Radio Mast Data Analytics

SrinivasRengarajan_R00183599

03/05/2020

a) Exploratory Data Analysis and Data Cleaning/Reduction:

The training and scoring data set has been loaded as tibbles from the excel file using readxl package's read_xlsx function.

The tibbles are then casted into data frames using data.frame() for automatic cleaning of the column names or the variable names from a non-standard name into a standard ones.

The entire training data is summarised using Data Explorer package's intro function and visualised via its intro_plot function. Upon summarising and visualising, few interesting and important things were observed. Firstly, Data was not balanced. There were around 90% of records for 'okay' class and just around 10% records for 'under' class. Because of this, any machine learning algorithm would learn more about the 'okay' class than the 'under' class and tend to only predict 'okay' class for all the instances in the unseen test set. This had to be handled by either balancing the data using resampling techniques like upsampling/downsampling or by choosing different evaluation metrics like Kappa which penalises the class with more number of records, Sensitivity/Specificity curve (ROC-AUC) or Precision/Recall curve (PR-AUC). Precision Recall (PR AUC) was used in for all classifiers implemented across this project because of the growing improvements on their usage in many researches evaluating the classifier with high data imbalance.

In the case of imbalanced datasets, the interpretability of ROC plots can be deceptive with respect to conclusions about the reliability of classification performance, owing to an intuitive but wrong interpretation of specificity. Precision-recall curve plots, on the other hand, can provide the viewer with an accurate prediction of future classification performance due to the fact that they evaluate the fraction of true positives among positive predictions" -— "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets, 2015." [4]

If the ratio of positive to negative instances changes in a test set, then the ROC curves will not change. But metrics such as precision, F scores use values from both columns of the confusion matrix. As a class distribution changes these measures will change as well. ROC graphs are based upon TP rate and FP rate, in which each dimension is a columnar ratio, so does not depend on class distributions. — "ROC Graphs: Notes and Practical Considerations for Data Mining Researchers, 2003". [5]

There were around 95% of rows with complete records while having just around 7% of missing values. About 75% of the features were continuous while the rest were discrete in nature. There were totally 79 features initially.

Features like AntennaFileName1, AntennaFileName2, RFDBid etc.. which didn't look relevant in solving the problem (after reading the domain knowledge pdf on radio masts) were dropped from the train and score data.

Then, Correlation matrix heat map is plotted using ggplot for the numerical features and found that ThermalFadeMargin1, ThermalFadeMargin1, EffectiveFadeMargin1, EffectiveFadeMargin2, Flatfademarginmultipathd B2 etc.. were strongly correlated while FrequencyMHZ appeared to be a predictor which doesn't have any strong correlations with other predictors. The correlation cut-off used is 0.75.

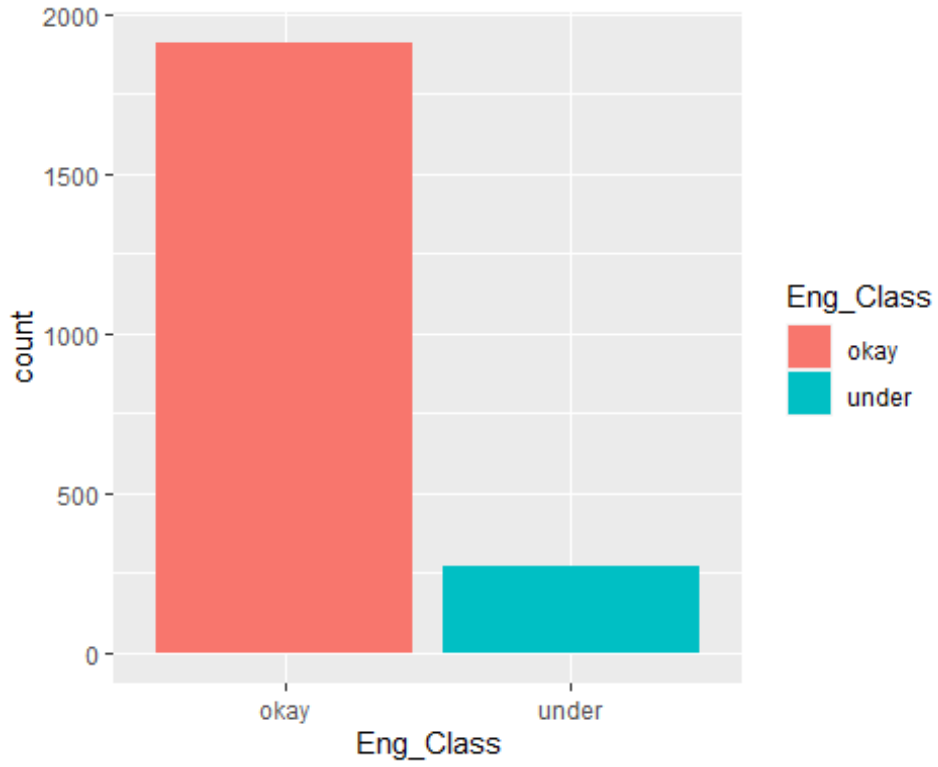
The categorical variables ("Eng_Class","DispersivefademargindB1","EmissiondesignatorI","MiscellaneouslossdB1","Passivegain2dB","Polarization","RXthresholdcriteria1","RXthresholdcriteria2") were then factorized in the training and scoring set using factor() function applied inside lapply() function.

Predictors with zero variance and near-zero variances were found using nearzerovar function and removed from the train and score data as those predictors may cause our model to be unstable or crashing. Zero variance means a variable or predictor having only one unique value whereas near zero variance means a predictor having very few unique values. In this radio mast analysis problem, Dispersivefadeoccurrencefactor was found to be a zero variance predictor and predictors related to AntennaHeight, Circulatorbranchingloss, Diffractionloss, Dispersivefademargin, Passivegain were to name a few out of 10 near zero variance predictors.

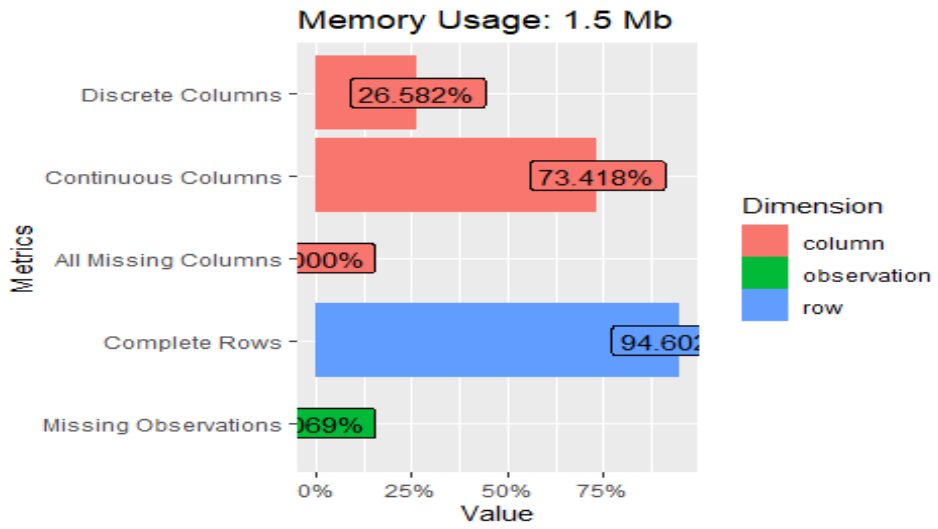
The missing value columns were then found (DpQ_R2 (9 missing values) and Fullmint1 (6 missing values)) and the number of missing values in each column has been obtained using aggr function under VIM package. Those predictors with missing values were then imputed with their respective median values using the preProcess function available in the Caret package.

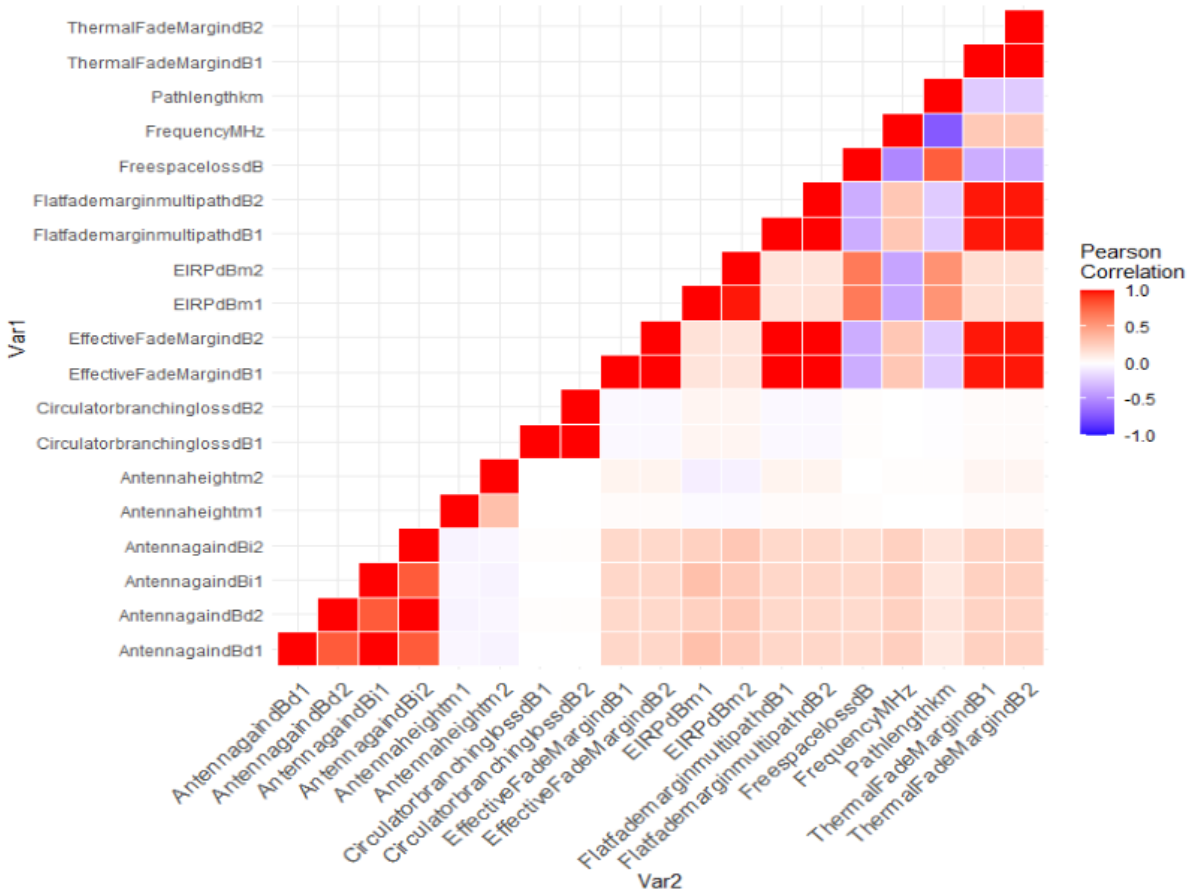
The entire structure of the training and scoring set has been verified after the data cleaning process using the str function.

```
##
## Variables having zero variance (having only one unique value):

## Dispersivefadeoccurrencefactor
##                              1

ggplot(train.data, aes(Eng_Class)) + geom_bar(aes(fill=Eng_Class))
```




```
library(DataExplorer)

introduce(train.data)

##   rows columns discrete_columns continuous_columns all_missing_columns
## 1 2186      79               21                 58                   0
##   total_missing_values complete_rows total_observations memory_usage
## 1                  120          2068              172694      1570160

plot_intro(train.data)
```



