UNIVERSITY OF
ILLINOIS CHICAGO

Assignment 2

IDS 572 – DATA MINING

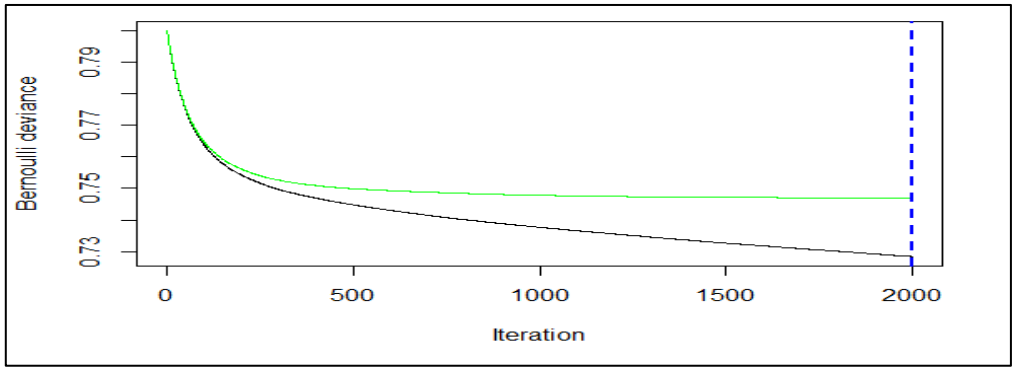
ZOHAIB SHEIKH, SHUBHAM KHODE & ANIRUDHA BALKRISHNA

Q1. Develop boosted tree models (using either gbm or xgBoost) to predict loan_status. Experiment with different parameters using a grid of parameter values. Use cross-validation. Explain the rationale for your experimentation. How does performance vary with parameters, and which parameter setting you use for the 'best' model.

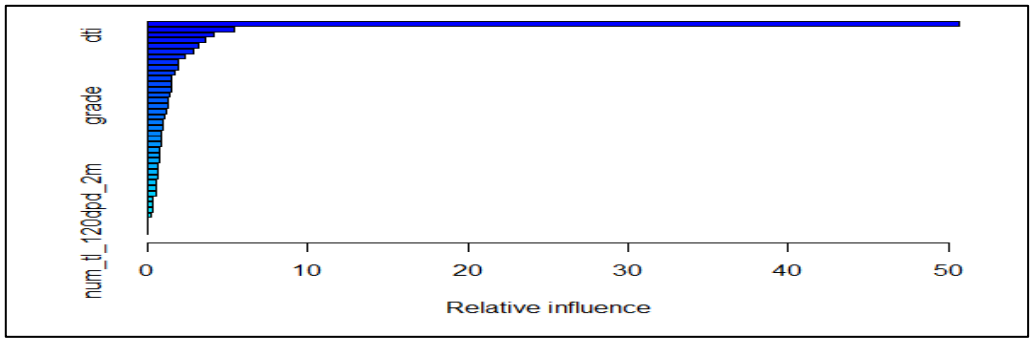
For predicting 'loan status' using tree-based methods, we used gradient boosted model using GBM package in R. The default settings in gbm includes a learning rate (shrinkage) of 0.001. This is a very small learning rate and typically requires a large number of trees to find the minimum MSE. However, gbm uses a default number of trees of 100, which is rarely sufficient. Consequently, we crank it up to 2,000 trees. The default depth of each tree (interaction.depth) is 4, which means we are ensembling a bunch of stumps. Lastly, we also include cv.folds to perform a 5 fold cross validation.

After our initial gradient boosted model was built, we plot the loss function as a result of n trees added to the ensemble.

Model 1 -



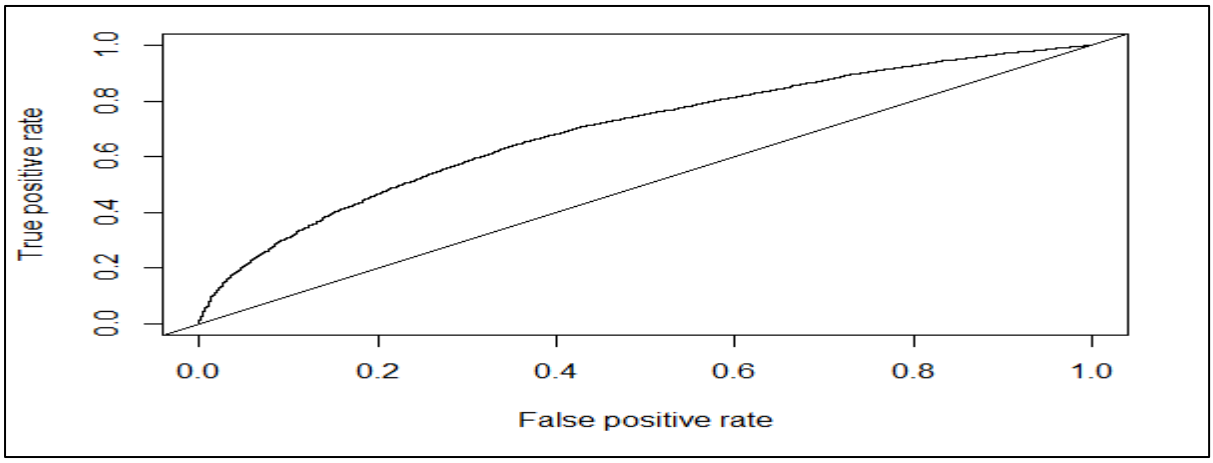
We then looked at the RMSE of the model and the number of trees at which the model converge. For our model, the number of trees where our model converged was at 1991 trees. This proved the fact that we don't need to build more than 2000 trees so as for our model to converge. We also checked the variable importance to check if that is aligned with our initial variable selection. And found out it is very much aligned with the results of variable selection initially.
For ex. Sub-grade, DTI, interest rate are some of the variable with highest influence on the target class.



However, rarely do the default settings suffice. We could tune parameters one at a time to see how the results change. For example, we increased the learning rate to take larger steps down the gradient descent, reduced the number of trees (since we are reducing the learning rate), and increase the depth of each tree from using a single split to 3 splits. This model achieves a significantly lower RMSE than our initial model with only 1,260 trees.
Then we evaluated our model on the unseen data. We used AUC, ROC and the confusion matrix for our evaluation.

For the above model, we observe the AUC to be 66% which points towards better than the no-model scenario.

The ROC curve also shows great lift as below:



	Actual	
Prediction	Charged Off	Fully Paid
Charged Off	19	17
Fully Paid	2712	17225

Given our class of interest is 'Fully-paid', we're doing good as per the confusion matrix-based evaluation metrices.
Specificity : 0.9990140

However, a better option than manually tweaking hyperparameters one at a time is to perform a grid search which iterates over every combination of hyperparameter values and allows us to assess which combination tends to perform well. To perform a manual grid search, first we want to construct our grid of hyperparameter combinations. We're going to search across 6 models with varying learning rates and tree depth. Below is our grid:

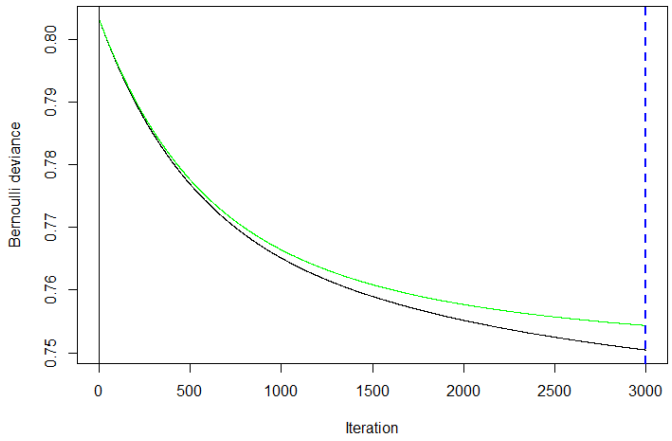
```
shrinkage = c(0.001,.01, .1),
  tree.depth = c(2, 5),
```

After about 30 minutes of training time our grid search ends. We see a few important results pop out. First, our top model has better performance than our previously fitted model above, with lower RMSE.

	shrinkage	tree.depth	optimal_trees	minError
1	0.001	2	3000	0.7511463
2	0.010	2	2652	0.7424064
3	0.100	2	221	0.7439839
4	0.001	5	3000	0.7476569
5	0.010	5	1859	0.7426449
6	0.100	5	102	0.7453248

Model 2 -
One possible option is to build a GBM model with 3000 trees, a shrinkage of 0.001 & interaction depth of 5

When we examine the loss function, the loss is smooth, but the number of trees can be further reduced.

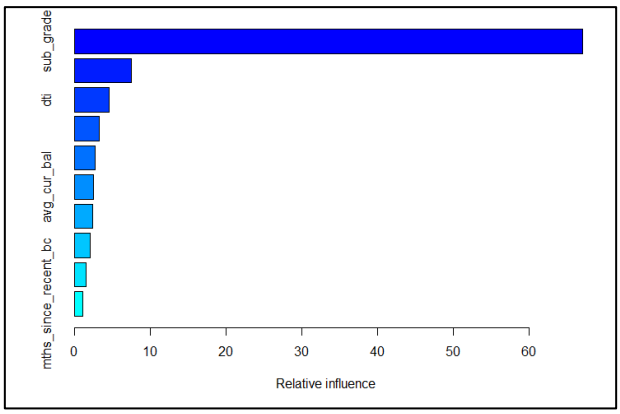


The confusion matrix at a cut-off value of 0.5 is as follows –

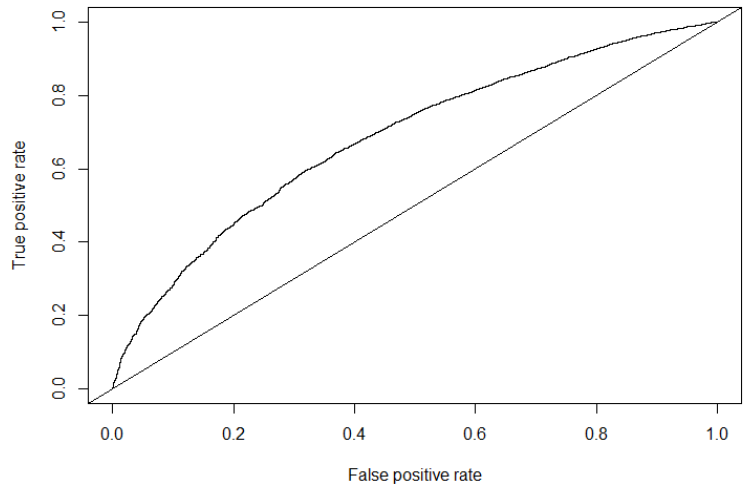
	Actual	
Prediction	Charged Off	Fully Paid
Charged Off	0	1
Fully Paid	2676	17296

As evident, the performance is not better compared to model 1 while predicting default loans. The accuracy is 86% and the specificity remains 99%.

If we look at the relative importance, we see that the variables Sub Grade, Account openings in past 24 months, Debt to income ratio, annual income and home ownership are at the top. The values are as follows -
sub_grade = 67.14561155
acc_open_past_24mths = 7.46705570
dti = 4.50457749
annual_inc = 3.21634609
home_ownership = 2.63679193



The ROC on unseen data is comparable to the first model.



