Assignment-2

Joseline Luvena Tanujaya

Sweta Bansal

Vibhanshu

IDS 572 - Assignment 2

Joseline Tanujaya, Sweta Bansal, Vibhanshu

```
#split the data into trn, tst subsets
set.seed(123)
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
```

Predicting Loan Status with GLM

In building the glm model, we adjusted several parameters: type.measure, alpha, and weights (balanced data).

Below is the complete tabulation of the parameters and their resulting AUC.

```
##                X   X.1      X.2 Lambda.min        X.3 Lambda.1se        X.4
## 1  type.measure alpha balanced AUC on Trn AUC on Tst AUC on Trn AUC on Tst
## 2      deviance     1        N  0.6954079  0.6954079   0.687354   0.687354
## 3           auc     1        N  0.6954079  0.6910984
## 4         class     1        N  0.6854198  0.6809635
## 5      deviance     0        N  0.6946417  0.6908809
## 6           auc     0        N  0.6946417  0.6908809
## 7         class     0        N   0.692251  0.6899955
## 8      deviance     1        Y  0.6954781  0.6910646
## 9      deviance   0.3        Y  0.6954573  0.6910745
## 10     deviance   0.5        Y  0.6954791  0.6910773
## 11     deviance   0.7        Y  0.6954865  0.6910773
```

We used AUC measures to define the best glm model. The models that we got are not too different with each other. However with very slight differences, we can still determine the best model. The best model is the one with balanced class of loan status (using weights), with ***type.measure = "deviance"***, ***alpha = 1*** with AUC of 0.6953326.

We ran the code several times, and the model with balanced weights always give slightly better results. However, depending on the Train and Test data we build and test the models on, alpha = 1 or alpha = 0.7 can be the best options. When comparing the glm model with other models (ranger, xgboost), we will use alpha = 1.

For variable selection method, we will scale the coefficients using the Lasso and Ridge regularization parameters. This is done by setting different values to alpha. When alpha = 1, we are using purely Lasso (L1). When alpha = 0, we are using purely Ridge (L2). Any value between 0-1 is attributed to the percentage using L1. For example, if alpha = 0.2, then L1 = 0.2 and L2 = (1 - 0.2) = 0.8. We have tried several values between 0-1, but the best result still comes from alpha = 1 (Lasso only). Lasso puts pressure on the coefficients to approach 0, therefore we are discarding some variables with low importance.

We experimented using lambda 1se using the best models (with balanced weights). Lambda min has proven to be the better lambda option, since it has higher accuracy. There is no overfitting problems with our models, therefore we do not need to use a bigger lambda than lambda min.

The experimentation with type.measure will be documented at section 1B.

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.0.4
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-1
```

```
library(tidyverse)
library(lubridate)
library(tidyr)
library(caret)
```

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

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

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

```
library(prediction)
```

```
## Warning: package 'prediction' was built under R version 4.0.4
```

```
library(ROCR)
```

```
##
## Attaching package: 'ROCR'
```

```
## The following object is masked from 'package:prediction':
##
##     prediction
```

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

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

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

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

```
library(broom)

levels(lcdfTrn$loan_status)
```

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

```
yTrn<-factor(if_else(lcdfTrn$loan_status=="Fully Paid", '1', '0') )
yTst<-factor(if_else(lcdfTst$loan_status=="Fully Paid", '1', '0') )

xDTrn<-lcdfTrn %>% select(-loan_status, -actualTerm, -annRet, -actualReturn, -total_pymnt)
xDTst<-lcdfTst %>% select(-loan_status, -actualTerm, -annRet, -actualReturn, -total_pymnt)

#Use Lasso regularization (alpha = 1, default)
glmls_cv<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial")

glmls_cv$lambda.min
```

```
## [1] 0.0001133834
```

```
glmls_cv$lambda.1se
```

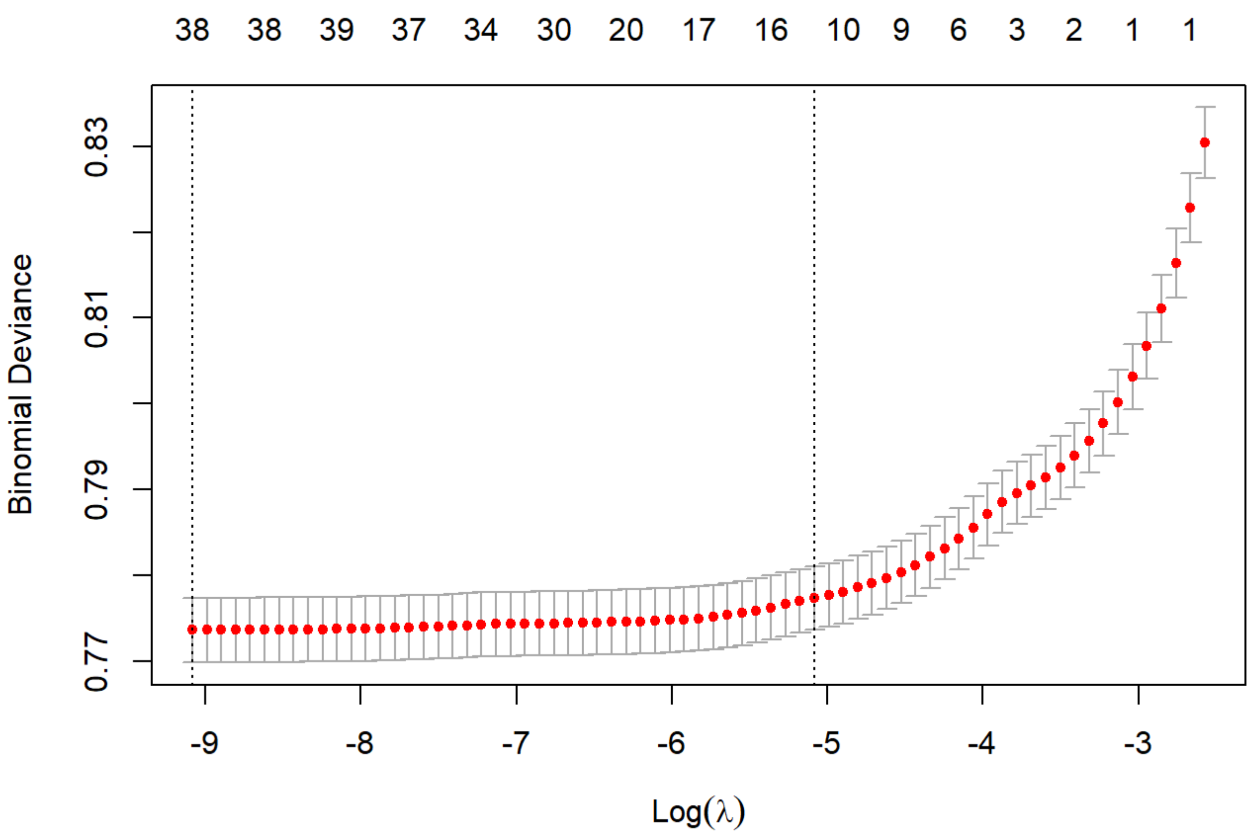
```
## [1] 0.006193316
```

```
#get the variables with non-zero coefficients from the regularized model
nzCoef<-tidy(coef(glmls_cv, s= glmls_cv$lambda.1se))
```

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")
```

```
## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

```
nzCoefVars <- nzCoef[-1,1]

plot(glmls_cv)
```




```
coef(glmls_cv, s = glmls_cv$lambda.min)
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                         1
## (Intercept)                  3.462405e+00
## loan_amnt                    .
## int_rate                    -3.618328e-02
## installment                 -2.326383e-04
## grade                       -7.475791e-02
## sub_grade                   -3.770364e-02
## emp_length                   1.195171e-02
## home_ownership              -6.004841e-02
## annual_inc                  -3.735868e-07
## purpose                      3.193568e-03
## dti                         -1.840644e-02
## initial_list_status         -2.126423e-02
## total_rev_hi_lim             4.428945e-06
## acc_open_past_24mths        -4.566196e-02
## avg_cur_bal                  1.358973e-06
## bc_open_to_buy              -4.258079e-06
## bc_util                     -2.839648e-03
## chargeoff_within_12_mths    -3.864755e-02
## delinq_amnt                  2.765696e-05
## mo_sin_old_il_acct          -1.073440e-04
## mo_sin_old_rev_tl_op         2.288770e-04
## mo_sin_rcnt_rev_tl_op        .
## mo_sin_rcnt_tl               2.409513e-03
## mort_acc                     2.626384e-02
## mths_since_recent_bc        -1.606128e-04
## mths_since_recent_inq        3.707635e-03
## num_bc_sats                  .
## num_bc_tl                   -3.728922e-02
## num_il_tl                    5.091431e-03
## num_op_rev_tl               -2.973017e-02
## num_rev_accts                2.132643e-02
## num_sats                     4.396394e-03
## num_tl_30dpd                -2.203274e-01
## pct_tl_nvr_dlq               1.377219e-03
## tax_liens                   -2.946152e-02
## tot_hi_cred_lim              5.718190e-07
## total_bal_ex_mort           -4.584606e-06
## total_bc_limit               7.652567e-06
## total_il_high_credit_limit  4.809257e-06
## propSatisBankcardAccts      -1.311651e-01
## prop_OpAccts_to_TotAccts     3.519362e-01
## propLoanAmt_to_AnnInc       -4.766838e-01
```

```
coef(glmls_cv, s = glmls_cv$lambda.1se)
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## (Intercept)                3.360645e+00
## loan_amnt                  .
## int_rate                  -2.977258e-02
## installment                .
## grade                     -4.364347e-02
## sub_grade                 -5.318270e-02
## emp_length                 .
## home_ownership            -3.442481e-02
## annual_inc                 .
## purpose                    .
## dti                       -1.009910e-02
## initial_list_status        .
## total_rev_hi_lim           .
## acc_open_past_24mths      -3.572049e-02
## avg_cur_bal                .
## bc_open_to_buy             3.792271e-07
## bc_util                   -9.069142e-04
## chargeoff_within_12_mths   .
## delinq_amnt                .
## mo_sin_old_il_acct         .
## mo_sin_old_rev_tl_op       .
## mo_sin_rcnt_rev_tl_op      .
## mo_sin_rcnt_tl             .
## mort_acc                   1.062411e-02
## mths_since_recent_bc       .
## mths_since_recent_inq      1.901960e-04
## num_bc_sats                .
## num_bc_tl                  .
## num_il_tl                  .
## num_op_rev_tl              .
## num_rev_accts              .
## num_sats                   .
## num_tl_30dpd               .
## pct_tl_nvr_dlq             .
## tax_liens                  .
## tot_hi_cred_lim            6.093373e-07
## total_bal_ex_mort          .
## total_bc_limit             .
## total_il_high_credit_limit .
## propSatisBankcardAccts     .
## prop_OpAccts_to_TotAccts   .
## propLoanAmt_to_AnnInc     -4.149634e-01
```

```
#find the index of the best lambda
which(glmls_cv$lambda == glmls_cv$lambda.1se)
```

```
## [1] 28
```

```
glmls_cv$glmnet.fit$dev.ratio[which(glmls_cv$lambda == glmls_cv$lambda.1se) ]
```

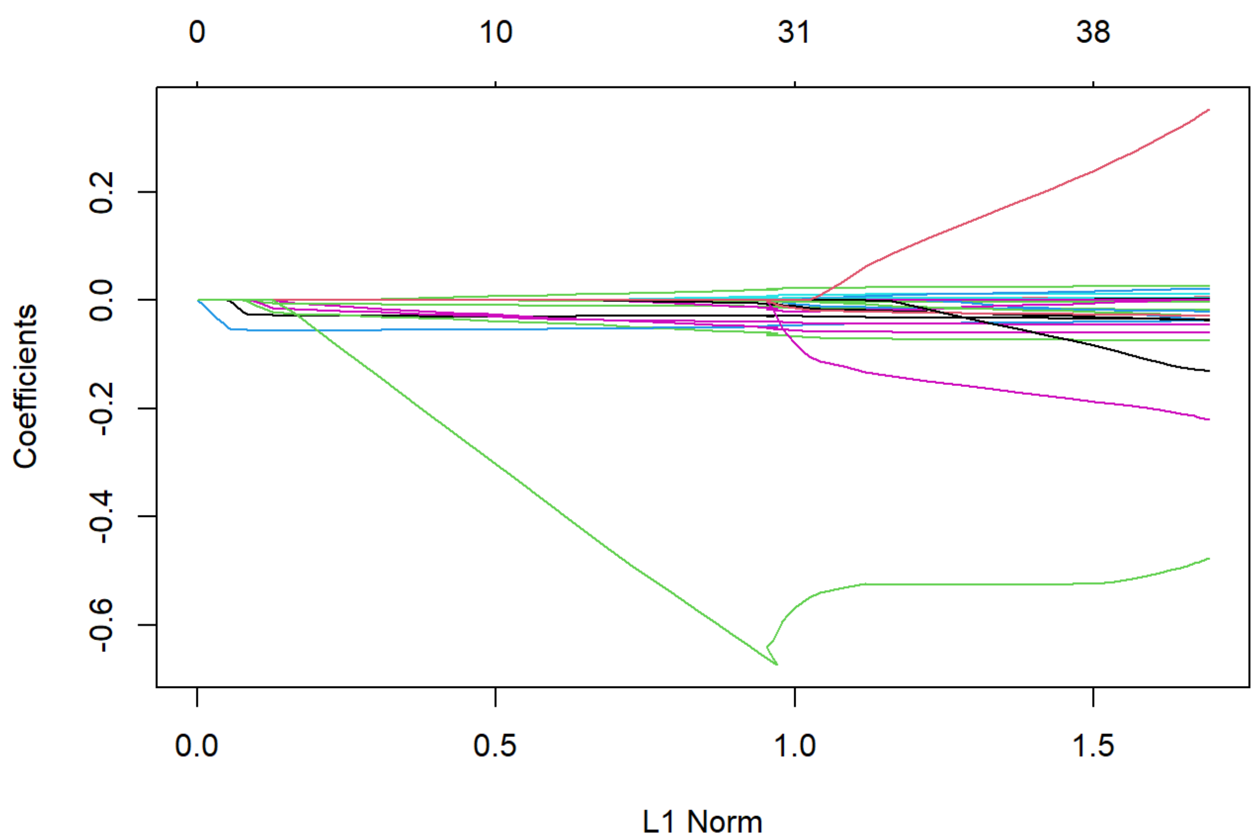
```
## [1] 0.06481165
```

```
glmls_cv$glmnet.fit
```

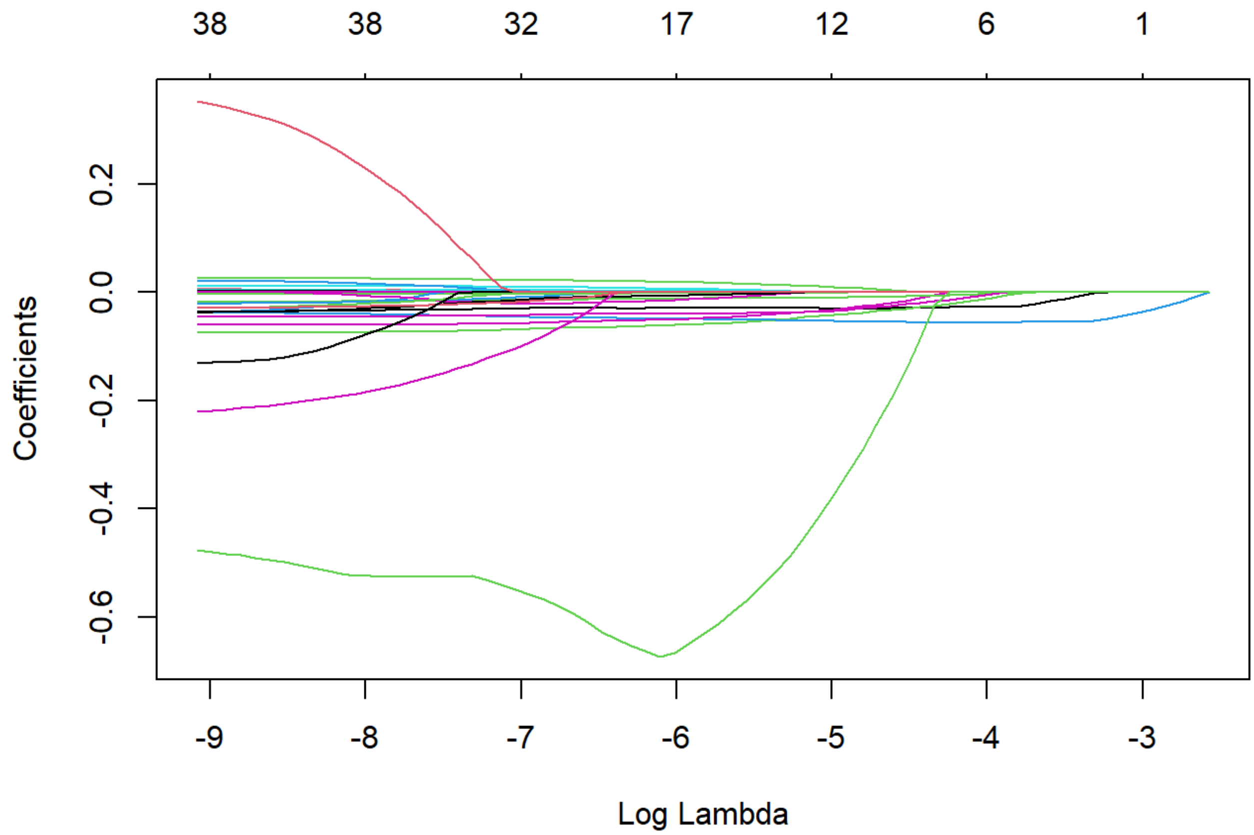
```
##
## Call:  glmnet(x = data.matrix(xDTrn), y = yTrn, family = "binomial")
##
##    Df %Dev   Lambda
## 1   0 0.00 0.076350
## 2   1 0.95 0.069570
## 3   1 1.72 0.063390
## 4   1 2.36 0.057760
## 5   1 2.88 0.052630
## 6   1 3.32 0.047950
## 7   1 3.68 0.043690
## 8   1 3.97 0.039810
## 9   2 4.22 0.036270
## 10  2 4.42 0.033050
## 11  2 4.60 0.030120
## 12  2 4.74 0.027440
## 13  3 4.86 0.025000
## 14  3 4.95 0.022780
## 15  5 5.08 0.020760
## 16  6 5.27 0.018910
## 17  6 5.45 0.017230
## 18  6 5.61 0.015700
## 19  6 5.75 0.014310
## 20  8 5.87 0.013040
## 21  9 5.99 0.011880
## 22  9 6.09 0.010820
## 23  9 6.18 0.009862
## 24 10 6.25 0.008985
## 25 10 6.32 0.008187
## 26 10 6.38 0.007460
## 27 11 6.43 0.006797
## 28 12 6.48 0.006193
## 29 14 6.53 0.005643
## 30 16 6.58 0.005142
## 31 16 6.63 0.004685
## 32 16 6.67 0.004269
## 33 16 6.71 0.003890
## 34 17 6.74 0.003544
## 35 17 6.76 0.003229
## 36 17 6.79 0.002942
## 37 17 6.80 0.002681
## 38 17 6.82 0.002443
## 39 20 6.83 0.002226
## 40 20 6.85 0.002028
## 41 20 6.86 0.001848
## 42 22 6.87 0.001684
## 43 24 6.88 0.001534
## 44 26 6.89 0.001398
## 45 28 6.90 0.001274
## 46 30 6.91 0.001161
## 47 31 6.92 0.001057
## 48 31 6.93 0.000963
## 49 32 6.94 0.000878
## 50 33 6.95 0.000800
```

```
## 51 34 6.96 0.000729
## 52 35 6.98 0.000664
## 53 36 6.99 0.000605
## 54 36 7.00 0.000551
## 55 37 7.01 0.000502
## 56 37 7.02 0.000458
## 57 37 7.03 0.000417
## 58 38 7.03 0.000380
## 59 38 7.04 0.000346
## 60 38 7.04 0.000316
## 61 39 7.05 0.000288
## 62 39 7.05 0.000262
## 63 39 7.06 0.000239
## 64 39 7.06 0.000218
## 65 39 7.06 0.000198
## 66 38 7.07 0.000180
## 67 38 7.07 0.000164
## 68 38 7.07 0.000150
## 69 38 7.07 0.000137
## 70 38 7.07 0.000124
## 71 38 7.07 0.000113
```

```
plot(glmls_cv$glmnet.fit)
```




```
plot(glmls_cv$glmnet.fit, xvar="lambda")
```




```
#as.matrix(coef(glmls_cv, s = glmls_cv$lambda.min))
#as.matrix(coef(glmls_cv, s = glmls_cv$lambda.1se))


#the labmda values used are in glmls_cv$lambda
glmls_cv$lambda
```

```
##  [1] 0.0763540922 0.0695710051 0.0633905088 0.0577590707 0.0526279140
##  [6] 0.0479525952 0.0436926188 0.0398110870 0.0362743798 0.0330518638
## [11] 0.0301156273 0.0274402380 0.0250025229 0.0227813677 0.0207575338
## [16] 0.0189134917 0.0172332692 0.0157023131 0.0143073629 0.0130363362
## [21] 0.0118782240 0.0108229953 0.0098615102 0.0089854409 0.0081871991
## [26] 0.0074598709 0.0067971565 0.0061933158 0.0056431187 0.0051417996
## [31] 0.0046850163 0.0042688124 0.0038895828 0.0035440430 0.0032292000
## [36] 0.0029423268 0.0026809386 0.0024427714 0.0022257623 0.0020280317
## [41] 0.0018478670 0.0016837076 0.0015341316 0.0013978436 0.0012736630
## [46] 0.0011605143 0.0010574174 0.0009634794 0.0008778865 0.0007998975
## [51] 0.0007288368 0.0006640890 0.0006050931 0.0005513383 0.0005023589
## [56] 0.0004577308 0.0004170672 0.0003800161 0.0003462565 0.0003154961
## [61] 0.0002874683 0.0002619304 0.0002386612 0.0002174592 0.0001981407
## [66] 0.0001805384 0.0001644999 0.0001498862 0.0001365707 0.0001244381
## [71] 0.0001133834
```

```
# and the cross-validation 'loss' at each lambda is in glmls_cv$cvm
glmls_cv$cvm
```

```
##  [1] 0.8304065 0.8228304 0.8163836 0.8110919 0.8067410 0.8031442 0.8001722
##  [8] 0.7977096 0.7956683 0.7939745 0.7925666 0.7913910 0.7904136 0.7895797
## [15] 0.7885381 0.7870839 0.7855542 0.7842445 0.7831289 0.7821250 0.7811829
## [22] 0.7803680 0.7796745 0.7790821 0.7785507 0.7780645 0.7776579 0.7773076
## [29] 0.7769670 0.7766061 0.7762183 0.7758683 0.7755752 0.7753335 0.7751331
## [36] 0.7749673 0.7748375 0.7747401 0.7746698 0.7746088 0.7745580 0.7745132
## [43] 0.7744731 0.7744365 0.7744090 0.7743851 0.7743627 0.7743417 0.7743137
## [50] 0.7742775 0.7742361 0.7741572 0.7740726 0.7739959 0.7739332 0.7738777
## [57] 0.7738309 0.7737931 0.7737645 0.7737415 0.7737202 0.7736960 0.7736730
## [64] 0.7736527 0.7736363 0.7736223 0.7736116 0.7736033 0.7735973 0.7735931
## [71] 0.7735910
```

```
#So, to get the 'loss' value at lambda == lambda.1se
glmls_cv$cvm [ which(glmls_cv$lambda == glmls_cv$lambda.1se) ]
```

```
## [1] 0.7773076
```

```
#PREDICTIONS on Train

glmPredls_1=predict(glmls_cv,data.matrix(xDTrn), s="lambda.min" )

glmPredls_1p=predict(glmls_cv,data.matrix(xDTrn), s="lambda.min", type="response" ) #gives the pr
ob values
# gives the the ln(p/(1-p)) values
#i.e. the values of w1*x1 + ...+w2*x2


# AUC using default type.measure = "deviance"
predsauc <- prediction(glmPredls_1p, lcdfTrn$loan_status, label.ordering = c("Charged Off", "Full
y Paid"))
aucPerf <- performance(predsauc, "auc")
aucPerf@y.values
```

```
## [[1]]
## [1] 0.6929972
```

```
#PREDICTIONS on Test

glmPredls_1_Tst=predict ( glmls_cv,data.matrix(xDTst), s="lambda.min" )

glmPredls_1p_Tst=predict(glmls_cv,data.matrix(xDTst), s="lambda.min", type="response" ) #gives th
e prob values
# gives the the ln(p/(1-p)) values
#i.e. the values of w1*x1 + ...+w2*x2


# AUC using default type.measure = "deviance"
predsauc_Tst <- prediction(glmPredls_1p_Tst, lcdfTst$loan_status, label.ordering = c("Charged Of
f", "Fully Paid"))
aucPerf_Tst <- performance(predsauc, "auc")
aucPerf_Tst@y.values
```