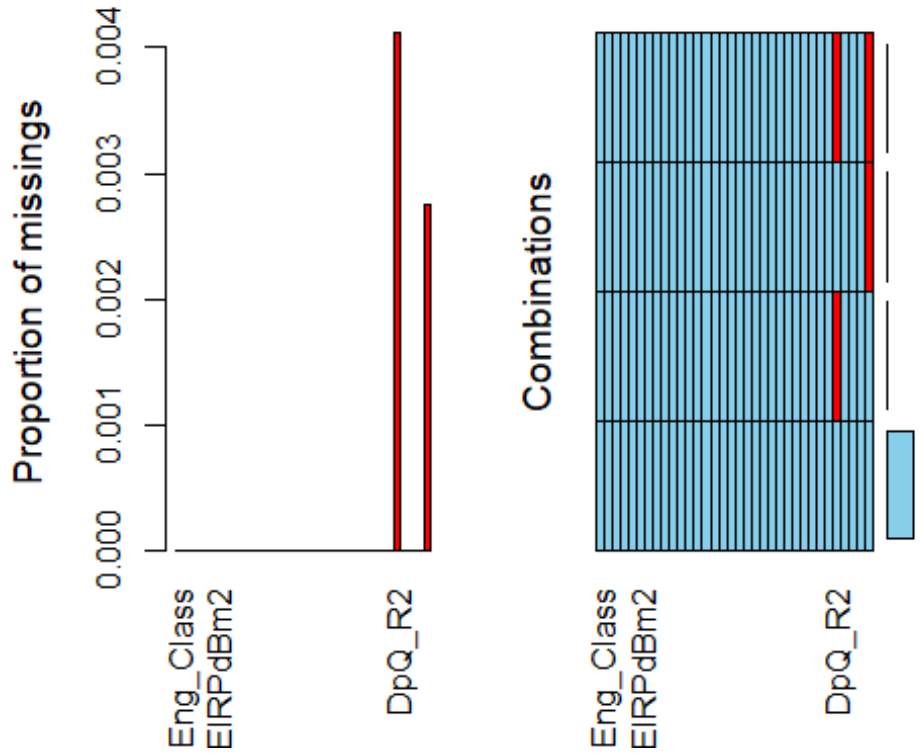



```
##  [1] "ThermalFadeMargindB2"       "ThermalFadeMargindB1"
##  [3] "FlatfademarginmultipathdB2" "EffectiveFadeMargindB2"
##  [5] "EffectiveFadeMargindB1"     "FreespacelossdB"
##  [7] "AntennagaindBi1"            "AntennagaindBd1"
##  [9] "AntennagaindBi2"            "EIRPdBm1"
## [11] "CirculatorbranchinglossdB2"
```

```
##                              freqRatio percentUnique zeroVar  nzv
## Antennaheightm1               58.51429    3.06495883   FALSE TRUE
## Antennaheightm2               55.32432    2.97346752   FALSE TRUE
## CirculatorbranchinglossdB1 1091.00000    0.13723696   FALSE TRUE
## DiffractionlossdB           706.33333    2.83623056   FALSE TRUE
## DispersivefademargindB1    1090.50000    0.18298262   FALSE TRUE
## MiscellaneouslossdB1       2185.00000    0.09149131   FALSE TRUE
## Passivegain2dB             2178.00000    0.41171089   FALSE TRUE
## XPDfademarginmultipathdB1   402.60000    6.72461116   FALSE TRUE
## XPDfademarginmultipathdB2   402.60000    6.72461116   FALSE TRUE
## NA                                 NA            NA      NA   NA
```

```
##                             freqRatio percentUnique zeroVar  nzv
## Antennaheightm1              62.64286     3.0982906   FALSE TRUE
## Antennaheightm2              52.00000     2.9914530   FALSE TRUE
```

```
## CirculatorbranchinglossdB1 932.00000     0.5341880   FALSE TRUE
## DiffractionlossdB          301.66667     2.7777778   FALSE TRUE
## DispersivefademargindB1    935.00000     0.2136752   FALSE TRUE
## MiscellaneouslossdB1         0.00000     0.1068376    TRUE TRUE
## Passivegain2dB             933.00000     0.4273504   FALSE TRUE
## XPDfademarginmultipathdB1  288.66667     6.5170940   FALSE TRUE
## XPDfademarginmultipathdB2  288.66667     6.5170940   FALSE TRUE
## NA                                  NA            NA      NA   NA

## [1] 2186   34

## [1] 936  33

## Warning in plot.aggr(res, ...): not enough horizontal space to display
## frequencies
```




```
##
##  Missings in variables:
##   Variable Count
##      DpQ_R2     9
##  Fullmint1     6
```

On further analysis of a few predictors using barplot, boxplots in ggplot and chi-square statistical test, it was observed that:

1) The under engineered radio masts had a median FlatfademarginmultipathdB1 of around 15 whereas the well engineered radio masts had around 25.

2) The distribution of FrequencyMhz for both okay and under engineered masts are left skewed. The under engineered radio masts had a median FrequencyMhz of around 26 whereas the well engineered radio masts had around 22.

3) About 85 % of the vertically polarized masts were okay and the rest under engineered.

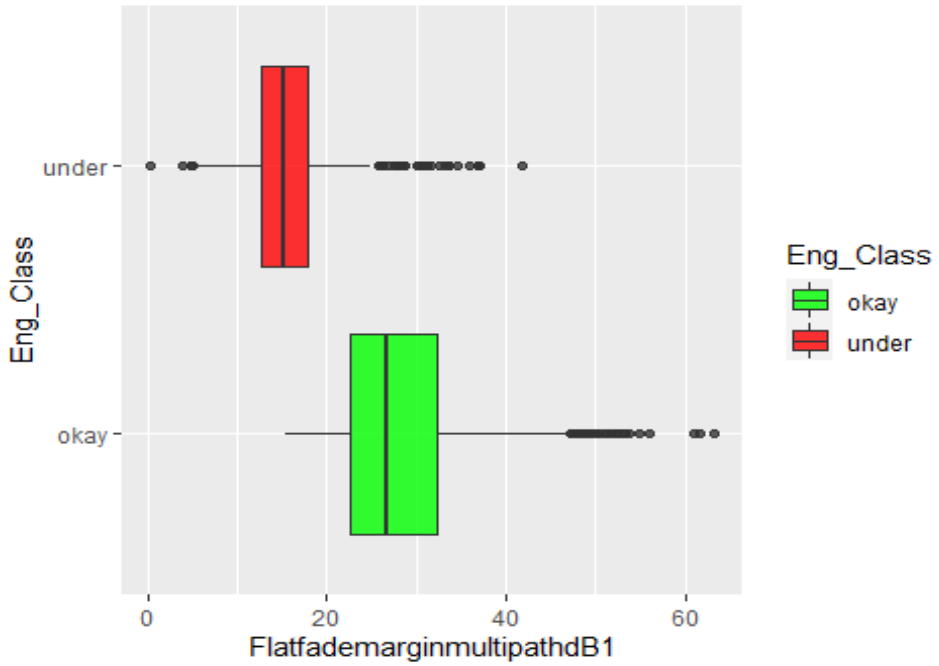
4) The median of MainreceivesignaldBm1 for both type of masts are almost the same around -50.

5) The chi-squared tests showed that the Polarization and RxThresholdCriteria1 factors were significant at 95% confidence in classifying the radio mast as okay or under engineered.

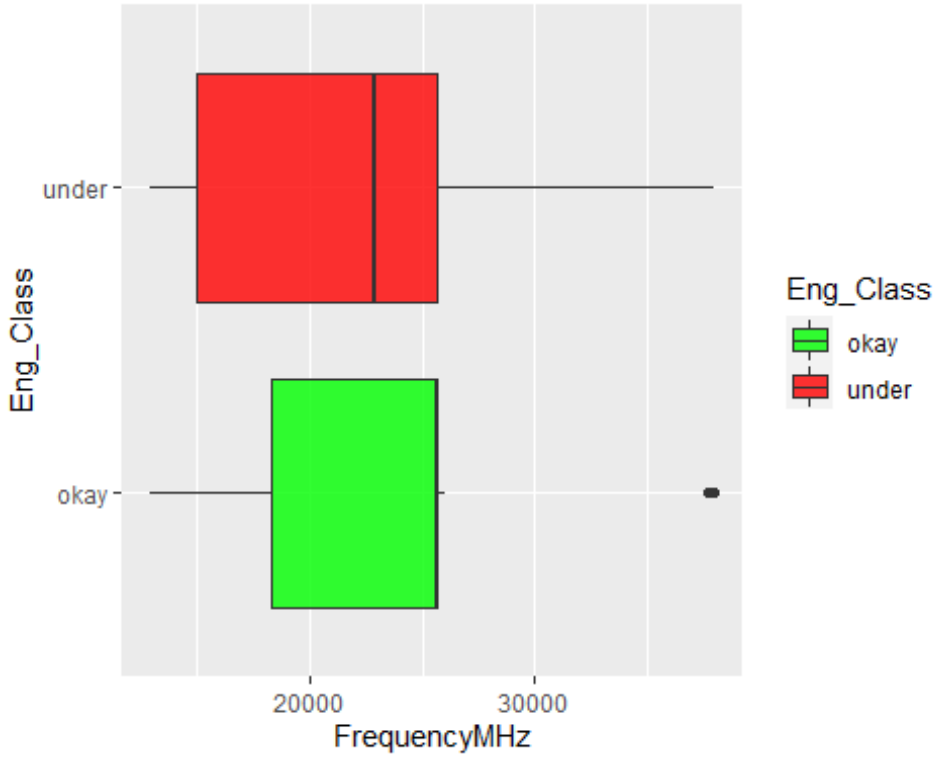
6) EDA Actions & Proposals:

With the help of all the Exploratory Data Analysis done above, we can observe that nearly half the amount of predictors are not so influential in one way or the other in classifying the labels. So,this could introduce noise in the model which might lead to low bias(more accuracy in training) and high variance(bad predictions on unseen data) i.e., overfitting. Thus, dropped those features from the training and scoring data.

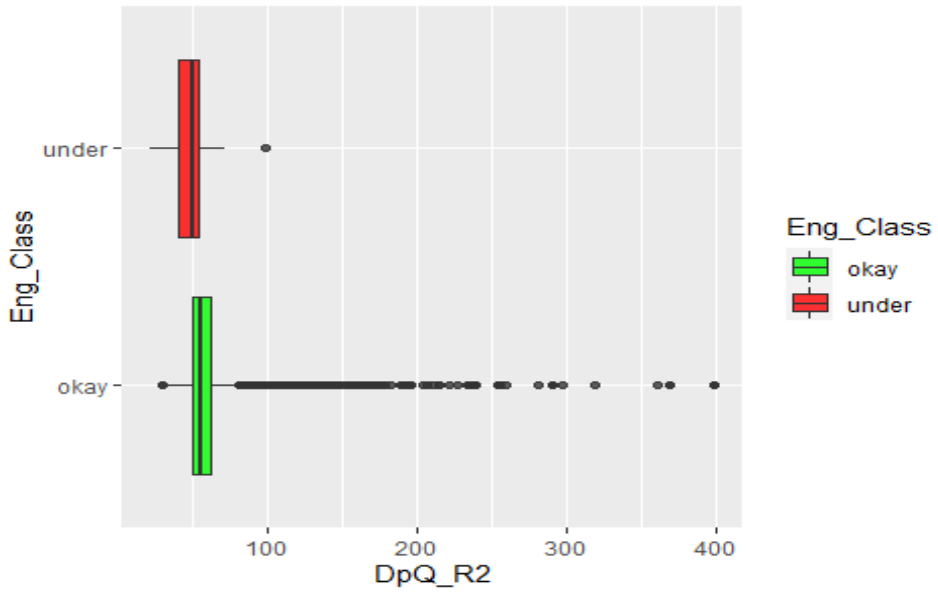
```
#Colours used in plots generated by ggplots
myfillcolors=c("green","red")

#Boxplot for FlatfademarginmultipathdB1 vs Engineering_Class
train.data%>%ggplot(aes(x=Eng_Class,y=FlatfademarginmultipathdB1,fill=Eng_Cla
ss))+geom_boxplot(alpha=0.8)+scale_fill_manual(values=myfillcolors)+coord_fli
p()
```