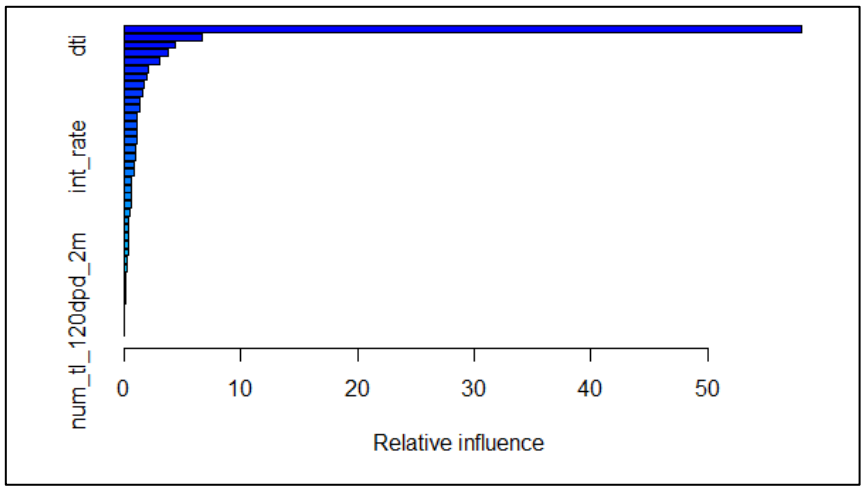
The AUC value is 68%, which is higher than Model 1.

Final Model –

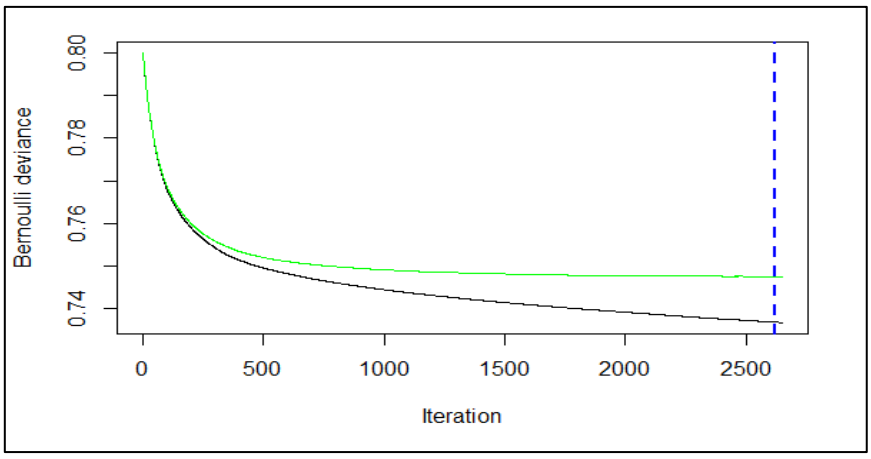
From the above grid, we can see that the best model performance is for tree depth of 2 and shrinkage parameter of 0.01.

We then built our final gradient boosted model using the above parameters which gave us lowest mean squared error.
After building our model, we evaluated our model on the unseen data and looked at various evaluation metrics.

Looking at the relative importance of variables, our top five predictors are
sub_grade -> 72.65975317
acc_open_past_24mths -> 8.22205701
dti -> 4.77541014
annual_inc -> 3.16289313
grade -> 3.02213693



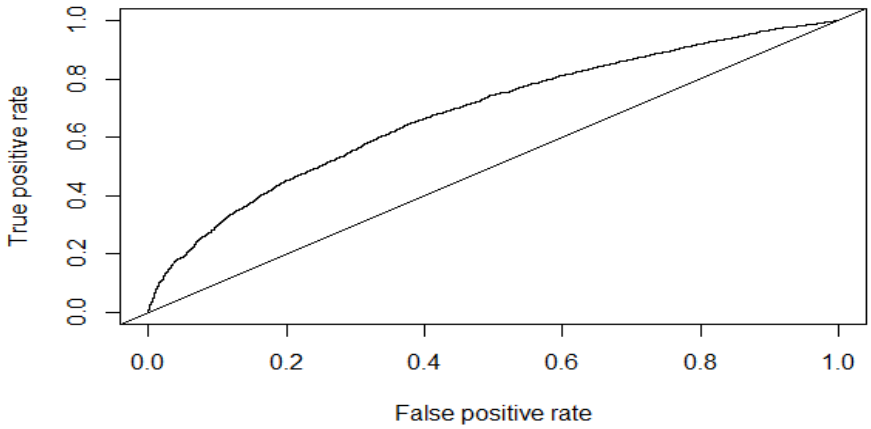
We plot the loss function as a result of n trees added to the ensemble. It is evident from the plot below that the error reduces smoothly as the number of trees grow and converges at 2622 trees.



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

The ROC curve for the unseen data is:

ROC



We also got an AUC value of 70% which is a 4% increase from our initial model (Model 1). And as per the confusion matrix we have a specificity of 100% that means all our actual 'Fully Paid' class are predicted as 'Fully Paid'. Hence, this gradient boosted model performs better than all the different combinations from the hyper grid as well as the initial base model.

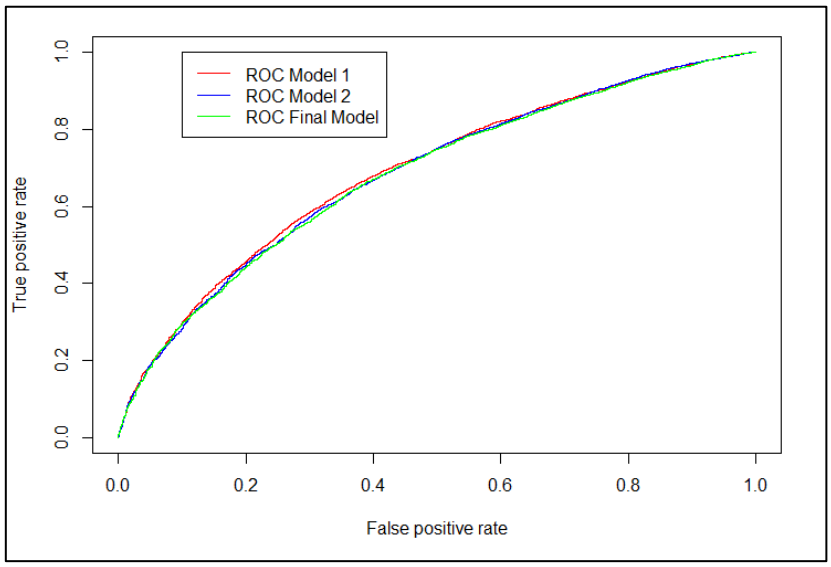
The first Gradient boosted tree gives a profit of $503320.1, the second tree gives a profit of $503367.4 & the final model gives a profit of $503806.6 on the test dataset.

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

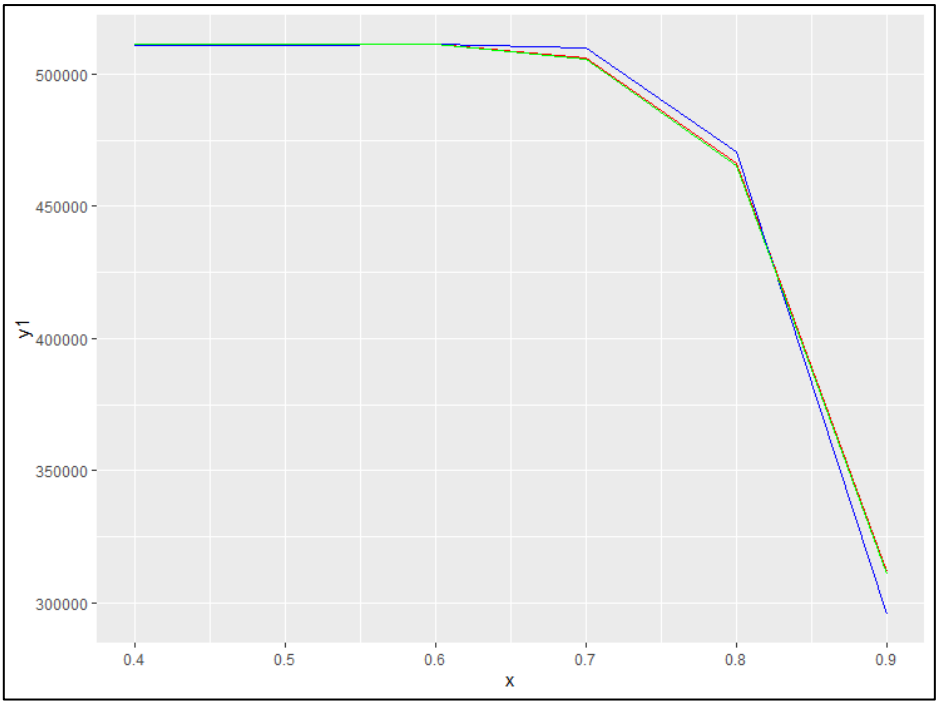
Comparison Table –

Parameter	Model 1	Model 2	Final Model
ntrees	2000	3000	2652
Shrinkage	0.01	0.001	0.01
Interaction Depth	4	5	2
Best CV iteration	1982	3000	2652
Non Zero predictors	37	36	37
Accuracy (Cutoff of 0.5)	86.3%	86.5%	87%
AUC	66%	68%	70%
Profit (Cutoff of 0.5)	503320.1	503367.4	503806.6

Combined ROC plot – As evident below, if we plot all the curves, the models perform quite similarly on unseen data but, the final model is marginally better when compared to model 1 & model 2. Model 1 performs better at the start but as evident from AUC analysis, the final model has 4% more area under the curve.



Profit Curve - The profit curves at different cut-off values are shown in the graph below. All models perform best at a cut-off of 0.5. Among the three models, the final model gives a higher profit. At lower cut-off values, the models classify all loans as Fully paid. Therefore, the analysis starts at a threshold of "0.4". The Red line indicates Model 1, blue line is Model 2 & the green line is the Final Model.



Accuracy values

Cutoff	Model 1	Model 2	Final Model
0.4	83.3%	83.2%	83.5%
0.5	86.3%	86.5%	87.0%
0.6	86.1%	86.1%	86.6%
0.7	85.7%	86.0%	85.8%
0.8	76.0%	77.6%	76.5%
0.9	44.4%	38.0%	40.4%

2. (a) Develop linear (glm) models to predict loan_status. Experiment with different parameter values, and identify which gives 'best' performance. Use cross-validation. Describe how you determine 'best' performance. How do you handle variable selection? Experiment with Ridge and Lasso, and show how you vary these parameters, and what performance is observed

For developing linear model to predict 'loan_status', we have split our cleaned dataset randomly into two sets: Training set and Testing set. We trained our models on the training data using different combinations of model parameters and evaluated the models using AUC and ROC measures and verified it on test set. The split used here as follows:

Training set – 80% | Test Set – 20%

Training set – 79890 records | Test Set – 19972 records

We have excluded any leakage variables while performing data cleaning. Our final cleaned dataset that has been used for building the model has 59 variables with 99862 data points.

