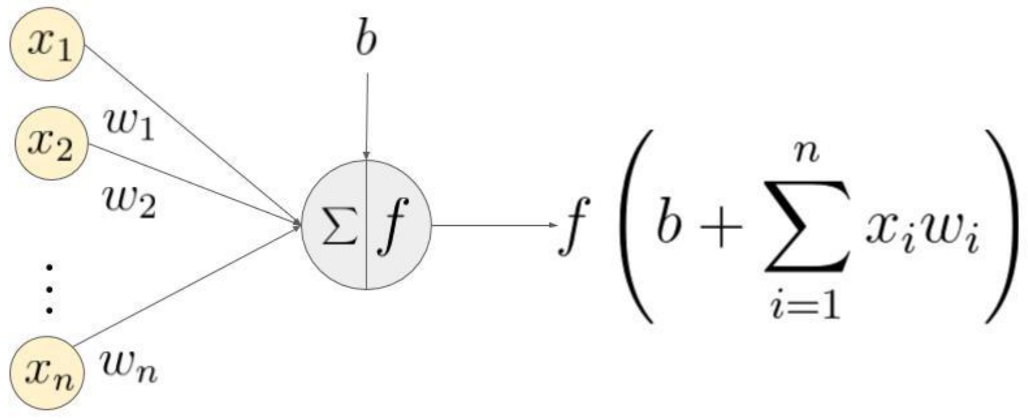
As seen below, it works in two steps –

It calculates the weighted sum of its inputs and then applies an activation function to normalize the sum. Bias can also be treated as another input. The bias allow to shift the line. The weights determine the slope

The activation functions can be linear or nonlinear. Also, there are weights associated with each input of a neuron. These (weights and bias) are the parameters which the network has to learn during the training phase

Neuron can be thought of as the basic processing unit of a Neural Network.

.



An example of a neuron showing the input (x_1 - x_n), their corresponding weights (w_1 - w_n), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Figure 4[5]

There is another parameter called activation function, this is also called squashing function.

It depict the output of neuron in biological sense. Neuron can be active or inactive.

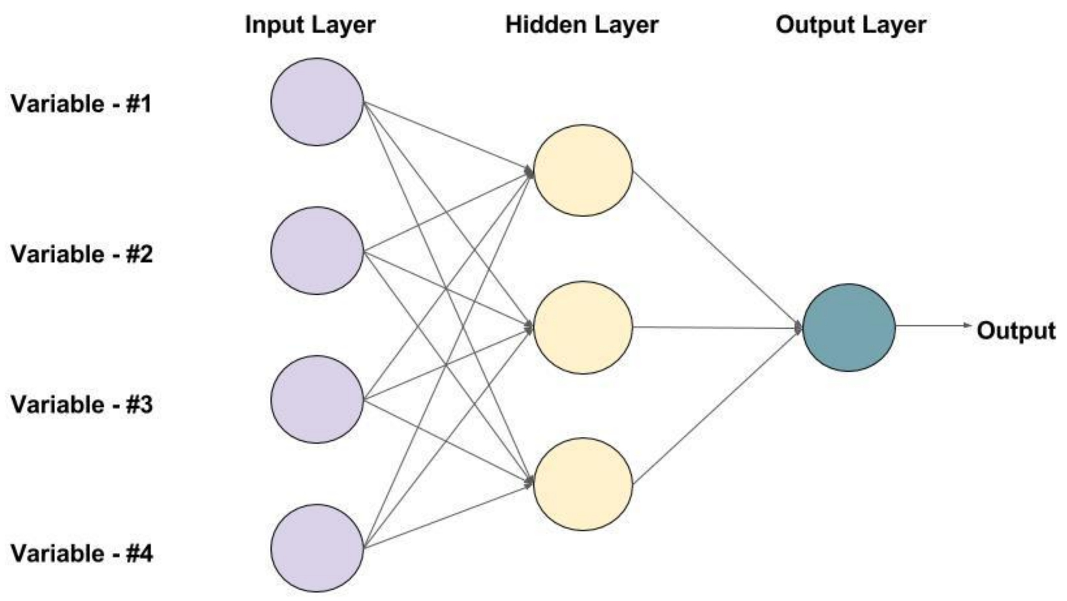
So depicting here in ANN same way, activate function gives output in between zero to one [0, 1].

There are many functions can be used as activate functions.

Here in ANN some activate functions are,

- Sigmoid
- Tanh
- Rectifier linear unit (ReLU)

This each activate function has its own merits and demerits.



An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)

Figure 5[6]

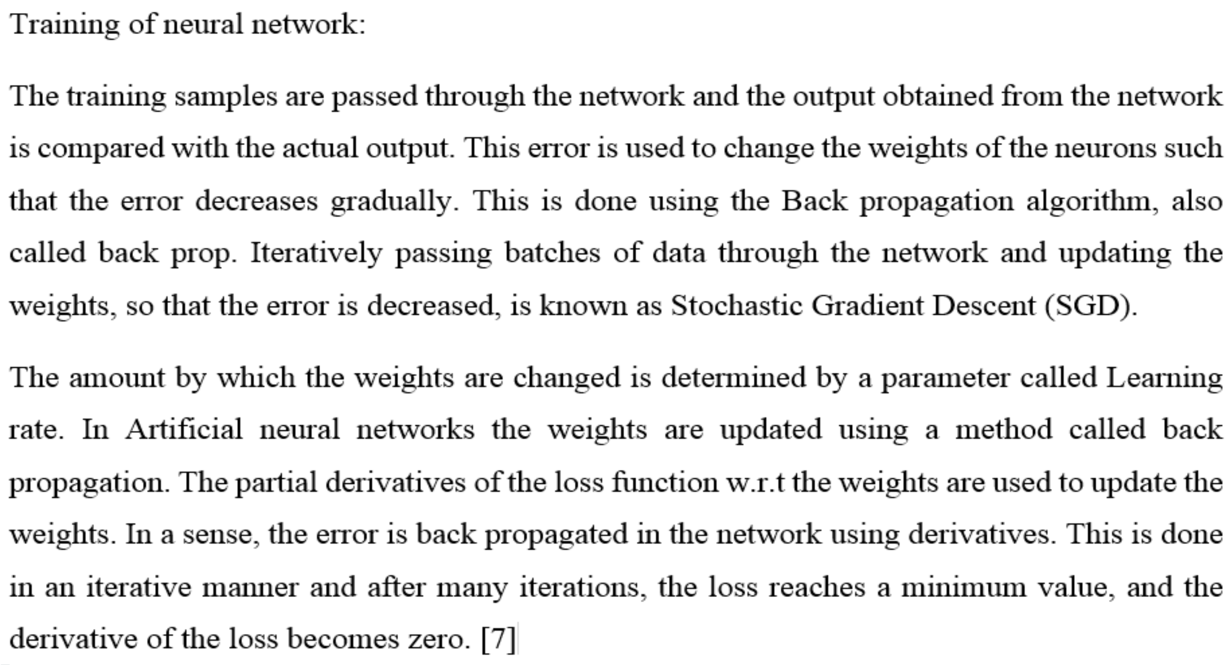
Given above is an example of a feed forward Neural Network. It is a directed acyclic Graph which means that there are no feedback connections or loops in the network. It has an input layer, an output layer, and a hidden layer. In each layer we have stacked some neurons. In general, there can be multiple hidden layers. [6]

The leftmost layer in this network is called the input layer, and the neurons within the layer are called input neurons. The rightmost or output layer contains the output neurons, or, as in this case, a single output neuron. The middle layer is called a hidden layer, since the neurons in this layer are neither inputs nor outputs. By having hidden layers we can basically approximate nonlinear relationship.

A feed forward network applies a series of functions to the input. By having multiple hidden layers, we can compute complex functions by cascading simpler functions.

A Neural Network with a single hidden layer with nonlinear activation functions is considered to be a Universal Function Approximator.

The choice of hidden units is a very active research area in Machine Learning. The type of hidden layer distinguishes the different types of Neural Networks like CNNs, RNNs etc. The number of hidden layers is termed as the depth of the neural network.



Training of neural network:

The training samples are passed through the network and the output obtained from the network is compared with the actual output. This error is used to change the weights of the neurons such that the error decreases gradually. This is done using the Back propagation algorithm, also called back prop. Iteratively passing batches of data through the network and updating the weights, so that the error is decreased, is known as Stochastic Gradient Descent (SGD).

The amount by which the weights are changed is determined by a parameter called Learning rate. In Artificial neural networks the weights are updated using a method called back propagation. The partial derivatives of the loss function w.r.t the weights are used to update the weights. In a sense, the error is back propagated in the network using derivatives. This is done in an iterative manner and after many iterations, the loss reaches a minimum value, and the derivative of the loss becomes zero. [7]

7

2. Multi -Label Classification Problem

Multi-level classification is a special type of classification technique where multiple labels are assigned to each instance and these instances are further assigned to multiple categories. This type of classification is used when a group of target levels are attempted at the same time.[8]

It is a generalization of multiclass classification, which is the single-label problem of categorizing instances into accurately one of more than two classes; in the multi-label problem, there is no restraint on how many of the classes the instance can be assigned to. This method comprises of determining a map between inputs and a binary vector output. The need for this kind of classification is the fact that most classification techniques use the correlation between labels. This may lead to inaccuracy in predicting at different labels. The model which performs well in one label may not be good at others. This explains the need for multi-label classification technique.[9]

The fundamental idea behind this technique is to transform the multi label problem into binary classification for each label. At each label, the previously predicted labels are used as features and they help in predicting other labels. There are many different types of multi label classification. The most basic approach is the **binary relevance** method. In this method, each label is treated as an individual single classification problem and a binary classifier is trained for each label. However, this method does not take into account the correlation of each class.

In **classifier chain**, the first classifier is trained just on the input data and then each next classifier is trained on the input space and all the previous classifiers in the chain. This is quite analogous to binary relevance method and it overcomes the only drawback in the aforementioned method i.e. it considers the correlation between each label.

In **label power set**, we transform the problem into a multi-class problem with one multi-class classifier is trained on all unique label combinations found in the training data. As models are chained in classifier chain, while here it counts each subset of response and makes model for every possible subset. The only demerit of this is that as the training data increases, number of classes become more. Thus, increasing the model intricacy, and would result in a lower accuracy.[10]

There are other options also. Apart from making new classifier, you can use existing classifier and use it as the multi-label classifier. This method is called **Adapted algorithm**. For example, we can use KNN classifier or Random Forest classifier and we can transform into binary multi label classifier or classifier chain or nested multi-label classifier.

IMPLEMENTATION DETAILS

1. TREE[11]

Decision trees can be used for both regression and classification problems. Here we focus on classification trees. Classification trees are a very different approach to classification than prototype methods such as k-nearest neighbours. The basic idea of these methods is to partition the space and identify some representative centroids.

They also differ from linear methods, e.g., linear discriminant analysis, quadratic discriminant analysis and logistic regression. These methods use hyper planes as classification boundaries.

Classification trees are a hierarchical way of partitioning the space. We start with the entire space and recursively divide it into smaller regions. At the end, every region is assigned with a class label.

Prune back the tree to avoid over fitting the data. Typically, you will want to select a tree size that minimizes the cross-validated error

Code

```
        library(tree)

#upselling

        tree. Upselling=tree(y_upselling~.,upselling_train)
        summary(tree.upselling)
        y=upselling_valid$y_upselling
        plot(tree.upselling)
        text(tree.upselling,pretty=0)
        tree.upselling
        tree.pred=predict(tree.upselling,upselling_valid,type="vector")
        ypred=rep('no',5000)
        ypred[tree.pred[,2] > 0.2] = "yes"
        table(ypred,y)
        roc_auc(tree.pred[,2],y)

        cv.ups=cv.tree(tree.upselling)
        plot(cv.ups$size,cv.ups$dev,type='b')

        prune.ups=prune.tree(tree.ups,best=20)
        plot(prune.ups)
        text(prune.ups,pretty=0)
        tree.pred=predict(prune.ups,upselling_valid,type="vector")
        ypred=rep('no',5000)
        ypred[tree.pred[,2] > 0.2] = "yes"
        table(ypred,y)

#appetency

        tree.appetency=tree(y_appetency~.,appetency_train)
        summary(tree.appetency)
        plot(tree.appetency)
        text(tree.appetency,pretty=0)
        tree.appetency
        y=appetency_valid$y_appetency
        tree.pred=predict(tree.appetency,appetency_valid,type="vector")
        ypred=rep('no',5000)
        ypred[tree.pred[,2] > 0.2] = "yes"
        table(ypred,y)
        roc_auc(tree.pred[,2],y)
        cv.app=cv.tree(tree.appetency)
        plot(cv.app$size,cv.app$dev,type='b')

        prune.app=prune.tree(tree.app,best=28)
        plot(prune.app)
        text(prune.app,pretty=0)

        tree.pred=predict(prune.app,appetency_valid,type="vector")
        ypred=rep('no',5000)
        ypred[tree.pred[,2] > 0.2] = "yes"
        table(ypred,y)
        roc_auc(tree.pred[,2],y)

#churn

        tree.churn=tree(y_churn~.,churn_train)
        summary(tree.churn)
        plot(tree.churn)
        text(tree.churn,pretty=0)
        tree.churn
        y=churn_valid$y_churn
        tree.pred=predict(tree.churn,churn_valid,type="vector")
        ypred=rep('no',5000)
        ypred[tree.pred[,2] > 0.2] = "yes"
        table(ypred,y)
        roc_auc(tree.pred[,2],y)

        cv.churn=cv.tree(tree.churn)
        plot(cv.churn$size,cv.churn$dev,type='b')

        prune.churn=prune.tree(tree.churn,best=19)
        plot(prune.churn)
        text(prune.churn,pretty=0)
```

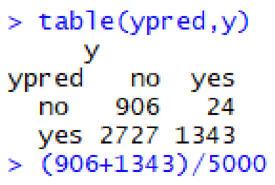
```
tree.pred=predict(prune.churn,churn_valid,type="vector")
ypred=rep('no',5000)
ypred[tree.pred[,2] > 0.2] = "yes"
table(ypred,y)
roc_auc(tree.pred[,2],y)
```

Tree is implemented through tree () function. As we know tree get over fitted easily. So, we have used cv.tree () function to cross validate the results. And from that we got the best tree size according to lowest misclassification error.

After getting the best model tree size, we pruned that tree up to that size. This best model's performance is described below and further diagnosed.

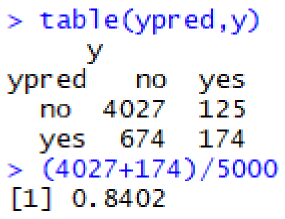
<u>Confusion Matrix</u>

Churn



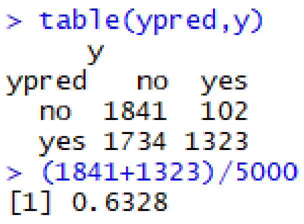
```
> table(ypred,y)
     y
ypred   no  yes
  no   906   24
  yes 2727 1343
> (906+1343)/5000
```

Appetency



```
> table(ypred,y)
     y
ypred   no  yes
  no  4027  125
  yes  674  174
> (4027+174)/5000
[1] 0.8402
```

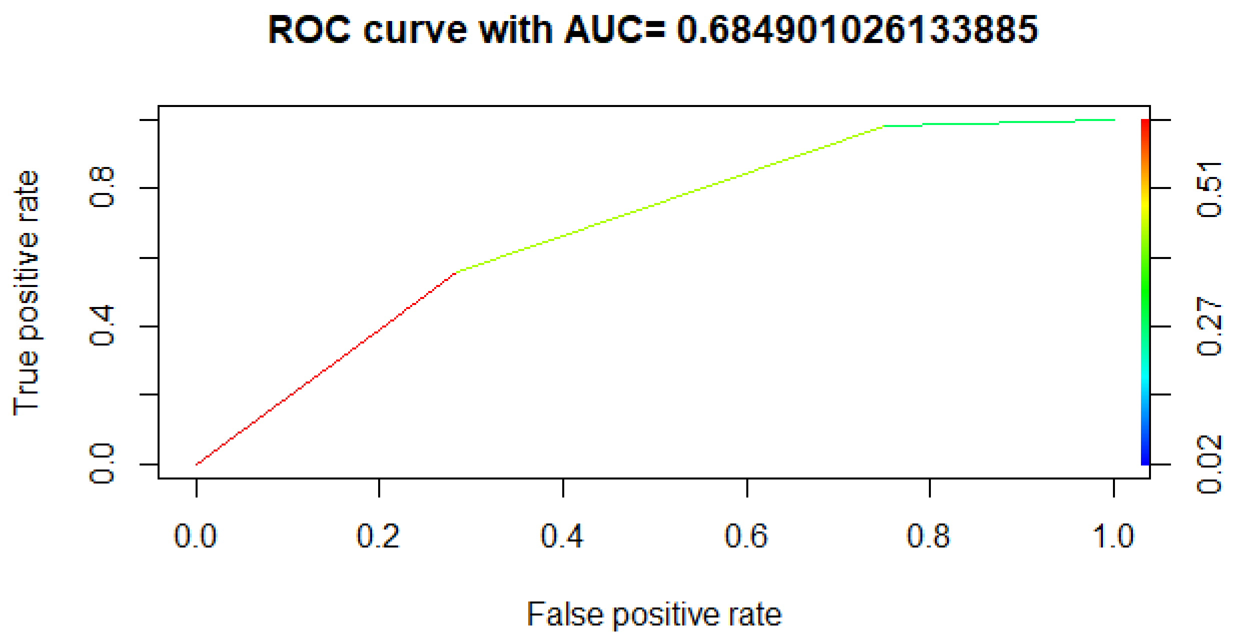
Upselling



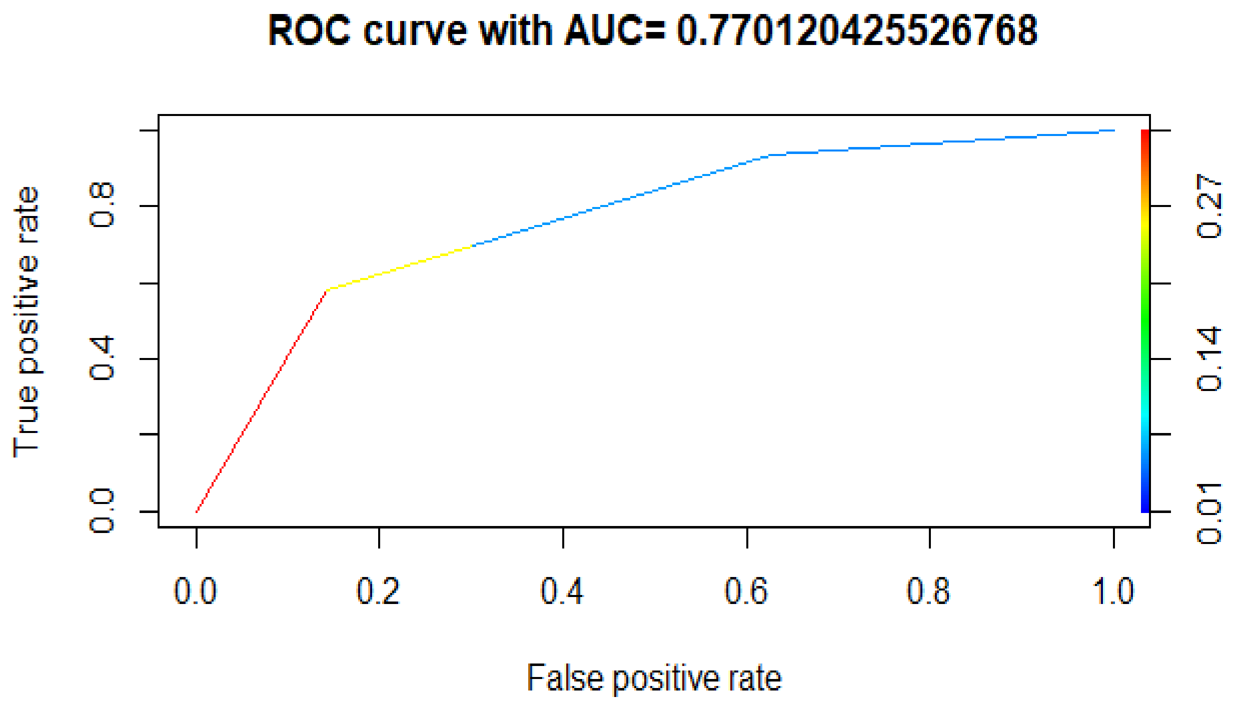
```
> table(ypred,y)
     y
ypred   no  yes
  no  1841  102
  yes 1734 1323
> (1841+1323)/5000
[1] 0.6328
```

ROC curves

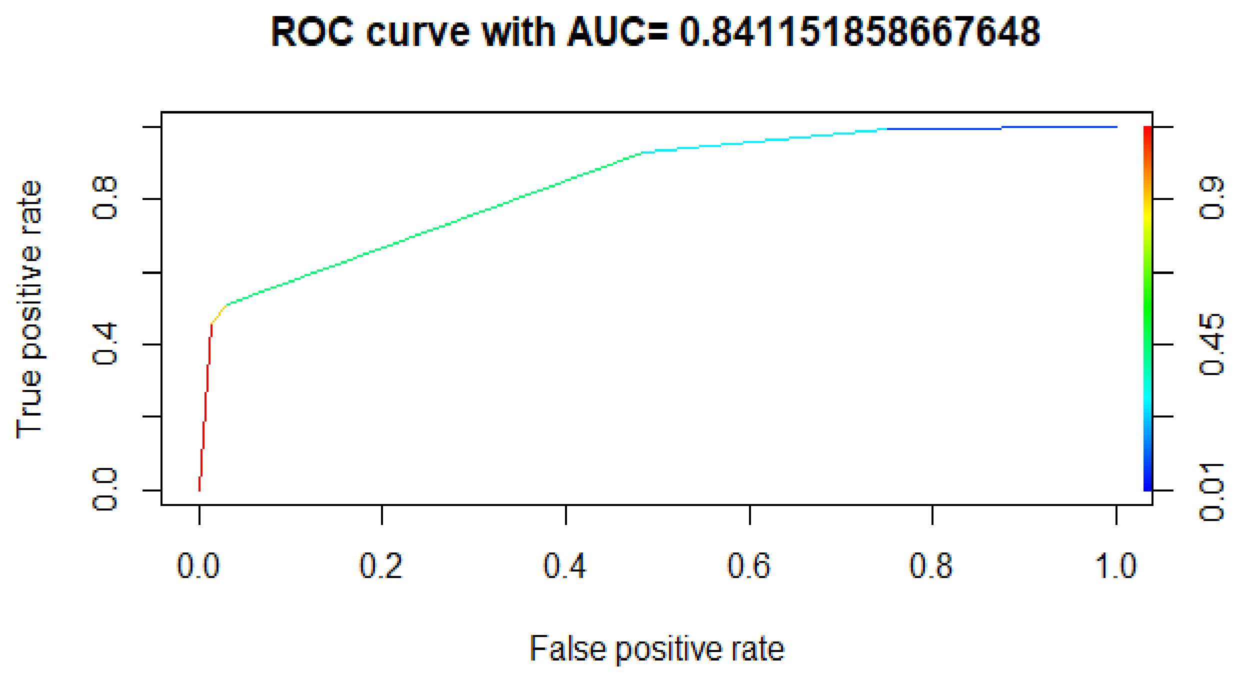
Churn



Appetency



Upselling



Criterion	Churn	Appetency	Upselling
AUC (%)	68.49	77.01	84.11
FPR (%)	75.06	14.33	48.50
FNR (%)	1.75	41.80	7.15
TPR (%) Sensitivity	24.94	85.67	51.50
TNR (%) Specificity	98.25	58.20	92.85
Accuracy	0.4498	0.8402	0.6328

Interpretation

- Here as can be seen n>>p, so the over fitting of the data isn't going to be the case.
- While coding and setting up the threshold main emphasis has been given upon improving the true positive rate (i.e. sensitivity) as these give a measure of increasing the company's profit and improving the customer value.
- AUC value for all three classes i.e. churn, appetency and upselling are decent.
- Also, TPR for appetency we are getting is highest of all 85.67% then for churn and upselling the values aren't that great.