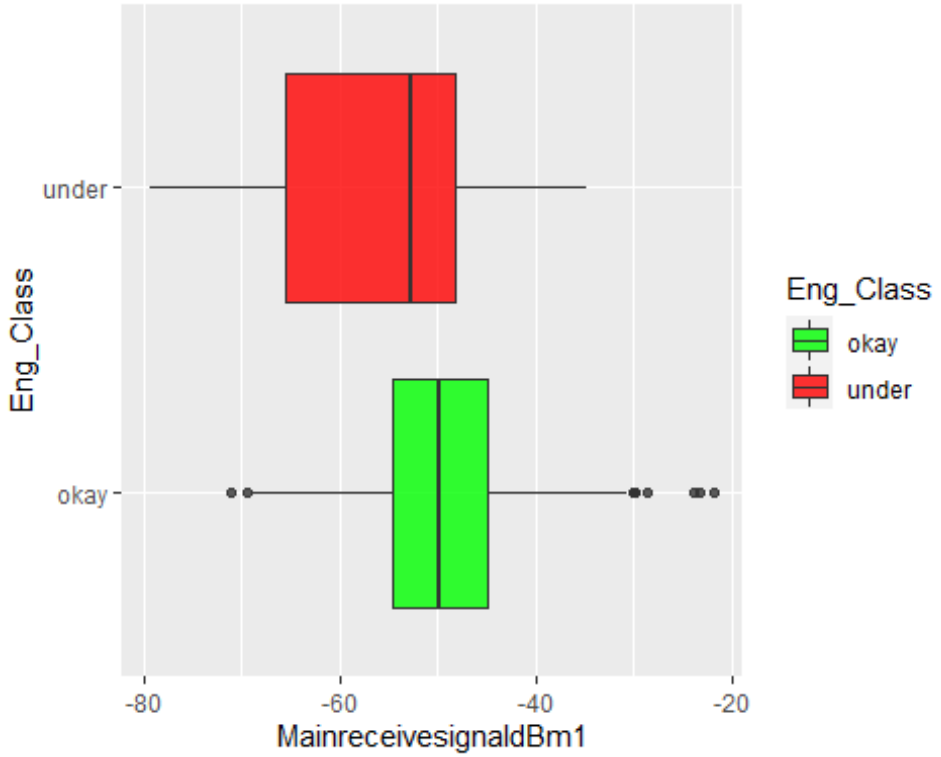
```
#Boxplot for FrequencyMHz vs Engineering_Class
train.data%>%ggplot(aes(x=Eng_Class,y=FrequencyMHz,fill=Eng_Class))+geom_boxp
lot(alpha=0.8)+scale_fill_manual(values=myfillcolors)+coord_flip()
```



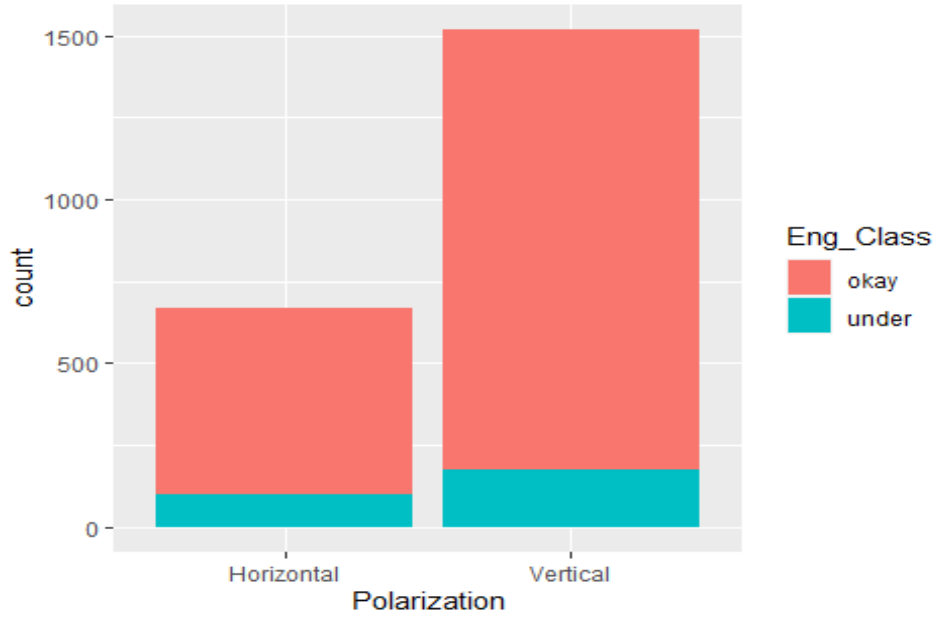

```
#Boxplot for DpQ_R2 vs Engineering_Class
train.data%>%ggplot(aes(x=Eng_Class,y=DpQ_R2,fill=Eng_Class))+geom_boxplot(al
pha=0.8)+scale_fill_manual(values=myfillcolors)+coord_flip()

## Warning: Removed 9 rows containing non-finite values (stat_boxplot).
```



```
#Boxplot for MainreceivesignaldBm1 vs Engineering_Class
train.data%>%ggplot(aes(x=Eng_Class,y=MainreceivesignaldBm1,fill=Eng_Class))+
geom_boxplot(alpha=0.8)+scale_fill_manual(values=myfillcolors)+coord_flip()
```




```
#Barplot for Polarization vs Engineering_Class
ggplot(train.data, aes(Polarization)) + geom_bar(aes(fill=Eng_Class))
```



```
#Performing chi-square test to analyse the reltionship between Polarization a
nd Engineering_Class

#Null Hypothesis: No relationship exists between Polarization and Engineering
_Class
#Alternate Hypothesis: Polarization and Engineering_Class have relationship b
etween them

chisq.test(train.data$Polarization,train.data$Eng_Class,correct = F)

##
##  Pearson's Chi-squared test
##
## data:  train.data$Polarization and train.data$Eng_Class
## X-squared = 5.2907, df = 1, p-value = 0.02144

#p-value=0.02144 which is lesser than 0.05(5% significance level), so rejecte
d the null hypothesis.
#Therefore, with 95% confidence, we could say that there was a relationship e
xisting between Polarization and Engineering_Class.

#Performing chi-square test to analyse the reltionship between Polarization a
nd Engineering_Class

#Null Hypothesis: No relationship exists between RXthresholdcriteria1 and Eng
ineering_Class
#Alternate Hypothesis: RXthresholdcriteria1 and Engineering_Class have relati
onship between them

chisq.test(train.data$RXthresholdcriteria1,train.data$Eng_Class,correct = F)

##
##  Pearson's Chi-squared test
##
## data:  train.data$RXthresholdcriteria1 and train.data$Eng_Class
## X-squared = 190.67, df = 1, p-value < 2.2e-16

#p-value < 2.2e-16 which is way lesser than 0.05(5% significance level), so r
ejected the null hypothesis.
#Therefore, with 95% confidence, we could say that a relationship exist betwe
en Polarization and Engineering_Class.
```

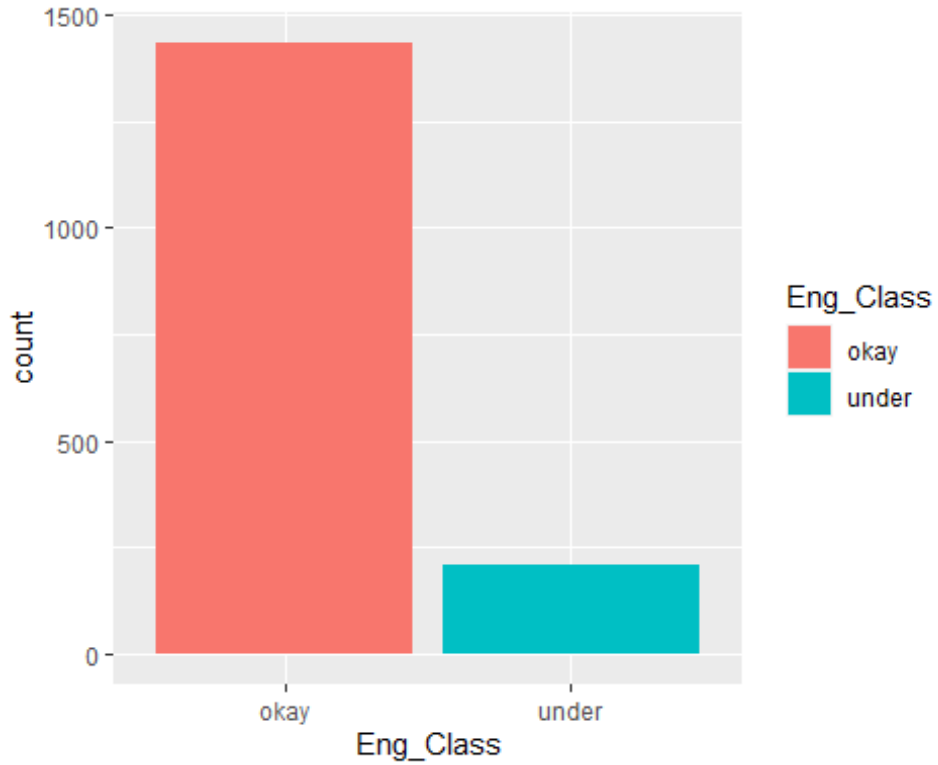
b) Train/Test methodology:

There were around 2186 observations in original dataset. So, it had been split in the ratio of 75:25 into train and test set using createDataPartition function available in the Caret package. This function does random resampling of the data and split them. With the help of bar plot in ggplot, it was observed that both the split train and test set had around 85% of data for 'okay' class and around 15% in 'under' class indicating data imbalance.

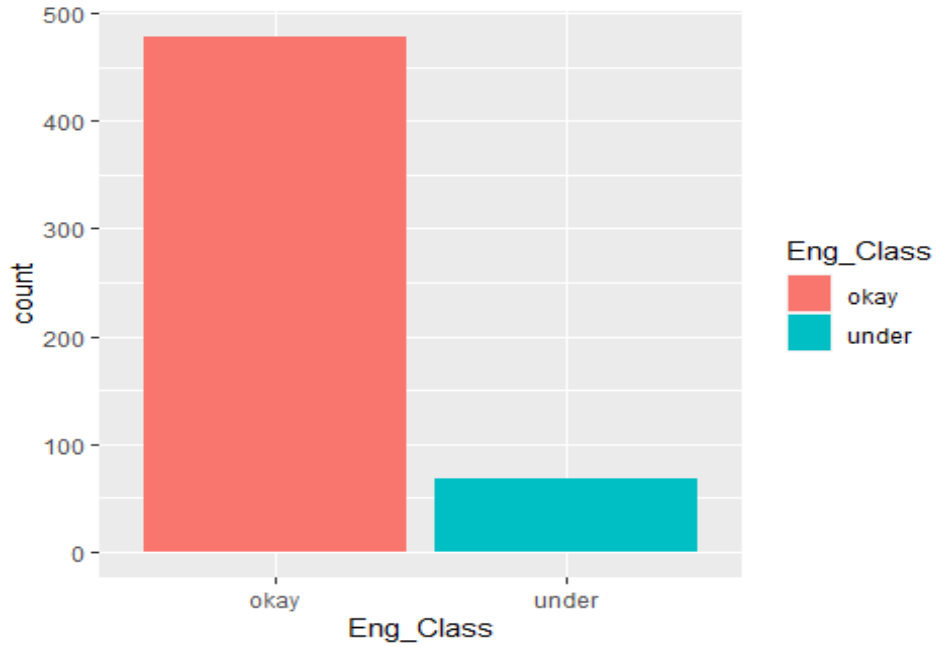
```
## [1] 1640   34
```

```
## [1] 546  34
```

```
## Value Counts in Train set
```




```
## Value Counts in Test set
```



The missing values for the predictors were imputed in the train and test data using the median imputation available in the preProcess function in the caret package.

```
## After Pre-Processing, the number of predictors with missing values:  0
```

The three algorithms chosen for classification were knn, decision tree and random forest. All the three classifiers were trained in the train set, cross-validated using 10-fold cross validation and evaluated in the test set. The standard performance metric used for model comparisons and final model choosing was PR-AUC.

