IDS 572 - Assignment 1 Part B_Draft

Joseline Tanujaya, Sweta Bansal, Vibhanshu

2/28/2021

Data preparation:

1. Keep derived new attributes those are not derived from leakage variables, and have significant AUC (above .52):

- propSatisBankcardAccts, which is the proportion of satisfactory bankcard accounts of all bankcard for each data point.
- prop_OpAccts_to_TotAccts, which is the percentage of accounts not yet paid off from all accounts.
- propLoanAmt_to_AnnInc, which is the ratio of loan amount to reported annual incom. Indicates the person's ability to pay back the loan based on his/her income.

*Note: some leakage derived variables like annRet, actualReturn, and actualTerm are kept for analysis only, because they will be useful in evaluating the models. They are not used in building/training the model.

2. Remove variables with NAs more than 60%, and treat the remaining NAs by passing in median, max values as appropriate.

3. Remove Leakage variables. After this process, the data has only 46 variables left.

4. Treat variable types for building models:

- Character variables changed into factors, which values are categorical
- Target variable into ordered factors
- Recode minority classes of purposes into "others" to minimize the number of dummy variables created in xgboost. This will be useful in the last part of our evaluation.

Q5 Analysis

Typically the proportion of Train to Test is around 80%-20% to 70%-30%. We are giving more data to Train set because it is required to make the classification model better. We set apart 30% for Testing, because our dataset is big enough (more than 80,000 rows), therefore 70% is enough to build the model.

```
#split the data into trn, tst subsets
nr=nrow(lcdf)
trnIndex = sample(1:nr, size = round(0.7*nr), replace=FALSE)
lcdfTrn=lcdf[trnIndex,]
lcdfTst = lcdf[-trnIndex,]

dim(lcdfTrn)
```

```
## [1] 56715    46
```

```
dim(lcdfTst)
```

```
## [1] 24307    46
```

Build Decision Tree using rpart:

We set various values to CP, prior, minsplit, and split (gini/information). Please refer to the **Table 1 on Appendix** for the complete documentation of our parameters and the results.

Because the data is highly unbalanced (82.9% Fully Paid, 17.1% Charged Off on Training set), we encountered problems during the development of the tree. The tree would not split, because classifying everything as “Fully Paid” will result in 17.1% error rate. We tweaked the CP parameter to split it, but it resulted in unexpected CP table: the xerror is not going down, instead it increases. So we set the ‘prior’ (0.5, 0.5) and saw expected pattern (xerror is going down). But the overall relative error could not go lower than 0.50358 and xerror could not go lower than 0.79532 even with the best model (highest accuracy). The best model with ‘prior’ (0.5, 0.5) is saved in the name rpDT1.

To reflect the unbalanced data, we set ‘prior’ (1-0.171, 0.171). The resulting CP went back to the previous pattern, where xerror increases as rel error decreases. However, we got the best accuracy from this model saved in rpDT2, which yields 70% accuracy on training data and 63% on test data.

The evaluation of rpDT2 is done with AUC, ROC, and confusion matrices.

```
library(rpart)
library(ROCR)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
## Set prior to 0.5 - 0.5: decreasing xerror pattern as expected in the CP table, but low accura
cy.
rpDT1 <- rpart(loan_status ~., data=subset(lcdfTrn, select=-c(annRet, actualTerm, actualReturn,
 total_pymnt)), method="class", parms = list(split = "information", prior=c(0.5,0.5)), control =
rpart.control(minsplit = 30, cp=0.00))

rpDT1<-prune(rpDT1,cp=0.00022)
printcp(rpDT1)
```

```
##
## Classification tree:
## rpart(formula = loan_status ~ ., data = subset(lcdfTrn, select = -c(annRet,
##     actualTerm, actualReturn, total_pymnt)), method = "class",
##     parms = list(split = "information", prior = c(0.5, 0.5)),
##     control = rpart.control(minsplit = 30, cp = 0))
##
## Variables actually used in tree construction:
##  [1] acc_open_past_24mths      annual_inc
##  [3] avg_cur_bal               bc_open_to_buy
##  [5] bc_util                   dti
##  [7] emp_length                home_ownership
##  [9] initial_list_status       installment
## [11] int_rate                  loan_amnt
## [13] mo_sin_old_il_acct        mo_sin_old_rev_tl_op
## [15] mo_sin_rcnt_rev_tl_op     mo_sin_rcnt_tl
## [17] mort_acc                  mths_since_recent_bc
## [19] mths_since_recent_inq     num_bc_sats
## [21] num_bc_tl                 num_il_tl
## [23] num_op_rev_tl             num_rev_accts
## [25] num_sats                  pct_tl_nvr_dlq
## [27] prop_OpAccts_to_TotAccts  propLoanAmt_to_AnnInc
## [29] propSatisBankcardAccts    purpose
## [31] sub_grade                 tax_liens
## [33] tot_hi_cred_lim           total_bal_ex_mort
## [35] total_bc_limit            total_il_high_credit_limit
## [37] total_rev_hi_lim
##
## Root node error: 28358/56715 = 0.5
##
## n= 56715
##
##            CP nsplit rel error  xerror      xstd
## 1  0.25676510      0   1.00000 1.01618 0.0073630
## 2  0.00279109      1   0.74323 0.74323 0.0055791
## 3  0.00188099      5   0.72993 0.73814 0.0062372
## 4  0.00186016      8   0.72428 0.73857 0.0062882
## 5  0.00150118      9   0.72242 0.73761 0.0062848
## 6  0.00148673     10   0.72092 0.73878 0.0063006
## 7  0.00115380     12   0.71795 0.74027 0.0062865
## 8  0.00111946     13   0.71679 0.74133 0.0062378
## 9  0.00107771     18   0.71081 0.74238 0.0062499
## 10 0.00080629     22   0.70608 0.74190 0.0061811
## 11 0.00079635     24   0.70446 0.74265 0.0062710
## 12 0.00066236     25   0.70367 0.74566 0.0063273
## 13 0.00062595     27   0.70234 0.74866 0.0063934
## 14 0.00061940     30   0.70047 0.74795 0.0063909
## 15 0.00061940     31   0.69985 0.74761 0.0063863
## 16 0.00059344     32   0.69923 0.74737 0.0063665
## 17 0.00058074     38   0.69567 0.74907 0.0063819
## 18 0.00056078     39   0.69509 0.74991 0.0063834
## 19 0.00056009     50   0.68820 0.74998 0.0064004
## 20 0.00053701     52   0.68708 0.74932 0.0064299
```

```
## 21  0.00053681     60   0.68193 0.74887 0.0063971
## 22  0.00051233     61   0.68139 0.75003 0.0064013
## 23  0.00050552     63   0.68037 0.75087 0.0064292
## 24  0.00050248     66   0.67885 0.75097 0.0064301
## 25  0.00049552     68   0.67785 0.75128 0.0064374
## 26  0.00048207     79   0.67145 0.75154 0.0064307
## 27  0.00047991     83   0.66917 0.75192 0.0064362
## 28  0.00047514     84   0.66869 0.75357 0.0064364
## 29  0.00045423     85   0.66821 0.75462 0.0064490
## 30  0.00045302     86   0.66776 0.75690 0.0064993
## 31  0.00044331     87   0.66730 0.75718 0.0064964
## 32  0.00044030    102   0.65866 0.75756 0.0065176
## 33  0.00043478    106   0.65657 0.75784 0.0065277
## 34  0.00042098    109   0.65527 0.75779 0.0065480
## 35  0.00041918    113   0.65353 0.75923 0.0065612
## 36  0.00041710    116   0.65228 0.75935 0.0065575
## 37  0.00041102    120   0.65059 0.75959 0.0065638
## 38  0.00040703    121   0.65018 0.75947 0.0065629
## 39  0.00040367    133   0.64515 0.75966 0.0065684
## 40  0.00039612    138   0.64314 0.75991 0.0065682
## 41  0.00039229    141   0.64195 0.76113 0.0065897
## 42  0.00037944    142   0.64155 0.76403 0.0066350
## 43  0.00037491    144   0.64080 0.76621 0.0066539
## 44  0.00037356    155   0.63504 0.76628 0.0066521
## 45  0.00037164    158   0.63385 0.76610 0.0066449
## 46  0.00036835    159   0.63348 0.76716 0.0066544
## 47  0.00036660    162   0.63238 0.76795 0.0066577
## 48  0.00035939    163   0.63201 0.76905 0.0066626
## 49  0.00035915    166   0.63093 0.76814 0.0066630
## 50  0.00035777    167   0.63057 0.76879 0.0066645
## 51  0.00035099    172   0.62873 0.76871 0.0066600
## 52  0.00035099    173   0.62838 0.76921 0.0066806
## 53  0.00035090    179   0.62627 0.76921 0.0066806
## 54  0.00034415    190   0.62215 0.76970 0.0066885
## 55  0.00033851    192   0.62147 0.77264 0.0067429
## 56  0.00033827    193   0.62113 0.77300 0.0067526
## 57  0.00033539    196   0.62011 0.77429 0.0067609
## 58  0.00033347    197   0.61978 0.77541 0.0067603
## 59  0.00033269    202   0.61811 0.77555 0.0067620
## 60  0.00033226    206   0.61678 0.77563 0.0067548
## 61  0.00033191    207   0.61645 0.77548 0.0067504
## 62  0.00033162    209   0.61578 0.77552 0.0067504
## 63  0.00033035    212   0.61479 0.77549 0.0067558
## 64  0.00033035    213   0.61446 0.77541 0.0067594
## 65  0.00032314    215   0.61380 0.77600 0.0067493
## 66  0.00032002    217   0.61315 0.77638 0.0067643
## 67  0.00031762    220   0.61204 0.77745 0.0067896
## 68  0.00031750    225   0.61017 0.77812 0.0067848
## 69  0.00031282    227   0.60954 0.77819 0.0067768
## 70  0.00030970    228   0.60922 0.77913 0.0067915
## 71  0.00030970    229   0.60891 0.77885 0.0067943
## 72  0.00030970    233   0.60768 0.77885 0.0067943
## 73  0.00030956    234   0.60737 0.77878 0.0067970
## 74  0.00030372    239   0.60582 0.77918 0.0068021
```

```
## 75  0.00030034    254   0.60037 0.78110 0.0068109
## 76  0.00029721    256   0.59977 0.78247 0.0068209
## 77  0.00029721    257   0.59947 0.78355 0.0068601
## 78  0.00029530    258   0.59918 0.78339 0.0068628
## 79  0.00029009    259   0.59888 0.78489 0.0068779
## 80  0.00028905    262   0.59801 0.78536 0.0068829
## 81  0.00028803    266   0.59686 0.78541 0.0068820
## 82  0.00028425    274   0.59433 0.78654 0.0068998
## 83  0.00028241    279   0.59279 0.78727 0.0068977
## 84  0.00028029    282   0.59195 0.78726 0.0068995
## 85  0.00027933    284   0.59139 0.78731 0.0068986
## 86  0.00027873    287   0.59054 0.78731 0.0068977
## 87  0.00027657    290   0.58967 0.78813 0.0069043
## 88  0.00027177    291   0.58940 0.78930 0.0069421
## 89  0.00027153    294   0.58858 0.78982 0.0069392
## 90  0.00027153    296   0.58804 0.78982 0.0069392
## 91  0.00027153    297   0.58777 0.78982 0.0069392
## 92  0.00027075    298   0.58749 0.78996 0.0069417
## 93  0.00027032    302   0.58641 0.78996 0.0069417
## 94  0.00026841    303   0.58614 0.78887 0.0069336
## 95  0.00026841    309   0.58392 0.78968 0.0069445
## 96  0.00026841    310   0.58365 0.78968 0.0069445
## 97  0.00026720    313   0.58285 0.78968 0.0069445
## 98  0.00026589    314   0.58258 0.79001 0.0069661
## 99  0.00026312    316   0.58205 0.79021 0.0069686
## 100 0.00026025    318   0.58152 0.79140 0.0069991
## 101 0.00025964    319   0.58126 0.79249 0.0070098
## 102 0.00025808    321   0.58074 0.79315 0.0070163
## 103 0.00025316    326   0.57934 0.79359 0.0070092
## 104 0.00025292    329   0.57846 0.79327 0.0070050
## 105 0.00025280    336   0.57666 0.79327 0.0070050
## 106 0.00025220    338   0.57615 0.79310 0.0070060
## 107 0.00025128    341   0.57534 0.79574 0.0070647
## 108 0.00024968    350   0.57248 0.79631 0.0070644
## 109 0.00024968    351   0.57223 0.79572 0.0070579
## 110 0.00024941    352   0.57198 0.79572 0.0070579
## 111 0.00024864    367   0.56733 0.79604 0.0070543
## 112 0.00024776    371   0.56626 0.79594 0.0070535
## 113 0.00024776    372   0.56601 0.79584 0.0070536
## 114 0.00024776    373   0.56576 0.79584 0.0070536
## 115 0.00024212    379   0.56428 0.79561 0.0070485
## 116 0.00024132    387   0.56189 0.79571 0.0070545
## 117 0.00023996    396   0.55933 0.79715 0.0070845
## 118 0.00023744    398   0.55885 0.79835 0.0070873
## 119 0.00023600    406   0.55675 0.79823 0.0070754
## 120 0.00023527    420   0.55318 0.79967 0.0070661
## 121 0.00023400    422   0.55271 0.79922 0.0070646
## 122 0.00023311    425   0.55201 0.79955 0.0070636
## 123 0.00023306    462   0.53981 0.79955 0.0070636
## 124 0.00023215    466   0.53887 0.79991 0.0070728
## 125 0.00023023    468   0.53841 0.80011 0.0070701
## 126 0.00022867    469   0.53818 0.80050 0.0070759
## 127 0.00022841    473   0.53726 0.80199 0.0071355
## 128 0.00022774    487   0.53296 0.80326 0.0071551
```

```
## 129 0.00022711    492   0.53182 0.80340 0.0071559
## 130 0.00022711    494   0.53137 0.80351 0.0071584
## 131 0.00022711    504   0.52910 0.80351 0.0071584
## 132 0.00022711    507   0.52842 0.80351 0.0071584
## 133 0.00022711    510   0.52774 0.80351 0.0071584
## 134 0.00022711    511   0.52751 0.80351 0.0071584
## 135 0.00022327    519   0.52547 0.80399 0.0071640
## 136 0.00022279    526   0.52364 0.80432 0.0071672
## 137 0.00022207    530   0.52250 0.80441 0.0071672
## 138 0.00022000    532   0.52205 0.80463 0.0071687
```

```
#confusion matrix on training data
table(pred=predict(rpDT1,lcdfTrn, type="class"), true=lcdfTrn$loan_status)
```

```
##              true
## pred          Fully Paid Charged Off
##   Fully Paid       33513        1772
##   Charged Off      14921        6509
```

```
predTrn1=predict(rpDT1,lcdfTrn, type="class")
mean(predTrn1 == lcdfTrn$loan_status)
```

```
## [1] 0.7056687
```

```
#confusion matrix on test data
table(pred=predict(rpDT1,lcdfTst, type="class"), true=lcdfTst$loan_status)
```

```
##              true
## pred          Fully Paid Charged Off
##   Fully Paid       13649        1547
##   Charged Off       7112        1999
```

```
predTst1=predict(rpDT1, lcdfTst, type='class')
mean(predTst1 == lcdfTst$loan_status)
```

```
## [1] 0.6437652
```

```
rpDT1$variable.importance
```

```
##                 int_rate                 sub_grade
##             2.821061e+03              2.814310e+03
##                    grade             bc_open_to_buy
##             2.316900e+03              9.262679e+02
##           total_bc_limit          total_rev_hi_lim
##             8.558698e+02              7.427034e+02
##               emp_length               avg_cur_bal
##             7.064697e+02              6.514431e+02
##           tot_hi_cred_lim                       dti
##             5.193992e+02              5.112571e+02
##      propLoanAmt_to_AnnInc         total_bal_ex_mort
##             4.732786e+02              4.356182e+02
##              installment                 loan_amnt
##             4.045556e+02              3.624980e+02
##                  bc_util                annual_inc
##             3.452639e+02              3.343131e+02
##       mo_sin_old_rev_tl_op         mo_sin_old_il_acct
##             3.227715e+02              3.014743e+02
##       mths_since_recent_bc   prop_OpAccts_to_TotAccts
##             2.954576e+02              2.808082e+02
##      mo_sin_rcnt_rev_tl_op total_il_high_credit_limit
##             2.613067e+02              2.555920e+02
##              num_bc_sats               num_op_rev_tl
##             2.507502e+02              2.287432e+02
##                  purpose                  num_sats
##             2.260236e+02              2.139925e+02
##                 mort_acc                 num_il_tl
##             2.134077e+02              2.094714e+02
##       acc_open_past_24mths     propSatisBankcardAccts
##             1.949714e+02              1.818906e+02
##           pct_tl_nvr_dlq               mo_sin_rcnt_tl
##             1.790054e+02              1.747671e+02
##             num_rev_accts                 num_bc_tl
##             1.712434e+02              1.537718e+02
##           home_ownership      mths_since_recent_inq
##             1.443202e+02              1.424510e+02
##        initial_list_status                tax_liens
##             1.500645e+01              1.316946e+01
##   chargeoff_within_12_mths               delinq_amnt
##             2.788897e+00              1.466074e+00
##              num_tl_30dpd
##             5.588072e-02
```

```
write.csv(rpDT1$variable.importance,"output_rpart_var_importance.csv")

## Set prior according to the class proportions. Increasing xerror, but higher accuracy.
rpDT2 <- rpart(loan_status ~., data=subset(lcdfTrn, select=-c(annRet, actualTerm, actualReturn,
 total_pymnt)), method="class", parms = list(split = "information", prior=c(1-0.171,0.171)), con
trol = rpart.control(minsplit = 30, cp=0.00))

#prior=c(0.5,0.5)),
rpDT2<-prune(rpDT2,cp=0.0002)
#printcp(rpDT2)

#confusion matrix on training data
table(pred=predict(rpDT2,lcdfTrn, type="class"), true=lcdfTrn$loan_status)
```

```
##              true
## pred           Fully Paid Charged Off
##   Fully Paid       47583        6420
##   Charged Off        851        1861
```

```
predTrn2=predict(rpDT2,lcdfTrn, type="class")
mean(predTrn2 == lcdfTrn$loan_status)
```

```
## [1] 0.8717976
```

```
#confusion matrix on test data
table(pred=predict(rpDT2,lcdfTst, type="class"), true=lcdfTst$loan_status)
```

```
##              true
## pred           Fully Paid Charged Off
##   Fully Paid       19977        3247
##   Charged Off        784         299
```

```
predTst2=predict(rpDT2, lcdfTst, type='class')
mean(predTst2 == lcdfTst$loan_status)
```

```
## [1] 0.834163
```

```
rpDT2$variable.importance
```

```
##                sub_grade                 int_rate
##             1476.5855312             1429.7379950
##                    grade           bc_open_to_buy
##             1214.2195108              440.4899045
##           total_bc_limit         total_rev_hi_lim
##              437.3026713              354.2690733
##               emp_length          tot_hi_cred_lim
##              253.9306762              221.7708933
##              avg_cur_bal              installment
##              192.3656348              180.6857437
##        total_bal_ex_mort                loan_amnt
##              170.4700320              168.6987207
##     propLoanAmt_to_AnnInc                      dti
##              153.8803969              129.3179512
## total_il_high_credit_limit      mths_since_recent_bc
##              127.7713710              127.7337393
##               annual_inc                  bc_util
##              123.2157855              120.4565764
##      mo_sin_old_rev_tl_op                  purpose
##              114.6782588              104.7278662
##        mo_sin_old_il_acct   propSatisBankcardAccts
##               95.8216878               93.8632218
##   prop_OpAccts_to_TotAccts             num_rev_accts
##               90.1359283               88.2400660
##              num_bc_sats     mo_sin_rcnt_rev_tl_op
##               84.3590314               83.4167852
##      acc_open_past_24mths                num_il_tl
##               81.8728575               81.6034817
##                 num_sats                num_bc_tl
##               78.4902304               76.6865324
##            mo_sin_rcnt_tl    mths_since_recent_inq
##               74.5078692               74.4267049
##              num_op_rev_tl                mort_acc
##               69.5899367               58.8109277
##             pct_tl_nvr_dlq           home_ownership
##               55.2445563               32.2071251
##        initial_list_status              num_tl_30dpd
##                9.4829707                0.6227979
##                 tax_liens               delinq_amnt
##                0.3044544                0.2428237
```

```
#write.csv(rpDT2$variable.importance,"output_rpart_var_importance.csv") #This code allows us to
 print the variable importance in excel format

#AUC ROC rpDT2
#On Train Data
predTrnProb_rpDT2=predict(rpDT2, lcdfTrn, type='prob')
predTrnProb_rpDT2_FP <- predTrnProb_rpDT2[, 'Fully Paid']

auc_rpDT2_Trn <- auc(lcdfTrn$loan_status, predTrnProb_rpDT2_FP)
```

```
## Setting levels: control = Fully Paid, case = Charged Off
```

```
## Setting direction: controls > cases
```