Model 1 – All in

We built our first linear model using 'glm' with all available predictors. The summary of call is as follows,

glmBasic <- glm(formula = y_train_data ~ ., data = x_train_data, family = "binomial")

```
Call:
glm(formula = y_train_data ~ ., family = "binomial", data = x_train_data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.1399   0.3090   0.4270   0.5499   1.6676

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)              2.215e+01  5.658e+01   0.391 0.695438
loan_amnt                3.262e-05  2.558e-05   1.275 0.202242
int_rate                 2.101e-02  1.375e-02   1.528 0.126599
installment             -1.412e-03  7.419e-04  -1.904 0.056956 .
grade                    3.210e-02  3.960e-02   0.811 0.417550
sub_grade               -8.556e-02  1.232e-02  -6.946 3.76e-12 ***
home_ownership          -1.021e-01  1.522e-02  -6.706 2.00e-11 ***
annual_inc               1.276e-06  4.329e-07   2.947 0.003207 **
verification_status     -6.164e-02  1.502e-02  -4.103 4.08e-05 ***
purpose                 -5.692e-03  5.739e-03  -0.992 0.321316
addr_state               9.875e-04  7.826e-04   1.262 0.207031
dti                     -1.904e-02  1.771e-03 -10.749  < 2e-16 ***
delinq_2yrs             -9.623e-02  1.754e-02  -5.487 4.08e-08 ***
inq_last_6mths          -2.963e-02  1.381e-02  -2.146 0.031886 *
open_acc                -9.387e-03  4.160e-02  -0.226 0.821485
pub_rec                 -6.745e-02  4.532e-02  -1.488 0.136695
revol_bal                2.808e-06  2.747e-06   1.022 0.306655
revol_util              -1.515e-04  9.952e-04  -0.152 0.878966
total_acc               -3.126e-03  1.753e-02  -0.178 0.858507
initial_list_status     -2.025e-02  2.306e-02  -0.878 0.379903
application_type         1.137e+00  7.473e-01   1.521 0.128205
tot_cur_bal              4.631e-07  8.639e-07   0.536 0.591900
total_rev_hi_lim         1.914e-06  2.362e-06   0.810 0.417938
acc_open_past_24mths    -6.038e-02  5.905e-03 -10.225  < 2e-16 ***
avg_cur_bal              1.061e-06  2.168e-06   0.489 0.624576
bc_open_to_buy           4.336e-06  3.478e-06   1.247 0.212486
bc_util                  2.761e-03  1.023e-03   2.698 0.006971 **
chargeoff_within_12_mths -4.243e-02  9.553e-02  -0.444 0.656947
delinq_amnt              8.725e-06  2.132e-05   0.409 0.682353
mo_sin_old_il_acct      -2.534e-04  2.390e-04  -1.060 0.289106
mo_sin_old_rev_tl_op     2.818e-04  1.451e-04   1.942 0.052189 .
mo_sin_rcnt_rev_tl_op   -1.869e-04  1.180e-03  -0.158 0.874159
mo_sin_rcnt_tl           4.828e-03  1.952e-03   2.473 0.013381 *
mort_acc                 2.279e-02  1.901e-02   1.199 0.230632
```

```
mths_since_recent_bc         1.785e-03  5.423e-04   3.291 0.000998 ***
mths_since_recent_inq        7.166e-03  2.535e-03   2.827 0.004693 **
num_accts_ever_120_pd       -5.807e-03  1.116e-02  -0.520 0.602828
num_actv_bc_tl               9.606e-03  1.433e-02   0.670 0.502744
num_actv_rev_tl              4.251e-02  2.224e-02   1.911 0.055966 .
num_bc_sats                 -1.798e-02  1.069e-02  -1.681 0.092804 .
num_bc_tl                   -2.489e-02  6.704e-03  -3.713 0.000205 ***
num_il_tl                    9.596e-03  1.756e-02   0.546 0.584724
num_op_rev_tl               -3.880e-03  9.523e-03  -0.407 0.683715
num_rev_accts                2.539e-02  1.804e-02   1.408 0.159276
num_rev_tl_bal_gt_0         -6.444e-02  2.307e-02  -2.794 0.005210 **
num_sats                     1.582e-02  4.168e-02   0.380 0.704305
num_tl_120dpd_2m             3.632e-01  4.174e-01   0.870 0.384186
num_tl_30dpd                -9.259e-02  1.546e-01  -0.599 0.549239
num_tl_90g_dpd_24m           4.697e-02  2.877e-02   1.632 0.102608
num_tl_op_past_12m           5.876e-03  9.734e-03   0.604 0.546045
pct_tl_nvr_dlq              -7.368e-04  1.749e-03  -0.421 0.673555
percent_bc_gt_75            -2.490e-03  5.824e-04  -4.275 1.91e-05 ***
pub_rec_bankruptcies        -7.843e-03  5.297e-02  -0.148 0.882298
tax_liens                    8.723e-02  5.475e-02   1.593 0.111112
tot_hi_cred_lim              6.159e-08  7.879e-07   0.078 0.937689
total_bal_ex_mort           -7.396e-06  1.257e-06  -5.882 4.05e-09 ***
total_bc_limit               5.922e-06  1.966e-06   3.013 0.002589 **
total_il_high_credit_limit   7.719e-06  1.284e-06   6.013 1.82e-09 ***
debt_settlement_flag        -2.001e+01  5.657e+01  -0.354 0.723542
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 63921  on 79889  degrees of freedom
Residual deviance: 54621  on 79831  degrees of freedom
AIC: 54739

Number of Fisher Scoring iterations: 15
```

As we can see from the summary, there are lot of predictors with p-values high enough that make them statistically insignificant predictors for our model. Thus, our next step would be remove such variables and rerun our model selected subset of predictors.

Variable Selection – Stepwise Backward Elimination

We run stepwise backward elimination using ‘step’, starting from our base model with all the variables selected, and removing the least significant one at each step, until none meet the criterion (here AIC statistic).

stepBckElim = step(glmBasic, direction = "backward", trace = 0)

After going through multiple iterations, the final selected model by step() has following formula,

formula(stepBckElim)

```
y_train_data ~ installment + sub_grade + home_ownership + annual_inc +
    verification_status + dti + delinq_2yrs + inq_last_6mths +
    pub_rec + revol_bal + application_type + tot_cur_bal + acc_open_past_24mths +
    bc_open_to_buy + bc_util + mo_sin_old_rev_tl_op + mo_sin_rcnt_tl +
    mort_acc + mths_since_recent_bc + mths_since_recent_inq +
    num_actv_rev_tl + num_bc_sats + num_bc_tl + num_il_tl + num_rev_accts +
    num_rev_tl_bal_gt_0 + num_tl_90g_dpd_24m + percent_bc_gt_75 +
    tax_liens + total_bal_ex_mort + total_bc_limit + total_il_high_credit_limit +
    debt_settlement_flag
```

The process selected only 34 predictors from initial list of 58 predictors. We can find the predictors that were eliminated by calling the anova attribute on last model as,

stepBckElim$anova

```
                         Step Df    Deviance Resid. Df Resid. Dev      AIC
1                            NA          NA     79831   54620.92 54738.92
2           - tot_hi_cred_lim  1 0.006121874     79832   54620.93 54736.93
3       - pub_rec_bankruptcies  1 0.022726780     79833   54620.95 54734.95
4                - revol_util  1 0.021667617     79834   54620.97 54732.97
5     - mo_sin_rcnt_rev_tl_op  1 0.027048918     79835   54621.00 54731.00
6                  - total_acc  1 0.031070463     79836   54621.03 54729.03
7                   - open_acc  1 0.054890366     79837   54621.09 54727.09
8              - num_op_rev_tl  1 0.138045455     79838   54621.22 54725.22
9                - delinq_amnt  1 0.179998899     79839   54621.40 54723.40
10 - chargeoff_within_12_mths  1 0.200381427     79840   54621.60 54721.60
11               - avg_cur_bal  1 0.215244691     79841   54621.82 54719.82
12           - pct_tl_nvr_dlq  1 0.229121896     79842   54622.05 54718.05
13    - num_accts_ever_120_pd  1 0.142127287     79843   54622.19 54716.19
14       - num_tl_op_past_12m  1 0.393244715     79844   54622.58 54714.58
15           - num_actv_bc_tl  1 0.430390378     79845   54623.01 54713.01
16             - num_tl_30dpd  1 0.438658326     79846   54623.45 54711.45
17                     - grade  1 0.661242041     79847   54624.11 54710.11
18      - initial_list_status  1 0.829375993     79848   54624.94 54708.94
19         - total_rev_hi_lim  1 0.855224026     79849   54625.80 54707.80
20                   - purpose  1 0.969428671     79850   54626.77 54706.77
21         - num_tl_120dpd_2m  1 1.116824344     79851   54627.88 54705.88
22       - mo_sin_old_il_acct  1 1.133310842     79852   54629.02 54705.02
23                - loan_amnt  1 1.523098729     79853   54630.54 54704.54
24               - addr_state  1 1.535667431     79854   54632.08 54704.08
25                 - num_sats  1 1.559314702     79855   54633.64 54703.64
26                 - int_rate  1 1.773257992     79856   54635.41 54703.41
```

We further eliminate the predictors with p-values > 0.1 from our selected subset 34 predictors, leaving us with a final subset of **30 variables selected**.

Model 2 – by using variables selected by 'Step-wise Backward Elimination'

Now we build our second model based on new subset of variables that we selected in previous stepwise elimination method.

glmTuned <- glm(formula = y_train_data ~ ., data = x_train_data, family = "binomial")

We looked at how our model perform on train and test datasets. We have identified 'Non-Default' as our class of interest. Our evaluation results are as follows,

Confusion Matrix:

Train Data:

	Predicted	
Actual	**Default**	**Non-Default**
Default	89	10881
Non-Default	93	68827

Accuracy = 0.8626

Test Data:

	Predicted	
Actual	**Default**	**Non-Default**
Default	17	2725
Non-Default	20	17210

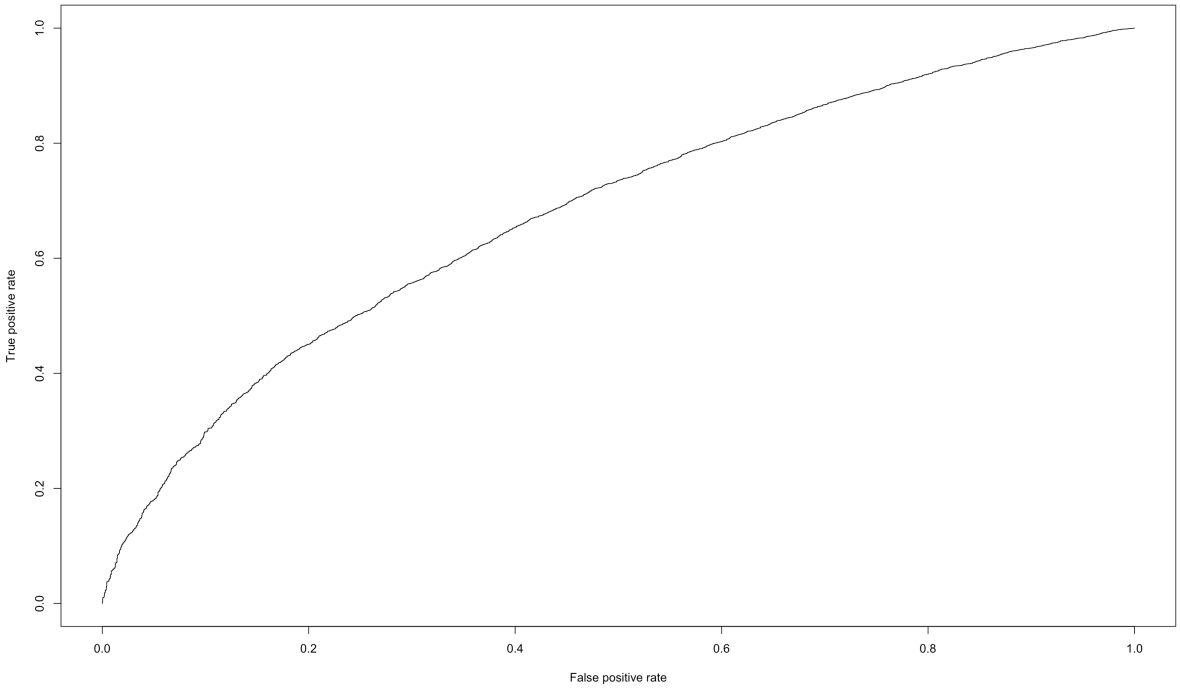
Accuracy = 0.8625

AUC:

prediction(predTunedModel, y_test_data) %>% performance(measure = "auc") %>% .@y.values

AUC = 0.6788571

ROC:



Takeaway:

As per the confusion matrix above, we can deduct that the model is giving overall decent accuracy, however the performance can still be improved on misclassification of "Default" as "Non-Default".

