IDS 572 Assignment 1 Part B

Lauren Sansone, Joshua Pollack, Lina Quiceno Bejarano

Variable Modifications: Remove loans with a status other than charged off and fully paid, changing emp_length to factor

5 Develop decision tree models to predict default.

(a) **Split the data into training and validation sets. What proportions do you consider, why?**

The Lending Club data was separated into three sets to build a decision tree model: training, validation and test to help build an accurate model. The training set was split into 70% of the data, using the majority of the data to train the model on predicting loan status. The validation set was split into 10% of the data, to cross evaluate the performance of the training data. The test set was split into 20% of the data, to test how well the model predicts loan status after training and validation. Set.seed set to 200 to generate the random number sequence.

5 (b) Train decision tree models (use both rpart, c50)

The decision tree model was trained using rpart and C50. Variables due to leakage were removed including debt settlement variables, hardship variables, payment amount variables, account balance variables, late fee variables, payment plan variables and others that would not be available at the time of the loan. The data set was cleaned, removing variables that could cause bias or were not necessary to predict loan status. Variables were removed if they had more than 60% of missing variables, including employment length and the X1 variable. Some variables with fewer missing values were replaced with median values such as months since recent inquiry. A derived attribute was created for the proportion of bank cards in satisfactory standing.

The rpart model was initially trained using the information index and min. split of 30. The model resulted in over fit, with very high accuracy and no charged off predictions. The same result was achieved when changing the model to the gini index. A complexity parameter was added, beginning with 0.0001. The decision tree had about an 85% accuracy but the tree was so large, it could not even be plotted – a result of extreme over fit. The cp values were experimented with and decreased, resulting in slightly smaller decision trees.

A summary of the training data displayed a total much larger number of fully paid loans than charged off loans. Since the totals were unbalanced, 3 to 1 weights were added to create a more balanced data set and encourage the model to predict charge offs.

A print out of the complexity parameters were given to find the best tree. The cp value within one standard deviation of the lowest x-error was determined to be 0.05 and incorporated in the training model to prune the tree. Before pruning, the training data predicted loan status with 80.7% accuracy and 88% specificity. After pruning, the accuracy increased to 81.4% and specificity of 90.7%. Since it was above 80%, the cross-validation data was run.

With the same parameters, the cross validation predicted 79.02% accuracy and 89.2% specificity. The test data was then run and predicted 77% accuracy and 90% specificity (precision). The performance of the tests was based on accuracy and specificity.

The performance of the model was evaluated using an ROC curve. The initial run of the curve had fairly poor separation and was mostly convex with small concavities. The curve was then lifted into 10 groups to evaluate performance.

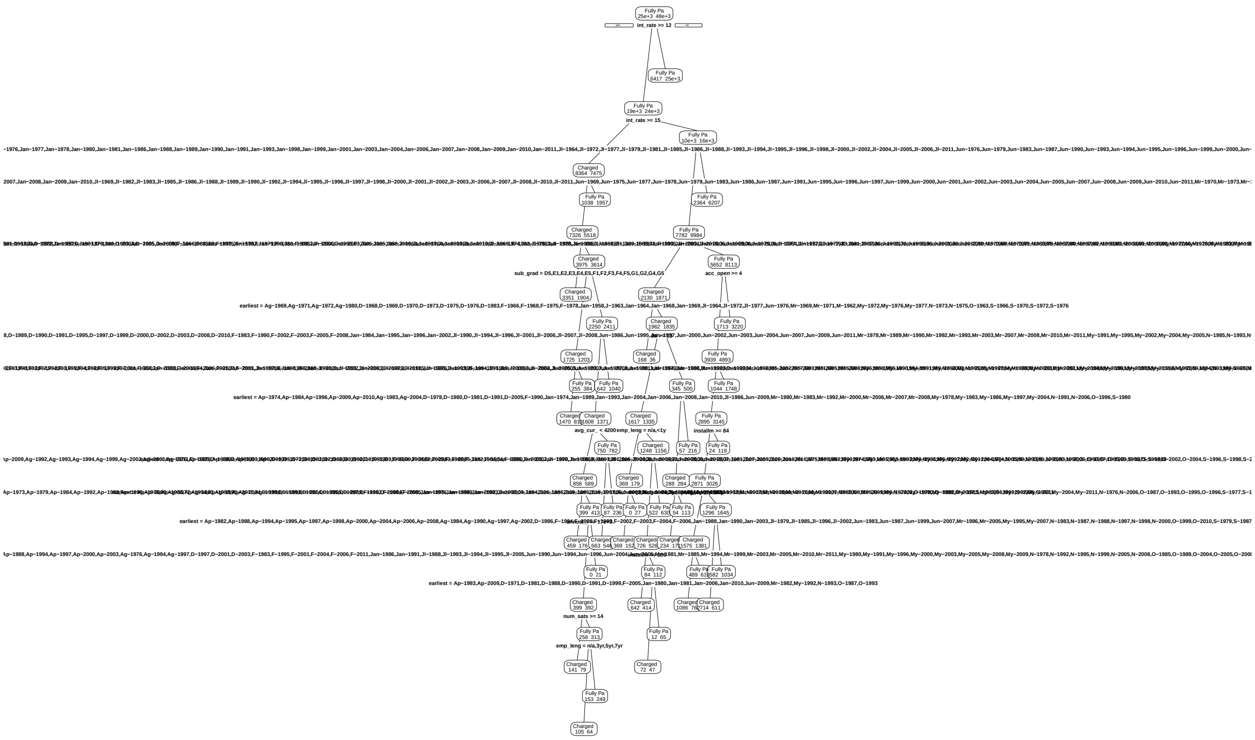
Creating a Weighted Tree For the Training Set

```
myweights = ifelse(lcdfTrn$loan_status == "Charged Off", 3, 1 )

Wghtd_lcDT <- rpart(loan_status ~., data=lcdfTrn, method="class", weights = myweights, parms = list(spli

pred_wghtTrn=predict(Wghtd_lcDT,lcdfTrn, type='class')

#Confusion table
confusionMatrix(table(predWghtTrain = pred_wghtTrn, true=lcdfTrn$loan_status))
```

```
## Confusion Matrix and Statistics
##
##              true
## predWghtTrain Charged Off Fully Paid
##   Charged Off        3158       5781
##   Fully Paid         5151      42624
##
##                Accuracy : 0.8072
##                  95% CI : (0.804, 0.8105)
##     No Information Rate : 0.8535
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2527
##
##  Mcnemar's Test P-Value : 1.789e-09
##
##             Sensitivity : 0.38007
##             Specificity : 0.88057
##          Pos Pred Value : 0.35328
##          Neg Pred Value : 0.89218
##              Prevalence : 0.14651
##          Detection Rate : 0.05568
##    Detection Prevalence : 0.15762
##       Balanced Accuracy : 0.63032
##
##        'Positive' Class : Charged Off
##
```

Display the rpart Tree for training set

```
rpart.plot::prp(Wghtd_lcDT, type=2, extra=1)
```



Summary of lcDT

```
Call:
rpart(formula = loan_status ~ ., data = lcdfTrn, weights = myweights,
    method = "class", parms = list(split = "information"), control =
rpart.control(cp = 0.001))
  n= 56714

            CP nsplit rel error    xerror        xstd
1  0.024177264      0 1.0000000 1.0000000 0.005145924
2  0.005322208      3 0.9274682 0.9764914 0.005115765
3  0.005195170      6 0.9115016 1.0016849 0.005148022
4  0.005175111      8 0.9011112 1.0008826 0.005147024
5  0.003109079      9 0.8959361 1.0038512 0.005150708
6  0.002988727     11 0.8897180 1.0043327 0.005151303
7  0.002166326     13 0.8837405 1.0046937 0.005151749
8  0.001945681     15 0.8794079 1.0105508 0.005158927
9  0.001263690     21 0.8622779 1.0087857 0.005156774
10 0.001123280     23 0.8597505 1.0099089 0.005158145
11 0.001083163     24 0.8586272 1.0107514 0.005159171
12 0.001016301     25 0.8575440 1.0104305 0.005158780
13 0.001002929     29 0.8528503 1.0119549 0.005160633
14 0.001000000     30 0.8518474 1.0119549 0.005160633
```

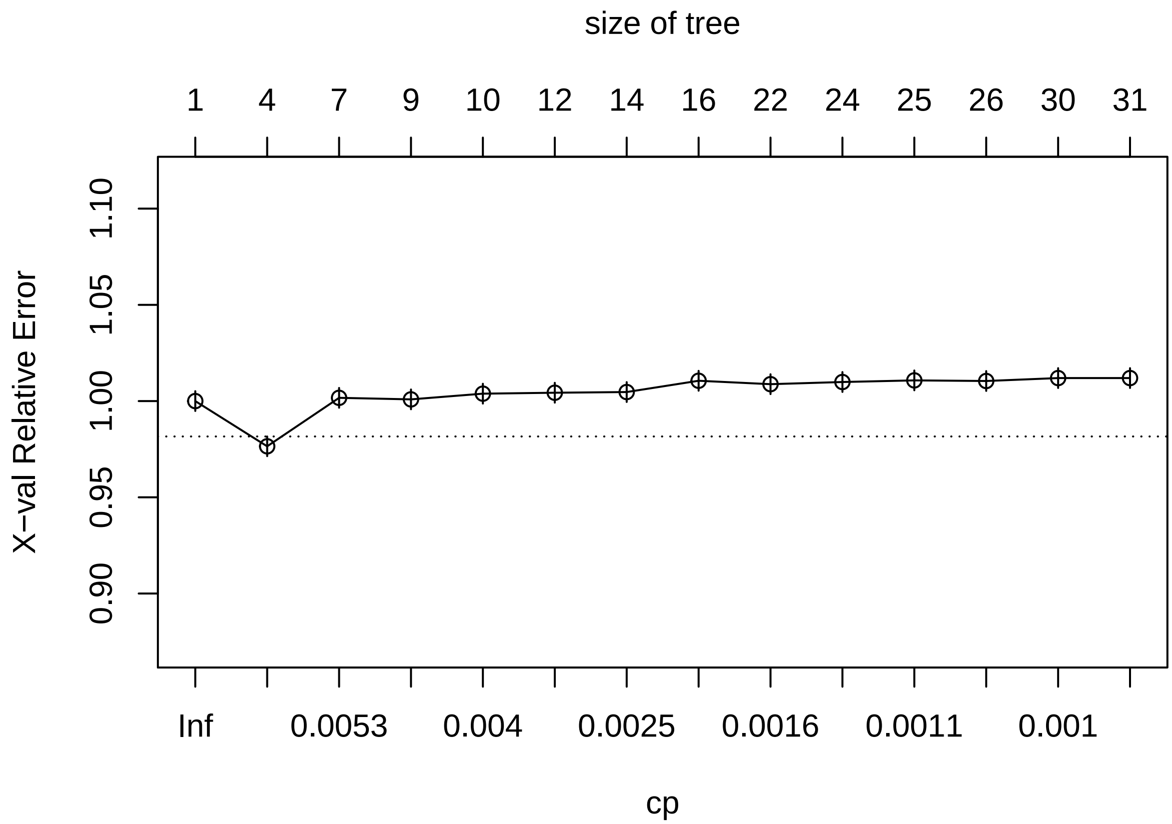
Details About the Training Set

```r
#tree size and performance for different complexity parameter values
printcp(Wghtd_lcDT)
```

```
##
## Classification tree:
```

```
## rpart(formula = loan_status ~ ., data = lcdfTrn, weights = myweights,
##     method = "class", parms = list(split = "information"), control = rpart.control(cp = 0.001))
##
## Variables actually used in tree construction:
##  [1] acc_open_past_24mths annual_inc           avg_cur_bal
##  [4] dti                  earliest_cr_line     emp_length
##  [7] installment          int_rate             num_sats
## [10] sub_grade
##
## Root node error: 24927/56714 = 0.43952
##
## n= 56714
##
##           CP nsplit rel error  xerror      xstd
## 1  0.0241773      0   1.00000 1.00000 0.0051459
## 2  0.0053222      3   0.92747 0.97649 0.0051158
## 3  0.0051952      6   0.91150 1.00168 0.0051480
## 4  0.0051751      8   0.90111 1.00088 0.0051470
## 5  0.0031091      9   0.89594 1.00385 0.0051507
## 6  0.0029887     11   0.88972 1.00433 0.0051513
## 7  0.0021663     13   0.88374 1.00469 0.0051517
## 8  0.0019457     15   0.87941 1.01055 0.0051589
## 9  0.0012637     21   0.86228 1.00879 0.0051568
## 10 0.0011233     23   0.85975 1.00991 0.0051581
## 11 0.0010832     24   0.85863 1.01075 0.0051592
## 12 0.0010163     25   0.85754 1.01043 0.0051588
## 13 0.0010029     29   0.85285 1.01195 0.0051606
## 14 0.0010000     30   0.85185 1.01195 0.0051606
```

```
#Plot
plotcp(Wghtd_lcDT)
```




```
#Variable importance as given by a decision tree model
Wghtd_lcDT$variable.importance
```