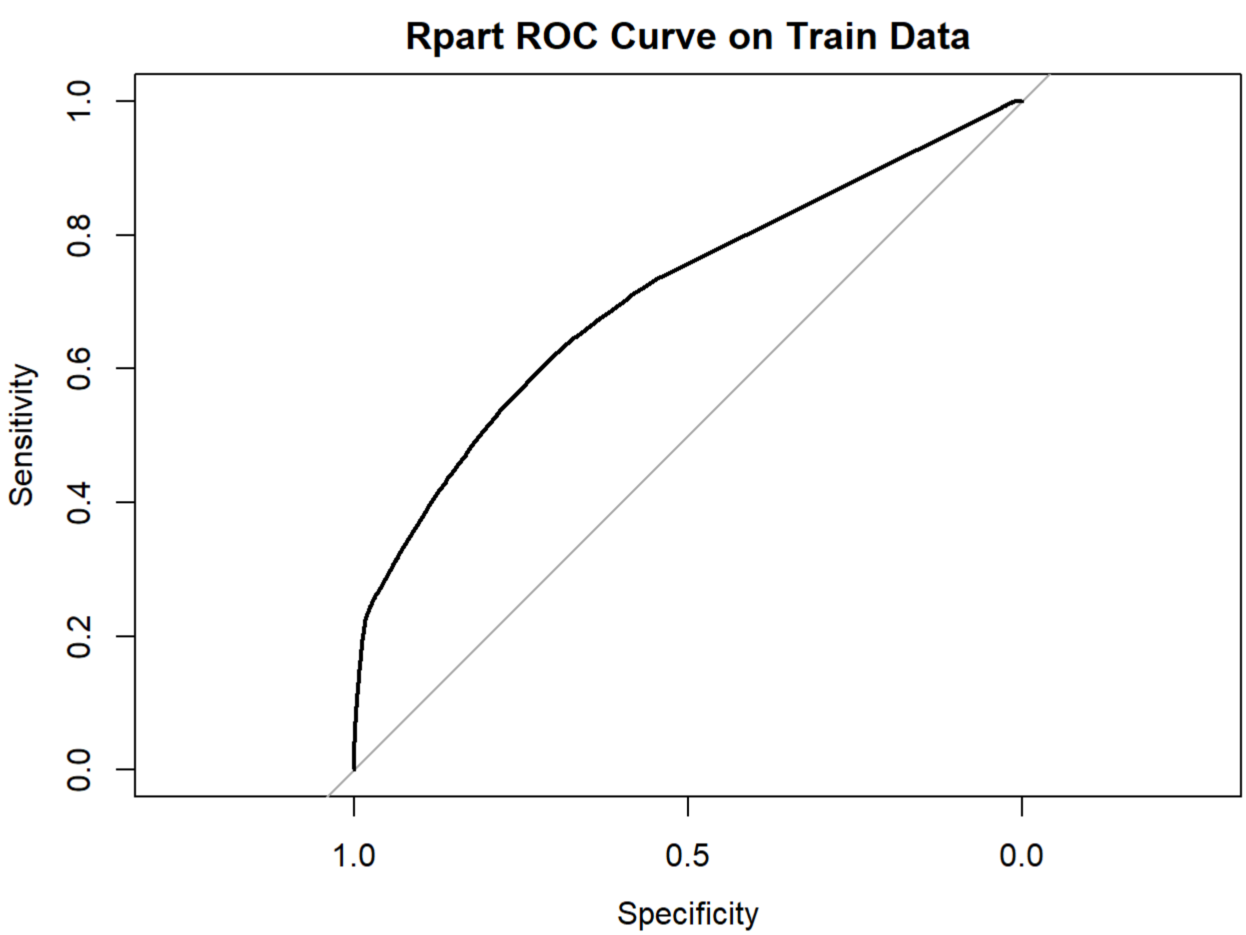
```
auc_rpDT2_Trn
```

```
## Area under the curve: 0.7095
```

```
rpDT2_roc_Trn = roc(lcdfTrn$loan_status, predTrnProb_rpDT2_FP)
```

```
## Setting levels: control = Fully Paid, case = Charged Off
## Setting direction: controls > cases
```

```
plot(rpDT2_roc_Trn, main = "Rpart ROC Curve on Train Data")
```




```
#On Test Data
predTstProb_rpDT2=predict(rpDT2, lcdfTst, type='prob')
predTstProb_rpDT2_FP <- predTstProb_rpDT2[, 'Fully Paid']

auc_rpDT2_Tst <- auc(lcdfTst$loan_status, predTstProb_rpDT2_FP)
```

```
## Setting levels: control = Fully Paid, case = Charged Off
## Setting direction: controls > cases
```

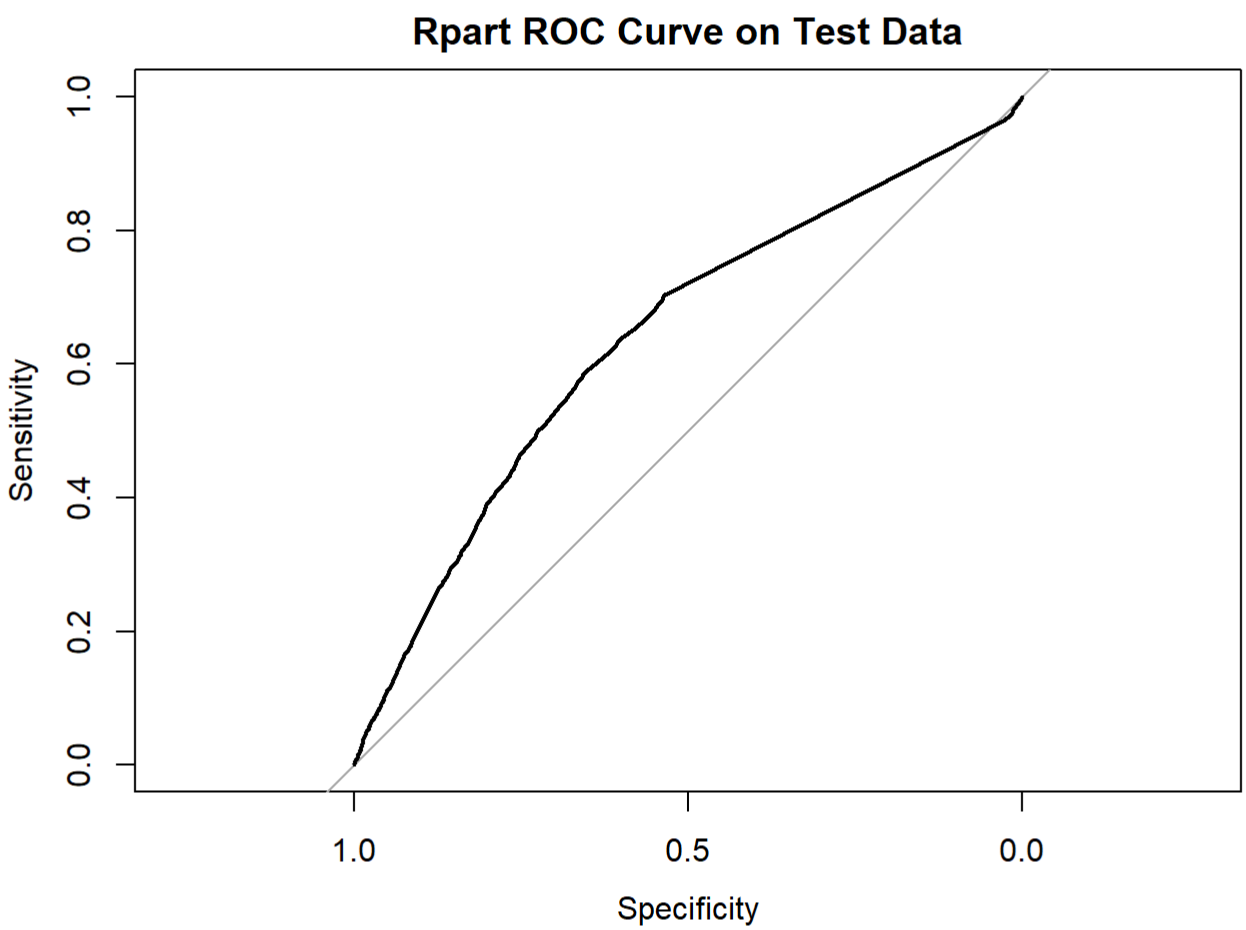
```
auc_rpDT2_Tst
```

```
## Area under the curve: 0.639
```

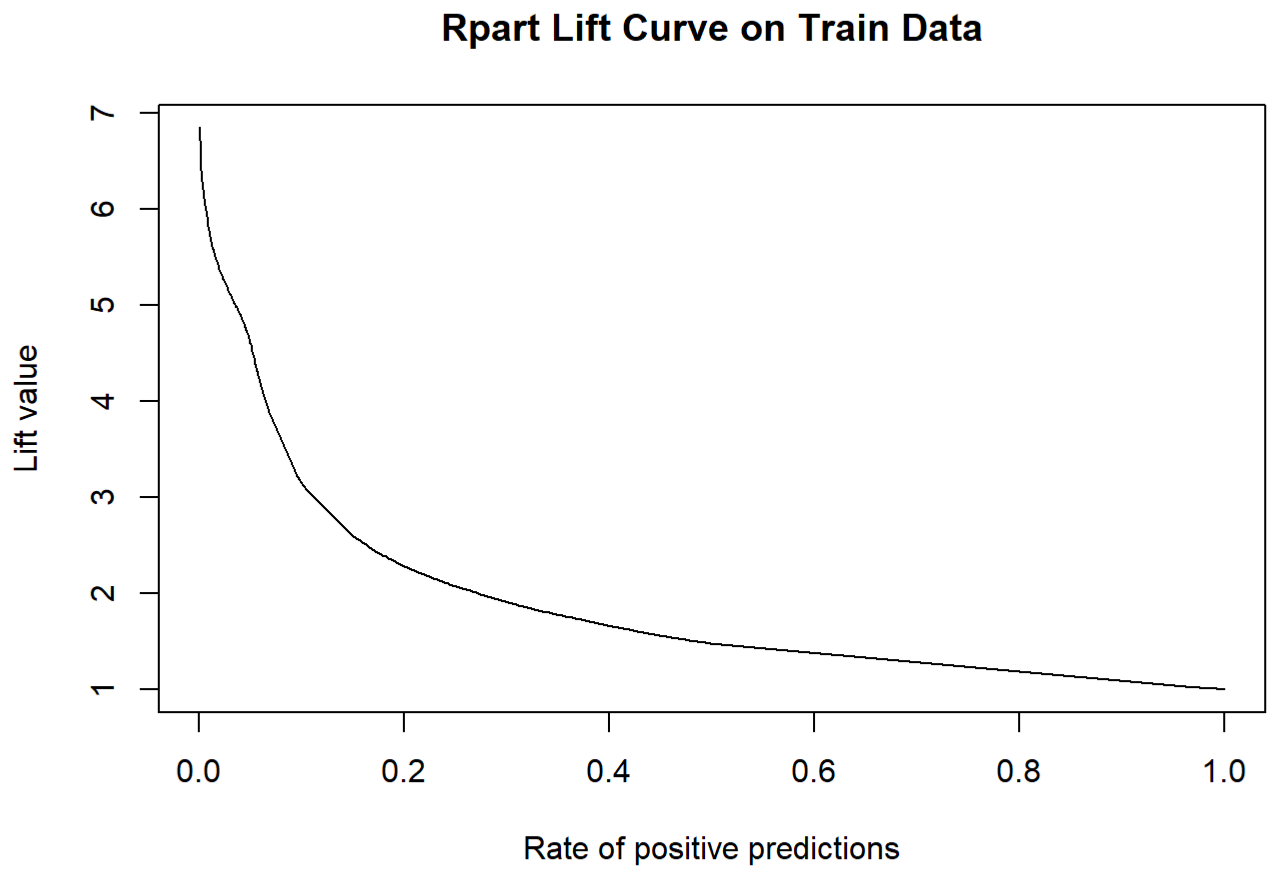
```
rpDT2_roc_Tst = roc(lcdfTst$loan_status, predTstProb_rpDT2_FP)
```

```
## Setting levels: control = Fully Paid, case = Charged Off
## Setting direction: controls > cases
```

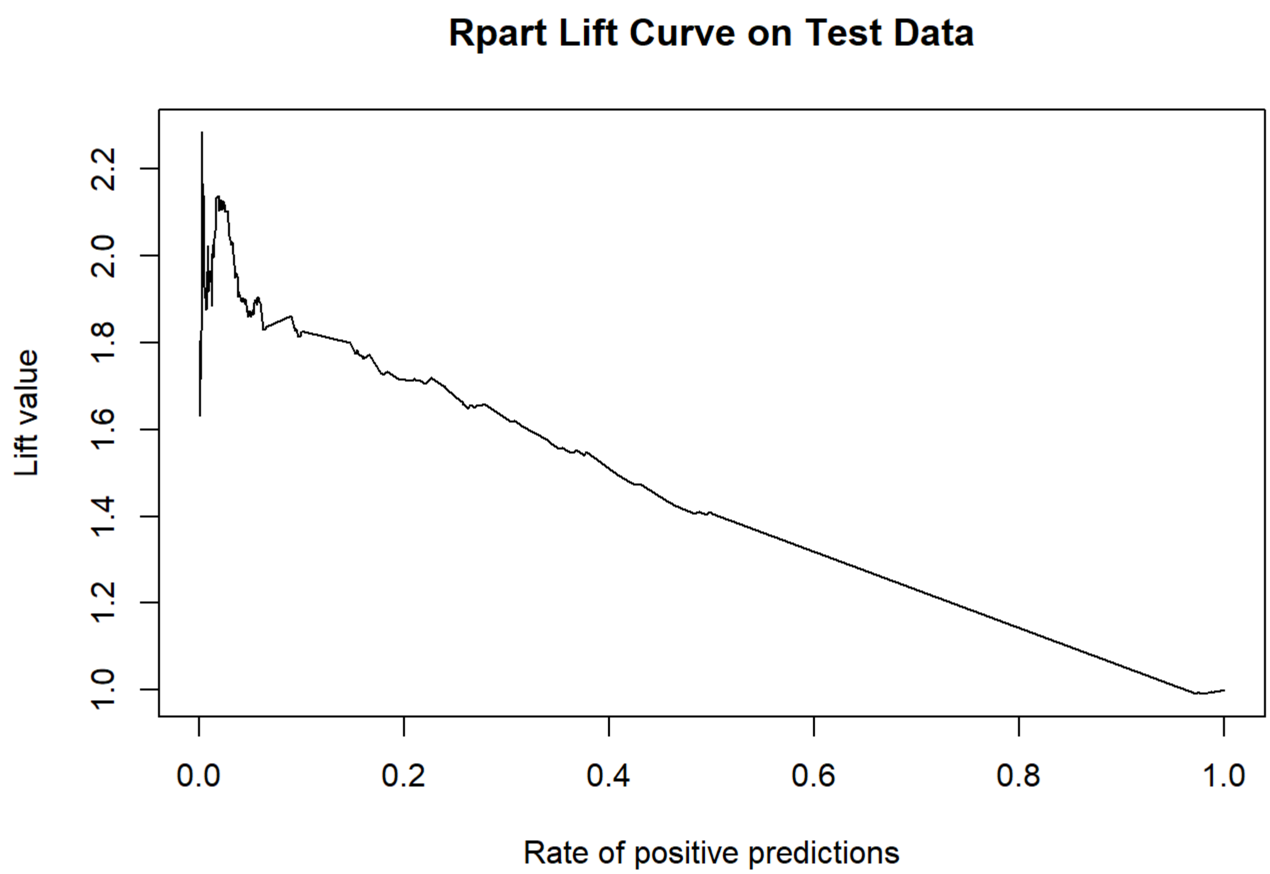
```
plot(rpDT2_roc_Tst, main = "Rpart ROC Curve on Test Data")
```




```
#Lift Curve on Train Data
scoreTrn_rpart <- predict(rpDT2, lcdfTrn, type="prob")[,'Charged Off']
rocPredTrn_rpart <- prediction(scoreTrn_rpart, lcdfTrn$loan_status, label.ordering = c('Fully Pa
id', 'Charged Off'))
liftPerf_rpart_Trn <-performance(rocPredTrn_rpart, "lift", "rpp")
plot(liftPerf_rpart_Trn, main = "Rpart Lift Curve on Train Data")
```




```
#Lift Curve on Test Data
scoreTst_rpart <- predict(rpDT2, lcdfTst, type="prob")[,'Charged Off']
rocPredTst_rpart <- prediction(scoreTst_rpart, lcdfTst$loan_status, label.ordering = c('Fully Pa
id', 'Charged Off'))
liftPerf_rpart_Tst <-performance(rocPredTst_rpart, "lift", "rpp")
plot(liftPerf_rpart_Tst, main = "Rpart Lift Curve on Test Data")
```



Build Decision Tree using C50

To treat the unbalanced distribution within classes as we did in rpart, we upsampled the minority cases using caret to create a balanced dataset. We then passed the upsampled training set to the model, and we experimented with the parameters in C5.0. Please refer to **Table 2 in Appendix** for the complete tabulation of our parameter tuning.