- In order to improve, these rates we can set threshold a little bit less, but as it is obvious from the code that it has been already set to 0.2 which is way lenient value for bagging.
- Hence, it can be concluded that in case of Churn and Upselling decision trees are performing decent and worst respectively.
- The we can change the value of accuracy of the model by changing the threshold values. So main factor to finding our best model is not accuracy but TPR or AUC. AUC will remain constant throughout our model prediction even if we change the model threshold. AUC of model upselling is higher in 80s. By making model more complex, we will be able to increase the AUC.
- Churn is considered to harder to predict based on this dataset. TPR of churn can be increased by decreasing the threshold.

2. Random Forest (RF) [11]

Random forest is a very efficient technique to improve the idea of bagging. It essentially decor relates the trees by choosing a set of prediction terms randomly. This random selection leads to a good reduction in variance. In general, if a data set has a very strong predictor which appears in all the bagged trees, averaging does not help much in reducing the variance because of similar variances in all the bagged trees. However, in random forests only a random set of predictors is selected and hence, the bagged trees will have significantly different variance and averaging ultimately results in a significant reduction in variance. Thus, random forest becomes a powerful tool to improve bagged trees. If the total number of predictors is p, random forests choose a set with m number of predictors where $m > \sqrt{p}$. Moreover, random forests provide us a list of important prediction terms.

Code:

```
        library(randomForest)
#Appetency

        set.seed(1)
        bag.ups=randomForest(y_appetency~.,data=appetency_train,      mtry=52,importance
        =TRUE)
        yhat=predict(bag.ups,appetency_valid, type = "prob")
        y=appetency_valid$y_appetency
        ypred=rep('no',5000)
        ypred[yhat[,2] > 0.2] = "yes"
```

```
table(ypred,y)
varImpPlot(bag.ups)


roc_auc <- function(probabilities,dataset){
  #Command - roc_auc(probabilities,dataset)
  #probabilities are those obtained from predict function
  #dataset is the actual data (0s and 1s)
  library(ROCR)   #Install ROCR library before running
  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(yhat[,2],y)


auc=rep(0,10)
ntr=c(50,100,150,200,250,300,350,400,450,500)
mtr=c(20,24,28,32,36,40,44,48,52,56)

for (i in 1:10){
bag.ups=randomForest(y_appetency~.,data=appetency_train,
mtry=mtr,ntry=ntr[i],importance =TRUE)
yhat=predict(bag.ups,appetency_valid, type = "prob")
pr=prediction(yhat, appetency_valid$y_appetencyt)
auc=performance(pr,measure = "auc")
auc[i]=auc@y.values[[1]]
}



bag.ups=randomForest(y_appetency~.,data=appetency_train,
mtry=52,ntree=250,importance =TRUE)
yhat=predict(bag.ups,appetency_valid, type = "prob")
y=appetency_valid$y_appetency
ypred=rep('no',5000)
ypred[yhat[,2] > 0.2] = "yes"
table(ypred,y)
varImpPlot(bag.ups)
roc_auc(yhat[,2],y)

#Churn
```

```
bag.churn =randomForest(y_churn~.,data=churn_train, mtry=48,importance =TRUE)
yhat=predict(bag.ups,churn_valid, type = "prob")
y=churn_valid$y_churn
ypred=rep('no',5000)
ypred[yhat[,2] > 0.2] = "yes"
table(ypred,y)
varImpPlot(bag.ups)
roc_auc(yhat[,2],y)

auc=rep(0,10)
ntr=c(50,100,150,200,250,300,350,400,450,500)
mtr=c(20,24,28,32,36,40,44,48,52,56)
for (i in 1:10){
  bag.ups=randomForest(y_churn~.,data=churn_train,
mtry=mtr[i],ntry=ntr[i],importance =TRUE)
  yhat=predict(bag.ups,churn_valid, type = "prob")

  pr=prediction(yhat, churn_valid$y_churnt)
  auc=performance(pr,measure = "auc")
  auc[i]=auc@y.values[[1]]
}

bag.ups  =randomForest(y_churn~.,data=churn_train,  mtry=48,ntree=300,importance
=TRUE)
yhat=predict(bag.ups,churn_valid, type = "prob")
y=churn_valid$y_churn
ypred=rep('no',5000)
ypred[yhat[,2] > 0.2] = "yes"
table(ypred,y)
varImpPlot(bag.ups)
roc_auc(yhat[,2],y)



#Upselling
bag.ups    =randomForest(y_upselling~.,data=upselling_train,    mtry=58,importance
=TRUE)
yhat=predict(bag.ups,upselling_valid, type = "prob")
y=upselling_valid$y_upselling
ypred=rep('no',5000)
ypred[yhat[,2] > 0.2] = "yes"
table(ypred,y)
varImpPlot(bag.ups)

roc_auc(yhat[,2],y)
```

```
auc=rep(0,10)
ntr=c(50,100,150,200,250,300,350,400,450,500)
mtr=c(20,24,28,32,36,40,44,48,52,58)
for( i in 1:10){

bag.ups=randomForest(y_upselling~.,data=upselling_train,
mtry=mtr,ntry=ntr[i],importance =TRUE)
yhat=predict(bag.ups,upselling_valid, type = "prob")

  pr=prediction(yhat, upselling_valid$y_upsellingt)
  auc=performance(pr,measure = "auc")
  auc[i]=auc@y.values[[1]]
}
bag.ups=randomForest(y_upselling~.,data=upselling_train,
mtry=58,ntree=200,importance =TRUE)
yhat=predict(bag.ups,upselling_valid, type = "prob")
y=upselling_valid$y_upselling
ypred=rep('no',5000)
ypred[yhat[,2] > 0.2] = "yes"
table(ypred,y)
varImpPlot(bag.ups)

roc_auc(yhat[,2],y)
```

Here we have implemented random forest individually on all the three responses. For random forest we have created 3 models. Random forest have 2 parameters that can be tweaked or cross validated for optimum results.

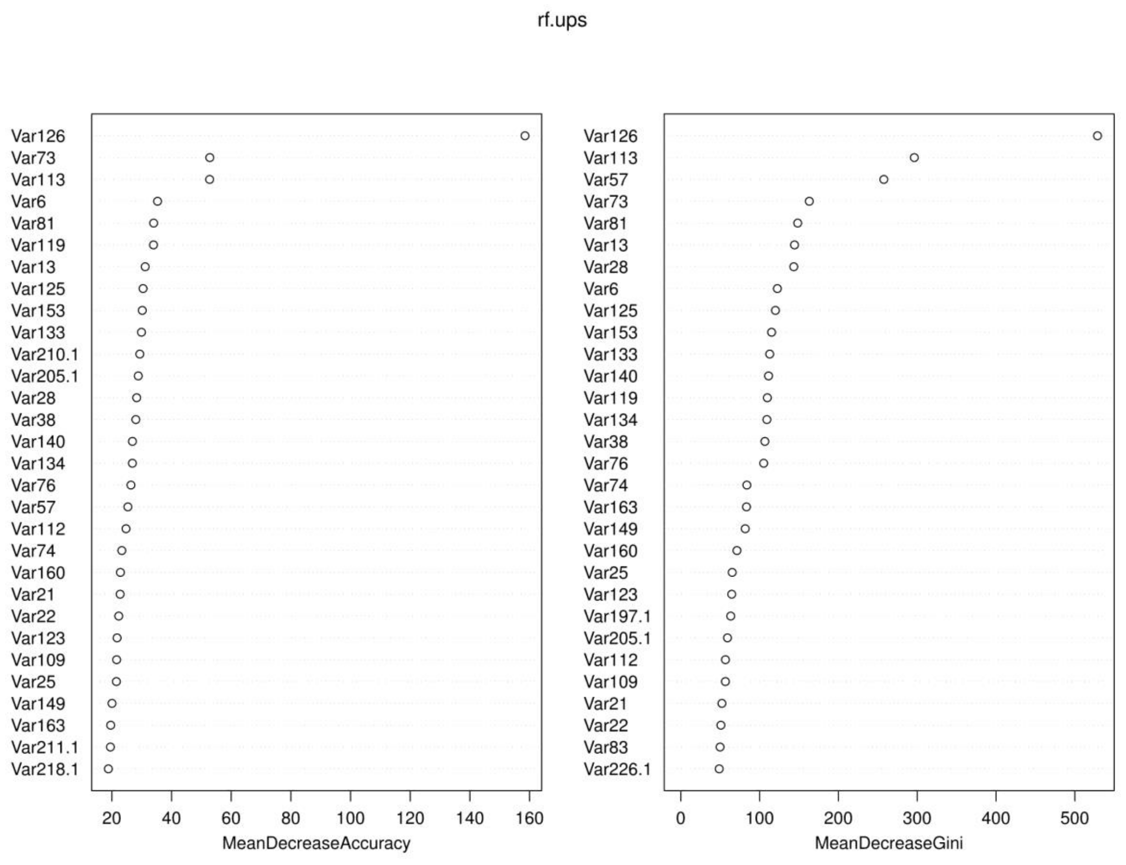
Here we have applied for loop on the model to calculate the optimum value based on AUC. We have selected AUC as optimum comparison criterion. AUC works well for intra model comparison as well as inter comparison of the model.

One advantage of using random forest is that we can get significance variable chart that can be used when implementation of model with all the predictor is taxing and time consuming. So, the noise variables are removed to get the robust result.

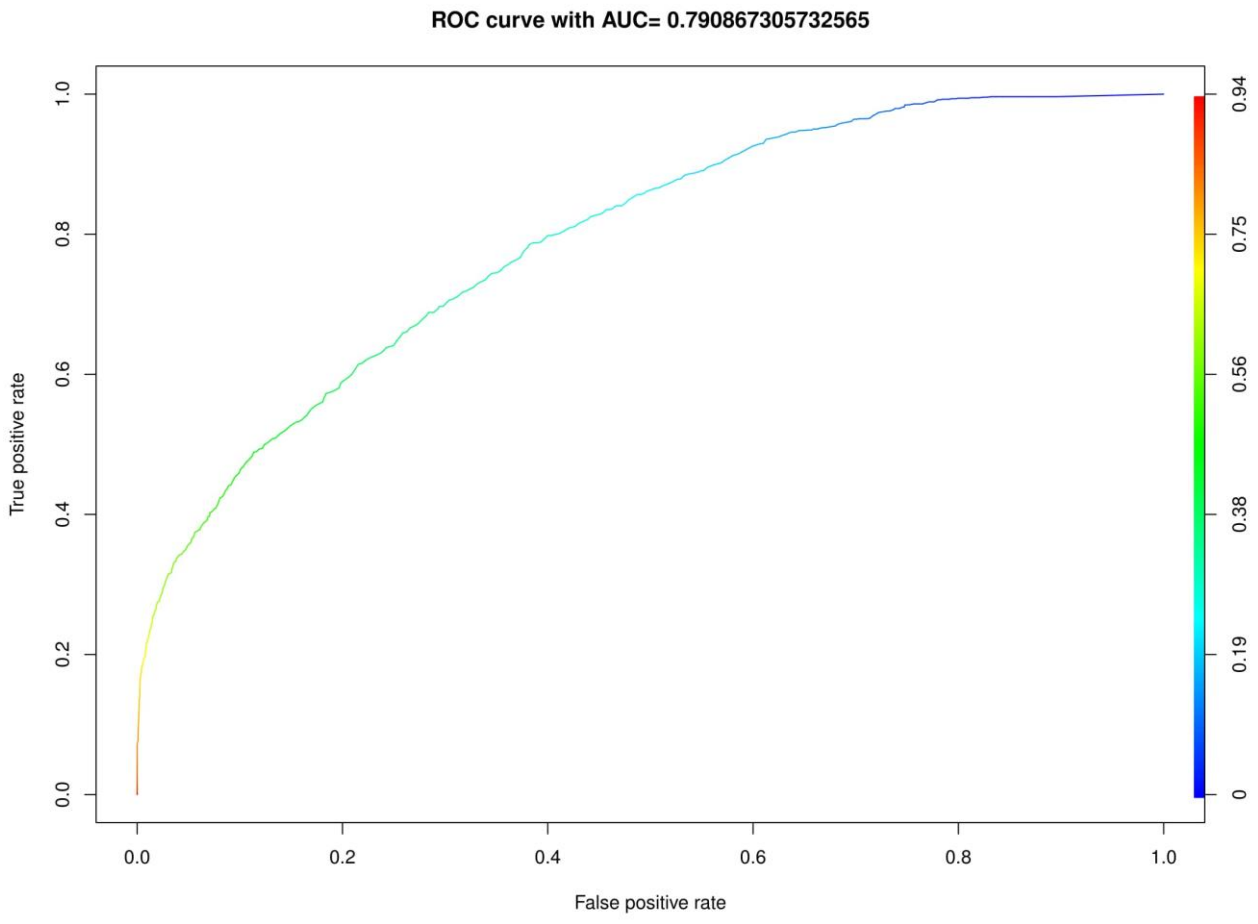
The summarized results of all the 3 outcomes are presented below for the further diagnosis.

Results:

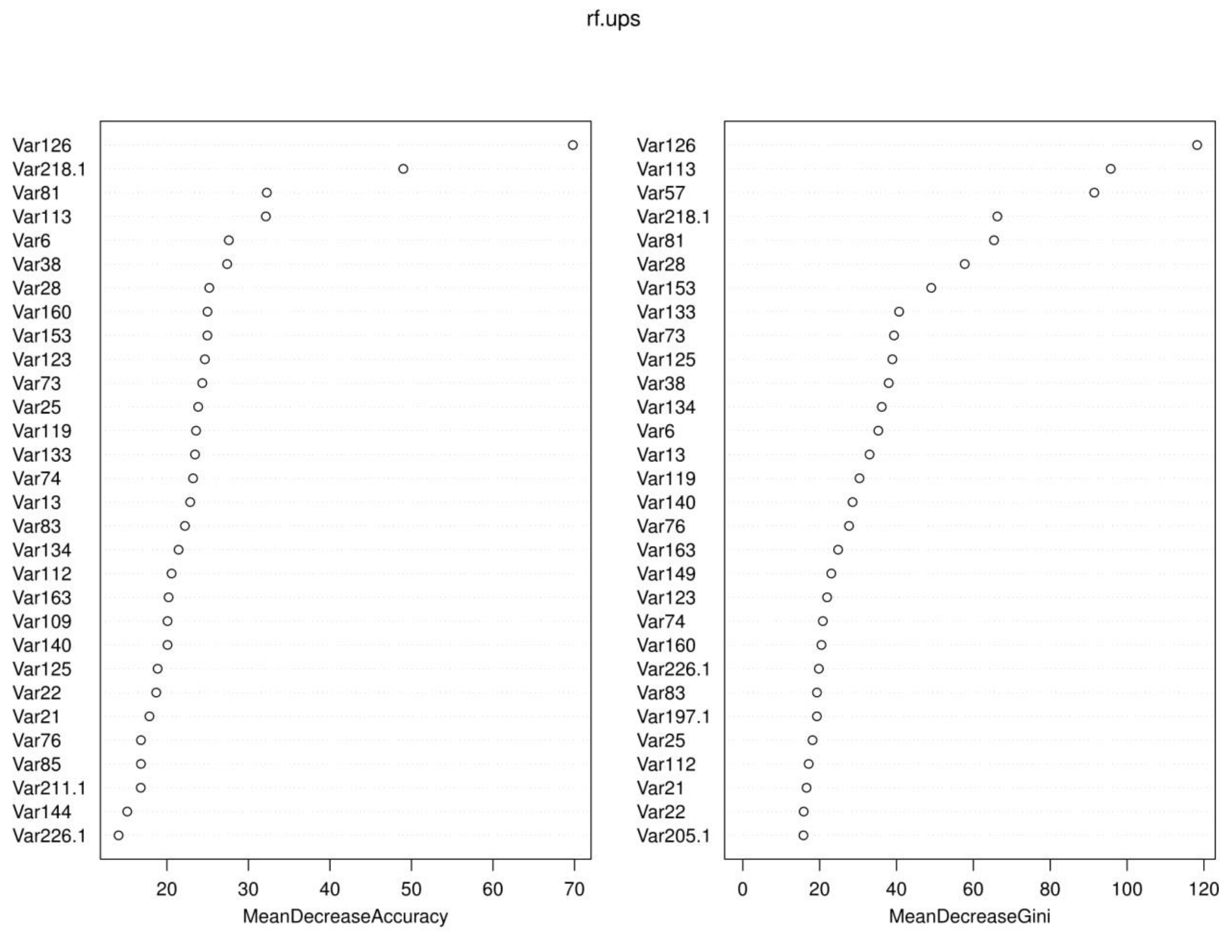
Churn:



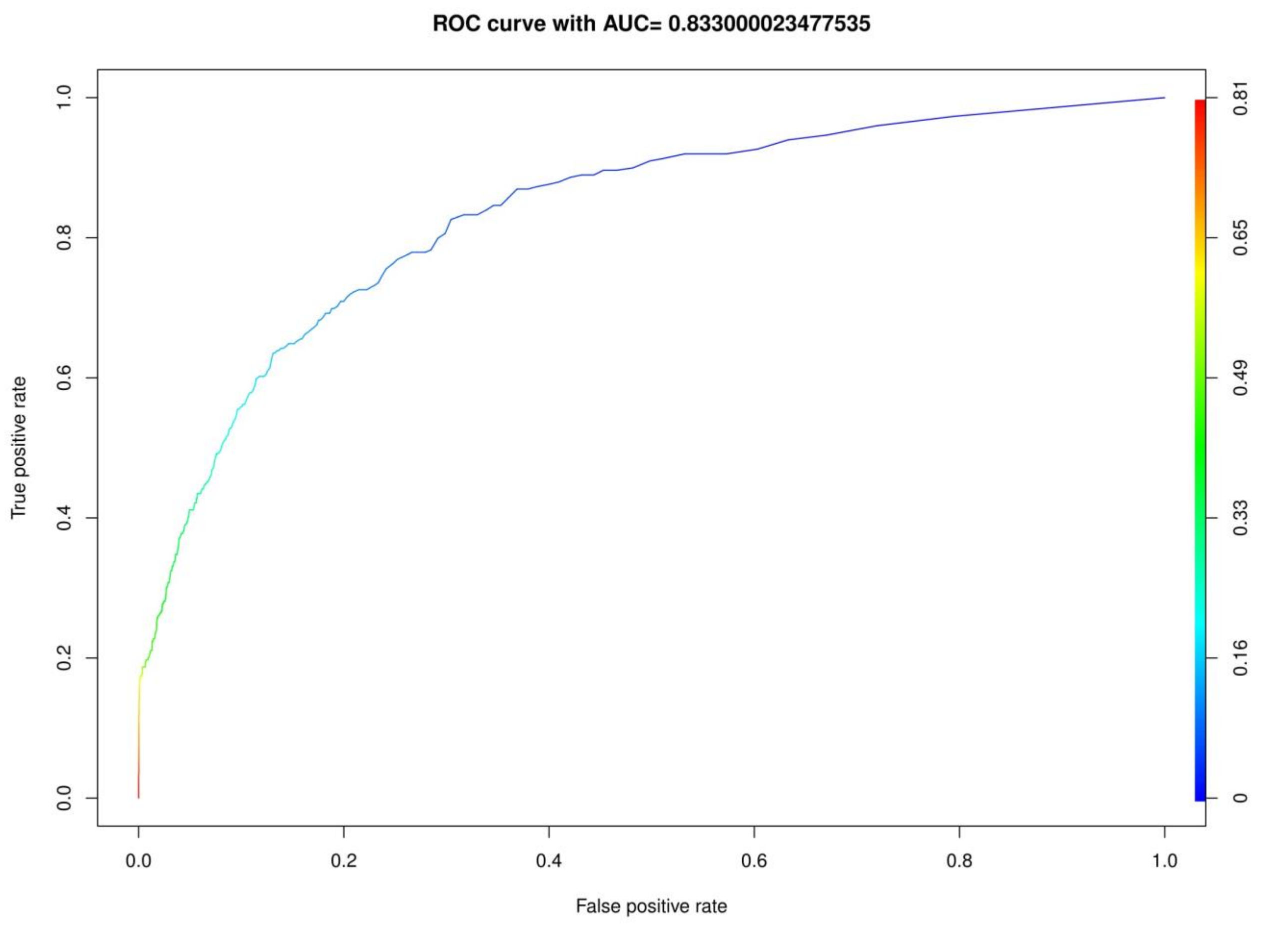
```
table(ypred,y)
        y
ypred   no  yes
  no  1646  151
  yes 1987 1216
```


ROC curve with AUC= 0.790867305732565
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0
0
0.19
0.38
0.56
0.75
0.94

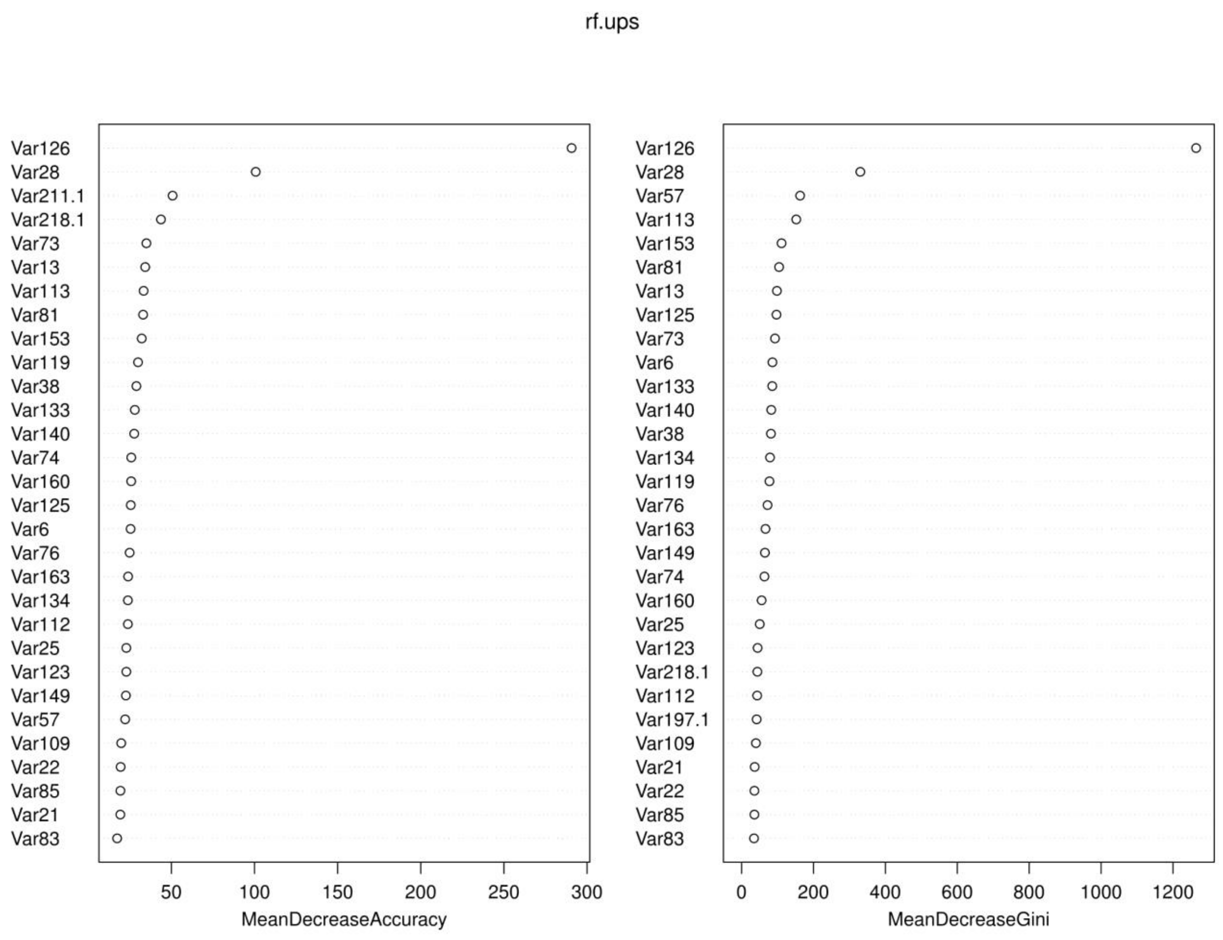
Appetency:



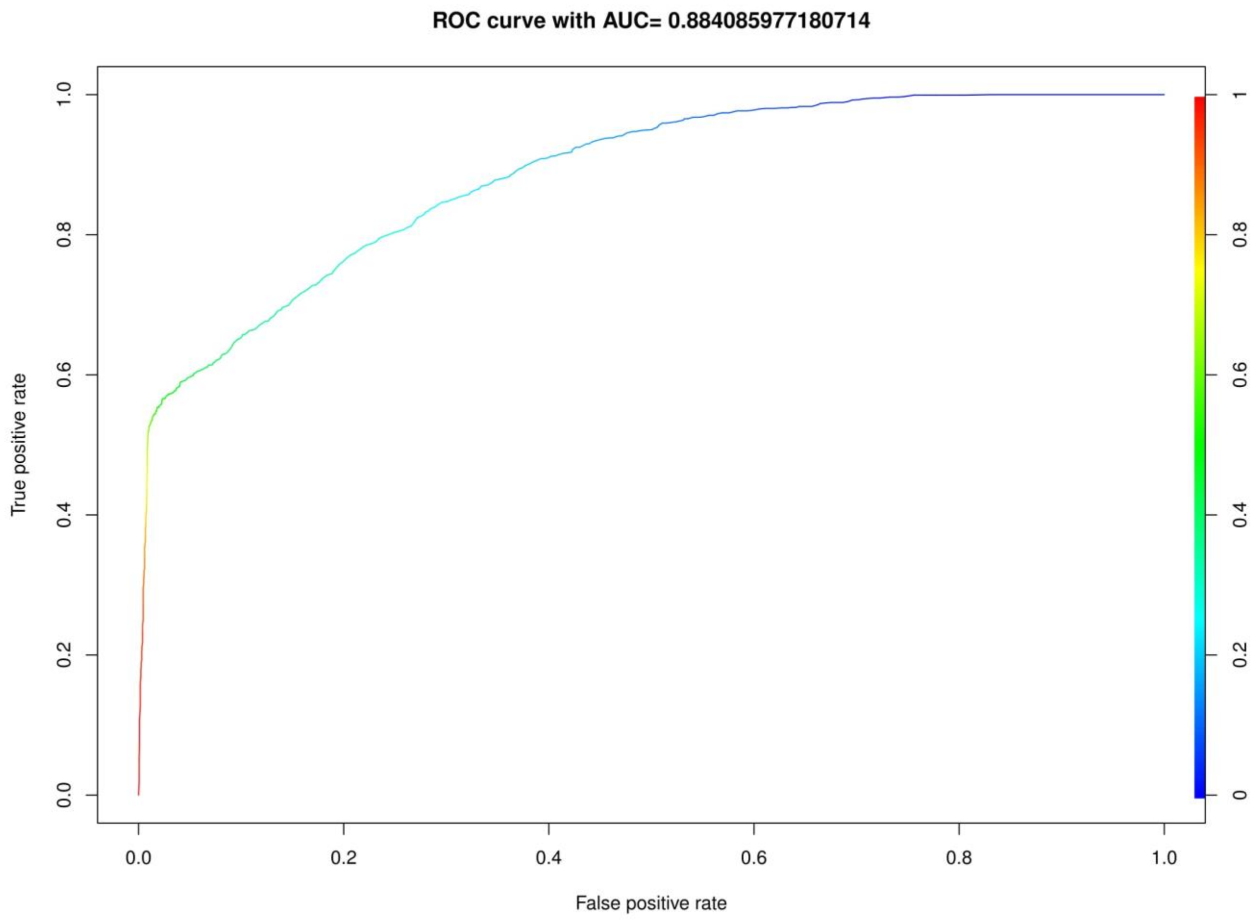
```
table(ypred,y)
           y
ypred   no  yes
  no  4216  130
  yes  485  169
```


ROC curve with AUC= 0.833000023477535
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
0
0.16
0.33
0.49
0.65
0.81

Upselling:



```
table(ypred,y)
     y
ypred   no  yes
  no  2233  139
  yes 1342 1286
```



Criterion	Churn	Appetency	Upselling
AUC	0.79	0.83	0.88
FPR	0.54	0.10	0.375
FNR	0.11	0.43	0.09
TPR	0.89	0.57	0.91
TNR	0.46	0.9	0.625
Accuracy	0.57	0.877	0.70

Interpretation:

- Since n>>p, over fitting should not be an issue.
- ‘mtry’ has been chosen for all the three levels based on several iterations.
- While selecting threshold value, emphasis has been given on TPR. Therefore, the threshold has been chosen as 0.2 to get good TPR. Further decrease in TPR results in a good number of false positives. Hence, 0.2 is taken to be the most suitable threshold value. Same threshold has been observed in all three cases.

- AUC for all three levels are satisfactory.
- The method works best for upselling and worst for appetency if TPR (or FNR) is considered. However, the method works best for appetency and worst for churn in terms of TNR (or FPR).

3. Bagging[11]

Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid over fitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

Given a standard training set D of size n, bagging generates m new training sets D_i, each of size n′, by sampling from D uniformly and with replacement. By sampling with replacement, some observations may be repeated in each D_i. If n′=n, then for large n the set D_i is expected to have the fraction (1 - 1/e) (≈63.2%) of the unique examples of D, the rest being duplicates. This kind of sample is known as a bootstrap sample. The m models are fitted using the above m bootstrap samples and combined by averaging the output (for regression) or voting (for classification).

Code:

```
#appetency
set.seed (1)

bag.app =randomForest(y_appetency~.,data=appetency_train, mtry=58,importance
=TRUE)
yhat=predict(bag.app,appetency_valid,type = "prob")
y=appetency_valid$y_appetency
ypred=rep('no',5000)
ypred[yhat[,2] > 0.2] = "yes"
table(ypred,y)
varImpPlot(bag.app)
roc_auc(yhat[,2],y)

auc=rep(0,10)
ntr=c(50,100,150,200,250,300,350,400,450,500)
for (i in 1:10){
```

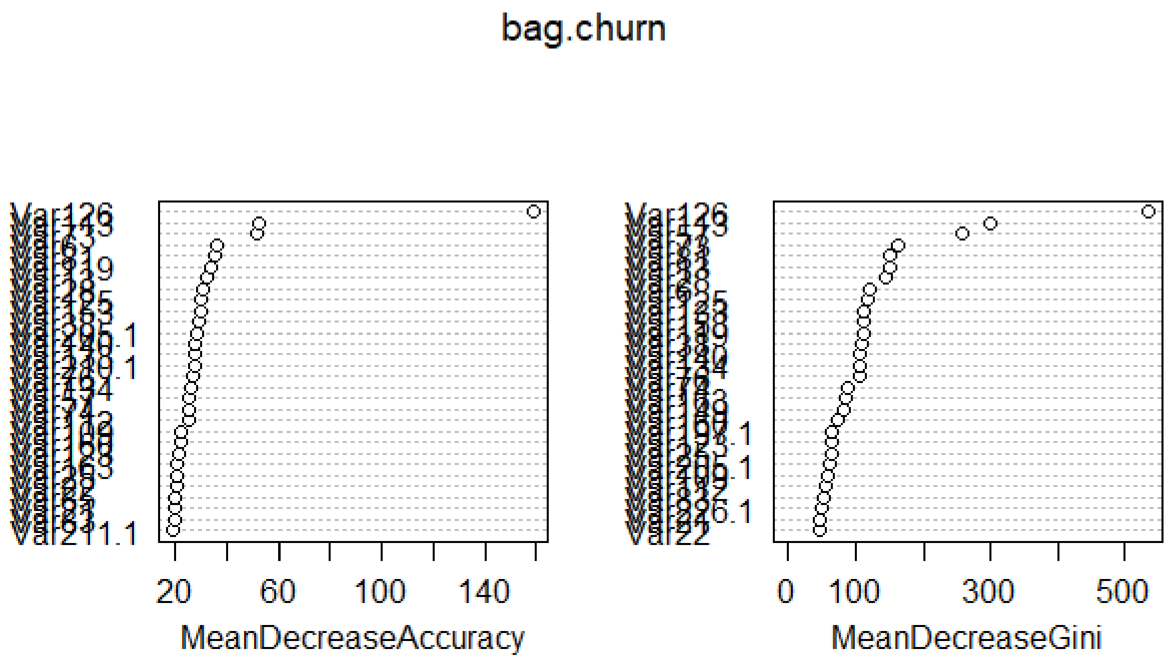
```
bag.ups=randomForest(y_appetency~.,data=appetency_train,
mtry=58,ntry=ntr[i],importance =TRUE)
yhat=predict(bag.ups,appetency_valid, type = "prob")

  pr=prediction(yhat, appetency_valid$y_appetencyt)
  auc=performance(pr,measure = "auc")
  auc[i]=auc@y.values[[1]]
}
bag.app=randomForest(y_appetency~.,data=appetency_train,
mtry=58,ntree=400,importance =TRUE)
yhat=predict(bag.app,appetency_valid,type = "prob")
y=appetency_valid$y_appetency
ypred=rep('no',5000)
ypred[yhat[,2] > 0.2] = "yes"
table(ypred,y)
varImpPlot(bag.app)
roc_auc(yhat[,2],y)

#upselling

bag.ups    =randomForest(y_upselling~.,data=upselling_train,    mtry=58,importance
=TRUE)
yhat=predict(bag.ups,upselling_valid, type = "prob")
y=upselling_valid$y_upselling
ypred=rep('no',5000)
ypred[yhat[,2] > 0.2] = "yes"
table(ypred,y)
varImpPlot(bag.ups)
roc_auc(yhat[,2],y)


auc=rep(0,10)
ntr=c(50,100,150,200,250,300,350,400,450,500)
for (i in 1:10){
bag.ups=randomForest(y_upselling~.,data=upselling_train,
mtry=58,ntry=ntr[i],importance =TRUE)
yhat=predict(bag.ups,upselling_valid, type = "prob")

  pr=prediction(yhat, upselling_valid$y_upsellingt)
  auc=performance(pr,measure = "auc")
  auc[i]=auc@y.values[[1]]
}
bag.ups=randomForest(y_upselling~.,data=upselling_train,
mtry=58,,ntree=500,importance =TRUE)
yhat=predict(bag.ups,upselling_valid, type = "prob")
y=upselling_valid$y_upselling
```

```
        ypred=rep('no',5000)
        ypred[yhat[,2] > 0.2] = "yes"
        table(ypred,y)
        varImpPlot(bag.ups)
        roc_auc(yhat[,2],y)

#churn

        bag.churn =randomForest(y_churn~.,data=churn_train, mtry=58,importance =TRUE)
        yhat=predict(bag.churn,churn_valid, type = "prob")
        y=churn_valid$y_churn
        ypred=rep('no',5000)
        ypred[yhat[,2] > 0.2] = "yes"
        table(ypred,y)
        varImpPlot(bag.churn)
        roc_auc(yhat[,2],y)

        auc=rep(0,10)
        ntr=c(50,100,150,200,250,300,350,400,450,500)

        for (i in 1:10){
         bag.ups =randomForest(y_churn~.,data=churn_train, mtry=58,ntry=ntr[i],importance
        =TRUE)
         yhat=predict(bag.ups,churn_valid, type = "prob")

          pr=prediction(yhat, churn_valid$y_churnt)
          auc=performance(pr,measure = "auc")
          auc[i]=auc@y.values[[1]]
        }

        bag.churn =randomForest(y_churn~.,data=churn_train, mtry=58,nree=450,importance
        =TRUE)
        yhat=predict(bag.churn,churn_valid, type = "prob")
        y=churn_valid$y_churn
        ypred=rep('no',5000)
        ypred[yhat[,2] > 0.2] = "yes"
        table(ypred,y)

        roc_auc(yhat[,2],y)
```

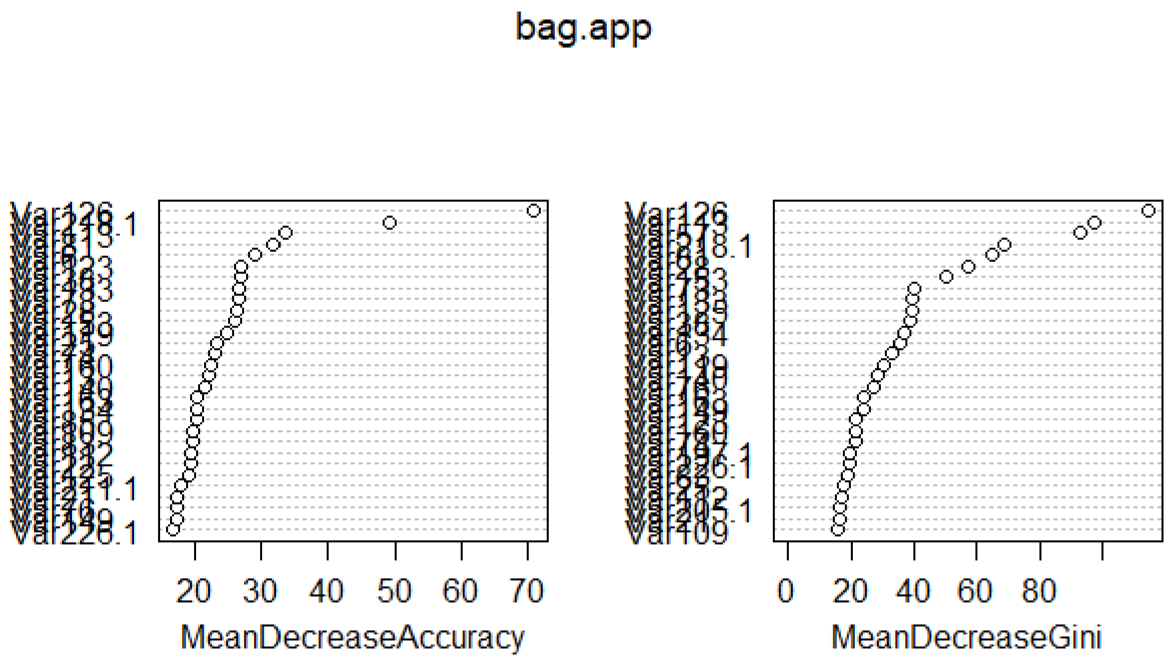
As bagging requires all the predictors for the making of the trees. Here we have applied for loop only for n tree. As we have 3 models, three models have different optimum n trees. The importance variable chart and diagnosis of the plot is described below.

Bagging importance

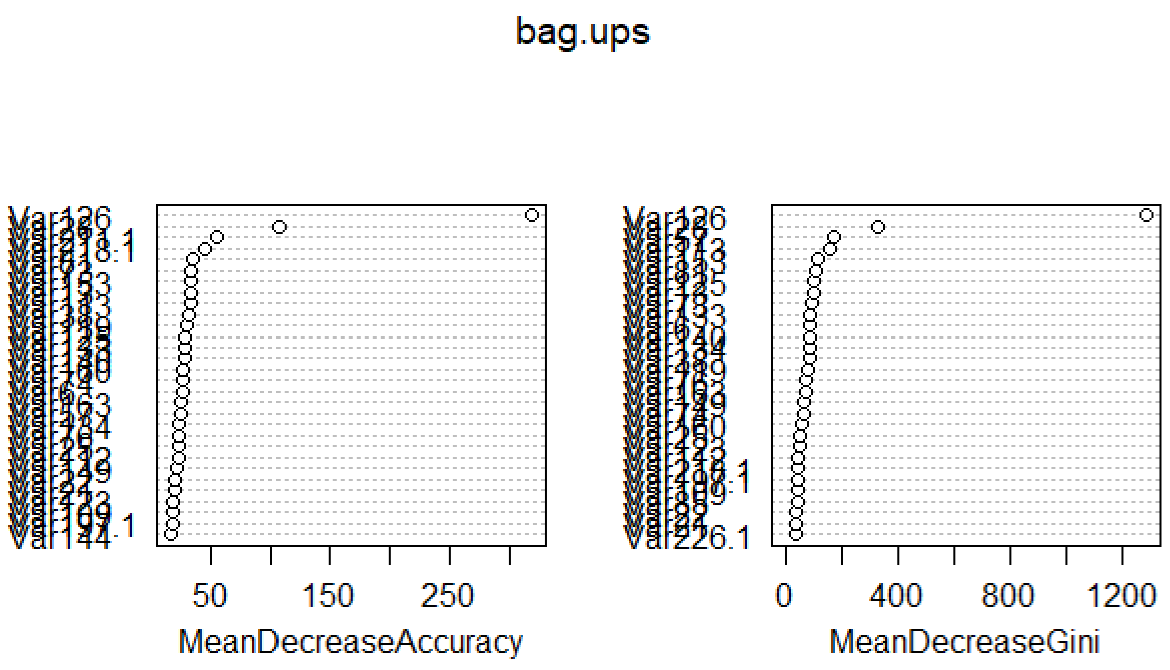
Churn



Appetency

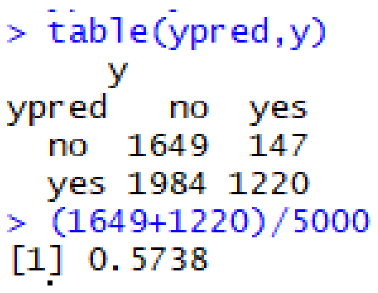


Upselling



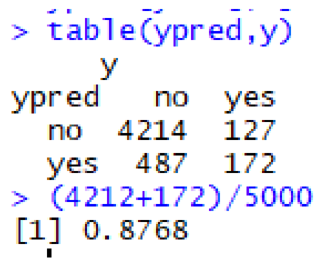
Confusion matrix

Churn



```
> table(ypred,y)
      y
ypred    no  yes
  no   1649  147
  yes  1984 1220
> (1649+1220)/5000
[1] 0.5738
```

Appetency



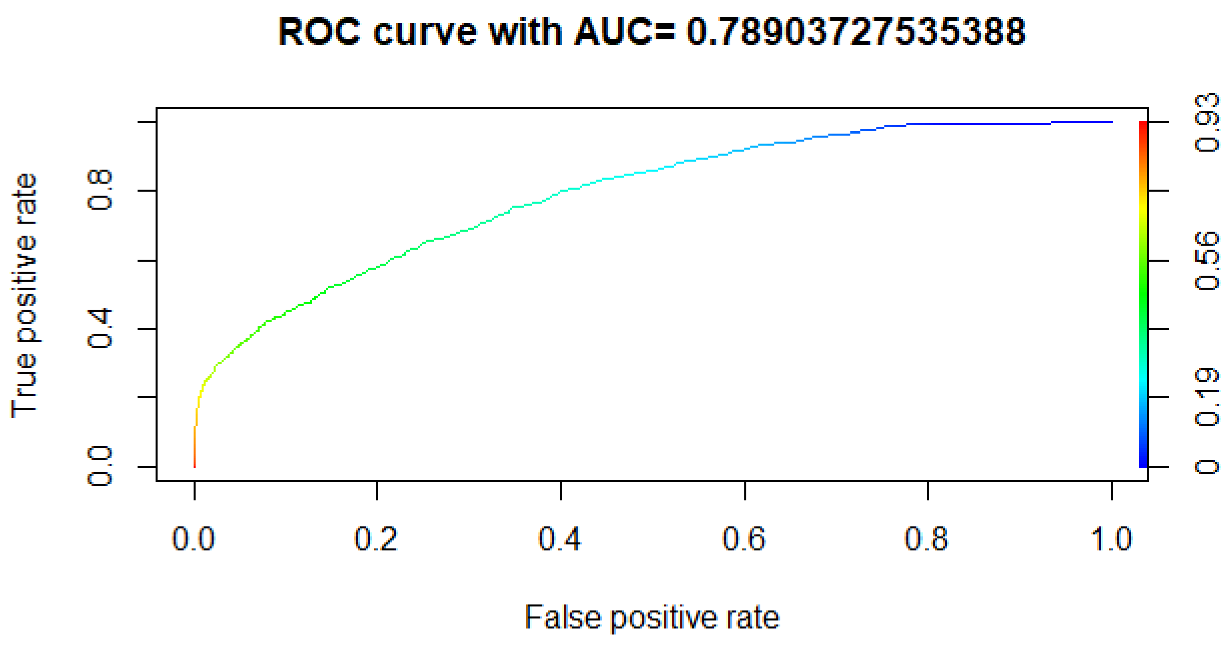
```
> table(ypred,y)
      y
ypred    no  yes
  no   4214  127
  yes   487  172
> (4212+172)/5000
[1] 0.8768
```

Upselling

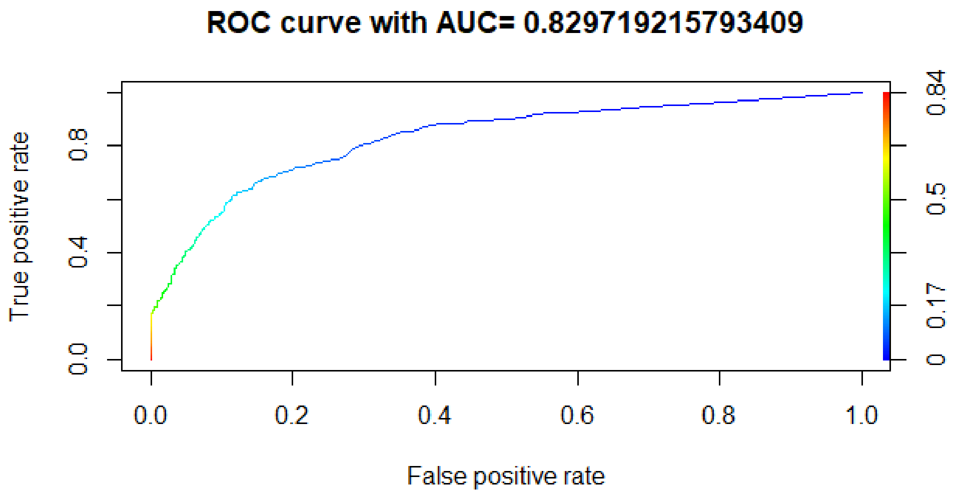
```
> table(ypred,y)
      y
ypred    no  yes
   no  2226  145
   yes 1349 1280
> (2226+1280)/5000
[1] 0.7012
```

ROC Curves

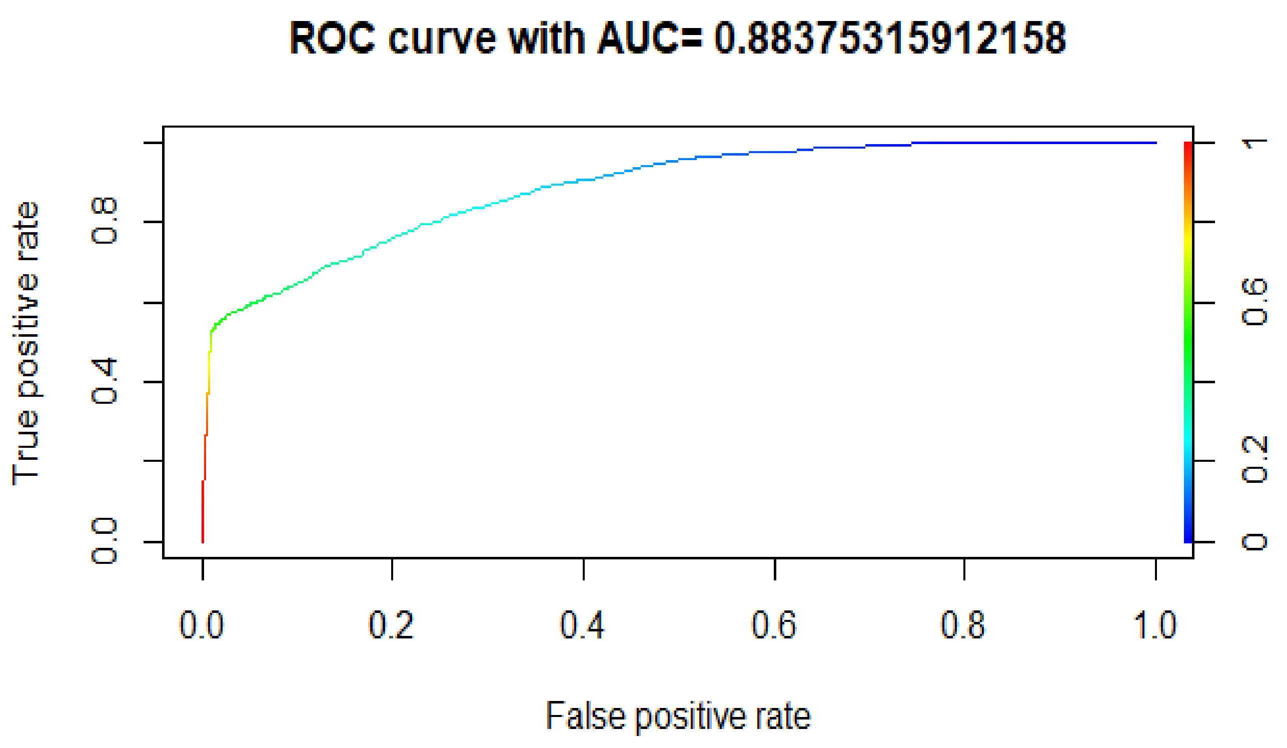
Churn



Appetency



Upselling



Criterion	Churn	Appetency	Upselling
AUC (%)	78.90	82.97	88.37
FPR (%)	54.61	10.35	37.73
FNR (%)	10.75	42.47	10.17
TPR (%) Sensitivity	45.39	89.65	62.26
TNR (%) Specificity	89.24	57.52	89.82
Accuracy	0.5738	0.8768	0.7012

Interpretation:

- Here as can be seen n>>p, so the over fitting of the data isn't going to be the case.
- While coding and setting up the threshold main emphasis has been given upon improving the true positive rate (i.e. sensitivity) as these give a measure of increasing the company's profit and improving the customer value.
- AUC value for all three classes i.e. churn, appetency and upselling are decent.