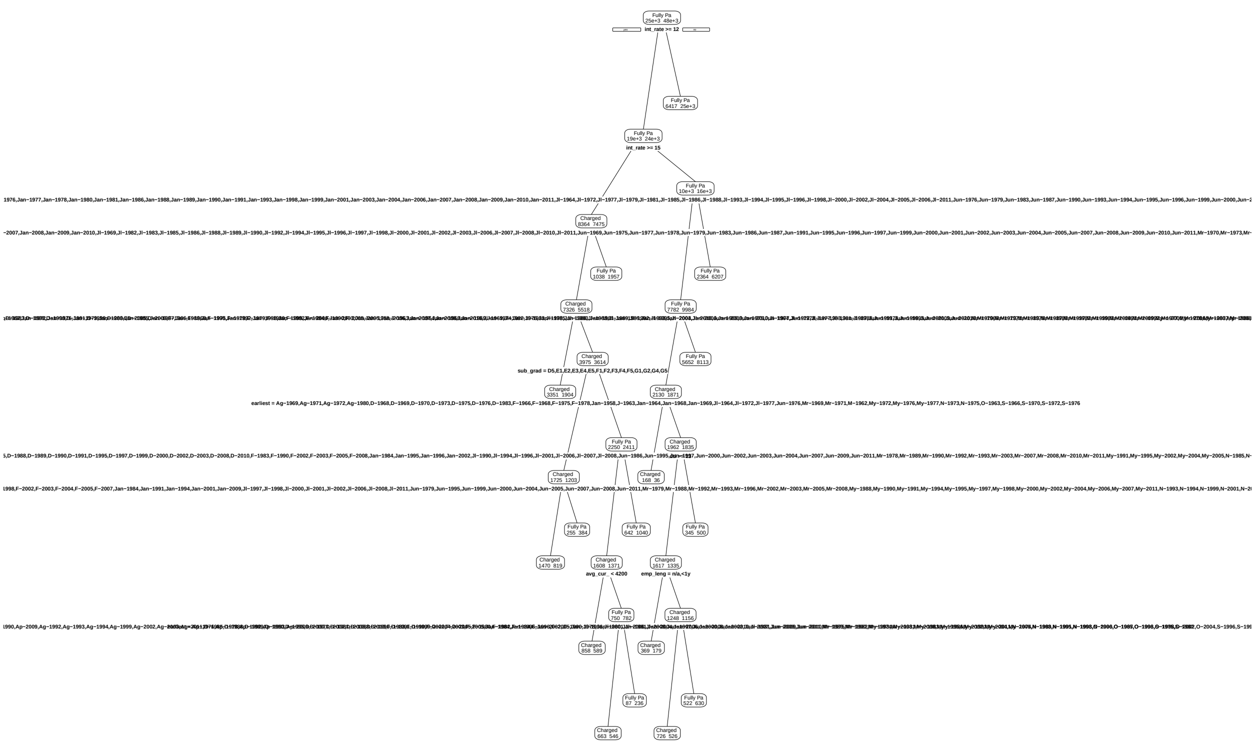
```
##                  int_rate                sub_grade
##              2691.4752931             2287.6435343
##                     grade         earliest_cr_line
##              1918.5165763             1224.5864403
##                bc_open_to_buy           total_bc_limit
##               473.4999214              405.5368413
##            total_rev_hi_lim     acc_open_past_24mths
##               331.4927267               64.8185512
##               installment               emp_length
##                64.3916895               57.8532145
##        mo_sin_old_rev_tl_op              avg_cur_bal
##                57.3582421               44.7476097
##                 loan_amnt             tot_hi_cred_lim
##                43.4657633               37.9086366
##                       dti         num_tl_op_past_12m
##                34.4490394               34.4139499
##          total_bal_ex_mort                 num_sats
##                29.4843837               27.1732068
##                  mort_acc           home_ownership
##                22.8458601               20.2000658
## total_il_high_credit_limit     mo_sin_rcnt_rev_tl_op
##                19.9855626               17.1194350
##                annual_inc           mo_sin_rcnt_tl
##                16.6797016               14.9495017
##        mths_since_recent_bc        mo_sin_old_il_acct
```

```
##                 13.7747058                 12.1795358
##                    purpose               num_op_rev_tl
##                 12.1711694                  7.7797112
##            num_actv_rev_tl          num_rev_tl_bal_gt_0
##                  4.7089597                  4.7071841
##               num_rev_accts                  num_bc_tl
##                  3.0915479                  2.5854790
##                pct_tl_nvr_dlq            percent_bc_gt_75
##                  2.3126132                  2.1498416
##      propSatisBankcardAccts                num_bc_sats
##                  2.0298433                  1.0148114
##       pub_rec_bankruptcies                  num_il_tl
##                  0.5082239                  0.4139405
##                    bc_util      num_accts_ever_120_pd
##                  0.3726118                  0.2691630
```

Prune Tree based on cp

```r
prn_lcDT <- prune(Wghtd_lcDT, cp=0.0019664)
```

```r
rpart.plot::prp(prn_lcDT, type=2, extra=1)
```



Check Performance for Validation Set

```r
#Evaluate performance
predVal=predict(prn_lcDT,lcdfVal, type='class')
table(predictValidation = predVal, true=lcdfVal$loan_status)
```

```
##                  true
## predictValidation Charged Off Fully Paid
##       Charged Off         245        814
##       Fully Paid          886       6157
```

```r
mean(predVal == lcdfVal$loan_status)
```

```
## [1] 0.7901753
```

```r
#Confusion table
confusionMatrix(table(predictValidation = predVal, true=lcdfVal$loan_status))
```

```
## Confusion Matrix and Statistics
##
##                    true
## predictValidation Charged Off Fully Paid
##       Charged Off         245        814
##       Fully Paid          886       6157
##
##                Accuracy : 0.7902
##                  95% CI : (0.7811, 0.799)
##     No Information Rate : 0.8604
##     P-Value [Acc > NIR] : 1.00000
##
##                   Kappa : 0.1026
##
##  Mcnemar's Test P-Value : 0.08507
##
##             Sensitivity : 0.21662
##             Specificity : 0.88323
##          Pos Pred Value : 0.23135
##          Neg Pred Value : 0.87420
##              Prevalence : 0.13960
##          Detection Rate : 0.03024
##    Detection Prevalence : 0.13071
##       Balanced Accuracy : 0.54993
##
##        'Positive' Class : Charged Off
##
```

Check the Performance of The Test Set

```r
#Evaluate performance on the test set
predTst=predict(prn_lcDT, lcdfTst, type = 'class')
table(predictTest = predTst, true=lcdfTst$loan_status)
```

```
##              true
## predictTest   Charged Off Fully Paid
##   Charged Off         521       1493
##   Fully Paid         1866      12325
```

```r
mean(predTst ==lcdfTst$loan_status)
```

```
## [1] 0.7927183
```

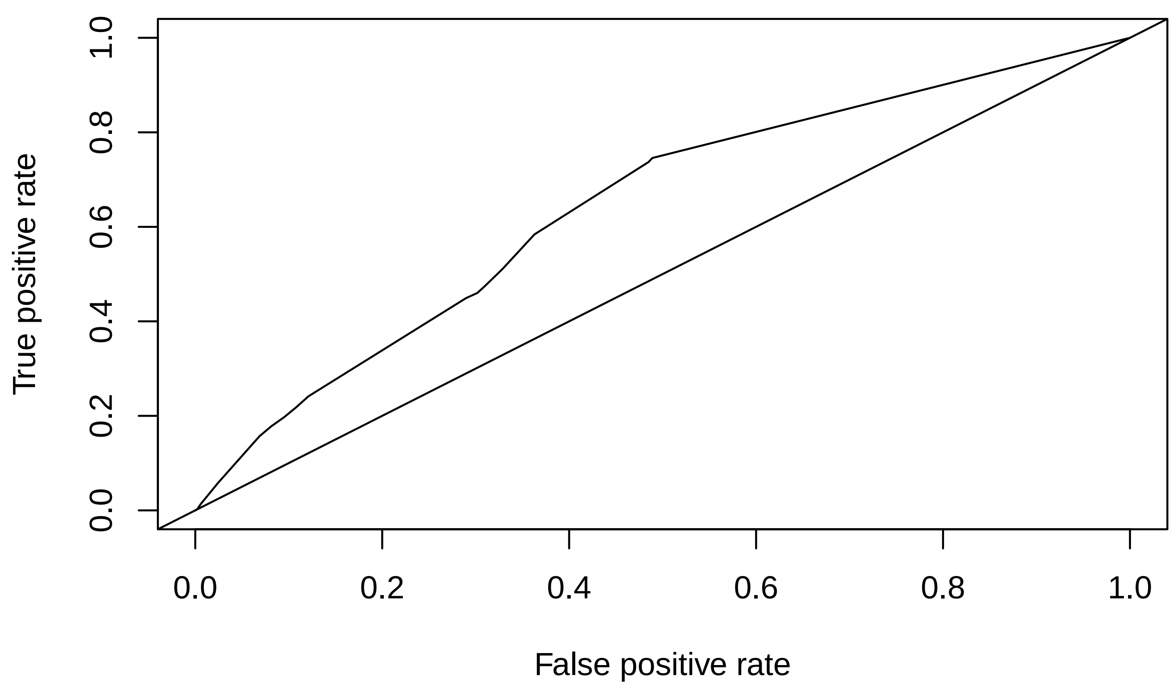
```r
confusionMatrix(table(predictTest = predTst, true=lcdfTst$loan_status))
```

```
## Confusion Matrix and Statistics
##
##              true
## predictTest   Charged Off Fully Paid
##   Charged Off         521       1493
```

```
##   Fully Paid         1866      12325
##
##                Accuracy : 0.7927
##                  95% CI : (0.7864, 0.7989)
##     No Information Rate : 0.8527
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1178
##
##  Mcnemar's Test P-Value : 1.376e-10
##
##             Sensitivity : 0.21827
##             Specificity : 0.89195
##          Pos Pred Value : 0.25869
##          Neg Pred Value : 0.86851
##              Prevalence : 0.14730
##          Detection Rate : 0.03215
##    Detection Prevalence : 0.12428
##       Balanced Accuracy : 0.55511
##
##        'Positive' Class : Charged Off
##
```

ROCR For Weighted Rpart Tree

```
scoreTst=predict(prn_lcDT, lcdfTst, type="prob")[,'Charged Off']

#apply the prediction function from ROCR to get a prediction object
rocPredTst = prediction(scoreTst, lcdfTst$loan_status, label.ordering = c('Fully Paid', 'Charged Off'))

perfROCTst=performance(rocPredTst, "tpr", "fpr")
plot(perfROCTst)
abline(0,1)
```



Lifts for Weighted Rpart tree

```
# 'scores' from applying the model to the data
predTrnProb=predict(prn_lcDT, lcdfTrn, type='prob')
head(predTrnProb)
```

```
##   Charged Off Fully Paid
## 1   0.2758138  0.7241862
## 2   0.2059635  0.7940365
## 3   0.2758138  0.7241862
## 4   0.2758138  0.7241862
## 5   0.2059635  0.7940365
## 6   0.5798722  0.4201278
```

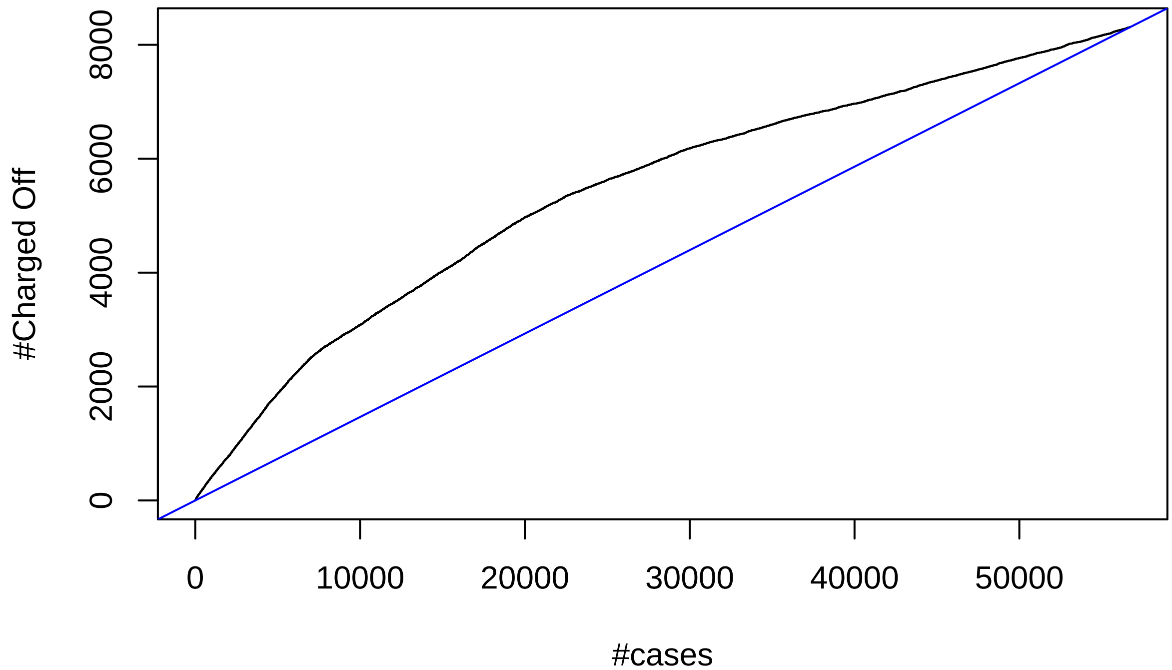
```
#Create a data-frame with only the model scores and the actual class
trnSc <- lcdfTrn %>%  select("loan_status")
trnSc$score<-predTrnProb[, 1]

#take a look at trnSc
head(trnSc)
```

```
## # A tibble: 6 x 2
##   loan_status score
##   <fct>       <dbl>
## 1 Fully Paid  0.276
## 2 Fully Paid  0.206
## 3 Fully Paid  0.276
## 4 Fully Paid  0.276
## 5 Fully Paid  0.206
## 6 Fully Paid  0.580
```

```
#sort by score
trnSc<-trnSc[order(trnSc$score, decreasing=TRUE),]

#generate the cumulative sum of "default" OUTCOME values
trnSc$cumDefault<-cumsum(trnSc$loan_status == "Charged Off")

#first 10 row in trnSc
trnSc[1:10,]
```

```
## # A tibble: 10 x 3
##    loan_status score cumDefault
##    <fct>       <dbl>      <int>
##  1 Fully Paid  0.824          0
##  2 Charged Off 0.824          1
##  3 Charged Off 0.824          2
##  4 Charged Off 0.824          3
##  5 Charged Off 0.824          4
##  6 Fully Paid  0.824          4
##  7 Fully Paid  0.824          4
##  8 Fully Paid  0.824          4
##  9 Charged Off 0.824          5
## 10 Charged Off 0.824          6
```

```
#Plot the cumDefault values (y-axis) by numCases (x-axis)
plot( trnSc$cumDefault, type = "l", xlab='#cases', ylab='#Charged Off')
abline(0,max(trnSc$cumDefault)/56714, col="blue")  #diagonal line
```