Our C5.0 Decision Tree is prone to overfit. Our model was returning very good results with training set, but the accuracy dropped severely for test data. With the limitations in parameters available in C5.0, we tried setting multiple values to minCases. At minCases = 10, we got the best accuracy scores. However the difference between Training and Testing accuracy is so profound (23%) that we believe this model is not stable enough. The best accuracy we got from the C5.0 model is much less than that of rpart, with the minCases of 80. It is a good balance between good accuracy scores and less severe overfit.

We store the best model in c5_DT1. The evaluation of c5_DT1 is done with AUC, ROC, and confusion matrices.

```
library(C50)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
#upsample "Charged Off" in lcdfTrn
up_lcdfTrn <- upSample(x = lcdfTrn[, -ncol(lcdfTrn)], y = lcdfTrn$loan_status) %>% select(-c("Cl
ass"))
table(up_lcdfTrn$loan_status)
```

```
##
##  Fully Paid Charged Off
##       48434       48434
```

```
up_lcdfTst <- upSample(x = lcdfTst[, -ncol(lcdfTst)], y = lcdfTst$loan_status) %>% select(-c("Cl
ass"))
table(up_lcdfTst$loan_status)
```

```
##
##  Fully Paid Charged Off
##       20761       20761
```

```
c5_DT1 <- C5.0(as.factor(up_lcdfTrn$loan_status) ~ ., data=subset(up_lcdfTrn, select=-c(annRet,
actualTerm, actualReturn, total_pymnt)), method = "class", control = C5.0Control(minCases = 80))
#summary(c5_DT1)


print(C5imp(c5_DT1))
```

```
##                             Overall
## int_rate                     100.00
## avg_cur_bal                   93.50
## emp_length                    72.82
## installment                   67.01
## sub_grade                     62.19
## purpose                       61.11
## acc_open_past_24mths          55.14
## grade                         47.36
## mort_acc                      46.74
## dti                           46.18
## mo_sin_old_il_acct            44.49
## bc_util                       43.84
## num_sats                      43.65
## prop_OpAccts_to_TotAccts      41.68
## tax_liens                     41.43
## mths_since_recent_bc          41.20
## num_bc_tl                     38.61
## pct_tl_nvr_dlq                36.42
## total_bc_limit                32.46
## mo_sin_rcnt_tl                31.49
## num_bc_sats                   29.76
## tot_hi_cred_lim               29.52
## total_bal_ex_mort             26.00
## mo_sin_rcnt_rev_tl_op         24.63
## propSatisBankcardAccts        23.39
## home_ownership                21.72
## mo_sin_old_rev_tl_op          20.13
## total_rev_hi_lim              20.00
## bc_open_to_buy                19.83
## num_op_rev_tl                 18.28
## loan_amnt                     17.54
## chargeoff_within_12_mths      17.46
## num_rev_accts                 17.13
## annual_inc                    14.16
## mths_since_recent_inq         13.59
## num_il_tl                     10.16
## total_il_high_credit_limit    10.15
## initial_list_status            2.74
## delinq_amnt                    0.00
## num_tl_30dpd                   0.00
```

```
#temp<-C5imp(c5_DT1)
#write.csv(temp,"output_c50_var_importance.csv")
#plot(c5_DT1)


predTrnC5<-predict(c5_DT1,lcdfTrn, type='class')
table( predTrnC5, true=lcdfTrn$loan_status)
```

```
##               true
## predTrnC5     Fully Paid Charged Off
##   Fully Paid       33662        2039
##   Charged Off      14772        6242
```

```
mean(predTrnC5 == lcdfTrn$loan_status)
```

```
## [1] 0.7035881
```

```
table(pred = predict(c5_DT1,lcdfTst, type='class'), true=lcdfTst$loan_status)
```

```
##               true
## pred          Fully Paid Charged Off
##   Fully Paid       13661        1747
##   Charged Off       7100        1799
```

```
mean(predict(c5_DT1,lcdfTst, type='class') ==lcdfTst$loan_status)
```

```
## [1] 0.6360308
```

```
#ROC AUC
#On Train Data
predTrnProb_c5=predict(c5_DT1, lcdfTrn, type='prob')
predTrnProb_c5_FP <- predTrnProb_c5[, 'Fully Paid']

auc_c5_Trn <- auc(lcdfTrn$loan_status, predTrnProb_c5_FP)
```

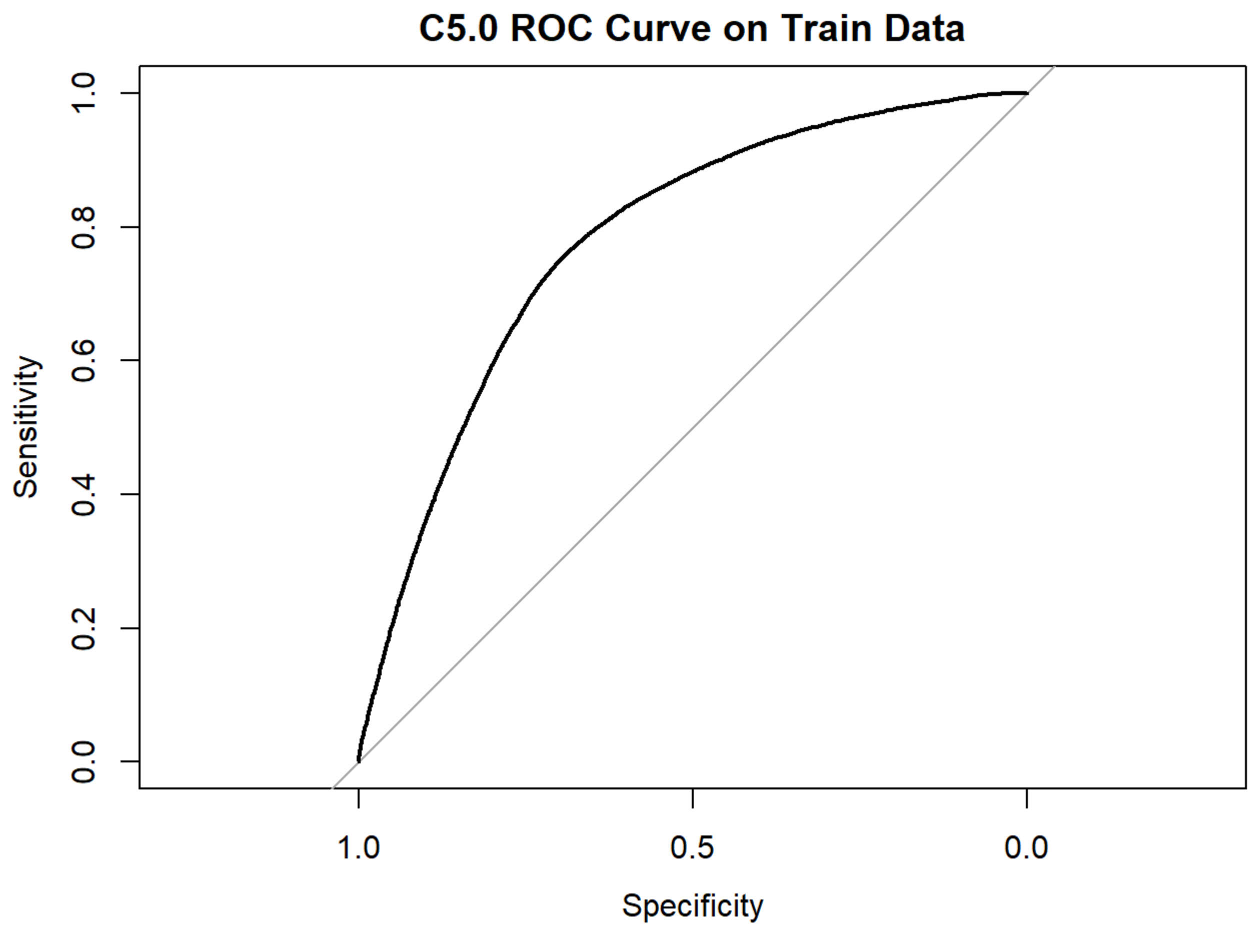
```
## Setting levels: control = Fully Paid, case = Charged Off
```

```
## Setting direction: controls > cases
```

```
c5_roc_Trn <- roc(lcdfTrn$loan_status, predTrnProb_c5_FP)
```

```
## Setting levels: control = Fully Paid, case = Charged Off
## Setting direction: controls > cases
```

```
plot(c5_roc_Trn, main = "C5.0 ROC Curve on Train Data")
```



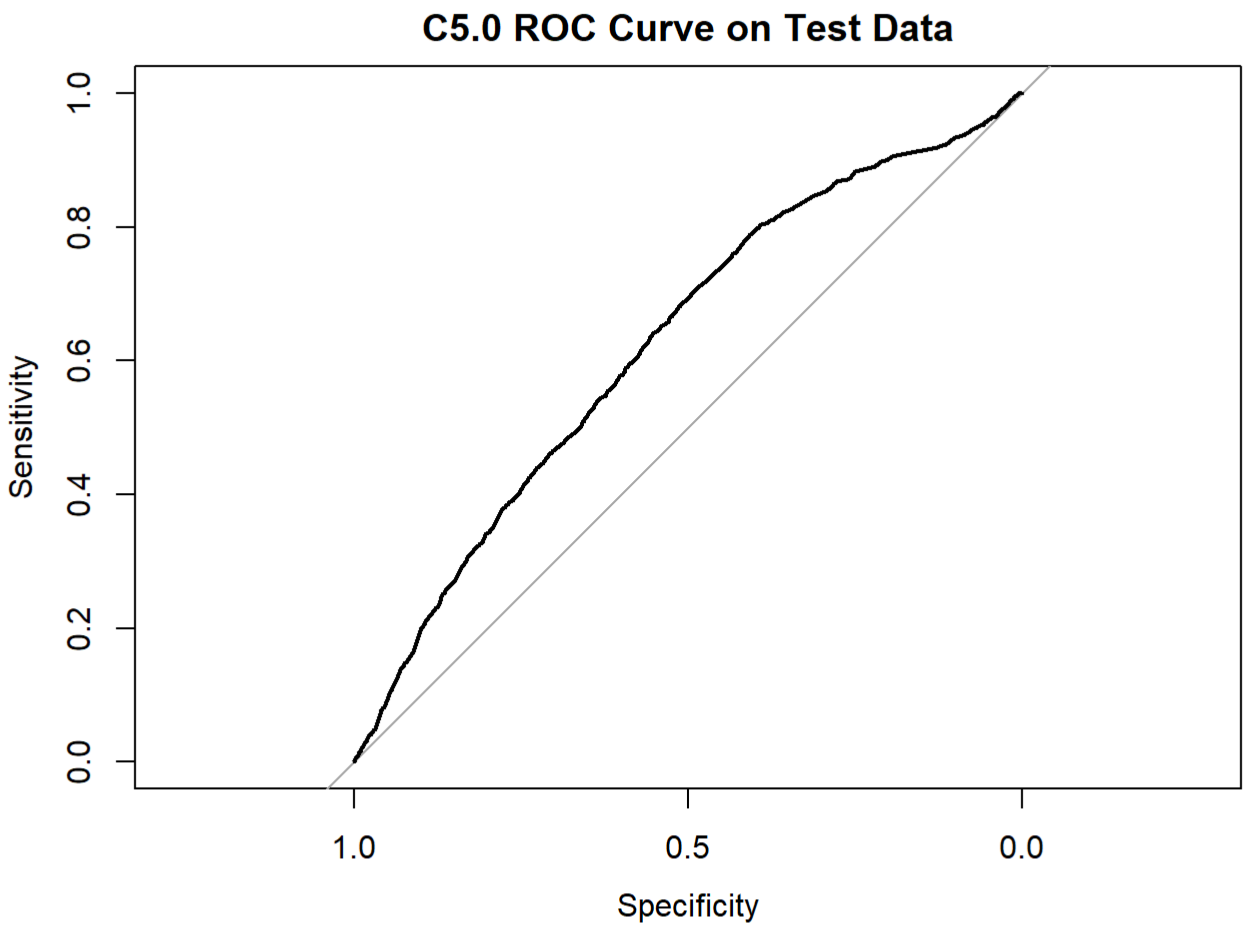

```
#On Test Data
predTstProb_c5=predict(c5_DT1, lcdfTst, type='prob')
predTstProb_c5_FP <- predTstProb_c5[, 'Fully Paid']

auc_c5_Tst <- auc(lcdfTst$loan_status, predTstProb_c5_FP)
```

```
## Setting levels: control = Fully Paid, case = Charged Off
## Setting direction: controls > cases
```

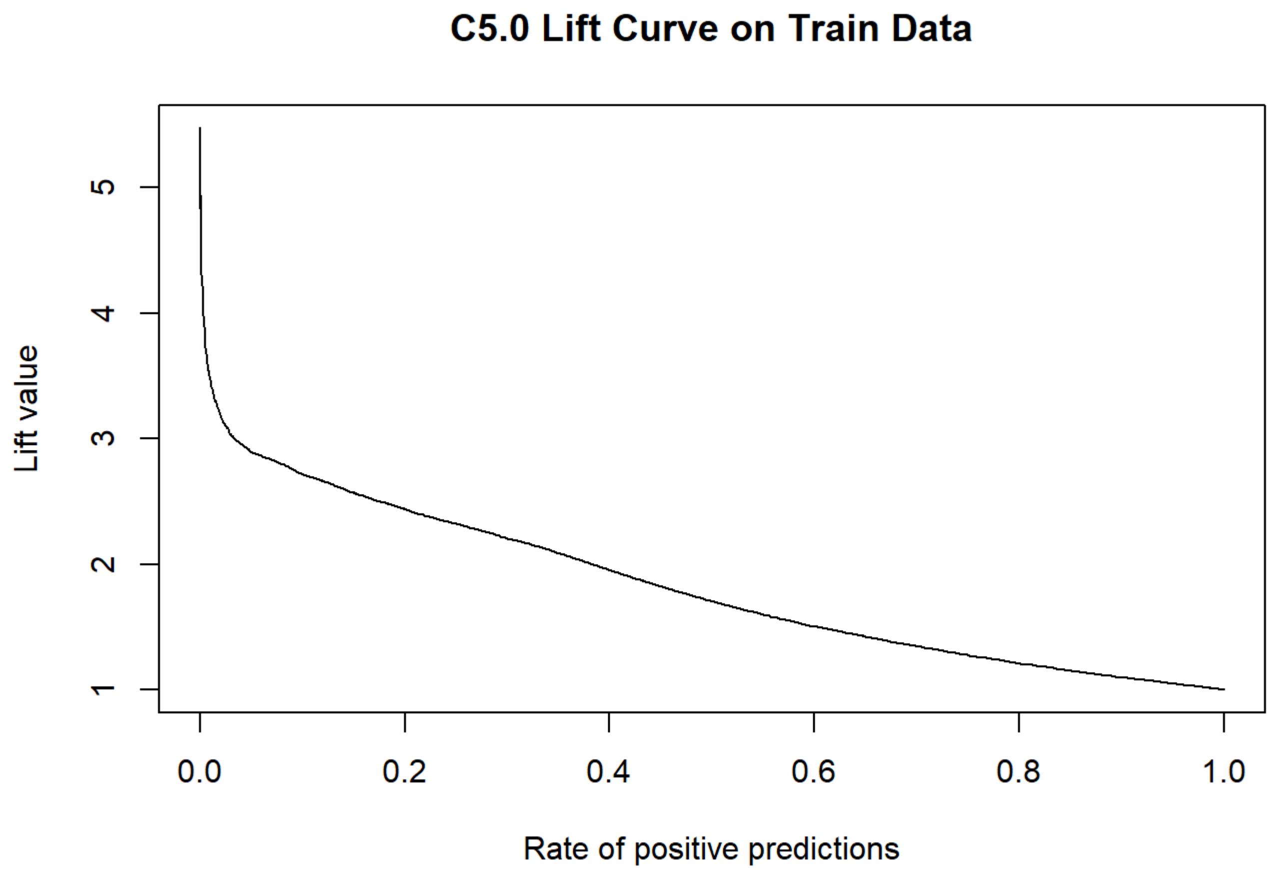
```
c5_roc_Tst <- roc(lcdfTst$loan_status, predTstProb_c5_FP)
```

```
## Setting levels: control = Fully Paid, case = Charged Off
## Setting direction: controls > cases
```