Model 3 – Ridge Regularization

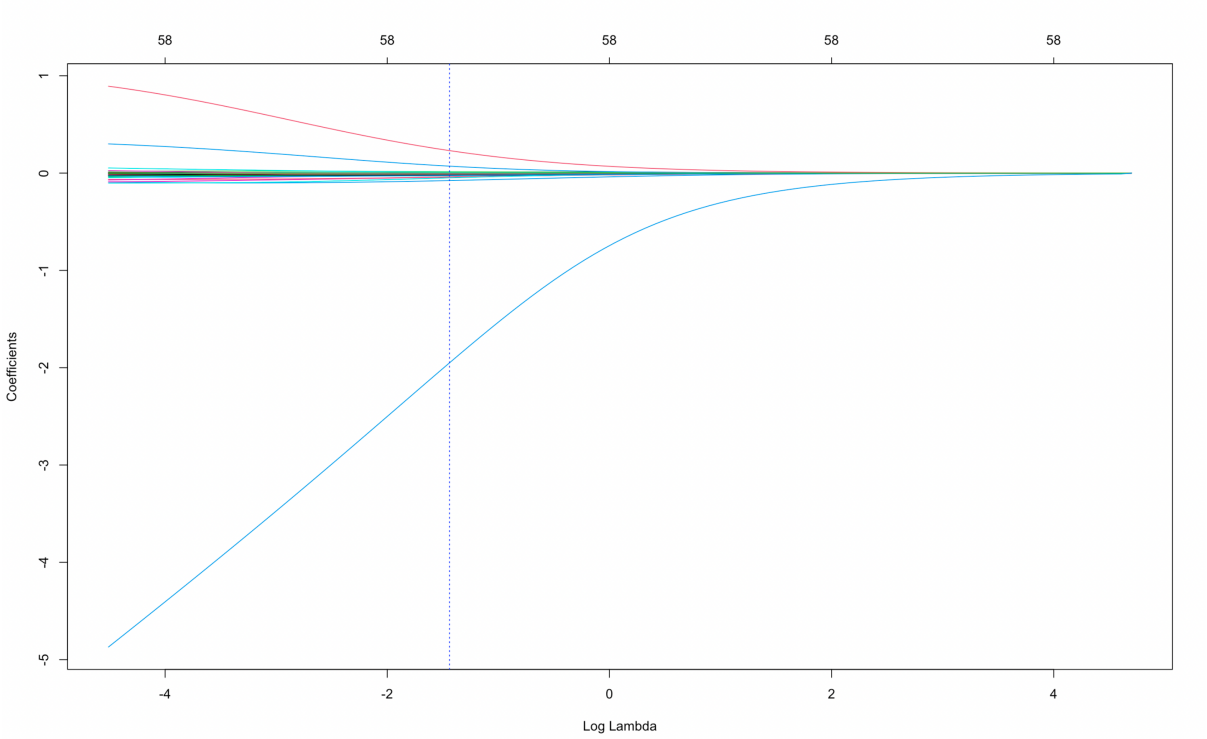
We now implement our third model using 'cv.glmnet' that does k-fold cross-validation for 'glmnet' and returns values of lambda used in the fits. To control the coefficients by adding "L2" penalty, we set alpha = 1.

glmRidgeReg <- cv.glmnet(data.matrix(x_train_data), y_train_data, family = "binomial", alpha = 0, type.measure="auc")

To see the effect of lamba on coefficients of fit, we can get a plot as follows,

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

abline(v = log(glmRidgeReg$lambda.1se), col = "blue", lty = "dotted")



Here, we observe that as λ=0 there is no effect and our objective function equals the normal OLS regression objective function. However, as λ→∞, the penalty becomes large and forces our coefficients to go closer and closer to zero. The blue vertical line depicts the λ at '1se'. On top plot we see the number of corresponding predictors retained after L2 regularization at that λ value, which in our case is 58 as expected.

Additionally, we can get the AUC values directly at minimum and 1 std deviation of lambda from 'cvm' attribute (since we used type.measure="auc" while training our model).

> glmRidgeReg$cvm[glmRidgeReg$lambda == glmRidgeReg$lambda.min]

[1] 0.7268062

> glmRidgeReg$cvm[glmRidgeReg$lambda == glmRidgeReg$lambda.1se]

[1] 0.7242639

Model 4 – Lasso Regularization

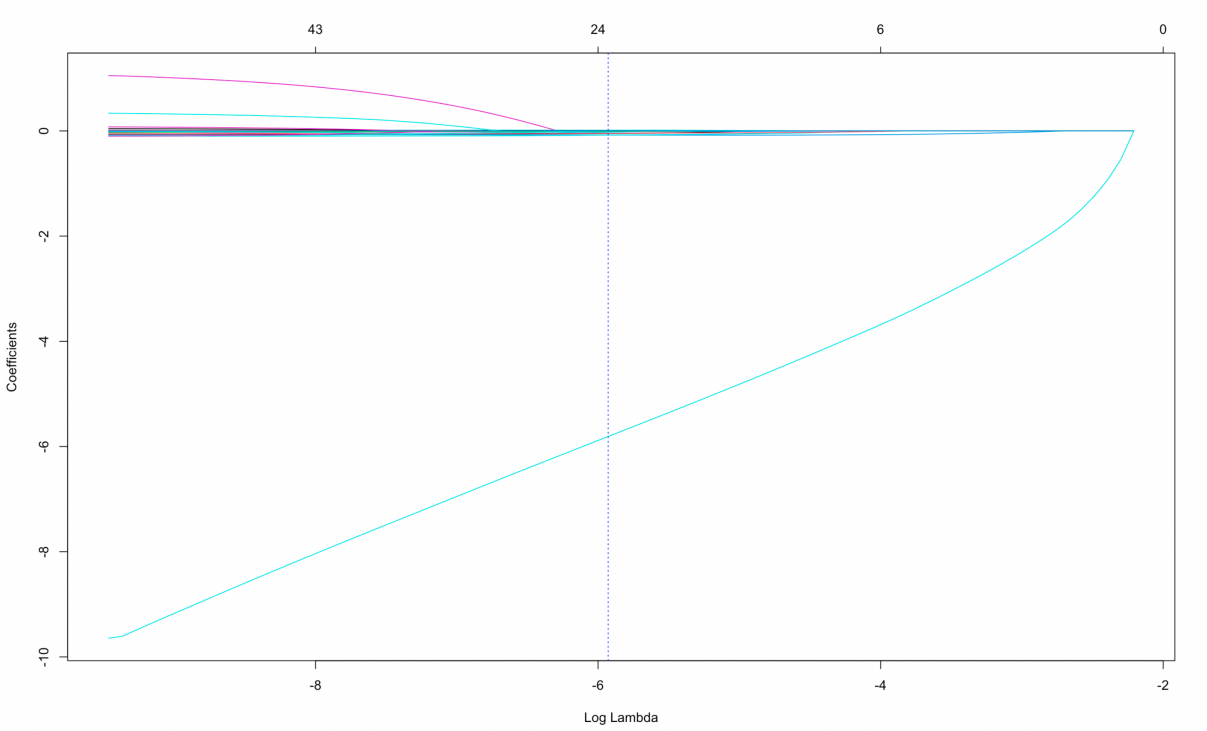
We now implement our fourth model using 'cv.glmnet' that does k-fold cross-validation for 'glmnet' and returns values of lambda used in the fits. To control the coefficients by adding "L1" penalty, we set alpha = 0.

glmLassoReg <- cv.glmnet(data.matrix(x_train_data), y_train_data, family = "binomial", alpha = 1, type.measure="auc")

To see the effect of lamba on coefficients of fit, we can get a plot as follows,

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

abline(v = log(glmLassoReg$lambda.1se), col = "blue", lty = "dotted")



Here, we observe that unlike ridge regression approach which pushes variables to approximately but not equal to zero, the lasso penalty will actually push coefficients to zero. Thus the lasso model not only improves the model with regularization but it also conducts automated feature selection. The blue vertical line depicts the λ at '1se'. On top of the plot we see the number of corresponding predictors retained after L1 regularization at that λ value, which in our case has reduced to 22 from 58 as expected.

Additionally, we can get the AUC values directly at minimum and 1 std deviation of lambda from 'cvm' attribute (since we used type.measure="auc" while training our model).

> glmLassoReg$cvm[glmLassoReg$lambda == glmLassoReg$lambda.min]

[1] 0.7283397

> glmLassoReg$cvm[glmLassoReg$lambda == glmLassoReg$lambda.1se]

[1] 0.7258324

Regularization Parameter Tuning

'cv.glmnet' does not search for alpha values. Thus, to understand the effect on alpha and find our best model, we run a grid search by changing alpha from 0 to 1.

To achieve this, 'cv.glmnet' should be called with a pre-computed vector 'foldid', and then use this same fold vector in separate calls to cv.glmnet with different values of alpha.

```
foldId <- sample(1:10, size = length(y_train_data), replace=TRUE)
params_grid <- expand.grid(
 alpha     = seq(0, 1, by = .1),
 auc.min    = NA,
 auc.1se    = NA,
 lambda.min = NA,
 lambda.1se = NA
)
for(i in 1:nrow(params_grid)) {
 glmModel <- cv.glmnet(data.matrix(x_train_data),
                       y_train_data,
                       family = "binomial",
                       alpha = params_grid$alpha[i],
                       foldid = foldId,
                       type.measure = "auc")
 params_grid$auc.min[i]    <- glmModel$cvm[glmModel$lambda == glmModel$lambda.min]
 params_grid$auc.1se[i]    <- glmModel$cvm[glmModel$lambda == glmModel$lambda.1se]
 params_grid$lambda.min[i] <- glmModel$lambda.min
 params_grid$lambda.1se[i] <- glmModel$lambda.1se
}
```

The results after grid search were as follows,

alpha	auc.min	auc.1se	lambda.min	lambda.1se
0	0.727069739	0.724746	0.010999523	0.179264819
0.1	0.728193259	0.725886	0.00019222	0.01834831
0.2	0.728209972	0.725832	0.000153034	0.011050302
0.3	0.728229446	0.725952	0.000375278	0.007366868
0.4	0.728250667	0.725839	0.000339018	0.006063847
0.5	0.728262291	0.725889	0.000271215	0.004851077
0.6	0.728270359	0.72592	0.000272232	0.004042564
0.7	0.728273824	0.725963	0.000256093	0.003465055
0.8	0.728276712	0.726001	0.000204174	0.003031923
0.9	0.728280013	0.725994	0.000218603	0.002695043
1	0.728282855	0.725981	0.000196743	0.002425539

Takeaway:

If we observe the AUC ± one standard deviation for the optimal λ value for each alpha setting, we see that they all fall within the same level of accuracy. Consequently, we could select a full lasso model with **λ=0.002425539**, gain the benefits of its feature selection capability and reasonably assume no loss in accuracy.