```
## [[1]]
## [1] 0.6929972
```

Experiment with Ridge regularization as a variable selection method.

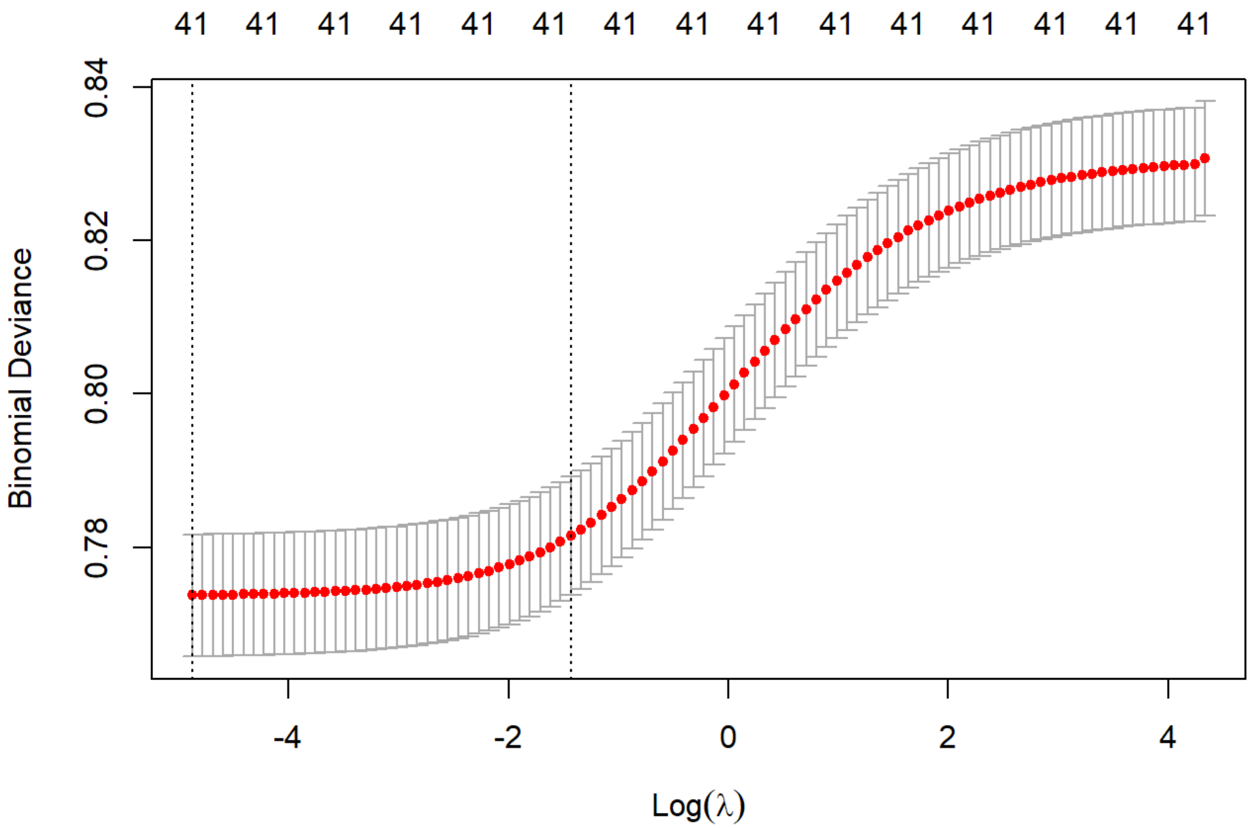
```
#Use Ridge regularization (alpha = 0)
glmls_cv_L2<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial", alpha = 0)

glmls_cv_L2$lambda.min
```

```
## [1] 0.007635409
```

```
glmls_cv_L2$lambda.1se
```

```
## [1] 0.2386612
```

```
plot(glmls_cv_L2)
```




```
coef(glmls_cv_L2, s = glmls_cv_L2$lambda.min)
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                          1
## (Intercept)                   3.483800e+00
## loan_amnt                    -1.094290e-06
## int_rate                     -4.191654e-02
## installment                  -1.338318e-04
## grade                        -1.070678e-01
## sub_grade                    -2.802382e-02
## emp_length                    1.141786e-02
## home_ownership               -6.030660e-02
## annual_inc                   -1.944640e-07
## purpose                       3.599765e-03
## dti                          -1.556590e-02
## initial_list_status          -2.379598e-02
## total_rev_hi_lim              2.220878e-06
## acc_open_past_24mths         -4.325442e-02
## avg_cur_bal                   1.795154e-06
## bc_open_to_buy                1.253395e-06
## bc_util                      -2.557541e-03
## chargeoff_within_12_mths     -4.185435e-02
## delinq_amnt                   2.594939e-05
## mo_sin_old_il_acct           -9.960001e-05
## mo_sin_old_rev_tl_op          2.502301e-04
## mo_sin_rcnt_rev_tl_op        -2.315311e-05
## mo_sin_rcnt_tl                2.741097e-03
## mort_acc                      2.678016e-02
## mths_since_recent_bc         -1.698771e-04
## mths_since_recent_inq         3.610931e-03
## num_bc_sats                  -1.624826e-02
## num_bc_tl                    -1.991454e-02
## num_il_tl                     3.147844e-03
## num_op_rev_tl                -1.416594e-02
## num_rev_accts                 9.194141e-03
## num_sats                      4.837194e-03
## num_tl_30dpd                 -2.030509e-01
## pct_tl_nvr_dlq                1.417863e-03
## tax_liens                    -3.212956e-02
## tot_hi_cred_lim               4.972057e-07
## total_bal_ex_mort            -1.451971e-06
## total_bc_limit                4.584778e-06
## total_il_high_credit_limit    1.673504e-06
## propSatisBankcardAccts       -5.312934e-02
## prop_OpAccts_to_TotAccts      1.936422e-01
## propLoanAmt_to_AnnInc        -5.692460e-01
```

```
coef(glmls_cv_L2, s = glmls_cv_L2$lambda.1se)
```

```
## 42 x 1 sparse Matrix of class "dgCMatrix"
##                                          1
## (Intercept)                  2.873218e+00
## loan_amnt                   -3.652773e-07
## int_rate                    -2.736051e-02
## installment                 -4.858538e-05
## grade                       -8.660296e-02
## sub_grade                   -1.810468e-02
## emp_length                   5.049587e-03
## home_ownership              -3.509874e-02
## annual_inc                   3.728905e-07
## purpose                     -3.244608e-03
## dti                         -6.565991e-03
## initial_list_status         -2.553814e-02
## total_rev_hi_lim             8.137283e-07
## acc_open_past_24mths        -1.847017e-02
## avg_cur_bal                  1.893942e-06
## bc_open_to_buy               2.178659e-06
## bc_util                     -1.250212e-03
## chargeoff_within_12_mths    -9.273238e-03
## delinq_amnt                  5.700583e-06
## mo_sin_old_il_acct          -1.410239e-05
## mo_sin_old_rev_tl_op         2.018724e-04
## mo_sin_rcnt_rev_tl_op        8.876395e-04
## mo_sin_rcnt_tl               2.432139e-03
## mort_acc                     1.493114e-02
## mths_since_recent_bc        -3.200099e-05
## mths_since_recent_inq        1.956409e-03
## num_bc_sats                 -4.399570e-03
## num_bc_tl                   -2.487589e-03
## num_il_tl                   -1.114332e-04
## num_op_rev_tl               -3.617182e-03
## num_rev_accts               -4.871645e-04
## num_sats                    -8.986079e-04
## num_tl_30dpd                -6.352455e-02
## pct_tl_nvr_dlq               3.508673e-04
## tax_liens                   -1.127259e-02
## tot_hi_cred_lim              2.027922e-07
## total_bal_ex_mort            8.237045e-08
## total_bc_limit               1.486659e-06
## total_il_high_credit_limit  1.557624e-07
## propSatisBankcardAccts      -1.407273e-02
## prop_OpAccts_to_TotAccts    -4.296392e-02
## propLoanAmt_to_AnnInc       -3.709809e-01
```

```
#find the index of the best lambda
which(glmls_cv_L2$lambda == glmls_cv_L2$lambda.1se)
```

```
## [1] 63
```

```
glmls_cv_L2$glmnet.fit$dev.ratio[which(glmls_cv_L2$lambda == glmls_cv_L2$lambda.1se) ]
```

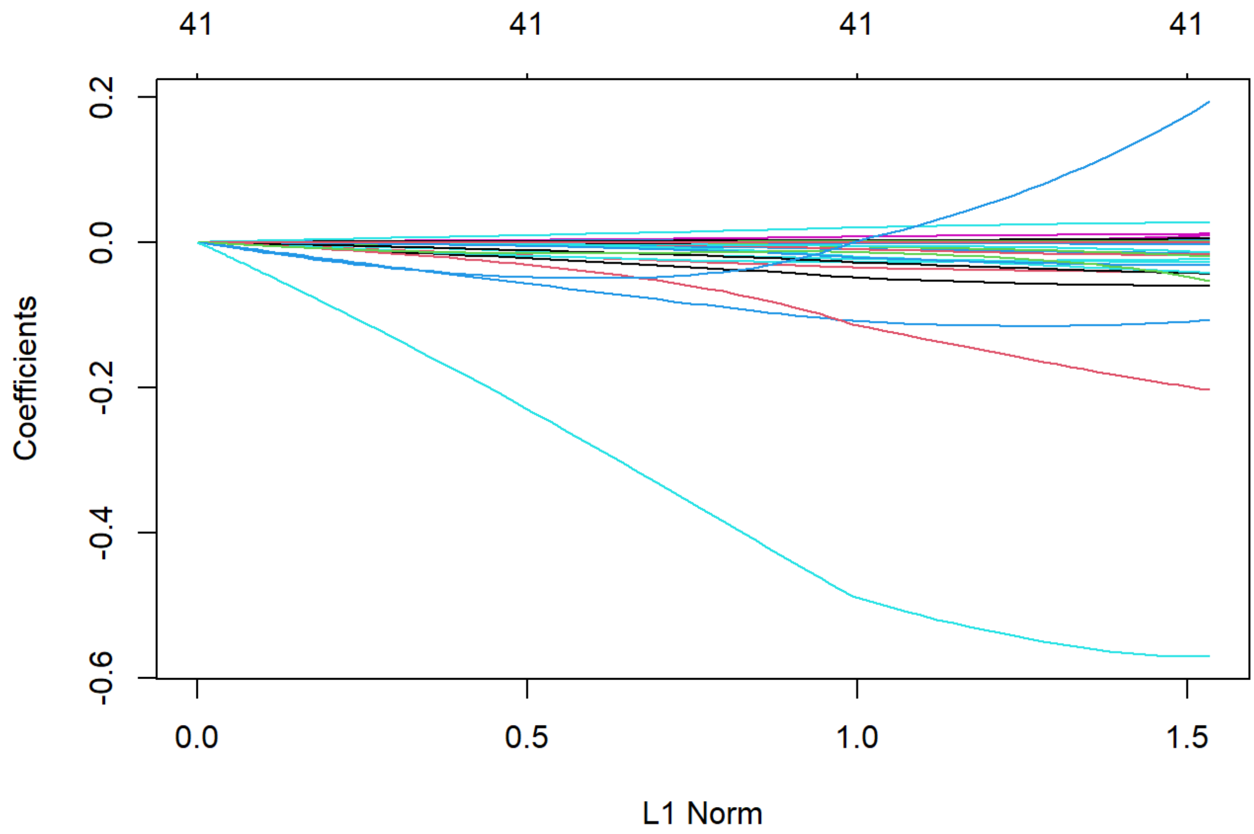
```
## [1] 0.05971116
```

```
glmls_cv_L2$glmnet.fit
```

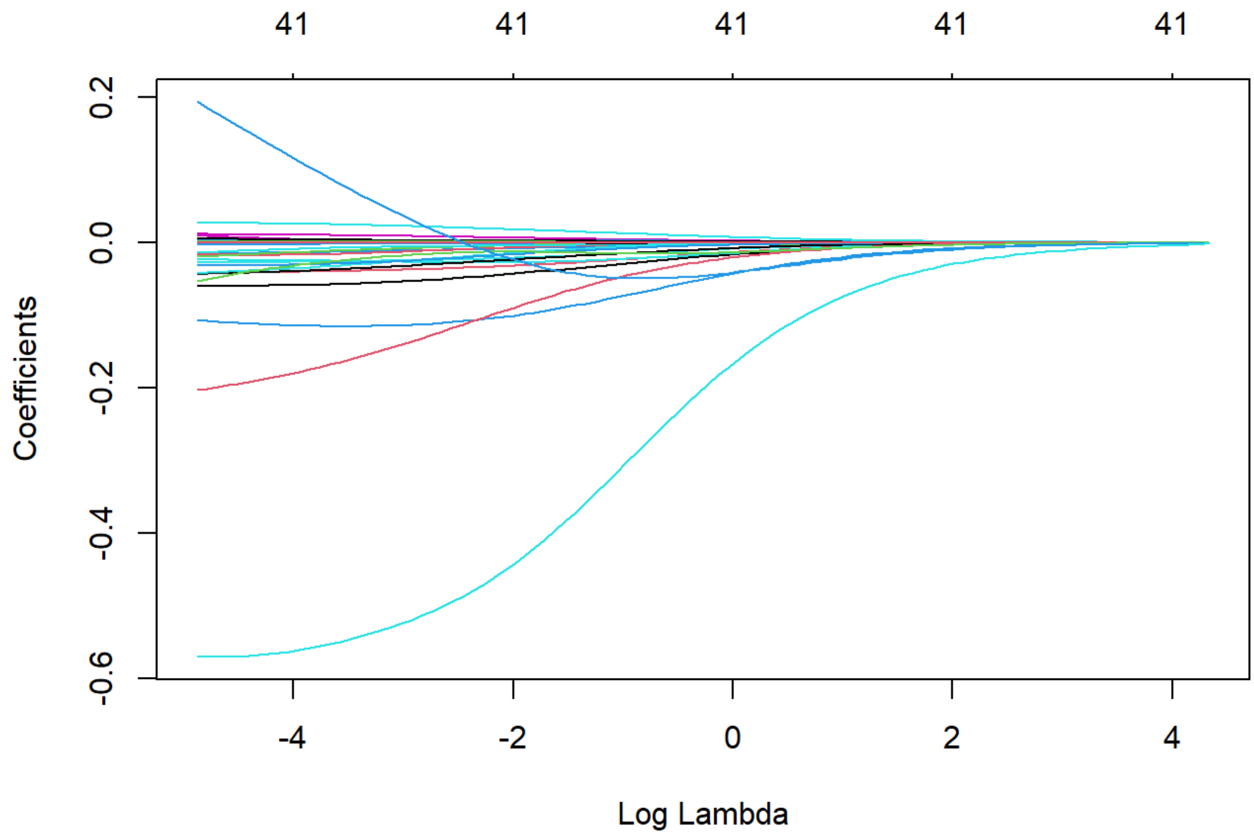
```
##
## Call:  glmnet(x = data.matrix(xDTrn), y = yTrn, family = "binomial",      alpha = 0)
##
##     Df %Dev Lambda
## 1   41 0.00 76.350
## 2   41 0.10 69.570
## 3   41 0.11 63.390
## 4   41 0.12 57.760
## 5   41 0.13 52.630
## 6   41 0.14 47.950
## 7   41 0.15 43.690
## 8   41 0.17 39.810
## 9   41 0.18 36.270
## 10  41 0.20 33.050
## 11  41 0.22 30.120
## 12  41 0.24 27.440
## 13  41 0.27 25.000
## 14  41 0.29 22.780
## 15  41 0.32 20.760
## 16  41 0.35 18.910
## 17  41 0.38 17.230
## 18  41 0.41 15.700
## 19  41 0.45 14.310
## 20  41 0.49 13.040
## 21  41 0.54 11.880
## 22  41 0.59 10.820
## 23  41 0.64  9.862
## 24  41 0.70  8.985
## 25  41 0.76  8.187
## 26  41 0.83  7.460
## 27  41 0.90  6.797
## 28  41 0.97  6.193
## 29  41 1.06  5.643
## 30  41 1.14  5.142
## 31  41 1.24  4.685
## 32  41 1.34  4.269
## 33  41 1.45  3.890
## 34  41 1.56  3.544
## 35  41 1.68  3.229
## 36  41 1.81  2.942
## 37  41 1.94  2.681
## 38  41 2.08  2.443
## 39  41 2.22  2.226
## 40  41 2.37  2.028
## 41  41 2.53  1.848
## 42  41 2.69  1.684
## 43  41 2.86  1.534
## 44  41 3.03  1.398
## 45  41 3.21  1.274
## 46  41 3.38  1.161
## 47  41 3.56  1.057
## 48  41 3.74  0.964
## 49  41 3.92  0.878
## 50  41 4.10  0.800
```

```
## 51  41 4.27  0.729
## 52  41 4.45  0.664
## 53  41 4.62  0.605
## 54  41 4.78  0.551
## 55  41 4.94  0.502
## 56  41 5.10  0.458
## 57  41 5.24  0.417
## 58  41 5.38  0.380
## 59  41 5.52  0.346
## 60  41 5.64  0.316
## 61  41 5.76  0.288
## 62  41 5.87  0.262
## 63  41 5.97  0.239
## 64  41 6.07  0.218
## 65  41 6.15  0.198
## 66  41 6.23  0.180
## 67  41 6.31  0.164
## 68  41 6.37  0.150
## 69  41 6.44  0.137
## 70  41 6.49  0.124
## 71  41 6.54  0.113
## 72  41 6.59  0.103
## 73  41 6.63  0.094
## 74  41 6.67  0.086
## 75  41 6.70  0.078
## 76  41 6.73  0.071
## 77  41 6.76  0.065
## 78  41 6.78  0.059
## 79  41 6.80  0.054
## 80  41 6.82  0.049
## 81  41 6.84  0.045
## 82  41 6.86  0.041
## 83  41 6.87  0.037
## 84  41 6.89  0.034
## 85  41 6.90  0.031
## 86  41 6.91  0.028
## 87  41 6.92  0.026
## 88  41 6.93  0.023
## 89  41 6.94  0.021
## 90  41 6.95  0.019
## 91  41 6.95  0.018
## 92  41 6.96  0.016
## 93  41 6.97  0.015
## 94  41 6.97  0.013
## 95  41 6.98  0.012
## 96  41 6.99  0.011
## 97  41 6.99  0.010
## 98  41 7.00  0.009
## 99  41 7.00  0.008
## 100 41 7.01  0.008
```

```
plot(glmls_cv_L2$glmnet.fit)
```




```
plot(glmls_cv_L2$glmnet.fit, xvar="lambda")
```




```
#as.matrix(coef(glmls_cv_L2, s = glmls_cv_L2$lambda.min))
#as.matrix(coef(glmls_cv_L2, s = glmls_cv_L2$lambda.1se))


#the labmda values used are in glmls_cv_L2$lambda
glmls_cv_L2$lambda
```

```
##   [1] 76.354092223 69.571005116 63.390508773 57.759070690 52.627914045
##   [6] 47.952595214 43.692618822 39.811086991 36.274379750 33.051863832
##  [11] 30.115627347 27.440238018 25.002522903 22.781367680 20.757533765
##  [16] 18.913491677 17.233269206 15.702313068 14.307362854 13.036336172
##  [21] 11.878223997 10.822995316  9.861510242  8.985440852  8.187199053
##  [26]  7.459870854  6.797156489  6.193315840  5.643118731  5.141799617
##  [31]  4.685016310  4.268812374  3.889582849  3.544043029  3.229200014
##  [36]  2.942326785  2.680938583  2.442771389  2.225762311  2.028031722
##  [41]  1.847866974  1.683707565  1.534131625  1.397843600  1.273663027
##  [46]  1.160514315  1.057417421  0.963479372  0.877886520  0.799897501
##  [51]  0.728836812  0.664088958  0.605093126  0.551338320  0.502358943
##  [56]  0.457730760  0.417067220  0.380016118  0.346256534  0.315496057
##  [61]  0.287468257  0.261930370  0.238661198  0.217459195  0.198140719
##  [66]  0.180538444  0.164499906  0.149886188  0.136570712  0.124438146
##  [71]  0.113383404  0.103310735  0.094132894  0.085770387  0.078150783
##  [76]  0.071208082  0.064882153  0.059118201  0.053866303  0.049080969
##  [81]  0.044720751  0.040747883  0.037127953  0.033829608  0.030824279
##  [86]  0.028085935  0.025590858  0.023317437  0.021245980  0.019358546
##  [91]  0.017638786  0.016071805  0.014644030  0.013343095  0.012157731
##  [96]  0.011077672  0.010093562  0.009196878  0.008379852  0.007635409
```

```
# and the cross-validation 'loss' at each lambda is in glmls_cv_L2$cvm
glmls_cv_L2$cvm
```

```
##   [1] 0.8307081 0.8299185 0.8298323 0.8297472 0.8296541 0.8295522 0.8294408
##   [8] 0.8293189 0.8291856 0.8290399 0.8288808 0.8287070 0.8285174 0.8283105
##  [15] 0.8280850 0.8278392 0.8275717 0.8272806 0.8269641 0.8266203 0.8262473
##  [22] 0.8258431 0.8254141 0.8249423 0.8244328 0.8238834 0.8232920 0.8226564
##  [29] 0.8219745 0.8212445 0.8204645 0.8196330 0.8187487 0.8178107 0.8168184
##  [36] 0.8157717 0.8146711 0.8135176 0.8123128 0.8110590 0.8097592 0.8084172
##  [43] 0.8070374 0.8056249 0.8041856 0.8027259 0.8012527 0.7997733 0.7982954
##  [50] 0.7968267 0.7953750 0.7939478 0.7925523 0.7911954 0.7898833 0.7886214
##  [57] 0.7874145 0.7862663 0.7851797 0.7841569 0.7831988 0.7823057 0.7814773
##  [64] 0.7807121 0.7800076 0.7793631 0.7787752 0.7782408 0.7777566 0.7773193
##  [71] 0.7769255 0.7765716 0.7762544 0.7759706 0.7757172 0.7754913 0.7752901
##  [78] 0.7751111 0.7749520 0.7748107 0.7746852 0.7745736 0.7744744 0.7743862
##  [85] 0.7743075 0.7742371 0.7741743 0.7741177 0.7740662 0.7740198 0.7739775
##  [92] 0.7739387 0.7739029 0.7738693 0.7738377 0.7738086 0.7737808 0.7737544
##  [99] 0.7737292 0.7737056
```

```
#So, to get the 'loss' value at lambda == lambda.1se
glmls_cv_L2$cvm [ which(glmls_cv_L2$lambda == glmls_cv_L2$lambda.1se) ]
```

```
## [1] 0.7814773
```

```
#PREDICTIONS on Trn

glmPredls_1_L2=predict ( glmls_cv_L2,data.matrix(xDTrn), s="lambda.min" )

glmPredls_1p_L2=predict(glmls_cv_L2,data.matrix(xDTrn), s="lambda.min", type="response" ) #gives
 the prob values
# gives the the ln(p/(1-p)) values
#i.e. the values of w1*x1 + ...+w2*x2


# AUC using default type.measure = "deviance"
preds_L2 <- prediction(glmPredls_1p_L2, lcdfTrn$loan_status, label.ordering = c("Charged Off", "F
ully Paid"))
aucPerf_L2 <- performance(preds_L2, "auc")
aucPerf_L2@y.values
```

```
## [[1]]
## [1] 0.6922717
```

```
#PREDICTIONS on Tst

glmPredls_1_L2_Tst=predict(glmls_cv_L2,data.matrix(xDTst), s="lambda.min" )

glmPredls_1p_L2_Tst=predict(glmls_cv_L2,data.matrix(xDTst), s="lambda.min", type="response" ) #gi
ves the prob values
# gives the the ln(p/(1-p)) values
#i.e. the values of w1*x1 + ...+w2*x2


# AUC using default type.measure = "deviance"
preds_L2_Tst <- prediction(glmPredls_1p_L2_Tst, lcdfTst$loan_status, label.ordering = c("Charged
 Off", "Fully Paid"))
aucPerf_L2_Tst <- performance(preds_L2_Tst, "auc")
aucPerf_L2_Tst@y.values
```

```
## [[1]]
## [1] 0.6967046
```

```
#glm models with different Alpha values
# alpha default, 1
glmRet_cv<- cv.glmnet(data.matrix(xDTrn), lcdfTrn$actualReturn, family="gaussian")
plot(glmRet_cv)
glmRet_cv$lambda.min
```

```
## [1] 5.194606e-06
```

```
glmRet_cv$lambda.1se
```

```
## [1] 0.002646204
```

```
coef(glmRet_cv, s = glmRet_cv$lambda.min) %>% tidy()
```

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")
```

```
## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

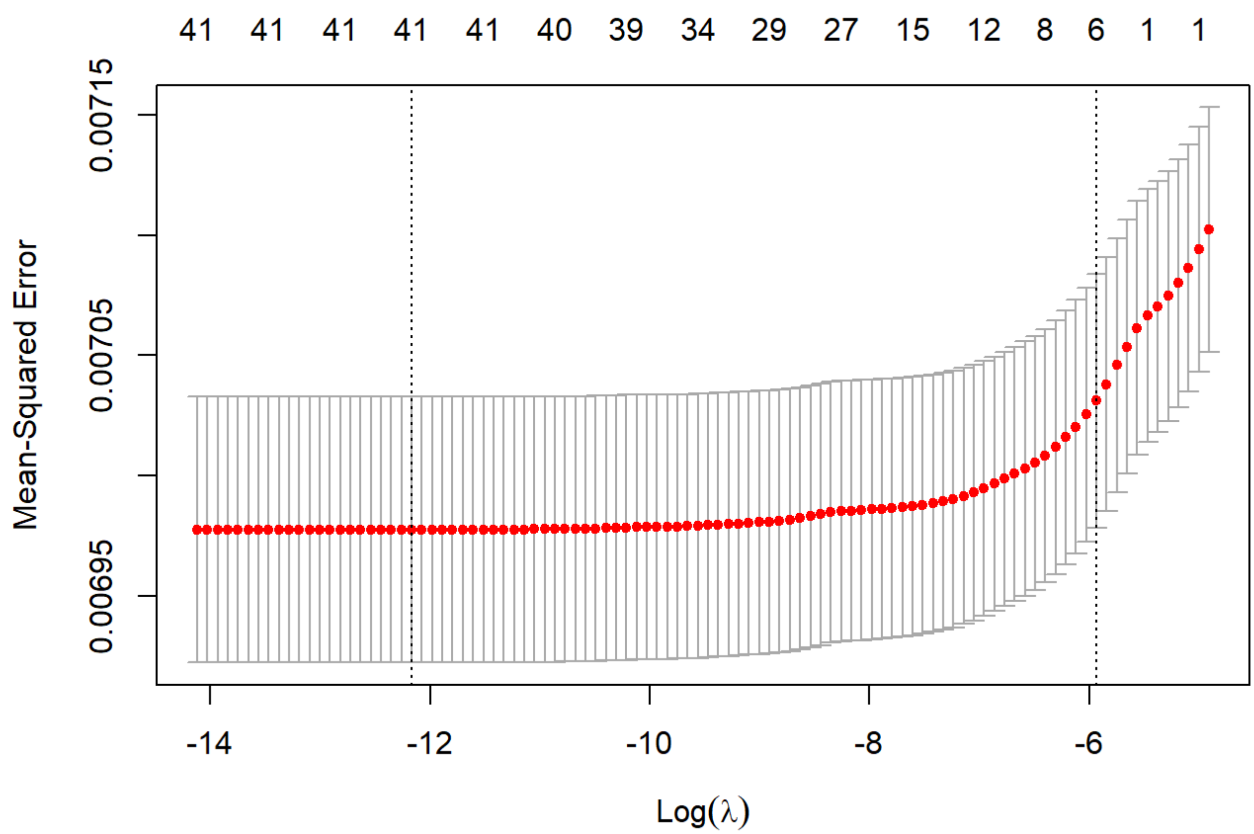
```
##                           row column         value
## 1                 (Intercept)      1  2.551692e-02
## 2                   loan_amnt      1  7.711433e-08
## 3                    int_rate      1  6.733179e-03
## 4                 installment      1 -3.558124e-06
## 5                       grade      1 -4.877115e-03
## 6                   sub_grade      1 -1.558048e-03
## 7                  emp_length      1  4.944113e-04
## 8              home_ownership      1 -2.042390e-03
## 9                  annual_inc      1 -2.768122e-08
## 10                    purpose      1 -4.498478e-05
## 11                        dti      1 -6.011703e-04
## 12        initial_list_status      1  3.474949e-04
## 13           total_rev_hi_lim      1  7.894354e-08
## 14       acc_open_past_24mths      1 -8.177517e-04
## 15                avg_cur_bal      1  4.116041e-08
## 16             bc_open_to_buy      1 -2.021713e-07
## 17                    bc_util      1 -9.341336e-05
## 18   chargeoff_within_12_mths      1 -2.829240e-03
## 19                delinq_amnt      1  6.748106e-07
## 20         mo_sin_old_il_acct      1 -3.732481e-06
## 21       mo_sin_old_rev_tl_op      1 -1.037648e-06
## 22      mo_sin_rcnt_rev_tl_op      1 -2.339319e-05
## 23             mo_sin_rcnt_tl      1  6.906570e-06
## 24                   mort_acc      1  8.831819e-04
## 25       mths_since_recent_bc      1 -8.402183e-06
## 26      mths_since_recent_inq      1  3.532926e-05
## 27                num_bc_sats      1  5.099128e-04
## 28                  num_bc_tl      1 -1.035082e-03
## 29                  num_il_tl      1  2.026320e-04
## 30              num_op_rev_tl      1 -7.447672e-04
## 31              num_rev_accts      1  6.797010e-04
## 32                   num_sats      1 -1.631254e-04
## 33               num_tl_30dpd      1 -7.533966e-03
## 34             pct_tl_nvr_dlq      1  1.312548e-05
## 35                  tax_liens      1 -1.649921e-03
## 36            tot_hi_cred_lim      1  4.796931e-09
## 37          total_bal_ex_mort      1 -8.295016e-08
## 38             total_bc_limit      1  1.215017e-07
## 39 total_il_high_credit_limit      1  1.116969e-07
## 40     propSatisBankcardAccts      1 -6.587370e-03
## 41   prop_OpAccts_to_TotAccts      1  9.566931e-03
## 42      propLoanAmt_to_AnnInc      1 -2.997153e-02
```