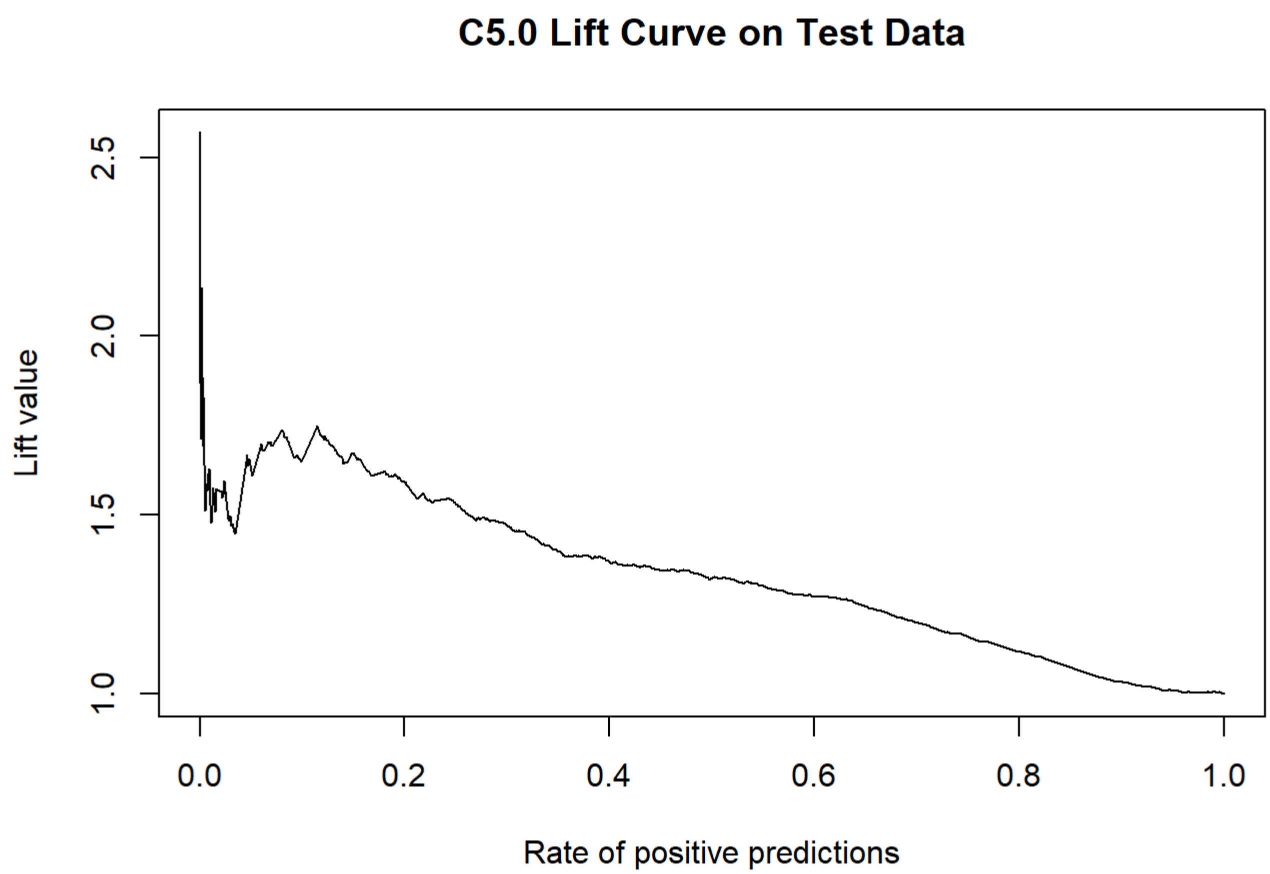
```
plot(c5_roc_Tst, main = "C5.0 ROC Curve on Test Data")
```




```
#Lift Curve on Train Data
scoreTrn_c50 <- predict(c5_DT1, lcdfTrn, type="prob")[,'Charged Off']
rocPredTrn_c50 <- prediction(scoreTrn_c50, lcdfTrn$loan_status, label.ordering = c('Fully Paid',
'Charged Off'))
liftPerf_c50_Trn <-performance(rocPredTrn_c50, "lift", "rpp")
plot(liftPerf_c50_Trn, main = "C5.0 Lift Curve on Train Data")
```




```
#Lift Curve on Test Data
scoreTst_c50 <- predict(c5_DT1, lcdfTst, type="prob")[,'Charged Off']
rocPredTst_c50 <- prediction(scoreTst_c50, lcdfTst$loan_status, label.ordering = c('Fully Paid',
'Charged Off'))
liftPerf_c50 <-performance(rocPredTst_c50, "lift", "rpp")
plot(liftPerf_c50, main = "C5.0 Lift Curve on Test Data")
```



Q6 Analysis

We built our Random Forest model using the ranger package. Although there are many parameters to play with, we found that drastically changing some of these parameters do not yield different results. Our RF models were heavily overfitting, as the trees by themselves are very complex (max depth 30, min node 20). Our original training accuracy was around 97% and testing accuracy around 68%. To solve this issue, we decreased the number of trees from 1000 to 500 and below in hope to get simpler models. We picked out three best models out of our many attempts to build the best model. The most effective number of trees hovers around 200-400. We also decreased the max depth and increase the min node parameter.

Please refer to **Table 3 in Appendix** for the complete tabulation of our best ranger models. Out of the top three alternatives, we resort to the simplest model that bears the same Testing accuracy but with slightly milder overfit than its counterparts. The final model is stored in rgModel1. The evaluation of rgModel1 is done with AUC, ROC, confusion matrices, and lift curve.

Tree parameters tuning principles: - num of trees should be around 10 x num of variables (avoid overfit) - mtry by default in ranger is sqrt of p, but because we have noisy predictors, we will set a higher mtry. We declare our data “noisy” because of the relatively low average AUC scores. No variable scored higher than 70% in AUC score. - Tree tuning parameters: we should build less complex tree, because we are keeping higher values of mtry (our data has many noisy predictors). What we are doing: limit the tree depth, bigger node size, etc. - To maintain low correlation between trees, we will set a lower sample.fraction. The default is 1, so we will set it to a fraction lower than 1. - respect.unordered.factors set to TRUE for better results. - replace = TRUE, to generate proper bootstrap samples. - set seed to create reproducible results.

https://bradleyboehmke.github.io/HOML/random-forest.html (https://bradleyboehmke.github.io/HOML/random-forest.html)

```
library(ranger)

rgModel1 <- ranger(loan_status ~., data=subset(up_lcdfTrn, select=-c(annRet, actualTerm, actualReturn, total_pymnt)), num.trees =200, importance='permutation', mtry = 7, max.depth = 10, min.node.size = 30, sample.fraction = 0.5, replace=FALSE, respect.unordered.factors = "order" , verbose = TRUE , seed=0)
```

```
## Computing permutation importance.. Progress: 41%. Estimated remaining time: 47 seconds.
## Computing permutation importance.. Progress: 94%. Estimated remaining time: 4 seconds.
```

```
#Summary()
vimpRg_1 <- ranger::importance(rgModel1)
#write.csv(vimpRg_1,"output_Ranger_var_importance.csv")

#Predict model using training data
predTrn<- predict(rgModel1,up_lcdfTrn)
predTst<- predict(rgModel1,up_lcdfTst)

#create confusion matrix for Training data
Conf_Trn<-table(predictions(predTrn), up_lcdfTrn$loan_status)
confusionMatrix(Conf_Trn,positive = "Charged Off")
```

```
## Confusion Matrix and Statistics
##
##
##               Fully Paid Charged Off
##   Fully Paid       30019        9212
##   Charged Off      18415       39222
##
##                Accuracy : 0.7148
##                  95% CI : (0.7119, 0.7176)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4296
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8098
##             Specificity : 0.6198
##          Pos Pred Value : 0.6805
##          Neg Pred Value : 0.7652
##              Prevalence : 0.5000
##          Detection Rate : 0.4049
##    Detection Prevalence : 0.5950
##       Balanced Accuracy : 0.7148
##
##        'Positive' Class : Charged Off
##
```

```
#create confusion matrix for Test data
Conf_Tst<-table(predictions(predTst), up_lcdfTst$loan_status)
confusionMatrix(Conf_Tst,positive = "Charged Off")
```

```
## Confusion Matrix and Statistics
## 
## 
##               Fully Paid Charged Off
##   Fully Paid       12473        6791
##   Charged Off       8288       13970
## 
##                Accuracy : 0.6368
##                  95% CI : (0.6322, 0.6415)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.2737
## 
##  Mcnemar's Test P-Value : < 2.2e-16
## 
##             Sensitivity : 0.6729
##             Specificity : 0.6008
##          Pos Pred Value : 0.6276
##          Neg Pred Value : 0.6475
##              Prevalence : 0.5000
##          Detection Rate : 0.3364
##    Detection Prevalence : 0.5361
##       Balanced Accuracy : 0.6368
## 
##        'Positive' Class : Charged Off
## 
```

Our ranger model has the highest accuracy so far. The training data accuracy is higher than the test accuracy, as expected. There is some mild overfitting, which has been greatly reduced by our parameter tuning.

```
#Build a probability type ranger from our rgModel1 for AUC and ROC
rgModelProb <- ranger(loan_status ~., data=subset(lcdfTrn, select=-c(annRet, actualTerm, actualReturn, total_pymnt)),
num.trees =200, importance='permutation', mtry = 7, max.depth = 10, min.node.size = 30, sample.fraction = 0.5, replace=FALSE, respect.unordered.factors = "order" , verbose = TRUE , seed=0, probability = TRUE)

scoreTrn <- predict(rgModelProb,lcdfTrn)
#head(scoreTrn)$predictions

scoreTrn_FP <- scoreTrn$predictions[,"Fully Paid"]
vimpRg_1 <- ranger::importance(rgModelProb)
vimpRg_1
```

```
##                loan_amnt                    int_rate
##             1.626977e-03                3.584207e-03
##              installment                       grade
##             1.930910e-03                2.286640e-03
##                sub_grade                  emp_length
##             3.615613e-03                1.358502e-04
##           home_ownership                  annual_inc
##             2.810281e-04                2.034758e-03
##                  purpose                         dti
##             8.416420e-05                1.146663e-03
##      initial_list_status             total_rev_hi_lim
##             1.035574e-06                2.637738e-03
##     acc_open_past_24mths                 avg_cur_bal
##             1.520575e-03                3.603743e-03
##           bc_open_to_buy                     bc_util
##             2.621961e-03                1.067853e-03
##   chargeoff_within_12_mths                 delinq_amnt
##             6.798860e-06               -6.630420e-06
##        mo_sin_old_il_acct        mo_sin_old_rev_tl_op
##             3.563899e-04                5.215287e-04
##     mo_sin_rcnt_rev_tl_op               mo_sin_rcnt_tl
##             2.544117e-04                3.731402e-04
##                 mort_acc        mths_since_recent_bc
##             4.753646e-04                4.034827e-04
##     mths_since_recent_inq                 num_bc_sats
##             1.369951e-04                6.666955e-04
##                num_bc_tl                   num_il_tl
##             7.662004e-04                2.938669e-04
##              num_op_rev_tl               num_rev_accts
##             8.501814e-04                9.783811e-04
##                 num_sats                num_tl_30dpd
##             7.249467e-04                9.603233e-06
##            pct_tl_nvr_dlq                   tax_liens
##             1.266186e-04                1.289679e-05
##           tot_hi_cred_lim           total_bal_ex_mort
##             5.321540e-03                1.248459e-03
##           total_bc_limit total_il_high_credit_limit
##             2.978520e-03                1.011715e-03
##    propSatisBankcardAccts   prop_OpAccts_to_TotAccts
##             3.219610e-04                6.099329e-04
##     propLoanAmt_to_AnnInc
##             1.117700e-03
```

```
#evaluate AUC and ROC
auc_rg_Trn <- auc(lcdfTrn$loan_status, scoreTrn_FP)
```

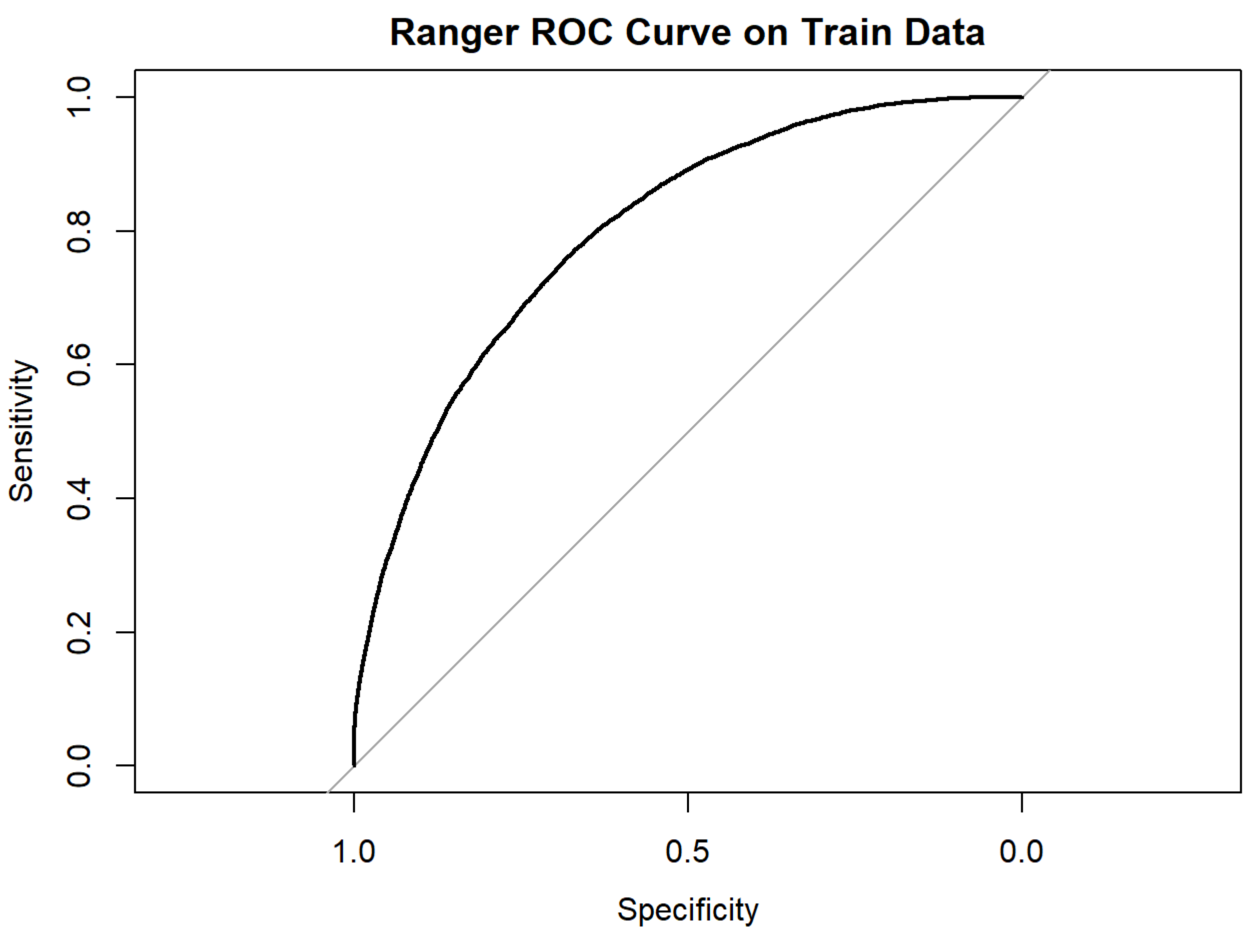
```
## Setting levels: control = Fully Paid, case = Charged Off
```

```
## Setting direction: controls > cases
```

```
rg_roc_Trn <- roc(lcdfTrn$loan_status, scoreTrn_FP)
```

```
## Setting levels: control = Fully Paid, case = Charged Off
## Setting direction: controls > cases
```

```
plot(rg_roc_Trn, main = "Ranger ROC Curve on Train Data")
```