Model 5 – Optimal – by using variables filtered via full lasso regularization

We filter the variables which had non-zero coefficients based full lasso regularization as,

nzCoef <- tidy(coef(glmLassoReg, s=glmLassoReg$lambda.1se))

nzCoefVars <- nzCoef[-1,1]

Our final model can then be built as using 'glm' using filtered variables,

glmOptimalModel <- glm(formula = y_train_data ~ ., data = x_train_data, family = "binomial")

summary(glmOptimalModel)

```
Call:
glm(formula = y_train_data ~ ., family = "binomial", data = x_train_data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.0574   0.3127   0.4282   0.5498   1.5159

Coefficients:
                        Estimate Std. Error z value Pr(>|z|)
(Intercept)            2.568e+01  5.164e+01   0.497 0.618930
installment           -4.175e-04  5.193e-05  -8.040 9.01e-16 ***
sub_grade             -7.171e-02  2.349e-03 -30.531  < 2e-16 ***
home_ownership        -1.044e-01  1.497e-02  -6.971 3.14e-12 ***
annual_inc             1.660e-06  3.945e-07   4.209 2.57e-05 ***
verification_status   -6.896e-02  1.482e-02  -4.652 3.29e-06 ***
dti                   -1.494e-02  1.525e-03  -9.797  < 2e-16 ***
delinq_2yrs           -6.916e-02  1.187e-02  -5.828 5.62e-09 ***
inq_last_6mths        -2.612e-02  1.358e-02  -1.923 0.054463 .
total_acc              7.411e-03  1.321e-03   5.611 2.01e-08 ***
acc_open_past_24mths  -5.572e-02  4.636e-03 -12.018  < 2e-16 ***
bc_open_to_buy         3.480e-06  2.312e-06   1.505 0.132231
mo_sin_old_rev_tl_op   1.468e-04  1.350e-04   1.087 0.276915
mo_sin_rcnt_tl         4.273e-03  1.654e-03   2.583 0.009783 **
mort_acc               1.194e-02  7.896e-03   1.512 0.130533
mths_since_recent_bc   1.833e-03  4.815e-04   3.807 0.000141 ***
mths_since_recent_inq  7.303e-03  2.520e-03   2.899 0.003749 **
num_actv_bc_tl        -3.010e-02  9.423e-03  -3.194 0.001401 **
num_rev_tl_bal_gt_0   -7.408e-03  6.088e-03  -1.217 0.223695
percent_bc_gt_75      -9.717e-04  3.869e-04  -2.511 0.012027 *
tot_hi_cred_lim        6.196e-07  1.252e-07   4.951 7.39e-07 ***
total_bc_limit         4.830e-06  1.638e-06   2.949 0.003183 **
debt_settlement_flag  -2.226e+01  5.164e+01  -0.431 0.666428
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 63921  on 79889  degrees of freedom
Residual deviance: 54745  on 79867  degrees of freedom
AIC: 54791

Number of Fisher Scoring iterations: 15
```

We looked at how our optimal model perform on train and test datasets. We have identified 'Non-Default' as our class of interest. Our evaluation results are as follows,

Confusion Matrix:

Train Data:

	Predicted	
Actual	**Default**	**Non-Default**
Default	1325	9645

Non-Default	52	68868

Accuracy = 0.8786

Test Data:

	Predicted	
Actual	**Default**	**Non-Default**
Default	339	2403
Non-Default	12	17218

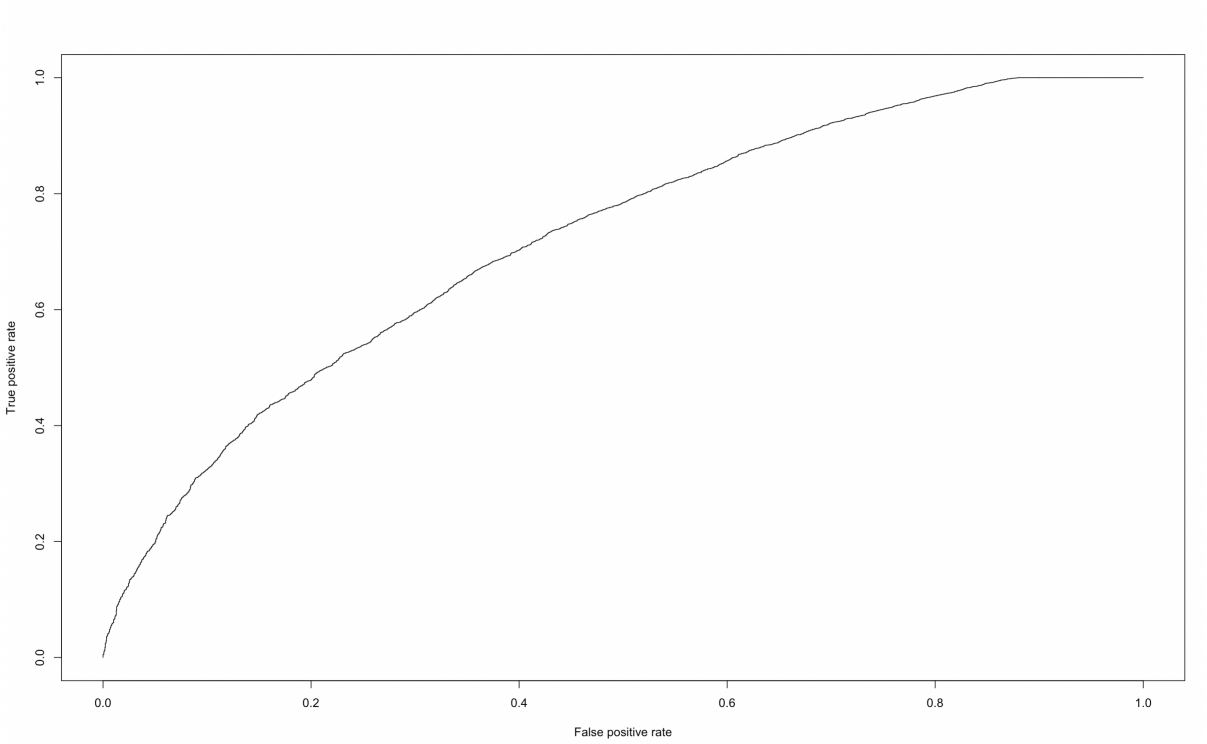
Accuracy = 0.8790

AUC:

prediction(predOptimalModel, y_test_data) %>% performance(measure = "auc") %>% .@y.values

AUC = 0.7190832

ROC:



Takeaway:

As per the confusion matrix above, we can deduct that the model is better overall accuracy, with a slight improvement in ROC and AUC and lower number of predictors.

(b) For the linear model, what is the loss function, and link function you use? (Write the expression for these, and briefly describe).

The logistic equation is stated in terms of the probability that Y = 1, which is π, and the probability that Y = 0, which is 1 - π.

$$\ln\left(\frac{\pi}{1-\pi}\right) = \alpha + \beta X$$

The left-hand side of the equation represents the logit transformation, which takes the natural log of the ratio of the probability that Y is equal to 1 compared to the probability that it is not equal to one. As we know, the probability, π, is just the mean of the Y values, assuming 0,1 coding, which is often expressed as μ. The logit transformation could then be written in terms of the mean rather than the probability,

$$\ln\left(\frac{\mu}{1-\mu}\right) = \alpha + \beta X$$

The transformation of the mean represents a link to the central tendency of the distribution, sometimes called the location, one of the important defining aspects of any given probability distribution. For logistic regression, this is known as the **logit link function**.

The purpose of the loss function is to evaluate how good (or bad) are the predicted probabilities. It should return high values for bad predictions and low values for good predictions. For binary classification, instead of Mean Squared Error, we use a cost function called Cross-Entropy, also known as **Log Loss**. Cross-entropy loss can be divided into two separate cost functions: one for y=1 and one for y=0. The cost function penalizes confident and wrong predictions more than it rewards confident and right predictions. The corollary is increasing prediction accuracy (closer to 0 or 1) has diminishing returns on reducing cost due to the logistic nature of our cost function.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if } y = 1$$

$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if } y = 0$$

The above equations can be combined and final **loss function** for 'glm' looks like,

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))]$$

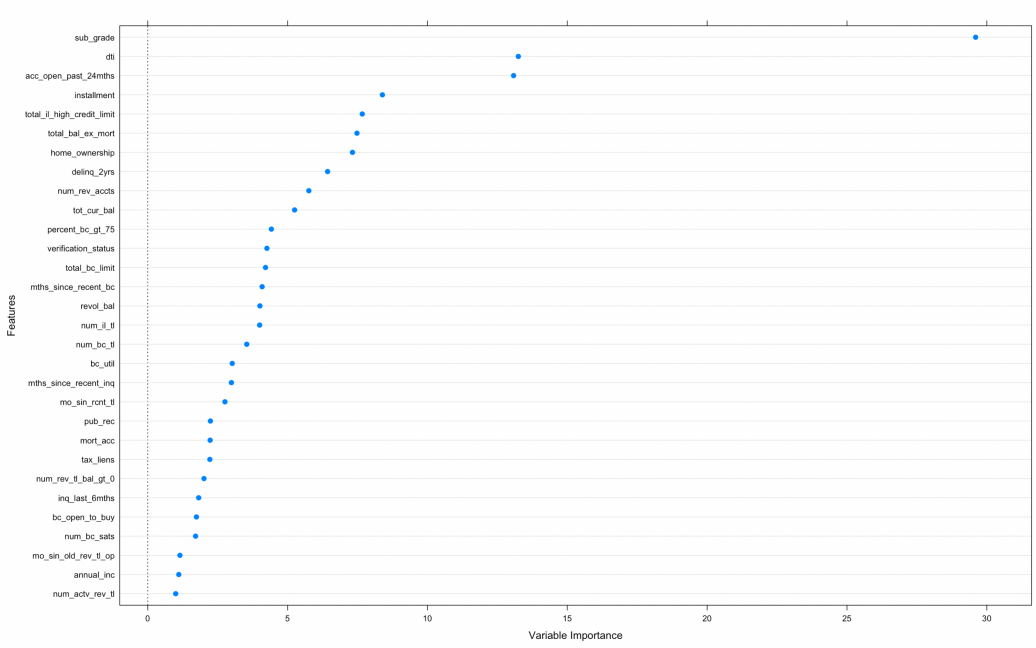
(c) Compare performance of models with that of random forests (from last assignment) and gradient boosted tree models.

	Random Forest	GBM	GLM
Accuracy	**90.65 %**	**87%**	**87.90%**

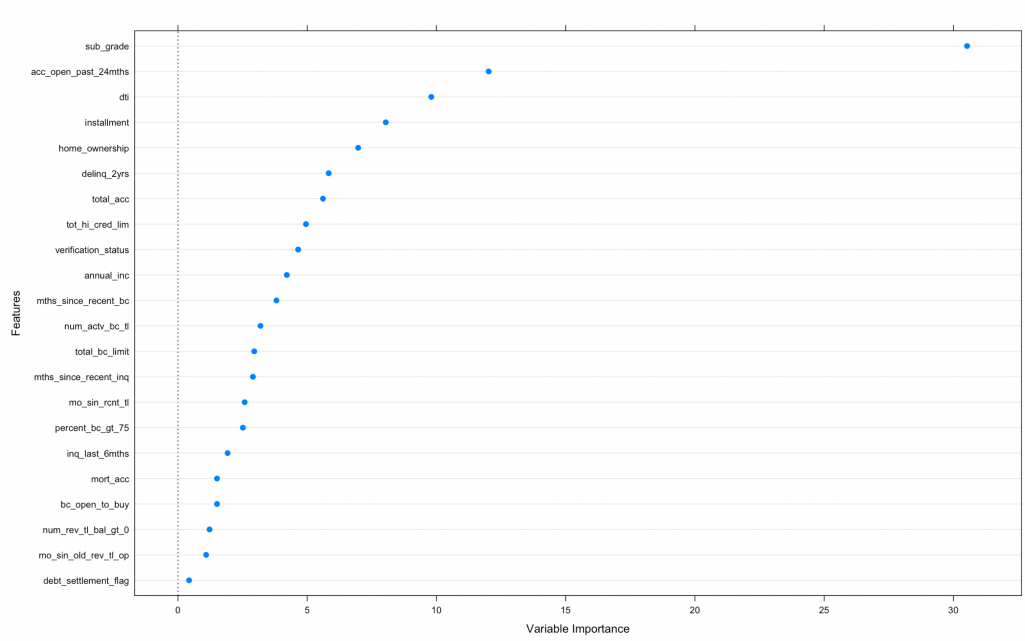
AUC	**73.82%**	**70%**	**71.9%**

(d) Examine which variables are found to be important by the best models from the different methods, and comment on similarities, difference. What do you conclude?