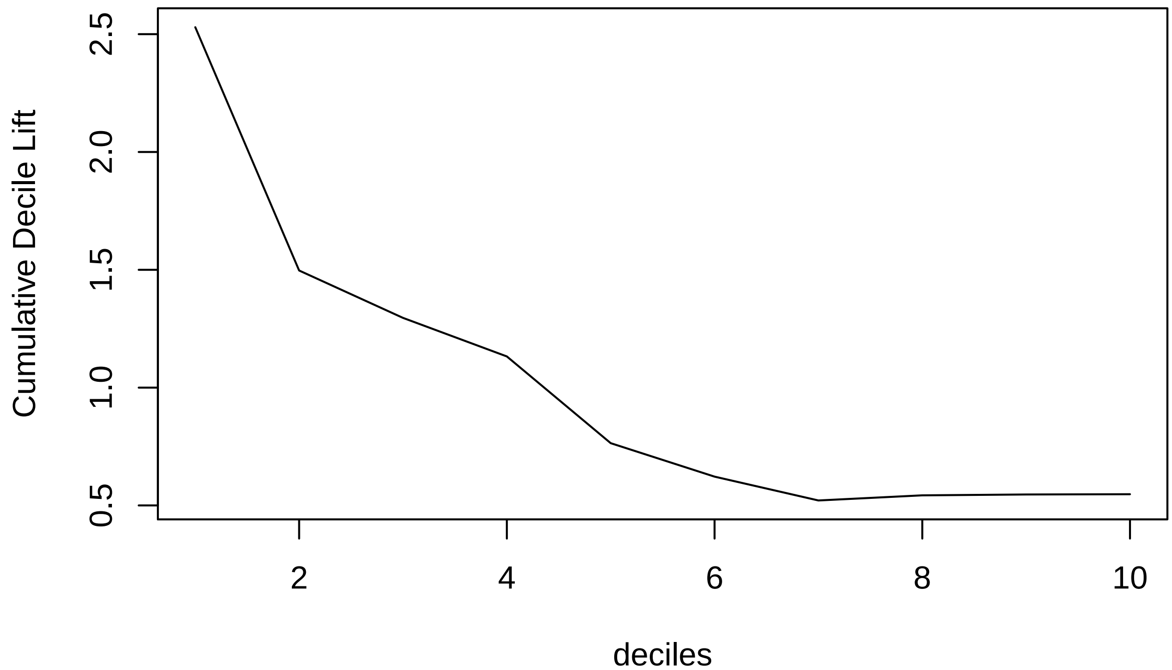


Calculate the decile lift table.

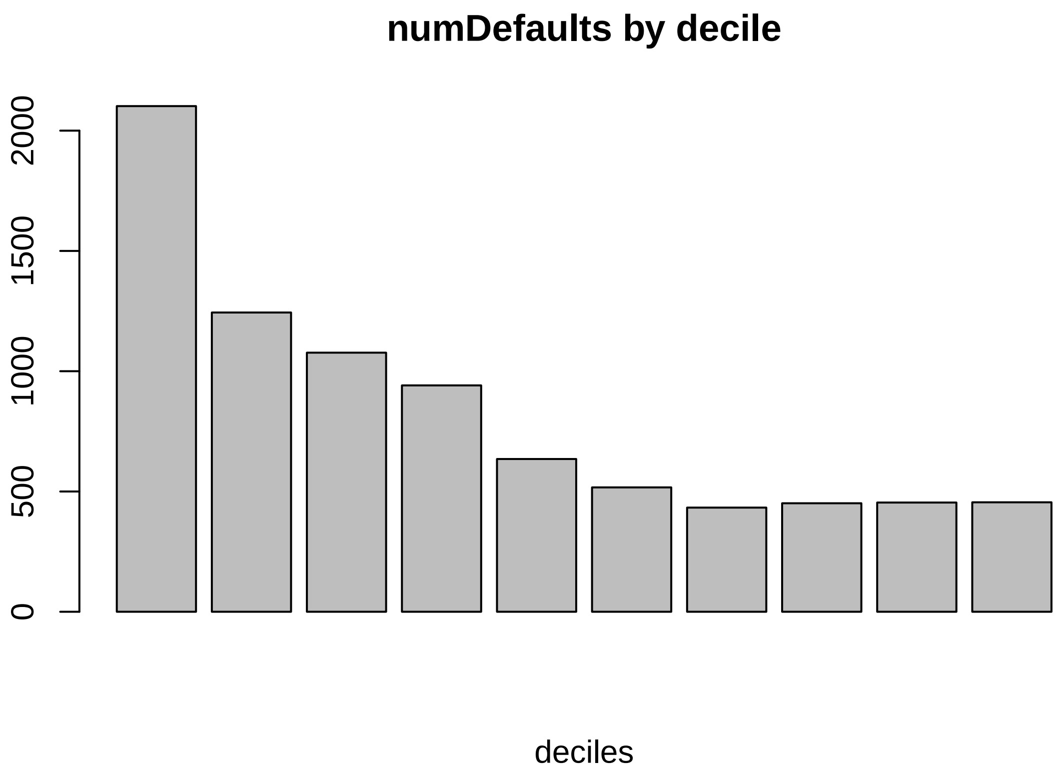
```
#Divide the data into 10 for decile lift equal groups
trnSc["bucket"]<- ntile(-trnSc[,"score"], 10)

#group the data by the 'buckets', and obtain summary statistics
dLifts <- trnSc %>% group_by(bucket) %>% summarize(count=n(), numDefaults=sum(loan_status=="Charged Off
              defRate=numDefaults/count,  cumDefRate=cumsum(numDefaults)/cumsum(count),
              lift = cumDefRate/(sum(trnSc$loan_status=="Charged Off")/nrow(trnSc)) )

#look at the table
dLifts
```

```
## # A tibble: 10 x 6
##    bucket count numDefaults defRate cumDefRate  lift
##  *  <int> <int>       <int>   <dbl>      <dbl> <dbl>
##  1      1  5672        2102  0.371      0.371  2.53
##  2      2  5672        1244  0.219      0.219  1.50
##  3      3  5672        1077  0.190      0.190  1.30
##  4      4  5672         941  0.166      0.166  1.13
##  5      5  5671         635  0.112      0.112  0.764
##  6      6  5671         517  0.0912     0.0912 0.622
##  7      7  5671         433  0.0764     0.0764 0.521
##  8      8  5671         451  0.0795     0.0795 0.543
##  9      9  5671         454  0.0801     0.0801 0.546
## 10     10  5671         455  0.0802     0.0802 0.548
```

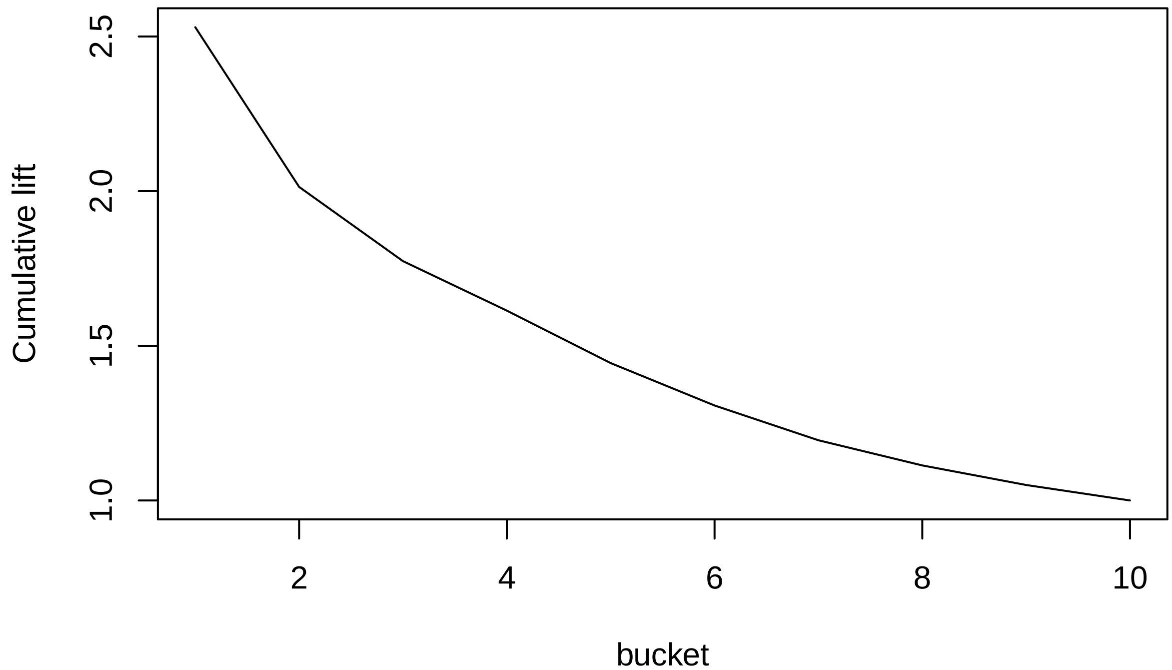
```
#various plots,
plot(dLifts$bucket, dLifts$lift, xlab="deciles", ylab="Cumulative Decile Lift", type="l")
```




```
barplot(dLifts$numDefaults, main="numDefaults by decile", xlab="deciles")
```




```
library('lift')

plotLift(trnSc$score, trnSc$loan_status == "Charged Off")
```




```r
#value of lift in the top decile
TopDecileLift(trnSc$score, trnSc$loan_status)
```

```
## [1] NA
```

5 b) continued - C50 Tree

The same model process used for rpart was followed for C50 using the separate training, validation and test sets, and the data was weighted to balance the fully paid and charged off. The training data confusion matrix resulted in 79.71% accuracy and 88.14% specificity. As the training data was at about 80%, the cross validation was run and resulted in 79.1% accuracy and 87.5% specificity. The test model resulted in 78.96% accuracy and 87.9% specificity. The roc curve was mostly convex with fairly poor separation, then the curve was lifted to evaluate performance.