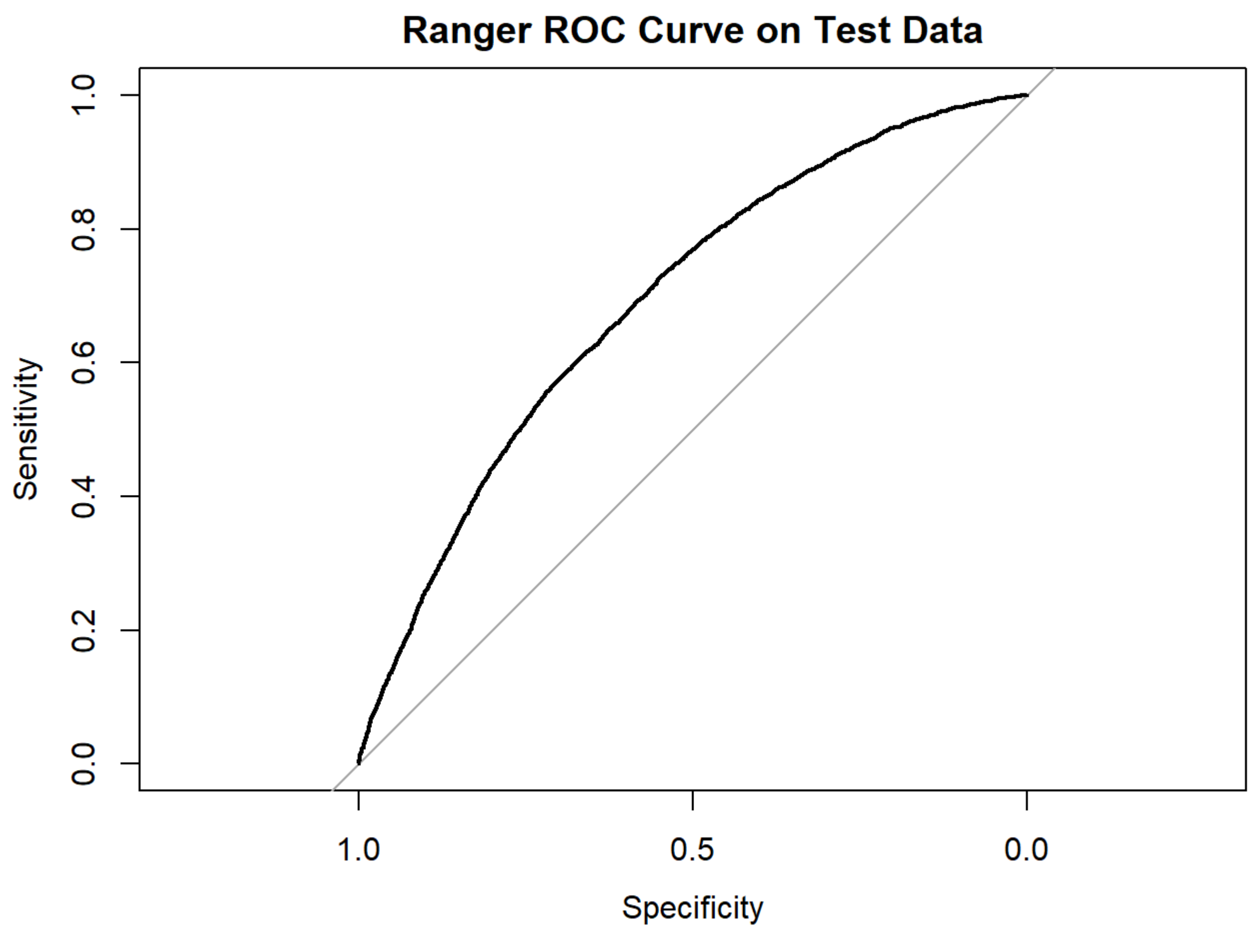

```
#Test data ROC AUC
scoreTst <- predict(rgModelProb,lcdfTst)
#head(scoreTst)$predictions

scoreTst_FP <- scoreTst$predictions[,"Fully Paid"]

#evaluate AUC and ROC
auc_rg_Tst <- auc(lcdfTst$loan_status, scoreTst_FP)
```

```
## Setting levels: control = Fully Paid, case = Charged Off
## Setting direction: controls > cases
```

```
rg_roc_Tst <- roc(lcdfTst$loan_status, scoreTst_FP)
```

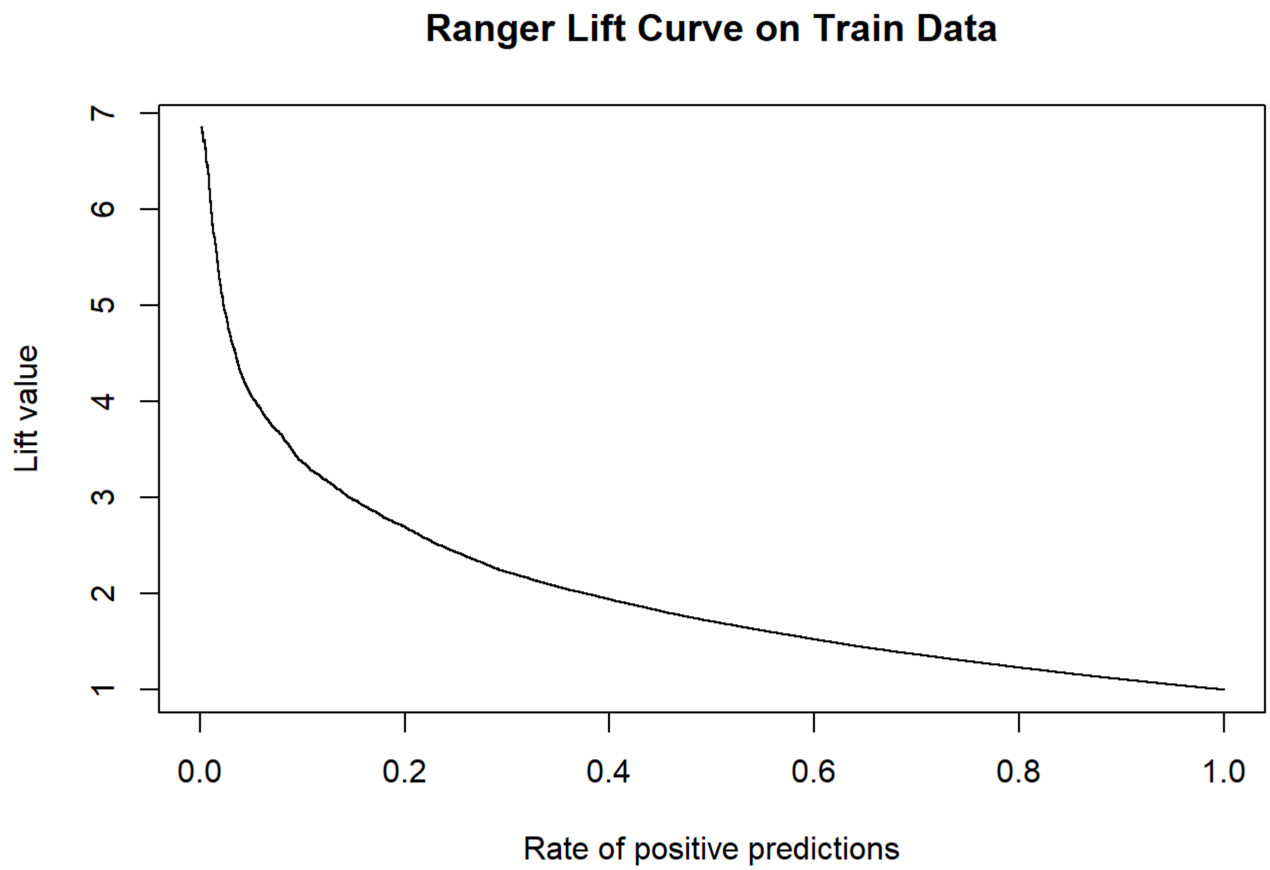
```
## Setting levels: control = Fully Paid, case = Charged Off
## Setting direction: controls > cases
```

```
plot(rg_roc_Tst, main = "Ranger ROC Curve on Test Data")
```

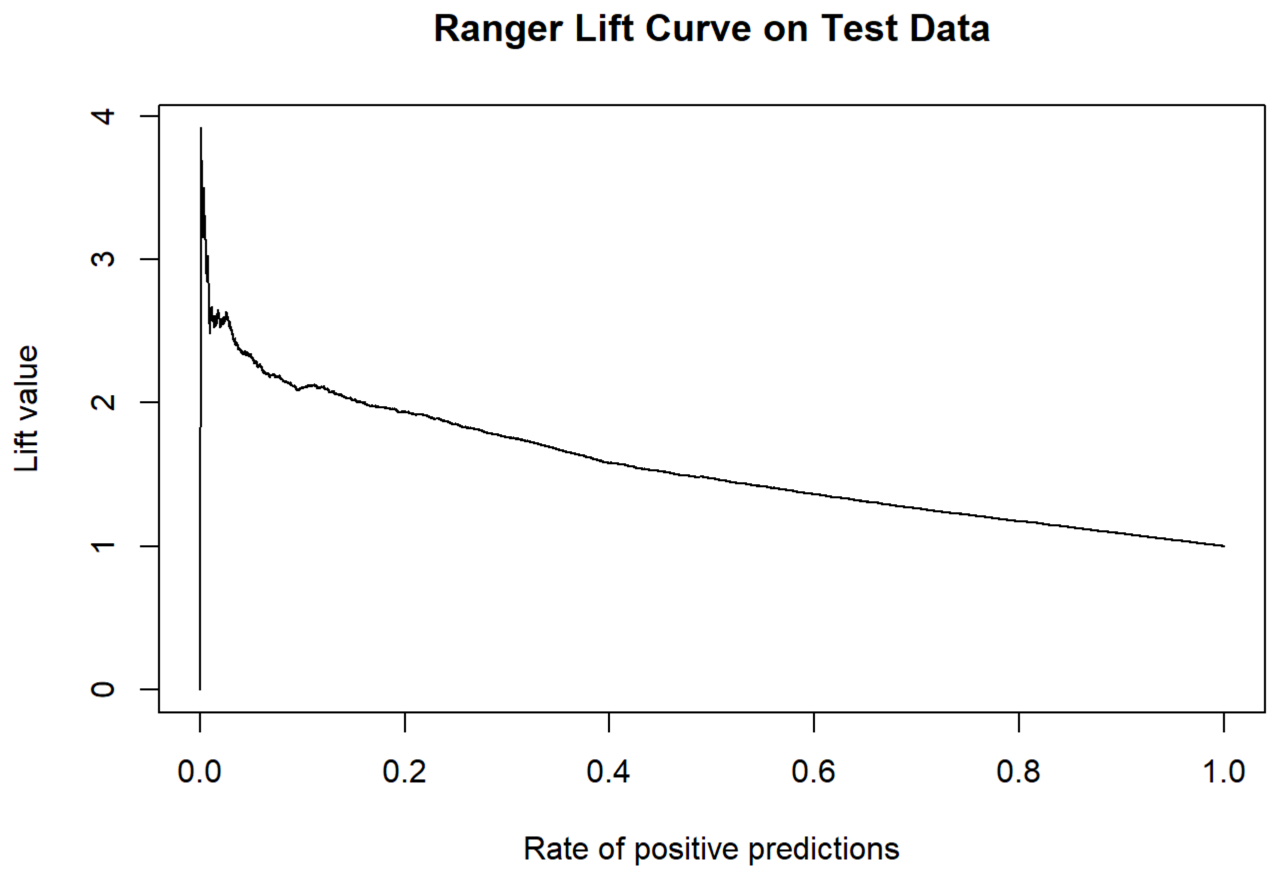


The lift curve shows that our ranger model has a predicting value, although the results are not exponentially better than no model scenario. This is consistent with the accuracy depicted in the ROC curve.

```
#Lift Curve on Train Data
scoreTrn_CO <- scoreTrn$predictions[,"Charged Off"]

rocPredTrn_rg <- prediction(scoreTrn_CO, lcdfTrn$loan_status, label.ordering = c('Fully Paid','C
harged Off'))

liftPerf_rg_Trn <-performance(rocPredTrn_rg, "lift", "rpp")
plot(liftPerf_rg_Trn, main = "Ranger Lift Curve on Train Data")
```




```
#Lift Curve on Test Data
scoreTst_CO <- scoreTst$predictions[,"Charged Off"]

rocPredTst_rg <- prediction(scoreTst_CO, lcdfTst$loan_status, label.ordering = c('Fully Paid','C
harged Off'))

liftPerf_rg_Tst <-performance(rocPredTst_rg, "lift", "rpp")
plot(liftPerf_rg_Tst, main = "Ranger Lift Curve on Test Data")
```



Q7 Analysis:

7a

This part of the assignment seems like a natural extension to question 2(V). Some aspect of the questions have been answered in 2(V). For the sake of completion we have decided to repeat our previous analysis for the purpose of providing reader a exhaustive report. We have previously calculated/received several factors such as interest rate, average interest rate, average return rate.

Interest rate cannot be direct indicator of profit because we also have to consider the amount we investor has put in to earn the money. Using the well known formula for profit: Profit= Revenue - Cost

total payment based on Interest rate basically present the overall revenue from which we need to get rid of Cost price which in this case in funded amount. We get actual return using the mentioned method which we divide by duration to achieve the annual return aka actual interest rate to achieve the annual return.

In the table below we have presented an average estimate about how the return looks like in 3 year. we have considered the fact that average duration for paid off loans is not 3 years but averages around 2.1 year. We have added 2 % certificate of deposit rate which is the risk free rate. 3 year rate is averaging around 8% and even reaches up to 10% for most profitable business.

```
lcdf %>% group_by(grade) %>% tally()
```

```
## # A tibble: 7 x 2
##   grade     n
##   <fct> <int>
## 1 A     20402
## 2 B     23399
## 3 C     22577
## 4 D     10802
## 5 E      3191
## 6 F       560
## 7 G        91
```

```
lcdf %>% group_by(grade) %>% summarise(mean(loan_amnt))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 7 x 2
##   grade `mean(loan_amnt)`
##   <fct>             <dbl>
## 1 A                14146.
## 2 B                12458.
## 3 C                11466.
## 4 D                12150.
## 5 E                12558.
## 6 F                10169.
## 7 G                12509.
```

```
lcdf %>% group_by(loan_status, grade) %>% tally()
```

```
## # A tibble: 14 x 3
## # Groups:   loan_status [2]
##    loan_status grade     n
##    <fct>       <fct> <int>
##  1 Fully Paid  A     19294
##  2 Fully Paid  B     20717
##  3 Fully Paid  C     18461
##  4 Fully Paid  D      8155
##  5 Fully Paid  E      2146
##  6 Fully Paid  F       369
##  7 Fully Paid  G        53
##  8 Charged Off A      1108
##  9 Charged Off B      2682
## 10 Charged Off C      4116
## 11 Charged Off D      2647
## 12 Charged Off E      1045
## 13 Charged Off F       191
## 14 Charged Off G        38
```

```
lcdf %>% 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_pymnt))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # 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      23399     2682       10.7       1.22       12458.  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.
```

```
lcdf %>% group_by(grade) %>% summarise(nLoans=n(), defaults=sum(loan_status=="Charged Off"), avg
Interest= mean(int_rate),
stdInterest=sd(int_rate), avgLoanAMt=mean(loan_amnt), avgPmnt=mean(total_pymnt), avgRet=mean(ann
Ret), stdRet=sd(annRet),
minRet=min(annRet), maxRet=max(annRet))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # 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      23399     2682       10.7       1.22       12458.  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>
```

```
lcdf %>% select(loan_status, loan_amnt, total_pymnt, int_rate, actualTerm, actualReturn ) %>% vi
ew()


#Summaries

lcdf %>% group_by(loan_status) %>% summarise(nLoans=n(), avgInterest= mean(int_rate), avgLoanAmt
=mean(loan_amnt), avgRet=mean(annRet),
avgActualRet=mean(actualReturn), avgActualTerm=mean(actualTerm), minActualRet=min(actualReturn),
maxActualRet=max(actualReturn))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 9
##   loan_status nLoans avgInterest avgLoanAmt avgRet avgActualRet avgActualTerm
##   <fct>        <int>       <dbl>      <dbl>  <dbl>        <dbl>         <dbl>
## 1 Fully Paid   69195        11.6     12613.   4.98       0.0803          2.10
## 2 Charged Off  11827        13.9     12208. -11.7       -0.117           3
## # ... with 2 more variables: minActualRet <dbl>, maxActualRet <dbl>
```

```
#)Loans - performance by grade

lcdf %>% group_by(grade) %>% summarise(nLoans=n(), defaults=sum(loan_status=="Charged Off"), def
aultRate=defaults/nLoans,
avgInterest= mean(int_rate), avgLoanAmt=mean(loan_amnt), avgRet=mean(annRet), avgActualRet=mean
(actualReturn)*100,
avgActualTerm=mean(actualTerm), minActualRet=min(actualReturn)*100, maxActualRet=max(actualRetur
n)*100)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # 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      23399     2682      0.115        10.7      12458.   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>
```

```
lcdf %>% group_by(grade, loan_status) %>% summarise(nLoans=n(), defaults=sum(loan_status=="Charg
ed Off"), defaultRate=defaults/nLoans,
avgInterest= mean(int_rate), avgLoanAmt=mean(loan_amnt), avgRet=mean(annRet), avgActualRet=mean
(actualReturn),
avgActualTerm=mean(actualTerm), minActualRet=min(actualReturn), maxActualRet=max(actualReturn))
```

```
## `summarise()` regrouping output by 'grade' (override with `.groups` argument)
```

```
## # 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     Fully Paid   19294        0           0        7.24     14187.   3.18
##  2 A     Charged Off   1108     1108           1        7.47     13438. -11.2 
##  3 B     Fully Paid   20717        0           0       10.7      12484.   4.64
##  4 B     Charged Off   2682     2682           1       10.9      12261. -11.0 
##  5 C     Fully Paid   18461        0           0       13.7      11424.   5.83
##  6 C     Charged Off   4116     4116           1       13.7      11652. -11.6 
##  7 D     Fully Paid    8155        0           0       16.5      12038.   7.04
##  8 D     Charged Off   2647     2647           1       16.6      12495. -12.2 
##  9 E     Fully Paid    2146        0           0       19.7      12611.   8.31
## 10 E     Charged Off   1045     1045           1       19.8      12449. -12.8 
## 11 F     Fully Paid     369        0           0       24.1       9657.  10.4 
## 12 F     Charged Off    191      191           1       24.2      11158. -12.5 
## 13 G     Fully Paid      53        0           0       25.8      13412.  11.5 
## 14 G     Charged Off     38       38           1       25.8      11250  -11.0 
## # ... with 4 more variables: avgActualRet <dbl>, avgActualTerm <dbl>,
## #   minActualRet <dbl>, maxActualRet <dbl>
```

```
#Cost based performance - what cost/profit values to use?

lcdf %>% group_by(loan_status) %>% summarise(avgInt=mean(int_rate),avgActInt = mean(actualReturn
*100))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 3
##   loan_status avgInt avgActInt
##   <fct>        <dbl>     <dbl>
## 1 Fully Paid    11.6      8.03
## 2 Charged Off   13.9    -11.7 
```

```
#final table for the purpose of analysis

lcdf %>% group_by(grade,loan_status) %>% summarise(count=n(),avgInt=mean(int_rate),avgActInt = m
ean(actualReturn*100),avgActualTerm=mean(actualTerm),avgRet=mean(annRet))
```

```
## `summarise()` regrouping output by 'grade' (override with `.groups` argument)
```

```
## # A tibble: 14 x 7
## # Groups:   grade [7]
##    grade loan_status count avgInt avgActInt avgActualTerm avgRet
##    <fct> <fct>       <int>  <dbl>     <dbl>         <dbl>  <dbl>
##  1 A     Fully Paid  19294   7.24      4.81          2.18   3.18
##  2 A     Charged Off  1108   7.47    -11.2           3    -11.2
##  3 B     Fully Paid  20717  10.7       7.32          2.11   4.64
##  4 B     Charged Off  2682  10.9     -11.0           3    -11.0
##  5 C     Fully Paid  18461  13.7       9.60          2.04   5.83
##  6 C     Charged Off  4116  13.7     -11.6           3    -11.6
##  7 D     Fully Paid   8155  16.5      11.8           2.02   7.04
##  8 D     Charged Off  2647  16.6     -12.2           3    -12.2
##  9 E     Fully Paid   2146  19.7      14.3           1.97   8.31
## 10 E     Charged Off  1045  19.8     -12.8           3    -12.8
## 11 F     Fully Paid    369  24.1      17.5           1.99  10.4
## 12 F     Charged Off   191  24.2     -12.5           3    -12.5
## 13 G     Fully Paid     53  25.8      18.7           2.07  11.5
## 14 G     Charged Off    38  25.8     -11.0           3    -11.0
```

```
lcdf %>% group_by(grade) %>% summarise(count=n(),avgInt=mean(int_rate),avgActInt = mean(actualRe
turn*100),avgActualTerm=mean(actualTerm),avgRet=mean(annRet),default_rate=sum(loan_status=="Char
ged Off")/n()*100, ThreeYearRet=(mean(annRet)*3+(3-mean(actualTerm))*2))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 7 x 8
##   grade count avgInt avgActInt avgActualTerm avgRet default_rate ThreeYearRet
##   <fct> <int>  <dbl>     <dbl>         <dbl>  <dbl>        <dbl>        <dbl>
## 1 A     20402   7.25      3.94          2.22   2.39         5.43         8.74
## 2 B     23399  10.7       5.22          2.21   2.85        11.5         10.1 
## 3 C     22577  13.7       5.73          2.22   2.65        18.2          9.50
## 4 D     10802  16.5       5.89          2.26   2.32        24.5          8.46
## 5 E      3191  19.8       5.45          2.31   1.40        32.7          5.60
## 6 F       560  24.1       7.29          2.33   2.55        34.1          8.98
## 7 G        91  25.8       6.33          2.46   2.09        41.8          7.36
```

Further to that we have chosen our best models that we created using rpart,ranger and c50 strategy and tried to come up with investment strategy about investing 100usd. we allocated weights to all 4 possible scenarios in confusion matrix and processed our 100 dollars through all the different models for different sets of thresholds. we have presented our findings and different set of confusion matrix in the appendix section.