```
coef(glmRet_cv, s = glmRet_cv$lambda.1se) %>% tidy()
```

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")

## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

```
##                      row column         value
## 1            (Intercept)      1  3.950458e-02
## 2               int_rate      1  1.404950e-03
## 3         home_ownership      1 -7.944462e-05
## 4                    dti      1 -1.727085e-04
## 5            avg_cur_bal      1  2.702681e-08
## 6               mort_acc      1  1.862549e-04
## 7 propLoanAmt_to_AnnInc      1 -9.786915e-03
```

```
# alpha = 0
glmRet_cv_a0<- cv.glmnet(data.matrix(xDTrn), lcdfTrn$actualReturn, family="gaussian", alpha=0)
plot(glmRet_cv)
```




```
glmRet_cv_a0$lambda.min
```

```
## [1] 0.000736322
```

```
glmRet_cv_a0$lambda.1se
```

```
## [1] 0.4116636
```

```
coef(glmRet_cv_a0, s = glmRet_cv_a0$lambda.min) %>% tidy()
```

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")

## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

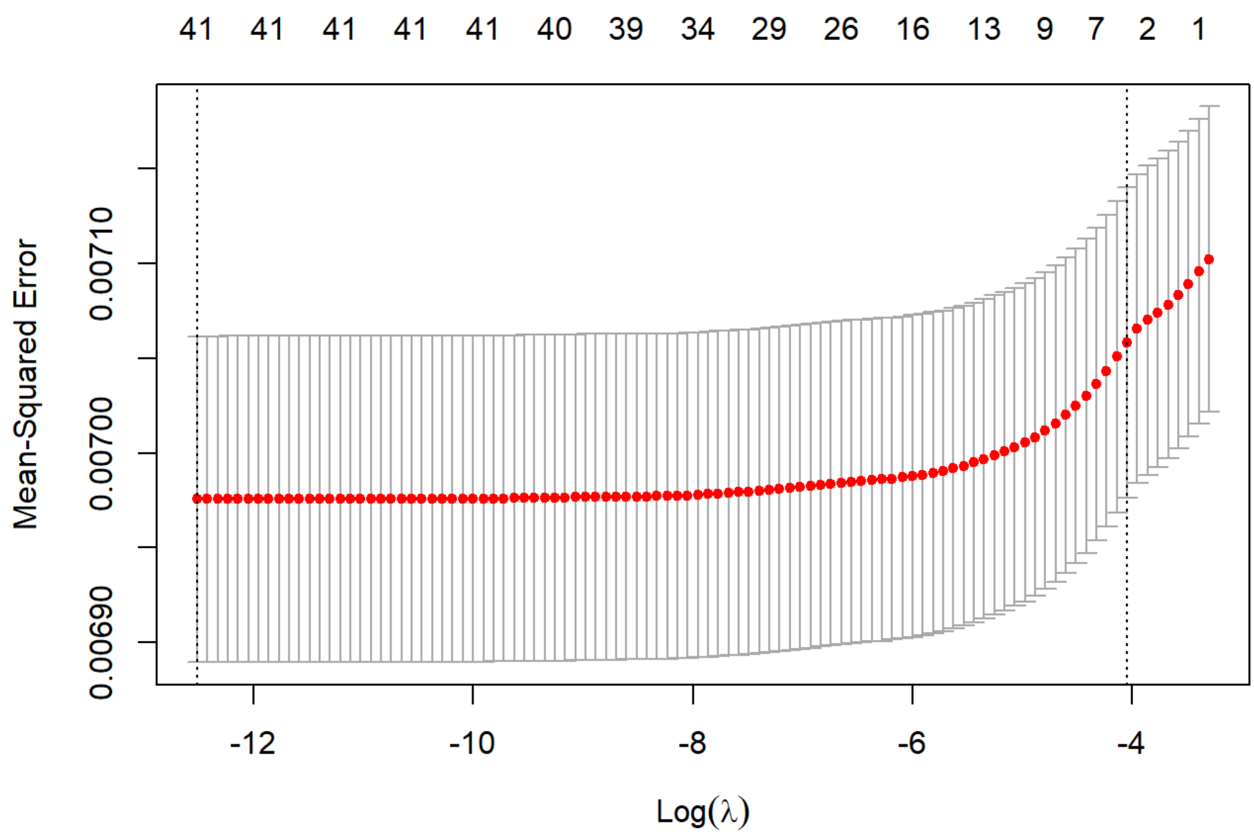
```
##                          row column         value
## 1                (Intercept)      1  4.172457e-02
## 2                  loan_amnt      1  8.442883e-08
## 3                   int_rate      1  3.624728e-03
## 4                installment      1 -3.480015e-06
## 5                      grade      1 -4.165564e-03
## 6                  sub_grade      1  3.110412e-04
## 7                 emp_length      1  4.930394e-04
## 8             home_ownership      1 -2.020220e-03
## 9                 annual_inc      1 -2.623814e-08
## 10                   purpose      1 -3.575762e-05
## 11                       dti      1 -5.808739e-04
## 12       initial_list_status      1  3.045555e-04
## 13          total_rev_hi_lim      1  6.343752e-08
## 14       acc_open_past_24mths      1 -8.040540e-04
## 15               avg_cur_bal      1  3.612892e-08
## 16             bc_open_to_buy      1 -1.698464e-07
## 17                   bc_util      1 -8.963615e-05
## 18   chargeoff_within_12_mths      1 -2.776782e-03
## 19               delinq_amnt      1  6.744389e-07
## 20          mo_sin_old_il_acct      1 -3.630714e-06
## 21       mo_sin_old_rev_tl_op      1 -1.121481e-06
## 22      mo_sin_rcnt_rev_tl_op      1 -2.550418e-05
## 23             mo_sin_rcnt_tl      1  8.269895e-06
## 24                  mort_acc      1  8.729466e-04
## 25       mths_since_recent_bc      1 -8.219525e-06
## 26      mths_since_recent_inq      1  3.368706e-05
## 27               num_bc_sats      1  3.188200e-04
## 28                 num_bc_tl      1 -8.790961e-04
## 29                 num_il_tl      1  1.821432e-04
## 30              num_op_rev_tl      1 -6.159420e-04
## 31              num_rev_accts      1  5.808111e-04
## 32                  num_sats      1 -1.470538e-04
## 33              num_tl_30dpd      1 -7.359863e-03
## 34              pct_tl_nvr_dlq      1  1.052302e-05
## 35                 tax_liens      1 -1.685867e-03
## 36             tot_hi_cred_lim      1  5.221320e-09
## 37           total_bal_ex_mort      1 -6.406478e-08
## 38             total_bc_limit      1  1.043928e-07
## 39 total_il_high_credit_limit      1  9.207458e-08
## 40     propSatisBankcardAccts      1 -5.551993e-03
## 41   prop_OpAccts_to_TotAccts      1  8.017315e-03
## 42      propLoanAmt_to_AnnInc      1 -3.044067e-02
```

```
coef(glmRet_cv_a0, s = glmRet_cv_a0$lambda.1se) %>% tidy()
```

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")

## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

```
##                          row column         value
## 1                (Intercept)      1  5.209616e-02
## 2                  loan_amnt      1 -2.033871e-08
## 3                   int_rate      1  2.568672e-04
## 4                installment      1 -3.136385e-07
## 5                      grade      1  7.079644e-04
## 6                  sub_grade      1  1.616860e-04
## 7                 emp_length      1  9.138598e-05
## 8             home_ownership      1 -4.227668e-04
## 9                 annual_inc      1  3.118059e-09
## 10                   purpose      1  2.063172e-04
## 11                       dti      1 -6.922019e-05
## 12       initial_list_status      1  2.435018e-04
## 13          total_rev_hi_lim      1 -2.012277e-09
## 14      acc_open_past_24mths      1 -1.163193e-05
## 15               avg_cur_bal      1  2.412000e-08
## 16            bc_open_to_buy      1 -1.124526e-08
## 17                   bc_util      1 -4.001027e-06
## 18   chargeoff_within_12_mths      1 -8.015508e-05
## 19               delinq_amnt      1  1.552436e-07
## 20         mo_sin_old_il_acct      1 -8.516995e-08
## 21      mo_sin_old_rev_tl_op      1 -7.351034e-07
## 22     mo_sin_rcnt_rev_tl_op      1 -6.686794e-06
## 23            mo_sin_rcnt_tl      1 -1.276974e-05
## 24                  mort_acc      1  2.110347e-04
## 25      mths_since_recent_bc      1 -2.158574e-07
## 26     mths_since_recent_inq      1 -8.759551e-06
## 27               num_bc_sats      1 -1.174914e-04
## 28                 num_bc_tl      1 -5.357285e-05
## 29                 num_il_tl      1  2.285276e-06
## 30             num_op_rev_tl      1 -6.221624e-05
## 31              num_rev_accts      1 -1.230585e-05
## 32                  num_sats      1 -3.814338e-05
## 33              num_tl_30dpd      1 -2.255909e-04
## 34             pct_tl_nvr_dlq      1 -2.476893e-05
## 35                tax_liens      1 -1.782960e-04
## 36            tot_hi_cred_lim      1  1.876635e-09
## 37          total_bal_ex_mort      1  6.393171e-10
## 38            total_bc_limit      1 -8.612982e-09
## 39 total_il_high_credit_limit      1  1.055736e-09
## 40      propSatisBankcardAccts      1 -3.630662e-04
## 41    prop_OpAccts_to_TotAccts      1 -8.039163e-04
## 42      propLoanAmt_to_AnnInc      1 -4.832487e-03
```

```
# alpha = 0.2
glmRet_cv_a2<- cv.glmnet(data.matrix(xDTrn), lcdfTrn$actualReturn, family="gaussian", alpha=0.2)
plot(glmRet_cv_a2)
```




```
glmRet_cv_a2$lambda.min
```

```
## [1] 3.68161e-06
```

```
glmRet_cv_a2$lambda.1se
```

```
## [1] 0.01749063
```

```
coef(glmRet_cv_a2, s = glmRet_cv_a2$lambda.min) %>% tidy()
```

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")

## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

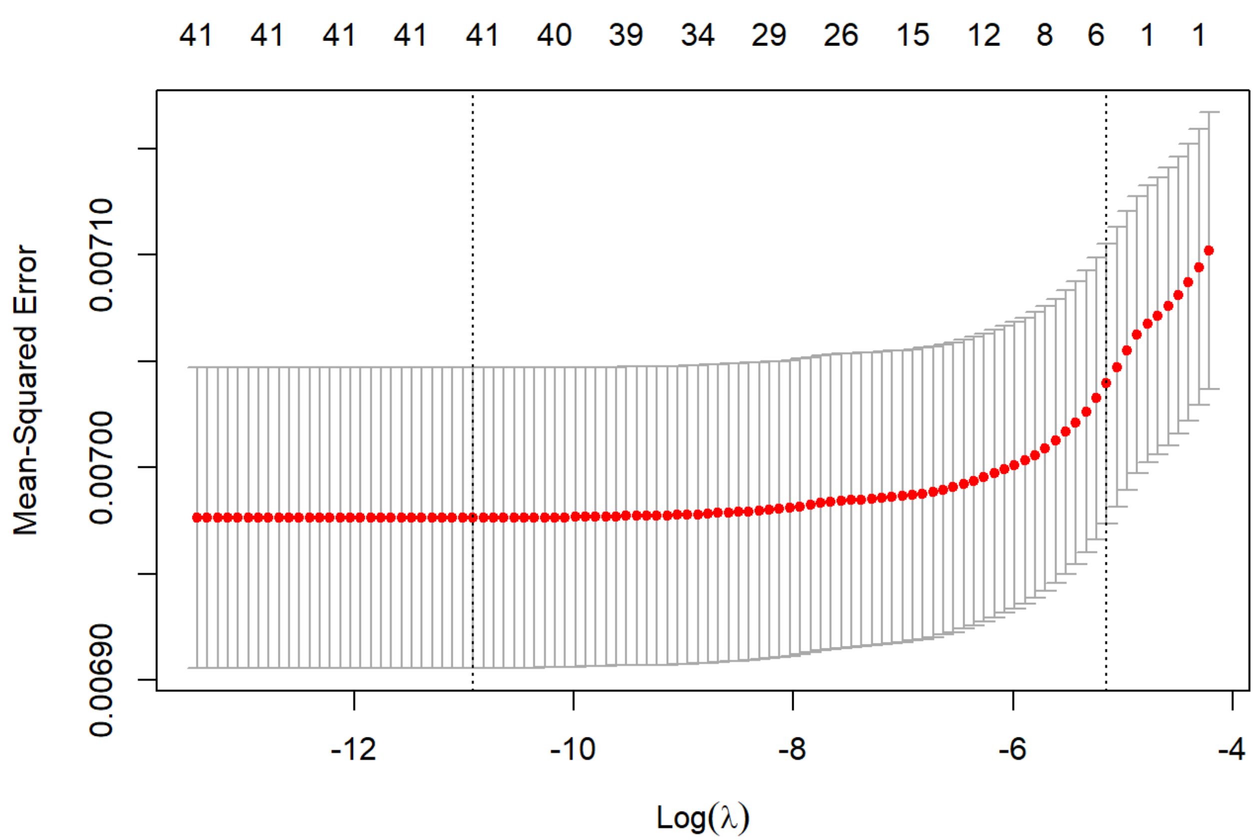
```
##                           row column         value
## 1                 (Intercept)      1  2.376534e-02
## 2                   loan_amnt      1  3.142668e-07
## 3                    int_rate      1  7.001775e-03
## 4                 installment      1 -1.059921e-05
## 5                       grade      1 -4.696537e-03
## 6                   sub_grade      1 -1.741281e-03
## 7                  emp_length      1  4.961373e-04
## 8              home_ownership      1 -2.052012e-03
## 9                  annual_inc      1 -2.813165e-08
## 10                    purpose      1 -4.855901e-05
## 11                        dti      1 -6.037904e-04
## 12        initial_list_status      1  3.597045e-04
## 13           total_rev_hi_lim      1  8.158254e-08
## 14       acc_open_past_24mths      1 -8.180106e-04
## 15                avg_cur_bal      1  4.200113e-08
## 16             bc_open_to_buy      1 -2.086637e-07
## 17                    bc_util      1 -9.427909e-05
## 18   chargeoff_within_12_mths      1 -2.862435e-03
## 19                delinq_amnt      1  6.816663e-07
## 20         mo_sin_old_il_acct      1 -3.788225e-06
## 21       mo_sin_old_rev_tl_op      1 -1.090545e-06
## 22      mo_sin_rcnt_rev_tl_op      1 -2.396128e-05
## 23             mo_sin_rcnt_tl      1  8.344806e-06
## 24                   mort_acc      1  8.887078e-04
## 25       mths_since_recent_bc      1 -8.548070e-06
## 26      mths_since_recent_inq      1  3.570044e-05
## 27                num_bc_sats      1  5.699533e-04
## 28                  num_bc_tl      1 -1.074260e-03
## 29                  num_il_tl      1  2.121088e-04
## 30              num_op_rev_tl      1 -7.685054e-04
## 31              num_rev_accts      1  7.023511e-04
## 32                   num_sats      1 -1.783152e-04
## 33                num_tl_30dpd      1 -7.614860e-03
## 34              pct_tl_nvr_dlq      1  1.346322e-05
## 35                  tax_liens      1 -1.652604e-03
## 36             tot_hi_cred_lim      1  4.702166e-09
## 37           total_bal_ex_mort      1 -8.589938e-08
## 38             total_bc_limit      1  1.229475e-07
## 39 total_il_high_credit_limit      1  1.146485e-07
## 40     propSatisBankcardAccts      1 -6.969269e-03
## 41   prop_OpAccts_to_TotAccts      1  1.011881e-02
## 42      propLoanAmt_to_AnnInc      1 -3.004989e-02
```

```
coef(glmRet_cv_a2, s = glmRet_cv_a2$lambda.1se) %>% tidy()
```

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")

## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

```
##                      row column         value
## 1            (Intercept)      1  4.327400e-02
## 2               int_rate      1  6.371532e-04
## 3              sub_grade      1  2.094240e-04
## 4                    dti      1 -5.444355e-05
## 5 propLoanAmt_to_AnnInc      1 -2.340863e-03
```

```
# alpha = 0.5
glmRet_cv_a5<- cv.glmnet(data.matrix(xDTrn), lcdfTrn$actualReturn, family="gaussian", alpha=0.5)
plot(glmRet_cv_a5)
```




```
glmRet_cv_a5$lambda.min
```

```
## [1] 1.815544e-05
```

```
glmRet_cv_a5$lambda.1se
```

```
## [1] 0.005808411
```

```
coef(glmRet_cv_a5, s = glmRet_cv_a5$lambda.min) %>% tidy()
```

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")

## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

```
##                          row column         value
## 1                (Intercept)      1  2.772876e-02
## 2                  loan_amnt      1  3.054229e-08
## 3                   int_rate      1  6.331441e-03
## 4                installment      1 -2.117222e-06
## 5                      grade      1 -4.906894e-03
## 6                  sub_grade      1 -1.296690e-03
## 7                 emp_length      1  4.934160e-04
## 8             home_ownership      1 -2.036826e-03
## 9                 annual_inc      1 -2.733392e-08
## 10                   purpose      1 -4.208124e-05
## 11                       dti      1 -5.981707e-04
## 12       initial_list_status      1  3.332394e-04
## 13           total_rev_hi_lim      1  7.613381e-08
## 14       acc_open_past_24mths      1 -8.160141e-04
## 15               avg_cur_bal      1  4.011380e-08
## 16            bc_open_to_buy      1 -1.951264e-07
## 17                   bc_util      1 -9.260309e-05
## 18   chargeoff_within_12_mths      1 -2.790830e-03
## 19               delinq_amnt      1  6.685484e-07
## 20          mo_sin_old_il_acct    1 -3.674688e-06
## 21        mo_sin_old_rev_tl_op    1 -9.921886e-07
## 22       mo_sin_rcnt_rev_tl_op    1 -2.308049e-05
## 23             mo_sin_rcnt_tl     1  5.897328e-06
## 24                  mort_acc      1  8.761491e-04
## 25        mths_since_recent_bc    1 -8.300803e-06
## 26       mths_since_recent_inq    1  3.492583e-05
## 27               num_bc_sats      1  4.625985e-04
## 28                 num_bc_tl      1 -1.002729e-03
## 29                 num_il_tl      1  1.930228e-04
## 30              num_op_rev_tl     1 -7.280963e-04
## 31              num_rev_accts     1  6.601328e-04
## 32                   num_sats     1 -1.464486e-04
## 33              num_tl_30dpd      1 -7.445361e-03
## 34             pct_tl_nvr_dlq     1  1.272934e-05
## 35                 tax_liens      1 -1.648265e-03
## 36            tot_hi_cred_lim     1  4.858040e-09
## 37          total_bal_ex_mort     1 -7.962659e-08
## 38            total_bc_limit      1  1.184235e-07
## 39 total_il_high_credit_limit     1  1.082874e-07
## 40     propSatisBankcardAccts     1 -6.259324e-03
## 41   prop_OpAccts_to_TotAccts     1  9.033403e-03
## 42      propLoanAmt_to_AnnInc     1 -3.003045e-02
```

```
coef(glmRet_cv_a5, s = glmRet_cv_a5$lambda.1se) %>% tidy()
```

```
## Warning: 'tidy.dgCMatrix' is deprecated.
## See help("Deprecated")

## Warning: 'tidy.dgTMatrix' is deprecated.
## See help("Deprecated")
```

```
##                    row column         value
## 1          (Intercept)      1  4.022190e-02
## 2             int_rate      1  1.249527e-03
## 3                  dti      1 -1.343694e-04
## 4          avg_cur_bal      1  1.840565e-08
## 5             mort_acc      1  8.777688e-05
## 6 propLoanAmt_to_AnnInc      1 -7.451321e-03
```

```
sum(yTrn == 0)
```

```
## [1] 8268
```

```
sum(yTrn == 1)
```

```
## [1] 48447
```

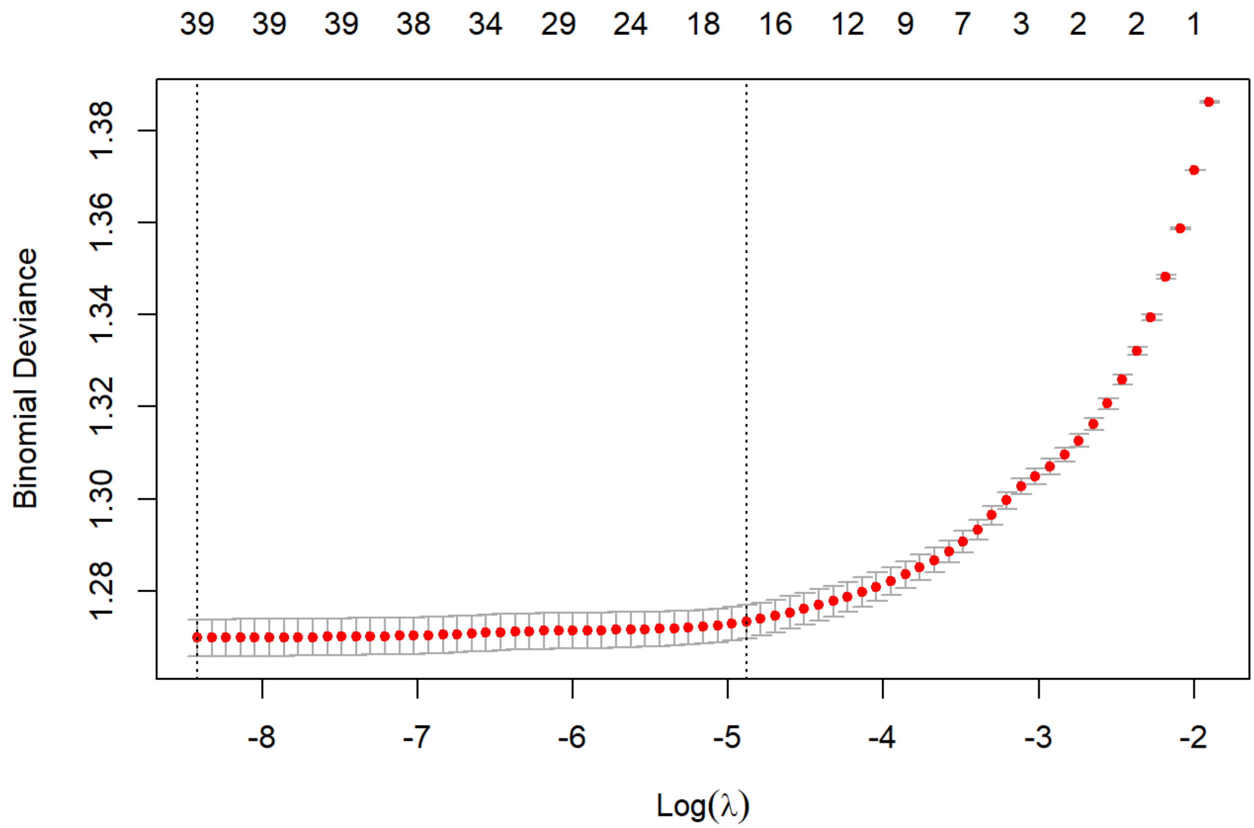
```
1-sum(yTrn == 0)/length(yTrn)
```

```
## [1] 0.8542185
```

```
1-sum(yTrn == 1)/length(yTrn)
```

```
## [1] 0.1457815
```

```
wts <- if_else(yTrn == 0, 1-sum(yTrn == 0)/length(yTrn), 1-sum(yTrn == 1)/length(yTrn))

# Lasso regularization, balanced weights, type.measure='deviance'
glmlsw_cv<- cv.glmnet(data.matrix(xDTrn), yTrn, family= "binomial", weights = wts, alpha = 1)
plot(glmlsw_cv)
```




```
#Prediction on Trn
glmPredls_balp=predict(glmlsw_cv,data.matrix(xDTrn), s="lambda.min", type="response" )

preds_auc_bal <- prediction(glmPredls_balp, lcdfTrn$loan_status, label.ordering = c("Charged Off"
, "Fully Paid"))
aucPerf_bal <- performance(preds_auc_bal, "auc")
aucPerf_bal@y.values
```

```
## [[1]]
## [1] 0.6930635
```

```
#Prediction on Tst
glmPredls_balp_Tst=predict(glmlsw_cv,data.matrix(xDTst), s="lambda.min", type="response" )

preds_auc_bal_Tst <- prediction(glmPredls_balp_Tst, lcdfTst$loan_status, label.ordering = c("Char
ged Off", "Fully Paid"))
aucPerf_bal_Tst <- performance(preds_auc_bal_Tst, "auc")
aucPerf_bal_Tst@y.values
```

```
## [[1]]
## [1] 0.6965404
```

We tried building a non-regularized model, but the resulting AUC is no better than the best regularized glmnet model we have so far. The AUC on Test data is 0.6886417.