First, a KNN model was built with hyperparameter tuning using functions in caret package. knn-classification algorithm is an instance based learning algorithm. All the numerical features were centred and scaled using caret's preProcess function. Centering and scaling was done to bring all the numerical features into the same scale (with mean 0 and standard deviation 1) so that it would be easier for the algorithm to find the Euclidean distance. Given an unseen test instance, the label was predicted for it by taking the majority vote of the labels among the k closest neighbours in the training set. k is the hyperparameter here. It was tuned to get the optimal parameter using grid search having the search space of k=5 to 30. The highest AUC (0.41) was obtianed for k=30 as could be seen in the line plot below.

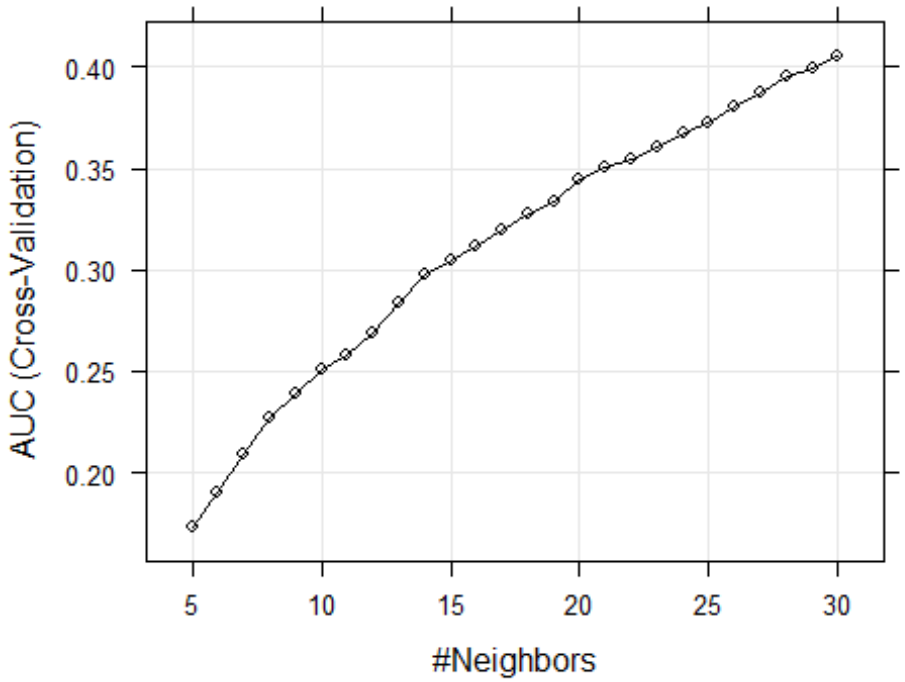
The AUC of the tuned knn model on the test set is 0.727. As this was less, went with buiding the decision tree and random forest models and then compared their performances.

```
#For getting prSummary function
library(MLmetrics)

##
## Attaching package: 'MLmetrics'

## The following objects are masked from 'package:caret':
##
##     MAE, RMSE

## The following object is masked from 'package:base':
##
##     Recall

library(PRROC)

#setting the training controls
fitControl <- trainControl(method="cv",number = 10,classProbs=TRUE,summaryFun
ction = prSummary)

#Setting the search space for the optimal hyperparameter in the grid
knnGrid <- expand.grid(k=5:30)

set.seed(599)

#Fitting the model with the control and grid parameters
knnFit <- train(Eng_Class~.,data = train,method = "knn",trControl=fitControl,
```

```
tuneGrid=knnGrid,
                preProcess = c("center","scale"), metric="AUC")
knnFit
```

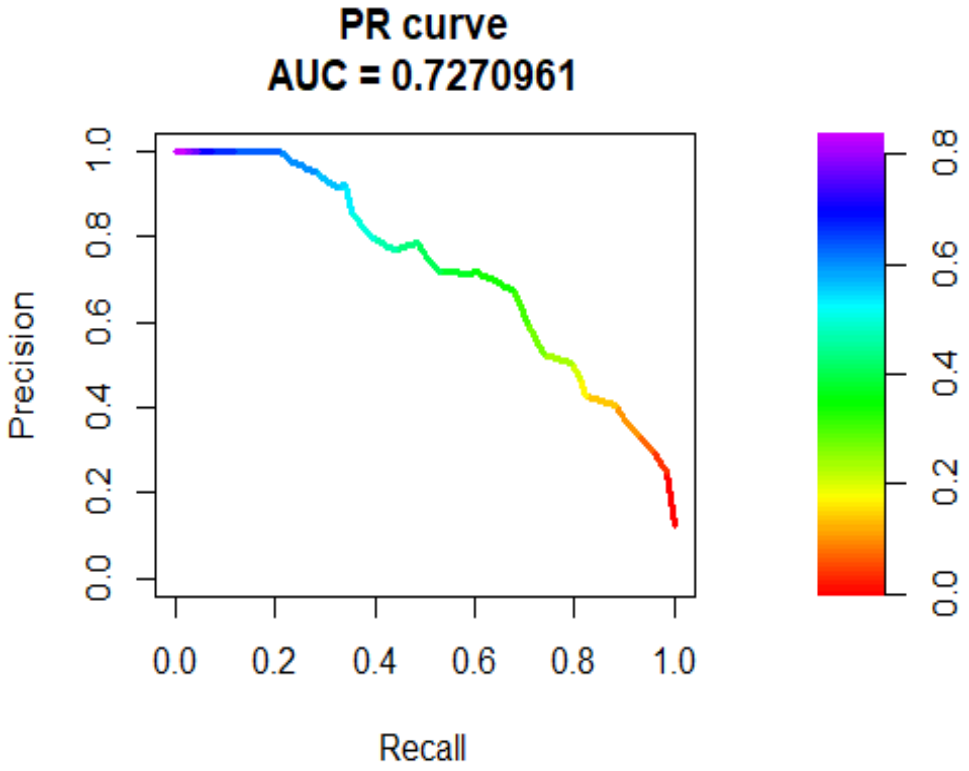
```
## k-Nearest Neighbors
##
## 1640 samples
##   33 predictor
##    2 classes: 'okay', 'under'
##
## Pre-processing: centered (33), scaled (33)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1477, 1476, 1477, 1476, 1476, 1476, ...
## Resampling results across tuning parameters:
##
##   k   AUC        Precision  Recall     F
##    5  0.1728150  0.9274399  0.9679293  0.9471297
##    6  0.1904163  0.9265093  0.9735043  0.9493649
##    7  0.2093164  0.9196789  0.9797786  0.9487208
##    8  0.2271892  0.9205209  0.9818910  0.9501174
##    9  0.2385944  0.9199335  0.9818813  0.9498050
##   10  0.2510218  0.9185991  0.9797883  0.9481195
##   11  0.2578698  0.9182173  0.9832653  0.9495350
##   12  0.2690510  0.9164021  0.9825758  0.9482221
##   13  0.2838242  0.9165338  0.9846639  0.9492994
##   14  0.2976654  0.9194789  0.9839695  0.9505494
##   15  0.3049794  0.9174837  0.9818667  0.9485059
##   16  0.3112811  0.9183004  0.9839695  0.9499235
##   17  0.3195257  0.9166874  0.9867521  0.9503475
##   18  0.3279955  0.9161114  0.9860480  0.9497007
##   19  0.3340424  0.9178436  0.9860480  0.9506411
##   20  0.3451251  0.9154366  0.9853535  0.9490229
##   21  0.3507874  0.9149641  0.9867424  0.9494139
##   22  0.3541172  0.9120336  0.9874466  0.9481474
##   23  0.3609370  0.9107756  0.9867473  0.9471695
##   24  0.3673739  0.9149816  0.9860431  0.9490689
##   25  0.3728164  0.9133146  0.9867521  0.9484875
##   26  0.3805249  0.9118832  0.9853535  0.9470984
##   27  0.3872494  0.9125438  0.9860480  0.9477791
##   28  0.3956464  0.9087155  0.9839501  0.9447815
##   29  0.3996951  0.9112517  0.9853487  0.9467836
##   30  0.4058668  0.9113164  0.9860528  0.9471433
##
## AUC was used to select the optimal model using the largest value.
## The final value used for the model was k = 30.
```

```
trellis.par.set(caretTheme())

plot(knnFit)
```




```
#Evaluation of the fitted model in the test set by predicting their labels

knnTest <- predict(knnFit,test)

confusionMatrix(knnTest,test$Eng_Class)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction okay under
##      okay   473    44
##      under    5    24
##
##                Accuracy : 0.9103
##                  95% CI : (0.8831, 0.9329)
##     No Information Rate : 0.8755
##     P-Value [Acc > NIR] : 0.006445
##
##                   Kappa : 0.4542
##
##  Mcnemar's Test P-Value : 5.681e-08
##
##             Sensitivity : 0.9895
##             Specificity : 0.3529
##          Pos Pred Value : 0.9149
##          Neg Pred Value : 0.8276
##              Prevalence : 0.8755
```

```
##          Detection Rate : 0.8663
##    Detection Prevalence : 0.9469
##       Balanced Accuracy : 0.6712
##
##        'Positive' Class : okay
##

knnTest.prob <- predict(knnFit,test,type="prob")

index_class2 <- test$Eng_Class=="under"
index_class1 <- test$Eng_Class=="okay"

plot(pr.curve(knnTest.prob$under[index_class2],knnTest.prob$under[index_class
1],curve=TRUE))
```



Next, a decision tree model was built on the training set data with tree pruning implicitly implemented using the rpart library inside the caret library's train function.

Decision tree is a simple but easily interpretable machine learning model. The tree starts splitting from the root node (most important node) and further splits into the internal nodes until it reaches the leaf nodes or label nodes. The nodes chosen for splitting could be found either using information gain or gini index. The variable having the higest

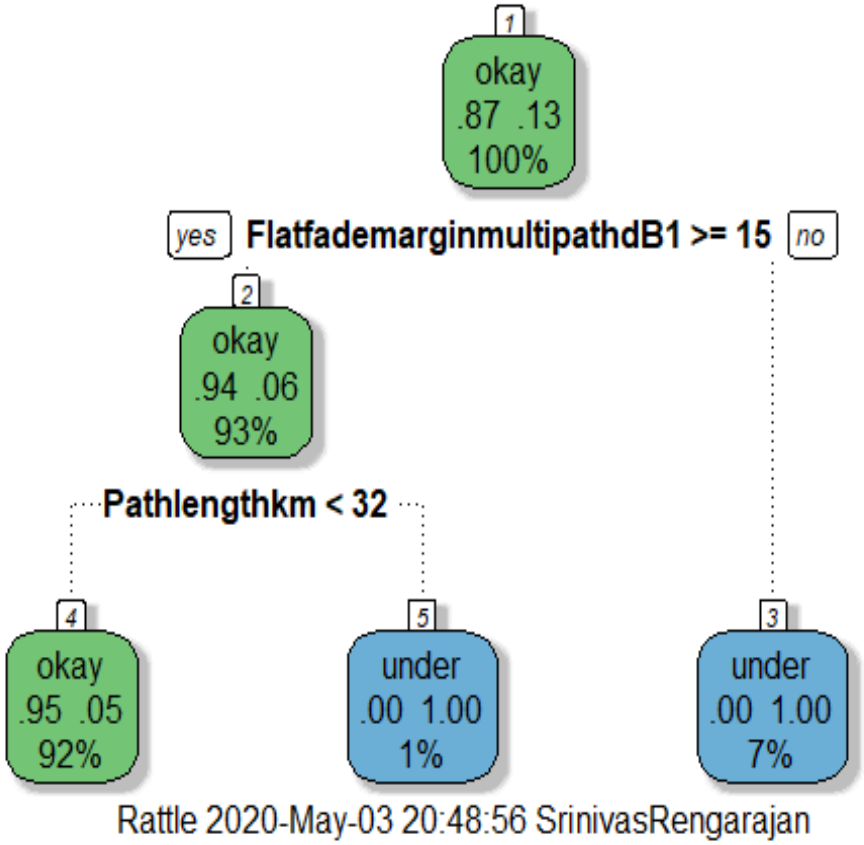
information gain is selected as the root node. Info Gain is found by subtracting the entropy (impurity or uncertainity) of the independent variable from the entropy of the target variable. Lower the probablity of a variable, higher the entropy and higher its probability, lesser the entropy. The highest AUC (0.46) was obtianed for cp = 0.04368932. [cp-complexity parameter]

The AUC of the pruned decision tree model on the test set is 0.726 which was similar to the AUC obtained by knn model. So, went ahead with building a random forest model.