It turns out all of our models irrespective of the threshold are giving better profit than the risk free rate of 2% per annum.

7B Analysis

We have performed Profit Evaluation based on Ranger model as it was proven the best model in our threshold analysis we got the maximum profit on Ranger model at the threshold of 0.3. Best model for rpart model came at the threshold 0.3 and for c50 the threshold for best performance was 0.8.

Please note that the threshold analysis has been provided in the Appendix Section for all the best models from different libraries such as rpart, C50, ranger, and xgboost.

The model that gave the best profit and return per $100 is XGBoost. However, the model that provided the best accuracy rate is ranger.

We prepared a Decile lift gain chart for ranger model and sorted the data based on the probability of loan to be paid off successfully.

In the given table we analyzed the profit for the given loan. Based on our previous analysis we figured out the weights to be allocated in case our prediction for paid-off is correct: ***PROFITVAL <- 100*****(0.0803*****2.1+0.02*****0.9) (+18) and COSTVAL <- 100-*****0.117*****3 (-35) when prediction is incorrect**.

We have created 3 charts to figure out the correct cut off on prob(fully paid)

1. In chart-1 that plots cumulative profit. Using that we found the prob(Fully-paid) for max(CumProfit) which turned out to be 0.643198. All the loans below the given prob(Fully-paid) are loss making. As an investor it would be a wise decision to invest in loans that has prob(Fully-paid) below 0.643198

Maximum cumulative profit: 266337.7 Associated prob(Fully-paid)= 0.643198

2. In Chart-2 we plotted Average cumulative profit where we divided the cumulative profit by the number of loans. We found out the maximum average cumulative profit and the associated prob(Fully-paid) which came out to be

Maximum Average cumulative profit: 18.10151 Associated prob(Fully-paid)= 0.9734769

3. Analysis using chart-1 is beneficial but not complete because it only tells us the prob(Fully-paid) till the time we are making profit but does not take into account that Certificate of deposit option is available which is giving 6% return in 3 years. So we will have to exclude loans where average profit is less than 6. To perform the analysis we plotted the cumulative Profit reverse. Our aim is to exclude the loans with the lowest prob(Fully-paid). For Profit of 6 the prob(Fully-paid) came out to be : 0.8297552

PLEASE REFER TO THE CHART-3

So to summarize the above mentioned cut-offs we can say that :

1. Loans with prob(Fully-paid) : 0.8297552 are better investments than Certificate of Deposit CD rate that is 6%
2. Loans with prob(Fully-paid)= 0.9734769 or above are the best investment that we can get as they offer average profit of 18.1%
3. Loans with prob(Fully-paid) greater than 0.643198 and less than 0.8297552 offers profits between 0 to 6%. profits provided by these loans are worse that risk free rate of CD but investor is at least not losing money
4. loans with prob(Fully-paid) less than 0.643198 are down right loss making and should be avoided at all cost.

```
#Decile Lift Gain
prPerfRF <- data.frame(scoreTst_FP)
prRetPerfRF <- cbind(prPerfRF, status=lcdfTst$loan_status, grade=lcdfTst$grade, actRet=lcdfTst$a
ctualReturn, actTerm = lcdfTst$actualTerm)
prRetPerfRF <- prRetPerfRF %>% mutate(decile = ntile(-scoreTst_FP, 10))
prRetPerfRF %>% group_by(decile) %>% summarise(count=n(), numDefaults=sum(status=="Charged Off"
), avgActRet=mean(actRet),
minRet=min(actRet), maxRet=max(actRet), avgTer=mean(actTerm), totA=sum(grade=="A"), totB=sum(gra
de=="B" ), totC=sum(grade=="C"),
totD=sum(grade=="D"), totE=sum(grade=="E"), totF=sum(grade=="F") )
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 10 x 13
##    decile count numDefaults avgActRet minRet maxRet avgTer  totA  totB  totC
##     <int> <int>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int> <int>
##  1      1  2431          70    0.0418 -0.303  0.143   2.16  2423     8     0
##  2      2  2431         145    0.0423 -0.313  0.147   2.18  2067   364     0
##  3      3  2431         194    0.0485 -0.291  0.185   2.20  1058  1357    14
##  4      4  2431         242    0.0528 -0.323  0.238   2.19   353  1946   126
##  5      5  2431         285    0.0547 -0.333  0.223   2.21   123  1793   481
##  6      6  2431         367    0.0579 -0.323  0.233   2.20    44  1061  1194
##  7      7  2431         372    0.0633 -0.333  0.259   2.17    10   385  1769
##  8      8  2430         498    0.0563 -0.322  0.281   2.24     8   117  1743
##  9      9  2430         625    0.0494 -0.310  0.271   2.29     1    25  1283
## 10     10  2430         748    0.0513 -0.333  0.367   2.32     3     4   204
## # ... with 3 more variables: totD <int>, totE <int>, totF <int>
```

```
#Investment Returns per 100 Dollars on Decile Lift Gain
prRetPerfRF %>% group_by(decile) %>% summarise(count=n(), numDefaults=sum(status=="Charged Off"
),goodloan=n()-sum(status=="Charged Off"), avgActRet=mean(actRet), riskfreerate=0.02,
avgTer=mean(actTerm),
NetMoneyLCon100USD=  mean(actRet)* mean(actTerm)*100 + (3-mean(actTerm))*0.02*100,
NetMoneyCDon100USD=  0.02*3*100)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 10 x 9
##    decile count numDefaults goodloan avgActRet riskfreerate avgTer
##     <int> <int>       <int>    <int>     <dbl>        <dbl>  <dbl>
##  1      1  2431          70     2361    0.0418         0.02   2.16
##  2      2  2431         145     2286    0.0423         0.02   2.18
##  3      3  2431         194     2237    0.0485         0.02   2.20
##  4      4  2431         242     2189    0.0528         0.02   2.19
##  5      5  2431         285     2146    0.0547         0.02   2.21
##  6      6  2431         367     2064    0.0579         0.02   2.20
##  7      7  2431         372     2059    0.0633         0.02   2.17
##  8      8  2430         498     1932    0.0563         0.02   2.24
##  9      9  2430         625     1805    0.0494         0.02   2.29
## 10     10  2430         748     1682    0.0513         0.02   2.32
## # ... with 2 more variables: NetMoneyLCon100USD <dbl>, NetMoneyCDon100USD <dbl>
```

```
#Threshold Analysis for all best models
#Rpart
rpTHRESH=0.3
predTrnProb=predict(rpDT2, lcdfTrn, type='prob')
predTstProb=predict(rpDT2, lcdfTst, type='prob')

#Rpart Confusion table
predTrnRP = ifelse(predTrnProb[, 'Charged Off'] >= rpTHRESH, 'Charged Off', 'Fully Paid')
table( pred = predTrnRP, true=lcdfTrn$loan_status)
```

```
##              true
## pred          Fully Paid Charged Off
##   Charged Off       1322        2072
##   Fully Paid       47112        6209
```

```
predTstRP = ifelse(predTstProb[, 'Charged Off'] >= rpTHRESH, 'Charged Off', 'Fully Paid')
table( pred = predTstRP, true=lcdfTst$loan_status)
```

```
##              true
## pred          Fully Paid Charged Off
##   Charged Off       1003         386
##   Fully Paid       19758        3160
```

```
#C5.0
CTHRESH=0.8
predTrnProbC5=predict(c5_DT1, lcdfTrn, type='prob')
predTstProbC5=predict(c5_DT1, lcdfTst, type='prob')

#C5.0 Confusion table
predTrnC5 = ifelse(predTrnProbC5[, 'Charged Off'] >= CTHRESH, 'Charged Off', 'Fully Paid')
table( pred = predTrnC5, true=lcdfTrn$loan_status)
```

```
##              true
## pred          Fully Paid Charged Off
##   Charged Off       1026         808
##   Fully Paid       47408        7473
```

```
predTstC5 = ifelse(predTstProbC5[, 'Charged Off'] >= CTHRESH, 'Charged Off', 'Fully Paid')
table( pred = predTstC5, true=lcdfTst$loan_status)
```

```
##              true
## pred          Fully Paid Charged Off
##   Charged Off        630         171
##   Fully Paid       20131        3375
```

```
#ranger
rgTHRESH=0.3
predTrnProbRG=predict(rgModelProb, lcdfTrn)
predTstProbRG=predict(rgModelProb, lcdfTst)

#ranger confusion table
predTrnRG = ifelse(predTrnProbRG$predictions[, 'Charged Off'] >= rgTHRESH, 'Charged Off', 'Fully
Paid')
table( pred = predTrnRG, true=lcdfTrn$loan_status)
```

```
##              true
## pred           Fully Paid Charged Off
##   Charged Off       1117        1660
##   Fully Paid       47317        6621
```

```
lcdf %>% group_by(loan_status) %>% summarise(intRate = mean(int_rate), actTerm = mean(actualTer
m), actRet = mean(actualReturn))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 4
##   loan_status intRate actTerm  actRet
##   <fct>         <dbl>   <dbl>   <dbl>
## 1 Fully Paid     11.6    2.10  0.0803
## 2 Charged Off    13.9    3    -0.117
```

```
predTstRG = ifelse(predTstProbRG$predictions[, 'Charged Off'] >= rgTHRESH, 'Charged Off', 'Fully
Paid')
table( pred = predTstRG, true=lcdfTst$loan_status)
```

```
##              true
## pred           Fully Paid Charged Off
##   Charged Off        734         383
##   Fully Paid       20027        3163
```

```
lcdf %>% group_by(loan_status) %>% summarise(intRate = mean(int_rate), actTerm = mean(actualTer
m), actRet = mean(actualReturn))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 4
##   loan_status intRate actTerm  actRet
##   <fct>         <dbl>   <dbl>   <dbl>
## 1 Fully Paid     11.6    2.10  0.0803
## 2 Charged Off    13.9    3    -0.117
```

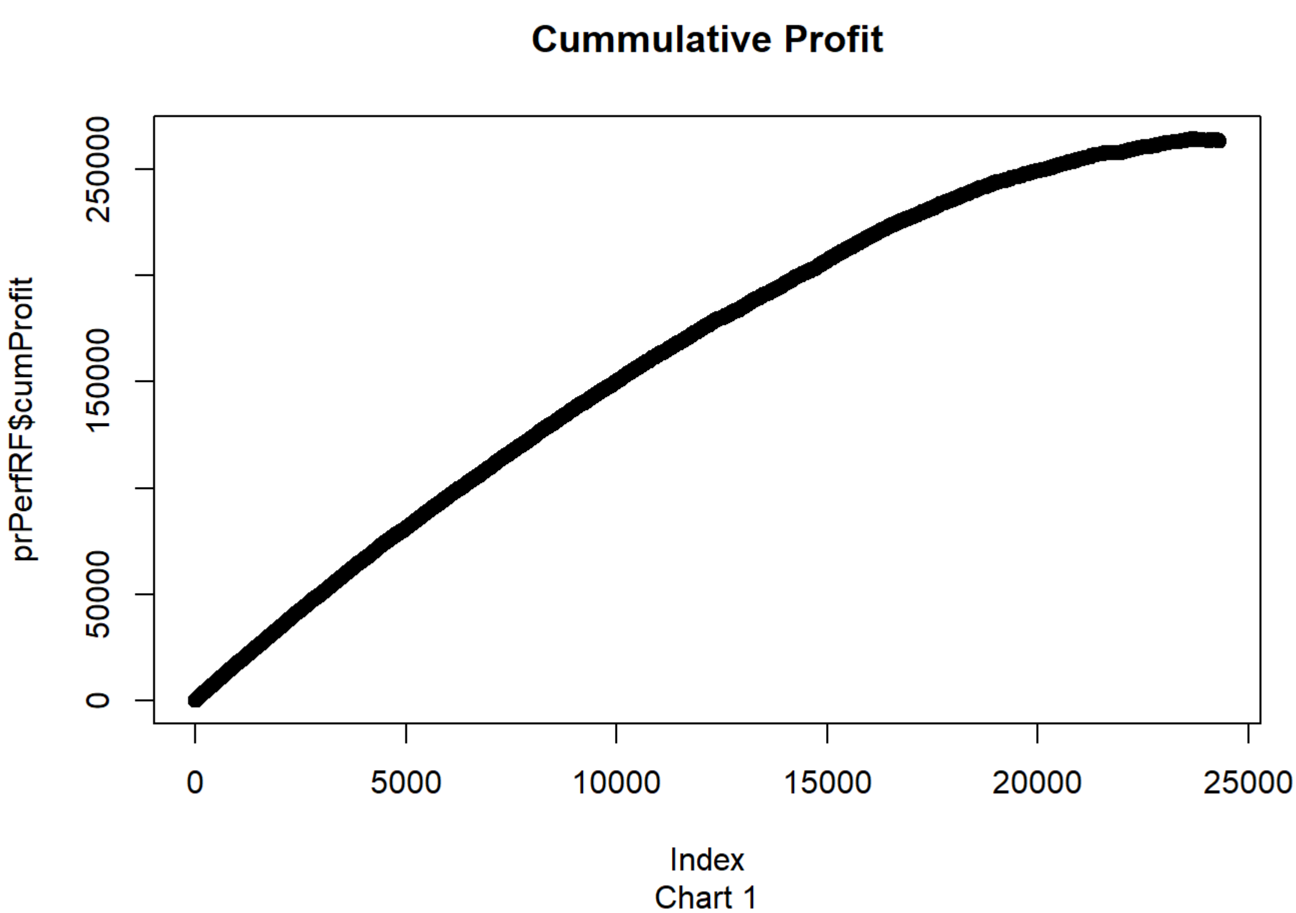
```
#Cutoff Analysis

#Assigning Profit and Loss Values
PROFITVAL <- 100*(0.0803*2.1+0.02*0.9)  #profit (on $100) from accurately identifying Fully_paid loans: 16.7827
COSTVAL <- 100*-0.117*3 # loss (on $100) from incorrectly predicting a Charged_Off loan as Full_paid: -35.1

prPerfRF <- cbind(prPerfRF, status=lcdfTst$loan_status)
prPerfRF <- prPerfRF[order(-scoreTst_FP) ,] #sort in desc order of prob(fully_paid)

#Chart 1 Cummulative Profit
prPerfRF$profit <- ifelse(prPerfRF$status == 'Fully Paid', PROFITVAL, COSTVAL)
prPerfRF$cumProfit <- cumsum(prPerfRF$profit)
max(prPerfRF$cumProfit)
```

```
## [1] 264271.8
```

```
prPerfRF$cumProfit[which.max(prPerfRF$cumProfit)]
```

```
## [1] 264271.8
```

```
plot(prPerfRF$cumProfit, main = "Cummulative Profit", sub = "Chart 1")
```



```
scoreTst_FP_Limit1=prPerfRF$scoreTst_FP[prPerfRF$cumProfit==max(prPerfRF$cumProfit)]
scoreTst_FP_Limit1
```

```
## [1] 0.6690258
```