```
#build glm model without regularization, using coefficient from the cross-validated model.
glmls_nzv_2 <- glm(yTrn ~ data.matrix(xDTrn %>% select(nzCoefVars)), family=binomial())
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(nzCoefVars)` instead of `nzCoefVars` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
summary(glmls_nzv_2)
```

```
##
## Call:
## glm(formula = yTrn ~ data.matrix(xDTrn %>% select(nzCoefVars)),
##     family = binomial())
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7950   0.3430   0.4671   0.6019   1.5698
##
## Coefficients:
##                                                                Estimate
## (Intercept)                                                   3.730e+00
## data.matrix(xDTrn %>% select(nzCoefVars))int_rate            -4.446e-02
## data.matrix(xDTrn %>% select(nzCoefVars))grade               -7.467e-02
## data.matrix(xDTrn %>% select(nzCoefVars))sub_grade           -3.563e-02
## data.matrix(xDTrn %>% select(nzCoefVars))home_ownership      -6.166e-02
## data.matrix(xDTrn %>% select(nzCoefVars))dti                 -1.291e-02
## data.matrix(xDTrn %>% select(nzCoefVars))acc_open_past_24mths -5.214e-02
## data.matrix(xDTrn %>% select(nzCoefVars))bc_open_to_buy       4.238e-06
## data.matrix(xDTrn %>% select(nzCoefVars))bc_util             -2.457e-03
## data.matrix(xDTrn %>% select(nzCoefVars))mort_acc             2.380e-02
## data.matrix(xDTrn %>% select(nzCoefVars))mths_since_recent_inq  4.450e-03
## data.matrix(xDTrn %>% select(nzCoefVars))tot_hi_cred_lim      8.551e-07
## data.matrix(xDTrn %>% select(nzCoefVars))propLoanAmt_to_AnnInc -8.644e-01
##                                                               Std. Error
## (Intercept)                                                   2.055e-01
## data.matrix(xDTrn %>% select(nzCoefVars))int_rate             3.476e-02
## data.matrix(xDTrn %>% select(nzCoefVars))grade                4.322e-02
## data.matrix(xDTrn %>% select(nzCoefVars))sub_grade            2.468e-02
## data.matrix(xDTrn %>% select(nzCoefVars))home_ownership       1.616e-02
## data.matrix(xDTrn %>% select(nzCoefVars))dti                  1.487e-03
## data.matrix(xDTrn %>% select(nzCoefVars))acc_open_past_24mths  4.152e-03
## data.matrix(xDTrn %>% select(nzCoefVars))bc_open_to_buy       1.505e-06
## data.matrix(xDTrn %>% select(nzCoefVars))bc_util              5.839e-04
## data.matrix(xDTrn %>% select(nzCoefVars))mort_acc             8.056e-03
## data.matrix(xDTrn %>% select(nzCoefVars))mths_since_recent_inq  1.016e-03
## data.matrix(xDTrn %>% select(nzCoefVars))tot_hi_cred_lim      1.211e-07
## data.matrix(xDTrn %>% select(nzCoefVars))propLoanAmt_to_AnnInc  1.181e-01
##                                                               z value Pr(>|z|)
## (Intercept)                                                    18.149  < 2e-16
## data.matrix(xDTrn %>% select(nzCoefVars))int_rate             -1.279 0.200912
## data.matrix(xDTrn %>% select(nzCoefVars))grade                -1.728 0.084066
## data.matrix(xDTrn %>% select(nzCoefVars))sub_grade            -1.443 0.148921
## data.matrix(xDTrn %>% select(nzCoefVars))home_ownership       -3.815 0.000136
## data.matrix(xDTrn %>% select(nzCoefVars))dti                  -8.682  < 2e-16
## data.matrix(xDTrn %>% select(nzCoefVars))acc_open_past_24mths -12.555  < 2e-16
## data.matrix(xDTrn %>% select(nzCoefVars))bc_open_to_buy        2.817 0.004854
## data.matrix(xDTrn %>% select(nzCoefVars))bc_util              -4.208 2.57e-05
## data.matrix(xDTrn %>% select(nzCoefVars))mort_acc              2.954 0.003135
## data.matrix(xDTrn %>% select(nzCoefVars))mths_since_recent_inq   4.379 1.19e-05
## data.matrix(xDTrn %>% select(nzCoefVars))tot_hi_cred_lim       7.059 1.68e-12
## data.matrix(xDTrn %>% select(nzCoefVars))propLoanAmt_to_AnnInc  -7.320 2.49e-13
##
## (Intercept)                                                   ***
```

```
## data.matrix(xDTrn %>% select(nzCoefVars))int_rate
## data.matrix(xDTrn %>% select(nzCoefVars))grade                .
## data.matrix(xDTrn %>% select(nzCoefVars))sub_grade
## data.matrix(xDTrn %>% select(nzCoefVars))home_ownership       ***
## data.matrix(xDTrn %>% select(nzCoefVars))dti                  ***
## data.matrix(xDTrn %>% select(nzCoefVars))acc_open_past_24mths  ***
## data.matrix(xDTrn %>% select(nzCoefVars))bc_open_to_buy       **
## data.matrix(xDTrn %>% select(nzCoefVars))bc_util              ***
## data.matrix(xDTrn %>% select(nzCoefVars))mort_acc             **
## data.matrix(xDTrn %>% select(nzCoefVars))mths_since_recent_inq ***
## data.matrix(xDTrn %>% select(nzCoefVars))tot_hi_cred_lim      ***
## data.matrix(xDTrn %>% select(nzCoefVars))propLoanAmt_to_AnnInc ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 47110  on 56714  degrees of freedom
## Residual deviance: 43930  on 56702  degrees of freedom
## AIC: 43956
##
## Number of Fisher Scoring iterations: 5
```

```
tidy(glmls_nzv_2)
```

```
## # A tibble: 13 x 5
##    term                                  estimate  std.error statistic  p.value
##    <chr>                                    <dbl>      <dbl>     <dbl>    <dbl>
##  1 (Intercept)                            3.73e+0    2.06e-1     18.1  1.31e-73
##  2 data.matrix(xDTrn %>% select(nzCoef~   -4.45e-2    3.48e-2     -1.28 2.01e- 1
##  3 data.matrix(xDTrn %>% select(nzCoef~   -7.47e-2    4.32e-2     -1.73 8.41e- 2
##  4 data.matrix(xDTrn %>% select(nzCoef~   -3.56e-2    2.47e-2     -1.44 1.49e- 1
##  5 data.matrix(xDTrn %>% select(nzCoef~   -6.17e-2    1.62e-2     -3.82 1.36e- 4
##  6 data.matrix(xDTrn %>% select(nzCoef~   -1.29e-2    1.49e-3     -8.68 3.90e-18
##  7 data.matrix(xDTrn %>% select(nzCoef~   -5.21e-2    4.15e-3    -12.6  3.71e-36
##  8 data.matrix(xDTrn %>% select(nzCoef~    4.24e-6    1.50e-6      2.82 4.85e- 3
##  9 data.matrix(xDTrn %>% select(nzCoef~   -2.46e-3    5.84e-4     -4.21 2.57e- 5
## 10 data.matrix(xDTrn %>% select(nzCoef~    2.38e-2    8.06e-3      2.95 3.14e- 3
## 11 data.matrix(xDTrn %>% select(nzCoef~    4.45e-3    1.02e-3      4.38 1.19e- 5
## 12 data.matrix(xDTrn %>% select(nzCoef~    8.55e-7    1.21e-7      7.06 1.68e-12
## 13 data.matrix(xDTrn %>% select(nzCoef~   -8.64e-1    1.18e-1     -7.32 2.49e-13
```

```
#PREDICTIONS on Trn
glmPredls_1_nonreg= predict(glmls_nzv_2,xDTrn)

glmPredls_1p_nonreg=predict(glmls_nzv_2, xDTrn, type="response" ) #gives the prob values

# AUC using non regularized glm
preds_nonreg <- prediction(glmPredls_1p_nonreg, lcdfTrn$loan_status, label.ordering = c("Charged
 Off", "Fully Paid"))
aucPerf_nonreg <- performance(preds_nonreg, "auc")
aucPerf_nonreg@y.values
```

```
## [[1]]
## [1] 0.6891976
```

```
#PREDICTIONS on Tst
glmPredls_1_nonreg_Tst= predict(glmls_nzv_2,xDTst)
```

```
## Warning: 'newdata' had 24307 rows but variables found have 56715 rows
```

```
glmPredls_1p_nonreg_Tst=predict(glmls_nzv_2, xDTst, type="response" ) #gives the prob values
```

```
## Warning: 'newdata' had 24307 rows but variables found have 56715 rows
```

```
# AUC using non regularized glm
preds_nonreg_Tst <- prediction(glmPredls_1p_nonreg_Tst, lcdfTrn$loan_status, label.ordering = c(
"Charged Off", "Fully Paid"))
aucPerf_nonreg_Tst <- performance(preds_nonreg, "auc")
aucPerf_nonreg_Tst@y.values
```

```
## [[1]]
## [1] 0.6891976
```

```
#Use no cross-validation model
glmls_1 <- glmnet(data.matrix(xDTrn), yTrn, family="binomial", lambda = glmls_cv$lambda.1se )
glmls_1
```

```
##
## Call:  glmnet(x = data.matrix(xDTrn), y = yTrn, family = "binomial",      lambda = glmls_cv$la
mbda.1se)
##
##   Df %Dev   Lambda
## 1 12 6.48 0.006193
```

1B. Experiment with loss and link function

Link function is used to fit our target variable to the requirement for the scale of glm. A glm, like any regression model, fits the target variable to a scale from negative infinity to positive infinity. In this case, a prediction of two classes (either “Fully Paid” or “Charged Off”) does not fulfil this criteria. Therefore, the family parameter helps to set the condition of our model to fit the glm algorithm.

For the binomial family, the valid link functions are logit, probit, cauchit. We set the family to binomial and we use the link function logit. Logit is taking the log of the odds of class 1 for the dependent variable loan status.

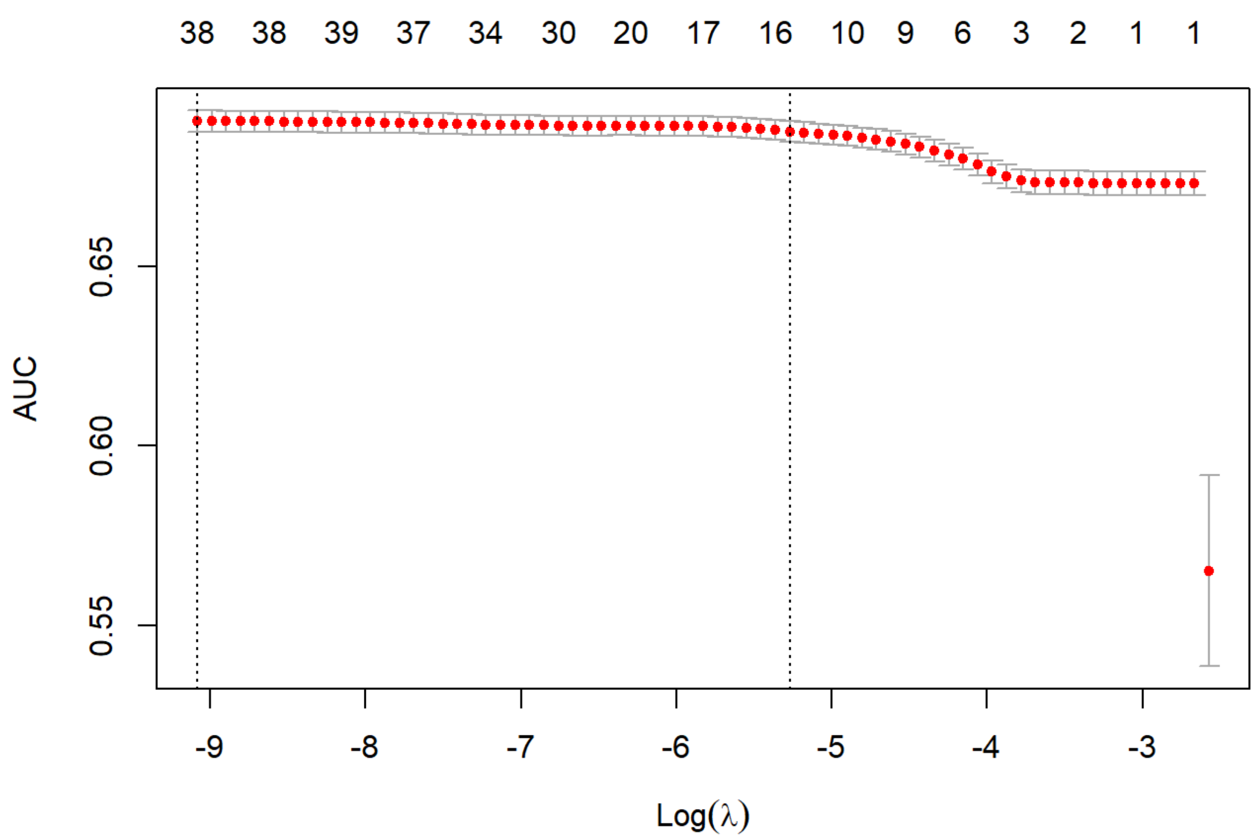
The loss function we choose will be tuned in the parameter “type.measure”. Since this is a logistic regression, there are three possible loss functions we can use.

We experimented on the loss functions: deviance, auc, and class. Please refer to the first table for the results for which one gave the best AUC performance. Type.measure = deviance constantly gives the best AUC performance despite other parameters. Type.measure = auc gives similar results, but deviance still gives the best results on test data.

For models using type.measure = deviance (default), please refer to the answers for question 1A.

Link Function source: https://www.r-bloggers.com/2018/10/generalized-linear-models-understanding-the-link-function/ (https://www.r-bloggers.com/2018/10/generalized-linear-models-understanding-the-link-function/)

```
#experiment with type.measure (LOSS FUNCTION), with alpha = 1

#type.measure = "auc"
glmls_cv_auc<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial", type.measure = "auc")
plot(glmls_cv_auc)
```



```
#PREDICTIONS with AUC model with Train Data
glmPredls_1_auc=predict(glmls_cv_auc,data.matrix(xDTrn), s="lambda.min" )

glmPredls_1p_auc=predict(glmls_cv_auc,data.matrix(xDTrn), s="lambda.min", type="response" ) #give
s the prob values
# gives the the ln(p/(1-p)) values
#i.e. the values of w1*x1 + ...+w2*x2


preds_auc <- prediction(glmPredls_1p_auc, lcdfTrn$loan_status, label.ordering = c("Charged Off",
"Fully Paid"))
aucPerf_auc <- performance(preds_auc, "auc")
aucPerf_auc@y.values
```

```
## [[1]]
## [1] 0.6929972
```

```
#PREDICTIONS with AUC model on Test Data
glmPredls_1_auc_Tst=predict(glmls_cv_auc,data.matrix(xDTst), s="lambda.min" )

glmPredls_1p_auc_Tst=predict(glmls_cv_auc,data.matrix(xDTst), s="lambda.min", type="response" ) #
gives the prob values
# gives the the ln(p/(1-p)) values
#i.e. the values of w1*x1 + ...+w2*x2


preds_auc_Tst <- prediction(glmPredls_1p_auc_Tst, lcdfTst$loan_status, label.ordering = c("Charge
d Off", "Fully Paid"))
aucPerf_auc_Tst <- performance(preds_auc_Tst, "auc")
aucPerf_auc_Tst@y.values
```

```
## [[1]]
## [1] 0.6966821
```

```
#the labmda values used
glmls_cv_auc$lambda
```

```
##  [1] 0.0763540922 0.0695710051 0.0633905088 0.0577590707 0.0526279140
##  [6] 0.0479525952 0.0436926188 0.0398110870 0.0362743798 0.0330518638
## [11] 0.0301156273 0.0274402380 0.0250025229 0.0227813677 0.0207575338
## [16] 0.0189134917 0.0172332692 0.0157023131 0.0143073629 0.0130363362
## [21] 0.0118782240 0.0108229953 0.0098615102 0.0089854409 0.0081871991
## [26] 0.0074598709 0.0067971565 0.0061933158 0.0056431187 0.0051417996
## [31] 0.0046850163 0.0042688124 0.0038895828 0.0035440430 0.0032292000
## [36] 0.0029423268 0.0026809386 0.0024427714 0.0022257623 0.0020280317
## [41] 0.0018478670 0.0016837076 0.0015341316 0.0013978436 0.0012736630
## [46] 0.0011605143 0.0010574174 0.0009634794 0.0008778865 0.0007998975
## [51] 0.0007288368 0.0006640890 0.0006050931 0.0005513383 0.0005023589
## [56] 0.0004577308 0.0004170672 0.0003800161 0.0003462565 0.0003154961
## [61] 0.0002874683 0.0002619304 0.0002386612 0.0002174592 0.0001981407
## [66] 0.0001805384 0.0001644999 0.0001498862 0.0001365707 0.0001244381
## [71] 0.0001133834
```

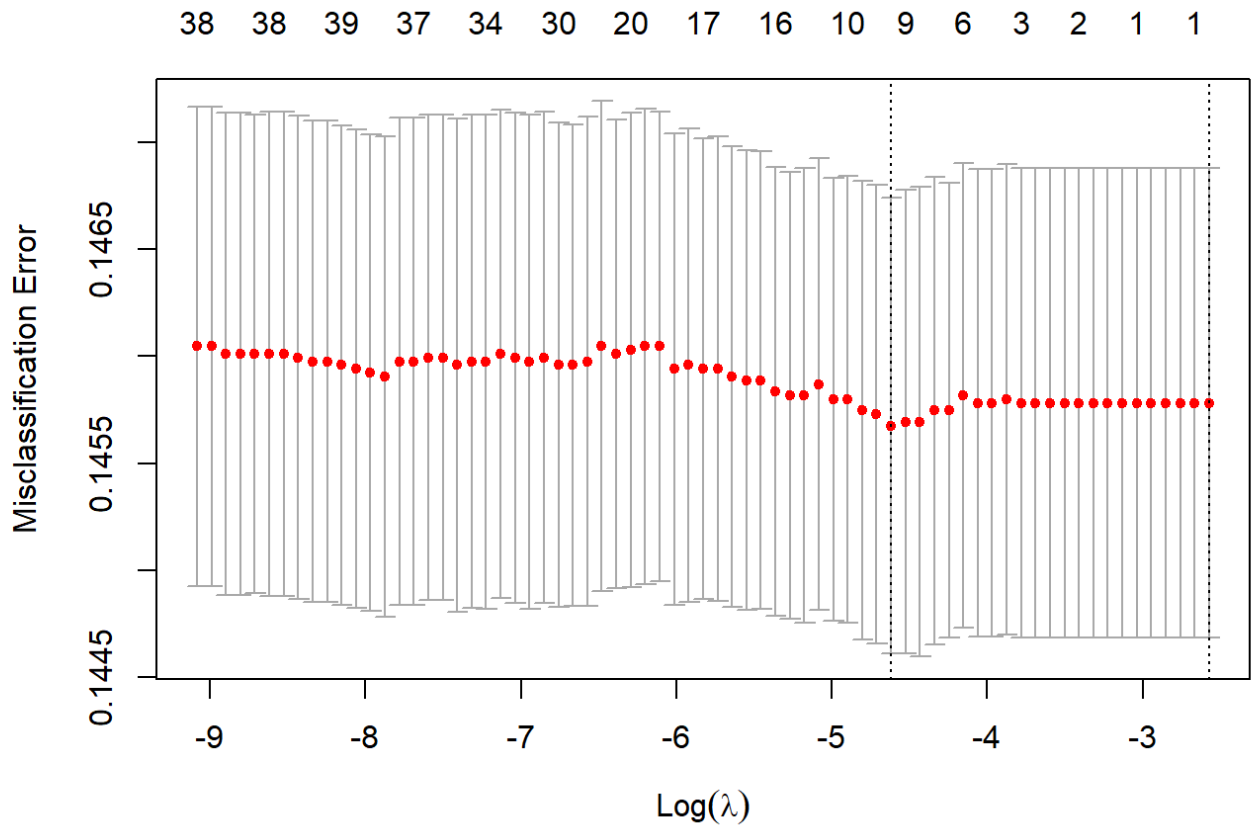
```
# and the cross-validation 'loss' at each lambda
glmls_cv_auc$cvm
```

```
##  [1] 0.5652220 0.6732776 0.6732776 0.6732776 0.6732479 0.6732479 0.6732212
##  [8] 0.6732212 0.6733284 0.6734491 0.6734535 0.6734432 0.6734499 0.6739759
## [15] 0.6751585 0.6764871 0.6784778 0.6800665 0.6812982 0.6823313 0.6833596
## [22] 0.6842020 0.6849100 0.6854603 0.6859668 0.6864247 0.6867836 0.6871123
## [29] 0.6874334 0.6877421 0.6880999 0.6884384 0.6887115 0.6889153 0.6890645
## [36] 0.6891757 0.6892723 0.6893261 0.6893524 0.6893606 0.6893761 0.6893781
## [43] 0.6893595 0.6893628 0.6893622 0.6893838 0.6894101 0.6894356 0.6894767
## [50] 0.6895196 0.6896009 0.6897022 0.6897960 0.6899101 0.6900071 0.6900846
## [57] 0.6901569 0.6902041 0.6902456 0.6902868 0.6903189 0.6903685 0.6904137
## [64] 0.6904563 0.6904842 0.6905095 0.6905321 0.6905518 0.6905686 0.6905758
## [71] 0.6905896
```

```
#So, to get the 'loss' value at lambda == lambda.1se
glmls_cv_auc$cvm [ which(glmls_cv_auc$lambda == glmls_cv_auc$lambda.1se) ]
```

```
## [1] 0.6877421
```

```
#type.measure = "class"
glmls_cv_class<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial", type.measure = "class")
plot(glmls_cv_class)
```




```
#PREDICTIONS with "class" model on Train
glmPredls_1_class=predict(glmls_cv_class,data.matrix(xDTrn), s="lambda.min" )

glmPredls_1p_class=predict(glmls_cv_class,data.matrix(xDTrn), s="lambda.min", type="response" ) #
gives the prob values
# gives the the ln(p/(1-p)) values
#i.e. the values of w1*x1 + ...+w2*x2


preds_class <- prediction(glmPredls_1p_class, lcdfTrn$loan_status, label.ordering = c("Charged Of
f", "Fully Paid"))
aucPerf_class <- performance(preds_class, "auc")
aucPerf_class@y.values
```

```
## [[1]]
## [1] 0.6855892
```

```
#PREDICTIONS with "class" model on Test
glmPredls_1_class_Tst=predict(glmls_cv_class,data.matrix(xDTst), s="lambda.min" )

glmPredls_1p_class_Tst=predict(glmls_cv_class,data.matrix(xDTst), s="lambda.min", type="response"
) #gives the prob values
# gives the the ln(p/(1-p)) values
#i.e. the values of w1*x1 + ...+w2*x2


preds_class_Tst <- prediction(glmPredls_1p_class_Tst, lcdfTst$loan_status, label.ordering = c("Ch
arged Off", "Fully Paid"))
aucPerf_class_Tst <- performance(preds_class_Tst, "auc")
aucPerf_class_Tst@y.values
```

```
## [[1]]
## [1] 0.6900923
```

```
#the labmda values used
glmls_cv_class$lambda
```

```
##  [1] 0.0763540922 0.0695710051 0.0633905088 0.0577590707 0.0526279140
##  [6] 0.0479525952 0.0436926188 0.0398110870 0.0362743798 0.0330518638
## [11] 0.0301156273 0.0274402380 0.0250025229 0.0227813677 0.0207575338
## [16] 0.0189134917 0.0172332692 0.0157023131 0.0143073629 0.0130363362
## [21] 0.0118782240 0.0108229953 0.0098615102 0.0089854409 0.0081871991
## [26] 0.0074598709 0.0067971565 0.0061933158 0.0056431187 0.0051417996
## [31] 0.0046850163 0.0042688124 0.0038895828 0.0035440430 0.0032292000
## [36] 0.0029423268 0.0026809386 0.0024427714 0.0022257623 0.0020280317
## [41] 0.0018478670 0.0016837076 0.0015341316 0.0013978436 0.0012736630
## [46] 0.0011605143 0.0010574174 0.0009634794 0.0008778865 0.0007998975
## [51] 0.0007288368 0.0006640890 0.0006050931 0.0005513383 0.0005023589
## [56] 0.0004577308 0.0004170672 0.0003800161 0.0003462565 0.0003154961
## [61] 0.0002874683 0.0002619304 0.0002386612 0.0002174592 0.0001981407
## [66] 0.0001805384 0.0001644999 0.0001498862 0.0001365707 0.0001244381
## [71] 0.0001133834
```

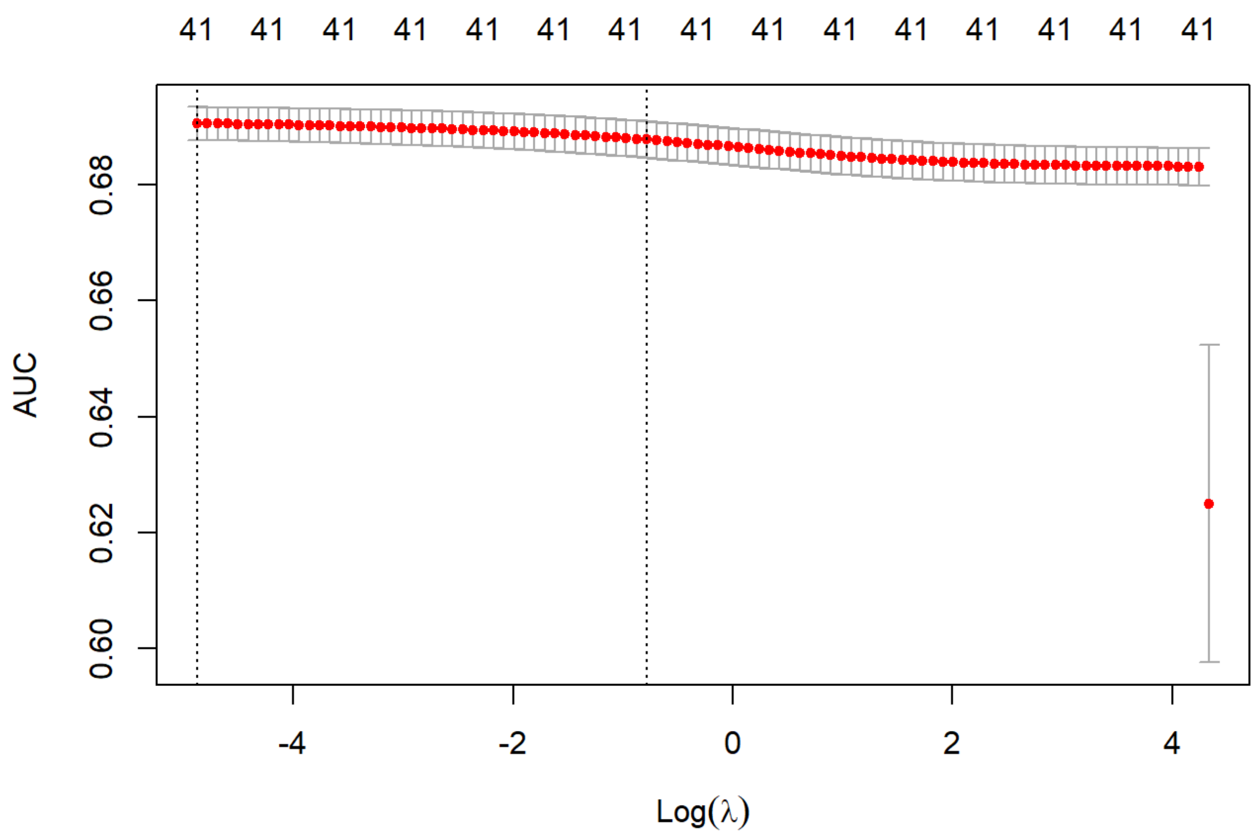
```
# and the cross-validation 'loss' at each lambda
glmls_cv_class$cvm
```

```
##  [1] 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815
##  [8] 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815
## [15] 0.1457992 0.1457815 0.1457815 0.1458168 0.1457463 0.1457463 0.1456934
## [22] 0.1456934 0.1456757 0.1457286 0.1457463 0.1457992 0.1457992 0.1458697
## [29] 0.1458168 0.1458168 0.1458344 0.1458873 0.1458873 0.1459050 0.1459402
## [36] 0.1459402 0.1459579 0.1459402 0.1460460 0.1460460 0.1460284 0.1460108
## [43] 0.1460460 0.1459755 0.1459579 0.1459579 0.1459931 0.1459755 0.1459931
## [50] 0.1460108 0.1459755 0.1459755 0.1459579 0.1459931 0.1459931 0.1459755
## [57] 0.1459755 0.1459050 0.1459226 0.1459402 0.1459579 0.1459755 0.1459755
## [64] 0.1459931 0.1460108 0.1460108 0.1460108 0.1460108 0.1460108 0.1460460
## [71] 0.1460460
```

