KNN

K-nearest neighbor algorithm (k-NN) is a non-parametric approach used for regression and classification. For both situations, the input consists of the most nearest or imminent samples of training in the feature space k. The real output usually depends on whether k-NN is used for classification based problem or regression based problem. Output is class membership of the k-NN classification.. An object is categorized by a majority vote of its neighbors, assigning the object to the most common class of its nearest k neighbors (k is a positive integer, usually small). If k = 1, the object is allocated to the closest single neighbor's class. In the regression k-NN the output is the value of the object 's property. The value obtained is the average/mean of all the values of the K_nearest_neighbors.

The K-nearest neighbor method is the simplest classification method that classifies based on distance measures.The plus side of going ahead with KNN approach is that it does not require configuration, and is strictly data-based, not based by model. Therefore, no assumptions are needed for this method.

There are some advantages and disadvantages to KNN.

Advantages include - It is intuitive and straightforward as we are working with the distance parameter. No assumptions are required about the dataset. It can be potent with a large training dataset

Disadvantages include - The required size of training dataset increases exponentially with the number of predictors

An extensive training dataset takes a long time to find distances to all the neighbors and then identify the nearest one(s).

Training the Knn model:

For training the model using the KNN algorithm, we employ the Caret package – train method. Before train() method, we will use train control() method. It controls controls the algorithmic complexities of the train() method.

We are setting three parameters of trainControl() method. The “method” parameter contains the details about the resampling method. We can set “method” with many other values like “boot”, “boot632”, “cv”, “repeatedcv”, “LOOCV”, “LGOCV” etc. For this project, we used repeatedcv, which is repeated cross-validation. The “number” parameter contains the number of resampling iterations. The “repeats ” parameter holds the entire sets of folds to compute our repeated cross-validation or "repeatedcvcv." We are using number =10 and repeats =3. This trainControl() method will return a list. We will pass this on our train() method before training our KNN classifier, set.seed(). For the training KNN classifier, the train() method should be passed with the “method” parameter as “knn”. We are passing our target variable y. Y ~. denotes the

formula for using all the attributes in our classifier and y as our output/target variable. The "trControl" parameter is passed with results from our trianControl() method. The "preProcess" parameter is for preprocessing our training data.
As discussed earlier for our data, preprocessing is a mandatory task. We are passing two values in our "preProcess" parameter "center" & "scale". These two helps to centering and scaling the data. After preprocessing, these convert our training data with mean value as approximately "0" and standard deviation as "1". The "tuneLength" parameter holds an integer value. This parameter is used for tuning our KNN algorithm.

K_NN model result:
We implemented the k-NN algorithm in 2 cases which included
Case 1: Considering all variables in the Training_Set and Testing_Set.
Case 2: Choosing the significant variables in Training_Set and Testing_Set.
(Reference - A specific set of variables that we found more prominent or impactful after performing the EDA, and we dropped the following variables: PhoneServices, MultipleLines, DeviceProtection, StreamingMovies, StreamingTV.yes, PaperlessBilling, PaymentMethod-ElectronicCheck).

Result Analysis
Case 1: Considering All Variables:

Cut Off	Accuracy	Error	Sensitivity	Specificity
0.3	0.7204	0.2796	0.8067	0.6882
0.4	0.7640	0.2360	0.7258	0.7783
0.5	0.7937	0.2063	0.5905	0.8697

Cutoff 0.3	**Predicted**		**Cutoff 0.4**	**Predicted**		**Cutoff 0.5**	**Predicted**	
Reference	**0**	**1**	**Reference**	**0**	**1**	**Reference**	**0**	**1**
0	**1046**	**474**	**0**	**1183**	**337**	**0**	**1322**	**198**
1	**110**	**459**	**1**	**156**	**413**	**1**	**233**	**336**