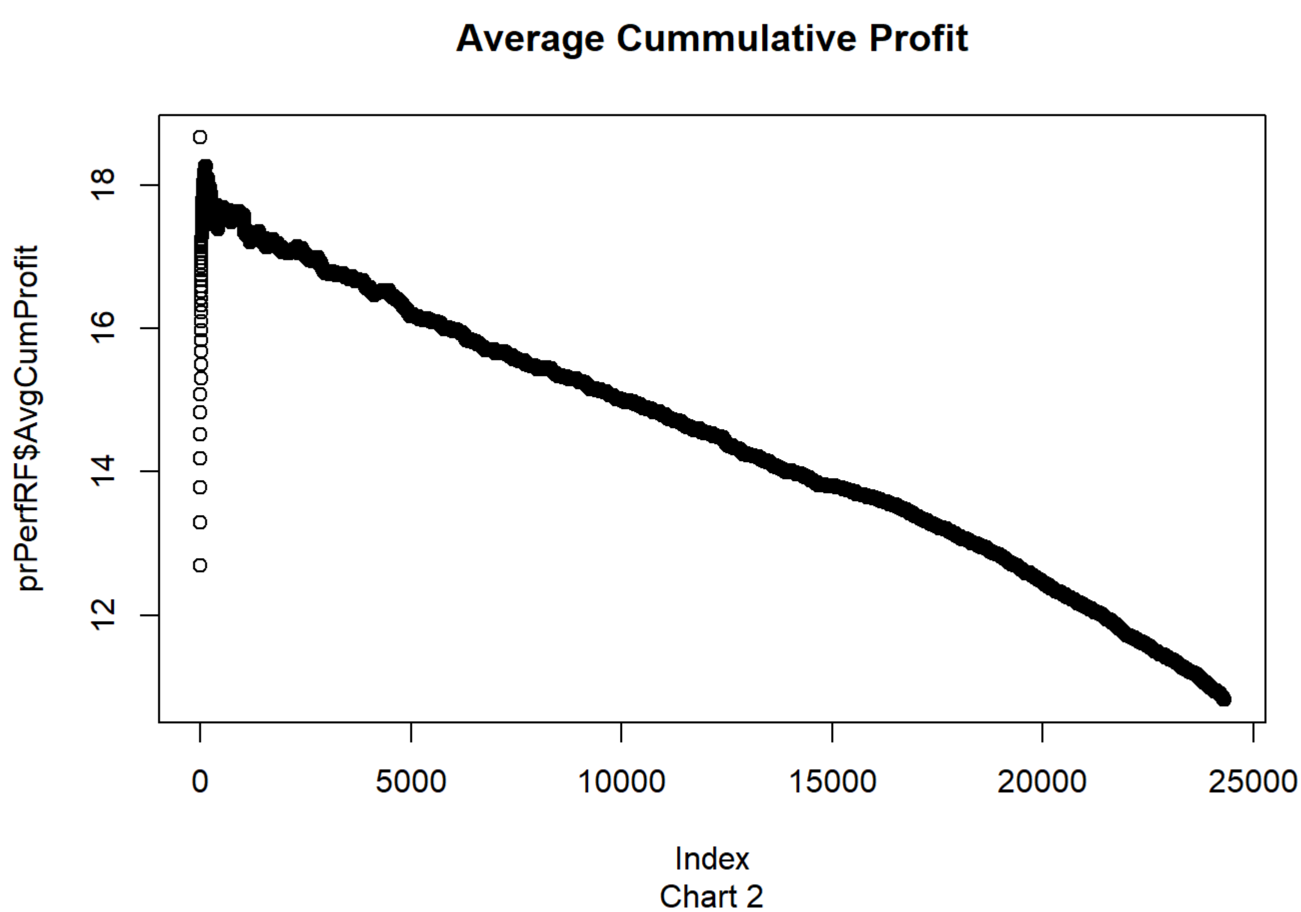
```
#Chart 2 Average Cummulative Profit
prPerfRF$cumilaive_count <-seq.int(nrow(prPerfRF))
prPerfRF$AvgCumProfit=prPerfRF$cumProfit/prPerfRF$cumilaive_count
max(prPerfRF$AvgCumProfit)
```

```
## [1] 18.663
```

```
scoreTst_FP_Limit2=prPerfRF$scoreTst_FP[prPerfRF$AvgCumProfit==max(prPerfRF$AvgCumProfit)]
scoreTst_FP_Limit2
```

```
## [1] 0.9856936 0.9856882 0.9855455 0.9849474 0.9846718 0.9844372 0.9844295
## [8] 0.9843501
```

```
plot(prPerfRF$AvgCumProfit, main = "Average Cummulative Profit", sub = "Chart 2")
```



```
#Chart 3: Inversed Average Cummulative Profit
prPerfRF$cumilaive_count_reverse <- nrow(prPerfRF)- seq.int(nrow(prPerfRF))

sum(prPerfRF$profit)
```

```
## [1] 262997.9
```

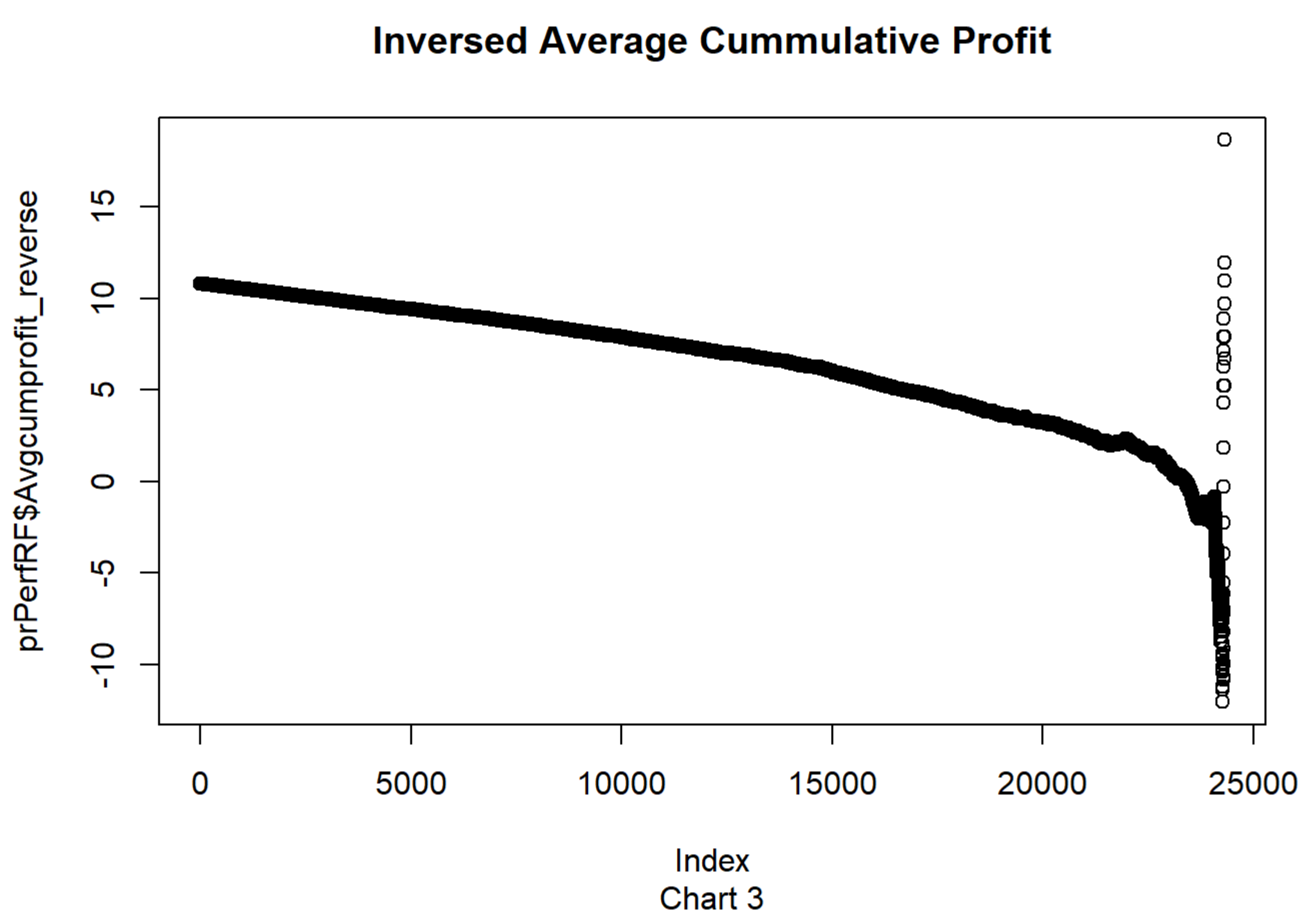
```
prPerfRF$cumprofit_reverse= sum(prPerfRF$profit)-prPerfRF$cumProfit
prPerfRF$Avgcumprofit_reverse= prPerfRF$cumprofit_reverse/prPerfRF$cumilaive_count_reverse
scoreTst_FP_Limit3<-prPerfRF$scoreTst_FP[prPerfRF$Avgcumprofit_reverse<6]
length(scoreTst_FP_Limit3)
```

```
## [1] 9278
```

```
scoreTst_FP_Limit3[1]
```

```
## [1] 0.8328987
```

```
plot(prPerfRF$Avgcumprofit_reverse, main = "Inversed Average Cummulative Profit", sub = "Chart
 3")
```



Q8 Analysis

We tried different parameters for Xgboost like max. depth, eta, nrounds, lambda, subsample, colsample_bytree, stopping criteria. 1. max.depth - Used to control overfitting as higher depth will allow model to learn relations very specific to a particular sample. Default value is 6 and the typical values are 3-10. Hence, we tried with different max.depth with different other parameters and found that max.depth = 4is giving the best result.

2. eta - It makes the model more robust by shrinking the weights on each step. Typical values are 0.01-0.2 and the default value is 0.3. After experimenting with all the values we found that default eta=0.3 is giving the best results when using with other parameters.

3. lambda - This is used to handle the regularization part of the Xgboost. Default is 1. We tried with different values like 0.5, 2 but found that the default value 1 with other parameters give the best result.

4. subsample, colsample_bytree - Used to prevent overfitting, Typical values between 0.5 to 1 and default is 1. We tried different values but found that our model was not overfitting and hence we removed these parameters.

5. Stopping Criteria - We used the stopping criteria 10 and also 0. But results were more faster using stopping criteria 10. Hence, we used it as 10.

6. nrounds - 1000, gave us better result when used with stopping criteria 10.

However, we couldn't find a very good accuracy after playing with all the parameters mentioned above. The maximum accuracy we got in training data is 72% and test data is 69%. Hence, then we experimented with the Cross validation method, which usually gives better accuracy. After experimenting with many parameters and their values, used max_depth as 4 which is the main parameter to be tuned with the Cross validation, eta=0.01, nrounds=1000 and stopping criteria 10 and found that the model gives a little better result than before i.e. 73% for the training data and 69% for the test data.

Please refer Table 4 in Appendix for the tabulation of the results with different values. The best model is stored in xgb_lsbest. The evaluation of xgb_lsbest is done with AUC, ROC, confusion matrices, and lift curve.

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
```

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

```
#Needs all data to be numeric -- so we convert categorical (i.e. factor) variables - # use the d
ummyVars function in the 'caret' package to convert factor variables to dummy-variables

dumVar<-dummyVars(~.,data=lcdf %>% select(-loan_status))
dxlcdf<- predict(dumVar,lcdf)

# for loan_status, check levels and convert to dummy vars and keep the class label of interest
levels(lcdf$loan_status)
```

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

```
dylcdf <- class2ind(lcdf$loan_status, drop2nd = FALSE)
# and then decide which one to keep
colcdf <- dylcdf [ , 1]# or,fplcdf <- dycldf [ , 2]

#Training, test subsets
dxlcdfTrn <- dxlcdf[trnIndex,]
colcdfTrn <- colcdf[trnIndex]
dxlcdfTst <- dxlcdf[-trnIndex,]
colcdfTst <- colcdf[-trnIndex]
dxTrn <- xgb.DMatrix(subset(dxlcdfTrn, select=-c(annRet, actualTerm, actualReturn, total_pymn
t)), label=colcdfTrn)
dxTst <- xgb.DMatrix( subset( dxlcdfTst,select=-c(annRet, actualTerm, actualReturn, total_pymn
t)), label=colcdfTst)

xgbWatchlist <- list(train = dxTrn, eval = dxTst)
#we can watch the progress of learning thru performance on these datasets
#list of parameters for the xgboost model development functions
xgbMyParam <- list (
max_depth = 4, eta = 0.01,
objective = "binary:logistic",
eval_metric="error", eval_metric = "auc")
#can specify which evaluation metrics we want to watch
xgb_lsM1 <- xgb.train( xgbMyParam, dxTrn, nrounds = 10, early_stopping_rounds = 10,
xgbWatchlist) #Stop if performance does not improve after 10 rounds
```

```
## [1]  train-error:0.146011    train-auc:0.679653  eval-error:0.145884 eval-auc:0.672680
## 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:0.146011    train-auc:0.679653  eval-error:0.145884 eval-auc:0.672680
## [3]  train-error:0.146011    train-auc:0.679955  eval-error:0.145884 eval-auc:0.673036
## [4]  train-error:0.146011    train-auc:0.683839  eval-error:0.145884 eval-auc:0.677284
## [5]  train-error:0.146011    train-auc:0.684750  eval-error:0.145884 eval-auc:0.678885
## [6]  train-error:0.146011    train-auc:0.685006  eval-error:0.145884 eval-auc:0.678816
## [7]  train-error:0.146011    train-auc:0.685298  eval-error:0.145884 eval-auc:0.679163
## [8]  train-error:0.146011    train-auc:0.686155  eval-error:0.145884 eval-auc:0.680479
## [9]  train-error:0.146011    train-auc:0.687143  eval-error:0.145884 eval-auc:0.681857
## [10] train-error:0.146011    train-auc:0.687592  eval-error:0.145884 eval-auc:0.682227
```

```
xgb_lsM1$best_iteration
```

```
## [1] 10
```

```
xpredTrg<-predict(xgb_lsM1, dxTrn)
head(xpredTrg)
```

```
## [1] 0.5230832 0.5210394 0.5284817 0.5314360 0.5123543 0.5420388
```

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

```
##     act
## pred     0     1
##    1  8281 48434
```

```
#use cross-validation on training dataset to determine best model
xgbParam <- list (
max_depth = 4, eta = 0.01,
objective = "binary:logistic",
eval_metric="error", eval_metric = "auc")
xgb_lscv <- xgb.cv( xgbParam, dxTrn, nrounds = 10, nfold=10, early_stopping_rounds = 10 )
```

```
## [1]  train-error:0.146009+0.000505   train-auc:0.679858+0.000735 test-error:0.146064+0.004585
test-auc:0.670316+0.009233
## Multiple eval metrics are present. Will use test_auc for early stopping.
## Will train until test_auc hasn't improved in 10 rounds.
##
## [2]  train-error:0.146005+0.000506   train-auc:0.682010+0.002324 test-error:0.146046+0.004555
test-auc:0.671174+0.009934
## [3]  train-error:0.146005+0.000506   train-auc:0.682775+0.001876 test-error:0.146029+0.004542
test-auc:0.672051+0.009392
## [4]  train-error:0.146007+0.000502   train-auc:0.683945+0.001691 test-error:0.146011+0.004512
test-auc:0.673473+0.009453
## [5]  train-error:0.146007+0.000502   train-auc:0.684968+0.001733 test-error:0.146011+0.004512
test-auc:0.673888+0.009205
## [6]  train-error:0.146007+0.000502   train-auc:0.685673+0.002236 test-error:0.146011+0.004512
test-auc:0.674479+0.009022
## [7]  train-error:0.146007+0.000502   train-auc:0.686384+0.002011 test-error:0.146011+0.004512
test-auc:0.674784+0.009032
## [8]  train-error:0.146005+0.000504   train-auc:0.687077+0.001682 test-error:0.146011+0.004512
test-auc:0.675242+0.009189
## [9]  train-error:0.146005+0.000504   train-auc:0.687389+0.001617 test-error:0.146011+0.004512
test-auc:0.675861+0.008940
## [10] train-error:0.146005+0.000504   train-auc:0.687916+0.001435 test-error:0.146011+0.004512
test-auc:0.676206+0.009134
```

```
#best iteration
xgb_lscv$best_iteration
```

```
## [1] 10
```

```
# or for the best iteration based on performance measure (among those specified in xgbParam)
best_cvIter <- which.max(xgb_lscv$evaluation_log$test_auc_mean)