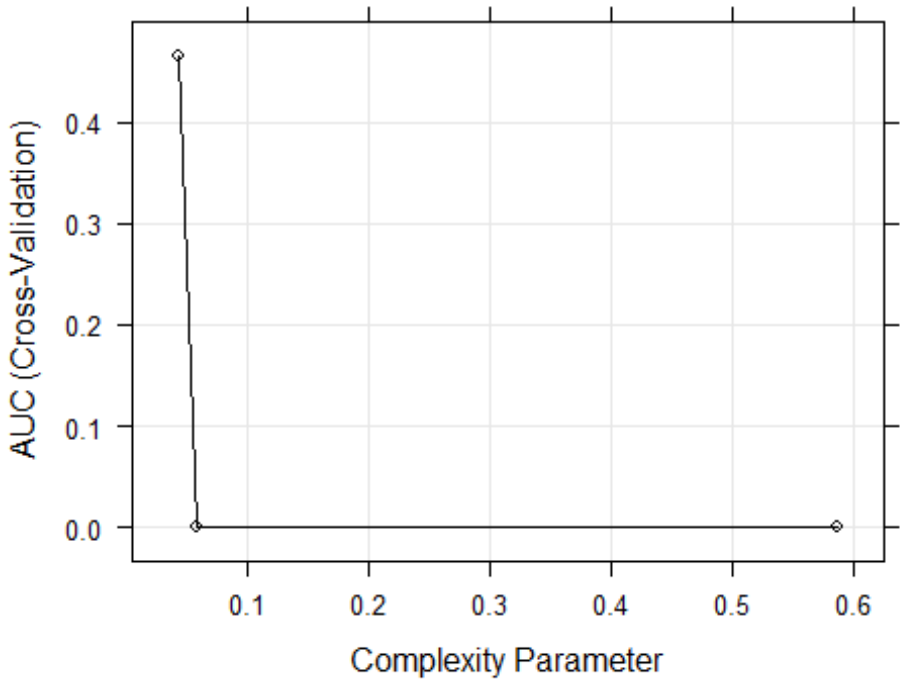
```
# ?train

#Decision Tree library
library(rpart)

#For interactive decision tree plotting
library(rattle)

## Rattle: A free graphical interface for data science with R.
## Version 5.3.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

##
## Attaching package: 'rattle'

## The following object is masked from 'package:VIM':
##
##     wine

library(rpart.plot)

library(RColorBrewer)

#For Reproducibility
set.seed(599)

#Fitting the model with the control parameters
decisionTreeFit=train(Eng_Class ~ .,
                  data=train,
                  method="rpart",
                  trControl = trainControl(method = "cv",number=10,classProbs
=T,summaryFunction=prSummary),
                  metric="AUC")

decisionTreeFit

## CART
##
## 1640 samples
##   33 predictor
```

```
##    2 classes: 'okay', 'under'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1477, 1476, 1477, 1476, 1476, 1476, ...
## Resampling results across tuning parameters:
##
##   cp          AUC           Precision  Recall     F
##   0.04368932  0.4656798942  0.9580878  0.9972028  0.9771587
##   0.05825243  0.0006445059  0.9473219  0.9993007  0.9725612
##   0.58737864  0.0000000000  0.8971352  1.0000000  0.9455385
##
## AUC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.04368932.
```

```
fancyRpartPlot(decisionTreeFit$finalModel)
```

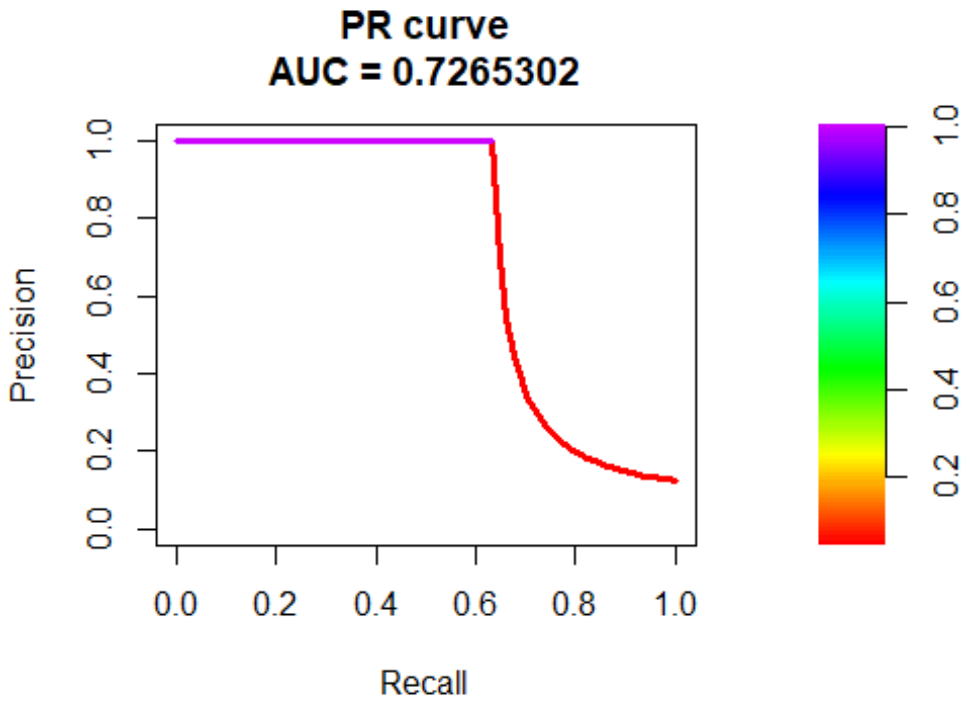



```
plot(decisionTreeFit)
```




```
#Evaluation of the fitted model in the test set by predicting their labels

decisionTreeTest <- predict(decisionTreeFit,test)

confusionMatrix(decisionTreeTest,test$Eng_Class)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction okay under
##      okay   478    25
##      under    0    43
##
##                Accuracy : 0.9542
##                  95% CI : (0.9331, 0.9702)
##     No Information Rate : 0.8755
##     P-Value [Acc > NIR] : 2.872e-10
##
##                   Kappa : 0.7507
##
##  Mcnemar's Test P-Value : 1.587e-06
##
##             Sensitivity : 1.0000
##             Specificity : 0.6324
##          Pos Pred Value : 0.9503
##          Neg Pred Value : 1.0000
##              Prevalence : 0.8755
```

```
##          Detection Rate : 0.8755
##    Detection Prevalence : 0.9212
##       Balanced Accuracy : 0.8162
##
##        'Positive' Class : okay
##

decisionTreeTest.prob <- predict(decisionTreeFit,test,type="prob")

index_class2 <- test$Eng_Class=="under"
index_class1 <- test$Eng_Class=="okay"

plot(pr.curve(decisionTreeTest.prob$under[index_class2],decisionTreeTest.prob
$under[index_class1],curve=TRUE))
```



Trying to improve the model performance using Random Forest:

Random Forest is an Ensemble technique (combination of two or more models) which uses the principle of Bagging(Bootstrapping and Aggregating) where the dataset is randomly sampled of equal observations and many number of independent trees are grown, all are modelled using decision trees and labels are predicted by majority voting as this is a classification problem, otherwise results would be averaged (regression problems).

The main parameter is mtry that is the number of features by which each tree should be split at random. We arrive at the best parameter for our combination of predictor variables

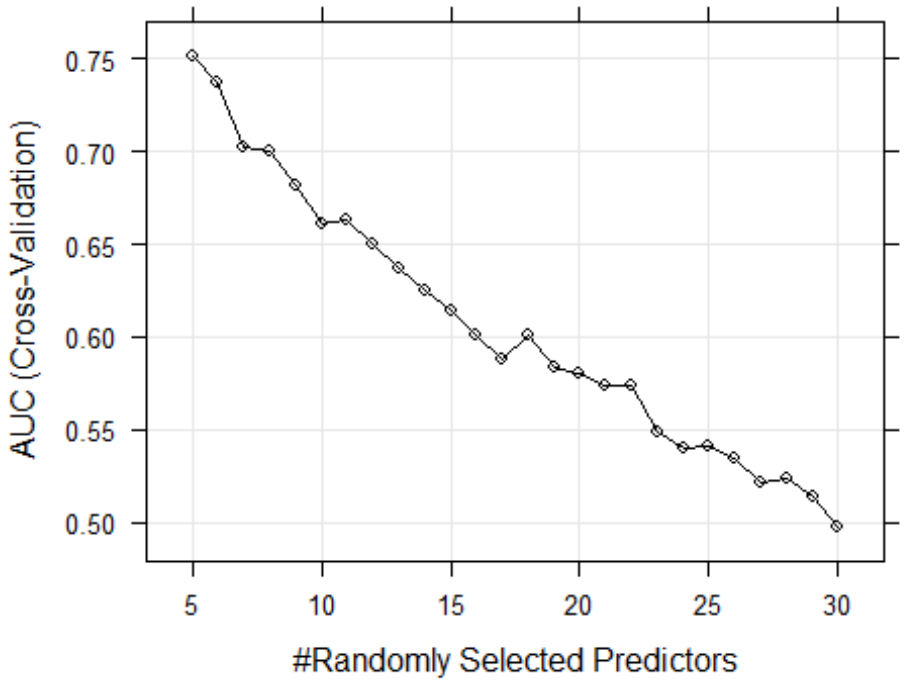
by grid searching for various mtry values ranging from 5 to 35. We obtain high AUC (75.01%) at mtry=5 comparatively to other values of mtry.

The AUC of the tuned random forest model on the test set is 0.942. This model's performance is incredibly higher than the previous two models. This is because of bootstrapping (Bagging + Aggregating). Random forest model reduces the variance by taking majority vote for the labels. It also decorrelates the trees by taking random subset of predictors for each split in the tree.