Variable importance for tuned model – using stepwise backward elimination



Variable importance for optimal model - using full lasso regularization



We observe that few top variables such 'sub_grade', 'dti', 'installment', 'acc_open_past_24mths', 'home_ownership' are similar in both models. For subsequent variables, there are some dissimilarities observed. Thus, the models are good in identifying predictors that have high significance (with higher importance values) but fails to differentiate well when significance of predictor is low.

(e) In developing models above, do you find larger training samples to give better models? Do you find balancing the training data examples across classes to give better models?

To test the effect of size of training sample on performance of model, we adopt the following approach.

We first split our entire dataset into two parts: Training and Test using initial ration of 80:20 -

Training set – 80% | Test Set – 20%

Training set – 79890 records | Test Set – 19972 records

Now, we keep the entire 'test set' intact so that the final evaluation on different models can be performed using same proportion of test data.

We now take smaller subsets from our training data, and train a same model with different sizes of subset of train data. Our observation were as follows,

SplitRatio	Number of Train Samples	Number of Test Samples	AUC
0.2	15978	19972	71.89%
0.5	39945	19972	71.93%
0.7	55923	19972	71.91%
0.9	71901	19972	71.90%
0.99	79890	19972	71.89%

The general thumb rule is that the performance of model should improve with more training data. However, we observe that this is not the case with our model and used dataset. One of the possible reasons why we don't observe expected improvement is probably because additional training data maybe noisy or doesn't match whatever we are trying to predict. Other possible explanation is due to highly imbalanced nature of dataset and fitting of high bias model, which do not benefit from more training samples.

The effects of imbalance in dataset can be checked by fitting the model again with balanced dataset. For this purpose, we use the 'ovun.sample' method from 'ROSE'

Undersampling

```
us_lcdfTrn<-ovun.sample(loan_status~., data = training_set, na.action = na.pass, method="under", p=0.5, seed = 123)$data
```

Oversampling

```
os_lcdfTrn<-ovun.sample(loan_status~., data = training_set, na.action = na.pass, method="over", p=0.5, seed = 123)$data
```

Both

```
bs_lcdfTrn<-ovun.sample(loan_status~., data = training_set, na.action = na.pass, method="both", p=0.5, seed = 123)$data
```

On training models based on variables from our optimal model on each of above sampling techniques, we have following observations,

Sampling	Number of Train Samples	Number of Test Samples	AUC	Accuracy
Under	21779	19972	71.88%	67.76%
Over	137636	19972	71.88%	68.51%
Both	79890	19972	71.89%	68.87%

Thus, overall AUC performance remains same but overall accuracy decreases since balancing the classes remove the bias that was created at the first place.

3. Develop models to identify loans *which provide the best returns*. Explain how you define returns? Does it include Lending Club's service costs? Develop glm, rf, gbm/xgb models for this. Show how you systematically experiment with different parameters to find the best models. Compare model performance – explain what performance criteria do you use, and why.

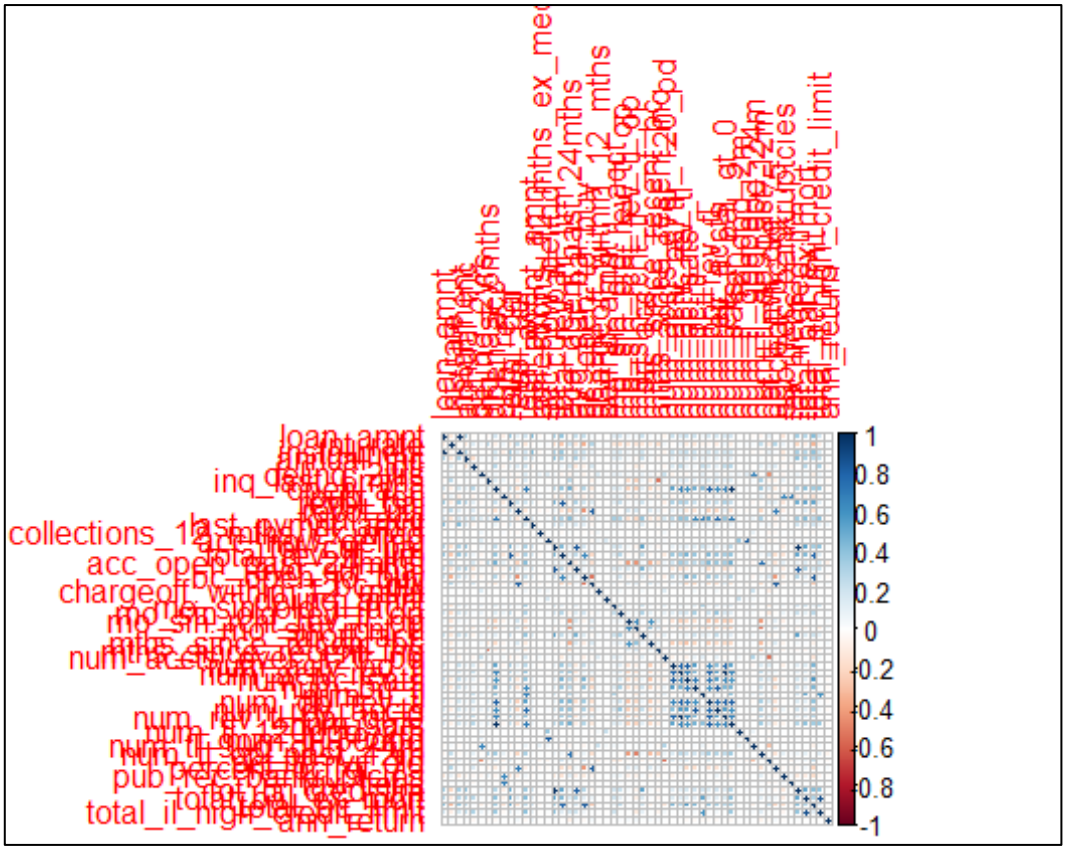
To calculate returns we've taken average of returns and then annualized it. We've taken the total payment for each loan as difference of 'funded amount' and 'total payment'. Since the data has both 'fully paid' and 'charged off' loans, the duration of total payment differs though the term for each loan is same i.e., 36 months. However, to maintain uniformity of calculations for a single time period, we've taken the duration of payment for each loan to be 36 months.
So, the difference calculated above is averaged by dividing with 36 (term for each loan). The result is then multiplied by 12 to get the annualized return. We have not included the Lending Club's service costs since this is the not available with us. However, it depends on the use case and either solution, excluding/including the service cost is excepted.

df$ann_return <- ((df$total_pymnt -df$funded_amnt)/df$funded_amnt)*(12/36)*100

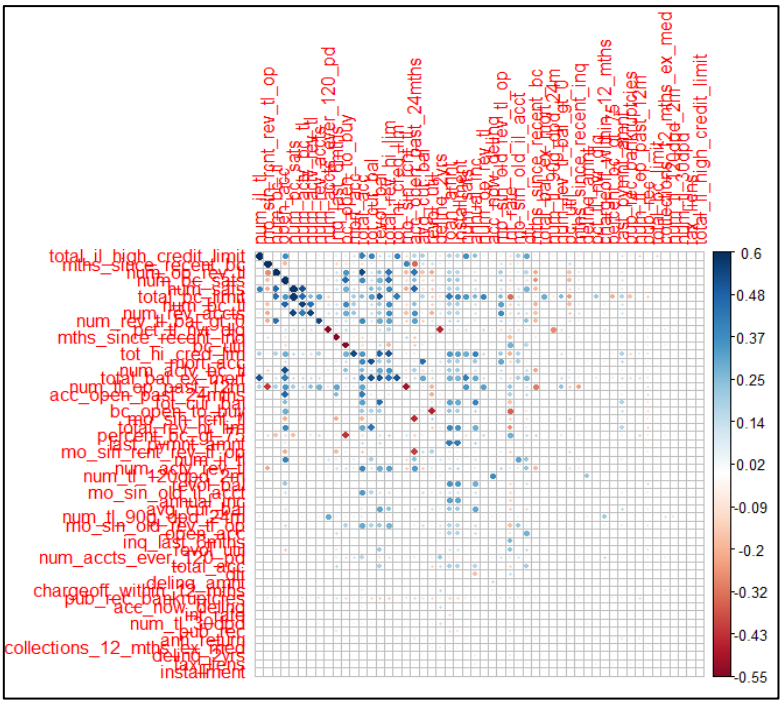
For predicting returns, we built three different models: Linear Model, Gradient Boosting Method and Random Forests.
We have evaluated our models on different metrics, but mainly we've looked at the 'R-Squared' value to make the decision. Since R-Squared is a very good estimation of the variations in the data that can be explained by the linear model.

Linear Model using GLM

Before building linear regression model, we've first looked at the correlation between the different numeric variables in the dataset to check for multicollinearity. We have first built the correlation plot between all the pairs of numerical variables.



We've further checked for the correlation between variables below a certain threshold. For implementing the threshold we've taken the cut-off value of 0.6.

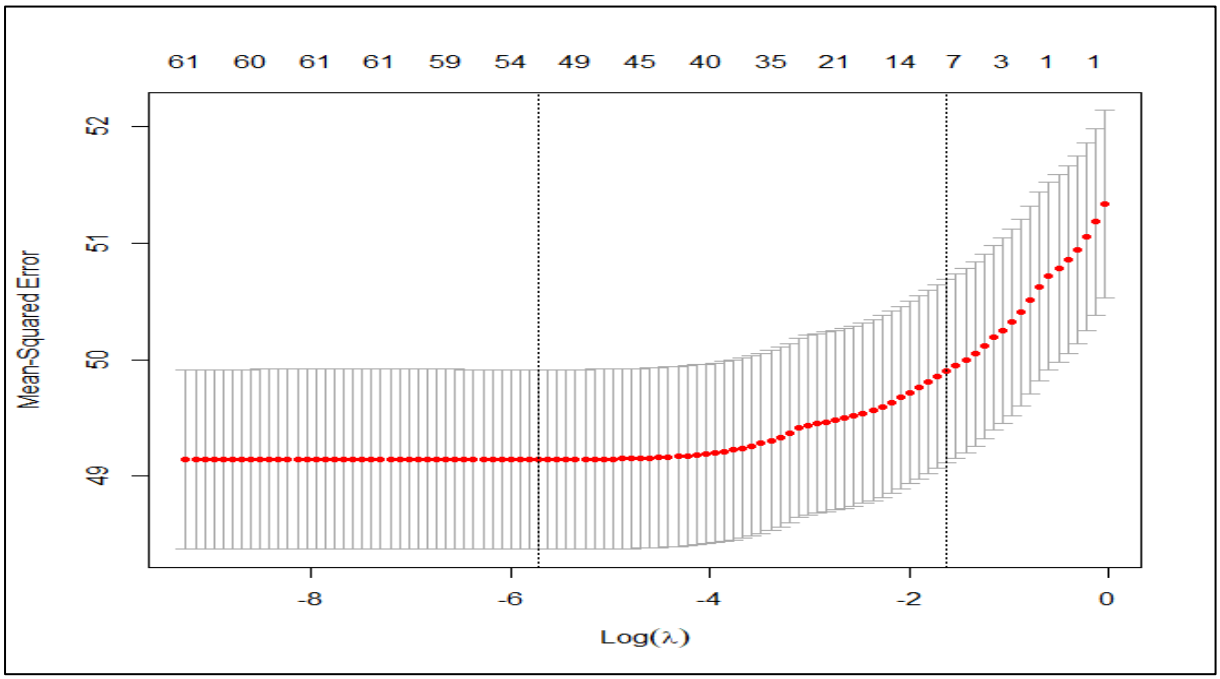


From the above plot, we looked at the top ten variables having high correlation with the predictor variable i.e. 'annual return'.