```
#So, to get the 'loss' value at lambda == lambda.1se
glmls_cv_class$cvm [ which(glmls_cv_class$lambda == glmls_cv_class$lambda.1se) ]
```

```
## [1] 0.1457815
```

```
#experiment with type.measure, with alpha = 0

#type.measure = "auc"
glmls_cv_auc_L2<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial", type.measure = "auc", al
pha = 0)
plot(glmls_cv_auc_L2)
```



```
#PREDICTIONS with AUC model on Trn
glmPredls_1_auc_L2=predict(glmls_cv_auc_L2,data.matrix(xDTrn), s="lambda.min" )

glmPredls_1p_auc_L2=predict(glmls_cv_auc_L2,data.matrix(xDTrn), s="lambda.min", type="response" )
#gives the prob values
# gives the the ln(p/(1-p)) values
#i.e. the values of w1*x1 + ...+w2*x2


preds_auc_L2 <- prediction(glmPredls_1p_auc_L2, lcdfTrn$loan_status, label.ordering = c("Charged
 Off", "Fully Paid"))
aucPerf_auc_L2 <- performance(preds_auc_L2, "auc")
aucPerf_auc_L2@y.values
```

```
## [[1]]
## [1] 0.6922717
```

```
#PREDICTIONS with AUC model on Test
glmPredls_1_auc_L2_Tst=predict(glmls_cv_auc_L2,data.matrix(xDTst), s="lambda.min" )

glmPredls_1p_auc_L2_Tst=predict(glmls_cv_auc_L2,data.matrix(xDTst), s="lambda.min", type="respons
e" ) #gives the prob values
# gives the the ln(p/(1-p)) values
#i.e. the values of w1*x1 + ...+w2*x2


preds_auc_L2_Tst <- prediction(glmPredls_1p_auc_L2_Tst, lcdfTst$loan_status, label.ordering = c(
"Charged Off", "Fully Paid"))
aucPerf_auc_L2_Tst <- performance(preds_auc_L2_Tst, "auc")
aucPerf_auc_L2_Tst@y.values
```

```
## [[1]]
## [1] 0.6967046
```

```
#the labmda values used
glmls_cv_auc_L2$lambda
```

```
##   [1] 76.354092223 69.571005116 63.390508773 57.759070690 52.627914045
##   [6] 47.952595214 43.692618822 39.811086991 36.274379750 33.051863832
##  [11] 30.115627347 27.440238018 25.002522903 22.781367680 20.757533765
##  [16] 18.913491677 17.233269206 15.702313068 14.307362854 13.036336172
##  [21] 11.878223997 10.822995316  9.861510242  8.985440852  8.187199053
##  [26]  7.459870854  6.797156489  6.193315840  5.643118731  5.141799617
##  [31]  4.685016310  4.268812374  3.889582849  3.544043029  3.229200014
##  [36]  2.942326785  2.680938583  2.442771389  2.225762311  2.028031722
##  [41]  1.847866974  1.683707565  1.534131625  1.397843600  1.273663027
##  [46]  1.160514315  1.057417421  0.963479372  0.877886520  0.799897501
##  [51]  0.728836812  0.664088958  0.605093126  0.551338320  0.502358943
##  [56]  0.457730760  0.417067220  0.380016118  0.346256534  0.315496057
##  [61]  0.287468257  0.261930370  0.238661198  0.217459195  0.198140719
##  [66]  0.180538444  0.164499906  0.149886188  0.136570712  0.124438146
##  [71]  0.113383404  0.103310735  0.094132894  0.085770387  0.078150783
##  [76]  0.071208082  0.064882153  0.059118201  0.053866303  0.049080969
##  [81]  0.044720751  0.040747883  0.037127953  0.033829608  0.030824279
##  [86]  0.028085935  0.025590858  0.023317437  0.021245980  0.019358546
##  [91]  0.017638786  0.016071805  0.014644030  0.013343095  0.012157731
##  [96]  0.011077672  0.010093562  0.009196878  0.008379852  0.007635409
```

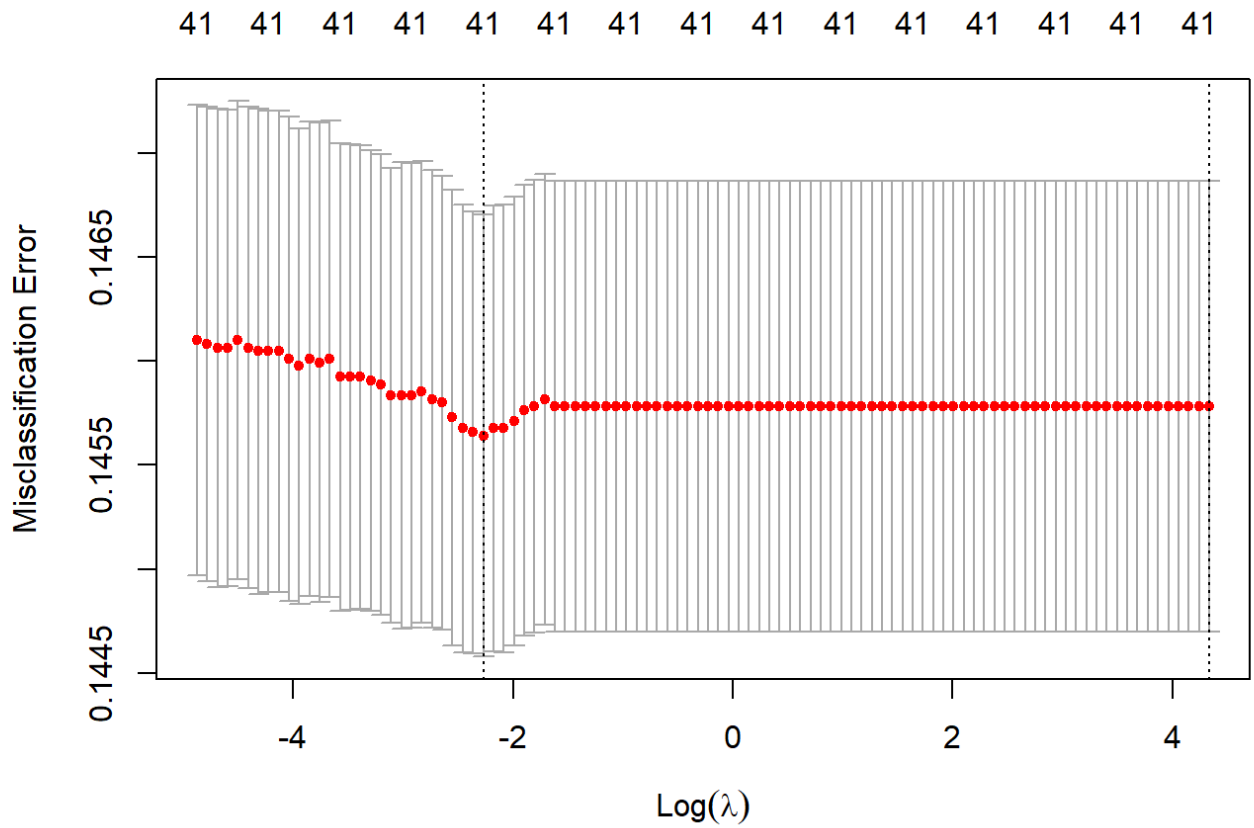
```
# and the cross-validation 'loss' at each lambda
glmls_cv_auc_L2$cvm
```

```
##   [1] 0.6249439 0.6831316 0.6831393 0.6831525 0.6831627 0.6831756 0.6831876
##   [8] 0.6832014 0.6832215 0.6832385 0.6832564 0.6832798 0.6833025 0.6833281
##  [15] 0.6833535 0.6833861 0.6834195 0.6834565 0.6834924 0.6835349 0.6835774
##  [22] 0.6836271 0.6836997 0.6837559 0.6838273 0.6838957 0.6839615 0.6840391
##  [29] 0.6841252 0.6842132 0.6843104 0.6844130 0.6845126 0.6846247 0.6847394
##  [36] 0.6848594 0.6849866 0.6851208 0.6852596 0.6853999 0.6855484 0.6857008
##  [43] 0.6858475 0.6860078 0.6861622 0.6863107 0.6864538 0.6866084 0.6867627
##  [50] 0.6869139 0.6870596 0.6872012 0.6873429 0.6874919 0.6876446 0.6877857
##  [57] 0.6879213 0.6880399 0.6881624 0.6882725 0.6883757 0.6884911 0.6885996
##  [64] 0.6887079 0.6888015 0.6888881 0.6889882 0.6890746 0.6891607 0.6892355
##  [71] 0.6893152 0.6893935 0.6894561 0.6895224 0.6895997 0.6896573 0.6897233
##  [78] 0.6897674 0.6898126 0.6898678 0.6899250 0.6899739 0.6900240 0.6900657
##  [85] 0.6901069 0.6901516 0.6901890 0.6902198 0.6902578 0.6902932 0.6903361
##  [92] 0.6903658 0.6903922 0.6904121 0.6904446 0.6904727 0.6905065 0.6905350
##  [99] 0.6905595 0.6905860
```

```
#So, to get the 'loss' value at lambda == lambda.1se
glmls_cv_auc_L2$cvm [ which(glmls_cv_auc_L2$lambda == glmls_cv_auc_L2$lambda.1se) ]
```

```
## [1] 0.6877857
```

```
#type.measure = "class"
glmls_cv_class_L2<- cv.glmnet(data.matrix(xDTrn), yTrn, family="binomial", type.measure = "class"
, alpha = 0)
plot(glmls_cv_class_L2)
```




```
#PREDICTIONS with "class" model on Trn
glmPredls_1_class_L2=predict(glmls_cv_class_L2,data.matrix(xDTrn), s="lambda.min" )

glmPredls_1p_class_L2=predict(glmls_cv_class_L2,data.matrix(xDTrn), s="lambda.min", type="respons
e" ) #gives the prob values
# gives the the ln(p/(1-p)) values
#i.e. the values of w1*x1 + ...+w2*x2


preds_class_L2 <- prediction(glmPredls_1p_class_L2, lcdfTrn$loan_status, label.ordering = c("Char
ged Off", "Fully Paid"))
aucPerf_class_L2 <- performance(preds_class_L2, "auc")
aucPerf_class_L2@y.values
```

```
## [[1]]
## [1] 0.6903123
```

```
#PREDICTIONS with "class" model on Tst
glmPredls_1_class_L2_Tst=predict(glmls_cv_class_L2,data.matrix(xDTst), s="lambda.min" )

glmPredls_1p_class_L2_Tst=predict(glmls_cv_class_L2,data.matrix(xDTst), s="lambda.min", type="res
ponse" ) #gives the prob values
# gives the the ln(p/(1-p)) values
#i.e. the values of w1*x1 + ...+w2*x2


preds_class_L2_Tst <- prediction(glmPredls_1p_class_L2_Tst, lcdfTst$loan_status, label.ordering =
c("Charged Off", "Fully Paid"))
aucPerf_class_L2_Tst <- performance(preds_class_L2_Tst, "auc")
aucPerf_class_L2_Tst@y.values
```

```
## [[1]]
## [1] 0.6955733
```

```
#the labmda values used
glmls_cv_class_L2$lambda
```

```
##   [1] 76.354092223 69.571005116 63.390508773 57.759070690 52.627914045
##   [6] 47.952595214 43.692618822 39.811086991 36.274379750 33.051863832
##  [11] 30.115627347 27.440238018 25.002522903 22.781367680 20.757533765
##  [16] 18.913491677 17.233269206 15.702313068 14.307362854 13.036336172
##  [21] 11.878223997 10.822995316  9.861510242  8.985440852  8.187199053
##  [26]  7.459870854  6.797156489  6.193315840  5.643118731  5.141799617
##  [31]  4.685016310  4.268812374  3.889582849  3.544043029  3.229200014
##  [36]  2.942326785  2.680938583  2.442771389  2.225762311  2.028031722
##  [41]  1.847866974  1.683707565  1.534131625  1.397843600  1.273663027
##  [46]  1.160514315  1.057417421  0.963479372  0.877886520  0.799897501
##  [51]  0.728836812  0.664088958  0.605093126  0.551338320  0.502358943
##  [56]  0.457730760  0.417067220  0.380016118  0.346256534  0.315496057
##  [61]  0.287468257  0.261930370  0.238661198  0.217459195  0.198140719
##  [66]  0.180538444  0.164499906  0.149886188  0.136570712  0.124438146
##  [71]  0.113383404  0.103310735  0.094132894  0.085770387  0.078150783
##  [76]  0.071208082  0.064882153  0.059118201  0.053866303  0.049080969
##  [81]  0.044720751  0.040747883  0.037127953  0.033829608  0.030824279
##  [86]  0.028085935  0.025590858  0.023317437  0.021245980  0.019358546
##  [91]  0.017638786  0.016071805  0.014644030  0.013343095  0.012157731
##  [96]  0.011077672  0.010093562  0.009196878  0.008379852  0.007635409
```

```
# and the cross-validation 'loss' at each lambda
glmls_cv_class_L2$cvm
```

```
##   [1] 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815
##   [8] 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815
##  [15] 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815
##  [22] 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815
##  [29] 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815
##  [36] 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815
##  [43] 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815
##  [50] 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815
##  [57] 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815 0.1457815
##  [64] 0.1457815 0.1457815 0.1458168 0.1457815 0.1457639 0.1457110 0.1456757
##  [71] 0.1456757 0.1456405 0.1456581 0.1456757 0.1457286 0.1457992 0.1458168
##  [78] 0.1458521 0.1458344 0.1458344 0.1458344 0.1458873 0.1459050 0.1459226
##  [85] 0.1459226 0.1459226 0.1460108 0.1459931 0.1460108 0.1459755 0.1460108
##  [92] 0.1460460 0.1460460 0.1460460 0.1460637 0.1460989 0.1460637 0.1460637
##  [99] 0.1460813 0.1460989
```

```
#So, to get the 'loss' value at lambda == lambda.1se
glmls_cv_class_L2$cvm [ which(glmls_cv_class_L2$lambda == glmls_cv_class_L2$lambda.1se) ]
```

```
## [1] 0.1457815
```

1C

Compared to random forest and xgboost models that we created last time, the results are pretty similar. The AUC scores on test data are all around .69. Ranger is especially accurate on the training set, scoring around .79. However, the ranger's random forest score for test data is no different to the other models.

1D

The variable importance of the glm model conflicts with those of ranger and xgboost (even with rpart and C5.0 decision trees). The different models have different algorithms, hence the different variable importance. To some extent, multicollinearity can help to explain the differences. In linear models, correlated independent variables can turn out to have weaker coefficients than they are supposed to have independently (IDS 570, Linear Regression Lecture Material).

Installment is ranked as the 11th most important in ranger, and 9th in XGboost, but it is ranked 29th in glmnet. This variable is at the top of the multicollinearity list, highly correlated with loan_amnt. This evidence alludes to our hypothesis that is multicollinearity exists, the coefficients do not reflect the true relationships of the independent variable with the target variable.

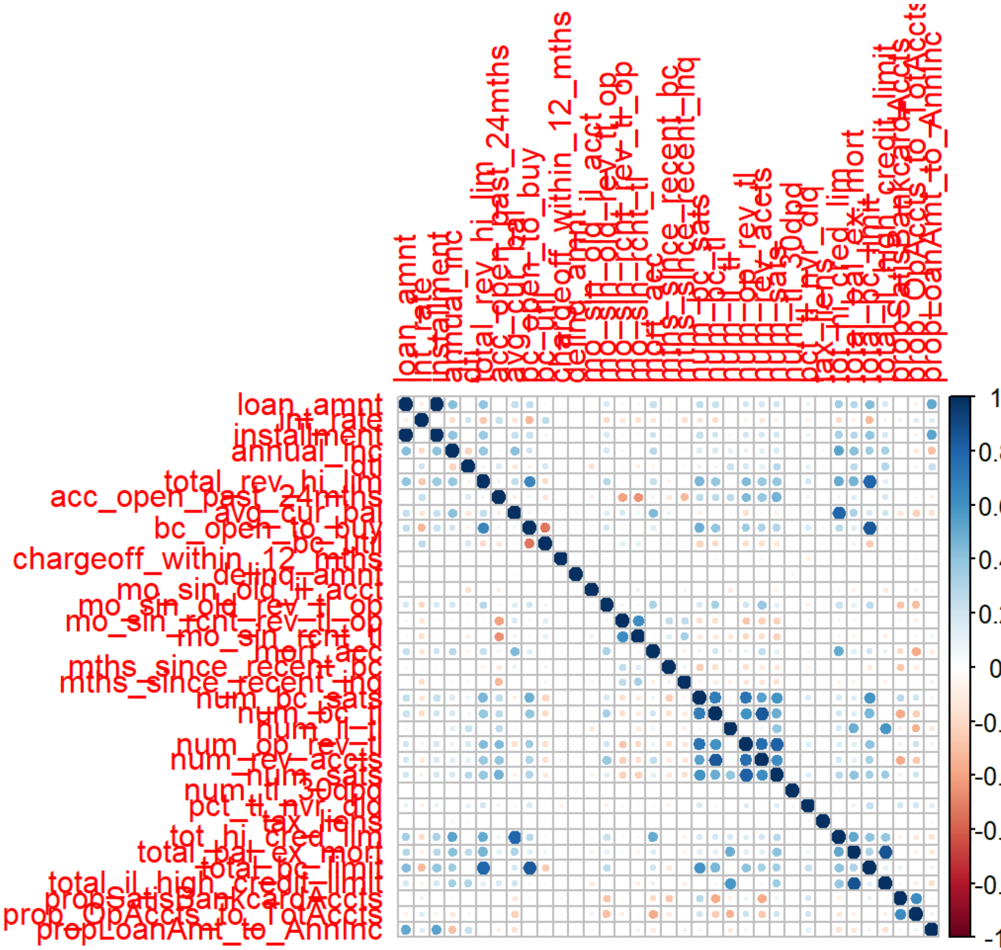
For the complete comparison of the variable importance between different models, please refer to our Appendix: Variable importance for different models.

Source: https://www.r-bloggers.com/2020/07/comparing-variable-importance-functions-for-modeling/ (https://www.r-bloggers.com/2020/07/comparing-variable-importance-functions-for-modeling/)

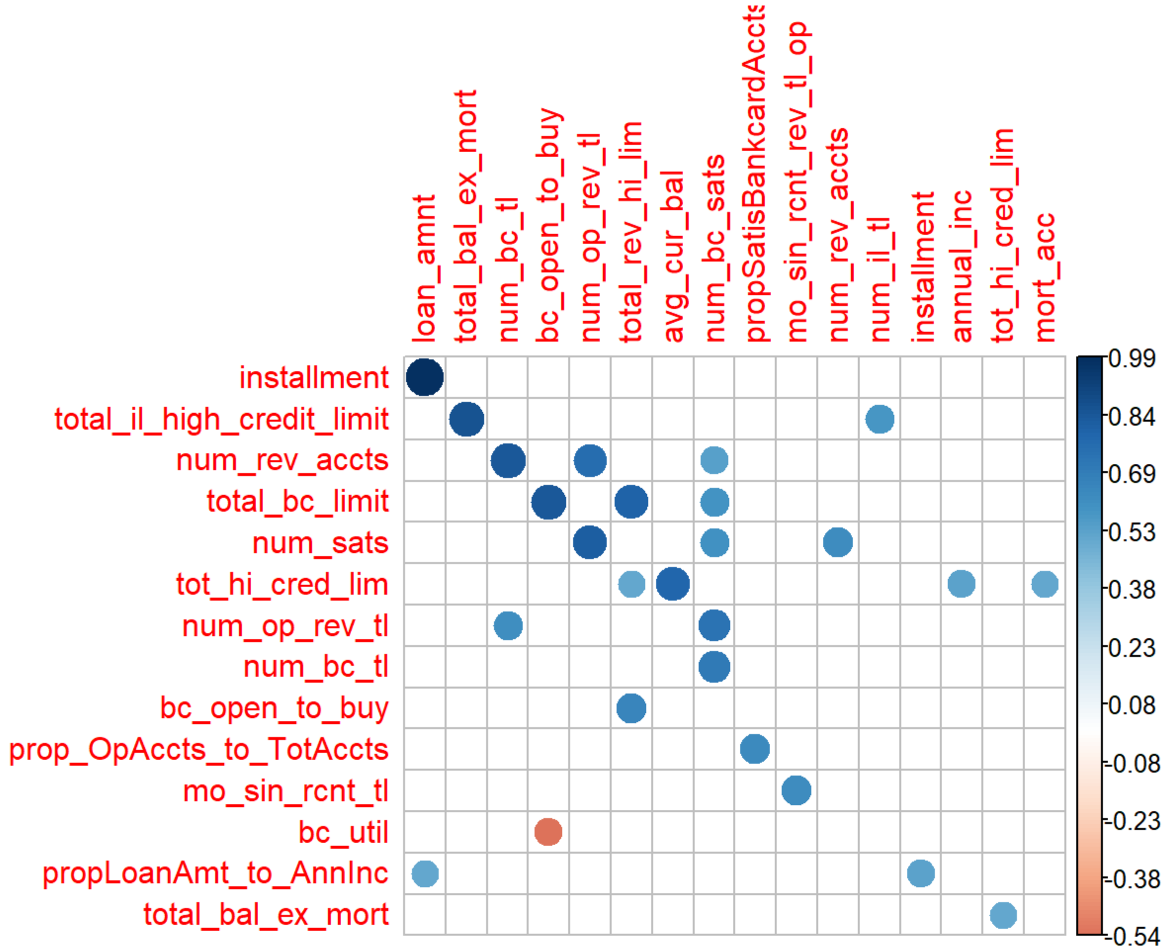
```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
xCorr <- xDTrn %>% select_if(is.numeric) %>% cor()
corrplot(xCorr, method="circle")
```




```
corrTH = 0.5
xCorr[upper.tri(xCorr, diag=TRUE)] <- NA #set the upper-diagonal values to NA
xCorr <- as.data.frame(as.table(xCorr))
xCorr <- na.omit(xCorr) #remove the rows corresponding to NA values
xCorr_th <- xCorr %>% filter(abs(Freq) > corrTH ) #remove the rows with abs(values) < corrTH
xCorr_th <- xCorr_th[order(-abs(xCorr_th$Freq)),] #order by the corr values

#Convert back to matrix form to use with corrPlot
xCorrMat <- xCorr_th %>% pivot_wider(names_from = Var2, values_from = Freq)
xCorrMat<-column_to_rownames(xCorrMat, var="Var1") #convert first column to rownames

corrplot(as.matrix(xCorrMat), is.corr=FALSE, na.label=" ", method="circle")
```



1E

We tried to balance the two classes represented in the target variable when building the model. We are using 3 methods: - under sampling (US): under sample the majority class, in this case “Fully Paid”. The resulting training dataset has less data points (about 16,000 data points) - over sampling (OS): over sample the majority class, in this case “Charged Off”. The resulting training dataset has more data points (about 96,000 data points) - both (under sampling and over sampling) (BS): combining both under sampling and over sampling, resulting in moderate number of data points (56,000 data points)

We build models with identical parameters (alpha = 1, type.measure = deviance) except for the training data set. The reason for this combination is that this combination resulted in the best AUC scores according to our experiment.

After running the code repeatedly, we have very similar results between OS, US, and BS. They only vary within less than .01 decimal points, and the order of AUC scores changes. This imply that the models we build are data dependent, and no method is ultimately better than the other.

We did not find evidence that concludes more data points caused by oversampling results in better models. This is intuitively true because copying/deleting existing examples do not enrich our dataset.

However, in the event of availability of new real data, more data is preferable. “Ideally, once you have more training examples you’ll have lower test-error (variance of the model decrease, meaning we are less overfitting), but theoretically, more data doesn’t always mean you will have more accurate model since high bias models will not benefit from more training examples.”

Source: https://stats.stackexchange.com/questions/31249/what-impact-does-increasing-the-training-data-have-on-the-overall-system-accurac# (https://stats.stackexchange.com/questions/31249/what-impact-does-increasing-the-training-data-have-on-the-overall-system-accurac#):~:text=Ideally%2C%20once%20you%20have%20more,benefit%20from%20more%20training%20examples.