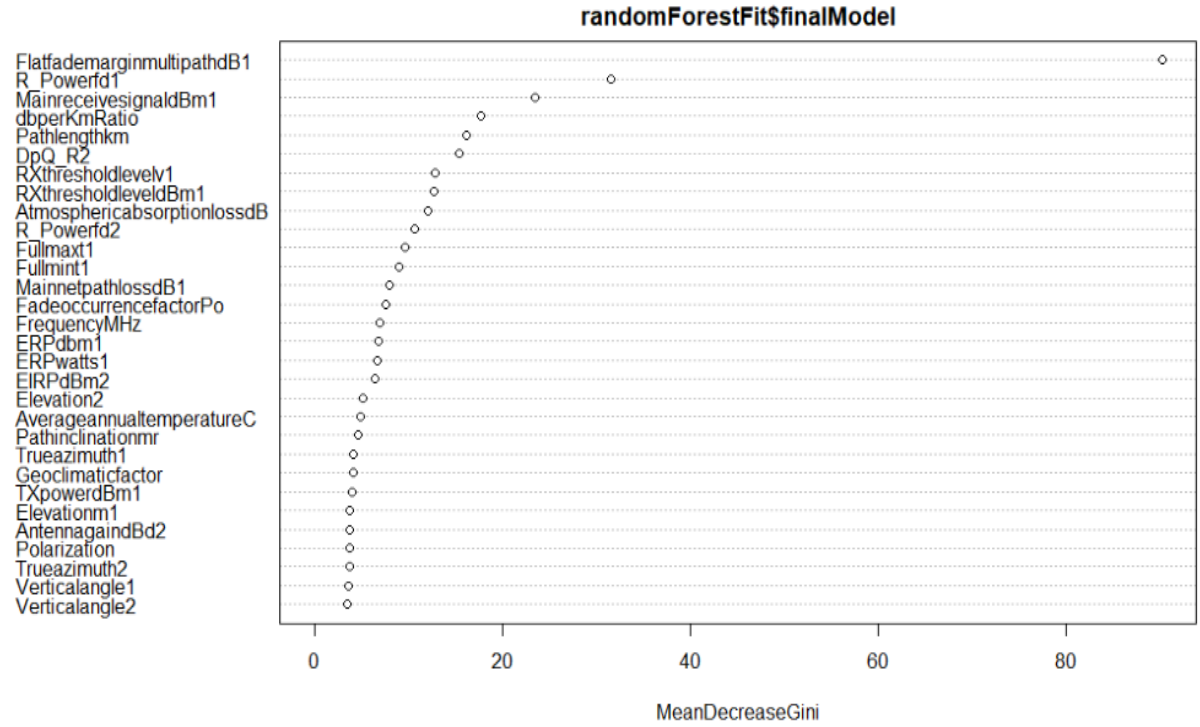
```
# ?train

#setting the grid
randomForestGrid <- expand.grid(mtry=5:30)

#setting the training control params
fitControl <- trainControl(method = "cv",number = 10,classProbs = T, summaryF
unction = prSummary)

library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:rattle':
##
##     importance

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin

#For reproducibility
set.seed(599)

#Fitting the random forest model on the training set data
randomForestFit=train(train[,2:34], train$Eng_Class,
                      method="rf",
                      trControl=fitControl,
                      tuneGrid=randomForestGrid,metric="AUC")
randomForestFit

## Random Forest
##
```

```
## 1640 samples
##   33 predictor
##    2 classes: 'okay', 'under'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1477, 1476, 1477, 1476, 1476, 1476, ...
## Resampling results across tuning parameters:
##
##   mtry  AUC        Precision  Recall     F
##    5    0.7518624  0.9610206  0.9930410  0.9766921
##    6    0.7373067  0.9616564  0.9930410  0.9770287
##    7    0.7025084  0.9616736  0.9930410  0.9770309
##    8    0.6998141  0.9656333  0.9937354  0.9794083
##    9    0.6813310  0.9675247  0.9923417  0.9797048
##   10    0.6610958  0.9668974  0.9944299  0.9804305
##   11    0.6626311  0.9694949  0.9944250  0.9817672
##   12    0.6502481  0.9675413  0.9944299  0.9807625
##   13    0.6368251  0.9695000  0.9951243  0.9821205
##   14    0.6249514  0.9708190  0.9944299  0.9824523
##   15    0.6146383  0.9715036  0.9951243  0.9831419
##   16    0.6006579  0.9714900  0.9951243  0.9831349
##   17    0.5881993  0.9721563  0.9951243  0.9834761
##   18    0.6008336  0.9721609  0.9951243  0.9834808
##   19    0.5833908  0.9728006  0.9951243  0.9838104
##   20    0.5799024  0.9734891  0.9951243  0.9841586
##   21    0.5738005  0.9748413  0.9958188  0.9851921
##   22    0.5737796  0.9754983  0.9951243  0.9851826
##   23    0.5486915  0.9761691  0.9951243  0.9855238
##   24    0.5401680  0.9768314  0.9958188  0.9862111
##   25    0.5409183  0.9768314  0.9958188  0.9862111
##   26    0.5341548  0.9774930  0.9958188  0.9865477
##   27    0.5214937  0.9768314  0.9958188  0.9862111
##   28    0.5235799  0.9774930  0.9958188  0.9865477
##   29    0.5139568  0.9774930  0.9958188  0.9865477
##   30    0.4970647  0.9774930  0.9958188  0.9865477
##
## AUC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 5.
```

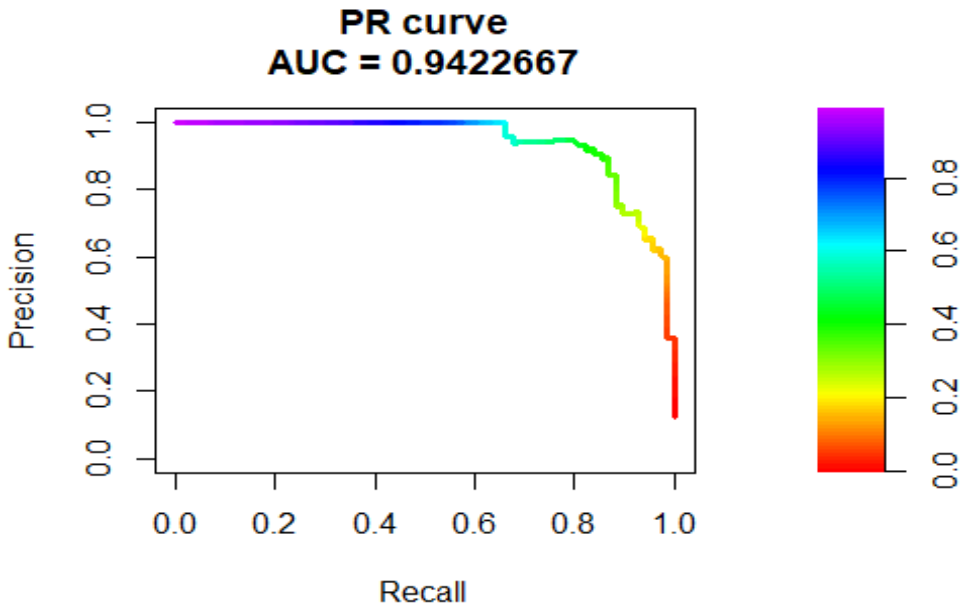
```
trellis.par.set(caretTheme())

plot(randomForestFit)
```




```r
#Checking the variable importance in the fitted model using varImpPlot functi
on
varImpPlot(randomForestFit$finalModel)
```



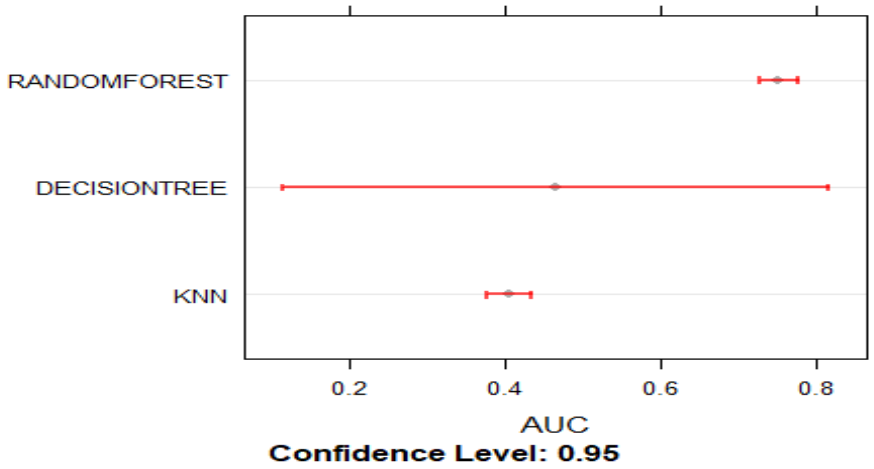
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction okay under
##      okay   475    18
##      under    3    50
##
##                Accuracy : 0.9615
##                  95% CI : (0.9418, 0.976)
##     No Information Rate : 0.8755
##     P-Value [Acc > NIR] : 2.622e-12
##
##                   Kappa : 0.8052
##
##  Mcnemar's Test P-Value : 0.00225
##
##             Sensitivity : 0.9937
##             Specificity : 0.7353
##          Pos Pred Value : 0.9635
##          Neg Pred Value : 0.9434
##              Prevalence : 0.8755
##          Detection Rate : 0.8700
##    Detection Prevalence : 0.9029
##       Balanced Accuracy : 0.8645
##
##        'Positive' Class : okay
##
```



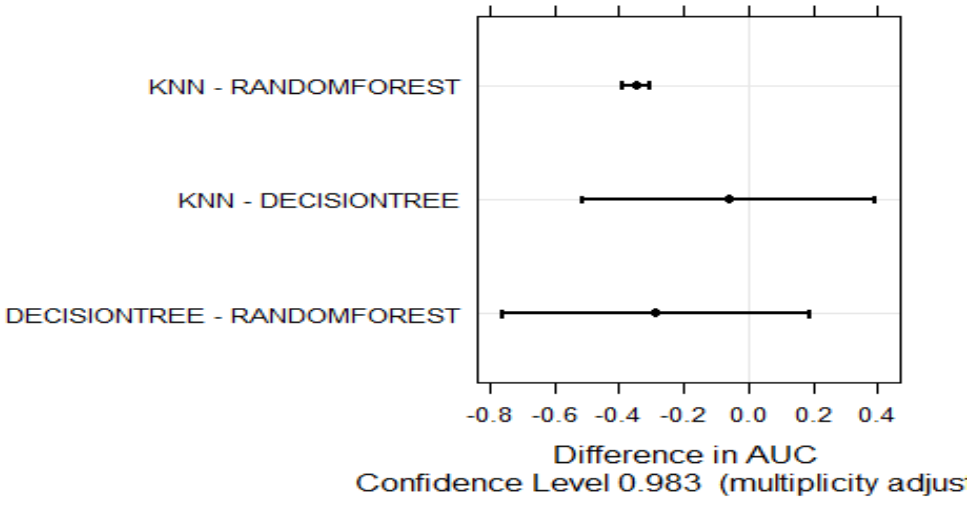
All the above three models' performances were compared by resampling function. This function provides methods for analyzing and visualising a set of resampling results (results from fitted models) using a common data taken from training set. On visualising, it could be very well seen that the median AUC's of 3 models (KNN - 0.41, Decision Tree - 0.46 and Random Forest - 0.75) for 10 samples were in alignment what what we had obtained for the entire training set data. When plotted the difference in AUC's between these models using dotplot, it could be explained that there was a very little difference in performance between KNN and decision tree whereas random forest model had a huge difference to those models of around 0.35. So, our final model choosen for classfying the scoring data was Random Forest.