C50 Tree

```r
#build a tree model
c5_DT1 <- C5.0(loan_status ~ ., data=lcdfTrn, control=C5.0Control(minCases=50), weights = myweights)
```

Prediction Train, Val, and Test all at once

```r
predTrnProb_c5dt1 <- predict(c5_DT1, lcdfTrn, type='class')
predValProb_c5dt1 <- predict(c5_DT1, lcdfVal, type='class')
predTstProb_c5dt1 <- predict(c5_DT1, lcdfTst, type='class')

#Training
mean(predTrnProb_c5dt1==lcdfTrn$loan_status)
```

```
## [1] 0.7971224
```

```r
#Validation
mean(predValProb_c5dt1==lcdfVal$loan_status)
```

```
## [1] 0.7910392
```

```r
#Test
mean(predTstProb_c5dt1==lcdfTst$loan_status)
```

```
## [1] 0.7895711
```

Predictions for Training

```r
#mehtod 2
predTrnProb_c5dt1 <- predict(c5_DT1, lcdfTrn, type='class')
confusionMatrix(table(predictC50Train = predTrnProb_c5dt1, true=lcdfTrn$loan_status))
```

```
## Confusion Matrix and Statistics
##
##                true
## predictC50Train Charged Off Fully Paid
##     Charged Off        2545       5742
##     Fully Paid         5764      42663
##
##                Accuracy : 0.7971
##                  95% CI : (0.7938, 0.8004)
##     No Information Rate : 0.8535
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : 0.1879
##
##  Mcnemar's Test P-Value : 0.8448
##
##             Sensitivity : 0.30629
##             Specificity : 0.88138
##          Pos Pred Value : 0.30711
##          Neg Pred Value : 0.88098
##              Prevalence : 0.14651
##          Detection Rate : 0.04487
##    Detection Prevalence : 0.14612
##       Balanced Accuracy : 0.59384
##
##        'Positive' Class : Charged Off
##
```

```r
mean(predTrnProb_c5dt1==lcdfTrn$loan_status)
```

```
## [1] 0.7971224
```

Predictions for Validation

```r
predValProb_c5dt1 <- predict(c5_DT1, lcdfVal, type='class')
confusionMatrix(table(predictC50Validation = predValProb_c5dt1, true=lcdfVal$loan_status))
```

```
## Confusion Matrix and Statistics
##
##                     true
## predictC50Validation Charged Off Fully Paid
##          Charged Off         307        869
##          Fully Paid          824       6102
##
##                Accuracy : 0.791
```

```
##                  95% CI : (0.782, 0.7998)
##     No Information Rate : 0.8604
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : 0.1444
##
##  Mcnemar's Test P-Value : 0.2849
##
##             Sensitivity : 0.27144
##             Specificity : 0.87534
##          Pos Pred Value : 0.26105
##          Neg Pred Value : 0.88103
##              Prevalence : 0.13960
##          Detection Rate : 0.03789
##    Detection Prevalence : 0.14515
##       Balanced Accuracy : 0.57339
##
##        'Positive' Class : Charged Off
##
```

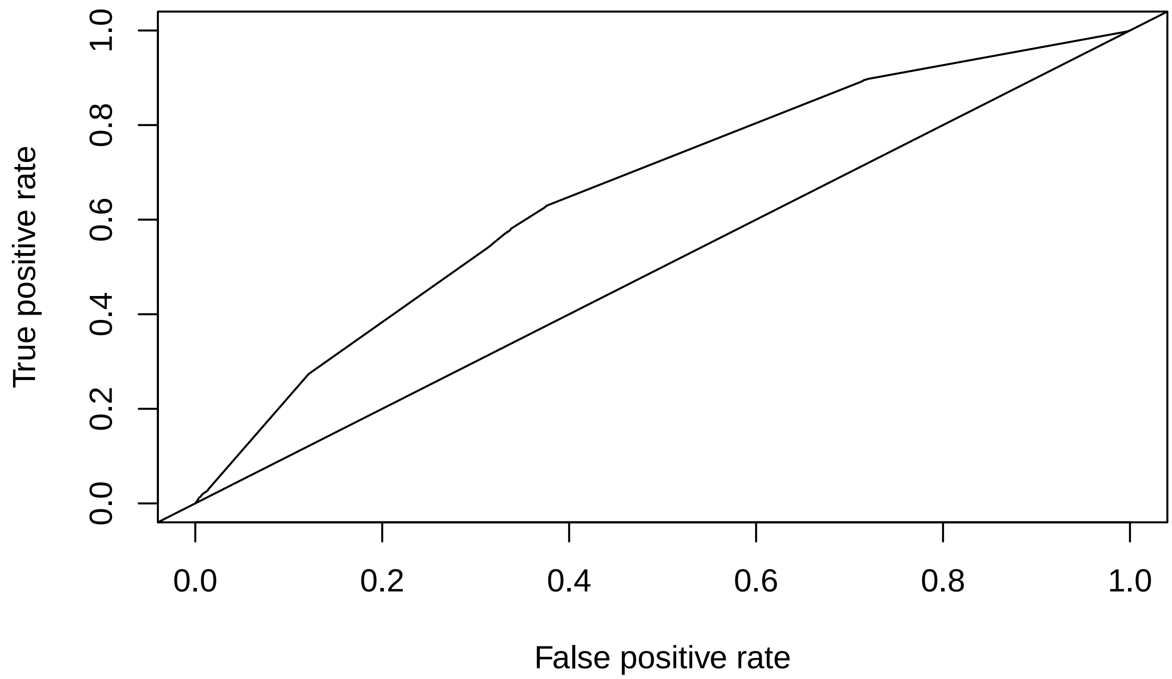
Predictions for Test

```
predTstProb_c5dt1 <- predict(c5_DT1, lcdfTst, type='class')
confusionMatrix(table(predictC50Test = predTstProb_c5dt1, true=lcdfTst$loan_status))
```

```
## Confusion Matrix and Statistics
##
##                true
## predictC50Test Charged Off Fully Paid
##    Charged Off         653       1676
##    Fully Paid         1734      12142
##
##                Accuracy : 0.7896
##                  95% CI : (0.7832, 0.7958)
##     No Information Rate : 0.8527
##     P-Value [Acc > NIR] : 1.000
##
##                   Kappa : 0.1538
##
##  Mcnemar's Test P-Value : 0.329
##
##             Sensitivity : 0.2736
##             Specificity : 0.8787
##          Pos Pred Value : 0.2804
##          Neg Pred Value : 0.8750
##              Prevalence : 0.1473
##          Detection Rate : 0.0403
##    Detection Prevalence : 0.1437
##       Balanced Accuracy : 0.5761
##
##        'Positive' Class : Charged Off
##
```

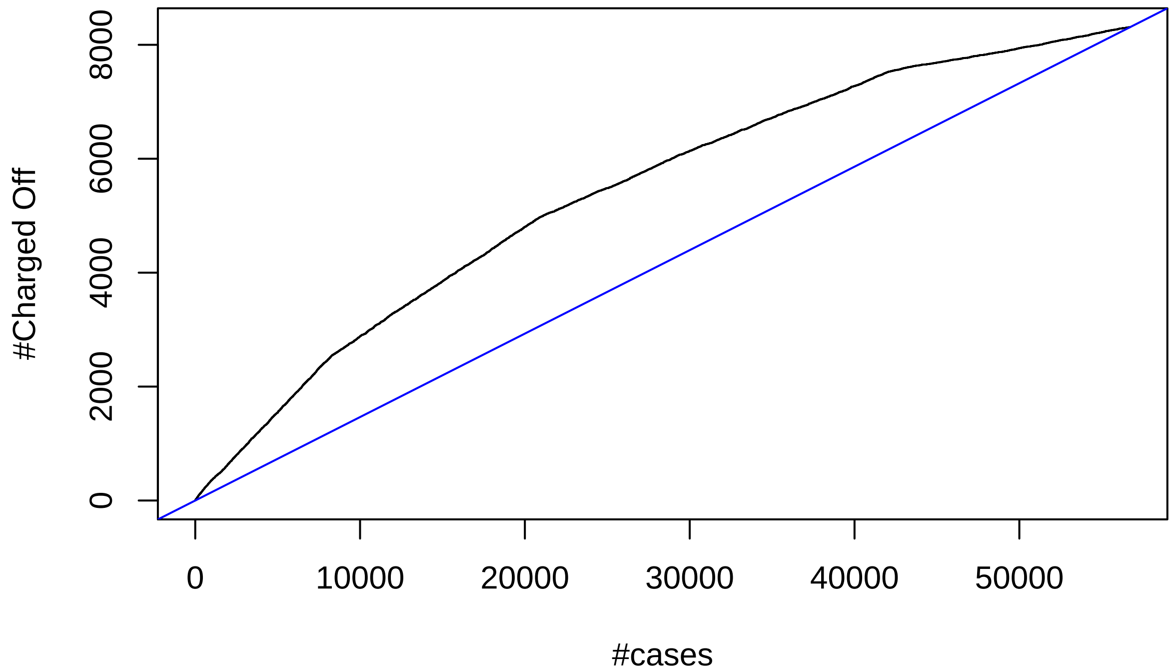
ROCR For Weighted C50 Tree

```r
#obtain the scores from the model for the class of interest
c5scoreTst=predict(c5_DT1, lcdfTst, type="prob")[,'Charged Off']

# apply the prediction function from ROCR to get a prediction object
c5rocPredTst = prediction(c5scoreTst, lcdfTst$loan_status, label.ordering = c('Fully Paid', 'Charged Off

c5perfROCTst=performance(c5rocPredTst, "tpr", "fpr")
plot(c5perfROCTst)
abline(0,1)
```



Lifts for Weighted Rpart tree

```r
#get the 'scores' from applying the model to the data
c5predTrnProb=predict(c5_DT1, lcdfTrn, type='prob')

c5trnSc <- lcdfTrn %>%  select("loan_status")   # selects the OUTCOME column into trnSc
c5trnSc$score<-c5predTrnProb[, 1]

#sort by score
c5trnSc<-c5trnSc[order(c5trnSc$score, decreasing=TRUE),]

#generate the cumulative sum of "default" OUTCOME values
c5trnSc$cumDefault<-cumsum(c5trnSc$loan_status == "Charged Off")


#Plot the cumDefault values (y-axis) by numCases (x-axis)
plot( c5trnSc$cumDefault, type = "l", xlab='#cases', ylab='#Charged Off')
abline(0,max(c5trnSc$cumDefault)/56714, col="blue")  #diagonal line
```

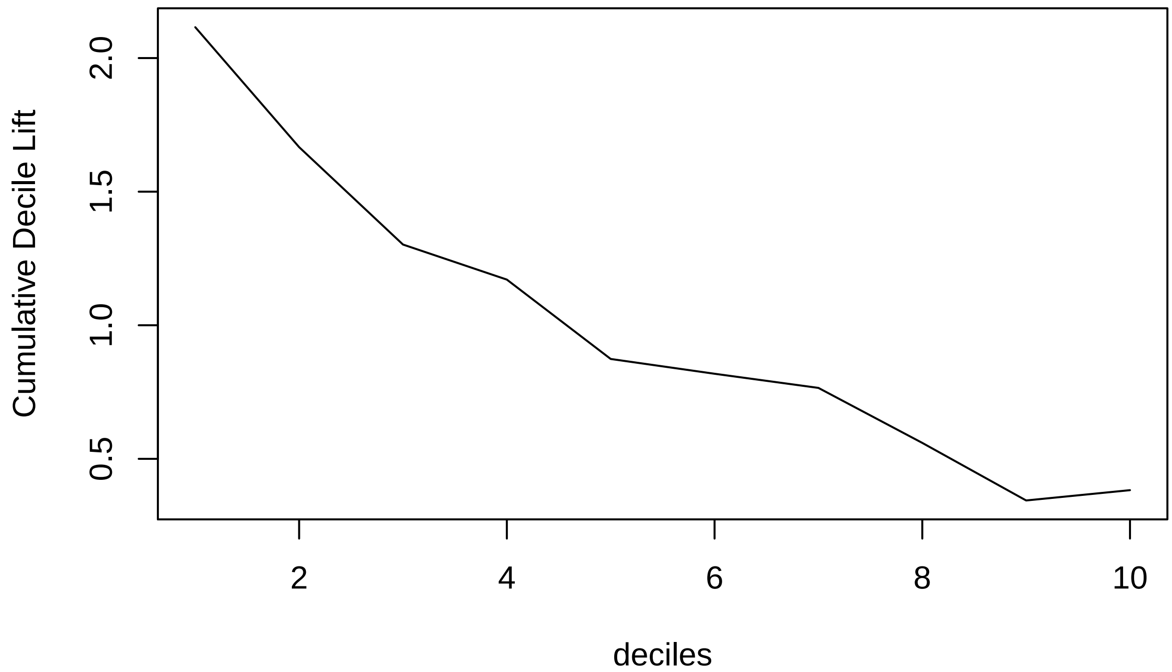


Calculate the decile lift table.