```
library(ROSE)
```

```
## Warning: package 'ROSE' was built under R version 4.0.4
```

```
## Loaded ROSE 0.0-3
```

```
#Balancing the (training) data -- with over- and under-sampling


dim(lcdfTrn)
```

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

```
dim(lcdfTst)
```

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

```
us_lcdfTrn<-ovun.sample(loan_status~., data = as.data.frame(lcdfTrn), na.action = na.pass, method
="under", p=0.5)$data
dim(us_lcdfTrn)
```

```
## [1] 16672    46
```

```
#dim(lcdfTrn)
us_lcdfTrn %>% group_by(loan_status) %>% tally()
```

```
## # A tibble: 2 x 2
##   loan_status     n
##   <fct>       <int>
## 1 Fully Paid   8404
## 2 Charged Off  8268
```

```
os_lcdfTrn<-ovun.sample(loan_status~., data = as.data.frame(lcdfTrn), na.action = na.pass, method
="over", p=0.5)$data
dim(os_lcdfTrn)
```

```
## [1] 96807    46
```

```
os_lcdfTrn %>% group_by(loan_status) %>% tally()
```

```
## # A tibble: 2 x 2
##   loan_status     n
##   <fct>       <int>
## 1 Fully Paid  48447
## 2 Charged Off 48360
```

```
bs_lcdfTrn<-ovun.sample(loan_status~., data = as.data.frame(lcdfTrn), na.action = na.pass, method
="both", p=0.5)$data
dim(bs_lcdfTrn)
```

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

```
bs_lcdfTrn %>% group_by(loan_status) %>% tally()
```

```
## # A tibble: 2 x 2
##   loan_status     n
##   <fct>       <int>
## 1 Fully Paid  28404
## 2 Charged Off 28311
```

```
yTrn_bs<-factor(if_else(bs_lcdfTrn$loan_status=="Fully Paid", '1', '0') )
yTrn_os<-factor(if_else(os_lcdfTrn$loan_status=="Fully Paid", '1', '0') )
yTrn_us<-factor(if_else(us_lcdfTrn$loan_status=="Fully Paid", '1', '0') )


bs_lcdfTrn_del<-bs_lcdfTrn %>% select(-loan_status, -actualTerm, -annRet, -actualReturn, -total_p
ymnt)
os_lcdfTrn_del<-os_lcdfTrn %>% select(-loan_status, -actualTerm, -annRet, -actualReturn, -total_p
ymnt)
us_lcdfTrn_del<-us_lcdfTrn %>% select(-loan_status, -actualTerm, -annRet, -actualReturn, -total_p
ymnt)






glmls_cv_bs<- cv.glmnet(data.matrix(bs_lcdfTrn_del), yTrn_bs, family="binomial")
glmls_cv_os<- cv.glmnet(data.matrix(os_lcdfTrn_del), yTrn_os, family="binomial")
glmls_cv_us<- cv.glmnet(data.matrix(us_lcdfTrn_del), yTrn_us, family="binomial")


#BS
glmPredls_1_bs=predict ( glmls_cv_bs,data.matrix(bs_lcdfTrn_del), s="lambda.min" )

glmPredls_1p_bs=predict(glmls_cv_bs,data.matrix(bs_lcdfTrn_del), s="lambda.min", type="response"
 )

predsauc_bs <- prediction(glmPredls_1p_bs, bs_lcdfTrn$loan_status, label.ordering = c("Charged Of
f", "Fully Paid"))
aucPerf_bs <- performance(predsauc_bs, "auc")
aucPerf_bs@y.values
```

```
## [[1]]
## [1] 0.694004
```

```
#OS
glmPredls_1_os=predict ( glmls_cv_os,data.matrix(os_lcdfTrn_del), s="lambda.min" )

glmPredls_1p_os=predict(glmls_cv_os,data.matrix(os_lcdfTrn_del), s="lambda.min", type="response"
 )

predsauc_os <- prediction(glmPredls_1p_os, os_lcdfTrn$loan_status, label.ordering = c("Charged Of
f", "Fully Paid"))
aucPerf_os <- performance(predsauc_os, "auc")
aucPerf_os@y.values
```

```
## [[1]]
## [1] 0.6915689
```

```
#US
glmPredls_1_us=predict ( glmls_cv_us,data.matrix(us_lcdfTrn_del), s="lambda.min" )

glmPredls_1p_us=predict(glmls_cv_us,data.matrix(us_lcdfTrn_del), s="lambda.min", type="response"
 )

predsauc_us <- prediction(glmPredls_1p_us, us_lcdfTrn$loan_status, label.ordering = c("Charged Of
f", "Fully Paid"))
aucPerf_us <- performance(predsauc_us, "auc")
aucPerf_us@y.values
```

```
## [[1]]
## [1] 0.6949481
```

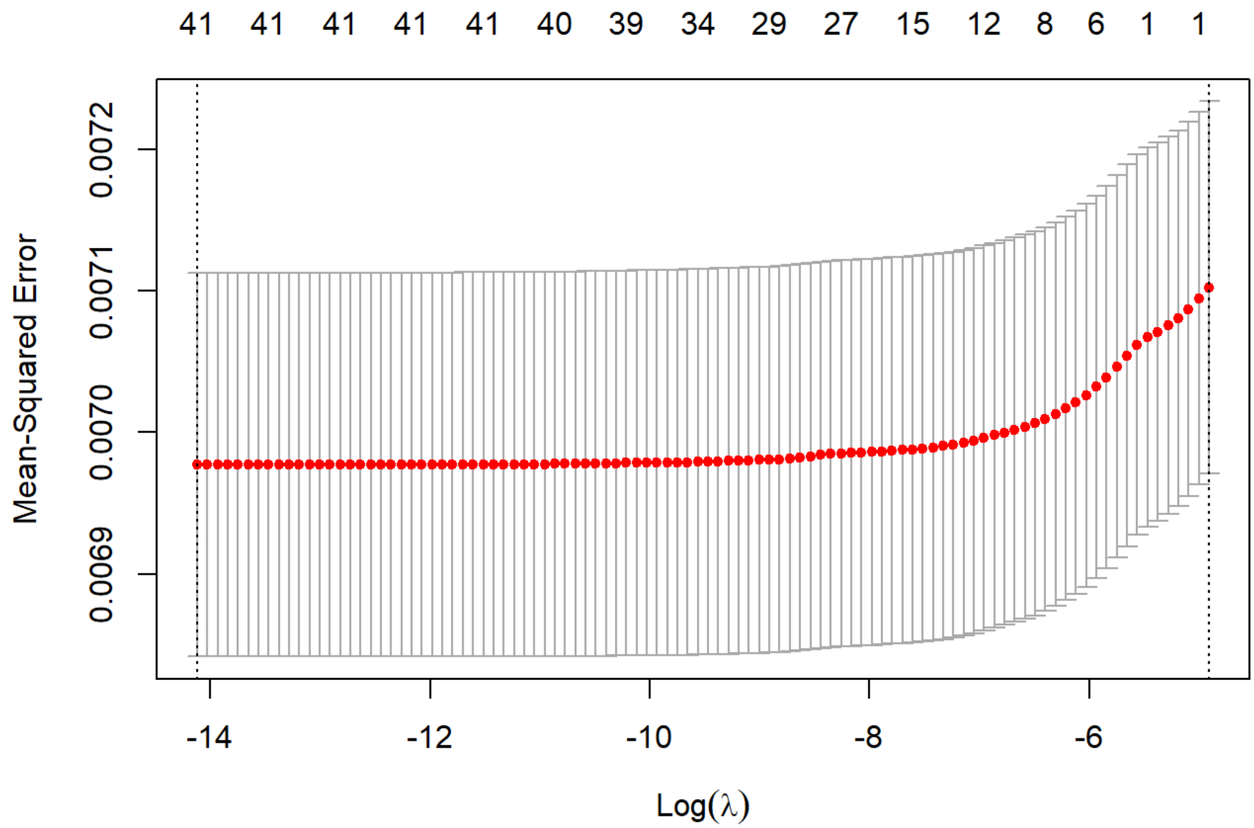
Q2: Create Models to Predict Actual Return

The variable “actualReturn” is the ratio of money made by the investors out of their total investment. It is created by subtracting the funded amount (amount invested by investors) from the total payment (amount paid to lending club by borrowers), divided by the funded amount. So far it only reflects the annual return, so we multiply it by the actual term of the loan (how long it takes for the loan to be paid back). This is true only if there was any attempt to pay back the loan at all (the actualTerm is > 0). If the loan was never paid back, this calculation does not reflect the actualReturn, so we assign the value 0 (there is no return/money made from that loan). The actualReturn variable DOES NOT account for Lending Club’s service fees (2-6% from the borrowers, and 1% from the investors).

“Lending Club makes money by charging borrowers an origination fee and investors a service fee. The size of the origination fee depends on the credit grade and ranges to be 2%-6.0% of the loan amount. The size of the service fee is 1% on all amounts the borrower pays.” Source: - https://www.investopedia.com/articles/investing/092315/7-best-peertopeer-lending-websites.asp (https://www.investopedia.com/articles/investing/092315/7-best-peertopeer-lending-websites.asp) - https://en.wikipedia.org/wiki/LendingClub (https://en.wikipedia.org/wiki/LendingClub)

To calculate the true income for the investors, the actualReturn should be subtracted by 1% if the value is not 0 (at least 1 pay back was made on the loan).

```
library(glmnet)
library(tidyverse)
library(lubridate)

#Decile table with glmnet for Actual Return

xD<- lcdfTrn %>% select(-loan_status, -actualTerm, -annRet, -actualReturn, -total_pymnt)
glmRet_cv<- cv.glmnet(data.matrix(xD), lcdfTrn$actualReturn, family="gaussian")
plot(glmRet_cv)
```




```
#plot ( (predict(glmRet_cv, data.matrix(lcdfTrn %>% select(-loan_status, -actualTerm, -annRet, -a
ctualReturn, -total_pymnt)), s="lambda.min" )), lcdfTrn$actualReturn, main = "GLM Actual Return M
odel on Training Data")
#plot ( (predict(glmRet_cv, data.matrix(lcdfTst %>% select(-loan_status, -actualTerm, -annRet, -a
ctualReturn, -total_pymnt)), s="lambda.min" )), lcdfTst$actualReturn, main = "GLM Actual Return M
odel on Test Data")

#On Train Data
predRet_Trn_glm <- lcdfTrn %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate(predRet= predict(glmRet_cv, data.matrix(lcdfTrn %>% select(-loan_status, -actualTerm, -ann
Ret, -actualReturn, -total_pymnt)),s="lambda.min" ))
predRet_Trn_glm <- predRet_Trn_glm %>% mutate(tile=ntile(-predRet, 10))
predRet_Trn_glm %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet), numDefaults
=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(act
ualTerm), totA=sum(grade=="A"),
totB=sum(grade=="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 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  5672     0.0731        1074    0.0727 -0.333  0.367   2.09    57   779
##  2     2  5672     0.0634         916    0.0658 -0.322  0.314   2.11   199  1409
##  3     3  5672     0.0590         924    0.0589 -0.333  0.307   2.19   398  1601
##  4     4  5672     0.0555         873    0.0563 -0.333  0.243   2.17   714  1743
##  5     5  5672     0.0525         850    0.0518 -0.311  0.259   2.21  1074  1809
##  6     6  5671     0.0496         799    0.0488 -0.333  0.225   2.24  1392  1928
##  7     7  5671     0.0467         766    0.0454 -0.323  0.248   2.27  1847  1834
##  8     8  5671     0.0435         720    0.0428 -0.312  0.225   2.28  2313  1842
##  9     9  5671     0.0396         699    0.0386 -0.323  0.210   2.31  2815  1720
## 10    10  5671     0.0324         647    0.0344 -0.323  0.229   2.39  3474  1664
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

```
#on Test Data
predRet_Tst_glm <- lcdfTst %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate(predRet= predict(glmRet_cv, data.matrix(lcdfTst %>% select(-loan_status, -actualTerm, -ann
Ret, -actualReturn, -total_pymnt)),
s="lambda.min" ))

predRet_Tst_glm <- predRet_Tst_glm %>% mutate(tile=ntile(-predRet, 10))

predRet_Tst_glm %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet), numDefaults
=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(act
ualTerm), totA=sum(grade=="A"),
totB=sum(grade=="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 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  2431     0.0733         449    0.0733 -0.322  0.444   2.08    14   345
##  2     2  2431     0.0635         396    0.0667 -0.333  0.334   2.13    71   583
##  3     3  2431     0.0589         386    0.0601 -0.333  0.277   2.14   190   712
##  4     4  2431     0.0555         393    0.0538 -0.322  0.397   2.19   310   785
##  5     5  2431     0.0524         358    0.0516 -0.312  0.262   2.23   441   805
##  6     6  2431     0.0495         374    0.0462 -0.309  0.223   2.24   604   815
##  7     7  2431     0.0467         330    0.0465 -0.312  0.248   2.27   805   783
##  8     8  2430     0.0436         311    0.0416 -0.312  0.238   2.31   993   788
##  9     9  2430     0.0397         273    0.0408 -0.322  0.208   2.31  1204   749
## 10    10  2430     0.0322         289    0.0341 -0.323  0.230   2.37  1487   705
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

We created the decile table for Training and Test data using the glmnet model. Our model is able to distinguish the loans with the highest actualReturn, which are concentrated on the top deciles. As we go down the deciles, the average predicted return decreases.

However, this model does not avoid defaults very well. Even though the Average Actual Return for the top deciles are the highest, there are too many defaults. Combining the Actual Return and Loan Status model is necessary.

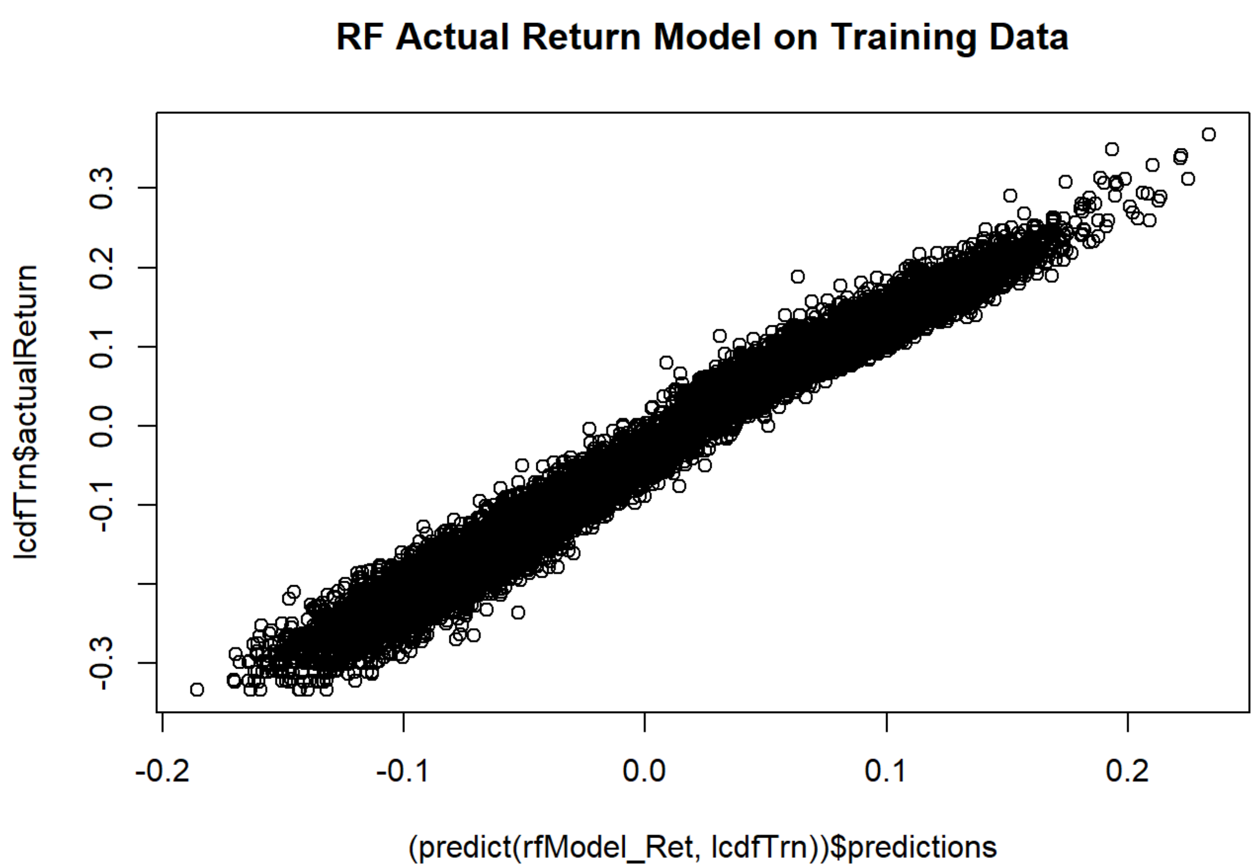
```
library(ranger)


rfModel_Ret <- ranger(actualReturn ~., data=subset(lcdfTrn, select=-c(loan_status, annRet, actual
Term, total_pymnt)), importance = "permutation", num.trees =200)
```

```
## Growing trees.. Progress: 98%. Estimated remaining time: 0 seconds.
## Computing permutation importance.. Progress: 52%. Estimated remaining time: 28 seconds.
```

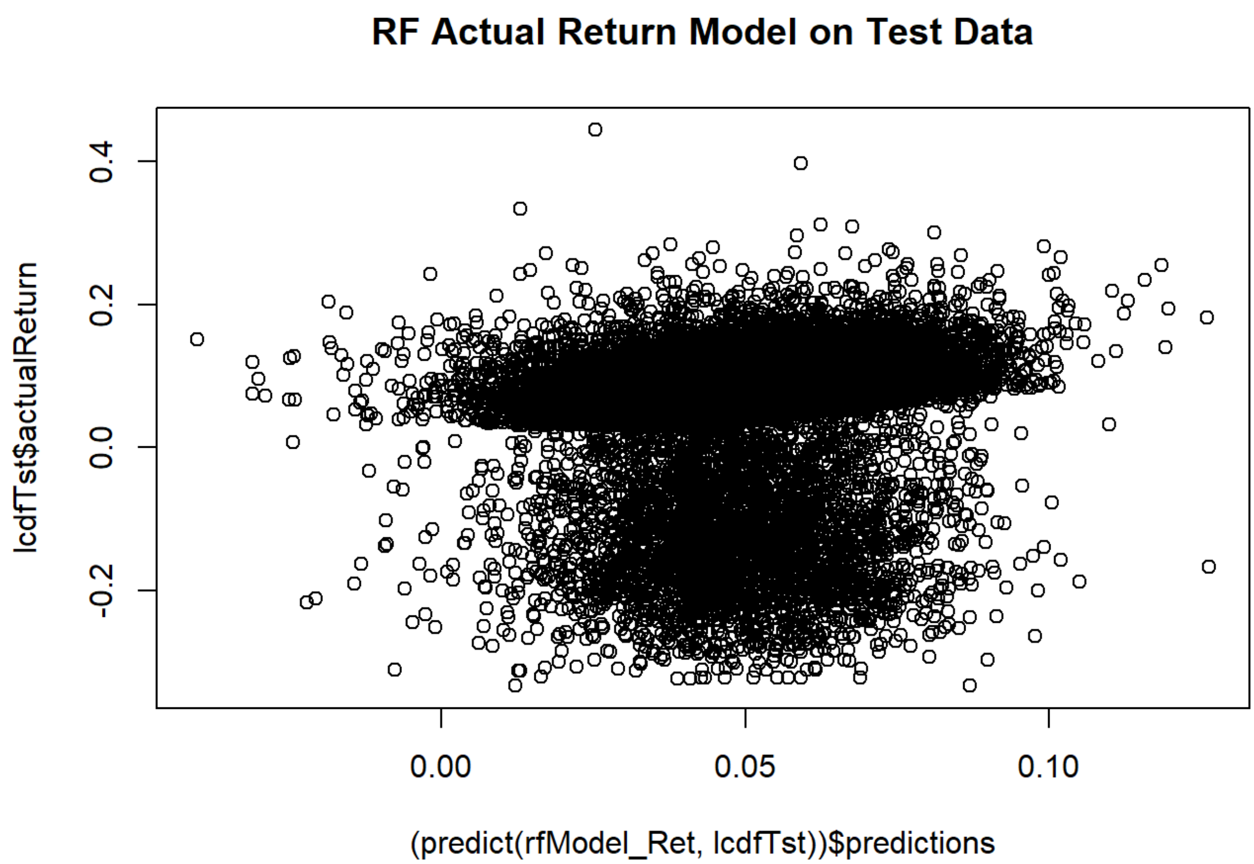
```
rfPredRet_trn<- predict(rfModel_Ret, lcdfTrn)
sqrt(mean( (rfPredRet_trn$predictions - lcdfTrn$actualReturn)^2))
```

```
## [1] 0.03759537
```

```
#sqrt(mean( ( (predict(rfModel_Ret, lcdfTst))$predictions - lcdfTst$actualReturn)^2))
plot ( (predict(rfModel_Ret, lcdfTrn))$predictions, lcdfTrn$actualReturn, main = "RF Actual Retur
n Model on Training Data")
```




```
plot ( (predict(rfModel_Ret, lcdfTst))$predictions, lcdfTst$actualReturn, main = "RF Actual Retur
n Model on Test Data")
```




```
#Performance by deciles on Trn
predRet_Trn_rf <- lcdfTrn %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
 mutate(predRet=(predict(rfModel_Ret, lcdfTrn))$predictions)

predRet_Trn_rf <- predRet_Trn_rf %>% mutate(tile=ntile(-predRet, 10))

predRet_Trn_rf %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet), numDefaults=
sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(act
ualTerm), totA=sum(grade=="A"), totB=sum(grade=="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 14
##     tile count avgpredRet numDefaults avgActRet   minRet   maxRet avgTer  totA
##    <int> <int>      <dbl>       <int>     <dbl>    <dbl>    <dbl>  <dbl> <int>
##  1     1  5672     0.113            2    0.147   0.0905   0.367    0.995     6
##  2     2  5672     0.0881          13    0.110   0.0742   0.180    1.58     10
##  3     3  5672     0.0767          20    0.0932  0.0566   0.177    2.05     34
##  4     4  5672     0.0683          35    0.0815  0.0367   0.157    2.27    218
##  5     5  5672     0.0611          66    0.0725  0.0352   0.188    2.32    739
##  6     6  5671     0.0542          82    0.0644  0        0.121    2.26   1646
##  7     7  5671     0.0471         135    0.0539  0.0113   0.110    2.34   3021
##  8     8  5671     0.0402         177    0.0442 -0.00461  0.102    2.60   4498
##  9     9  5671     0.0226        2067    0.0138 -0.115    0.114    2.85   3566
## 10    10  5671    -0.0702        5671   -0.166  -0.333   -0.00326  3       545
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

```
#Performance by deciles on Tst
predRet_Tst_rf <- lcdfTst %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
 mutate(predRet=(predict(rfModel_Ret, lcdfTst))$predictions)

predRet_Tst_rf <- predRet_Tst_rf %>% mutate(tile=ntile(-predRet, 10))

predRet_Tst_rf %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet), numDefaults=
sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(act
ualTerm), totA=sum(grade=="A"), totB=sum(grade=="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 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  2431     0.0771         410    0.0747 -0.333  0.300   2.05     0   218
##  2     2  2431     0.0646         406    0.0622 -0.312  0.311   2.12     0   726
##  3     3  2431     0.0583         383    0.0587 -0.322  0.397   2.19    15   966
##  4     4  2431     0.0533         338    0.0566 -0.322  0.240   2.21   157  1088
##  5     5  2431     0.0489         348    0.0530 -0.309  0.248   2.20   497   977
##  6     6  2431     0.0450         304    0.0478 -0.322  0.279   2.24   860   807
##  7     7  2431     0.0413         346    0.0393 -0.323  0.265   2.29  1152   627
##  8     8  2430     0.0377         255    0.0448 -0.323  0.284   2.26  1335   497
##  9     9  2430     0.0327         321    0.0394 -0.313  0.271   2.33  1241   572
## 10    10  2430     0.0204         448    0.0380 -0.333  0.444   2.39   862   592
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

The ranger RF model performed well on Train Data, but never quite as good on Test Data. This is a sign of overfit. To avoid overfit, we tried several parameters to tune our ranger model: max.depth = (30, 20, 10, 6, 4, default of NULL), mtry (4, 6, 8), min.node.size = (10, 20, default 5), num.trees = (200, 500, 1000), sample.fraction = 0.5 and default of 1, replace=FALSE, respect.unordered.factors = “order”, verbose = TRUE , seed=0. The problem with our experiment is that none of the changes of parameters lead to a change in the model’s performance on test data.

When we included total_pymnt in building the model, the performance increased significantly for Test Data. However, we are aware that total_pymnt is a leakage variable, therefore we cannot include it in the model.

We created the decile table for Training and Test data using the ranger model. Our model is able to distinguish the loans with the highest actualReturn, which are concentrated on the top deciles. As we go down the deciles, the average predicted return decreases.

However, this model does not avoid defaults very well. Even though the Average Actual Return for the top deciles are the highest, there are too many defaults. Combining the Actual Return and Loan Status model is necessary.

```
library(xgboost)
```

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

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

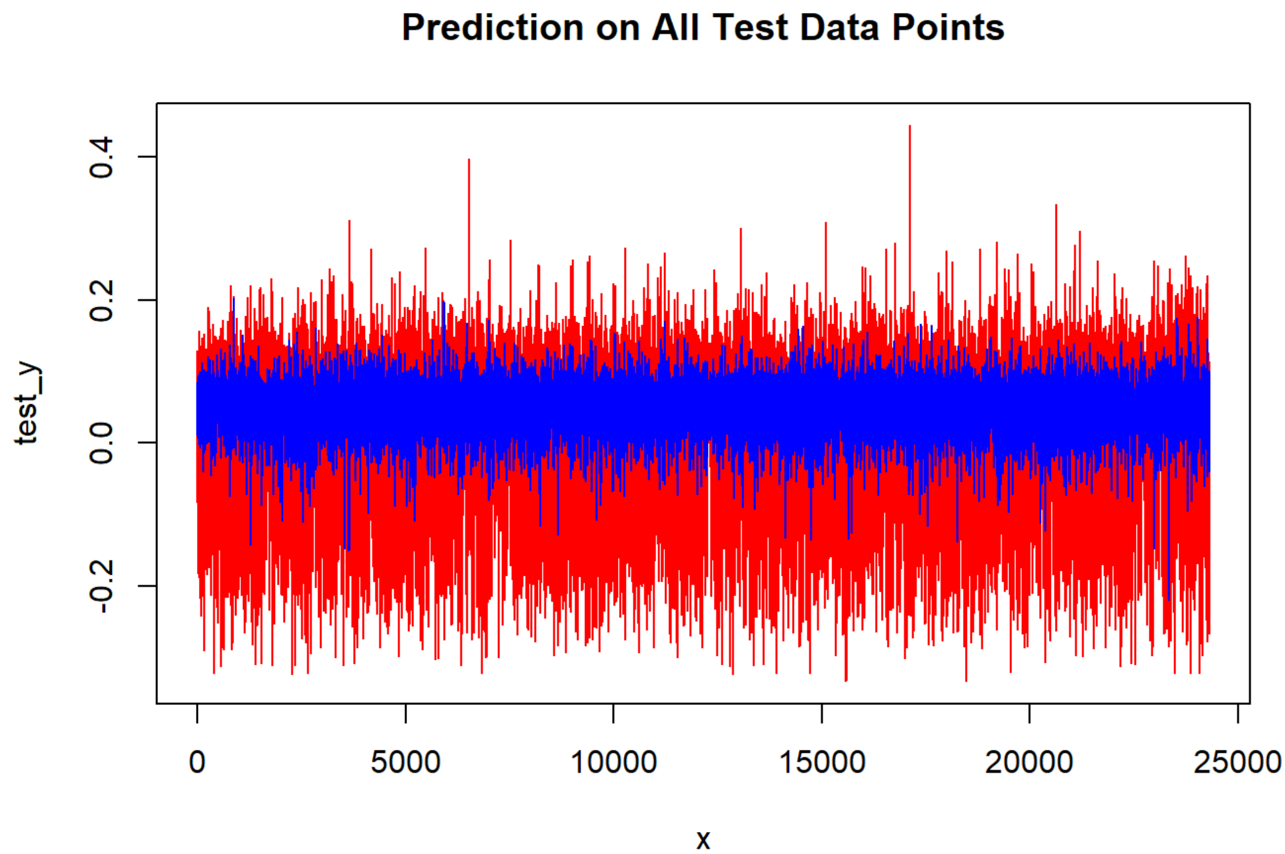
```
set.seed(12)

xDTrn_Ret <-lcdfTrn %>% select(-loan_status, -actualTerm, -annRet, -total_pymnt)
xDTst_Ret <-lcdfTst %>% select(-loan_status, -actualTerm, -annRet, -total_pymnt)


x = grep("actualReturn", colnames(xDTrn_Ret))

train_x = data.matrix(xDTrn_Ret[, -x])
train_y = data.matrix(xDTrn_Ret[,x])

test_x = data.matrix(xDTst_Ret[, -x])
test_y = data.matrix(xDTst_Ret[, x])

test_500_x = test_x[1:500,]
test_500_y = test_y[1:500,]

xgb_train = xgb.DMatrix(data = train_x, label = train_y)
xgb_test = xgb.DMatrix(data = test_x, label = test_y)
xgb_500_test = xgb.DMatrix(data = test_500_x, label = test_500_y)

xgbParam_1 <- list (max.depth = 8, eval_metric="error", eta = 0.1)