- Also, TPR for appetency we are getting is highest of all 89.65% then for churn and upselling the values aren't that great in case of bagging.
- In order to improve, these rates we can set threshold a little bit less, but as it is obvious from the code that it has been already set to 0.2 which is way lenient value for bagging.
- Hence, it can be concluded that in case of Churn and Upselling bagging is performing decent and worst respectively.
- Also, TNR (specificity) is better in case of churn and upselling while in case of appetency it isn't decent.
- We can change the value of accuracy of the model by changing the threshold values. So main factor to finding our best model is not accuracy but TPR or AUC. AUC will remain constant throughout our model prediction even if we change the model threshold. AUC of model upselling is higher in upper 80s. TPR of upselling can also be increased by decreasing the threshold but at the cost of increasing False positive.
- By making model more complex. we will be able to increase the AUC. By making model complex means increasing the number of trees and as well as increasing weight of important variables for significant predictors.
- Churn considered to harder to predict based on this dataset. TPR of churn can be increased by decreasing the threshold.

4. **Boosting**[11][12]

It is a supervised classification technique, which essentially is used for reducing bias and also variance. It basically constitutes of weak learners to make a strong classifiers. Boosting works similar way as bagging except that here tree is grown is sequentially (slow learning!). Boosting algorithm is as follows [12]:

1. Initially (b=1), Build a "small tree" with number of splits d<5

2. If the residuals (training errors) are large 1. Scale the residuals by a parameter 2. Fit another small tree to predict residuals

3. Repeat step 2 until the resulting number of steps B is reached or the error reaches desired level.

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

Figure 6

Coding:

```
#Upsellling

        #Import the dataset with Rstudio features
        #Import the dataset with Rstudio features
        upselling_train <-read.table('upselling_train.csv',header=TRUE,
        sep=',',stringsAsFactors=TRUE)
        upselling_valid <-read.table('upselling_valid.csv',header=TRUE,
        sep=',',stringsAsFactors=TRUE)

        upselling_train=upselling_train[,-1]
        upselling_valid=upselling_valid[,-1]

        datx = colnames(upselling_train[2:59])
        daty = colnames(upselling_train[1])
        trh=as.h2o(upselling_train,destination_frame = "trh")
        tth=as.h2o(upselling_valid,destination_frame="tth")


        ups_rf=h2o.randomForest(x = datx,  # column numbers for predictors
                        y = daty,   # column number for label
                        training_frame = "trh",
                        validation_frame = 'tth',nfolds=5,seed=1)

        model=h2o.gbm(x=datx,y=daty,training_frame="trh",
                nfolds=5,ntrees=1000,max_depth=2,
                distribution = "bernoulli",
                validation_frame = 'tth',
                seed=1,learn_rate=.2)



        pr =h2o.predict(model,tth)
        df_yhat_test <- as.data.frame(pr)

        table(df_yhat_test[,1],upselling_valid[,1])
        (894+3246)/5000

        roc_auc(df_yhat_test[,3],upselling_valid[,1])



#Appetency
```

```
          #import dataset with r studio inbuilt function
          appetency_train <-read.table('appetency_train.csv',header=TRUE,
          sep=',',stringsAsFactors=TRUE)
          appetency_valid <-read.table('appetency_valid.csv',header=TRUE,
          sep=',',stringsAsFactors=TRUE)

          appetency_train=appetency_train[,-1]
          appetency_valid=appetency_valid[,-1]



          datx = colnames(appetency_train[2:59])
          daty = colnames(appetency_train[1])
          trh=as.h2o(appetency_train,destination_frame = "trh")
          tth=as.h2o(appetency_valid,destination_frame="tth")

#training


#prediction
          model=h2o.gbm(x=datx,y=daty,training_frame="trh",
                    nfolds=5,ntrees=1000,max_depth=2,
                    distribution = "bernoulli",
                    validation_frame = 'tth',
                    seed=1,learn_rate=.2)



          pr =h2o.predict(model,tth)
          df_yhat_test <- as.data.frame(pr)
          table(df_yhat_test[,1],appetency_valid[,1])


          app_pred_y = rep("no", length=5000)
          app_pred_y[df_yhat_test[,3] > .2] = "yes"
          table(predict=app_pred_y,truth=appetency_valid$y_appetency)


          (4367+123)/5000
           roc_auc(df_yhat_test[,3],appetency_valid[,1])


#Churn

          #Import the dataset with Rstudio features
          churn_train <-read.table('churn_train.csv',header=TRUE, sep=',',stringsAsFactors=TRUE)
          churn_valid <-read.table('churn_valid.csv',header=TRUE, sep=',',stringsAsFactors=TRUE)

          churn_train=churn_train[,-1]
          churn_valid=churn_valid[,-1]
```

```
datx = colnames(churn_train[2:59])
daty = colnames(churn_train[1])
trh=as.h2o(churn_train,destination_frame = "trh")
tth=as.h2o(churn_valid,destination_frame="tth")
```

#training

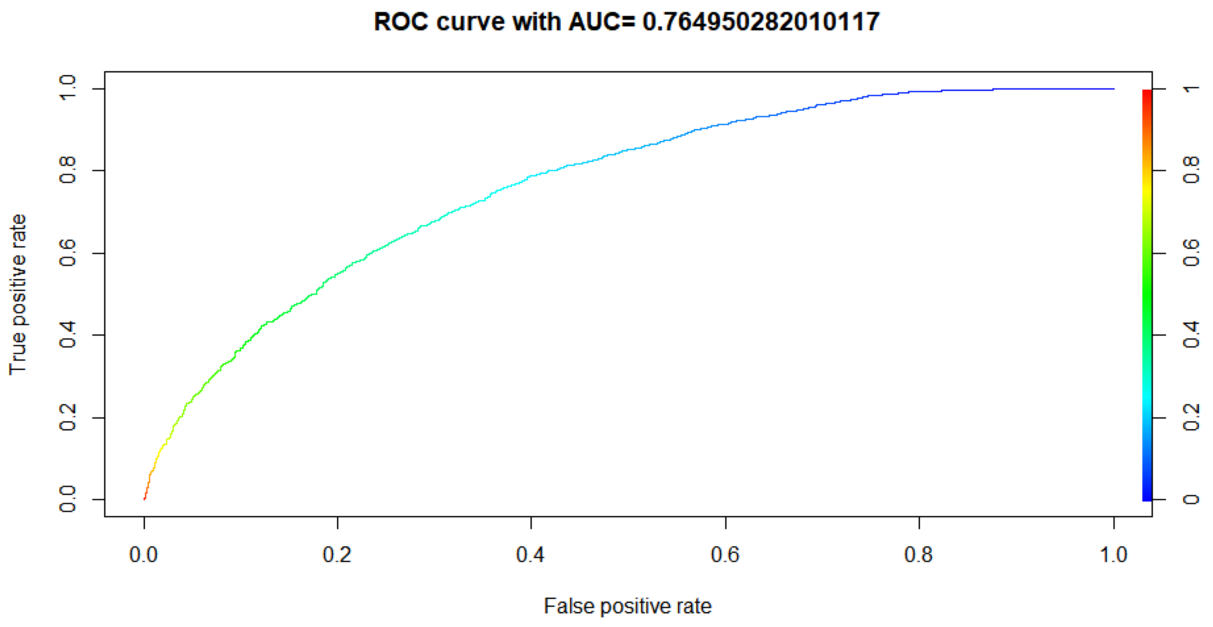
```
model=h2o.gbm(x=datx,y=daty,training_frame="trh",
        nfolds=5,ntrees=1000,max_depth=2,
        distribution = "bernoulli",
        validation_frame = 'tth',
        seed=1,learn_rate=.2)
```

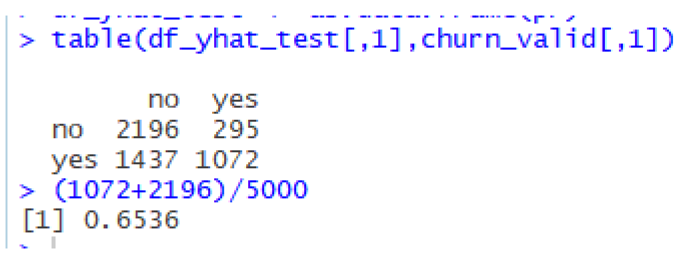
#prediction

```
pr =h2o.predict(model,tth)
df_yhat_test <- as.data.frame(pr)

table(df_yhat_test[,1],churn_valid[,1])
(1072+2196)/5000
roc_auc(df_yhat_test[,3],churn_valid[,1])
```

Results:

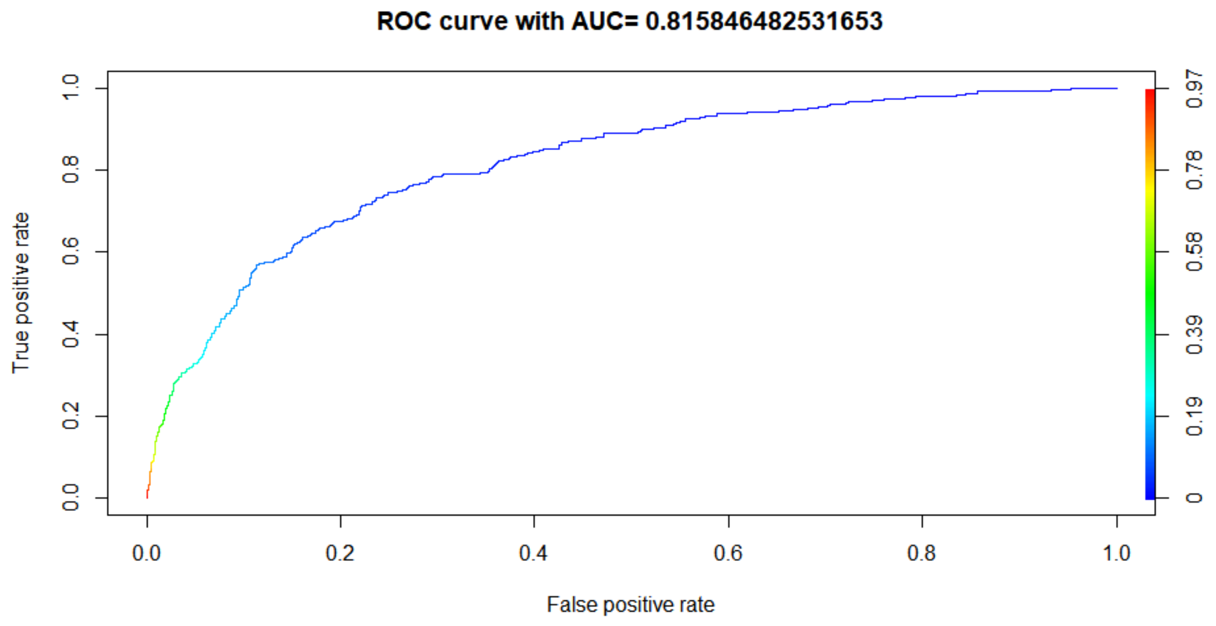
Churn

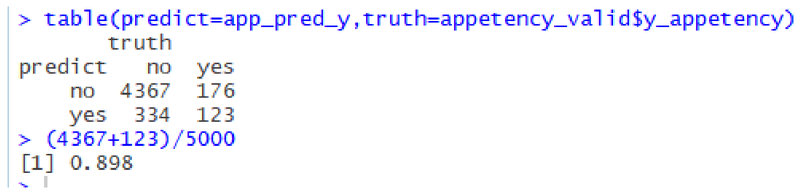




```
> table(df_yhat_test[,1],churn_valid[,1])

        no  yes
  no  2196  295
  yes 1437 1072
> (1072+2196)/5000
[1] 0.6536
```

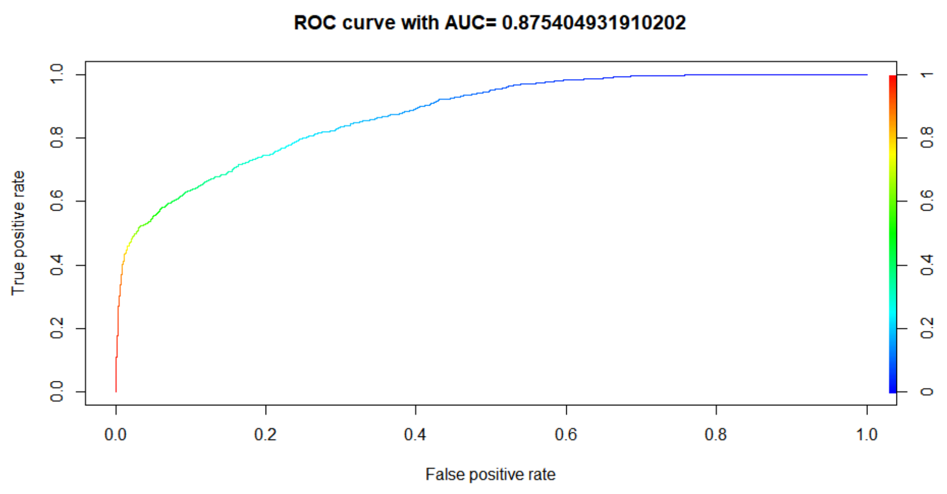
Appetency

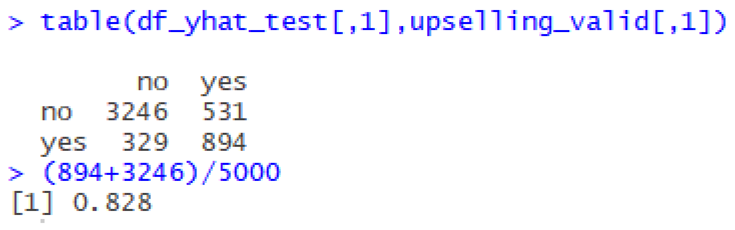




```
> table(predict=app_pred_y,truth=appetency_valid$y_appetency)
       truth
predict   no  yes
    no  4367  176
    yes  334  123
> (4367+123)/5000
[1] 0.898
```

Upselling





```
> table(df_yhat_test[,1],upselling_valid[,1])

        no  yes
  no  3246  531
  yes  329  894
> (894+3246)/5000
[1] 0.828
```

Diagnosis:

Criterion	Churn	Appetency	Upselling
AUC (%)	76.50	81.58	87.54
FPR (%)	39.56	7.11	9.20
FNR(%)	21.59	59	37.27
TPR(%) (sensitivity)	78.41	41	62.73
TNR(%) (specificity)	60.44	92.89	90.80
Accuracy (%)	65.36	89.8	82.8

Interpretation:

- Here as can be seen n>>p, so the over fitting of the data isn't going to be the case.
- While coding and setting up the threshold main emphasis has been given upon improving the true positive rate (i.e. sensitivity)
- AUC value for churn is 76.50 which can be said to have decent value while in case of appetency and upselling they are 81.58 and 87.54 % respectively, which is great in comparison of earlier methods used.
- Also, TPR for churn we are getting is highest of all 78.41% then for upselling the value isn't that great while in case of appetency it's least of all i.e. 41%.
- In order to improve, these rates we can set threshold a little bit less, but as it is obvious from the code that it has been already set to 0.2 which is way lenient value for the boosting.

- Hence, it can be concluded that in case of appetency boosting is performing worst.
- Also, TNR (specificity) isn't better in case of churn while for appetency and upselling it's working very nicely.

5. <u>Support Vector machine (SVM)</u>[11]

Classification approach that was developed in computer science community is one of the best "out of the box" approach. It is basically the generalization of the "maximal margin classifier". It is the ideal method for classifying the observations with non-linear boundaries and two classes however, it can be extended to work with more than two classes. Here, the essential thing to understand in SVM is the margin- which can defined as the maximum distance between the hyper plane (plane separating various classes) and the nearest point. However, in order to accommodate every observations there has to be some leeway or cost (C) of assigning the point in different class. And accordingly, it follows the following optimization model, which itself is a self-explanatory!

$$
\begin{aligned}
&\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \\
&\text{subject to } \sum_{j=1}^{p}\beta_j^2 = 1, \\
&y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1-\epsilon_i), \\
&\epsilon_i \geq 0, \quad \sum_{i=1}^{n}\epsilon_i \leq C,
\end{aligned}
$$

Figure 7

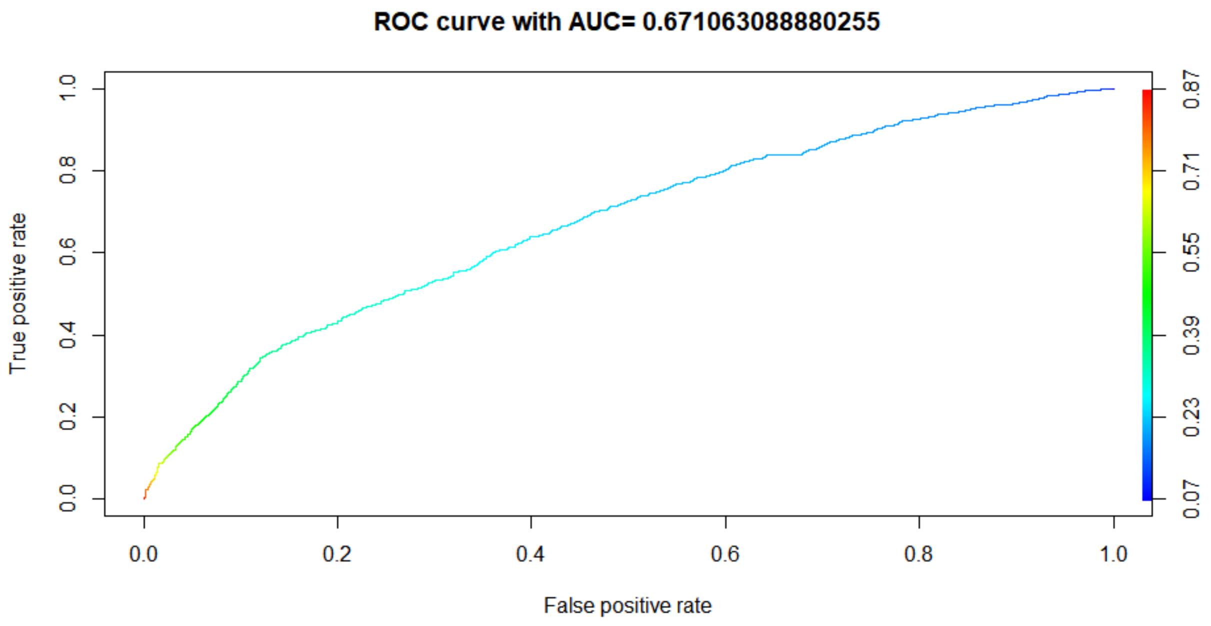
```
        #UsinG important predictors
#Upsellling

        #using parellal computing
        library(doParallel)
        detectCores()
        cl=makeCluster(3)
        registerDoParallel(cl)
        getDoParWorkers()

        library(dummies)
```

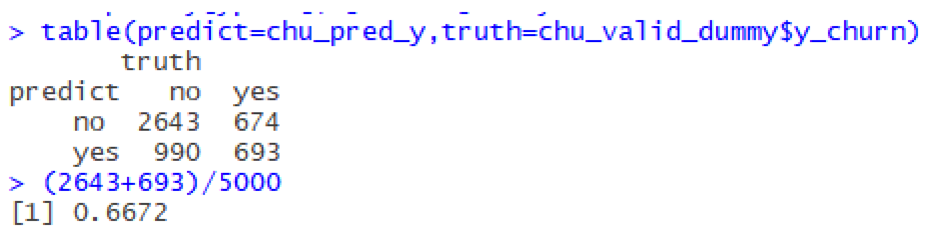
```
ups_train_dummy=dummy.data.frame(upselling_train,names=c("Var226.1","Var197.1","Var
205.1","Var206.1"),sep="_")
ups_valid_dummy=dummy.data.frame(upselling_valid,names=c("Var226.1","Var197.1","Var
205.1","Var206.1"),sep="_")
upselling_imp <-read.table('upselling_imp.csv',header=TRUE, sep=',',stringsAsFactors=F)

x=ups_train_dummy[,upselling_imp]
y=ups_train_dummy[,1]


y_upselling=y
ups_train_dummy=cbind(y_upselling,x)

library(kernlab)
svm_ups=ksvm(y_upselling~.,data=ups_train_dummy,
        type='C-svc',scaled=T,kernel='rbfdot',
        cost=10,prob.model=T,kpar=list(sigma=.1))
ypred=predict(svm_ups,ups_valid_dummy,type="probabilities")


ups_pred_y = rep("no", length=5000)
ups_pred_y[ypred[,2] > 0.2] = "yes"

table(predict=ups_pred_y,truth=ups_valid_dummy$y_upselling)



pred <- prediction(ypred[,2],upselling_valid[,1])
# Plot ROC curve
prf <- performance(pred, measure ="tpr",x.measure ="fpr")
auc=performance(pred,measure = "auc")
auc=auc@y.values[[1]]
plot(prf,colorize=TRUE,main=paste("ROC curve with AUC=",auc))



#churn

chu_train_dummy=dummy.data.frame(churn_train,names=c("Var226.1","Var197.1","Var205
.1","Var206.1"),sep="_")
chu_valid_dummy=dummy.data.frame(churn_valid,names=c("Var226.1","Var197.1","Var20
5.1","Var206.1"),sep="_")
churn_imp <-read.table('churn_imp.csv',header=TRUE, sep=',',stringsAsFactors=F)


x=chu_train_dummy[,churn_imp]
y=chu_train_dummy[,1]



y_churn=y
```

```
chu_train_dummy=cbind(y_churn,x)


svm_chu=ksvm(y_churn~.,data=chu_train_dummy,
          type='C-svc',scaled=T,kernel='rbfdot',
          cost=10,prob.model=T,kpar=list(sigma=.1))
ypred=predict(svm_chu,chu_valid_dummy,type="probabilities")


chu_pred_y = rep("no", length=5000)
chu_pred_y[ypred[,2] > 0.3] = "yes"

table(predict=chu_pred_y,truth=chu_valid_dummy$y_churn)



pred <- prediction(ypred[,2],churn_valid[,1])
# Plot ROC curve
prf <- performance(pred, measure ="tpr", x.measure ="fpr")
auc=performance(pred,measure = "auc")
auc=auc@y.values[[1]]
plot(prf,colorize=TRUE,main=paste("ROC curve with AUC=",auc))

#appetency

app_train_dummy=dummy.data.frame(appetency_train,names=c("Var226.1","Var197.1","Va
r205.1","Var206.1"),sep="_")
app_valid_dummy=dummy.data.frame(appetency_valid,names=c("Var226.1","Var197.1","V
ar205.1","Var206.1"),sep="_")
appetency_imp <-read.table('appetency_imp.csv',header=TRUE, sep=',',stringsAsFactors=F)



x=app_train_dummy[,appetency_imp]
y=app_train_dummy[,1]



y_appetency=y
app_train_dummy=cbind(y_appetency,x)


svm_app=ksvm(y_appetency~.,data=app_train_dummy,
          type='C-svc',scaled=T,kernel='rbfdot',
          cost=10,prob.model=T,kpar=list(sigma=.01))
ypred=predict(svm_app,app_valid_dummy,type="probabilities")


app_pred_y = rep("no", length=5000)
app_pred_y[ypred[,2] > 0.07] = "yes"
```