```
set.seed(599)

#Taking a set of common samples from the fitted models and running the model
on them for performance comparison
resamps <- resamples(list(KNN=knnFit, DECISIONTREE=decisionTreeFit, RANDOMFOR
EST=randomForestFit))
resamps


## Call:
## resamples.default(x = list(KNN = knnFit, DECISIONTREE =
##  decisionTreeFit, RANDOMFOREST = randomForestFit))
##
## Models: KNN, DECISIONTREE, RANDOMFOREST
## Number of resamples: 10
## Performance metrics: AUC, F, Precision, Recall
## Time estimates for: everything, final model fit

summary(resamps)

##
## Call:
## summary.resamples(object = resamps)
##
## Models: KNN, DECISIONTREE, RANDOMFOREST
## Number of resamples: 10
##
## AUC
##                  Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA
's
## KNN          0.353431 0.3750656 0.3987668 0.4058668 0.4362484 0.4776447
0
## DECISIONTREE 0.000000 0.0000000 0.4474919 0.4656799 0.9327245 0.9643183
0
## RANDOMFOREST 0.708796 0.7340355 0.7497702 0.7518624 0.7599802 0.8381519
0
##
## F
##                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max. N
```

```
A's
## KNN          0.9333333 0.9384438 0.9494949 0.9471433 0.9547132 0.9594595
0
## DECISIONTREE 0.9659864 0.9704700 0.9778616 0.9771587 0.9828154 0.9863014
0
## RANDOMFOREST 0.9692833 0.9727891 0.9758595 0.9766921 0.9819222 0.9828179
0
##
## Precision
##                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max. N
A's
## KNN          0.8867925 0.8952653 0.9132062 0.9113164 0.9264706 0.9342105
0
## DECISIONTREE 0.9403974 0.9426373 0.9566870 0.9580878 0.9712026 0.9859155
0
## RANDOMFOREST 0.9470199 0.9530984 0.9595918 0.9610206 0.9710061 0.9790210
0
##
## Recall
##                   Min.   1st Qu.    Median      Mean   3rd Qu.      Max. N
A's
## KNN          0.9790210 0.9860140 0.9860140 0.9860528 0.9861111 0.9930556
0
## DECISIONTREE 0.9790210 1.0000000 1.0000000 0.9972028 1.0000000 1.0000000
0
## RANDOMFOREST 0.9722222 0.9878351 0.9965278 0.9930410 1.0000000 1.0000000
resamps_theme <- trellis.par.get()
resamps_theme$plot.symbol$col=rgb(.2,.2,.2,.4)
resamps_theme$plot.symbol$pch=16
resamps_theme$plot.line$col=rgb(1,0,0,.7)
resamps_theme$plot.line$lwd <- 2
trellis.par.set(resamps_theme)

dotplot(resamps, metric = "AUC")
```



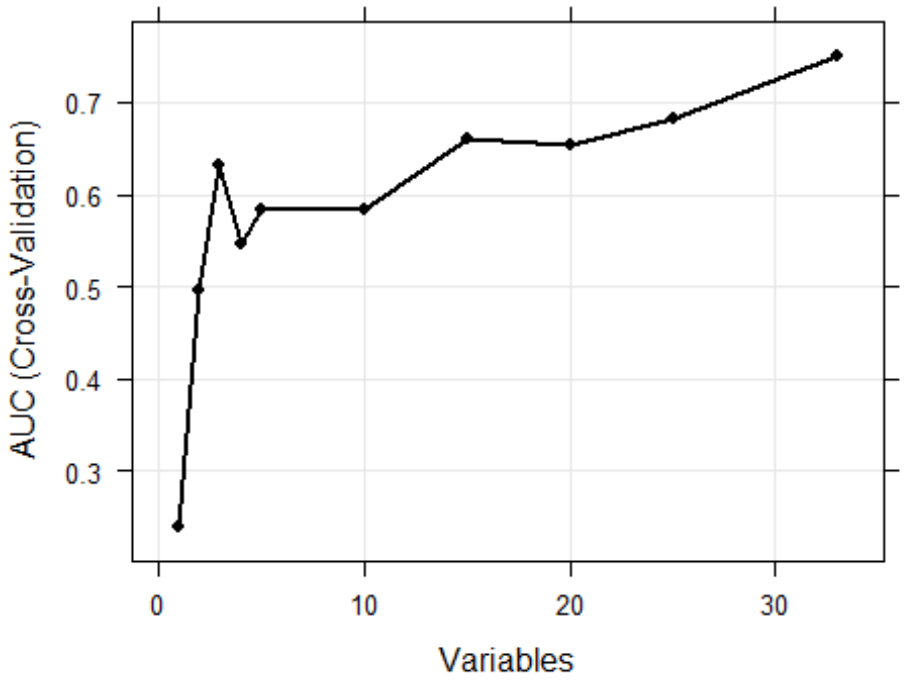
```r
#To check how much one model was differing from the other in terms of perform
ance metrics
difference <- diff(resamps)
summary(difference)
## Call:
## summary.diff.resamples(object = difference)
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
## AUC
##               KNN       DECISIONTREE RANDOMFOREST
## KNN                     -0.05981     -0.34600
## DECISIONTREE 1.0000                  -0.28618
## RANDOMFOREST 8.578e-09 0.3305
## F
##               KNN       DECISIONTREE RANDOMFOREST
## KNN                     -0.0300154   -0.0295488
## DECISIONTREE 7.617e-05               0.0004666
## RANDOMFOREST 3.720e-05 1
## Precision
##               KNN       DECISIONTREE RANDOMFOREST
## KNN                     -0.046771    -0.049704
## DECISIONTREE 0.0007864               -0.002933
## RANDOMFOREST 2.576e-05 1.0000000
## Recall
##               KNN      DECISIONTREE RANDOMFOREST
## KNN                     -0.011150    -0.006988
## DECISIONTREE 0.000598                0.004162
## RANDOMFOREST 0.044719 0.895048

trellis.par.set(caretTheme())
dotplot(difference)
```



c) **Recursive feature elimination** was used as the feature selection method here. This technique would build a model with all the variables, and then the algorithm would start removing the weakest features one by one until we reach the specified number of features. In Recursive Feature Elimination, we would need to specify the feature sizes to be used. This could be specified using the subset parameter. From the graph below, we could see that a better comparable AUC was obtained with 15 variables. After that it didn't increase much. So, the top 15 best performing features were selected for optimizing the random forest model.

```
library(mlbench)

#Predictors sizes
subsets <- c(1:5,10,15,20,25)

rfFuncs$summary=prSummary

# define the control using a random forest selection function
control <- rfeControl(functions=rfFuncs,method="cv",number=10, verbose = FALS
E)

fitControl <- trainControl(classProbs = T, summaryFunction = prSummary)

set.seed(599)

# running the Recursive Feature Elimination algorithm
results <- rfe(train[,2:34], train$Eng_Class, sizes=subsets, trControl=fitCon
trol, rfeControl=control, metric="AUC")

results

##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (10 fold)
##
## Resampling performance over subset size:
##
##  Variables    AUC Precision Recall      F   AUCSD PrecisionSD RecallSD
FSD
##          1 0.2380    0.9491 0.9610 0.9549 0.04448     0.01088 0.018062 0.0
11080
##          2 0.4958    0.9497 0.9840 0.9665 0.05562     0.01324 0.011861 0.0
09306
##          3 0.6314    0.9564 0.9909 0.9733 0.03419     0.01256 0.008054 0.0
06885
##          4 0.5457    0.9608 0.9889 0.9746 0.06201     0.01222 0.009964 0.0
08712
##          5 0.5847    0.9615 0.9902 0.9756 0.05236     0.01302 0.010508 0.0
```

```
08752
##         10 0.5834    0.9742 0.9937 0.9838 0.03614     0.01582 0.008334 0.0
10882
##         15 0.6614    0.9709 0.9944 0.9825 0.03577     0.01281 0.007907 0.0
08253
##         20 0.6540    0.9683 0.9944 0.9811 0.02955     0.01342 0.007907 0.0
08176
##         25 0.6825    0.9662 0.9937 0.9797 0.04443     0.01023 0.007665 0.0
06896
##         33 0.7514    0.9616 0.9930 0.9770 0.03749     0.01181 0.008038 0.0
07506
##  Selected
##
##
##
##
##
##
##
##
##
##
##         *
##
## The top 5 variables (out of 33):
##    FlatfademarginmultipathdB1, R_Powerfd1, Pathlengthkm, Atmosphericabsorp
tionlossdB, MainreceivesignaldBm1

#To get the optimal list of variables
results$optVariables

##  [1] "FlatfademarginmultipathdB1"  "R_Powerfd1"
##  [3] "Pathlengthkm"                "AtmosphericabsorptionlossdB"
##  [5] "MainreceivesignaldBm1"       "dbperKmRatio"
##  [7] "DpQ_R2"                      "RXthresholdleveldBm1"
##  [9] "RXthresholdlevelv1"          "Polarization"
## [11] "R_Powerfd2"                  "Fullmaxt1"
## [13] "Fullmint1"                   "EIRPdBm2"
## [15] "FadeoccurrencefactorPo"      "FrequencyMHz"
## [17] "TXpowerdBm1"                 "MainnetpathlossdB1"
## [19] "ERPwatts1"                   "ERPdbm1"
## [21] "RXthresholdcriteria1"        "AntennagaindBd2"
## [23] "OtherTXlossdB1"              "Elevation2"
## [25] "AverageannualtemperatureC"   "Pathinclinationmr"
## [27] "Verticalangle2"              "Geoclimaticfactor"
## [29] "Elevationm1"                 "Verticalangle1"
## [31] "Trueazimuth1"                "Trueazimuth2"
## [33] "OtherRXlossdB1"

# plotting the results
plot(results, type=c("g", "o"))
```




```
#varImp(randomForestFit)
```

Here, we had built a model with the top 15 best performing predictors selected from above and a random forest model was built with them and the optimized model was tested with the test set. From the graph below, we could see that the AUC (from 0.942 to 0.952) was increased slightly and also the other metrics have significantly improved like kappa (from 0.80 to 0.86) and specificity (from 0.73 to 0.79).

```
fitControl <- trainControl(method="cv",number=10, classProbs = T, summaryFunc
tion = prSummary)

#randomForestGrid <- expand.grid(mtry=2:10)

set.seed(599)

randomForestFit.optimal=train(Eng_Class~FlatfademarginmultipathdB1+R_Powerfd1
+Pathlengthkm+MainreceivesignaldBm1+AtmosphericabsorptionlossdB+dbperKmRatio+
DpQ_R2+RXthresholdlevelv1+RXthresholdleveldBm1+Polarization+R_Powerfd2+Fullma
xt1+Fullmint1+EIRPdBm2+FadeoccurrencefactorPo, data=train,method="rf", trCont
rol=fitControl, metric="AUC")

randomForestFit.optimal

## Random Forest
##
## 1640 samples
```

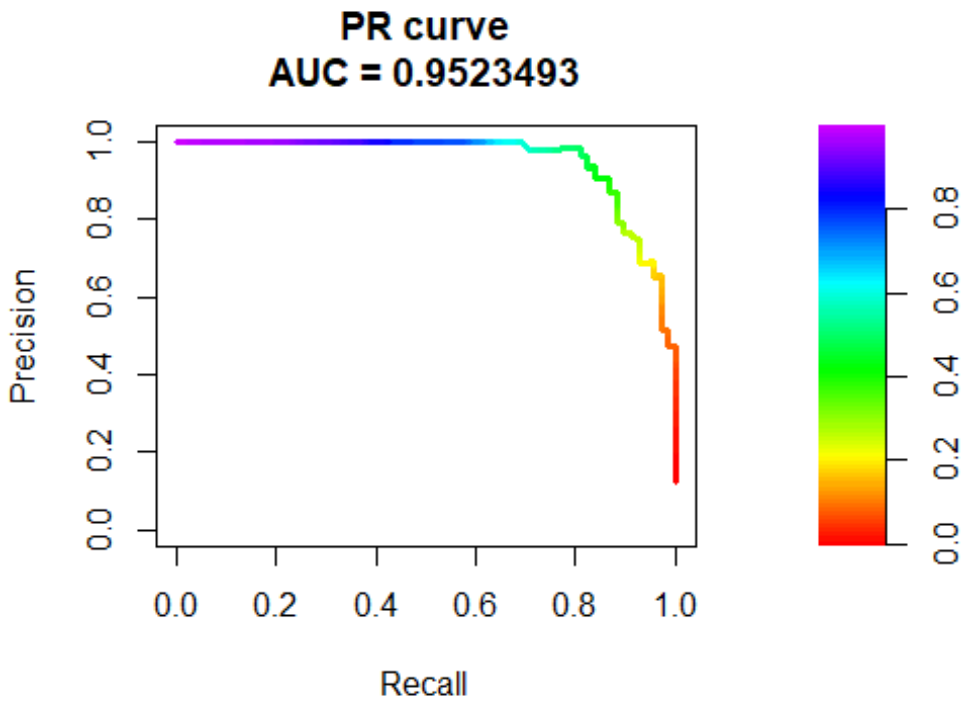
```
##   15 predictor
##    2 classes: 'okay', 'under'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1477, 1476, 1477, 1476, 1476, 1476, ...
## Resampling results across tuning parameters:
##
##   mtry  AUC        Precision  Recall     F
##    2    0.7243177  0.9655145  0.9916472  0.9783397
##    8    0.5807242  0.9742160  0.9951243  0.9845094
##   15    0.4716037  0.9801813  0.9951195  0.9875643
##
## AUC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
randomForestTest2 <- predict(randomForestFit.optimal,test)

confusionMatrix(randomForestTest2,test$Eng_Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction okay under
##      okay   477    14
##      under    1    54
##
##                Accuracy : 0.9725
##                  95% CI : (0.9551, 0.9845)
##     No Information Rate : 0.8755
##     P-Value [Acc > NIR] : 5.105e-16
##
##                   Kappa : 0.8628
##
##  Mcnemar's Test P-Value : 0.001946
##
##             Sensitivity : 0.9979
##             Specificity : 0.7941
##          Pos Pred Value : 0.9715
##          Neg Pred Value : 0.9818
##              Prevalence : 0.8755
##          Detection Rate : 0.8736
##    Detection Prevalence : 0.8993
##       Balanced Accuracy : 0.8960
##
##        'Positive' Class : okay
##
```