Case 2 : Considering a Particular Set of important Variables

Cut Off	Accuracy	Error	Sensitivity	Specificity

0.3	0.7276	0.2724	0.8243	0.6914
0.4	0.7707	0.2293	0.6995	0.7974
0.5	0.7946	0.2054	0.5712	0.8783

Cutoff 0.3	**Predicted**		**Cutoff 0.4**	**Predicted**		**Cutoff 0.5**	**Predicted**	
Reference	**0**	**1**	**Reference**	**0**	**1**	**Reference**	**0**	**1**
0	**1051**	**469**	**0**	**1212**	**308**	**0**	**1335**	**185**
1	**100**	**469**	**1**	**171**	**398**	**1**	**244**	**325**

As we know, k-NN works well with a small number of input variables; thus, this algorithm can benefit from feature selection through EDA that reduces the input feature space's dimensionality. We implemented the algorithm on a specific set of variables that we found more prominent or impactful after performing the EDA. Based on these specific sets of variables, our model obtained the Accuracy and Kappa metrics results for different k values, and our training model chooses No of neighbors = 35 as its final value for high accuracy rate. We can see the variation in Accuracy w.r.t K value by plotting these in a graph.

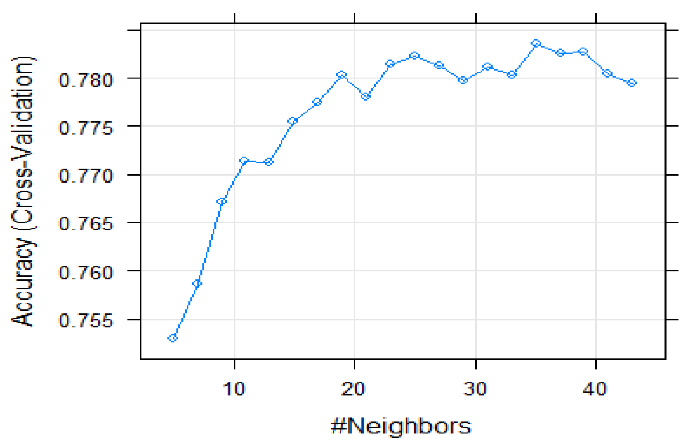

Figure iPlotting yields Number of Neighbours Vs Accuracy (based on repeated cross validation)

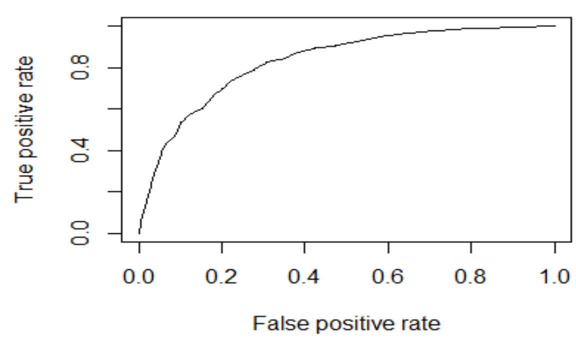

Figure 2: ROC Curve

Summary

Using confusion matrix analysis for case 2, **for cut-off = 0.3, we found that k-NN works best, or we can say the best optimum output is obtained with 72.76% Accuracy and 82.43% Sensitivity.** Even though a cut off 0.4 gives 77% accuracy as we need high sensitivity, we chose a Cut Off 0.3 with 72.76 accuracy with 82.43% sensitivity.