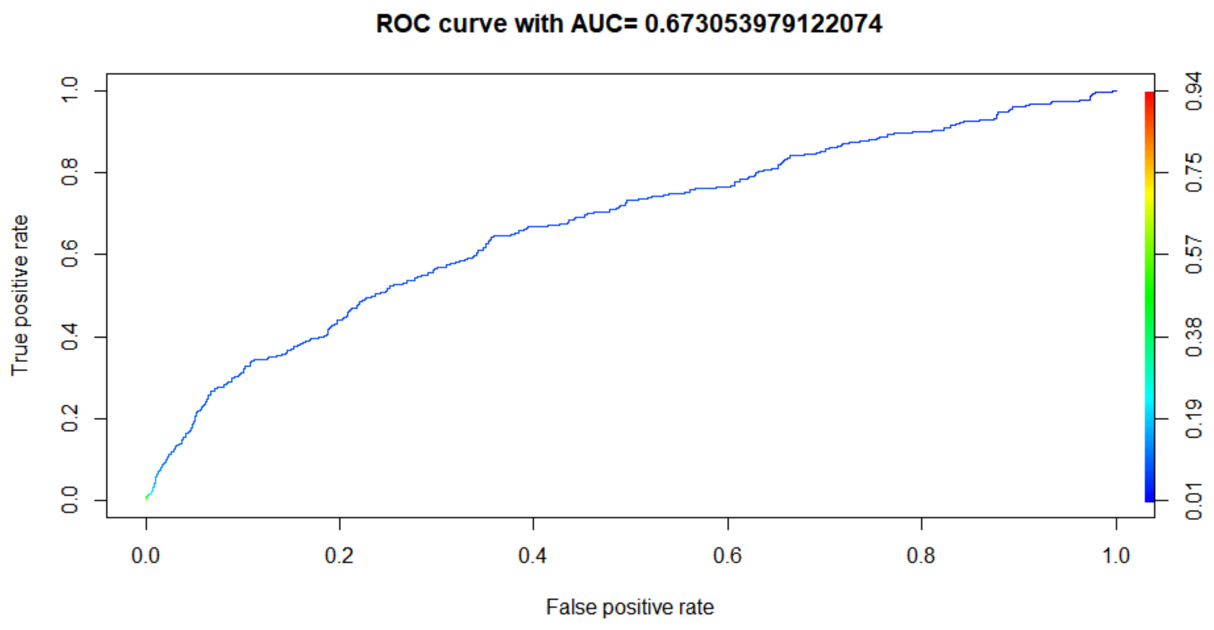
```
table(predict=app_pred_y,truth=app_valid_dummy$y_appetency)


pred <- prediction(ypred[,2],appetency_valid[,1])
# Plot ROC curve
prf <- performance(pred, measure ="tpr", x.measure ="fpr")
auc=performance(pred,measure = "auc")
auc=auc@y.values[[1]]
plot(prf,colorize=TRUE,main=paste("ROC curve with AUC=",auc))


stopCluster(cl)
```

Results:

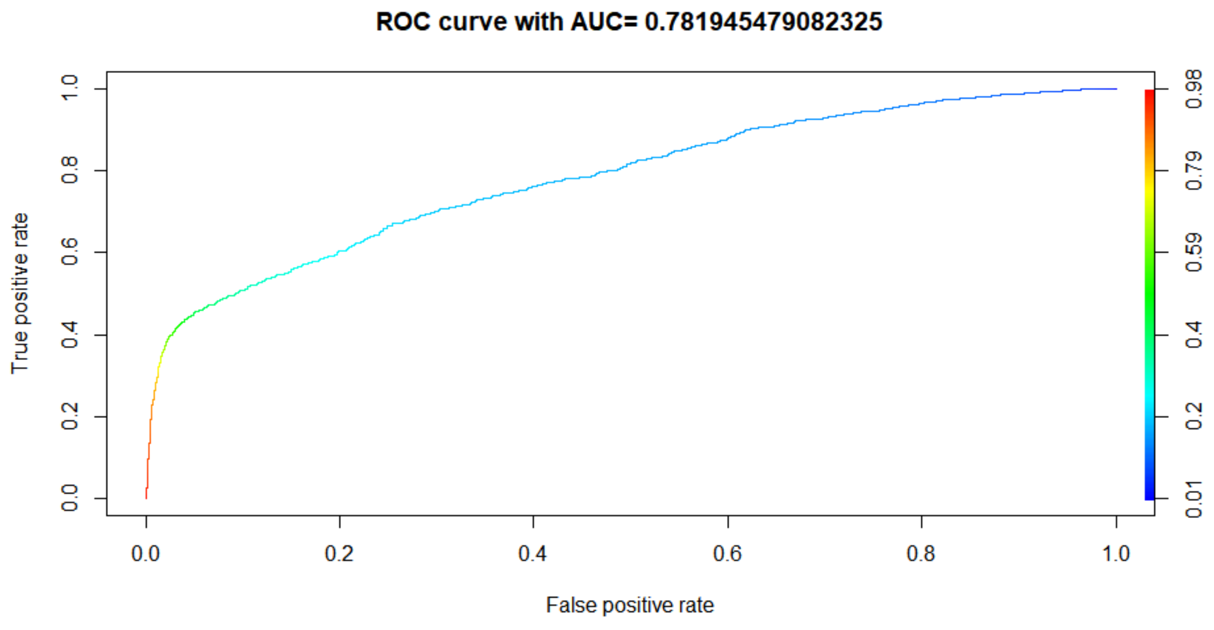


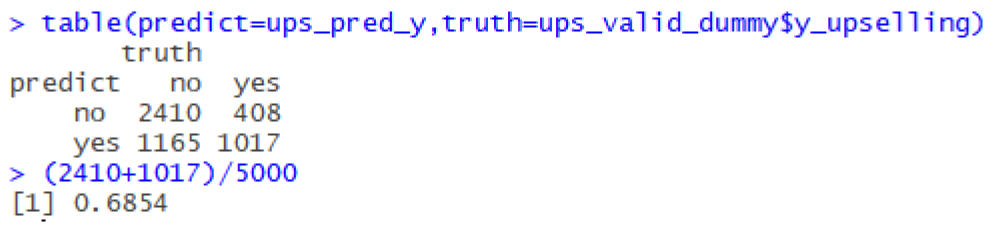


```
> table(predict=chu_pred_y,truth=chu_valid_dummy$y_churn)
       truth
predict   no  yes
    no  2643  674
    yes  990  693
> (2643+693)/5000
[1] 0.6672
```



```
> table(predict=app_pred_y,truth=app_valid_dummy$y_appetency)
       truth
predict   no  yes
    no  2493   89
    yes 2208  210
> (2493+210)/5000
[1] 0.5406
```





```
> table(predict=ups_pred_y,truth=ups_valid_dummy$y_upselling)
       truth
predict   no  yes
    no  2410  408
    yes 1165 1017
> (2410+1017)/5000
[1] 0.6854
```

Criterion	Churn	Appetency	Upselling
AUC (%)	67.10	67.30	78.19
FPR (%)	27.25	46.97	32.59
FNR (%)	49.30	29.77	71.27
TPR (%) Sensitivity	**50.69**	70.23	**28.63**
TNR (%) Specificity	72.74	**53.03**	67.41
Accuracy	0.6672	0.5406	0.6854

Interpretation:

- Here as can be seen n>>p, so the over fitting of the data isn't going to be the case.
- While coding and setting up the threshold main emphasis has been given upon improving the true positive rate (i.e. sensitivity)
- AUC value for all three classes i.e. churn, appetency and upselling are decent.
- Also, TPR for appetency we are getting is highest of all 70.23% then for churn and upselling the values aren't that great in case of SVM.
- In order to improve, these rates we can set threshold a little bit less, but as it is obvious from the code that it has been already set to 0.2 which is way lenient value for the SVM.
- Hence, it can be concluded that in case of Churn and Upselling SVM is performing decent and worst respectively.
- Also, TNR (specificity) is better in case of churn and upselling while in case of appetency it isn't decent.

6. Logistic Regression[11]

Logistic regression is a popular classification technique. In this technique, the probability of a response to belong to a certain class is modelled using a linear combination of the prediction terms. If X_1, X_2… X_n be the prediction terms, then the logistic function is given as:

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_n X_n}}{1+e^{\beta_0+\beta_1 X_1+\cdots+\beta_n X_n}}$$

To fit the model, the likelihood function is maximized and values of the coefficient βs are predicted. Once, the coefficients are predicted by training the model, the model is able to predict responses for given Xs.

Code:

```
#Churn
        library(ISLR)
        training_data = churn_train
        testing_data = churn_valid[, -1]
        testing_y = churn_valid$y_churn
        logistic_model=glm(y_churn
        ~Var126+Var73+Var81+Var28+Var153+Var140+Var133+Var134+Var38,data=train
        ing_data, family = "binomial")
        logistic_probs = predict(logistic_model, testing_data, type =    "response")
        head(logistic_probs)
        logistic_pred_y = rep("no", length(testing_y))
        logistic_pred_y[logistic_probs > 0.3] = "yes"
        table(logistic_pred_y, testing_y)
        mean(logistic_pred_y != testing_y)
        roc_auc <- function(probabilities,dataset){
          #Command - roc_auc(probabilities,dataset)
          #probabilities are those obtained from predict function
          #dataset is the actual data (0s and 1s)
          library(ROCR)   #Install ROCR library before running
          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(logistic_probs,testing_y)
```

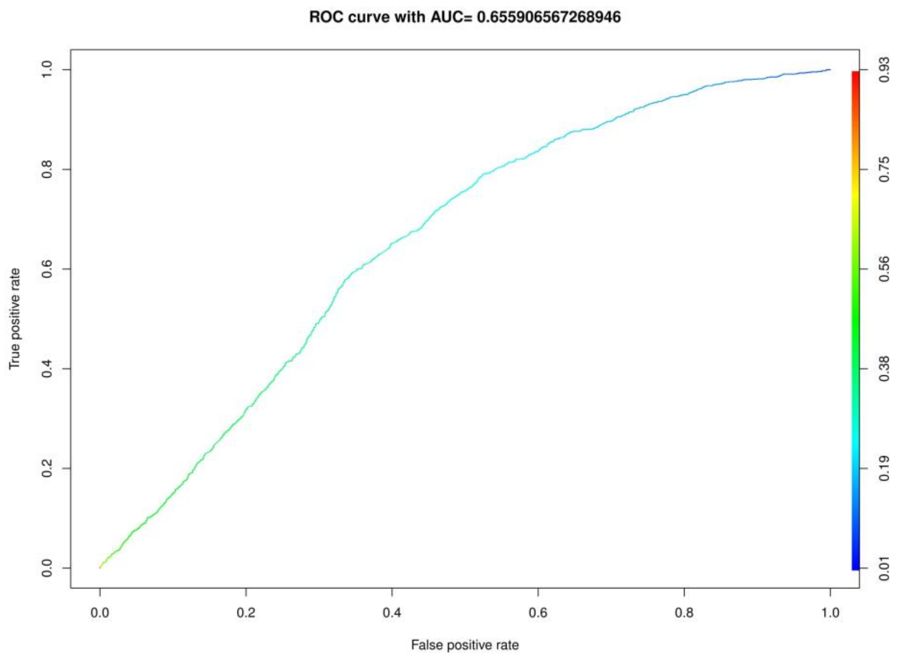
```
#Appetency
        library(ISLR)
        training_data = appetency_train
        testing_data = appetency_valid[, -1]
        testing_y = appetency_valid$y_appetency
        logistic_model=glm(y_appetency~Var126+Var113+Var218.1+Var81+Var28+Var153
        +Var133+Var73+Var38+Var125+Var134+Var6+Var13+Var119, data = training_data,
        family = "binomial")
        logistic_probs = predict(logistic_model, testing_data, type =    "response")
        head(logistic_probs)
        logistic_pred_y = rep("no", length(testing_y))
        logistic_pred_y[logistic_probs > 0.05] = "yes"
        table(logistic_pred_y, testing_y)
        mean(logistic_pred_y != testing_y)
        roc_auc <- function(probabilities,dataset){
          #Command - roc_auc(probabilities,dataset)
          #probabilities are those obtained from predict function
          #dataset is the actual data (0s and 1s)
          library(ROCR)   #Install ROCR library before running
          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(logistic_probs,testing_y)

#Upselling
        library(ISLR)
        training_data = upselling_train
        testing_data = upselling_valid[, -1]
        testing_y = upselling_valid$y_upselling
        logistic_model=glm(y_upselling~Var126+Var28+Var13+Var125+Var73+Var134+V
        ar38, data = training_data, family = "binomial")
        logistic_probs = predict(logistic_model, testing_data, type =    "response")
        head(logistic_probs)
        logistic_pred_y = rep("no", length(testing_y))
        logistic_pred_y[logistic_probs > 0.3] = "yes"
        table(logistic_pred_y, testing_y)
        mean(logistic_pred_y != testing_y)
        roc_auc <- function(probabilities,dataset){
```

```
  #Command - roc_auc(probabilities,dataset)
  #probabilities are those obtained from predict function
  #dataset is the actual data (0s and 1s)
  library(ROCR)   #Install ROCR library before running
  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(logistic_probs,testing_y)
```

Results:

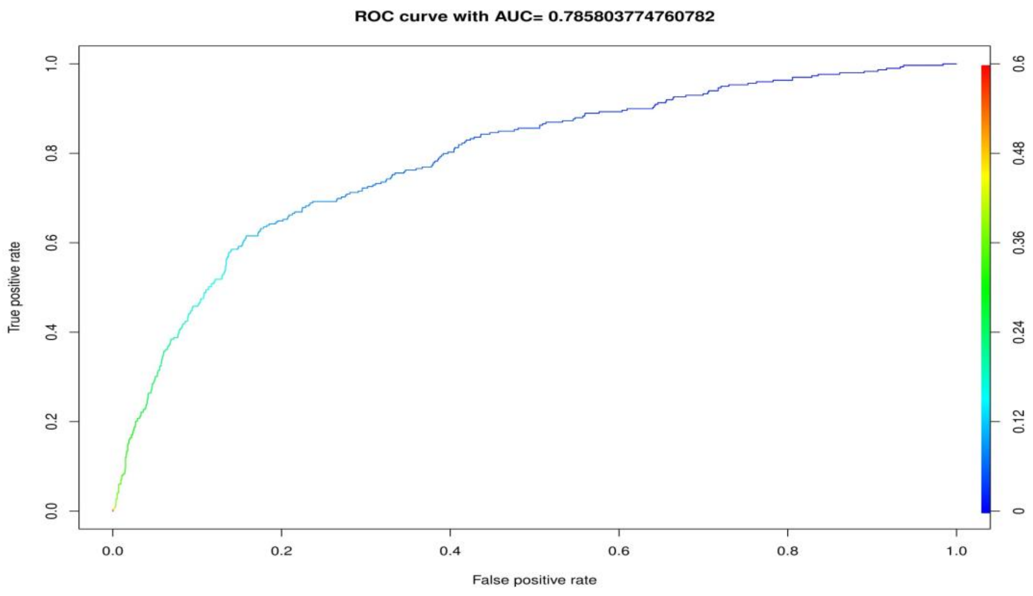
Churn:

```
table(logistic_pred_y, testing_y)
              testing_y
logistic_pred_y   no  yes
            no  2323  535
            yes 1310  832
```



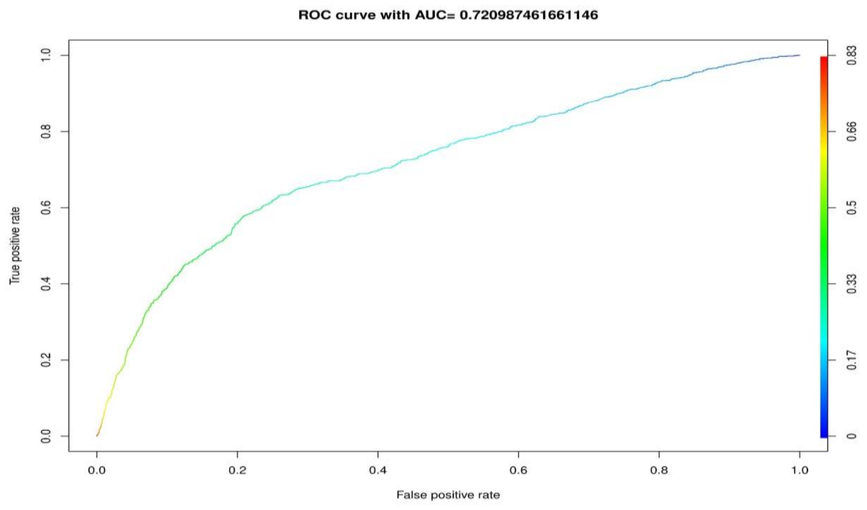
Appetency:

```
table(logistic_pred_y, testing_y)
               testing_y
logistic_pred_y   no  yes
            no  2845   60
            yes 1856  239
```



Upselling:

```
table(logistic_pred_y, testing_y)
               testing_y
logistic_pred_y   no  yes
            no  2635  523
            yes  940  902
```



Criterion	Churn	Appetency	Upselling
AUC	0.65	0.78	0.72
FPR	0.36	0.39	0.26
FNR	0.39	0.20	0.37
TPR	0.60	0.80	0.63
TNR	0.64	0.61	0.74
Accuracy	0.63	0.62	0.71

Interpretation:

- Since n>>p, over fitting should not be an issue.
- While selecting threshold value, emphasis has been given on TPR. Therefore, the threshold has been chosen as 0.2 to get good TPR. Further decrease in TPR results in a good amount of false positives. Hence, 0.2 is taken to be the most suitable threshold value. Same threshold has been observed in all three cases.
- AUC for all three levels are satisfactory.
- The method works best for appetency and worst for churn if TPR (or FNR) is considered. However, the method works best for upselling and worst for appetency in terms of TNR (or FPR).

7. KNN[11]

KNN is a non- parametric algorithm. Given a particular value of K, and an observation Xi, KNN algorithm identifies K points that are nearest to Xi represented by No. KNN then estimates the conditional probability for class j as a proportion of points in No. This is given by -

$$\Pr(Y = j | X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j).$$

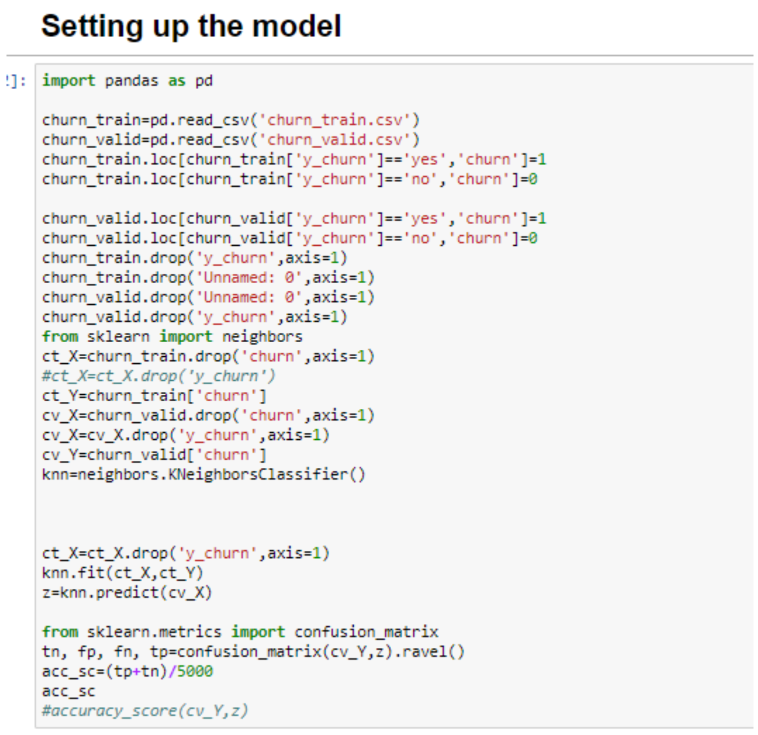
Figure 8

KNN applies the bayes rule and classifies the test observation Xi to the class with the highest probability.

The KNN model has been implemented in Python using the SciKit Learn library. Iterations have been done for various values of K and model accuracy and Positive Predictive Value has been computed for each. The reason for implementing KNN in python was time consumption as well as computational advantage.