Next, we built two different linear models. One with lasso regularization and the other with Ridge regularization.
We experimented with different value of parameters, for both the models. After evaluating both the models, we picked the best one.

Lasso Regularization

For lasso regularization, we used the value of alpha parameter as 1. We trained our model on training set and then checked the mean-squared error on cross validation samples for varying values of the regularization parameter (lambda).

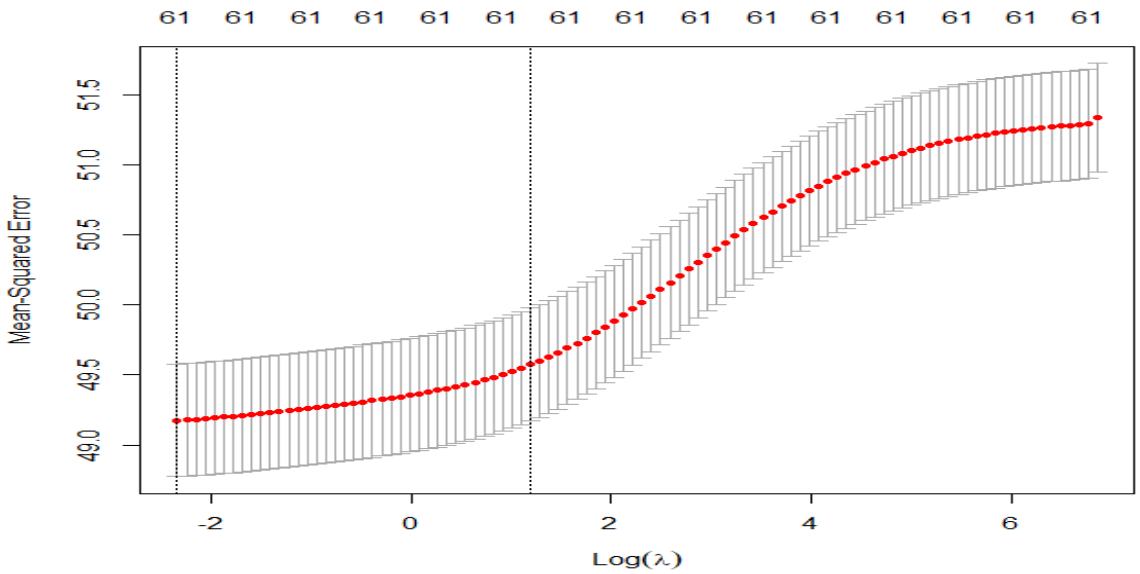


We then built the model with the values of lambda i.e. 'lambda min' and 'lambda 1 STD'. After building the two models, we evaluated the two lasso models on our test data.

We found the best model to be one with lambda value of 'lambda 1 STD' having a R-Squared value of **73.89%.** This means that around 74% of our variations is explained by our linear model.

Ridge Regression

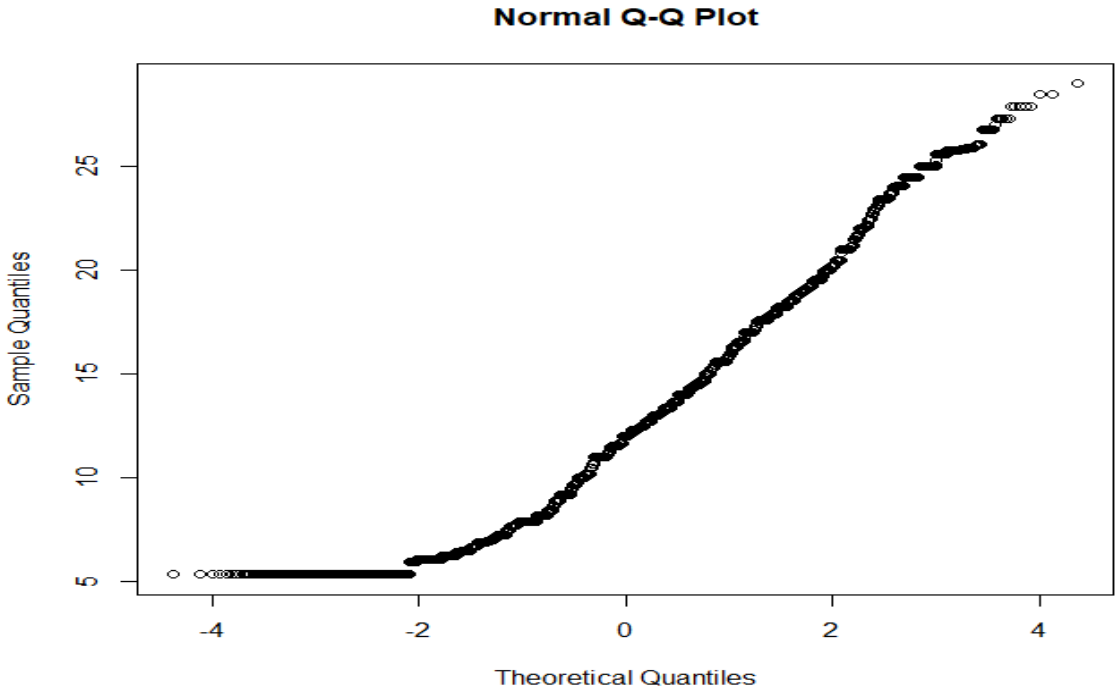
We also built a linear model with ridge regularization. For ridge regression, the value of our parameter, alpha would be 0. After building the model, we then looked at cross validation error for varying values of the regularization parameter i.e. lambda.



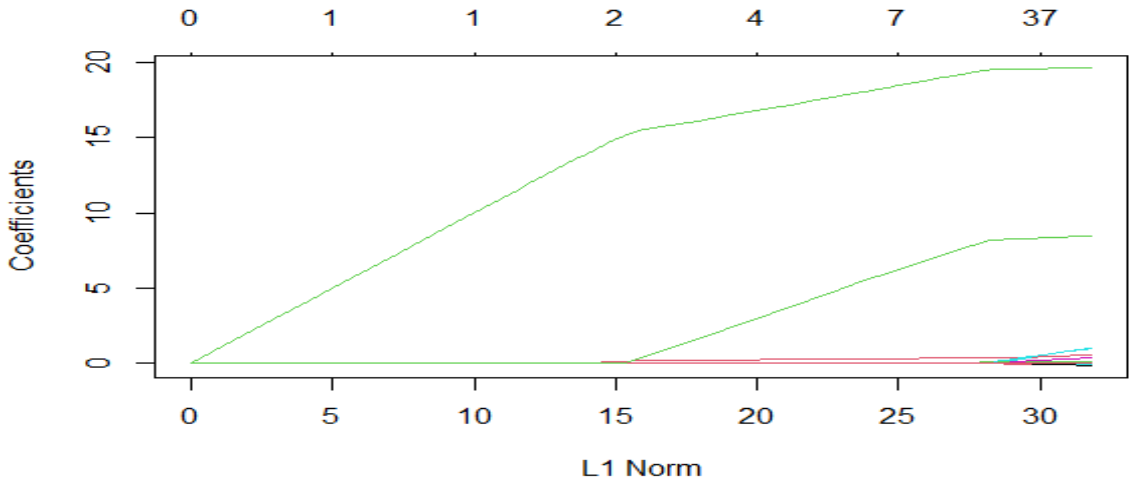
We then evaluated our ridge regression model on our test data for lambda value that gives the minimum cross validation error, i.e., lambda.min.

The ridge regression model gave us a R-Squared value of **73.26%** which is not substantially different from our lasso model. However, the lasso model seems to perform slightly better. Also, the lasso model has the inherent ability of variable selection. Hence, we chose final model to be lasso.

For our final model, we built the lasso model. We then checked the QQ plot of some variables to see if the variables selected by default by the lasso model is in fact significant. Below is the QQ plot of interest rate which is highly correlated with our response variable and it is visible from the plot that interest rate is normally distributed.



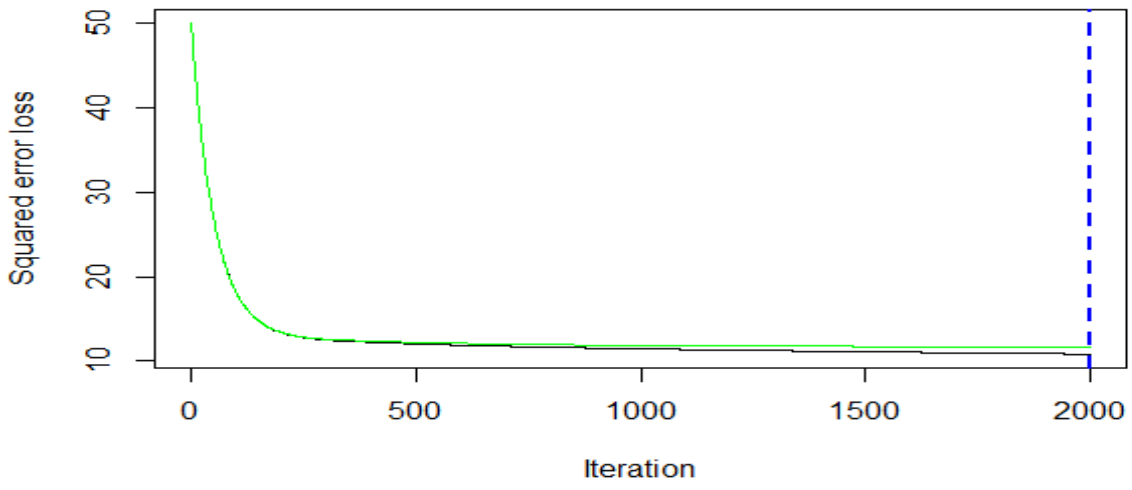
Also, we checked the L1 normalization plot of the lasso model to see how different variables have been handles by our lasso model.



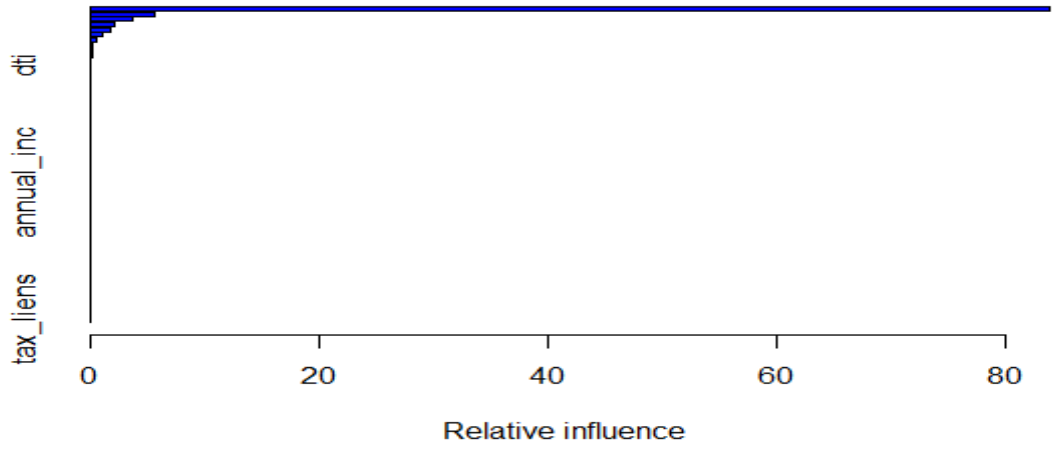
Gradient Boosted Model

After linear model, we built a gradient boosted model. The default settings in gbm includes a learning rate (shrinkage) of 0.001. This is a very small learning rate and typically requires a large number of trees to find the minimum MSE. However, gbm uses a default number of trees of 100, which is rarely sufficient. Consequently, we crank it up to 2,000 trees. The default depth of each tree (interaction.depth) is 4, which means we are ensembling a bunch of stumps. Lastly, we also include cv.folds to perform a 5 fold cross validation.
We came to above values of our input parameter to the GBM model after multiple iterations.