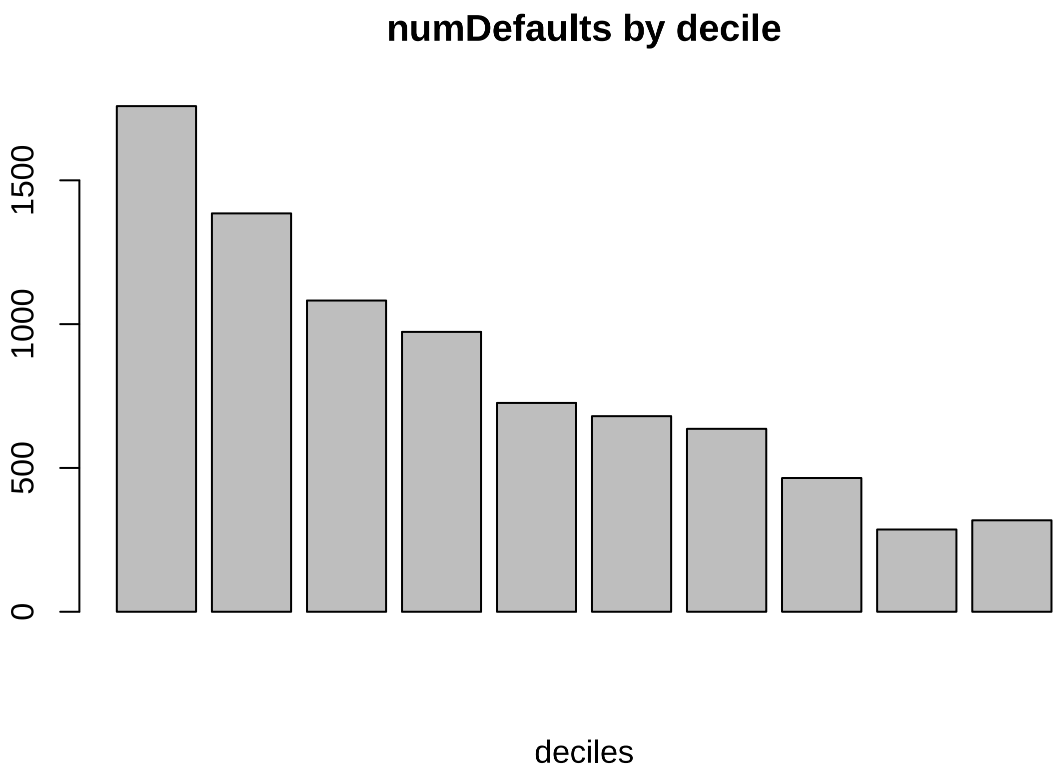
```
#Divide the data into 10 (for decile lift) equal groups
c5trnSc["bucket"]<- ntile(-c5trnSc[,"score"], 10)

#group the data by the 'buckets', and obtain summary statistics
c5dLifts <- c5trnSc %>% group_by(bucket) %>% summarize(count=n(), numDefaults=sum(loan_status=="Charged
              defRate=numDefaults/count,  cumDefRate=cumsum(numDefaults)/cumsum(count),
              lift = cumDefRate/(sum(c5trnSc$loan_status=="Charged Off")/nrow(c5trnSc)) )

#look at the table
c5dLifts
```

```
## # A tibble: 10 x 6
##    bucket count numDefaults defRate cumDefRate  lift
##  *  <int> <int>       <int>   <dbl>      <dbl> <dbl>
##  1      1  5672        1758  0.310      0.310  2.12
##  2      2  5672        1385  0.244      0.244  1.67
##  3      3  5672        1082  0.191      0.191  1.30
##  4      4  5672         973  0.172      0.172  1.17
##  5      5  5671         726  0.128      0.128  0.874
##  6      6  5671         680  0.120      0.120  0.818
##  7      7  5671         636  0.112      0.112  0.765
##  8      8  5671         465  0.0820     0.0820 0.560
##  9      9  5671         286  0.0504     0.0504 0.344
## 10     10  5671         318  0.0561     0.0561 0.383
```

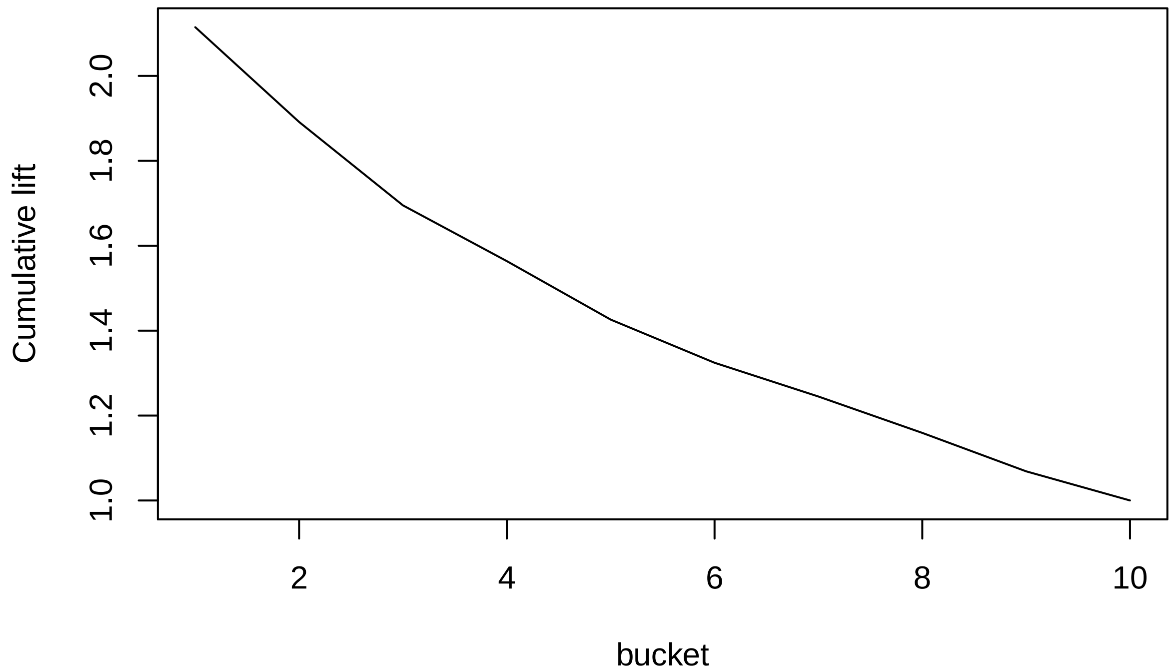
```
#you can do various plots, for example
plot(c5dLifts$bucket, c5dLifts$lift, xlab="deciles", ylab="Cumulative Decile Lift", type="l")
```




```
barplot(c5dLifts$numDefaults, main="numDefaults by decile", xlab="deciles")
```




```
plotLift(c5trnSc$score, c5trnSc$loan_status == "Charged Off")
```




```
#value of lift in the top decile
TopDecileLift(c5trnSc$score, c5trnSc$loan_status)
```

```
## Warning in TopDecileLift(c5trnSc$score, c5trnSc$loan_status): NAs introduced by
## coercion
```

```
## [1] NA
```

5 c) What is your best model?

Rpart is identified as the best decision tree model as it had consistently (although only slightly) higher accuracies for the training, validation and test sets. The size of the tree was fairly small in complexity. Variable importance was determined by the information index. The most important variable in the rpart model was interest rate, followed by subgrade and grade.

6. Develop a Random Forest Model

We decided to use the parameters min.node.size=1 for classification, importance='impurity' to use the Gini index because we are running the model for classification

Due to the imbalance of the data in loan status we used weight in ratio of 5 to 1 for charged off to be compensated. To develop the mode we use the library ranger. We decided to use the parameters min.node.size=1 for classification, importance='impurity' to use the Gini index because we are running the model for classification.Also, we used the parameter case.weights to balance the data.

We obtained accuracy of 0.85 with the model with the training set , 0.8413 on the test set and 0.8400 on the validation set. The ROC curve was used to evaluate the performance of the model.

```
#Random Forest

library(ranger)

myweights = ifelse(lcdfTrn$loan_status == "Charged Off", 5, 1)
```

```r
rgModel1 <- ranger(loan_status ~., data=lcdfTrn, num.trees =200, min.node.size=1, importance='impurity'
#We decided to use the parameters min.node.size=1 for classication, importance='impurity' to use the Gi

#variable importance
importance(rgModel1)
```

```
##                  loan_amnt                   int_rate
##                 725.626877                1270.915584
##                installment                      grade
##                 907.421801                 783.816622
##                  sub_grade                 emp_length
##                 994.349131                 500.401281
##             home_ownership                 annual_inc
##                 180.693203                 891.388135
##        verification_status                    purpose
##                 223.112197                 285.081721
##                        dti           earliest_cr_line
##                1065.277413                 899.598588
##        initial_list_status collections_12_mths_ex_med
##                 123.760840                  32.172330
##           total_rev_hi_lim       acc_open_past_24mths
##                 883.265767                 573.474678
##                avg_cur_bal             bc_open_to_buy
##                 986.236798                 928.222655
##                    bc_util   chargeoff_within_12_mths
##                 903.237585                  13.839041
##                delinq_amnt         mo_sin_old_il_acct
##                   7.339083                 861.358359
##       mo_sin_old_rev_tl_op      mo_sin_rcnt_rev_tl_op
##                 924.470823                 619.822463
##             mo_sin_rcnt_tl                   mort_acc
##                 569.573159                 340.533635
##       mths_since_recent_bc      mths_since_recent_inq
##                 740.265004                 628.440252
##      num_accts_ever_120_pd             num_actv_bc_tl
##                 212.571457                 368.049397
##            num_actv_rev_tl                num_bc_sats
##                 416.302106                 388.545446
##                  num_bc_tl                  num_il_tl
##                 551.080551                 613.152622
##              num_op_rev_tl               num_rev_accts
##                 478.100939                 621.357695
##        num_rev_tl_bal_gt_0                   num_sats
##                 416.022612                 534.198953
##           num_tl_120dpd_2m                num_tl_30dpd
##                   1.379972                   9.718062
##         num_tl_90g_dpd_24m         num_tl_op_past_12m
##                  79.121872                 379.027549
##             pct_tl_nvr_dlq           percent_bc_gt_75
##                 527.714033                 463.105603
##       pub_rec_bankruptcies                  tax_liens
##                 111.537678                  70.710621
##            tot_hi_cred_lim          total_bal_ex_mort
```

```
##                  992.520968                 841.593641
##              total_bc_limit total_il_high_credit_limit
##                  894.617361                 750.163657
##      propSatisBankcardAccts
##                  604.877471
```

```r
rgModel1[["confusion.matrix"]]
```

```
##               predicted
## true           Charged Off Fully Paid  <NA>
##   Charged Off         1579       6727     3
##   Fully Paid          1451      46954     0
```

```r
#pr <- predict (rgModel1, lcdfTst, predict.all = FALSE, proximity = FALSE, type = 'response')


rgModel1[["confusion.matrix"]]
```

```
##               predicted
## true           Charged Off Fully Paid  <NA>
##   Charged Off         1579       6727     3
##   Fully Paid          1451      46954     0
```

```r
#             predicted
#true           Charged Off Fully Paid
#   Charged Off        1588       6681
#   Fully Paid         1372      47073

(1588+47073)/(1588 +6681+1372+47073) # 0.85
```

```
## [1] 0.8580068
```

```r
#scoreTest
scoresRFTest <- predict(rgModel1, lcdfTst)
#confusion table test data
table(scoresRFTest$predictions,lcdfTst$loan_status)
```

```
##
##                Charged Off Fully Paid
##   Charged Off          192        386
##   Fully Paid          2195      13432
```

```r
#               Charged Off Fully Paid
# Charged Off          151        373
#Fully Paid          2198      13483
(151+13483)/(151+13483+373+2198) # 0.8413
```

```
## [1] 0.8413453
```

```r
#scoreVal
scoresRFVal <- predict(rgModel1, lcdfVal)
#confusion table validation
table(scoresRFVal$predictions,lcdfVal$loan_status)
```

```
##
##                Charged Off Fully Paid
##   Charged Off           97        203
##   Fully Paid          1034       6768
```

```
#               Charged Off Fully Paid
#Charged Off           79         166
#Fully Paid          1130        6727
(79+6727)/(79+6727+1130+166) #0.8400
```

```
## [1] 0.8400395
```

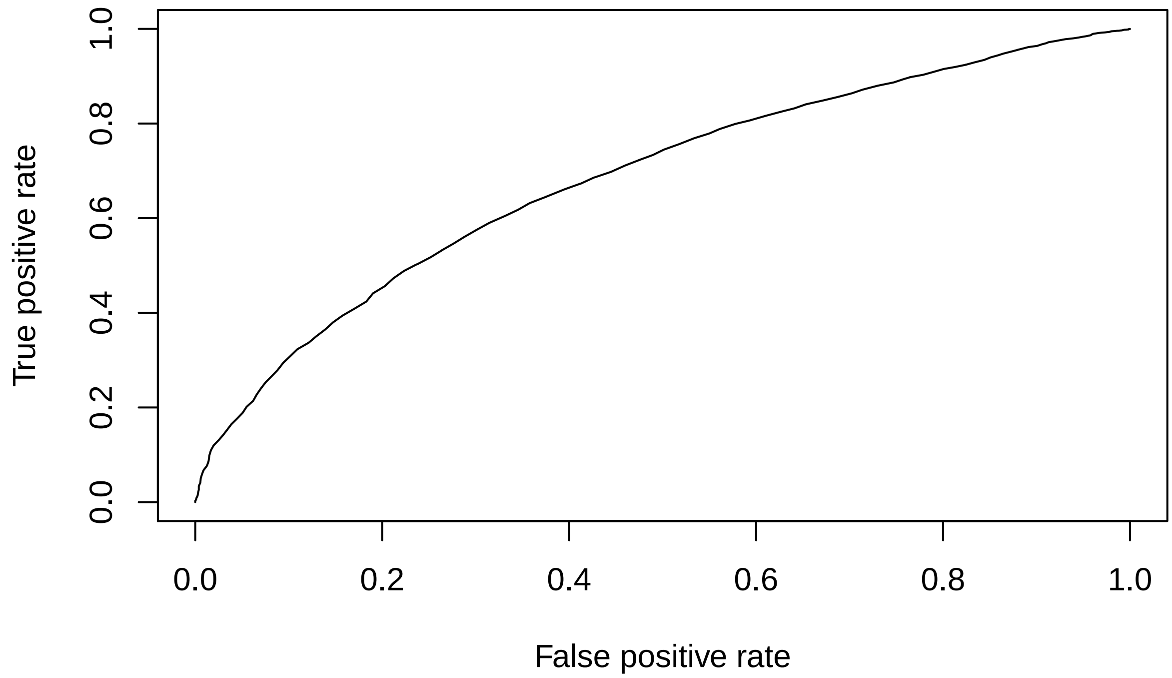
Predictions for Validation

```
predValProb_rgModel1 <- predict(rgModel1, lcdfVal, type='response')
```