#best model
xgb_lsbest <- xgb.train(xgbParam, dxTrn, nrounds = xgb_lscv$best_iteration)
#variable importance
#xgb.importance(model = xgb_lsbest) %>% view()

xgb_lscv$evaluation_log
```

```
##     iter train_error_mean train_error_std train_auc_mean train_auc_std
##  1:    1        0.1460088    0.0005047932      0.6798580  0.0007349634
##  2:    2        0.1460049    0.0005056219      0.6820098  0.0023244719
##  3:    3        0.1460049    0.0005056219      0.6827753  0.0018756040
##  4:    4        0.1460069    0.0005022608      0.6839449  0.0016911965
##  5:    5        0.1460069    0.0005022608      0.6849683  0.0017334772
##  6:    6        0.1460069    0.0005022608      0.6856731  0.0022360299
##  7:    7        0.1460069    0.0005022608      0.6863841  0.0020114511
##  8:    8        0.1460049    0.0005037474      0.6870771  0.0016822516
##  9:    9        0.1460049    0.0005037474      0.6873892  0.0016169444
## 10:   10        0.1460049    0.0005037474      0.6879164  0.0014346604
##     test_error_mean test_error_std test_auc_mean test_auc_std
##  1:       0.1460637    0.004585052     0.6703162  0.009233432
##  2:       0.1460461    0.004555020     0.6711744  0.009933865
##  3:       0.1460285    0.004541939     0.6720514  0.009391940
##  4:       0.1460108    0.004512001     0.6734726  0.009453360
##  5:       0.1460108    0.004512001     0.6738880  0.009204968
##  6:       0.1460108    0.004512001     0.6744789  0.009022471
##  7:       0.1460108    0.004512001     0.6747836  0.009031831
##  8:       0.1460108    0.004512001     0.6752423  0.009189340
##  9:       0.1460108    0.004512001     0.6758610  0.008940207
## 10:       0.1460108    0.004512001     0.6762060  0.009134321
```

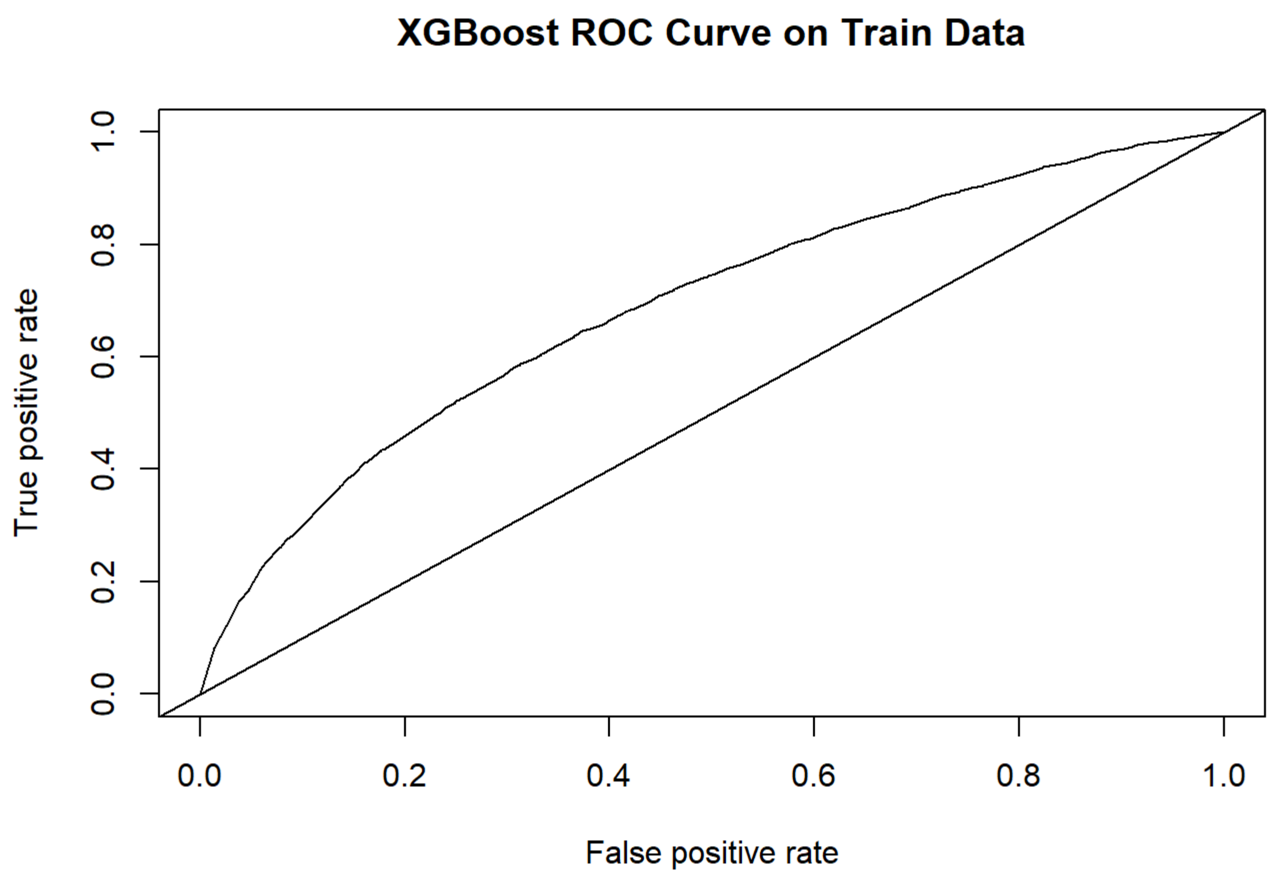
```
#confusion matrix for Train Data
xpredBestTrn<-predict(xgb_lsbest, dxTrn)
table(pred=as.numeric(xpredBestTrn>0.8), act=colcdfTrn)
```

```
##     act
## pred     0     1
##    0  8281 48434
```

```
#confusion matrix for Test Data
xpredBestTst<-predict(xgb_lsbest, dxTst)
table(pred=as.numeric(xpredBestTst>0.8), act=colcdfTst)
```

```
##     act
## pred     0     1
##    0  3546 20761
```

```
#ROC, AUC performance
#On Train Data
pred_xgb_lsbest_trn=prediction(xpredBestTrn, lcdfTrn$loan_status,label.ordering = c("Charged Of
f", "Fully Paid"))

aucPerf_xgb_lsbest_trn=performance(pred_xgb_lsbest_trn, "tpr", "fpr")
plot(aucPerf_xgb_lsbest_trn, main = "XGBoost ROC Curve on Train Data")
abline(a=0, b= 1)
```

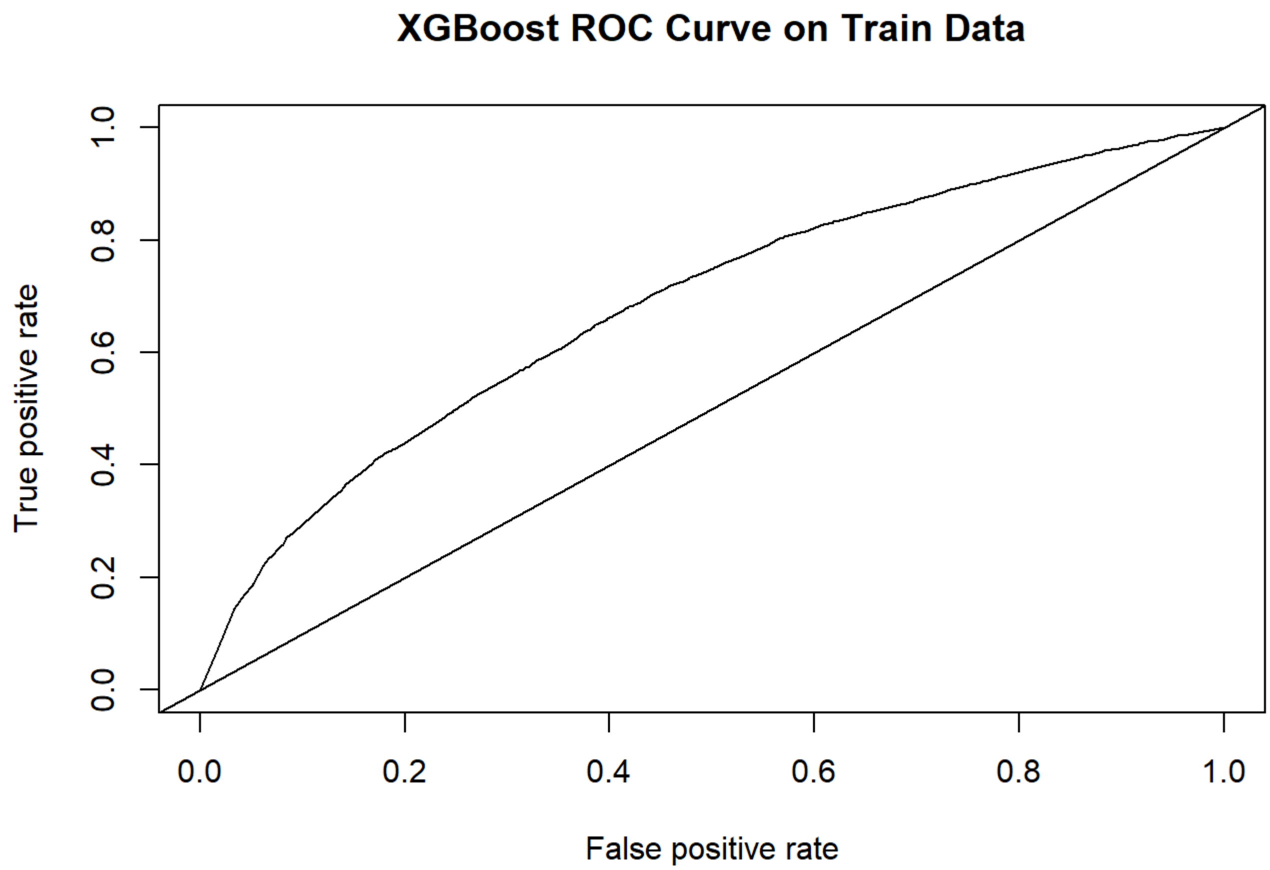



```
xg_roc_Trn <- roc(lcdfTrn$loan_status, xpredBestTrn)
```

```
## Setting levels: control = Fully Paid, case = Charged Off
```

```
## Setting direction: controls > cases
```

```
#On Test Data
pred_xgb_lsbest_tst=prediction(xpredBestTst, lcdfTst$loan_status,label.ordering = c("Charged Of
f", "Fully Paid"))
aucPerf_xgb_lsM1=performance(pred_xgb_lsbest_tst, "tpr", "fpr")
plot(aucPerf_xgb_lsM1, main = "XGBoost ROC Curve on Train Data")
abline(a=0, b= 1)
```

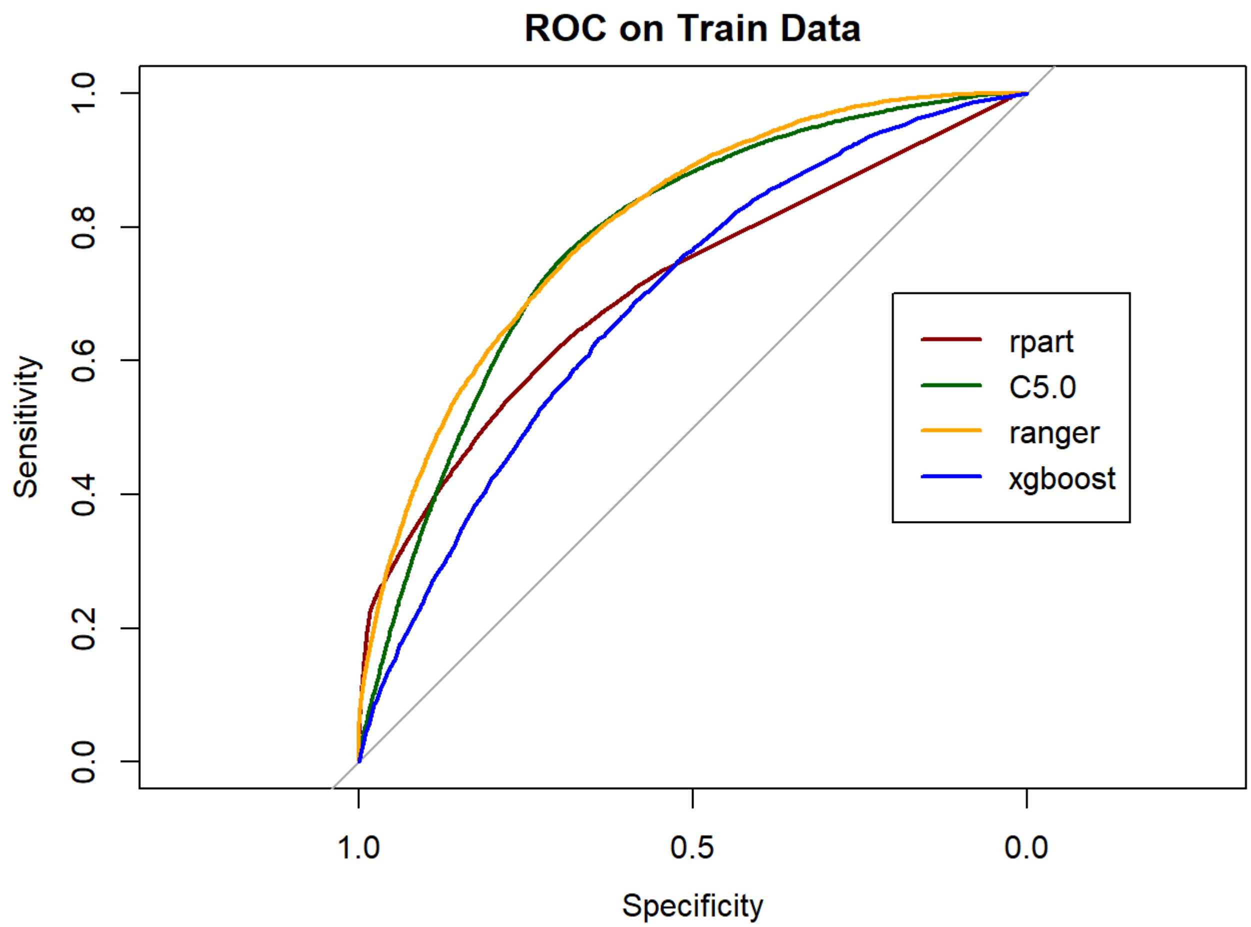



```
xg_roc_Tst <- roc(lcdfTst$loan_status, xpredBestTst)
```

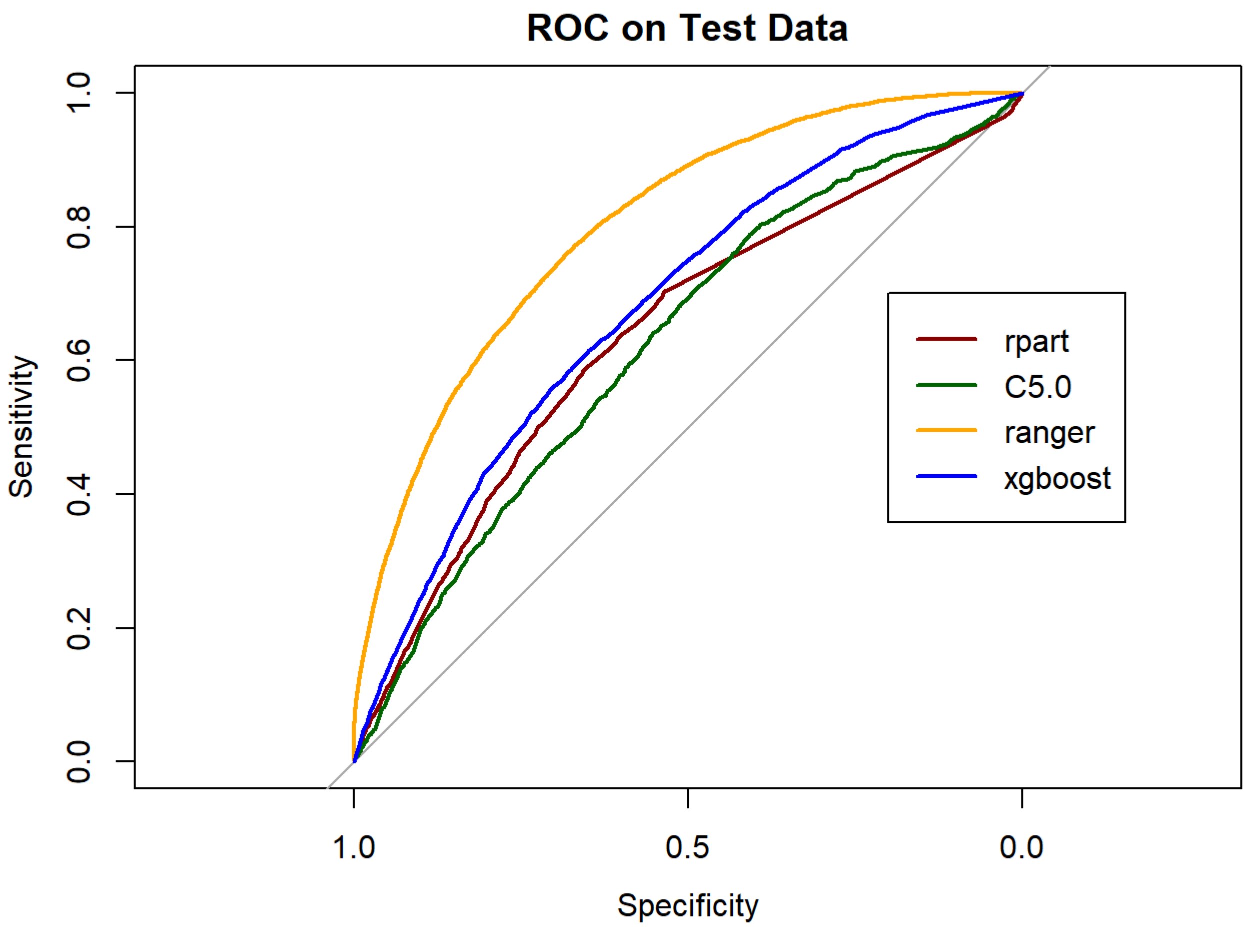
```
## Setting levels: control = Fully Paid, case = Charged Off
## Setting direction: controls > cases
```

The combined ROC Plots allows us to compare the best models. Consistent to our findings above, the ranger model provides the highest accuracy.

```
#For Training Dataset: combined ROC of rpart, C5.0, ranger, XGBoost
plot(rpDT2_roc_Trn,col="darkred", main = "ROC on Train Data")
plot(c5_roc_Trn,col="darkgreen",add=TRUE)
plot(rg_roc_Trn,col="orange",add=TRUE)
plot(xg_roc_Trn,col="blue",add=TRUE)
legend(0.2,0.7, c('rpart','C5.0','ranger','xgboost'),lty=c(1,1), lwd=c(2,2),col=c('darkred','dar
kgreen','orange','blue'))
```




```
#For Testing Dataset: combined ROC of rpart, C5.0, ranger, XGBoost
plot(rpDT2_roc_Tst,col="darkred", main = "ROC on Test Data")
plot(c5_roc_Tst,col="darkgreen",add=TRUE)
plot(rg_roc_Trn,col="orange",add=TRUE)
plot(xg_roc_Tst,col="blue",add=TRUE)
legend(0.2,0.7, c('rpart','C5.0','ranger','xgboost'),lty=c(1,1), lwd=c(2,2),col=c('darkred','dar
kgreen','orange','blue'))
```



Please refer to **Table 8.1** for the complete tabulation of our best models. The best model for accuracy is ranger Random Forest at 0.3 cutoff (83.77% on test data), but the model that yields better profit is XGboost at 0.8 cutoff (USD 274,761.4).

This happened because although our ranger model is better in overall accuracy, our XGBoost model is able to differentiate “Charged Off” class better. Since “Charged Off” ultimately incur more significant losses (35.1 dollars) than “Fully Paid” incurs profits (18.6 dollars). The impact of misclassifying a “Charged Off” is dramatically bigger than correctly classifying every “Fully Paid”.