For our model output, the RMSE was **3.41** which is fairly reasonable. Also, we looked at the CV error and found that 2000 trees works best for the gradient boosted model on our dataset as our model converges at 1990 trees.



We also looked at the variable importance of the model and found that it is aligned with the variable importance that we found when using our regression model.



int_rate	5.630284e+00
last_pymnt_amnt	3.692613e+00
debt_settlement_flag	2.188942e+00
addr_state	1.713770e+00
sub_grade	1.024626e+00
installment	5.870837e-01
loan_amnt	2.473857e-01

We then looked at the model's performance on the unseen data that is our test data. For this we looked at the R-squared value of the model. For our Gradient boosted Model, the R-Squared value is **77.33%** which shows that the GBM model performs fairly better than our linear model with approximately 4% increment in explaining the variation in our model.

Random Forest

Next, we built a random forest model for predicting the annual return. For building the random forest, we've used the 'ranger' library owing to its speed. Although, random forests typically perform quite well with categorical variables in their original columnar form, it is worth checking to see if alternative encodings can increase performance. We adjust our mtry parameter to search from 50-200 random predictor variables at each split and re-perform our grid search. The results suggest that one-hot encoding *does not* improve performance.

We built the random forest initially with the default settings. And then experimented with the parameter of 'number of trees' to arrive at the best. The best RMSE that we get is **3.55,** which is a bit greater than the RMSE of GBM.

Also, we tested the random forest model on our testing data to get a R-squared value of **75.5%.**

Conclusion

Out of the GLM, GBM and RF model that we built to predict the 'annual return' of our loan, the GBM model performed best with the least RMSE of **3.41** and the highest R-Squared value of **77.33%**.

Q4. Considering results from Questions 1 and 2 above – that is, considering the best model for predicting loan-status and that for predicting loan returns -- how would you select loans for investment? There can be multiple approaches for *combining information from the two models* - describe your approach and show performance. How does performance here compare with use of single models?

As evident from the previous questions, the GBM model gives us a better model for both prediction of loan classes and for selecting loans with higher returns.

Assumption - Let us assume that we have the budget to invest in 1000 loans.

Single Models –

1. Prediction Model (M1) –

If we take the top 1000 high return loans from the 1st Decile of the GBM model that predicts loan classes, we get the following –

Parameter	Value
count	1000
AvgGBMPredScore	0.97
AvgGBMInvstScore	2.66
NumDef	36
AvgIntrate	6.37
AvgRealRet	2.40
MinRealRet	-26.24
MaXRealRet	5.20
TOTGrA	993
TOTGrB	7
TOTGrC	0
TOTGrD	0
TOTGrE	0
TOTGrF	0
TOTGrG	0

2. Investment Model (M2) -

If we take the top 1000 fully paying loans from the 1st Decile of the GBM model that predicts investment returns, we get the following –

Parameter	Value
count	1000
AvgGBMPredScore	0.84
AvgGBMInvstScore	3.57
NumDef	169
AvgIntrate	15.95
AvgRealRet	3.93
MinRealRet	-30.88
MaXRealRet	14.89

Parameter	Value
TOTGrA	0
TOTGrB	156
TOTGrC	383
TOTGrD	324
TOTGrE	104
TOTGrF	31
TOTGrG	2

In single models, M 1 predicts fewer defaults, but the average annual return is 2.4 whereas the second model gives us a higher return of 3.93 but has much more defaults.

Therefore, we need to look at a combination of models M1 & M2.

Combined Models –

Therefore, we run two GBM models on the same training data and later run them on the same test set. Next, we combine the results from both these models. There are 2 approaches that we have considered.

1. Approach 1 –
    - Using two GBM models, get scores for prediction of loan classes and scores for returns.
    - Prepare a Decile table grouping the scores for investment returns.
    - From the top deciles of highest returns, shortlist the fully paying customers
    - As per our assumption, we will select 1000 customers and check the average annual return we would get if had invested in these loans.

2. Approach 2 –
    - Using two GBM models, get scores for prediction and scores for returns.
    - Multiply these scores
    - Sort the table based on these scores.
    - The highest scores belong to customers that will fully pay and have availed the loan at a high interest rate. Again, we select the Top 1000 customers and check the average return

Results –

First, we prepare a data frame with the scores and actual results from both the models. Now, we use the 2 methods to select loans for investment

1. Approach 1 –

We have a collection of scores and other relevant data in the same table, the table can be grouped by the decile scores of the second Model.

We get the following table –

Decile	Count	AvgGBMPredScore	AvgGBMInvstScore	NumDef	AvgIntrate	AvgRet	MinRealRet	MaXRealRet
1	1998	0.84	3.57	303	15.79	4.20	-31.01	14.91
2	1998	0.86	3.28	246	13.42	3.82	-31.05	15.18
3	1998	0.87	3.11	272	12.63	3.28	-32.24	14.06

4	1997	0.87	2.95	235	11.96	3.24	-32.25	14.94
5	1997	0.90	2.83	224	10.31	2.62	-32.19	13.64
6	1997	0.90	2.73	199	9.93	2.71	-32.25	12.98
7	1997	0.90	2.63	216	9.84	2.39	-29.21	16.12
8	1997	0.87	2.51	275	10.87	2.27	-32.13	14.86
9	1997	0.84	2.34	319	11.92	1.98	-32.20	14.33
10	1997	0.78	1.99	461	13.40	1.14	-32.18	15.90

Decile	TOTGrA	TOTGrB	TOTGrC	TOTGrD	TOTGrE	TOTGrF	TOTGrG
1	0	358	772	605	207	53	3
2	0	923	764	244	55	11	1
3	0	1105	613	226	46	7	1
4	145	1079	520	198	48	5	2
5	812	681	325	144	29	6	0
6	999	516	317	133	30	2	0
7	1028	468	335	121	36	7	2
8	777	525	433	200	56	5	1
9	485	612	553	271	60	14	2
10	283	467	682	396	128	36	5

In the table above, we have grouped the Average prediction score, Average Investment Return score, Number of defaults, Average interest rate, Average Annual Return, Minimum return, Maximum return & the count in each Grade.

The deciles have been prepared by taking the scores of high returns. Thus, the top decile (Decile 1) consists of customers that are predicted to give us the highest returns.

We proceed with this list of customers (Count – 1998). We arrange these 1998 customers in the order of their loan status (Fully paid) scores. Out of these we take the first 1000. These customers are customers that have a high probability to pay without defaults and give the highest return.

Parameter	Value
count	1000
AvgGBMPredScore	0.89
AvgGBMInvstScore	3.52
NumDef	96
AvgIntrate	13.98
AvgRealRet	4.69
MinRealRet	-31.01
MaXRealRet	13.56
TOTGrA	0
TOTGrB	341
TOTGrC	507
TOTGrD	129

Parameter	Value
TOTGrE	19
TOTGrF	4
TOTGrG	0

In this table, the average annual return is 4.69 the average interest rate is 13.98% & the average probability of a fully paying customer is 0.89 (Average score of a paying customer in the 1000 loans). Most of the loans are from Grade C and another majority in Grade B

2. Approach 2 –

In this approach, we multiply the two scores and get the Expected return (ExpecRet). We arrange the data as per the Expected return. We select the first 1000 and summarise the results.

Parameter	Value
count	1000
AvgGBMPredScore	0.89
AvgGBMInvstScore	3.54
NumDef	103
AvgIntrate	14.63
AvgRealRet	4.65
MinRealRet	-32.22
MaXRealRet	14.06
TOTGrA	0
TOTGrB	460
TOTGrC	262
TOTGrD	170
TOTGrE	81
TOTGrF	26
TOTGrG	1

As evident, using the second approach, majority scores are clustered in Grade B. The average score for a fully paying customer is same as Approach 1. The return is marginally lower than Approach 1. The average interest rate is higher. The number of defaults in this table is higher but comparable to the first approach.

Conclusion – Combination models perform better than single models. The first approach gives us a marginally better return & targets lower grade loan applicants whereas the second approach asks us to provide loans to many Grade B applicants. The first combined model performs better on most metrics.

Q5. As seen in data summaries and your work in the first assignment, higher grade loans are less likely to default, but also carry lower interest rates; many lower grad loans are fully paid, and these can yield higher returns. One approach may be to focus on lower grade loans (C and below), and try to identify those which are likely to be paid off. *Develop models from the data on lower grade loans*, and check if this can provide an effective investment approach – for this, you can use one of the methods (glm, rf, or gbm/xgb) which you find to give superior performance from earlier questions. Can this provide a useful approach for investment? Compare performance with that in Question 4.

Using the GBM models, we get the Predictions for loan classes and investment decision. We arrange the table based on the top scores for investment returns. Performing an analysis on Lower grade loans, we get the following results –

Parameter	Value
count	1000
AvgGBMPredScore	0.86
AvgGBMInvstScore	3.89
NumDef	123
AvgIntrate	15.94
AvgRealRet	4.82
MinRealRet	-32.22
MaXRealRet	14.39
TOTGrA	0
TOTGrB	0
TOTGrC	579
TOTGrD	287
TOTGrE	111
TOTGrF	22
TOTGrG	1

Here, we observe that,

On comparison with question 4, the average returns are high but, so are the number of defaults. Most loans are from Grade C (More than half the loans) and the overall performance is better than single models (M1 & M2). If combined to the combination approach, the defaults in Approach 1 & 2 are lower compared to lower grade loans.

Q6. Considering all your results, which approach(s) would you recommend for investing in LC loans? Explain your rationale.

Considering all results, we would use the first approach i.e., using the top decile elements from the highest returns and sorting these loans based on their class prediction. The return from this approach is higher than single models and a combination model obtained by multiplication of scores.

Since the first decile has the highest returns in the total dataset, sorting loans that have least likelihood to default among these high return loans will leave us with loans that give a good annual return and are predicted to be non-delinquent.

However, investors that are not risk averse may want to use the approach discussed in Q5 i.e., modelling the GBM trees based on lower grade loan data. This approach seems to give us the best return but the default rate for a 1000 loan sample is 12.3% (compared to 9.6% in Q4 – Approach 1).

To decide the best model amongst these 2 models, we look at how much the profit would be from these different methods to invest.

Let us assume we can invest $100 in each loan selected. ($100 x 1000 loans = $100,000 Investment)

Also, for any defaults, half the investment is lost.

1. Risk Averse approach (Q4 Approach 1)

In the first approach from Q4, the average annual return is 4.69%, number of defaults per 1000 are 96.

Profit = (Return) – 0.5 x (Loss)

Return = $100,000 x 4.69% x 3 years = $14,070

Loss = 96 loans x $100 = $9600

Profit = $14,070 – (0.5 x $9600) = $9270

2. Risk Tolerant approach (Q5 approach)

In the approach from Q5, the average annual return is 4.82%, number of defaults per 1000 are 123.

Profit = (Return) – 0.5 x (Loss)

Return = $100,000 x 4.82% x 3 years = $14,460

Loss = 123 loans x $100 = $12,300

Profit = $14,460 - 0.5 x $12,300 = $8310

Since profits in the risk averse approach are higher, we will proceed with it.