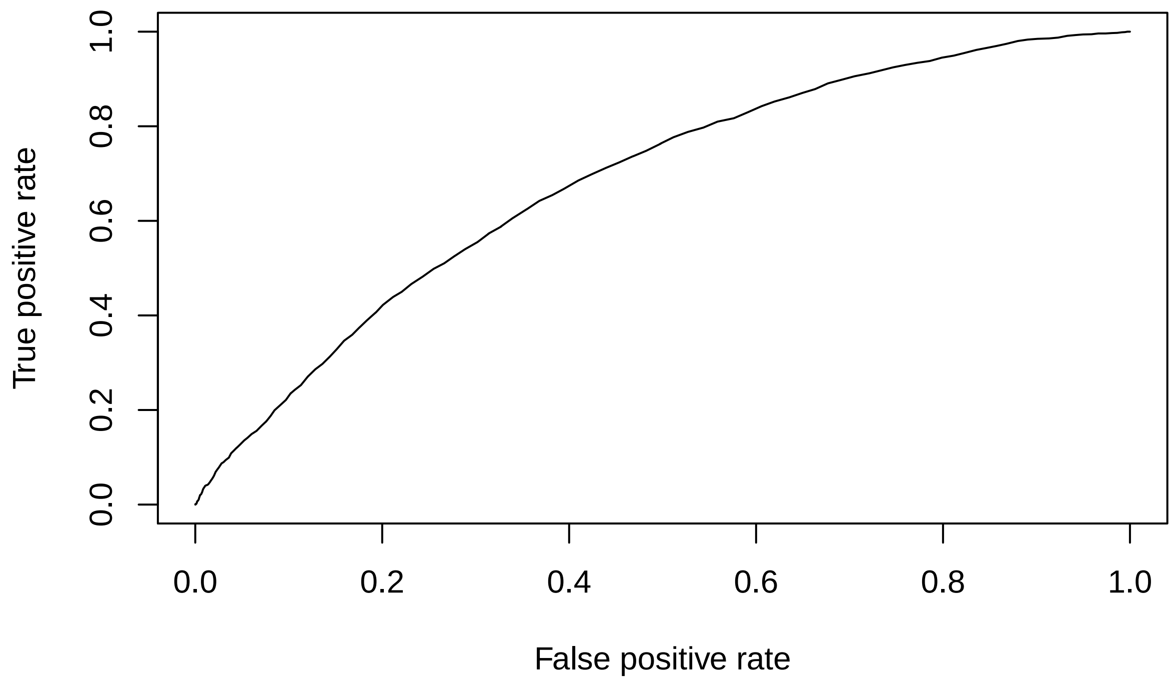
Predictions for Test

```
predTstrgModel1 <- predict(rgModel1, lcdfTst, type='response')
```

```
#ROC

library('ROCR')
rgModelROC <- ranger(loan_status ~., data=lcdfTrn, num.trees =200, min.node.size=1, importance='impurity
scoresRFTest <- predict(rgModelROC, lcdfTst, type="response")
  #now apply the prediction function from ROCR to get a prediction object for charge off
rocPredTst <- prediction(scoresRFTest [["predictions"]][,2], lcdfTst$loan_status, label.ordering = c('Ch
#obtain performance using the function from ROCR, then plot
perfROCTst <- performance(rocPredTst, "tpr", "fpr")
plot(perfROCTst)
```




```
 #now apply the prediction function from ROCR to get a prediction object for fully paid
rocPredTst <- prediction(scoresRFTest [["predictions"]][,1], lcdfTst$loan_status, label.ordering = c('Fu
#obtain performance using the function from ROCR, then plot
perfROCTst <- performance(rocPredTst, "tpr", "fpr")
plot(perfROCTst)
```



Loans Analysis

```
library(lubridate)

#loans by grade
lcdf2 %>% group_by(grade) %>% summarise(nLoans=n(), defaults=sum(loan_status=="Charged Off"),
avgInterest= mean(int_rate), stdInterest=sd(int_rate), avgLoanAMt=mean(loan_amnt), avgPmnt=mean(total_py
```

```
## # A tibble: 7 x 7
##   grade nLoans defaults avgInterest stdInterest avgLoanAMt avgPmnt
## * <fct>  <int>    <int>       <dbl>       <dbl>      <dbl>   <dbl>
## 1 A      20402     1108        7.25      0.796      14146.  15188.
## 2 B      23398     2682       10.7       1.22       12459.  13549.
## 3 C      22577     4116       13.7       0.850      11466.  12363.
## 4 D      10802     2647       16.5       0.895      12150.  12957.
## 5 E       3191     1045       19.8       1.10       12558.  13079.
## 6 F        560      191       24.1       0.798      10169.  10588.
## 7 G         91       38       25.8       0.0593     12509.  13576.
```

```
#calculate the annualized percentage return
lcdf2$annRet <- ((lcdf2$total_pymnt -lcdf2$funded_amnt)/lcdf2$funded_amnt)*(12/36)*100

#summarize by grade
lcdf2 %>% group_by(grade) %>% summarise(nLoans=n(), defaults=sum(loan_status=="Charged Off"), avgIntere
minRet=min(annRet), maxRet=max(annRet))
```

```
## # A tibble: 7 x 11
##   grade nLoans defaults avgInterest stdInterest avgLoanAMt avgPmnt avgRet stdRet
## * <fct>  <int>    <int>       <dbl>       <dbl>      <dbl>   <dbl>  <dbl>  <dbl>
## 1 A      20402     1108        7.25      0.796      14146.  15188.   2.39   3.88
## 2 B      23398     2682       10.7       1.22       12459.  13549.   2.85   5.95
## 3 C      22577     4116       13.7       0.850      11466.  12363.   2.65   8.00
## 4 D      10802     2647       16.5       0.895      12150.  12957.   2.32   9.76
## 5 E       3191     1045       19.8       1.10       12558.  13079.   1.40  11.7
```

```
## 6 F        560      191       24.1       0.798      10169.  10588.   2.55  12.7
## 7 G         91       38       25.8       0.0593     12509.  13576.   2.09  13.4
## # ... with 2 more variables: minRet <dbl>, maxRet <dbl>
```

```
#Some loans are paid back early - find out the actual loan term in months
lcdf2$last_pymnt_d<-paste(lcdf2$last_pymnt_d, "-01", sep = "")
lcdf2$last_pymnt_d<-parse_date_time(lcdf2$last_pymnt_d, "mYd")

# getting actual term
lcdf2 $actualTerm <- ifelse(lcdf2$loan_status=="Fully Paid", as.duration(lcdf2$issue_d %--% lcdf2$last_

#Then, considering this actual term, the actual annual return is
lcdf2$actualReturn <- ifelse(lcdf2$actualTerm>0, ((lcdf2$total_pymnt - lcdf2$funded_amnt)/lcdf2$funded_a

#loan performance by grade
lcdf2 %>% group_by(grade) %>% summarise(nLoans=n(), defaults=sum(loan_status=="Charged Off"), defaultRat
avgInterest= mean(int_rate), avgLoanAmt=mean(loan_amnt), avgRet=mean(annRet), avgActualRet=mean(actualRe
avgActualTerm=mean(actualTerm), minActualRet=min(actualReturn)*100, maxActualRet=max(actualReturn)*100)
```

```
## # A tibble: 7 x 11
##   grade nLoans defaults defaultRate avgInterest avgLoanAmt avgRet avgActualRet
## * <fct>  <int>    <int>       <dbl>       <dbl>      <dbl>  <dbl>        <dbl>
## 1 A      20402     1108      0.0543        7.25     14146.   2.39         3.94
## 2 B      23398     2682      0.115        10.7      12459.   2.85         5.22
## 3 C      22577     4116      0.182        13.7      11466.   2.65         5.73
## 4 D      10802     2647      0.245        16.5      12150.   2.32         5.89
## 5 E       3191     1045      0.327        19.8      12558.   1.40         5.45
## 6 F        560      191      0.341        24.1      10169.   2.55         7.29
## 7 G         91       38      0.418        25.8      12509.   2.09         6.33
## # ... with 3 more variables: avgActualTerm <dbl>, minActualRet <dbl>,
## #   maxActualRet <dbl>
```

```
#loan performance by grade and loan status
lcdf2 %>% group_by(grade, loan_status) %>% summarise(nLoans=n(), defaults=sum(loan_status=="Charged Off
avgInterest= mean(int_rate), avgLoanAmt=mean(loan_amnt), avgRet=mean(annRet), avgActualRet=mean(actualRe
avgActualTerm=mean(actualTerm), minActualRet=min(actualReturn), maxActualRet=max(actualReturn))
```

```
## # A tibble: 14 x 12
## # Groups:   grade [7]
##    grade loan_status nLoans defaults defaultRate avgInterest avgLoanAmt avgRet
##    <fct> <fct>        <int>    <int>       <dbl>       <dbl>      <dbl>  <dbl>
##  1 A     Charged Off   1108     1108           1        7.47     13438. -11.2
##  2 A     Fully Paid   19294        0           0        7.24     14187.   3.18
##  3 B     Charged Off   2682     2682           1       10.9      12261. -11.0
##  4 B     Fully Paid   20716        0           0       10.7      12484.   4.64
##  5 C     Charged Off   4116     4116           1       13.7      11652. -11.6
##  6 C     Fully Paid   18461        0           0       13.7      11424.   5.83
##  7 D     Charged Off   2647     2647           1       16.6      12495. -12.2
##  8 D     Fully Paid    8155        0           0       16.5      12038.   7.04
##  9 E     Charged Off   1045     1045           1       19.8      12449. -12.8
## 10 E     Fully Paid    2146        0           0       19.7      12611.   8.31
## 11 F     Charged Off    191      191           1       24.2      11158. -12.5
## 12 F     Fully Paid     369        0           0       24.1       9657.  10.4
## 13 G     Charged Off     38       38           1       25.8      11250  -11.0
## 14 G     Fully Paid      53        0           0       25.8      13412.  11.5
## # ... with 4 more variables: avgActualRet <dbl>, avgActualTerm <dbl>,
```

```
## #   minActualRet <dbl>, maxActualRet <dbl>
```

```
#profitValue based on
lcdf2 %>% group_by(loan_status) %>% summarise(avgInt=mean(int_rate),avgActInt = mean(actualReturn))
```

```
## # A tibble: 2 x 3
##   loan_status avgInt avgActInt
## * <fct>        <dbl>     <dbl>
## 1 Charged Off   13.9   -0.117
## 2 Fully Paid    11.6    0.0803
```

```
PROFITVAL <- 24 #profit
COSTVAL <- -35 # loss

Avg = 8.03 * 2.1 + 2*0.9

#Performance
scoreTstRF2 <- predict(rgModel1,lcdfTst, type="response")["Fully Paid"]

prPerfRF2 <- data.frame(scoreTstRF2)
#prPerfRF2 <- cbind(prPerfRF2, status=lcdfTst$loan_status)
#prPerfRF2 <- prPerfRF2[order(-scoreTstRF2) ,] #sort in desc order of prob(fully_paid) prPerfRF$profit
#max(prPerfRF$cumProfit) prPerfRF$cumProfit[which.max(prPerfRF$cumProfit)]
```

7. (a). Evaluate the Loans for Investment Decisions

rpart model

```
              true
 predictTest   Charged Off Fully Paid
   Charged Off         521       1493
   Fully Paid         1866      12325
```

8.03x2.1+(.9)*2= 17.04 *12325=210,054.98, (-11.7)*3= -35.1*1866=65,496.6.

`210,054.98-65496.6=144558.36 profit for the rpart model`

C50 model

```
               true
predictC50Test Charged Off Fully Paid
   Charged Off         653       1676
   Fully Paid         1734      12142
```

8.03x2.1+(.9)*2= 17.04*12142=206936.11, (-11.7)*3= -35.1*1734=60863.4

206,936.11-60863.4=146072.71

When doing these profit evaluations, we used the average returns on both "Charged off" and "Fully Paid" loans. We then used our models to look at the predictions and how we fared. In the matrix where we predicted it would be Fully paid and it was fully paid we multiplied that by the average amount for a fully paid loan to get profit. Then for the loans that we thought would be fully paid and end up be Charged off we multiplied that amount by average return to get what we would have lost be the prediction model. The other instances we would not have invested if we thought it would be charged off and it ended up being Fully paid, and the predictions we got correct for Charged Off we would not have invested in either so not lost money.

7 (b).

if you look at the data in a descending order by the probability of becoming Fully Paid, you can see certain points where the drop offs take place and the percentage of defaults greatly increases. We chose the score to cut off at .589 which has a 18.7% default rate. We felt that after this point the score went below .500 and that the default percentage got closer to 25% which can become riskier. We also looked at the amount of loans that were in each score and most of them were before this cutoff, this shows that a vast majority of these loans scored well in our model.