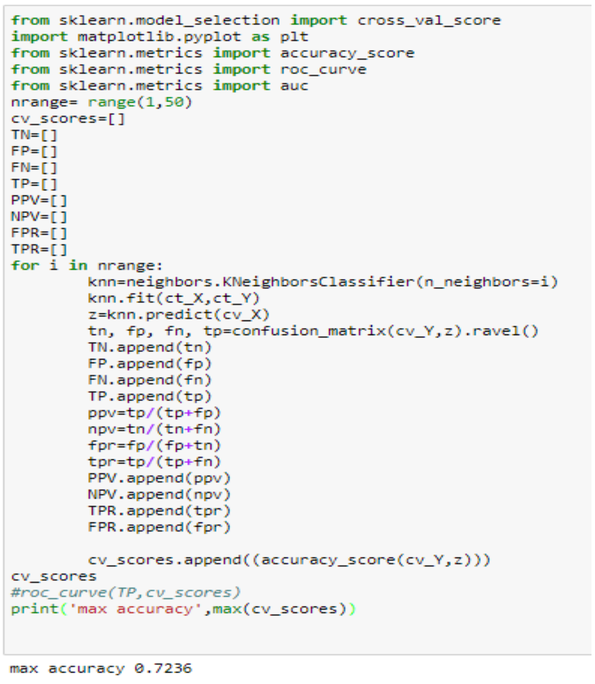
<u>Churn</u>

Code:Churn



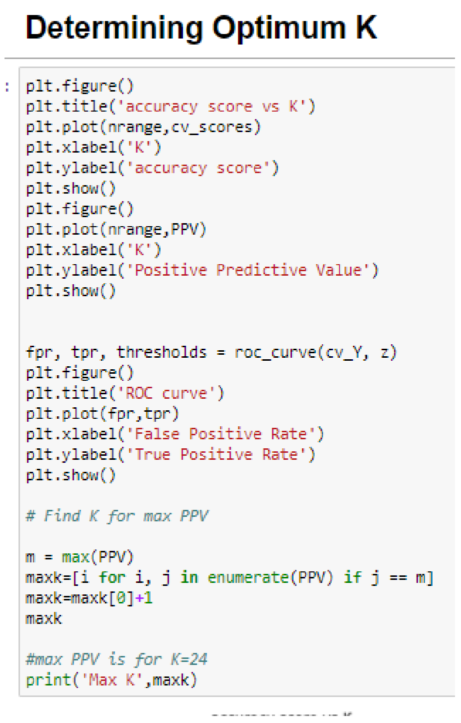
Setting up the model

```
import pandas as pd

churn_train=pd.read_csv('churn_train.csv')
churn_valid=pd.read_csv('churn_valid.csv')
churn_train.loc[churn_train['y_churn']=='yes','churn']=1
churn_train.loc[churn_train['y_churn']=='no','churn']=0

churn_valid.loc[churn_valid['y_churn']=='yes','churn']=1
churn_valid.loc[churn_valid['y_churn']=='no','churn']=0
churn_train.drop('y_churn',axis=1)
churn_train.drop('Unnamed: 0',axis=1)
churn_valid.drop('Unnamed: 0',axis=1)
churn_valid.drop('y_churn',axis=1)
from sklearn import neighbors
ct_X=churn_train.drop('churn',axis=1)
#ct_X=ct_X.drop('y_churn')
ct_Y=churn_train['churn']
cv_X=churn_valid.drop('churn',axis=1)
cv_X=cv_X.drop('y_churn',axis=1)
cv_Y=churn_valid['churn']
knn=neighbors.KNeighborsClassifier()



ct_X=ct_X.drop('y_churn',axis=1)
knn.fit(ct_X,ct_Y)
z=knn.predict(cv_X)

from sklearn.metrics import confusion_matrix
tn, fp, fn, tp=confusion_matrix(cv_Y,z).ravel()
acc_sc=(tp+tn)/5000
acc_sc
#accuracy_score(cv_Y,z)
```



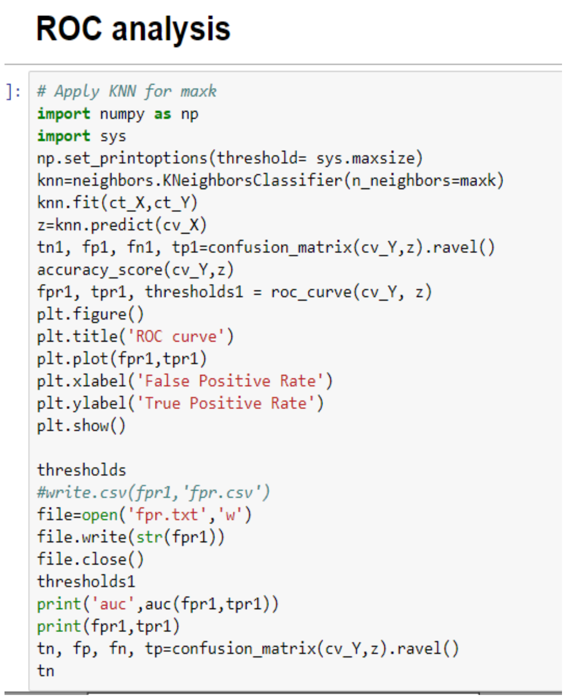
```
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
nrange= range(1,50)
cv_scores=[]
TN=[]
FP=[]
FN=[]
TP=[]
PPV=[]
NPV=[]
FPR=[]
TPR=[]
for i in nrange:
        knn=neighbors.KNeighborsClassifier(n_neighbors=i)
        knn.fit(ct_X,ct_Y)
        z=knn.predict(cv_X)
        tn, fp, fn, tp=confusion_matrix(cv_Y,z).ravel()
        TN.append(tn)
        FP.append(fp)
        FN.append(fn)
        TP.append(tp)
        ppv=tp/(tp+fp)
        npv=tn/(tn+fn)
        fpr=fp/(fp+tn)
        tpr=tp/(tp+fn)
        PPV.append(ppv)
        NPV.append(npv)
        TPR.append(tpr)
        FPR.append(fpr)

        cv_scores.append((accuracy_score(cv_Y,z)))
cv_scores
#roc_curve(TP,cv_scores)
print('max accuracy',max(cv_scores))
```

```
max accuracy 0.7236
```



Determining Optimum K

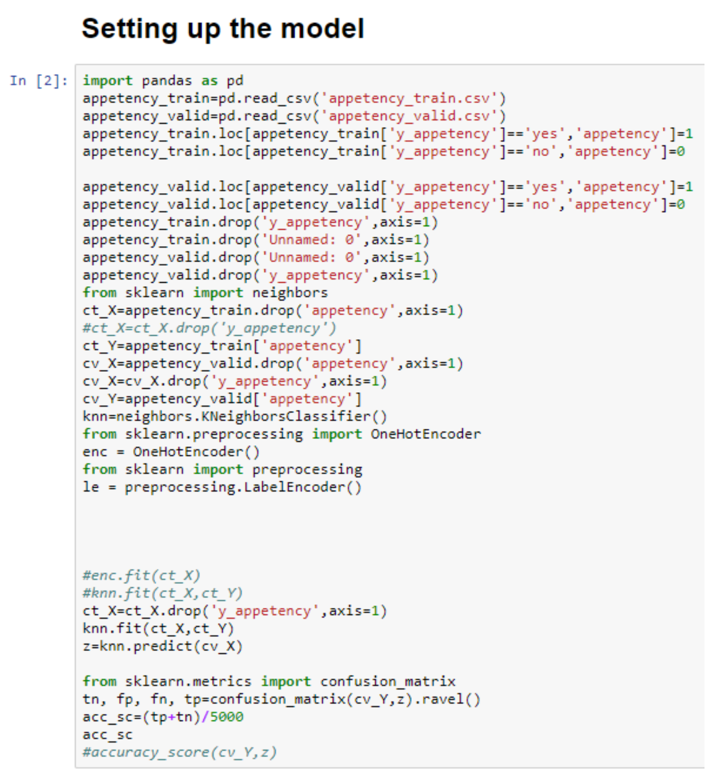
```
: plt.figure()
  plt.title('accuracy score vs K')
  plt.plot(nrange,cv_scores)
  plt.xlabel('K')
  plt.ylabel('accuracy score')
  plt.show()
  plt.figure()
  plt.plot(nrange,PPV)
  plt.xlabel('K')
  plt.ylabel('Positive Predictive Value')
  plt.show()


  fpr, tpr, thresholds = roc_curve(cv_Y, z)
  plt.figure()
  plt.title('ROC curve')
  plt.plot(fpr,tpr)
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.show()

  # Find K for max PPV

  m = max(PPV)
  maxk=[i for i, j in enumerate(PPV) if j == m]
  maxk=maxk[0]+1
  maxk

  #max PPV is for K=24
  print('Max K',maxk)
```



ROC analysis

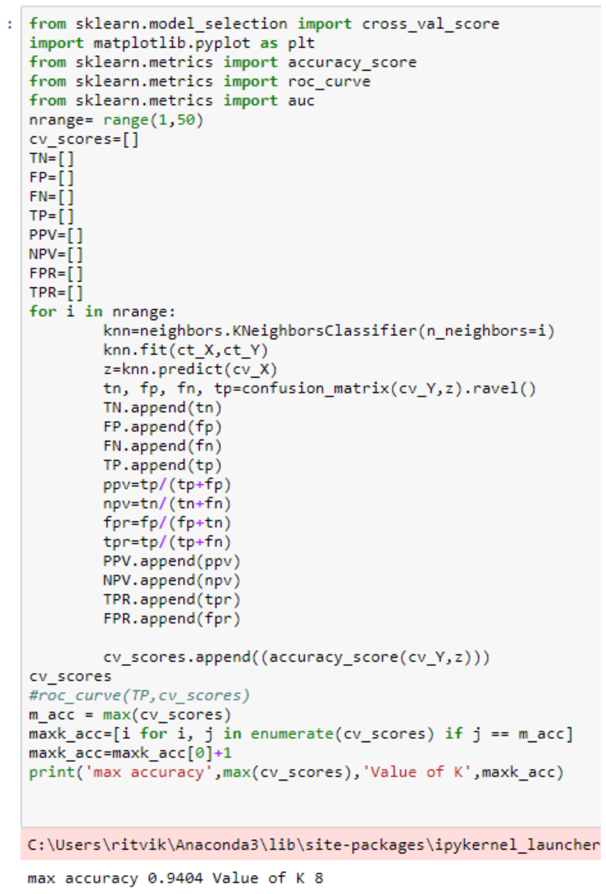
```
]: # Apply KNN for maxk
   import numpy as np
   import sys
   np.set_printoptions(threshold= sys.maxsize)
   knn=neighbors.KNeighborsClassifier(n_neighbors=maxk)
   knn.fit(ct_X,ct_Y)
   z=knn.predict(cv_X)
   tn1, fp1, fn1, tp1=confusion_matrix(cv_Y,z).ravel()
   accuracy_score(cv_Y,z)
   fpr1, tpr1, thresholds1 = roc_curve(cv_Y, z)
   plt.figure()
   plt.title('ROC curve')
   plt.plot(fpr1,tpr1)
   plt.xlabel('False Positive Rate')
   plt.ylabel('True Positive Rate')
   plt.show()

   thresholds
   #write.csv(fpr1,'fpr.csv')
   file=open('fpr.txt','w')
   file.write(str(fpr1))
   file.close()
   thresholds1
   print('auc',auc(fpr1,tpr1))
   print(fpr1,tpr1)
   tn, fp, fn, tp=confusion_matrix(cv_Y,z).ravel()
   tn
```

Code:Appetency



Setting up the model

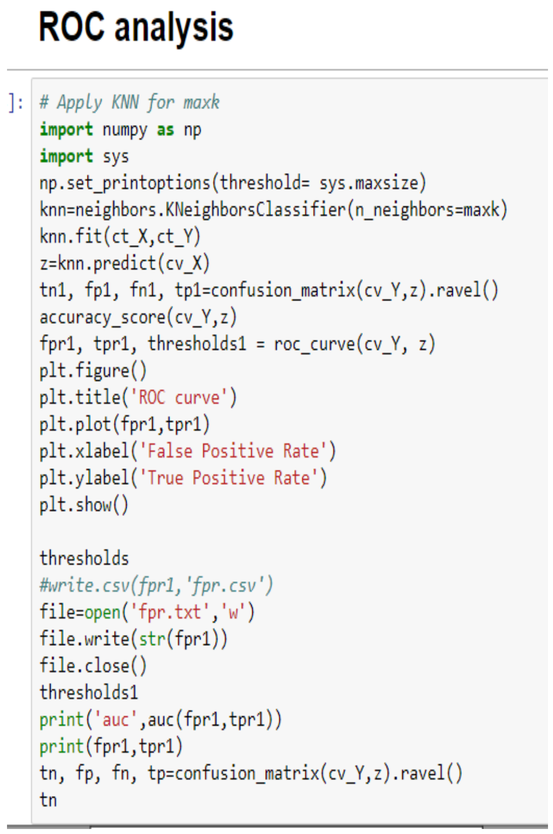
```
In [2]: import pandas as pd
        appetency_train=pd.read_csv('appetency_train.csv')
        appetency_valid=pd.read_csv('appetency_valid.csv')
        appetency_train.loc[appetency_train['y_appetency']=='yes','appetency']=1
        appetency_train.loc[appetency_train['y_appetency']=='no','appetency']=0

        appetency_valid.loc[appetency_valid['y_appetency']=='yes','appetency']=1
        appetency_valid.loc[appetency_valid['y_appetency']=='no','appetency']=0
        appetency_train.drop('y_appetency',axis=1)
        appetency_train.drop('Unnamed: 0',axis=1)
        appetency_valid.drop('Unnamed: 0',axis=1)
        appetency_valid.drop('y_appetency',axis=1)
        from sklearn import neighbors
        ct_X=appetency_train.drop('appetency',axis=1)
        #ct_X=ct_X.drop('y_appetency')
        ct_Y=appetency_train['appetency']
        cv_X=appetency_valid.drop('appetency',axis=1)
        cv_X=cv_X.drop('y_appetency',axis=1)
        cv_Y=appetency_valid['appetency']
        knn=neighbors.KNeighborsClassifier()
        from sklearn.preprocessing import OneHotEncoder
        enc = OneHotEncoder()
        from sklearn import preprocessing
        le = preprocessing.LabelEncoder()




        #enc.fit(ct_X)
        #knn.fit(ct_X,ct_Y)
        ct_X=ct_X.drop('y_appetency',axis=1)
        knn.fit(ct_X,ct_Y)
        z=knn.predict(cv_X)

        from sklearn.metrics import confusion_matrix
        tn, fp, fn, tp=confusion_matrix(cv_Y,z).ravel()
        acc_sc=(tp+tn)/5000
        acc_sc
        #accuracy_score(cv_Y,z)
```



```
: from sklearn.model_selection import cross_val_score
  import matplotlib.pyplot as plt
  from sklearn.metrics import accuracy_score
  from sklearn.metrics import roc_curve
  from sklearn.metrics import auc
  nrange= range(1,50)
  cv_scores=[]
  TN=[]
  FP=[]
  FN=[]
  TP=[]
  PPV=[]
  NPV=[]
  FPR=[]
  TPR=[]
  for i in nrange:
          knn=neighbors.KNeighborsClassifier(n_neighbors=i)
          knn.fit(ct_X,ct_Y)
          z=knn.predict(cv_X)
          tn, fp, fn, tp=confusion_matrix(cv_Y,z).ravel()
          TN.append(tn)
          FP.append(fp)
          FN.append(fn)
          TP.append(tp)
          ppv=tp/(tp+fp)
          npv=tn/(tn+fn)
          fpr=fp/(fp+tn)
          tpr=tp/(tp+fn)
          PPV.append(ppv)
          NPV.append(npv)
          TPR.append(tpr)
          FPR.append(fpr)

          cv_scores.append((accuracy_score(cv_Y,z)))
  cv_scores
  #roc_curve(TP,cv_scores)
  m_acc = max(cv_scores)
  maxk_acc=[i for i, j in enumerate(cv_scores) if j == m_acc]
  maxk_acc=maxk_acc[0]+1
  print('max accuracy',max(cv_scores),'Value of K',maxk_acc)
```

```
C:\Users\ritvik\Anaconda3\lib\site-packages\ipykernel_launcher
```

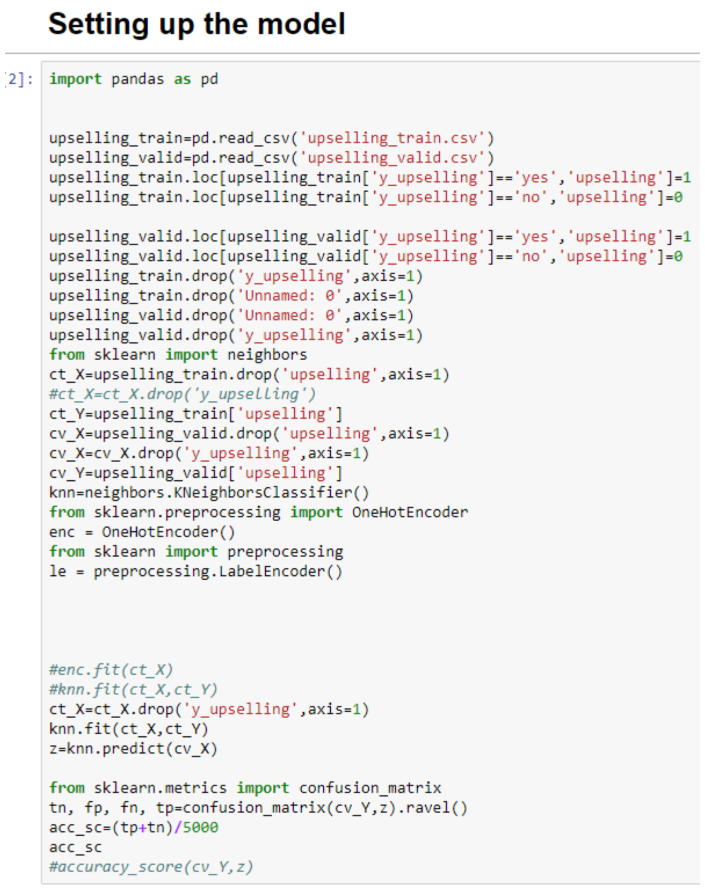
```
max accuracy 0.9404 Value of K 8
```



ROC analysis

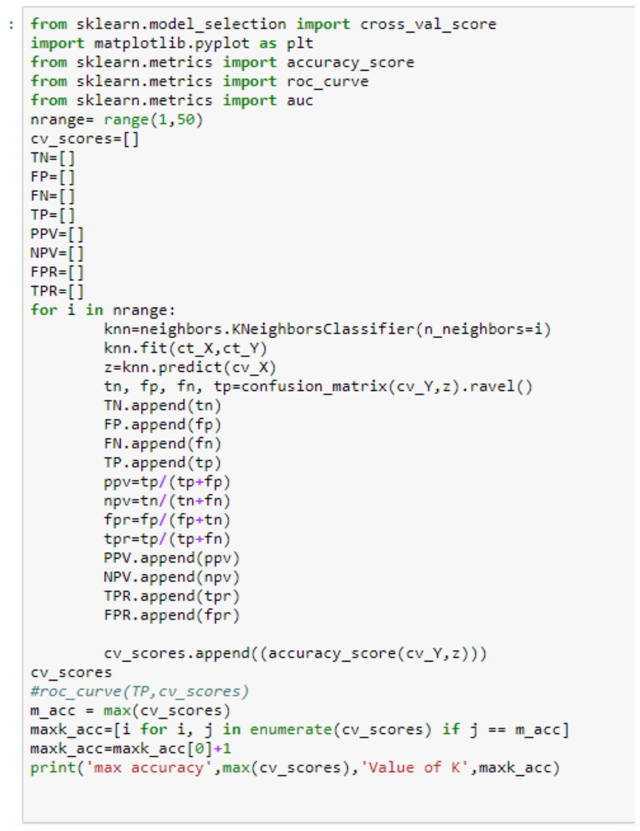
```
]: # Apply KNN for maxk
   import numpy as np
   import sys
   np.set_printoptions(threshold= sys.maxsize)
   knn=neighbors.KNeighborsClassifier(n_neighbors=maxk)
   knn.fit(ct_X,ct_Y)
   z=knn.predict(cv_X)
   tn1, fp1, fn1, tp1=confusion_matrix(cv_Y,z).ravel()
   accuracy_score(cv_Y,z)
   fpr1, tpr1, thresholds1 = roc_curve(cv_Y, z)
   plt.figure()
   plt.title('ROC curve')
   plt.plot(fpr1,tpr1)
   plt.xlabel('False Positive Rate')
   plt.ylabel('True Positive Rate')
   plt.show()

   thresholds
   #write.csv(fpr1,'fpr.csv')
   file=open('fpr.txt','w')
   file.write(str(fpr1))
   file.close()
   thresholds1
   print('auc',auc(fpr1,tpr1))
   print(fpr1,tpr1)
   tn, fp, fn, tp=confusion_matrix(cv_Y,z).ravel()
   tn
```

Code:Upselling

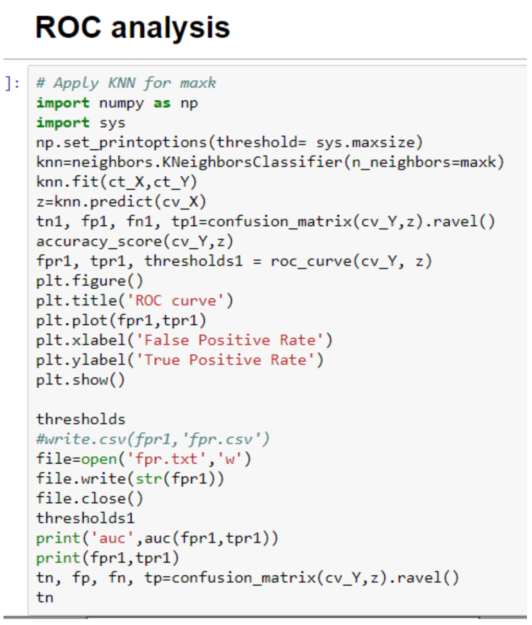


Setting up the model

```
[2]: import pandas as pd


upselling_train=pd.read_csv('upselling_train.csv')
upselling_valid=pd.read_csv('upselling_valid.csv')
upselling_train.loc[upselling_train['y_upselling']=='yes','upselling']=1
upselling_train.loc[upselling_train['y_upselling']=='no','upselling']=0

upselling_valid.loc[upselling_valid['y_upselling']=='yes','upselling']=1
upselling_valid.loc[upselling_valid['y_upselling']=='no','upselling']=0
upselling_train.drop('y_upselling',axis=1)
upselling_train.drop('Unnamed: 0',axis=1)
upselling_valid.drop('Unnamed: 0',axis=1)
upselling_valid.drop('y_upselling',axis=1)
from sklearn import neighbors
ct_X=upselling_train.drop('upselling',axis=1)
#ct_X=ct_X.drop('y_upselling')
ct_Y=upselling_train['upselling']
cv_X=upselling_valid.drop('upselling',axis=1)
cv_X=cv_X.drop('y_upselling',axis=1)
cv_Y=upselling_valid['upselling']
knn=neighbors.KNeighborsClassifier()
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
from sklearn import preprocessing
le = preprocessing.LabelEncoder()



#enc.fit(ct_X)
#knn.fit(ct_X,ct_Y)
ct_X=ct_X.drop('y_upselling',axis=1)
knn.fit(ct_X,ct_Y)
z=knn.predict(cv_X)

from sklearn.metrics import confusion_matrix
tn, fp, fn, tp=confusion_matrix(cv_Y,z).ravel()
acc_sc=(tp+tn)/5000
acc_sc
#accuracy_score(cv_Y,z)
```



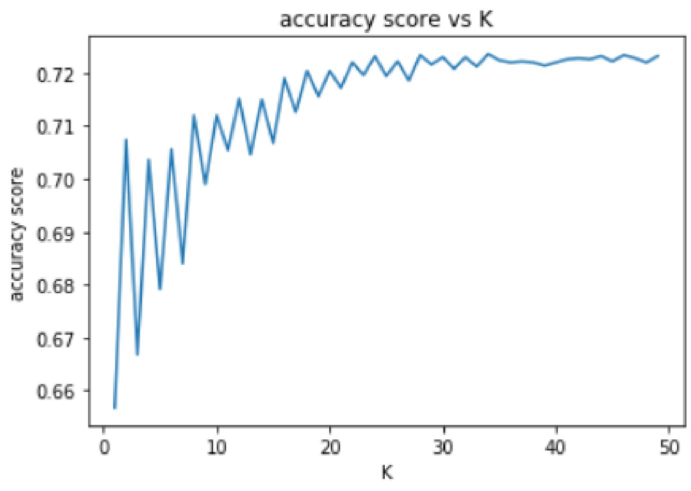
```
: from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
nrange= range(1,50)
cv_scores=[]
TN=[]
FP=[]
FN=[]
TP=[]
PPV=[]
NPV=[]
FPR=[]
TPR=[]
for i in nrange:
        knn=neighbors.KNeighborsClassifier(n_neighbors=i)
        knn.fit(ct_X,ct_Y)
        z=knn.predict(cv_X)
        tn, fp, fn, tp=confusion_matrix(cv_Y,z).ravel()
        TN.append(tn)
        FP.append(fp)
        FN.append(fn)
        TP.append(tp)
        ppv=tp/(tp+fp)
        npv=tn/(tn+fn)
        fpr=fp/(fp+tn)
        tpr=tp/(tp+fn)
        PPV.append(ppv)
        NPV.append(npv)
        TPR.append(tpr)
        FPR.append(fpr)

        cv_scores.append((accuracy_score(cv_Y,z)))
cv_scores
#roc_curve(TP,cv_scores)
m_acc = max(cv_scores)
maxk_acc=[i for i, j in enumerate(cv_scores) if j == m_acc]
maxk_acc=maxk_acc[0]+1
print('max accuracy',max(cv_scores),'Value of K',maxk_acc)
```