#best model
xgb_lsbest_ret <- xgb.train(xgbParam_1, xgb_train, nrounds = 1000)


print(xgb_lsbest_ret)
```

```
## ##### xgb.Booster
## raw: 16.5 Mb
## call:
##   xgb.train(params = xgbParam_1, data = xgb_train, nrounds = 1000)
## params (as set within xgb.train):
##   max_depth = "8", eval_metric = "error", eta = "0.1", validate_parameters = "TRUE"
## xgb.attributes:
##   niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
## # of features: 41
## niter: 1000
## nfeatures : 41
```

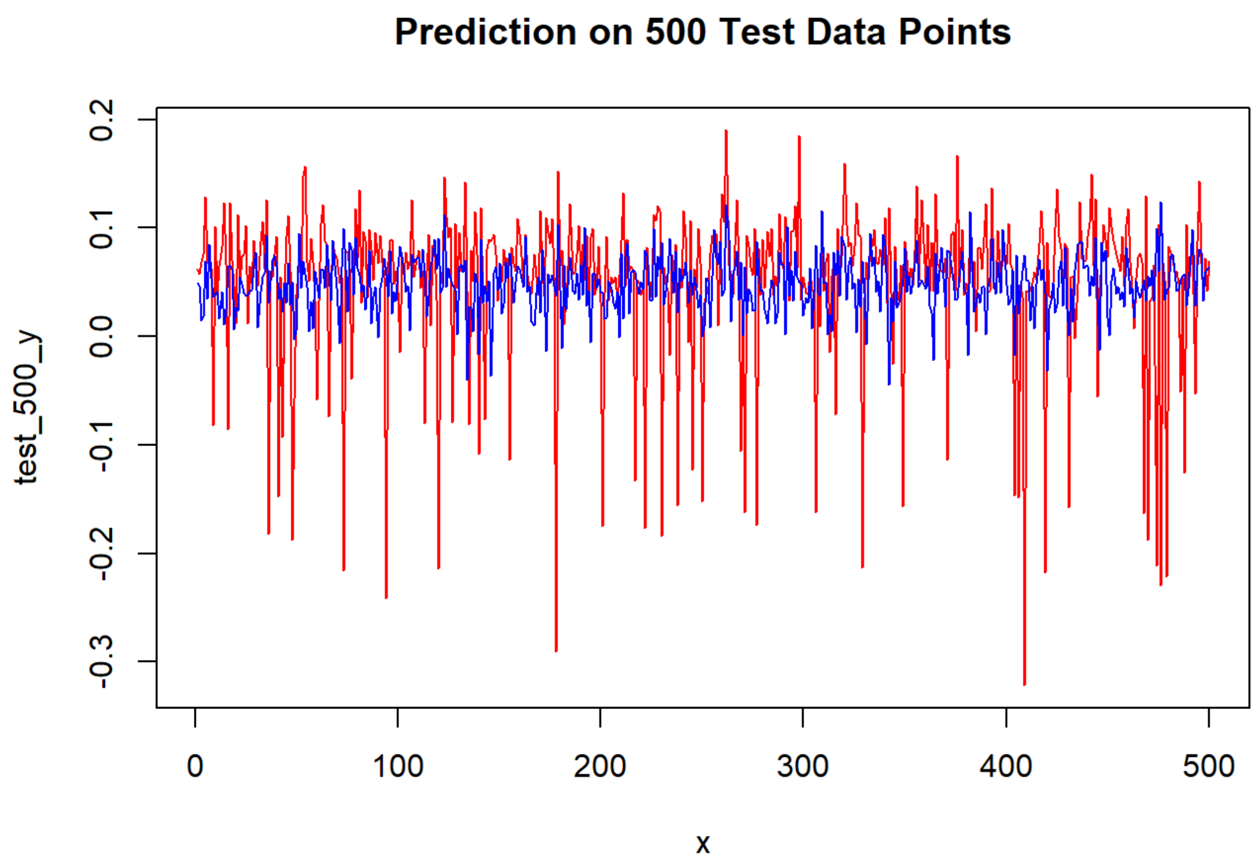
```
pred_y = predict(xgb_lsbest_ret, xgb_test)

mse = mean((test_y - pred_y)^2)
mae = caret::MAE(test_y, pred_y)
rmse = caret::RMSE(test_y, pred_y)

cat("MSE: ", mse, "MAE: ", mae, " RMSE: ", rmse)
```

```
## MSE:  0.007521148 MAE:  0.05448671  RMSE:  0.08672455
```

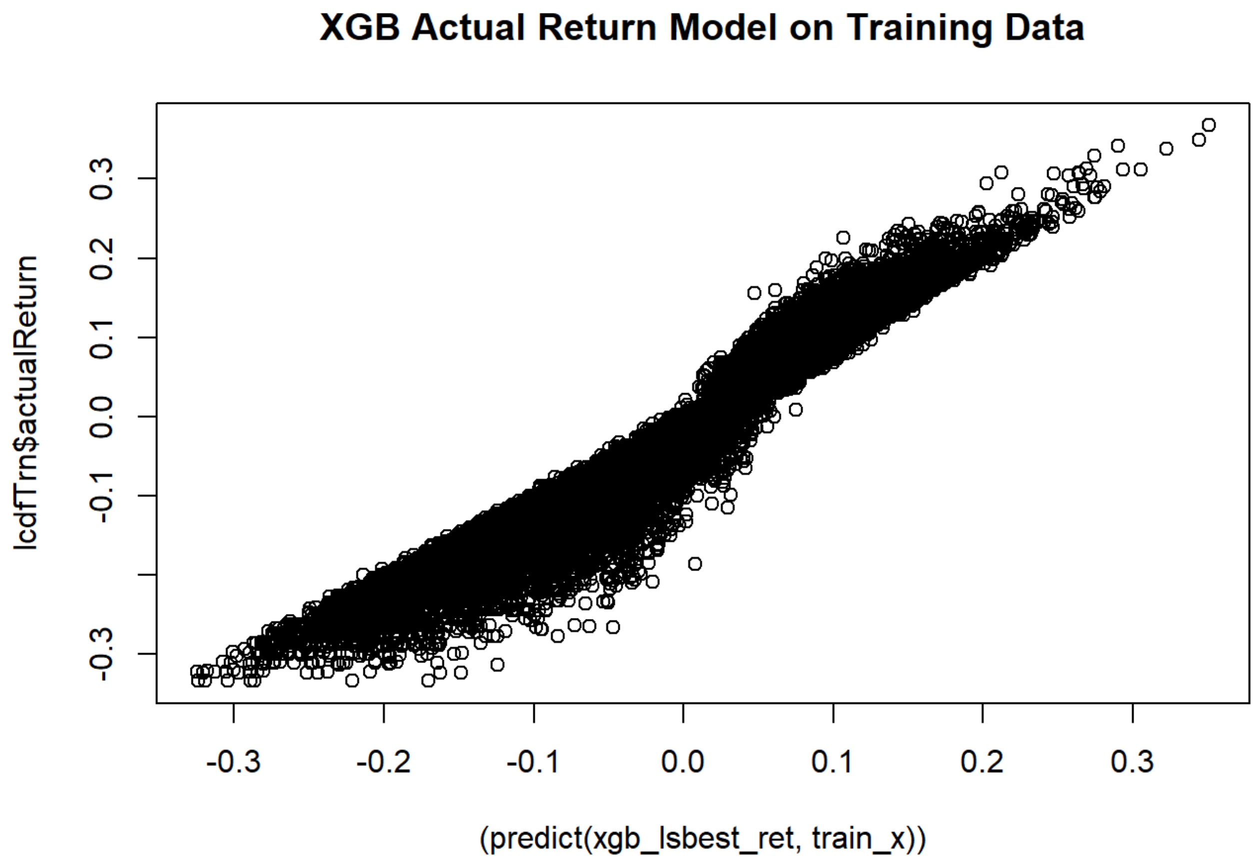
```
x = 1:length(test_y)
plot(x, test_y, col = "red", type = "l", main = "Prediction on All Test Data Points")
lines(x, pred_y, col = "blue", type = "l")
legend(x = 1, y = 38,  legend = c("original test_y", "predicted test_y"), col = c("red", "blue"),
box.lty = 1, cex = 0.8, lty = c(1, 1))
```



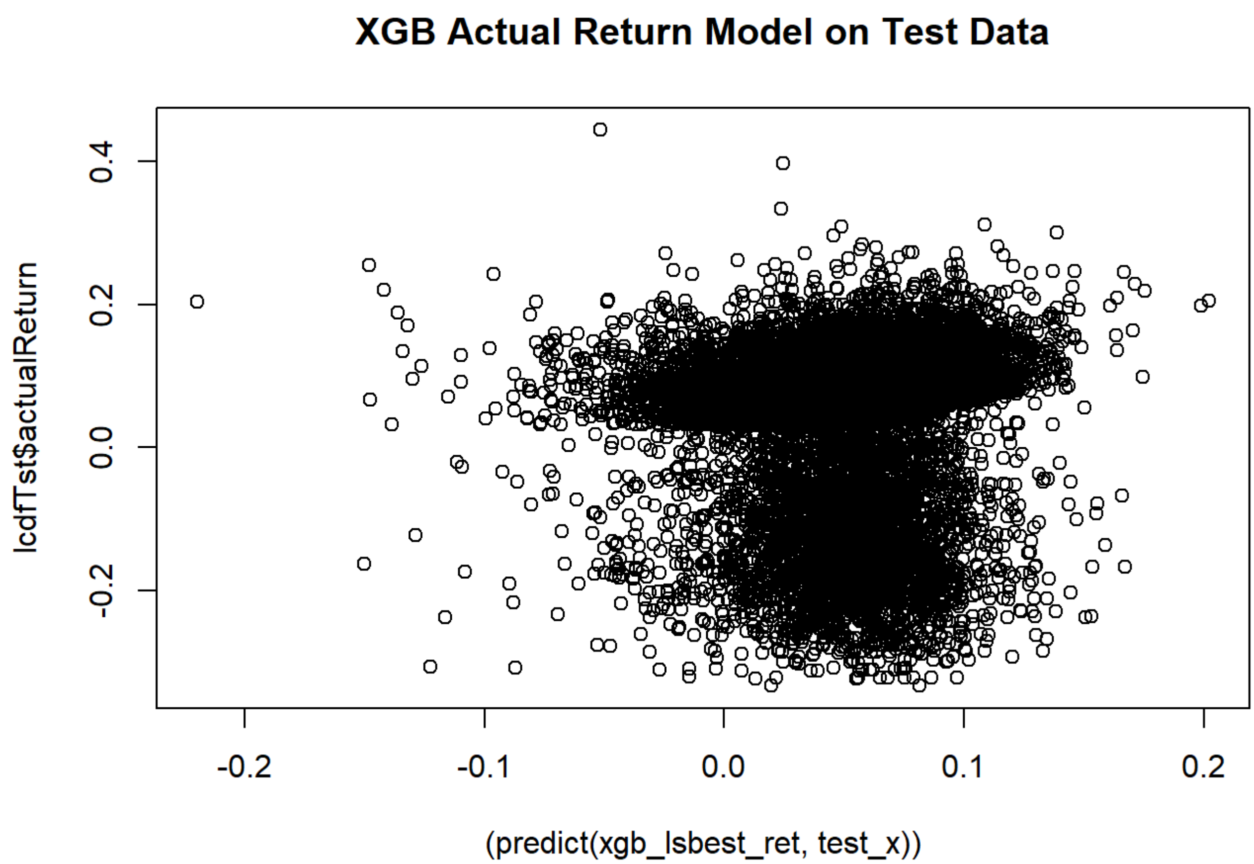

```
pred_500_y = predict(xgb_lsbest_ret, xgb_500_test)

x = 1:length(test_500_y)
plot(x, test_500_y, col = "red", type = "l", main = "Prediction on 500 Test Data Points")
lines(x, pred_500_y, col = "blue", type = "l")
legend(x = 1, y = 38,  legend = c("original test_500_y", "predicted test_500_y"), col = c("red",
"blue"), box.lty = 1, cex = 0.8, lty = c(1, 1))
```




```
plot ( (predict(xgb_lsbest_ret, train_x)), lcdfTrn$actualReturn, main = "XGB Actual Return Model
 on Training Data")
```




```
plot ( (predict(xgb_lsbest_ret, test_x)), lcdfTst$actualReturn, main = "XGB Actual Return Model o
n Test Data")
```




```
#Performance by deciles on Trn
predRet_Trn_xgb <- lcdfTrn %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate(predRet=(predict(xgb_lsbest_ret, train_x)))

predRet_Trn_xgb <- predRet_Trn_xgb %>% mutate(tile=ntile(-predRet, 10))

predRet_Trn_xgb %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet), numDefaults
=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(act
ualTerm), totA=sum(grade=="A"), totB=sum(grade=="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 14
##     tile count avgpredRet numDefaults avgActRet   minRet  maxRet avgTer  totA
##    <int> <int>      <dbl>       <int>     <dbl>    <dbl>   <dbl>  <dbl> <int>
##  1     1  5672     0.133            6    0.145   0.0809   0.367    1.19     3
##  2     2  5672     0.0988          12    0.108   0.0609   0.225    1.75    10
##  3     3  5672     0.0855          38    0.0932  0.0561   0.189    2.06    24
##  4     4  5672     0.0758          58    0.0819  0.00841  0.160    2.24   125
##  5     5  5672     0.0673          81    0.0731  0.0296   0.143    2.31   497
##  6     6  5671     0.0589         105    0.0645 -0.0123   0.160    2.22  1559
##  7     7  5671     0.0507         134    0.0542 -0.0141   0.155    2.28  3301
##  8     8  5671     0.0430         161    0.0453 -0.0652   0.100    2.46  4584
##  9     9  5671     0.0226        2002    0.0137 -0.209    0.0892   2.77  3662
## 10    10  5671    -0.120         5671   -0.164  -0.333   -0.0263   3      518
## # ... with 5 more variables: totB <int>, totC <int>, totD <int>, totE <int>,
## #   totF <int>
```

```
#Performance by deciles on Tst
predRet_Tst_xgb <- lcdfTst %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate(predRet=(predict(xgb_lsbest_ret, test_x)))

predRet_Tst_xgb <- predRet_Tst_xgb %>% mutate(tile=ntile(-predRet, 10))

predRet_Tst_xgb %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet), numDefaults
=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(act
ualTerm), totA=sum(grade=="A"), totB=sum(grade=="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 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  2431    0.0989          464    0.0665 -0.322  0.311   2.13     2   306
##  2     2  2431    0.0776          394    0.0611 -0.333  0.273   2.15    14   755
##  3     3  2431    0.0677          375    0.0584 -0.322  0.262   2.17    98   974
##  4     4  2431    0.0600          349    0.0545 -0.312  0.284   2.19   306  1012
##  5     5  2431    0.0532          311    0.0514 -0.323  0.265   2.21   661   920
##  6     6  2431    0.0470          290    0.0475 -0.308  0.309   2.23  1004   734
##  7     7  2431    0.0409          285    0.0450 -0.313  0.238   2.26  1153   641
##  8     8  2430    0.0339          292    0.0430 -0.312  0.271   2.28  1185   584
##  9     9  2430    0.0232          331    0.0454 -0.333  0.397   2.29   993   596
## 10    10  2430   -0.00624         468    0.0417 -0.323  0.444   2.37   703   548
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

source: https://www.datatechnotes.com/2020/08/regression-example-with-xgboost-in-r.html (https://www.datatechnotes.com/2020/08/regression-example-with-xgboost-in-r.html)

With the XGboost model, we tried the parameters max.depth and eta. 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. When experimenting with all the values and the parameters, we have troubles finding the correct combination to give better results on

Test Data. The change in parameters did not result in notable changes in performance for Test, but it did for Train Data. We proceed with eta = 0.1, nrounds = 1000, max.depth = 6. We did this because we needed to counter the high bias even in Training data predictions. This improved our performance in Train Data, but not Test Data (before this, the model was weak on both).

We created the decile table for Training and Test data using the XGB model. Our model is able to distinguish the loans with the highest actualReturn, which are concentrated on the top deciles. As we go down the deciles, the average predicted return decreases.

However, this model does not avoid defaults very well. Even though the Average Actual Return for the top deciles are the highest, there are too many defaults. Combining the Actual Return and Loan Status model is necessary.

Q3: Combining Loan Status and Actual Return Model

Combining Loan Status and Actual Return models is necessary because: - Loan Status prediction alone only contains loans from the higher grades (A or B), resulting in low average actual return. This makes sense because loans from higher grades are safer from defaults, but have lower interest rates for investors. - Prediction from Actual Return alone only prioritizes the actual returns, but it does a poor job in avoiding defaults. Since the cost of a default could be much higher than any return, we should account for the defaults.

The combined predictions is made by: - Taking the prediction on actual return. - Adding the probability of a fully paid loan for each data point (score from loan status model). - Taking the top decile of this combination, meaning we are taking the loans those are predicted to have the highest returns. - Sorting the top decile by the score from loan status model, from highest probability of a fully paid to the lowest.

This best Loan Status models for RF and XGB were copied from last assignment's best model. Please refer to Q1 for best GLM model on Loan Status.

```
rgModel1 <- ranger(loan_status ~., data=subset(bs_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)
```

```
## Computing permutation importance.. Progress: 97%. Estimated remaining time: 1 seconds.
```

```
library(xgboost)
library(caret)

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_pymnt)),
label=colcdfTrn)
dxTst <- xgb.DMatrix( subset( dxlcdfTst,select=-c(annRet, actualTerm, actualReturn, total_pymn
t)), label=colcdfTst)


#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:0.145766+0.000540   train-auc:0.679204+0.001121 test-error:0.145852+0.004941
test-auc:0.671531+0.011822
## 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:0.145766+0.000540   train-auc:0.680188+0.001523 test-error:0.145834+0.004937
test-auc:0.671795+0.011558
## [3]  train-error:0.145768+0.000536   train-auc:0.680967+0.001263 test-error:0.145834+0.004937
test-auc:0.672477+0.011810
## [4]  train-error:0.145768+0.000536   train-auc:0.681378+0.001238 test-error:0.145834+0.004937
test-auc:0.672844+0.011531
## [5]  train-error:0.145768+0.000536   train-auc:0.681748+0.001490 test-error:0.145834+0.004937
test-auc:0.673147+0.011112
## [6]  train-error:0.145768+0.000536   train-auc:0.682545+0.001473 test-error:0.145834+0.004937
test-auc:0.673779+0.011615
## [7]  train-error:0.145768+0.000536   train-auc:0.683019+0.001231 test-error:0.145834+0.004937
test-auc:0.674246+0.011497
## [8]  train-error:0.145770+0.000531   train-auc:0.683275+0.001214 test-error:0.145799+0.004859
test-auc:0.674474+0.011431
## [9]  train-error:0.145770+0.000531   train-auc:0.684062+0.001167 test-error:0.145799+0.004859
test-auc:0.675091+0.011158
## [10] train-error:0.145770+0.000531   train-auc:0.684271+0.001344 test-error:0.145799+0.004859
test-auc:0.675127+0.011018
```

```
#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)
```

Combining predictions on ranger RF models

```
#Loan Status Model RF

rpredTst<-predict(rgModel1, lcdfTst)
scoreTst_FP <- rpredTst$predictions[,"Fully Paid"]

prPerfRF <- data.frame(scoreTst_FP)
prRetPerfRF <- cbind(prPerfRF, status=lcdfTst$loan_status, grade=lcdfTst$grade, actRet=lcdfTst$ac
tualReturn, actTerm = lcdfTst$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(grad
e=="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          72    0.0408 -0.279  0.134   2.19  2411    20     0
##  2      2  2431         121    0.0432 -0.303  0.140   2.18  2050   377     4
##  3      3  2431         188    0.0490 -0.323  0.309   2.20  1079  1296    52
##  4      4  2431         229    0.0526 -0.322  0.238   2.22   443  1783   184
##  5      5  2431         297    0.0559 -0.333  0.218   2.20   118  1743   499
##  6      6  2431         351    0.0587 -0.312  0.217   2.20    18  1195  1083
##  7      7  2431         417    0.0605 -0.333  0.397   2.22     0   506  1639
##  8      8  2430         509    0.0550 -0.322  0.300   2.26     0   135  1706
##  9      9  2430         599    0.0517 -0.322  0.284   2.29     0    15  1314
## 10     10  2430         776    0.0472 -0.322  0.444   2.32     0     0   287
## # ... with 3 more variables: totD <int>, totE <int>, totF <int>
```

```
# Actual Return Model RF
predrfRet <- predict(rfModel_Ret, lcdfTst)
predrfRet_Tst <- lcdfTst %>% select(grade, loan_status, actualTerm, actualReturn, int_rate) %>%
mutate( predrfRet=predrfRet$predictions)
predrfRet_Tst <- predrfRet_Tst %>% mutate(tile=ntile(-predrfRet, 10))
predrfRet_Tst %>% group_by(tile) %>% summarise(count=n(), avgPredRet=mean(predrfRet),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualRetur
n),
maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="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 14
##     tile count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  2431     0.0771         410    0.0747 -0.333  0.300   2.05     0   218
##  2     2  2431     0.0646         406    0.0622 -0.312  0.311   2.12     0   726
##  3     3  2431     0.0583         383    0.0587 -0.322  0.397   2.19    15   966
##  4     4  2431     0.0533         338    0.0566 -0.322  0.240   2.21   157  1088
##  5     5  2431     0.0489         348    0.0530 -0.309  0.248   2.20   497   977
##  6     6  2431     0.0450         304    0.0478 -0.322  0.279   2.24   860   807
##  7     7  2431     0.0413         346    0.0393 -0.323  0.265   2.29  1152   627
##  8     8  2430     0.0377         255    0.0448 -0.323  0.284   2.26  1335   497
##  9     9  2430     0.0327         321    0.0394 -0.313  0.271   2.33  1241   572
## 10    10  2430     0.0204         448    0.0380 -0.333  0.444   2.39   862   592
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

```
#Combine both models using RF

d=1
pRetSc <- predrfRet_Tst %>% mutate(poScore=scoreTst_FP) #score scoreTst_rf_ls is from predicting
 loan_status. predrfRet_Tst is predicting actual return
pRet_d <- pRetSc %>% filter(tile<=d)
pRet_d<- pRet_d %>% mutate(tile2=ntile(-poScore, 20))
pRet_d %>% group_by(tile2) %>% summarise(count=n(), avgPredRet=mean(predrfRet),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualRetur
n),
maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="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: 20 x 14
##    tile2 count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1   122     0.0747           9    0.0749 -0.234  0.194   1.96     0    80
##  2     2   122     0.0765           9    0.0867 -0.195  0.234   1.94     0    46
##  3     3   122     0.0769           6    0.0927 -0.125  0.217   1.99     0    33
##  4     4   122     0.0775          13    0.0764 -0.218  0.160   2.05     0    19
##  5     5   122     0.0766          12    0.0847 -0.233  0.198   1.81     0    17
##  6     6   122     0.0756          14    0.0757 -0.243  0.199   1.98     0    16
##  7     7   122     0.0770          17    0.0767 -0.277  0.182   2.04     0     4
##  8     8   122     0.0761          11    0.0921 -0.194  0.244   1.90     0     2
##  9     9   122     0.0761          18    0.0703 -0.277  0.244   2.09     0     1
## 10    10   122     0.0768          23    0.0609 -0.333  0.212   2.19     0     0
## 11    11   122     0.0778          24    0.0692 -0.253  0.198   2.22     0     0
## 12    12   121     0.0772          18    0.0849 -0.263  0.246   1.97     0     0
## 13    13   121     0.0768          22    0.0756 -0.218  0.277   2.08     0     0
## 14    14   121     0.0785          23    0.0761 -0.322  0.300   2.10     0     0
## 15    15   121     0.0769          26    0.0678 -0.275  0.251   2.11     0     0
## 16    16   121     0.0769          28    0.0680 -0.258  0.224   2.12     0     0
## 17    17   121     0.0785          28    0.0673 -0.251  0.262   2.05     0     0
## 18    18   121     0.0796          31    0.0750 -0.286  0.273   2.16     0     0
## 19    19   121     0.0778          38    0.0594 -0.310  0.256   2.06     0     0
## 20    20   121     0.0784          40    0.0592 -0.293  0.266   2.19     0     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

We first start by combining the predictions with Loan Status and Actual Returns models, and then we took the top decile from the combined models and created a new table from it. It turns out that the combined model performs better than the two original ones individually.

The model managed to differentiate 2431 loans from the lower grades (mostly C-E) with highest returns (averaged at 7%), much higher than that of the loan status model (4%). However, we need to avoid the abundance of defaults in this top decile from Actual Return prediction.

By further dividing this decile into 20 deciles, and then sorting them in decending order of their loan status prediction score, we can isolate the loans with the highest probability of being paid in the top deciles. The resulting table allows us to identify 122 loans with average predicted return at around 7.4% with only 10 defaults.

Although our model does not boast the highest possible accuracy, we can see that as we go down the deciles at the last table, the number of defaults go up significantly from 10 to 38 at the last deciles.