The advantageous part about using the model like this is that it's weighted with so many high scores showing a default rate at only 7.96%. Almost half of all loans are in that score zone. When comparing using a model like this to invest in safe cd's the risk is the greatest factor here because these safe cds are guaranteed to be paid out. Safe cds provide the 2% interest every year. On 100 dollars that will turn in $106 by the end of year 3. Using our model is more profitable than investing in safe cds because if you have the same amount of loans as from our previous models you only end up with 97,230 in profit. You get this from taking the average return of 2 dollars for 3 years and then multiplying that by the number of loans at 16205. This is still a decent return but if you are doing a lot of loans at once you have the ability to spread the risk out more and find potentially more profitable loans. That 97,230 compared to above models at over $140,000 is a drastic difference over a short period of time.

```r
#get the 'scores' from applying the model to the data
predTrnProb2=predict(prn_lcDT, lcdfTrn, type='prob')

trnSc2 <- lcdfTrn %>%  select("loan_status")   # selects the OUTCOME column into trnSc
trnSc2$score<-predTrnProb2[, 2]  #add a column named 'Score' with prob(default) values in the first col

#sort by score
trnSc2<-trnSc2[order(trnSc2$score, decreasing=TRUE),]

trnSc2[1:50,]
```

```
## # A tibble: 50 x 2
##    loan_status score
##    <fct>       <dbl>
##  1 Fully Paid  0.794
##  2 Fully Paid  0.794
##  3 Fully Paid  0.794
##  4 Fully Paid  0.794
##  5 Charged Off 0.794
##  6 Fully Paid  0.794
##  7 Fully Paid  0.794
##  8 Fully Paid  0.794
##  9 Fully Paid  0.794
## 10 Fully Paid  0.794
## # ... with 40 more rows
```

```r
trnSc2 %>% group_by(score, loan_status)  %>% summarise(nloans = n())
```

```
## `summarise()` has grouped output by 'score'. You can override using the `.groups` argument.
```

```
## # A tibble: 32 x 3
## # Groups:   score [16]
##    score loan_status nloans
##    <dbl> <fct>        <int>
##  1 0.176 Charged Off     56
##  2 0.176 Fully Paid      36
##  3 0.327 Charged Off    123
##  4 0.327 Fully Paid     179
##  5 0.358 Charged Off    490
##  6 0.358 Fully Paid     819
```

```
##   7 0.362 Charged Off   1117
##   8 0.362 Fully Paid    1904
##   9 0.407 Charged Off    286
##  10 0.407 Fully Paid     589
## # ... with 22 more rows
```

```
trnSc2 %>% group_by(score) %>% summarise(nLoans=n(), defaults=sum(loan_status=="Charged Off")) %>% muta
```

```
## # A tibble: 16 x 4
##      score nLoans defaults prctCharged_off
##    * <dbl>  <int>    <int>           <dbl>
##   1 0.176      92       56           60.9
##   2 0.327     302      123           40.7
##   3 0.358    1309      490           37.4
##   4 0.362    3021     1117           37.0
##   5 0.407     875      286           32.7
##   6 0.420     768      242           31.5
##   7 0.452     767      221           28.8
##   8 0.547     804      174           21.6
##   9 0.589    9997     1884           18.8
##  10 0.592     615      115           18.7
##  11 0.601     469       85           18.1
##  12 0.618    1254      214           17.1
##  13 0.653    2303      346           15.0
##  14 0.724    6995      788           11.3
##  15 0.731     265       29           10.9
##  16 0.794   26878     2139            7.96
```

xgboost model

Additional models to consider - develop boosted tree models (using either gbm or XGBoost). Explain how you experiment with parameters, how performance varies, which parameter setting you use for the 'best' model.

Model performance should be evaluated through use of same set of criteria as for the other models - confusion matrix based, ROC analyses and AUC, cost-based performance.

Provide a table with comparative evaluation of all the best models from each methods; show their ROC curves in a combined plot. Also provide profit-curves and 'best' profit' and associated cutoff. At this cutoff, what are the accuracy values for the different models?

As a first step we prepared the data to be used in xgboost. For this the data was converted to numeric with the use of the caret library, all variables were converted to dummy vars except the dependent variable loan status. Then we created a new dataset called dxlcdf with the function predict. For loan status we converted as a dummy variable and kept the level charged off. Then we create the training, test and validation sets.

Next we took care of the unbalanced data for this we calculate the sqr(sum(negative instances) / sum(positive instances)) we apply this number later to the parameter scale_pos_weight. Before doing this the model would run the 500 nrounds but after balancing the weights we reduce to a best iteration of 49 with an accuracy of 0.85 We then calculate the xgboost for the validation data and get the best iteration on number 3.

```
library(caret)
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 3.6.2

##
```

```
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```
# Using dummyVars function in the 'caret' package to convert factor variables to dummy-variables.
fdum<-dummyVars(~.,data=lcdf %>% select(-loan_status))

#replacing the dummy variables in the dataset
dxlcdf <- predict(fdum, lcdf)

#checking levels to know how is composed loan status
levels(lcdf$loan_status)
```

```
## [1] "Charged Off" "Fully Paid"
```

```
#"Fully Paid"  "Charged Off"
#converting loan status to dummy variables
dylcdf <- class2ind(lcdf$loan_status, drop2nd = FALSE)
# we decided we want to keep charged off
fplcdf <- dylcdf [ , 2]

#Training, test subsets
dxlcdfTrn <- dxlcdf[indicesTraining,]
colcdfTrn <- fplcdf[indicesTraining]
dxlcdfTst <- dxlcdf[indicesTest,]
colcdfTst <- fplcdf[indicesTest]
dxlcdfVal <- dxlcdf[indicesValidation,]
colcdfVal <- fplcdf[indicesValidation]

#calculating the weights of the subsets
sum(dxlcdfTrn==1)
```

```
## [1] 547717
```

```
sum(dxlcdfTrn==0)
```

```
## [1] 38153380
```

```
sqrt(sum(dxlcdfTrn==0) / sum(dxlcdfTrn==1))   #8.823873
```

```
## [1] 8.346193
```

```
sum(dxlcdfTst==1)
```

```
## [1] 156661
```

```
sum(dxlcdfTst==0)
```

```
## [1] 10901452
```

```
sqrt(sum(dxlcdfTst==1) / sum(dxlcdfTst==0))
```

```
## [1] 0.1198777
```

```
sum(dxlcdfVal==1)
```

```
## [1] 78253
```

```r
sum(dxlcdfVal==0)
```

```
## [1] 5450468
```

```r
sqrt(sum(dxlcdfVal==1) / sum(dxlcdfVal==0))
```

```
## [1] 0.1198212
```

```r
#Creating of xgb.DMatrix
dxTrn <- xgb.DMatrix(subset(dxlcdfTrn), label=colcdfTrn)
dxTst <- xgb.DMatrix(subset(dxlcdfTst), label=colcdfTst)
dxVal <- xgb.DMatrix(subset(dxlcdfVal), label=colcdfVal)

## Process for training and test

#we use the xgbWatchlist to watch the progress of learning thru performance on these datasets
xgbWatchlist <- list(train = dxTrn, eval = dxTst)

#This is the list of parameters for the xgboost model development functions wich are going to use first
xgbParam <- list (
max_depth = 5, eta = 0.01, scale_pos_weight = 8.82,
objective = "binary:logistic", eval_metric="error", eval_metric = "auc")
```

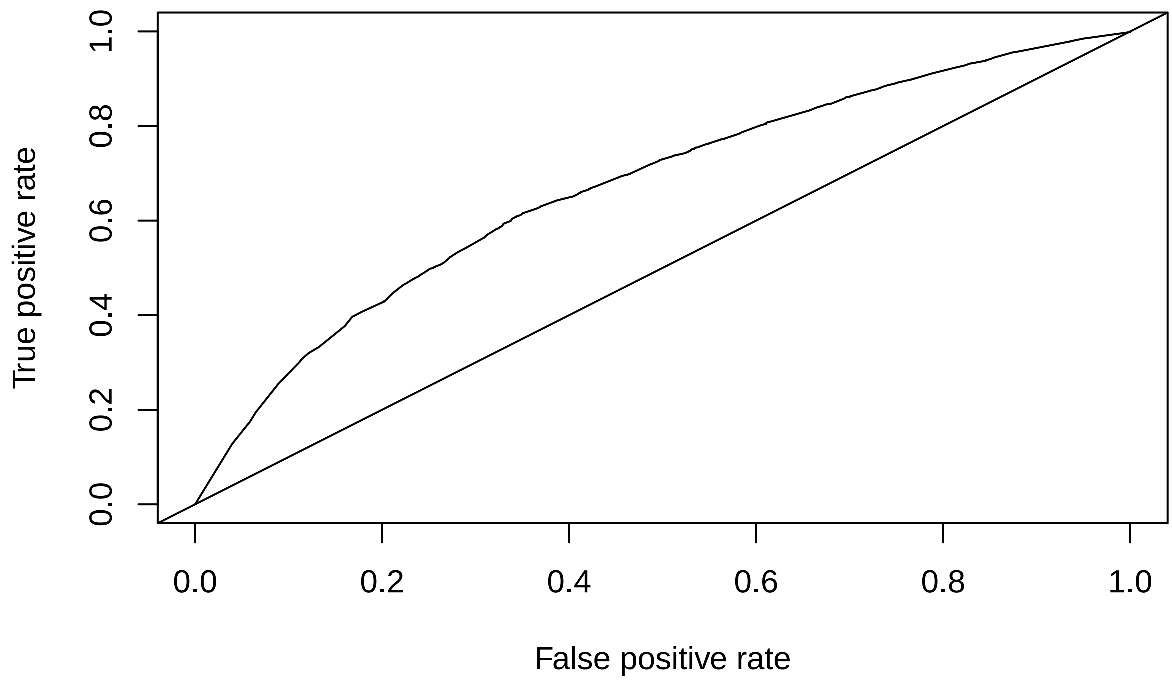
```r
#confusion matrix
table(pred=as.numeric(xpredTrg>0.5), act=colcdfTrn)
```

```
##     act
## pred     0     1
##    0    19     0
##    1  8290 48405
```

```r
#        act
# pred      0     1
#    0     19     0
#    1   8290 48405
 (19 + 48405) / (19+8290+48405)
```

```
## [1] 0.853828
```

```r
# = 0.85 accuracy

#ROC, AUC performance
xpredTst<-predict(xgb_lsM1, dxTst)

pred_xgb_lsM1<-prediction(xpredTst, lcdfTst$loan_status,
label.ordering = c("Charged Off", ("Fully Paid")))
aucPerf_xgb_lsM1<-performance(pred_xgb_lsM1, "tpr", "fpr")
plot(aucPerf_xgb_lsM1)
abline(a=0, b= 1)
```



Using cross-validation on training dataset to determine best model

```
xgbParamGrid
```