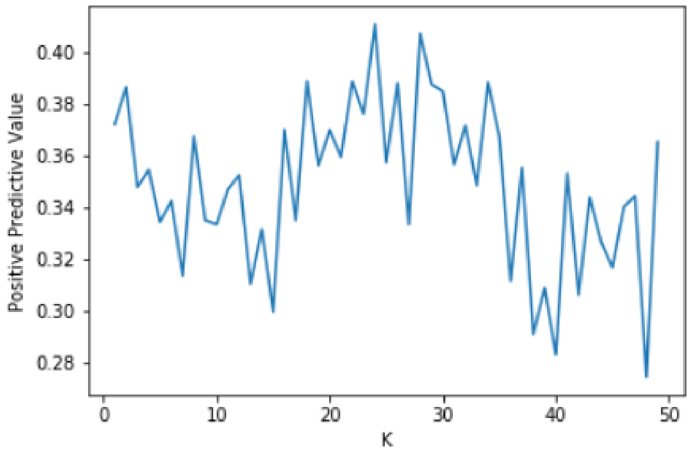


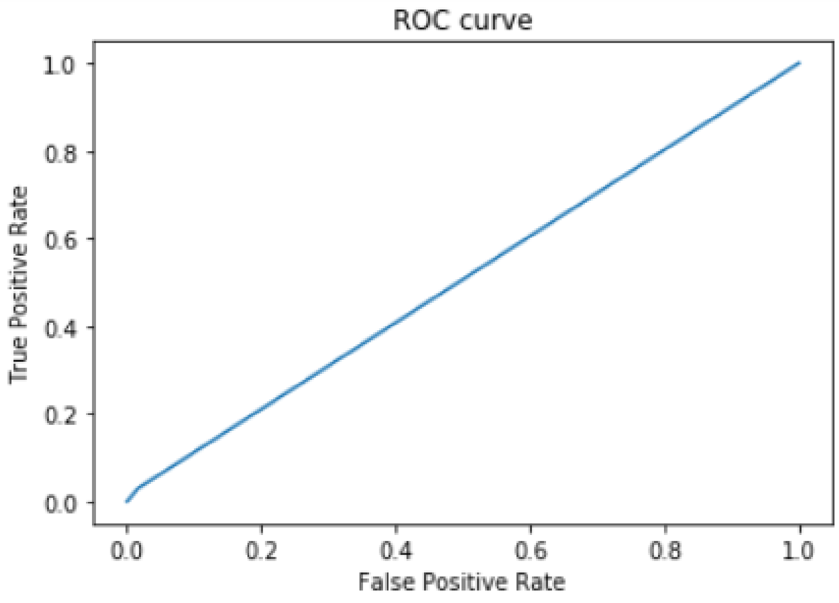
ROC analysis

```
]: # Apply KNN for maxk
import numpy as np
import sys
np.set_printoptions(threshold= sys.maxsize)
knn=neighbors.KNeighborsClassifier(n_neighbors=maxk)
knn.fit(ct_X,ct_Y)
z=knn.predict(cv_X)
tn1, fp1, fn1, tp1=confusion_matrix(cv_Y,z).ravel()
accuracy_score(cv_Y,z)
fpr1, tpr1, thresholds1 = roc_curve(cv_Y, z)
plt.figure()
plt.title('ROC curve')
plt.plot(fpr1,tpr1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()

thresholds
#write.csv(fpr1,'fpr.csv')
file=open('fpr.txt','w')
file.write(str(fpr1))
file.close()
thresholds1
print('auc',auc(fpr1,tpr1))
print(fpr1,tpr1)
tn, fp, fn, tp=confusion_matrix(cv_Y,z).ravel()
tn
```

Results: Churn







```
auc 0.506557684366
```

The Value of K that gives maximum Positive Predictive Value : 24

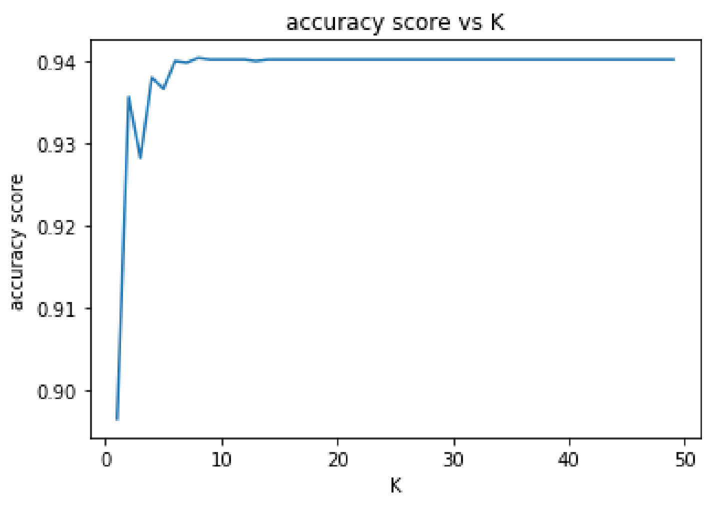
Max Positive Predictive Value : 0.410526315789

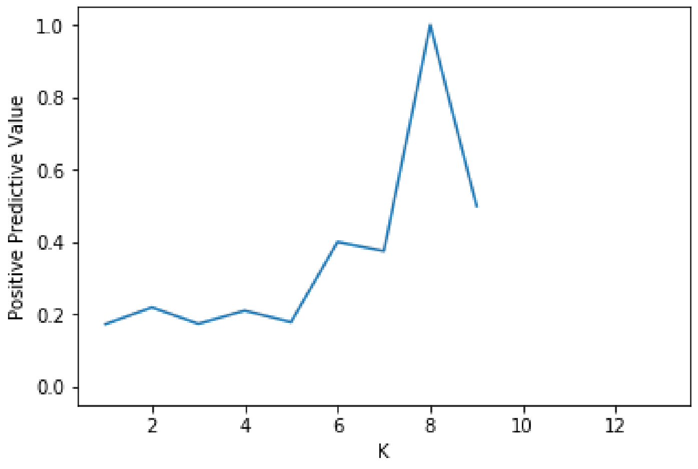
max accuracy=0.7236, Value of K=34

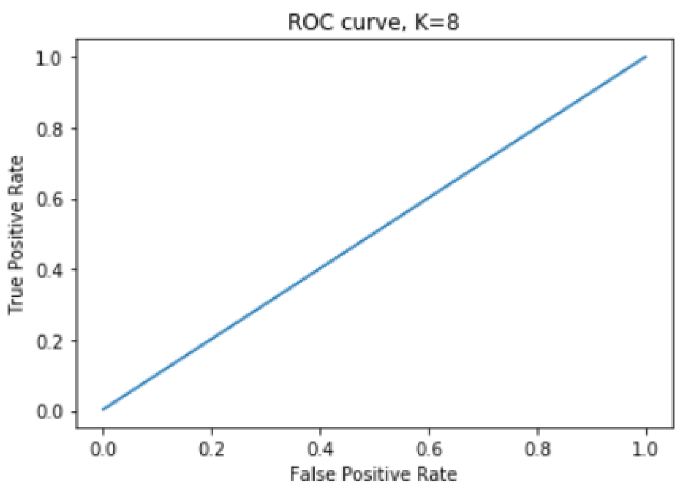
Churn Confusion Matrix
(K=24)

	truth	
predict	no	yes
no	3577	1328
yes	56	39

Results: Appetency





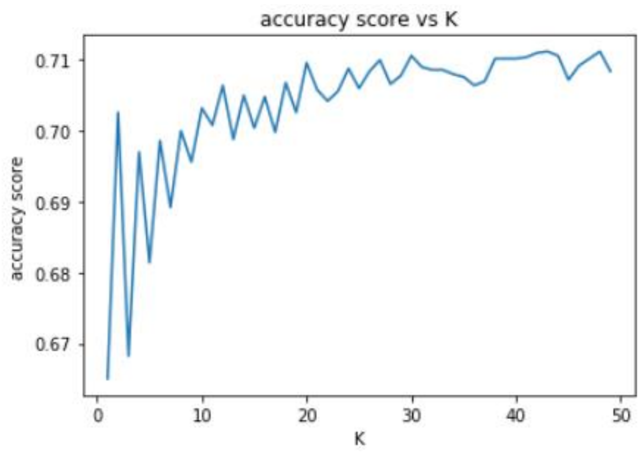


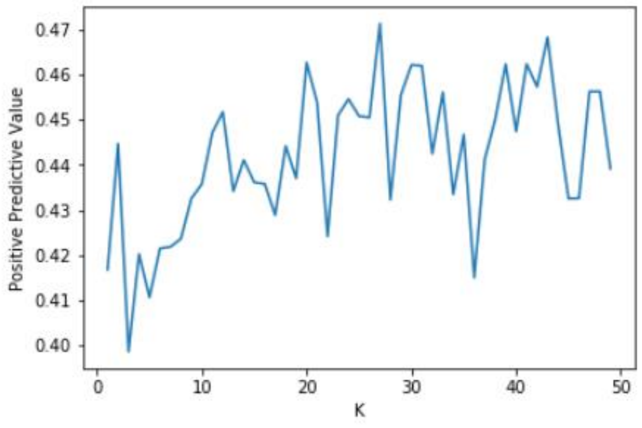
```
auc 0.501672240803
```

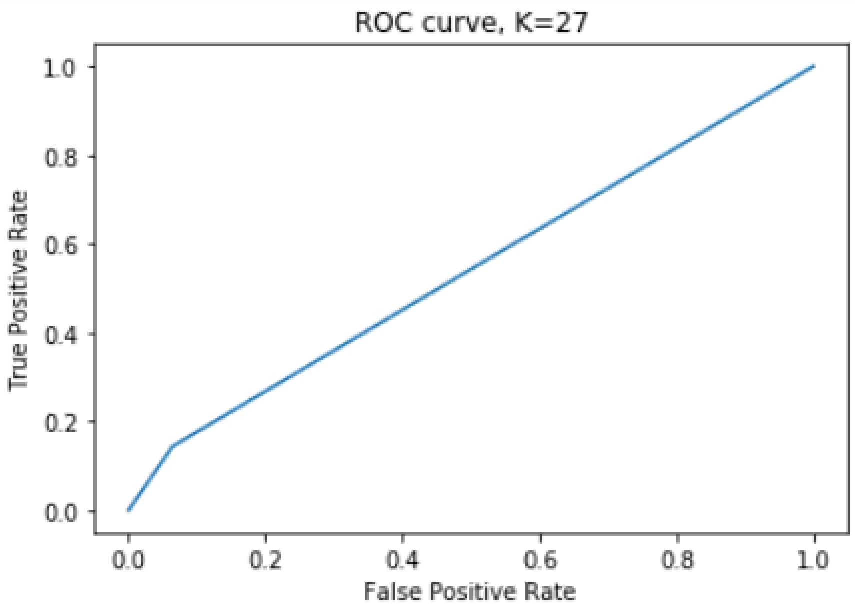
max accuracy 0.9404 Value of K 8
The Value of K that gives maximum Positive Predictive Value : 8 Max Positive Predictive Va
lue : 1.0

Appetency (Confusion Matrix) K=8		
	truth	
predict	no	yes
no	4701	298
yes	0	1

Results: Upselling







```
auc 0.539761992394
```

```
The Value of K that gives maximum Positive Predictive Value : 27 Max Positive Predictive V
alue : 0.471264367816

max accuracy 0.7112 Value of K 43
```

```
Upselling Confusion
Matrix K=27
              truth
predict       no       yes
no             3345     1220
yes             230      205
```

Criterion	Churn (K=24)	Appetency (K=8)	Upselling (K=27)
AUC (%)	50.65	50.16	53.97
FPR (%)	1.54	0	6.43
FNR (%)	97.15	99.66	85.61
TPR (%) Sensitivity	**2.85**	0.33	**14.39**
TNR (%) Specificity	98.46	**100.0**	93.57
Accuracy	0.7232	0.9404	0.71

Interpretation:

- While coding and setting up the threshold main emphasis has been given upon improving the positive predictive value for the model.
- AUC value for all therearound 50%, which suggests that the model is not good and needs to be improved upon.
- Also, TPR for upselling we are getting is highest of all 14.39% then for churn and appetency the values aren't that great in case of KNN.
- Reducing the threshold, will improve the true positives over the true negatives. However, The low value of AUC suggests that this alone is not sufficient. Fundamental changes in the model and data processing of categorical variables needs to be performed.

- In the case of many dimensions, inputs can commonly be "close" to many data points. This reduces the effectiveness of KNN, since the algorithm relies on a correlation between closeness and similarity.
- Dimension reduction must be performed and KNN must be implemented on only the important predictors
- More advanced methods of converting categorical data to numeric should be used, this may be able to improve the AUC values for the KNN algorithm

8. Artificial Neural Networks (ANN)

We have implemented neural network through R library H2O. We have used this library because of computational disadvantage of normal personal computer to implement neural network on large data set. H2O is an open source, in-memory, distributed, fast, and scalable machine learning and predictive analytics platform that allows you to build machine learning models on big data and provides easy productionalization of those models in an enterprise environment. The speed, quality, ease-of-use, and model-deployment for the various cutting edge Supervised and Unsupervised algorithms like Deep Learning, Tree Ensembles, and GLRM make H2O a highly sought-after API for big data, data science.

H2O Deep Learning models have many input parameters, many of which are only accessible via the expert mode. Here we mostly used default in many parameters. And concentrated on important parameters like hidden layer nodes, dropout ratio, and activation function.

For deep learning we used parallel computing to decrease the computational time. Using the code,

```
#parellal computing
Library(doParallel)
detectCores()
cl=makeCluster(3)
registerDoParallel(cl)
getDoParWorkers()
#stopCluster(cl)
```

First, we created model for Upselling

In order to train models with the H2O engine, I need to link the datasets to the H2O cluster first. There are many ways to do it. In this case, I linked a data frame (Upselling train) using the following code.

```
trh=as.h2o(upselling_train,destination_frame = "trh")
tth=as.h2o(upselling_valid,destination_frame="tth")
```

I trained deep learning for upselling through following code.

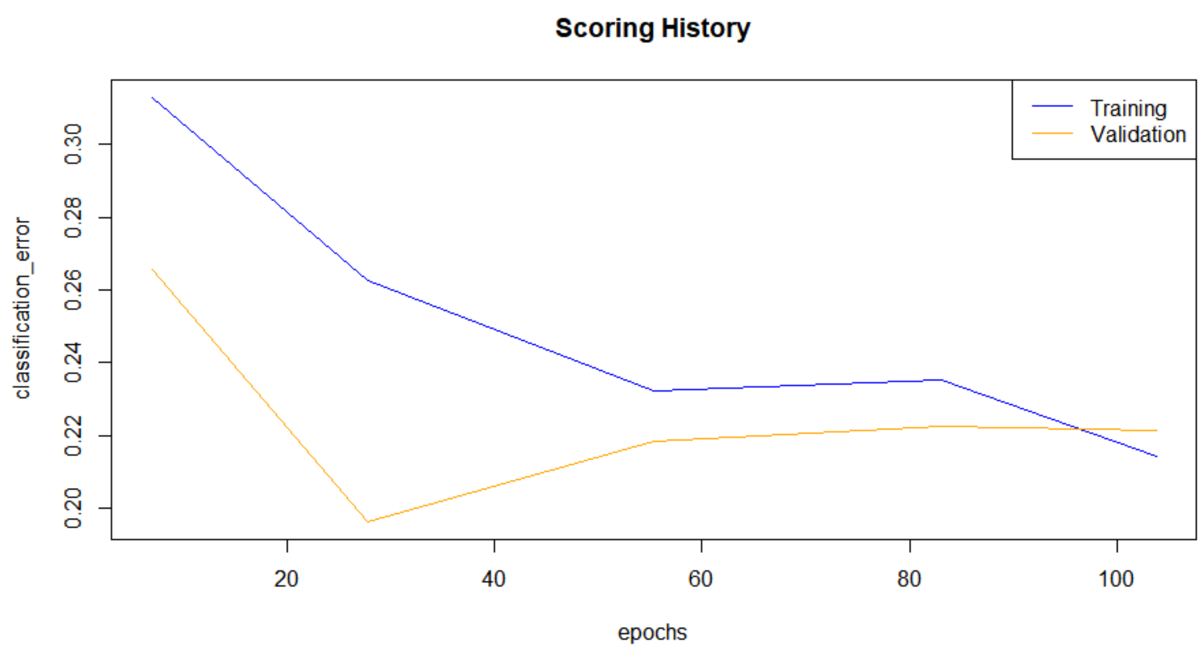
```
#training
```

```
model <-   h2o.deeplearning(x = datx,  # column numbers for predictors
                y = daty,   # column number for label
                training_frame = "trh", # data in H2O format
                activation = "RectifierWithDropout", # or 'Tanh'
                input_dropout_ratio = 0.2, # % of inputs dropout
                hidden_dropout_ratios = c(.2,.2,.2,.2), # % for nodes dropout
                balance_classes = TRUE,
                hidden = c(58,32,16,8),
                epochs = 100, # max. no. of epochs
                variable_importances=T,
                stopping_rounds=2,
                stopping_metric="misclassification", ## could be "MSE","logloss","r2"
                stopping_tolerance=0.001,nfolds = 5,seed = 1,fold_assignment = 'Modulo',
                validation_frame = 'tth',categorical_encoding ="OneHotInternal" )
```

For model definition we need to specify h2o about the predictor and response name. and give training data frame in h2o specific format.

We have used RectifierWithDropot(ReLU) activation function. Without activation function neural network will be simply linear model. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases. Without the differentiable nonlinear function, this would not be possible. the ReLU function is nonlinear, which means we can easily back propagate the errors and have multiple layers of neurons being activated by the ReLU function. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. If you look at the ReLU function if the input is negative it will convert it to zero and the neuron does not get activated. This means that at a time only a few neurons are activated making the network sparse making it efficient and easy for computation. Sigmoid function is most common choice for classification problem. But, here in our model it performed worse than ReLU. Sigmoid and Tanh functions are sometimes avoided due to the vanishing gradient problem

Increasing the number of epochs in a deep neural net may increase performance of the model, however, you have to be careful not to over fit your model to your training data. To automatically find the optimal number of epochs, you must use H2O’s early stopping functionality.



Early stopping (stop training before the specified number of epochs is completed to prevent over fitting) is enabled by default. If a validation frame is given, or if cross-validation is used (nfolds > 1), it will use validation error to determine the early stopping point. If just a training frame is given (and no CV), it will use the training set to perform early stopping. More on that below.

We have mentioned input dropout ratio to improve generalization. For better regularization we have also used dropout ratio in hidden layers. Here we have used 4 hidden layers, c(58,32,16,8). We thought instead of using single large hidden layer, if we use different layers than it will reduce model complexity and will reduce over fitting.

We have given stopping criterion to early stopping of model. Here we used 5 fold cross validation. And also provided valid data frame. In h2O model if lowest valid error is achieved in any model, than it is reported as final model.

Here we have asked model to do one hot encoding on categorical variables as h2o doesn’t support nested category classes.

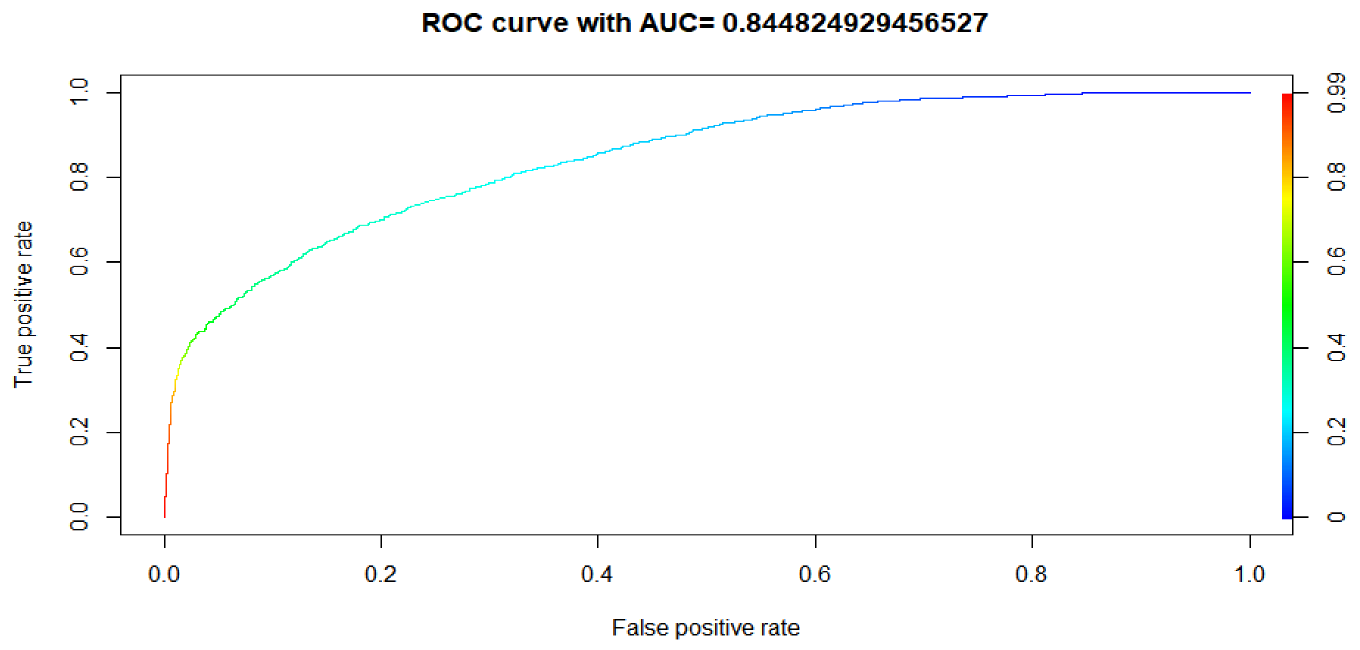
Here we predicted on the valid dataset.

We got this confusion table.

A confusion matrix of binary classification is a two by two table formed by counting of the number of the four outcomes of a binary classifier. We usually denote them as TP, FP, TN, and FN instead of "the number of true positives", and so on.

```
> table(df_yhat_test[,1],upselling_valid[,1])

       no  yes
  no  2938  453
  yes  637  972
> (2938+972)/5000
[1] 0.782
```

Here, we can play with the threshold to get better TPR. As we have mentioned earlier we wanted to get as many true result to help organization retrain customers.



The ROC plot is a model-wide evaluation measure that is based on two basic evaluation measures – specificity and sensitivity. Specificity is a performance measure of the whole negative part of a dataset, whereas sensitivity is a performance measure of the whole positive part. Another advantage of using the ROC plot is a single measure called the AUC (area under the ROC curve) score. As the name indicates, it is an area under the curve calculated in the ROC space.

We have plotted ROC curve for this model.

Second, We created model for appetency.

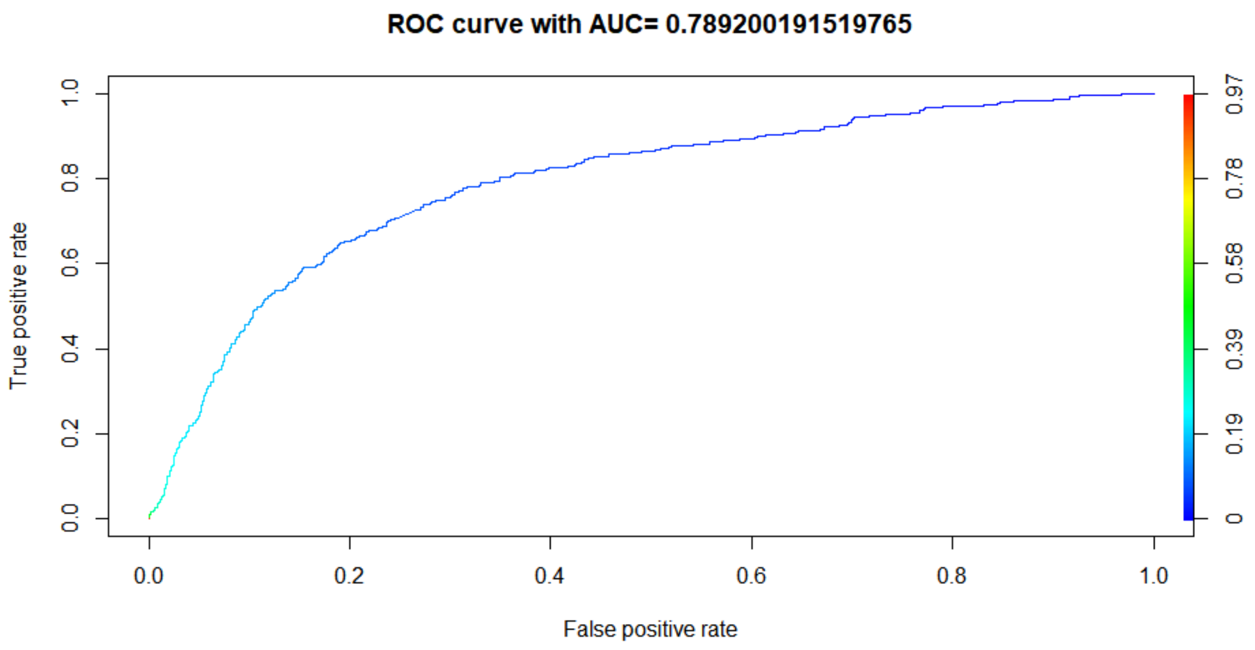
We have used a similar model as in upselling.

```
model_app <-   h2o.deeplearning(x = datx,  # column numbers for predictors
                               y = daty,   # column number for label
             training_frame = "trh",# data in H2O format
              activation = "RectifierWithDropout", # or 'Tanh'
               input_dropout_ratio = 0.2, # % of inputs dropout
               hidden_dropout_ratios = c(.2,.2,.2,.2), # % for nodes dropout
               balance_classes = TRUE,
             hidden = c(58,32,16,8), # three layers of 50 nodes
             epochs = 100, # max. no. of epochs
             variable_importances=T,
             stopping_rounds=2,
             stopping_metric="misclassification", ## could be "MSE","logloss","r2"
              stopping_tolerance=0.001,nfolds = 10,seed = 1,fold_assignment = 'Modulo',
              validation_frame = 'tth',categorical_encoding ="OneHotInternal",sparse=T)
```

Here we have iterated this model with number of epochs, activate function and different hidden layers.

```
> table(df_yhat_test[,1],appetency_valid[,1])

        no  yes
  no  4205  153
  yes  496  146
> (4205+146)/5000
[1] 0.8702
```

Here we got .8702 accuracy. This appetency valid data is very imbalanced. So, we cannot use this result as perfect.