Combining predictions on XGB Models

```
#Predicting loan status using XGB

xpredTst<-predict(xgb_lsbest, dxTst)

scoreTst_xgb_ls <- lcdfTst %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate(score=xpredTst)

scoreTst_xgb_ls <- scoreTst_xgb_ls %>% mutate(tile=ntile(-score, 10))

scoreTst_xgb_ls %>% group_by(tile) %>% summarise(count=n(), avgSc=mean(score), numDefaults=sum(lo
an_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(act
ualTerm), totA=sum(grade=="A"),
totB=sum(grade=="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 14
##     tile count avgSc numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int> <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  2431 0.544          97    0.0383 -0.313  0.115   2.22  2431     0
##  2     2  2431 0.542         118    0.0429 -0.313  0.142   2.19  2107   321
##  3     3  2431 0.540         196    0.0445 -0.323  0.175   2.25  1307  1087
##  4     4  2431 0.538         262    0.0536 -0.312  0.309   2.19   274  1914
##  5     5  2431 0.536         295    0.0570 -0.333  0.238   2.28     0  1795
##  6     6  2431 0.533         375    0.0516 -0.323  0.248   2.17     0  1800
##  7     7  2431 0.531         441    0.0622 -0.322  0.277   2.13     0   134
##  8     8  2430 0.529         496    0.0567 -0.322  0.397   2.26     0     0
##  9     9  2430 0.527         538    0.0548 -0.333  0.284   2.28     0     1
## 10    10  2430 0.519         741    0.0529 -0.322  0.444   2.32     0    18
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

```
#Predicting Actual Return using XGB

xpredTst_ret<-predict(xgb_lsbest_ret, xgb_test)
predXgbRet_Tst <- lcdfTst %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate( predXgbRet=xpredTst_ret)
predXgbRet_Tst <- predXgbRet_Tst %>% mutate(tile=ntile(-predXgbRet, 10))
predXgbRet_Tst %>% group_by(tile) %>% summarise(count=n(), avgPredRet=mean(predXgbRet),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualRetur
n),
maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="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 14
##     tile count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  2431    0.0989          464    0.0665 -0.322  0.311   2.13     2   306
##  2     2  2431    0.0776          394    0.0611 -0.333  0.273   2.15    14   755
##  3     3  2431    0.0677          375    0.0584 -0.322  0.262   2.17    98   974
##  4     4  2431    0.0600          349    0.0545 -0.312  0.284   2.19   306  1012
##  5     5  2431    0.0532          311    0.0514 -0.323  0.265   2.21   661   920
##  6     6  2431    0.0470          290    0.0475 -0.308  0.309   2.23  1004   734
##  7     7  2431    0.0409          285    0.0450 -0.313  0.238   2.26  1153   641
##  8     8  2430    0.0339          292    0.0430 -0.312  0.271   2.28  1185   584
##  9     9  2430    0.0232          331    0.0454 -0.333  0.397   2.29   993   596
## 10    10  2430   -0.00624         468    0.0417 -0.323  0.444   2.37   703   548
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

```
#Combining both models


d=1
pRetSc_xgb <- predXgbRet_Tst %>% mutate(poScore=scoreTst_xgb_ls$score)
#score scoreTst_xgb_ls is from predicting loan_status. predXgbRet_Tst is predicting actual return

pRet_d_xgb <- pRetSc_xgb %>% filter(tile<=d)
pRet_d_xgb <- pRet_d_xgb %>% mutate(tile2=ntile(-poScore, 20))
pRet_d_xgb %>% group_by(tile2) %>% summarise(count=n(), avgPredRet=mean(xpredTst_ret),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualRetur
n),
maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="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: 20 x 14
##    tile2 count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1   122     0.0496          11    0.0620 -0.300  0.126   2.14     2    95
##  2     2   122     0.0496          10    0.0688 -0.222  0.187   1.97     0   103
##  3     3   122     0.0496          11    0.0839 -0.205  0.234   1.98     0    19
##  4     4   122     0.0496          14    0.0847 -0.194  0.244   2.01     0    20
##  5     5   122     0.0496          14    0.0708 -0.230  0.185   1.94     0    68
##  6     6   122     0.0496          29    0.0555 -0.277  0.212   1.92     0     0
##  7     7   122     0.0496          16    0.0753 -0.266  0.187   2.04     0     0
##  8     8   122     0.0496          16    0.0768 -0.282  0.225   2.00     0     0
##  9     9   122     0.0496          24    0.0693 -0.293  0.225   2.09     0     0
## 10    10   122     0.0496          12    0.0845 -0.215  0.239   2.21     0     0
## 11    11   122     0.0496          25    0.0619 -0.264  0.281   2.12     0     0
## 12    12   121     0.0496          27    0.0535 -0.285  0.210   2.26     0     0
## 13    13   121     0.0496          22    0.0570 -0.276  0.159   2.35     0     0
## 14    14   121     0.0496          19    0.0704 -0.265  0.190   2.18     0     0
## 15    15   121     0.0496          31    0.0617 -0.297  0.271   2.29     0     0
## 16    16   121     0.0496          37    0.0498 -0.273  0.212   2.19     0     0
## 17    17   121     0.0496          35    0.0519 -0.322  0.230   2.18     0     0
## 18    18   121     0.0496          36    0.0591 -0.285  0.311   2.20     0     1
## 19    19   121     0.0496          33    0.0748 -0.321  0.253   2.20     0     0
## 20    20   121     0.0496          42    0.0569 -0.297  0.300   2.26     0     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

We followed the same method to combine the predictions of both XGB models.

The combined XGB models managed to differentiate loans from the lower grades with minimum defaults. The combined models did better than the individual models: - The Loan Status model alone was able to minimize defaults on the top deciles but only has 3.8% average actual return. - The Actual Return model alone managed to gather an average actual return of 6.6% (much higher than the Loan Status prediction), in the expense of an overwhelming amount of defaults (464 loans).

We further disected this top decile into 20 new deciles, and sort them by their probability of being fully paid (score from the loan status model).

The top 122 loans only has 11 defaults with average actual return of 6.2%. The combined deciles allowed us to focus on the top 122 loans with the minimum defaults which present an investment opportunity with comparatively lower risk with good actual returns that are better than what we saw in individual models. As we go down the 20 deciles, we can see that the number of defaults gradually goes up to 42 out of 122 loans. This means that our model has predictive capabilities, despite its shortcomings.

Combining predictions on GLM Models

```
# Loan Status Decile with glm
xDTst<-lcdfTst %>% select(-loan_status, -actualTerm, -annRet, -actualReturn, -total_pymnt)

glmPredls_Tst=predict(glmlsw_cv,data.matrix(xDTst), s="lambda.min", type="response" )

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

scoreTst_glm_ls <- lcdfTst %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate(score=glmPredls_Tst)

scoreTst_glm_ls <- scoreTst_glm_ls %>% mutate(tile=ntile(-score, 10))

scoreTst_glm_ls %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(score), numDefaults=s
um(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(act
ualTerm), totA=sum(grade=="A"),
totB=sum(grade=="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 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  2431      0.809          79    0.0419 -0.323  0.154   2.17  2218   199
##  2     2  2431      0.728         114    0.0463 -0.281  0.200   2.20  1855   537
##  3     3  2431      0.672         182    0.0473 -0.313  0.309   2.23  1252  1068
##  4     4  2431      0.619         245    0.0512 -0.333  0.238   2.19   564  1534
##  5     5  2431      0.567         295    0.0560 -0.322  0.239   2.18   174  1541
##  6     6  2431      0.519         356    0.0572 -0.322  0.244   2.21    47  1160
##  7     7  2431      0.471         436    0.0572 -0.333  0.271   2.22     6   658
##  8     8  2430      0.419         495    0.0554 -0.323  0.277   2.26     2   285
##  9     9  2430      0.357         571    0.0544 -0.322  0.397   2.29     1    86
## 10    10  2430      0.254         786    0.0476 -0.321  0.444   2.34     0     2
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

```
# Actual Return Decile with glm
glmPred_Tst_ret <- predict(glmRet_cv, data.matrix(lcdfTst %>% select(-loan_status, -actualTerm, -
annRet, -actualReturn, -total_pymnt)),s="lambda.min")
predRet_Tst_glm <- lcdfTst %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %>%
mutate(predRet= glmPred_Tst_ret)

predRet_Tst_glm <- predRet_Tst_glm %>% mutate(tile=ntile(-predRet, 10))

predRet_Tst_glm %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(predRet), numDefaults
=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(act
ualTerm), totA=sum(grade=="A"),
totB=sum(grade=="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 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  2431     0.0733         449    0.0733 -0.322  0.444   2.08    14   345
##  2     2  2431     0.0635         396    0.0667 -0.333  0.334   2.13    71   583
##  3     3  2431     0.0589         386    0.0601 -0.333  0.277   2.14   190   712
##  4     4  2431     0.0555         393    0.0538 -0.322  0.397   2.19   310   785
##  5     5  2431     0.0524         358    0.0516 -0.312  0.262   2.23   441   805
##  6     6  2431     0.0495         374    0.0462 -0.309  0.223   2.24   604   815
##  7     7  2431     0.0467         330    0.0465 -0.312  0.248   2.27   805   783
##  8     8  2430     0.0436         311    0.0416 -0.312  0.238   2.31   993   788
##  9     9  2430     0.0397         273    0.0408 -0.322  0.208   2.31  1204   749
## 10    10  2430     0.0322         289    0.0341 -0.323  0.230   2.37  1487   705
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

```
d=1
pRetSc_glm <- predRet_Tst_glm %>% mutate(poScore=scoreTst_glm_ls$score) #score scoreTst_glm_ls is
from predicting loan_status. predRet_Tst_glm is predicting actual return

pRet_d_glm <- pRetSc_glm %>% filter(tile<=d)
pRet_d_glm <- pRet_d_glm %>% mutate(tile2=ntile(-poScore, 20))
pRet_d_glm %>% group_by(tile2) %>% summarise(count=n(), avgPredRet=mean(glmPred_Tst_ret),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualRetur
n),
maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="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: 20 x 14
##    tile2 count avgPredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1   122     0.0515           6    0.0717 -0.210  0.177   1.93    14    80
##  2     2   122     0.0515           7    0.0796 -0.234  0.200   1.91     0    79
##  3     3   122     0.0515           9    0.0699 -0.223  0.152   2.12     0    76
##  4     4   122     0.0515          13    0.0726 -0.276  0.229   2.04     0    60
##  5     5   122     0.0515          14    0.0742 -0.234  0.234   2.00     0    27
##  6     6   122     0.0515          14    0.0830 -0.265  0.207   1.87     0    15
##  7     7   122     0.0515          11    0.0836 -0.322  0.239   1.98     0     6
##  8     8   122     0.0515          23    0.0650 -0.287  0.223   2.12     0     2
##  9     9   122     0.0515          15    0.0835 -0.223  0.188   1.98     0     0
## 10    10   122     0.0515          20    0.0777 -0.230  0.185   2.25     0     0
## 11    11   122     0.0515          30    0.0529 -0.298  0.199   2.13     0     0
## 12    12   121     0.0515          23    0.0762 -0.297  0.271   2.01     0     0
## 13    13   121     0.0515          20    0.0929 -0.297  0.244   2.03     0     0
## 14    14   121     0.0515          24    0.0740 -0.284  0.268   2.15     0     0
## 15    15   121     0.0515          30    0.0743 -0.298  0.246   2.16     0     0
## 16    16   121     0.0515          32    0.0692 -0.293  0.245   2.14     0     0
## 17    17   121     0.0515          33    0.0657 -0.285  0.272   2.20     0     0
## 18    18   121     0.0515          32    0.0816 -0.321  0.272   2.01     0     0
## 19    19   121     0.0515          44    0.0561 -0.284  0.311   2.29     0     0
## 20    20   121     0.0515          49    0.0628 -0.320  0.444   2.31     0     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

We pretty much saw similar trend in case of GLM as what we saw in xgb and Ranger. Model where loan status was target variable we saw low number of defaults(79) but relatively very low actual return(4.2%) whereas the model with Actual return as target variable had higher average actual return of 7.3 percent but it at a price of higher number of defaults (449) which is actually in line with our expectations.

Following the methods that we applied for RF and XGboost, we now combine our GLM models. Below is the results of our individual models: - the top decile of the loan status prediction managed to identify the safest loans, with only 79 defaults out of 2431. However, the average actual return is so only 4.2% - the prediction from our Actual Return model managed to predict the highest average actual return of 7.3%, while failing to separate the defaults from non-defaults (default of 449 loans).

We partitioned the top decile of the Actual Return prediction into 20 new deciles. We also added the score of loan status prediction and sort the new 20 deciles based on this score. The resulting combination decile managed to identify the 2 top deciles (122 loans in each) with 7.1% and 7.9% average actual return with only 6 and 7 defaults each. So far, the glm model has been the best performer for combination predictions.

Q4: Modelling with lower grade loans

Another approach to get the best returns from fully paid loans is by building a model to predict loan status on the lower grade loans. We will compare the results with the combination predictions above.

Modelling from lower loan grades with Ranger

```
lg_lcdfTst<-lcdfTst %>% filter(grade=='C'| grade=='D'| grade== 'E'| grade== 'F'| grade== 'G')
lg_lcdfTrn<-lcdfTrn %>% filter(grade=='C'| grade=='D'| grade== 'E'| grade== 'F'| grade== 'G')

rf_M1_lg <- ranger(loan_status ~., data=subset(lg_lcdfTrn, select=-c(annRet, actualTerm, actualRe
turn)), num.trees =200,
probability=TRUE, importance='permutation')
lg_scoreTstRF <- lg_lcdfTst %>% select(grade, loan_status, actualReturn, actualTerm, int_rate) %
>%
mutate(score=(predict(rf_M1_lg,lg_lcdfTst))$predictions[,"Fully Paid"])

lg_scoreTstRF <- lg_scoreTstRF %>% mutate(tile=ntile(-score, 10))

lg_scoreTstRF %>% group_by(tile) %>% summarise(count=n(), avgSc=mean(score),
numDefaults=sum(loan_status=="Charged Off"), avgActRet=mean(actualReturn), minRet=min(actualRetur
n),
maxRet=max(actualReturn), avgTer=mean(actualTerm), totA=sum(grade=="A"), totB=sum(grade=="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 14
##     tile count avgSc numDefaults avgActRet   minRet  maxRet avgTer  totA  totB
##    <int> <int> <dbl>       <int>     <dbl>    <dbl>   <dbl>  <dbl> <int> <int>
##  1     1  1112 0.960          16    0.0888  1.52e-2  0.217    2.35     0     0
##  2     2  1112 0.935          15    0.0926  1.95e-2  0.185    2.28     0     0
##  3     3  1112 0.917          23    0.0973 -3.58e-5  0.239    2.15     0     0
##  4     4  1112 0.899          39    0.0994 -5.13e-2  0.234    2.18     0     0
##  5     5  1112 0.879          44    0.104  -5.28e-2  0.277    2.04     0     0
##  6     6  1112 0.856          52    0.106  -1.05e-1  0.281    2.02     0     0
##  7     7  1112 0.826          80    0.110  -7.38e-2  0.256    1.96     0     0
##  8     8  1112 0.769         144    0.115  -1.52e-1  0.397    1.75     0     0
##  9     9  1111 0.526         900   -0.0478 -2.89e-1  0.444    2.64     0     0
## 10    10  1111 0.190        1111   -0.189  -3.33e-1 -0.0533   3        0     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

Our RF model on lower grade loans is able to identify the loans which are most likely to be fully paid (15 defaults out of 1112 loans) while maintaining a high average actual return of 8.9%.

```
xDTrn_lg<-lg_lcdfTrn %>% select(-loan_status, -actualTerm, -annRet, -actualReturn, -total_pymnt)
yTrn_lg<-factor(if_else(lg_lcdfTrn$loan_status=="Fully Paid", '1', '0') )
wts_lg <- if_else(yTrn_lg == 0, 1-sum(yTrn_lg == 0)/length(yTrn_lg), 1-sum(yTrn_lg == 1)/length(y
Trn_lg))

glmlsw_cv_lg<- cv.glmnet(data.matrix(xDTrn_lg), yTrn_lg, family= "binomial", weights = wts_lg, al
pha = 1)

xDTst_lg<-lg_lcdfTst %>% select(-loan_status, -actualTerm, -annRet, -actualReturn, -total_pymnt)

glmPredls_Tst_lg=predict(glmlsw_cv_lg,data.matrix(xDTst_lg), s="lambda.min", type="response" )

preds_glm_Tst_lg <- prediction(glmPredls_Tst_lg, lg_lcdfTst$loan_status, label.ordering = c("Char
ged Off", "Fully Paid"))

scoreTst_glm_ls_lg <- lg_lcdfTst %>% select(grade, loan_status, actualReturn, actualTerm, int_rat
e) %>% mutate(score=glmPredls_Tst_lg)

scoreTst_glm_ls_lg <- scoreTst_glm_ls_lg %>% mutate(tile=ntile(-score, 10))

scoreTst_glm_ls_lg %>% group_by(tile) %>% summarise(count=n(), avgpredRet=mean(score), numDefault
s=sum(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(act
ualTerm), totA=sum(grade=="A"),
totB=sum(grade=="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 14
##     tile count avgpredRet numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int>      <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  1112      0.677         125    0.0729 -0.333  0.239   2.07     0     0
##  2     2  1112      0.614         184    0.0617 -0.322  0.244   2.16     0     0
##  3     3  1112      0.582         177    0.0659 -0.333  0.271   2.16     0     0
##  4     4  1112      0.555         197    0.0621 -0.322  0.208   2.21     0     0
##  5     5  1112      0.531         225    0.0581 -0.322  0.271   2.23     0     0
##  6     6  1112      0.506         255    0.0544 -0.322  0.397   2.24     0     0
##  7     7  1112      0.480         245    0.0571 -0.298  0.277   2.26     0     0
##  8     8  1112      0.448         286    0.0530 -0.322  0.284   2.32     0     0
##  9     9  1111      0.409         331    0.0470 -0.310  0.296   2.33     0     0
## 10    10  1111      0.328         399    0.0445 -0.321  0.444   2.37     0     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

Our GLM model on lower grade loans is doing less compared to the our RF model on lower grade loans. However, its top decile has a good average actual return, which is 7.3% with only 125 defaults out of 1112 loans.

```
lg_lcdf<-lcdf %>% filter(grade=='C'| grade=='D'| grade== 'E'| grade== 'F'| grade== 'G')

nr_lg=nrow(lg_lcdf)
trnIndex_lg = sample(1:nr_lg, size = round(0.7*nr_lg), replace=FALSE)
lcdfTrn_lg=lg_lcdf[trnIndex_lg,]
lcdfTst_lg = lg_lcdf[-trnIndex_lg,]

dumVar_lg<-dummyVars(~.,data=lg_lcdf %>% select(-loan_status))
dxlcdf_lg<- predict(dumVar_lg,lg_lcdf)

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

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

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


#Training, test subsets
dxlcdfTrn_lg <- dxlcdf_lg[trnIndex_lg,]
colcdfTrn_lg <- colcdf_lg[trnIndex_lg]
dxlcdfTst_lg <- dxlcdf_lg[-trnIndex_lg,]
colcdfTst_lg <- colcdf_lg[-trnIndex_lg]
dxTrn_lg <- xgb.DMatrix(subset(dxlcdfTrn_lg, select=-c(annRet, actualTerm, actualReturn, total_py
mnt)), label=colcdfTrn_lg)
dxTst_lg <- xgb.DMatrix( subset( dxlcdfTst_lg,select=-c(annRet, actualTerm, actualReturn, total_p
ymnt)), label=colcdfTst_lg)


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

```
## [1]  train-error:0.214359+0.000709   train-auc:0.605757+0.002406 test-error:0.214815+0.005985
test-auc:0.583065+0.011516
## 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:0.214380+0.000688   train-auc:0.606434+0.003375 test-error:0.214738+0.005934
test-auc:0.583626+0.011221
## [3]  train-error:0.214388+0.000678   train-auc:0.608528+0.004824 test-error:0.214777+0.005956
test-auc:0.585515+0.012265
## [4]  train-error:0.214380+0.000688   train-auc:0.609129+0.004719 test-error:0.214738+0.005934
test-auc:0.586059+0.011969
## [5]  train-error:0.214380+0.000688   train-auc:0.609929+0.004977 test-error:0.214738+0.005934
test-auc:0.586818+0.012446
## [6]  train-error:0.214393+0.000672   train-auc:0.611182+0.004533 test-error:0.214623+0.005789
test-auc:0.588042+0.012515
## [7]  train-error:0.214393+0.000672   train-auc:0.612531+0.003918 test-error:0.214623+0.005789
test-auc:0.589305+0.012696
## [8]  train-error:0.214401+0.000681   train-auc:0.613162+0.003844 test-error:0.214623+0.005789
test-auc:0.589874+0.012443
## [9]  train-error:0.214414+0.000694   train-auc:0.614026+0.003905 test-error:0.214623+0.005789
test-auc:0.590258+0.012631
## [10] train-error:0.214431+0.000702   train-auc:0.614627+0.003259 test-error:0.214546+0.005753
test-auc:0.590537+0.013903
```

```
#best iteration
xgb_lscv_lg$best_iteration
```

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

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

#best model
xgb_lsbest_lg <- xgb.train(xgbParam_lg, dxTrn_lg, nrounds = xgb_lscv_lg$best_iteration)

xpredTst_lg<-predict(xgb_lsbest_lg, dxTst_lg)

scoreTst_xgb_ls_lg <- lcdfTst_lg %>% select(grade, loan_status, actualReturn, actualTerm, int_rat
e) %>% mutate(score=xpredTst_lg)

scoreTst_xgb_ls_lg <- scoreTst_xgb_ls_lg %>% mutate(tile=ntile(-score, 10))

scoreTst_xgb_ls_lg %>% group_by(tile) %>% summarise(count=n(), avgSc=mean(score), numDefaults=sum
(loan_status=="Charged Off"),
avgActRet=mean(actualReturn), minRet=min(actualReturn), maxRet=max(actualReturn), avgTer=mean(act
ualTerm), totA=sum(grade=="A"),
totB=sum(grade=="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 14
##     tile count avgSc numDefaults avgActRet minRet maxRet avgTer  totA  totB
##    <int> <int> <dbl>       <int>     <dbl>  <dbl>  <dbl>  <dbl> <int> <int>
##  1     1  1117 0.535         159    0.0709 -0.310  0.294   2.11     0     0
##  2     2  1117 0.533         174    0.0634 -0.300  0.239   2.11     0     0
##  3     3  1117 0.532         180    0.0606 -0.300  0.217   2.24     0     0
##  4     4  1117 0.530         223    0.0536 -0.333  0.214   2.19     0     0
##  5     5  1117 0.529         215    0.0557 -0.293  0.244   2.18     0     0
##  6     6  1117 0.528         265    0.0533 -0.322  0.214   2.25     0     0
##  7     7  1116 0.527         256    0.0548 -0.322  0.268   2.27     0     0
##  8     8  1116 0.522         287    0.0539 -0.320  0.300   2.32     0     0
##  9     9  1116 0.521         311    0.0504 -0.333  0.290   2.29     0     0
## 10    10  1116 0.516         377    0.0523 -0.321  0.444   2.32     0     0
## # ... with 4 more variables: totC <int>, totD <int>, totE <int>, totF <int>
```

Our XGBoost model on lower grade loans is doing less compared to the our RF and GLM models on lower grade loans. However, its top decile has a decent average actual return of 6.6% with only 144 defaults out of 1112 loans. As we go down the deciles, we encounter more defaults.

In comparing RF to RF, GLM to GLM, XGB to XGB:

1. Ranger Model seemed be working exceptionally well lower grade loans data. If we compare the top deciles of the two different Ranger models, one with lower grades loans and the other that contains all the grades: The Lower grade loan data model seems to be working better in all aspects be it default rate or the average actual return. Investing in lower grades loans seems to be a better choice if we go by the ranger model.
2. Comparison of xgb model with lower grade loan data with its counterpart that contains loans from all grade is pretty much in line with our expectations. Lower grade loan model has higher defaults but also higher actual average returns.
3. Comparison of glm model with lower grade loan data with its counterpart that contains loans from all grade is pretty much in line with our expectations. Lower grade loan model has higher defaults but also higher actual average returns.

Profit analysis for best investment approach

Now we will apply all the models through the profit analysis. Suppose we have to decide which top decile is the best to invest on (using what model?), and we are planning to invest $100 in every loan. The cost of investing in a Charged Off loan is $35, as was explained in our previous assignment. The return of a loan will follow the percentage of average actual return for that decile.

We would recommend the following models to use, because of their best profit calculation: 1. Use the RF model on lower grade loans, which boasts the highest profit. 2. Use the combined GLM models, and invest in the top 122 loans.

The complete tabulation for investment approaches using the best models can be found in the table in the Appendix: Best investment approach based on earned profit.

Appendix- Variable Importance for different models

Variables are ranked pretty much similarly in most of the models with GLM being an exception. Top 5 variables are common in Ranger and XGBoost followed by little bit of variation in variable rank after that.
GLMNet does have Grade in top 5 as one of the most imporant variables. But rest of the variables in the GLMNet list are different from other two models. Proportion of Loan Amount to Annual Income is apparantly important
in GLMNet model, this trend is clearly different from what we saw in other models. The effect of Multicollinearity can be clearly seen in GLMNet model as variable grade did not come out to be as important as we expected it to be.

Ranger	Importance factor	Importance NORM
int_rate	0.05	100%
sub_grade	0.04	89%
grade	0.03	60%
dti	0.02	46%
avg_cur_bal	0.02	34%
tot_hi_cred_lim	0.01	31%
acc_open_past_24mths	0.01	25%
annual_inc	0.01	24%
total_bc_limit	0.01	23%
bc_open_to_buy	0.01	23%
installment	0.01	23%
total_rev_hi_lim	0.01	17%
loan_amnt	0.01	14%
bc_util	0.01	14%
mort_acc	0.01	13%
mths_since_recent_inq	0.01	11%
total_bal_ex_mort	0.01	11%
mo_sin_old_rev_tl_op	0.01	11%
emp_length	0.00	11%
mo_sin_rcnt_tl	0.00	11%
num_op_rev_tl	0.00	10%
mths_since_recent_bc	0.00	10%
prop_OpAccts_to_TotAccts	0.00	10%
total_il_high_credit_limit	0.00	10%
home_ownership	0.00	10%
mo_sin_old_il_acct	0.00	9%
num_sats	0.00	8%
num_rev_accts	0.00	8%
num_bc_sats	0.00	7%
mo_sin_rcnt_rev_tl_op	0.00	6%
num_bc_tl	0.00	6%
num_il_tl	0.00	6%
propSatisBankcardAccts	0.00	5%
pct_tl_nvr_dlq	0.00	5%
purpose	0.00	3%
initial_list_status	0.00	1%
tax_liens	0.00	0%
chargeoff_within_12_mths	0.00	0%
delinq_amnt	0.00	0%
num_tl_30dpd	0.00	0%
num_tl_30dpd	0.00	0%

XGBoost	Importance	Importance NORM
int_rate	0.8010646	100%
grade	0.09476281	12%
dti	0.03740318	5%
avg_cur_bal	0.02048943	3%
tot_hi_cred_lim	0.01702559	2%
propLoanAmt_to_AnnInc	0.0165261	2%
acc_open_past_24mths	0.00614935	1%
emp_length.n/a	0.00313037	0%
installment	0.00168584	0%
chargeoff_within_12_mths	0.00114687	0%
annual_inc	0.00061587	0%

GLMNet	Importance	Importance NORM
propLoanAmt_to_AnnInc	61%	100%
num_tl_30dpd	26%	42%
prop_OpAccts_to_TotAccts	19%	32%
grade	11%	18%
home_ownership	7%	11%
propSatisBankcardAccts	7%	11%
int_rate	6%	9%
acc_open_past_24mths	5%	8%
mort_acc	3%	5%
num_bc_tl	3%	4%
num_op_rev_tl	3%	4%
tax_liens	2%	4%
sub_grade	2%	4%
dti	2%	3%
num_rev_accts	1%	2%
initial_list_status	1%	2%
chargeoff_within_12_mths	1%	2%
emp_length	1%	2%
num_bc_sats	1%	1%
num_sats	0%	1%
purpose	0%	1%
pct_tl_nvr_dlq	0%	1%
bc_util	0%	0%
num_il_tl	0%	0%
mths_since_recent_inq	0%	0%
mo_sin_rcnt_tl	0%	0%
mths_since_recent_bc	0%	0%
mo_sin_old_rev_tl_op	0%	0%
installment	0%	0%
mo_sin_old_il_acct	0%	0%
delinq_amnt	0%	0%
mo_sin_rcnt_rev_tl_op	0%	0%
total_bc_limit	0%	0%
bc_open_to_buy	0%	0%
total_il_high_credit_limit	0%	0%
total_rev_hi_lim	0%	0%
total_bal_ex_mort	0%	0%
avg_cur_bal	0%	0%
annual_inc	0%	0%
tot_hi_cred_lim	0%	0%

Best investment approach based on earned profit.

Models	nLoans	defaults	avgActualReturn	Investment	Loss_from_CO	Profit_from_FP	Profit/Loss
Combined RF	122	10	0.07	12200	-350	832.94	482.94
Combined XGB	122	25	0.06	12200	-875	600.33	-274.67
Combined GLM	122	6	0.07	12200	-210	831.95	621.95
Lower Grade RF	1112	15	0.09	111200	-525	9684.04	9159.04
Lower Grade XGB	1112	144	0.07	111200	-5040	6395.12	1355.12
Lower Grade GLM	1112	125	0.07	111200	-4375	7195.63	2820.63