KNN_R_Code

```
rm(list=ls()); gc()
usePackage <- function(p) {
  if (!is.element(p, installed.packages()[,1]))
    install.packages(p, dep = TRUE)
  require(p, character.only = TRUE)
}
usePackage('ggplot2')
# For Accuracy Calculation
usePackage('caret')
usePackage('cowplot')
usePackage('e1071')
usePackage('rpart.plot')
usePackage('rattle')
usePackage('randomForest')
usePackage('caTools')
usePackage('descr')
usePackage('rpart')
usePackage('RColorBrewer')
usePackage('knitr')
usePackage('tidyr')
usePackage('plyr')
usePackage('dplyr')
usePackage('ROCR')
usePackage('corrplot')
usePackage('gridExtra')
usePackage('ggthemes')
usePackage('stringr')
usePackage('ggcorrplot')

#Data Preprocessing
#Data Extraction
churn <- read.csv(file="C:/Users/Gayathri/Desktop/574/Project/WA_Fn-UseC_-Telco-Customer-
Churn.csv", header=TRUE)
str(churn) #structure of our data frame
dim(churn) # To check dimension of the data-set
#summary before cleaning
summary(churn) #statistical information about the data-set
#Removing the missing values
sapply(churn, function(x) sum(is.na(x)))
```

```
churn <- churn[complete.cases(churn), ]
#Changing "No internet service" to "No" for six columns:
cols_recode1 <- c(10:15)
for(i in 1:ncol(churn[,cols_recode1])) {
  churn[,cols_recode1][,i] <- as.factor(mapvalues
                                        (churn[,cols_recode1][,i], from =c("No internet service"),to=c("No")))
}
#Changing "No phone service" to "No" for column "MultipleLines"
churn$MultipleLines <- as.factor(mapvalues(churn$MultipleLines,
                                           from=c("No phone service"),
                                           to=c("No")))
min(churn$tenure) #1
max(churn$tenure) #72
#grouping tenure into five tenure groups:
group_tenure <- function(tenure){
  if (tenure >= 0 & tenure <= 12){
    return('0-12 Month')
  }else if(tenure > 12 & tenure <= 24){
    return('12-24 Month')
  }else if (tenure > 24 & tenure <= 48){
    return('24-48 Month')
  }else if (tenure > 48 & tenure <=60){
    return('48-60 Month')
  }else if (tenure > 60){
    return('> 60 Month')
  }}
churn$tenure_group <- sapply(churn$tenure,group_tenure)
churn$tenure_group <- as.factor(churn$tenure_group)

#Change the values in column "SeniorCitizen" from 0 or 1 to "No" or "Yes".
churn$SeniorCitizen <- as.factor(mapvalues(churn$SeniorCitizen,
                                           from=c("0","1"),
                                           to=c("No", "Yes")))
#Removing the columns we do not need for the analysis
churn$customerID <- NULL
churn$tenure <- NULL
--------------------------------------------------------------------------------------------------------------------------
#################### KNN Algorithm Implementation
##################################################

str(churn)
churn$churn_number <- 0
churn$churn_number[churn$Churn == 'Yes'] <- 1
churn$Churn <- NULL
```

```
#covert categorical variables to factor
churn$gender <- as.factor(churn$gender)
churn$SeniorCitizen <- as.factor(churn$SeniorCitizen)
churn$Partner <- as.factor(churn$Partner)
churn$Dependents <- as.factor(churn$Dependents)
churn$PhoneService <- as.factor(churn$PhoneService)
churn$MultipleLines <- as.factor(churn$MultipleLines)
churn$InternetService <- as.factor(churn$InternetService)
churn$OnlineSecurity <- as.factor(churn$OnlineSecurity)
churn$OnlineBackup <- as.factor(churn$OnlineBackup)
churn$DeviceProtection <- as.factor(churn$DeviceProtection)
churn$TechSupport <- as.factor(churn$TechSupport)
churn$StreamingTV <- as.factor(churn$StreamingTV)
churn$StreamingMovies <- as.factor(churn$StreamingMovies)
churn$Contract <- as.factor(churn$Contract)
churn$PaperlessBilling <- as.factor(churn$PaperlessBilling)