We have plotted Roc Curve for this predicted result. And we have got AUC (Area under curve)=0.7892

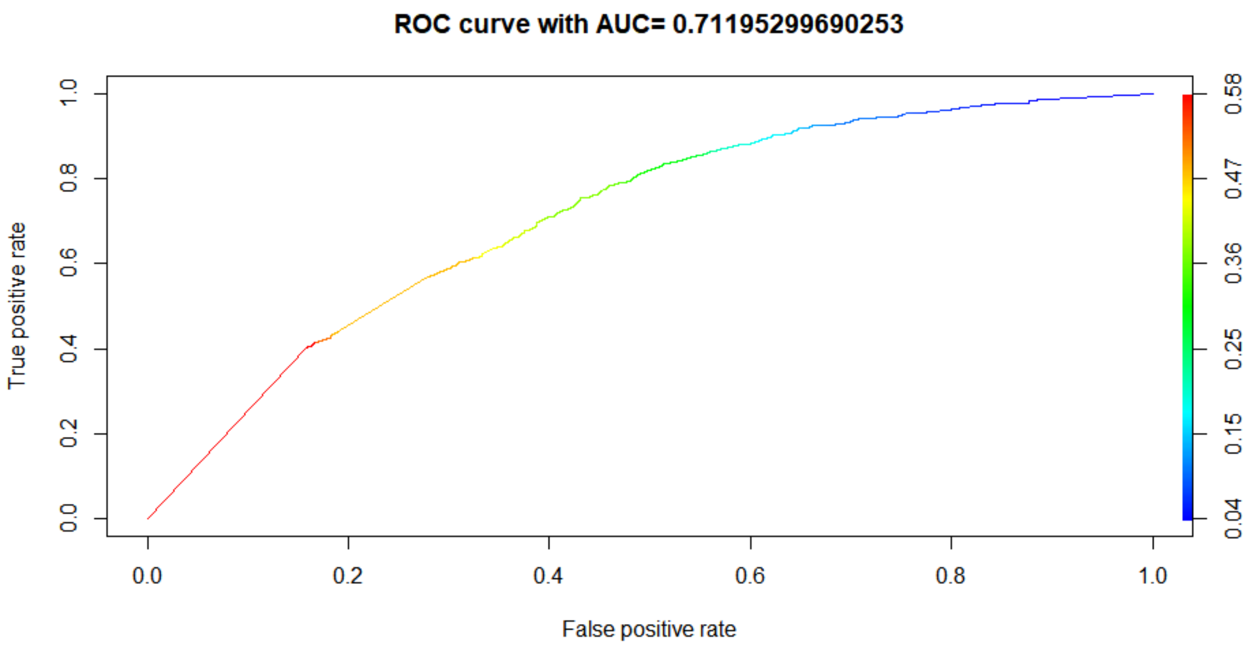
For churn we have used different activation function and different number of hidden nodes.

```
model <-   h2o.deeplearning(x = datx,  # column numbers for predictors
             y = daty,   # column number for label
             training_frame = "trh",# data in H2O format
             activation = "TanhWithDropout", # or 'Tanh'
             input_dropout_ratio = .1, # % of inputs dropout
             hidden_dropout_ratios = c(.2,.2,.1,.1,.1), # % for nodes dropout
             balance_classes = TRUE,
             hidden = c(20,20,20,20,20), # three layers of 50 nodes
             epochs = 50, # max. no. of epochs
             variable_importances=T,
             stopping_rounds=2,
             stopping_metric="misclassification", ## could be "MSE","logloss","r2"
             stopping_tolerance=0.001,nfolds = 5,seed = 1,fold_assignment = 'Modulo',
             validation_frame = 'tth')
```

And we got this confusion table when we predict on this model.

```
> table(df_yhat_test[,1],churn_valid[,1])

       no  yes
  no  2021  313
  yes 1612 1054
```

- (2021+1054)/5000
  0.615



We have plotted Roc Curve for this predicted result. And we have got AUC (Area under curve) =0.7119

Interpretation:

Criterion	Churn	Appetency	Upselling
AUC (%)	71.20	78.90	84.48
FPR (%)	44.89	07.11	09.21
FNR (%)	22.90	58.87	37.26
TPR (%) Sensitivity	77.10	41.13	62.73
TNR (%) Specificity	55.11	92.89	90.79
Accuracy	61.5	89.8	82.8

Interpretation of the table.

- As we have discussed earlier, we can change the value of accuracy of the model by changing the threshold values. So main factor to finding our best model is not accuracy but TPR or AUC. AUC will remain constant throughout our model prediction even if we change the model threshold.
- AUC of model upselling is higher in upper 80s. As we have used various hidden layer. TPR of upselling can also be increased by decreasing the threshold but at the cost of increasing false positive.
- AUC of model appetency is 78% this is still good. But the validation dataset is having imbalanced result, null model can also have higher accuracy but not AUC. Still by making model more complex, maybe we will be able to increase the AUC.
- AUC of model churn is 71.2 this is still high compare to other model. Churn considered to harder to predict based on this dataset. TPR of churn can be increased by decreasing the threshold.
- This NN model is fairly predicting good, but as we have discussed earlier by increasing the hidden layers as well as doing hyper parameter grid search will give us better result in the search of optimum model.
- We can also use deep net library in r to use multi label neural network to predict all three classes in single neural network.

9. Multi label Classification

We have used mlr library, which is a powerful and modularized toolbox for machine learning in R. The package offers a unified interface to more than a hundred learners from the areas classification, regression, cluster analysis and survival analysis. Furthermore, the package provides functions and tools that facilitate complex workflows such as hyper parameter and feature selection that can now also be applied to the multi label classification methods. In the following, we list the algorithm adaptation methods and problem transformation methods that are currently available in mlr.

The first task of multi label classification is to create a task. Learning tasks allow us to encapsulate the data set and specify information about a machine learning task. The first thing you have to do for multi label classification in mlr is to get your data in the right format. You need a data. Frame which consists of the features and a logical vector for each label which indicates if the label is present in the observation or not. This is important step as in real scenario classification problem have response as a factor but, here we are using logical argument as a response.

To, implement this we have used deplyr library to first map values to true and false.
But here still response is in factor, that need to be converted in Logical values.

```
#converting output to true false
fianl.with.y$appetency=mapvalues(fianl.with.y$appetency,c('yes','no'),to=c('TRUE','FALSE'))
fianl.with.y$churn=mapvalues(fianl.with.y$churn,c('yes','no'),to=c('TRUE','FALSE'))
fianl.with.y$upselling=mapvalues(fianl.with.y$upselling,c('yes','no'),to=c('TRUE','FALSE'))

#converting factor into logical argument
fianl.with.y$appetency=as.logical(fianl.with.y$appetency)
fianl.with.y$churn=as.logical(fianl.with.y$churn)
fianl.with.y$upselling=as.logical(fianl.with.y$upselling)
```

After that you can create a Multilabel Task like a normal Classif Task. Instead of one target name you have to specify a vector of targets which correspond to the names of logical variables in the data. Frame. In task you also specify the model structure.

```
labels = colnames(fianl.with.y)[59:61]
crp.task = makeMultilabelTask(id = "multi", data = fianl.with.y, target = labels)
```

In the constructor, you need to specify which learning method you want to use. This can be interpret as making a function that perform specific task such as classification or regression. Here we have specified as predict type= probability, means we want our prediction in probability. If we have not mentioned this than we'll only have final predicted logical values as output. Here we have made classification learner with trees.

```
binary.learner = makeLearner("classif.rpart", predict.type = "prob")
```

First, we are using problem transformation method.

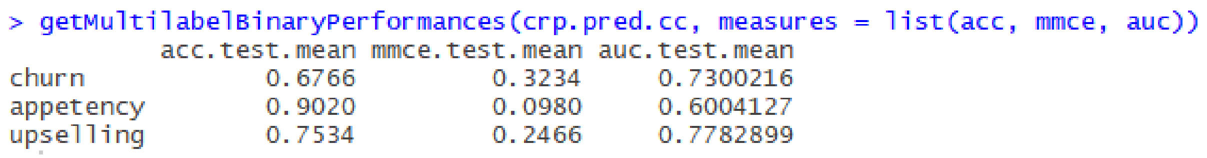
Here we are making multi-label classification learner using Chains Wrapper.

```
lrncc = makeMultilabelClassifierChainsWrapper(binary.learner)
```

You can train a model as usual with a multilabel learner and a multilabel task as input. And we predict on the validation model.

```
crp.mod.cc = train(lrncc, crp.task,subset=train.set)
crp.pred.cc = predict(crp.mod.cc, task = crp.task, subset=test.set)
performance(crp.pred.cc, measures = list(multilabel.f1, multilabel.acc))
getMultilabelBinaryPerformances(crp.pred.cc, measures = list(acc, mmce, auc))
```

As a performance matrices. We have used accuracy, mean misclassification error (mmce), area under curve. There are also number of other performance measures we can use to diagnose the result given by the multi label classification problem.



```
> getMultilabelBinaryPerformances(crp.pred.cc, measures = list(acc, mmce, auc))
           acc.test.mean mmce.test.mean auc.test.mean
churn             0.6766         0.3234     0.7300216
appetency         0.9020         0.0980     0.6004127
upselling         0.7534         0.2466     0.7782899
```

Here as we see multi label clasification model through chainswrapper learner it gives acuracy of .67 for churn that is ok in comparison of other models. As we have seen that churn Is not easy to predict with this dataset.

Appetency having highr accuracy but having imbalanced output for appetency we can see that auc is not that good for appetency for this model.

Upselling is also giving moderate result with 75% accuracy and simillar area under curve.

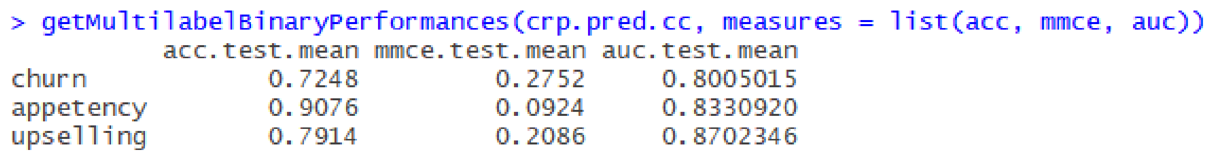
In mlr library for multilabel classification we cant get roc curve, but we can get individual auc curve in terms of results.

Second, we have used adaption method for our multilabel task.

First we have used randomforest through randomForestSRC package in r. this gives us the results in probabilty to get auc. To better comaprison between models.

```
set.seed(1)
lrn.rfsrc = makeLearner("multilabel.randomForestSRC", predict.type = "prob")
crp.mod.cc = train(lrn.rfsrc, crp.task,subset=train.set)
crp.pred.cc = predict(crp.mod.cc, task = crp.task, subset=test.set)
performance(crp.pred.cc, measures = list(multilabel.hamloss, multilabel.subset01,
multilabel.f1, multilabel.acc))
getMultilabelBinaryPerformances(crp.pred.cc, measures = list(acc, mmce, auc,tpr,fpr))
```

here implementation method is same but, we have used simple binary multilabel learner through randomforestSRC package's randomforest support.
As a result we get this table.



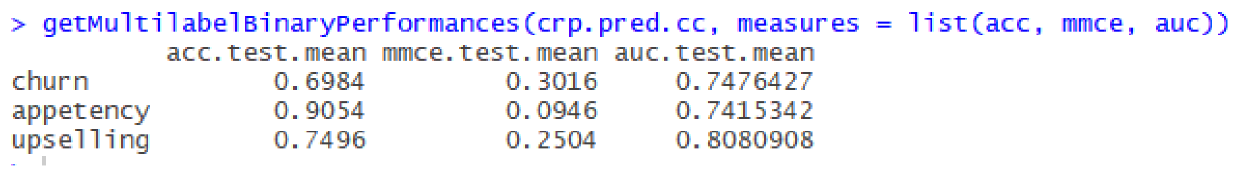
```
> getMultilabelBinaryPerformances(crp.pred.cc, measures = list(acc, mmce, auc))
           acc.test.mean mmce.test.mean auc.test.mean
churn             0.7248         0.2752     0.8005015
appetency         0.9076         0.0924     0.8330920
upselling         0.7914         0.2086     0.8702346
```

Here as we can see AUC is significantly improved over other algorithms. This algorithm is best predicting the churn model, that seems surprising as no other model have got this auc for solely churn model. This can be interpreted as multilabel random forest classification able to find correlation between the response.
This correlation resulted in higher auc as well as accuracy of model. Here we have used default parameter to train the randomforest. This randomforest parameters can be tuned to get further accurate result.

Finally, as a part of adaption method we have implemented svm.

```
lrn.br = makeLearner("classif.svm", predict.type = "prob")
lrn.br = makeMultilabelBinaryRelevanceWrapper(lrn.br)
crp.mod.cc = train(lrn.br, crp.task,subset=train.set)
crp.pred.cc = predict(crp.mod.cc, task = crp.task, subset=test.set)
performance(crp.pred.cc, measures = list(multilabel.f1, multilabel.acc))
getMultilabelBinaryPerformances(crp.pred.cc, measures = list(acc, mmce, auc))
```

Here we have used multilabel Binary relavance learner on svm method. So, this can be termed as combination of both transformation as well as adaption algorithm techniques.



```
> getMultilabelBinaryPerformances(crp.pred.cc, measures = list(acc, mmce, auc))
           acc.test.mean mmce.test.mean auc.test.mean
churn             0.6984         0.3016     0.7476427
appetency         0.9054         0.0946     0.7415342
upselling         0.7496         0.2504     0.8080908
```

As a result here we got a decent result compairing the other methods on multi label.
Here, we can got auc of .74 for churn this is still significant compairing the other single classification method.
So, from this we can conclude that for churn we should use multilable classification methods to build the model as correlation amongs response gives better result for predicting the outcome over here in churn.

Comparison

Techniques	Churn			Appetency			Upselling		
	AUC	TPR	Accuracy	AUC	TPR	Accuracy	AUC	TPR	Accuracy
TREE	68.49	24.94	**44.98**	77.01	85.67	84.02	84.12	51.50	63.28
RF	79.09	**89**	57	83.30	57	87	**88.41**	**91**	70
Bagging	78.90	45.39	**78.90**	82.97	**89.65**	82.97	88.38	62.26	**88.37**
Boosting	76.50	78.41	65.36	81.58	41	89.8	87.54	62.73	82.8
Logistics	65.59	60	63	78.58	80	62	72.10	63	71
SVM	67.10	50.69	66.72	67.30	70.23	**54.06**	78.19	28.63	**68.54**
KNN	**50.65**	**2.85**	72.32	**50.16**	**0.33**	**94.04**	**53.98**	**14.39**	71
ANN	71.20	77.10	61.5	78.90	41.13	89.8	84.48	62.73	82.8
Multi label	**80.05**	52.10	72.48	**83.30**	3.23	90.76	87.02	57.13	79.14

While making the comparison emphasis has been placed on the Area under curve (AUC), True positive rate (TPR) and accuracy (1-error rate) to come up with the best classification technique for individual responses. While comparing models with each other one should note that each classification has its own merits and demerits. No single classification method gives best result for all the problems,

Here as we have highlighted best values in column with the green ink and worse models with red ink.

Among all of these criterion, we believe AUC is going to be the most crucial factor. Because as it is evident from the AUC it is sensitivity vs. 1- specificity. And higher the AUC the higher will be the sensitivity (i.e. FPR) and higher specificity (FNR). And main advantage of using AUC as main criterion is that it doesn't depends on the threshold. Even if we change the value of threshold AUC will remain robust regardless of the threshold. While TPR and Accuracy could be inflated by meticulous use of threshold. And at the end of the comparison, for each class the best suitable technique is mentioned.

1)Churn

AUC is best for Multi label classification and worse for KNN.

TPR is best for Random forest and worse for KNN.

Accuracy is best for Bagging and worse for Tree.

For churn multi label classification gives the best AUC result (80.05%), while in case of TPR random forest works best (89%) and in case of accuracy bagging comes up with the best result of 78.90%. So, KNN perform mostly average. So, when there is mix of categorical and continuous variables in the model then we should be careful while using distance based techniques.

So, by going with our set criterion (AUC), Multi label classification techniques works best while KNN predicts the worst results. Also, new implemented classification technique ANN works moderately well in case of AUC, but doesn't beat the multi label classification. This could be justified as neural network is over fitting. While in the TPR, RF works best while KNN predicts very poor results. In case of accuracy (1-error rate), bagging produces the best results and TREE comes up with the worst result.

Best technique for Churn: Multi label classification

2) Appetency:

AUC is best for Multi label classification & Random Forest and worse for KNN.

TPR is best for Bagging and worse for KNN.

Accuracy is best for KNN and worse for SVM.

In case of appetency, RF and multi label classification comes with the best result (83.30). While for the TPR bagging and KNN comes up with the best and worst result respectively. On the other hand, in accuracy, KNN and SVM comes with the best and worst result respectively. This can be seen as because for appetency it has imbalanced dataset. This is because it results in the good accuracy for KNN. Any null predictor can get this accuracy if it predicts no.

Also, here going with our set criterion of AUC, ANN works moderately well but it can’t beat the RF and multi label classification. We recommend going for Random forest because of in multi label it depends upon the correlation between the responses. If it is not the case than our result for multi label can be highly skewed, that’s why best model for appetency to deploy is should be random forest.

Best technique for Appetency: Random Forest

3) Upselling

AUC is best for Random forest and worse for KNN.

TPR is best for Random forest and worse for KNN.

Accuracy is best for Bagging and worse for SVM.

In case of upselling, random forest comes with the best result of (88.41) in case of AUC, and KNN performs the worse in this term. And also, random forest excels in TPR with (91). This can be seen as robust result because of our objective of finding the customer that can purchase our upgrade. Even though accuracy of the random forest model is (70) still that satisfy our objective. Accuracy is highest by bagging and worse by SVM. This can be seen as SVM is also same technique based on the distance. Still it performs moderate in all the other techniques. So, it is still considered by good in comparison of the KNN.

Ann and multi label classification performs third best and second best. As we have seen earlier in multi label classification we have used random forest only. So, if we consider new technique ANN to comparison all other it performs fairly well. But, it might be over fitting the training data. So, in that case we should use some regularization technique for better results.

Best technique for Upselling: Random Forest

Conclusion

Here our goal was to use this dataset to provide organization insight into customer's intention. This intention may be of churn, buying upgrades or buying add-ons. As we describe earlier this may be used to retain customers or used to maximize the profit of the organization. For this objective, we concentrate only on True Positives. Because, even if we have the false positive prediction, that does not cost our objective at all. As companies are facing fierce competition, they want to retain their customer bases or increase the profit by providing offers. As a result, the company is making strong relationship with the customer. By making sure, that customer is not changing the company, we can only increase the customer base.

We have imputed dataset with basic technique of mean-mode. So, results that we got here may not be optimum. But, with this data set, we still are able to find out the true customer intentions. As we compared the results earlier churn was hardest to predict. This can be described in a number of ways. Maybe the dataset was not properly imputed, or it is possible in the real world that churn rate is random and varies from customer to customer. Although, multi label classification method was able to get AUC in the range of 80s for this data set. So, from this, we can conclude for the churn rate that, it is hard to predict churn rate individually, but churn rate can be predicted by using the correlation between the other responses.

For upselling we have got decent results individually for mostly all the models. Upselling can be easily predicted on this dataset. As we have seen that in the real world the intentions of customer buying upgrades can still be easily judged on general past behaviours.

Appetency has the imbalanced data set that is why the outcome of the model may not be easily implemented in real world. Appetency of the customer can be considered as the inclination of buying a particular item or service. This service is used by e-commerce giants to lure customer to buy the particular item for the certain discounts. For appetency, we can use the random forest to predict the rate.

We have implemented all the classification techniques taught as a part of the curriculum, for example; KNN, SVM, tree, Bagging, boosting, random forest and logistic regression. We have not used LDA and QDA. Because, in the discriminant analysis we assume the distribution of predictors in an individual class. But, categorical variables are not distributed normally. For this reason, discriminant techniques cannot be implemented for this model.

As a part of our course project, we learned two methods. Artificial neural network and multi label classification. This both techniques work fairly average in terms of AUC. As we compared results earlier Neural network is over fitting the results in a small number of iterations. This can be seen as the disadvantage while we implement the neural network for the small dataset. The neural network approach to the global minimum in short time.

Multi label classification is used when we have a correlation between the responses. This correlation can be used as an advantage in many cases where some responses are hard to predict individually. This was the case of churn rate. Multi label classification is found to give the best result for churn rate. This technique is also performed average for other responses.

For future scope, we can implement new imputation techniques with larger computer power. As well as we can also implement stacking. This technique is also part of ensemble methods. Ensemble methods also include bagging, random forest and boosting. In stacking, we combine the results of different models on different techniques and average the results in case of regression or the majority vote in case of classification. The main difference between random forest and stacking is that it does not limit itself to particular techniques. We can combine some different random forest models or various boosting models and also some deep learning models, and stack them together in one single group of models. This group of models can be used to predict the outcomes. These techniques are widely popular in competitions.

This project gave us hands-on experience with the real-world scenarios. As an analyst what decisions and what assumption one needs to make and what will be consequences of the decisions. Here for this project data set is in tabular form, this may not be the case in real world. In real-world analyst cannot rely on single technique, so as a part of the project, we also extended our base and learned new techniques.

To conclude, we present three models to predict the churn, upselling and appetency rate. The model we presented are robust, but we can further improve these techniques with ingenious imputation methods and ensemble learning algorithm stacking.

Reference

[1] https://www.salesforce.com/blog/2013/01/what-is-crm-your-business-nerve-center.html

[2] https://it.toolbox.com/blogs/inside-crm/cross-selling-and-upselling-with-crm-012313

[3] https://www.neopost.com/en/newsroom/blog/crm-key-facts-and-figures

[4] http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/deep-learning.html

[5] https://www.learnopencv.com/understanding-feedforward-neural-networks/

[6] https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7

[7] http://www.deeplearningbook.org/

[8] https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/

[9] https://en.wikipedia.org/wiki/Multi-label_classification

[10] https://www.r-bloggers.com/multilabel-classification-with-mlr/

[11] James, G., Witten, D., Hastie, T. and Tibshirani, R., 2013. *An introduction to statistical learning* (Vol. 112). New York: springer.

[12] https://en.wikipedia.org/wiki/Boosting_(machine_learning)