```
randomForestTest.optimal.prob <- predict(randomForestFit.optimal,test,type="p
rob")
```

```
index_class2 <- test$Eng_Class=="under"
index_class1 <- test$Eng_Class=="okay"

plot(pr.curve(randomForestTest.optimal.prob$under[index_class2],randomForestT
est.optimal.prob$under[index_class1],curve=TRUE))
```



d) MODEL EXPLANATION TO DANIEL (RANDOM FOREST):

In a bid to explain what is all about random forest, let me take an easy example, consider you are going to purchase a new laptop and that you have fixed the brand and variant, now you will surely ask your friends (obviously more than 1) for their reviews on the same brand/variant. Here, you are not depending on any one friend's review because that would be more biased. The reason is that he might be satisfied etremely well/bad with its performance whilst others might not be. So, you asked reviews from multiple people to get a generalised/collective opinion on the laptop brand. If most of their reviews were good enough, then you might consider purchasing that laptop brand otherwise you would not. So, that's it!! This is an anecdote of how random forest works from the top level.

The above methodology is called Ensembling, meaning running multiple models on a training data set and their respective outputs are combined using some rule to obtain the final result. In our radio mast classification problem context, the rule would be to choose the class or result by majority voting (i.e., selecting the class chosen by majority of the models for the given data as the final answer); the data would be the important features of the radio mast. Each model would be a decision tree in random forest algorithm. Decision tree is just a condition based tree which outputs the class label or result for the given input

by traversing the new input down the already trained tree. The conditions would be normal if..else questions asked based on the features. If the condition is evaluated to true, the tree goes down the left and if the condition is false, the tree goes down the right. The tree keeps on parsing or going down till it reaches the result node or till it predicts the class for that record or observation. The difference in random forest to that of a decision tree is that here, each tree would be built with random set of observations and the questions asked for building the tree would be based on random subset of features of radio mast for different trees. Thus, any bias in the model is removed and the predictive power of the model significantly improves. In business scenario, this model would be easy to build and use, compared to many other compilcated models but would still acheive greater performance.

Now that we knew the basics of how the random forest model works, I would like to brief you how the random forest model I built for our problem effectively address the question i.e., classification of radio mast into okay or under engineered.

The best number of features obtained to be considered for each split in the decision tree was 5 as this gave good performance in comparison with the others. The number of decision trees built was 500.

Loss function in our Random Forest model:

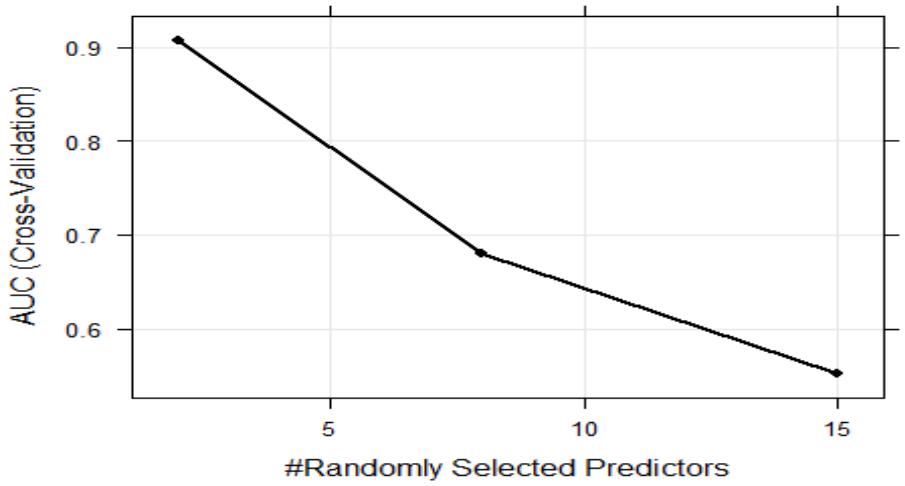
The loss function here was the mean decrease in information gain when a feature of the radio mast was removed from the model. The larger the decrease in information gain, the higher the importance of the feature By this, the top 5 important features were FlatfademarginmultipathdB1, R_powerfd1, MainreceivesignaldBm1, dbperKmRatio and PathlengthKm. Among them when FlatfademarginmultipathdB1 alone was removed, the information loss was about 90.

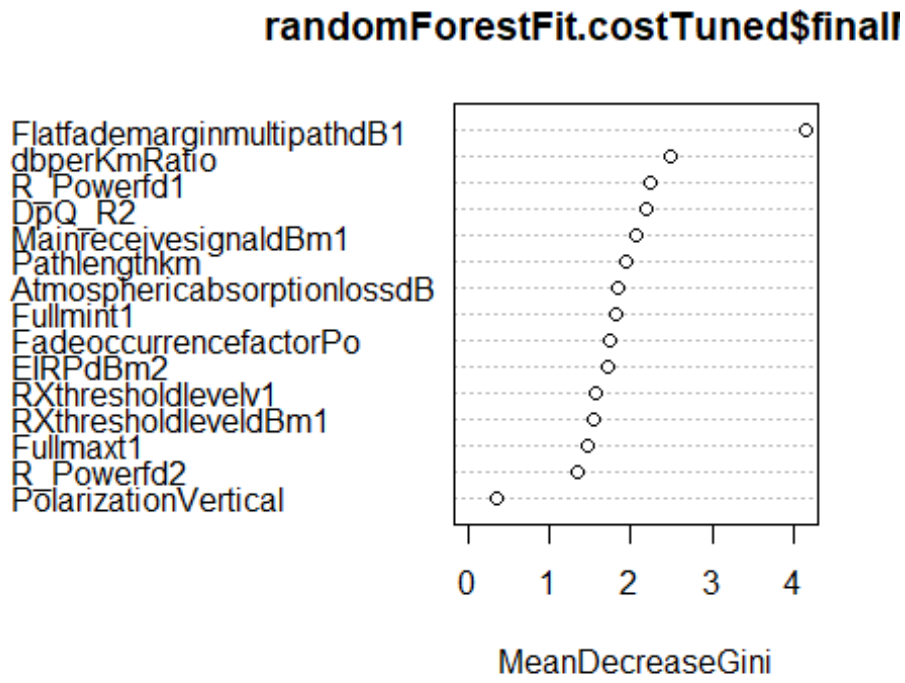
e) Cost-Tuning in Random Forest model:

Fitting the cost tuned random forest model. According to the costs given (1:h where h could be 8,16 or 24), 1:8 ratio was taken, so the samplesize parameter was set accordingly as sampsize=c(16,128), as there were 1640 samples in the train set. 1% is about 16 and 8% is about 128. In this way, we had given more weight to the rare class i.e., under engineered radio masts. As we could see in the results and graphs below, the AUC got reduced (0.794 from 0.9422, but specificity increased to 1 from 0.79) after cost tuning as the model gave more importance to reducing the number of false positives (the number of radio masts incorrectly scoreda as okay when it is under) as was required by Daniel. But this was done at the cost of model's reduced capability to find the true positives.

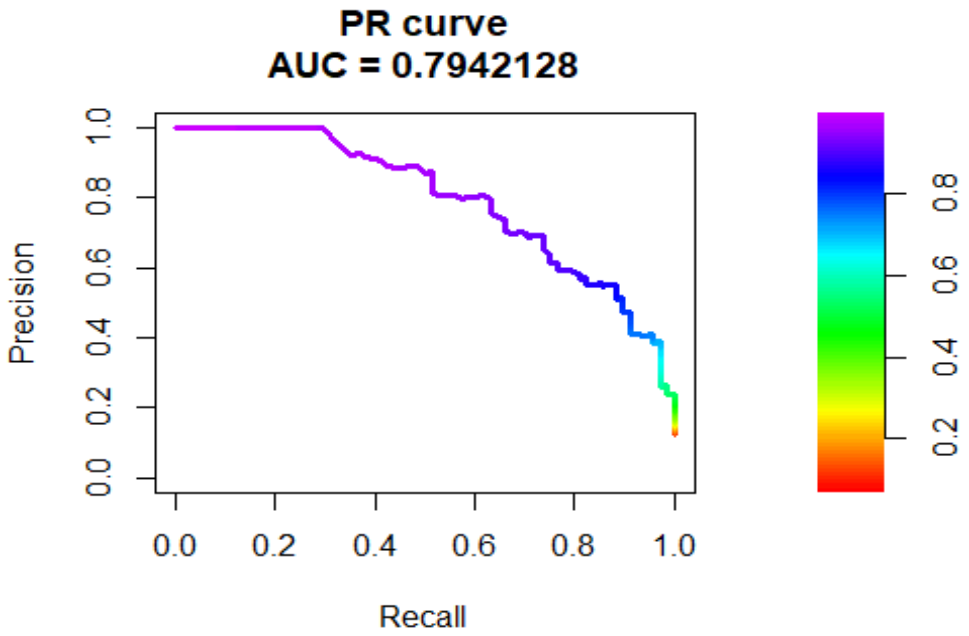
```
## Random Forest
##
## 1640 samples
##   15 predictor
##    2 classes: 'okay', 'under'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
```

```
## Summary of sample sizes: 1477, 1476, 1477, 1476, 1476, 1476, ...
## Resampling results across tuning parameters:
##
##   mtry  AUC        Precision  Recall     F
##    2    0.9077352  0.9424492  0.9881507  0.9646558
##    8    0.6807236  0.9530798  0.9895542  0.9709302
##   15    0.5514780  0.9538383  0.9909479  0.9719841
##
## AUC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```





```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction okay under
##      okay   251     0
##      under  227    68
##
##                Accuracy : 0.5842
##                  95% CI : (0.5416, 0.626)
##     No Information Rate : 0.8755
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2159
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.5251
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.2305
##              Prevalence : 0.8755
##          Detection Rate : 0.4597
##    Detection Prevalence : 0.4597
##       Balanced Accuracy : 0.7626
##
##        'Positive' Class : okay
##
```



f) The scoring set records had been classified using the cost tuned Random Forest model. Out of 936 radio masts, 432 had been classified as okay and 504 was classified as under engineered.

```
##
##  Count of labels in each class

## randomForestTest.scoringPredictions
##  okay under
##   432   504
```

The scoring set records had also been classified using the finalised model Random Forest without cost tuning. Out of 936 radio masts, 851 had been classified as okay and 85 was classified as under engineered.

```
##
##  Count of labels in each class

## randomForestTest.scoringPredictions2
##  okay under
##   851    85
```

4) References:

[1] Francois Chollet. (2017). Deep Learning with R. MEAP Edition. Manning Early Access Program. Version 1.

[2] Hidaka, Akinori & Kurita, Takio. (2017). Consecutive Dimensionality Reduction by Canonical Correlation Analysis for Visualization of Convolutional Neural Networks. Proceedings of the ISCIE International Symposium on Stochastic Systems Theory and its Applications. 2017. 160-167. 10.5687/sss.2017.160.

[3] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[4] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4349800/

[5] http://www.blogspot.udec.ugto.saedsayad.com/docs/ROC101.pdf