churn$PaymentMethod <- as.factor(churn$PaymentMethod)

#Creating Dummies
trainDummy <- dummyVars("~.", data = churn, fullRank = F)
train <- as.data.frame(predict(trainDummy,churn))
colnames(train)
#Coverting target variable to a factor
train$churn_number <- as.factor(ifelse(train$churn_number == 1,'yes','no'))
#split the data into training set and testing set
set.seed(123)
split <- sample(2,nrow(train), replace = T,prob = c(0.70,0.30))
trainDF<- train[split ==1,]
testDF <- train[split ==2,]
dim(trainDF)
dim(testDF)
colnames(trainDF)
colnames(testDF)
#check whether any NA value exists or not
anyNA(churn)
anyNA(trainDF)
anyNA(testDF)
#Defining the training controls for multiple models
fitControl <- trainControl(
method = "cv",
number = 10,
savePredictions = 'final',
classProbs = T)
```

```
#fitControl <- trainControl(method = "repeatedcv", number = 10, repeats = 3,savePredictions =
'final',classProbs = T)

#######Choose the significant variables in training and testing
set##################################
training_set <-
trainDF[,c(1,2,3,4,5,6,7,8,11,13,15,17,18,19,23,24,28,29,30,31,33,34,36,37,38,39,40,41,42,43)]
testing_set <-
testDF[,c(1,2,3,4,5,6,7,8,11,13,15,17,18,19,23,24,28,29,30,31,33,34,36,37,38,39,40,41,42,43)]

# Considering all variables
#training_set <- trainDF
#testing_set <- testDF

colnames(training_set)
colnames(testing_set)

#2--Training the knn model
#model_knn<-train(training_set,training_set[,outcomeName],
#         method='knn',trControl=fitControl,preProcess = c("center", "scale"),tuneLength=3)

model_knn <-train(churn_number~. , data = training_set,method = "knn",
               trControl = fitControl,
               preProcess = c("center","scale"),
               tuneLength = 20)
#plot accuracy vs K Value graph
plot(model_knn)
#Best K
model_knn

#Predicting using knn model
pred_knn <-predict( model_knn, testing_set,"prob" )[,2]
#- Area Under Curve
plot(performance(prediction(pred_knn, testing_set$churn_number),
              "tpr", "fpr"))
testing_set[,'churn_number'] <- as.factor(testing_set[,'churn_number'])

testing_set[,'churn_number']<-ifelse(testing_set$churn_number == "yes", 1,0)

################## use probability cut off : 0.3
###########################################
pred_knn = ifelse(pred_knn > 0.3, 1,0)
#Checking the accuracy of the KNN model
#Accuracy
```

```
mean(pred_knn == testing_set$churn_number)
misClasificError <- mean(pred_knn != testing_set$churn_number)
print(paste('KNN Accuracy',1-misClasificError))
#"KNN Accuracy 0.760056791292002"
#table
table(testing_set$churn_number, pred_knn)

#- confusion matrix
confusionMatrix(factor(pred_knn),factor(testing_set$churn_number))

################### use probability cut off : 0.4
###########################################
pred_knn = ifelse(pred_knn > 0.4, 1,0)

#Checking the accuracy of the KNN model

mean(pred_knn == testing_set$churn_number)

misClasificError <- mean(pred_knn != testing_set$churn_number)
print(paste('KNN Accuracy',1-misClasificError))

#table
table(testing_set$churn_number, pred_knn  > 0.4)

#- confusion matrix
confusionMatrix(factor(pred_knn),factor(testing_set$churn_number))


################### use probability cut off : 0.5
###########################################
pred_knn = ifelse(pred_knn > 0.5, 1,0)

#Checking the accuracy of the KNN model

mean(pred_knn == testing_set$churn_number)

misClasificError <- mean(pred_knn != testing_set$churn_number)
print(paste('KNN Accuracy',1-misClasificError))
#"KNN Accuracy 0.760056791292002"

#table

table(testing_set$churn_number, pred_knn  > 0.5)
```

```
#- confusion matrix
confusionMatrix(factor(pred_knn),factor(testing_set$churn_number))
```