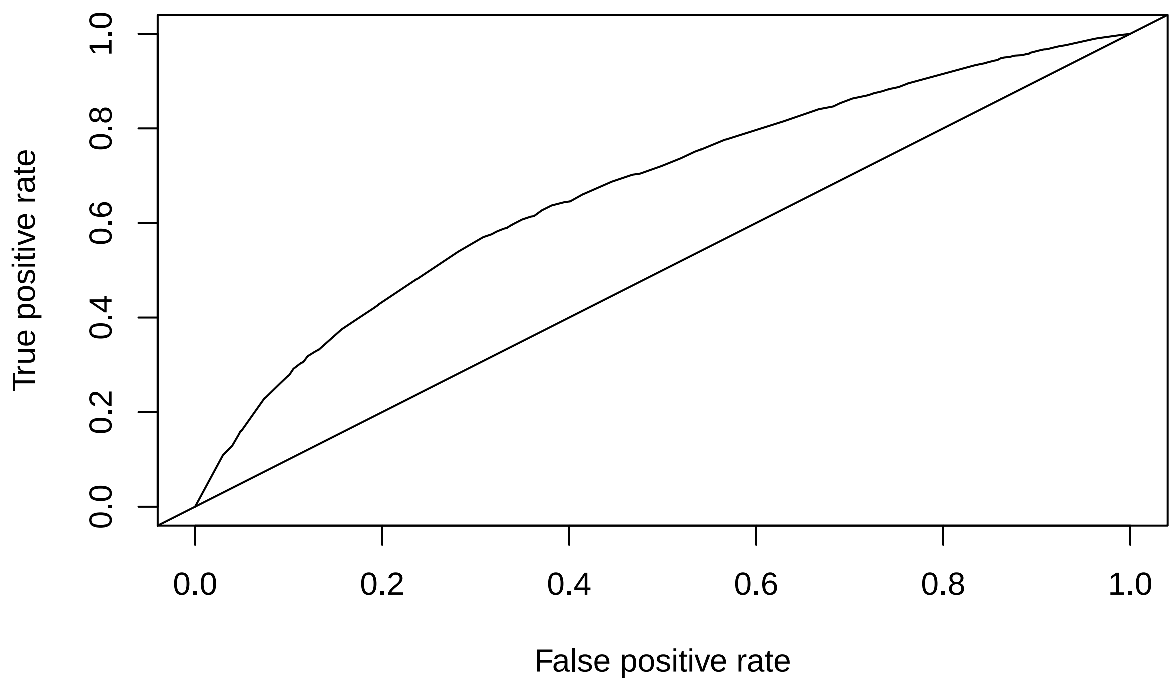
```
##   max_depth   eta bestTree bestPerf
## 1         2 0.001       51 0.670853
## 2         5 0.001       73 0.672626
## 3         2 0.010       15 0.665226
## 4         5 0.010       38 0.673593
## 5         2 0.100      147 0.685824
## 6         5 0.100      100 0.684785
```

```
# max_depth eta bestTree bestPerf
# 2 0.001   28  0.668216    <--
# 5 0.001   6   0.671976
# 2 0.010   10  0.670330
# 5 0.010   26  0.674678
# 2 0.100   84  0.682619
# 5 0.100   101 0.684982

#Best parameters
xgbParam_Best <- list (booster = "gbtree", objective = "binary:logistic", min_child_weight=1, colsample_

# XGBOOST running the model with the best parameters found with the for loop
xgb_lsM2 <- xgb.train(xgbParam_Best, dxTrn, nrounds = xgb_tune$best_iteration)

#XGBOOST evaluation of the model
#Using the predicting function to get the scores in the training data  set
xpredTrn<-predict(xgb_lsM2, dxTrn)

#Using the predicting function to get the scores in the test data set
xpredTst<-predict(xgb_lsM2, dxTst)

#confusion matrix
```

```r
table(pred=as.numeric(xpredTst>0.5), act=colcdfTst)
```

```
##     act
## pred     0     1
##    1  2387 13818
```

```r
#ROC, AUC performance
pred_xgb_lsM2<-prediction(xpredTst, lcdfTst$loan_status,
label.ordering = c("Charged Off", ("Fully Paid")))

aucPerf_xgb_lsM2<-performance(pred_xgb_lsM2, "tpr", "fpr")
plot(aucPerf_xgb_lsM2)
abline(a=0, b= 1)
```




```r
######
## Process for test and validation

#we can watch the progress of learning thru performance on these datasets
xgbWatchlistVal <- list(train = dxTst, eval = dxVal)
#list of parameters for the xgboost model development functions
xgbParam <- list (
max_depth = 5, eta = 0.01, scale_pos_weight = 8.82,
objective = "binary:logistic", eval_metric="error", eval_metric = "auc")

xgb_lsM2 <- xgb.train(xgbParam, dxTst, nrounds = 500, xgbWatchlistVal, early_stopping_rounds = 10 )
```

```
## [1]  train-error:0.146560    train-auc:0.652217  eval-error:0.140212 eval-auc:0.643278
## Multiple eval metrics are present. Will use eval_auc for early stopping.
## Will train until eval_auc hasn't improved in 10 rounds.
##
## [2]  train-error:0.146745    train-auc:0.667170  eval-error:0.140089 eval-auc:0.660555
## [3]  train-error:0.146560    train-auc:0.667212  eval-error:0.139595 eval-auc:0.660537
## [4]  train-error:0.146930    train-auc:0.667212  eval-error:0.139595 eval-auc:0.660538
```

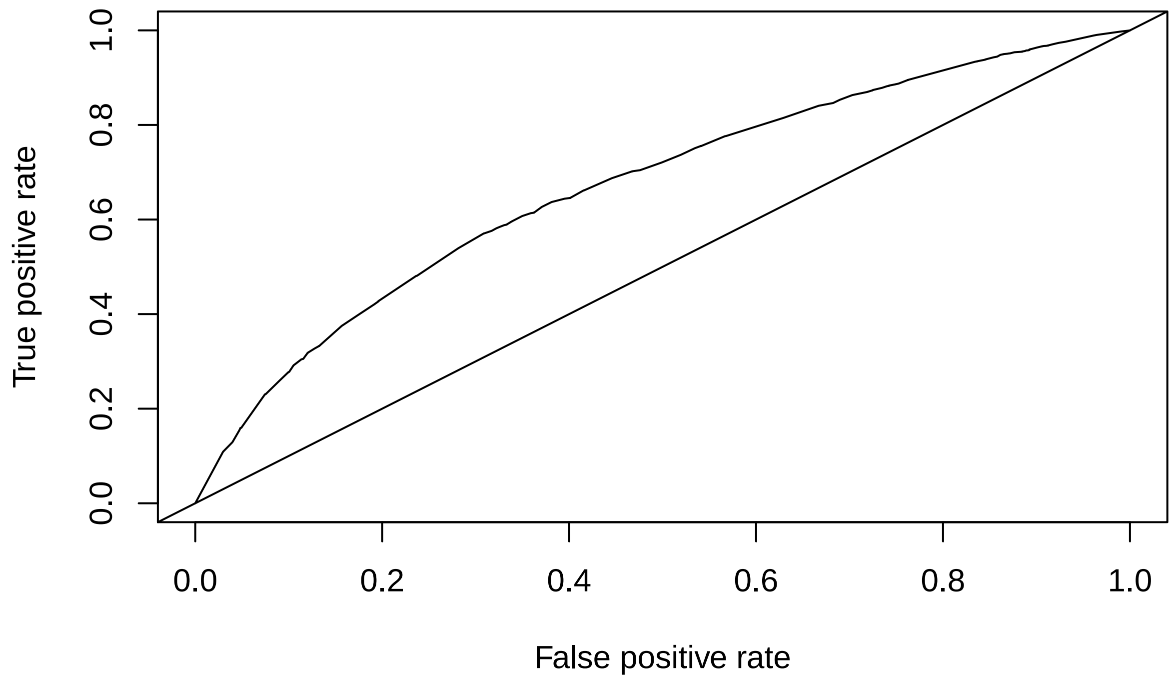
```
## [5]  train-error:0.146560    train-auc:0.667409  eval-error:0.139595 eval-auc:0.660500
## [6]  train-error:0.146683    train-auc:0.667410  eval-error:0.139348 eval-auc:0.660501
## [7]  train-error:0.146683    train-auc:0.668102  eval-error:0.139348 eval-auc:0.659921
## [8]  train-error:0.146683    train-auc:0.671860  eval-error:0.139348 eval-auc:0.662652
## [9]  train-error:0.146683    train-auc:0.671897  eval-error:0.139348 eval-auc:0.662473
## [10] train-error:0.146683    train-auc:0.671858  eval-error:0.139348 eval-auc:0.662522
## [11] train-error:0.146683    train-auc:0.671863  eval-error:0.139348 eval-auc:0.662488
## [12] train-error:0.146683    train-auc:0.675830  eval-error:0.139348 eval-auc:0.663956
## [13] train-error:0.146683    train-auc:0.675819  eval-error:0.139348 eval-auc:0.663961
## [14] train-error:0.146683    train-auc:0.675839  eval-error:0.139348 eval-auc:0.663951
## [15] train-error:0.146683    train-auc:0.675839  eval-error:0.139348 eval-auc:0.663951
## [16] train-error:0.146868    train-auc:0.676188  eval-error:0.139472 eval-auc:0.663943
## [17] train-error:0.146930    train-auc:0.676985  eval-error:0.139595 eval-auc:0.664118
## [18] train-error:0.146930    train-auc:0.676976  eval-error:0.139595 eval-auc:0.664119
## [19] train-error:0.146930    train-auc:0.677552  eval-error:0.139595 eval-auc:0.663914
## [20] train-error:0.146930    train-auc:0.677529  eval-error:0.139595 eval-auc:0.663927
## [21] train-error:0.146868    train-auc:0.677523  eval-error:0.139472 eval-auc:0.663936
## [22] train-error:0.146930    train-auc:0.678568  eval-error:0.139595 eval-auc:0.663667
## [23] train-error:0.146868    train-auc:0.678539  eval-error:0.139472 eval-auc:0.663680
## [24] train-error:0.146930    train-auc:0.678816  eval-error:0.139595 eval-auc:0.663697
## [25] train-error:0.146930    train-auc:0.679338  eval-error:0.139595 eval-auc:0.663410
## [26] train-error:0.146930    train-auc:0.679868  eval-error:0.139595 eval-auc:0.665170
## [27] train-error:0.146930    train-auc:0.681189  eval-error:0.139595 eval-auc:0.665171
## [28] train-error:0.146930    train-auc:0.681220  eval-error:0.139595 eval-auc:0.665179
## [29] train-error:0.146930    train-auc:0.681123  eval-error:0.139595 eval-auc:0.665193
## [30] train-error:0.146930    train-auc:0.681233  eval-error:0.139595 eval-auc:0.665020
## [31] train-error:0.146930    train-auc:0.681358  eval-error:0.139595 eval-auc:0.664875
## [32] train-error:0.146930    train-auc:0.681350  eval-error:0.139595 eval-auc:0.664754
## [33] train-error:0.146930    train-auc:0.681319  eval-error:0.139595 eval-auc:0.664805
## [34] train-error:0.146930    train-auc:0.681364  eval-error:0.139595 eval-auc:0.664789
## [35] train-error:0.146930    train-auc:0.681366  eval-error:0.139595 eval-auc:0.664799
## [36] train-error:0.146930    train-auc:0.681374  eval-error:0.139595 eval-auc:0.664749
## [37] train-error:0.146930    train-auc:0.684078  eval-error:0.139595 eval-auc:0.664669
## [38] train-error:0.146930    train-auc:0.684117  eval-error:0.139595 eval-auc:0.664630
## [39] train-error:0.146930    train-auc:0.684098  eval-error:0.139595 eval-auc:0.664640
## Stopping. Best iteration:
## [29] train-error:0.146930    train-auc:0.681123  eval-error:0.139595 eval-auc:0.665193
```

```r
xpredTrg2<-predict(xgb_lsM2, dxVal)

#confusion matrix
table(pred=as.numeric(xpredTrg2>0.5), act=colcdfVal)
```

```
##     act
## pred    0    1
##    1 1131 6971
```

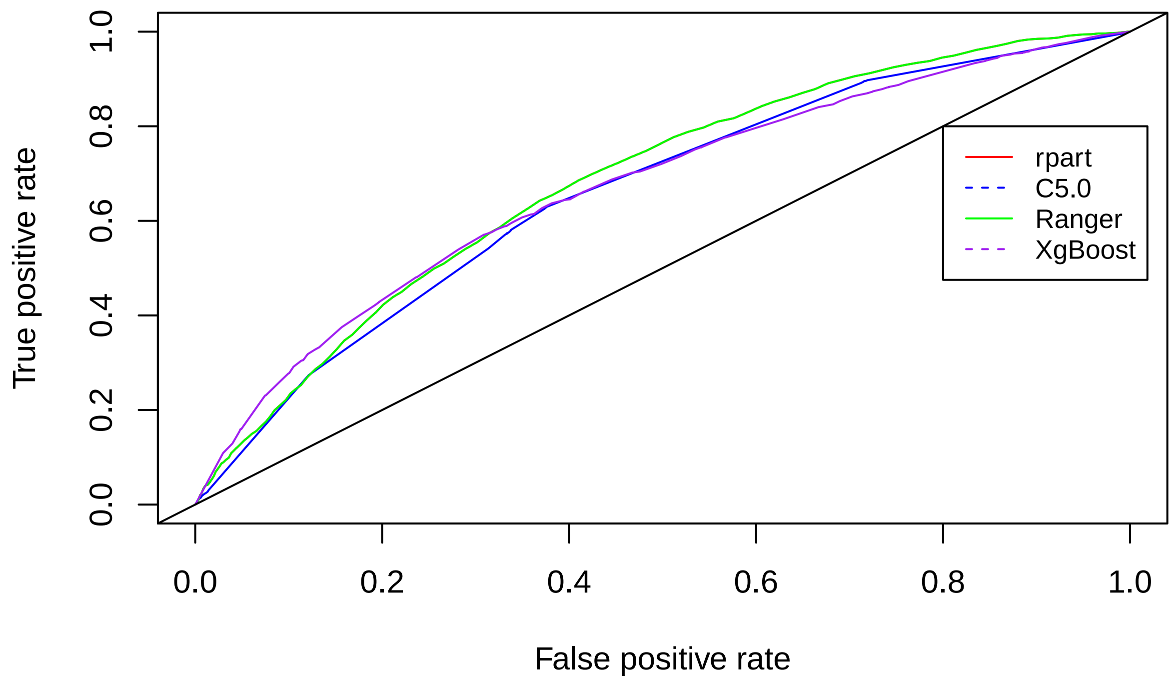
```r
#ROC, AUC performance
xpredVal<-predict(xgb_lsM1, dxVal)

pred_xgb_lsM2<-prediction(xpredTst, lcdfTst$loan_status,
label.ordering = c("Charged Off", ("Fully Paid")))

aucPerf_xgb_lsM2<-performance(pred_xgb_lsM2, "tpr", "fpr")
```

```r
plot(aucPerf_xgb_lsM2)
abline(a=0, b= 1)
```



Plotting Lines For ROC Curves

```r
plot(perfROCTst, col="red")
plot(c5perfROCTst, col="blue", add= TRUE)
plot(perfROCTst, col="green", add= TRUE)
plot(aucPerf_xgb_lsM2, col="purple", add=TRUE)
abline(a=0, b= 1, col="black")


legend(0.8, 0.8, legend=c("rpart", "C5.0", "Ranger", "XgBoost"),
       col=c("red", "blue", "green", "purple"), lty=1:2, cex=0.8)
```



True positive rate
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate
rpart
C5.0
Ranger
XgBoost

“‘