Team member

Vanisa Achakulvisut	UIN667903568	vachak2@uic.edu
Matt Carey	UIN653922201	mcarey21@uic.edu
Abhijit Zarekar	UIN676178928	azarek3@uic.edu

This is a continuation of the previous assignment where you developed decision tree based models to predict “Fully Paid” vs “Charged Off” loans in the Lending Club platform. In this second assignment, you will develop additional models – GBM, GLM (XGB) - to predict which loans are likely to be paid off and which will default. The previous assignment ended with the question on effective investment decisions based on your predictive models – we will examine this in more detail in the second assignment. We will also focus on parameter tuning, and reliable performance estimates through resampling and cross-validation.

1. (a1) Develop gradient boosted models to predict loan_status. Experiment with different parameter values, and identify which gives ‘best’ performance. How do you determine ‘best’ performance?

Before performing loan_status prediction, training and testing of the dataset should be balanced due to the large difference in proprotion between “Fully Paid” and “Charge Off”
Our dataset are follows
lcdfTrn - stands for train dataset
lcdfTst - stands for test dataset

<u>**Gradient boosted models (gbm model) : Loan Prediction**</u>
We develop gradient boosted models to predict loan_status. For status prediction the distribution should for gbm model is "bernoulli" distribution because of 0/1 prediction,

gbm : Loan Prediction
<u>Step 1 : Trial the parameters for the model. Therefore, we experiment with different parameters while developing the model.</u>
Our trials parameters are n.trees, shrinkage, interaction.depth, bag.fraction
<u>Step 2 : Applying the best parameter to the model</u>
<u>Step 3 : Comparing the AUC from Training and Testing dataset</u>
<u>Step 4 : Evaluating the results</u>

- Find the variable importances for the models
- Find Best Iteration or n.trees
- Develop the confusion matrix, and calculate AUC

Trying different values of parameter in gbm():

> paramGrid

	numtree	shrinkage	treeDepth	bagFraction	bestTree	minRMSE
1	**1000**	**0.01**	**5**	**1**	**1000**	**1.511700**
2	**1500**	**0.01**	**5**	**1**	**1500**	**1.507713**
3	**1000**	**0.01**	**10**	**1**	**1000**	**1.503146**
4	**1500**	**0.01**	**10**	**1**	**1500**	**1.496046**

The best parameters from trying the different parameters is numtree = 1500, shrinkage = 0.01, treeDepth = 10, bagFraction = 1 , bestTree = 1500
We applied these parameters to the gbm model.

<u>Variable Importance Results:</u>

Fig. 1: Variable Importance

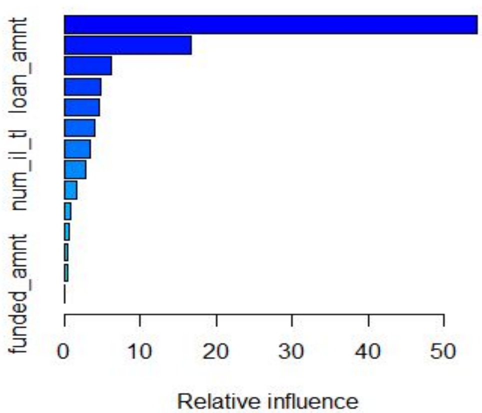
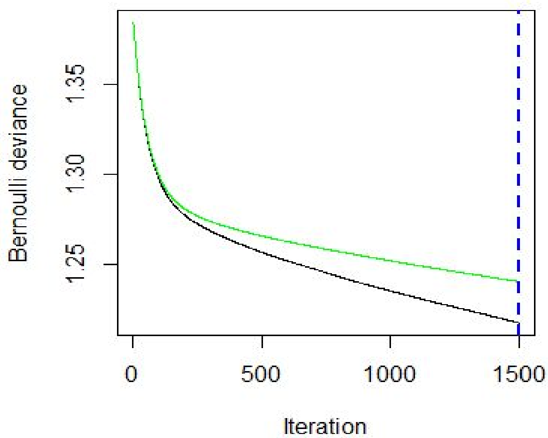
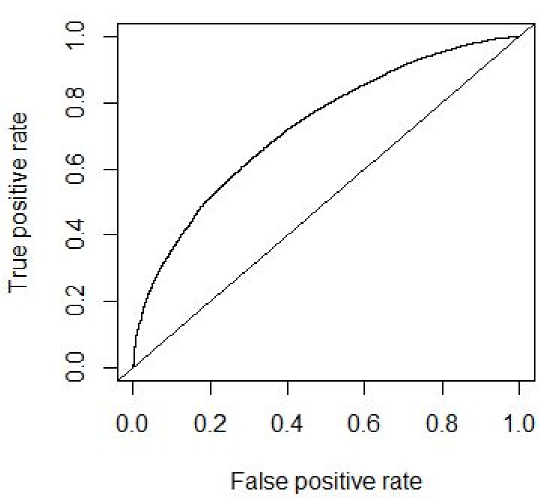


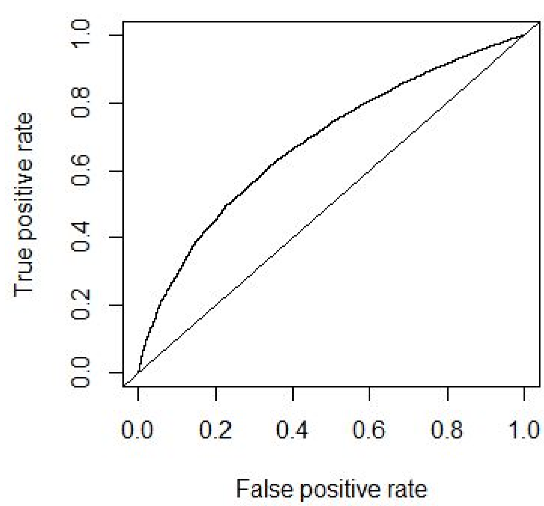
Fig. 2: Summary of the gbm model which gives variable importance

```
> summary(gbm_MM_b, cbars=TRUE)
                                        var    rel.inf
int_rate                           int_rate 54.2925281
sub_grade                         sub_grade 16.6160440
loan_amnt                         loan_amnt  6.1292832
purpose                             purpose  4.7002330
home_ownership               home_ownership  4.5043032
num_sats                           num_sats  3.9294598
num_rev_accts                 num_rev_accts  3.3894647
num_il_tl                         num_il_tl  2.8236104
verification_status     verification_status  1.5178581
pub_rec_bankruptcies pub_rec_bankruptcies  0.7770691
grade                                 grade  0.6439936
tax_liens                         tax_liens  0.3682576
initial_list_status     initial_list_status  0.3078952
funded_amnt                     funded_amnt  0.0000000
```

Fig. 3: Bernoulli Deviance and AUC Curve:



<table>
<tr>
<td>Fig. 4: ROC Curve - lcdfTrn
AUC = 0.728761</td>
<td>Fig. 5: ROC Curve - lcdfTst
AUC = 0.6815637</td>
</tr>
<tr>
<td>


</td>
<td>


</td>
</tr>
</table>

We also tried n.trees=500, shrinkage = 0.3, interaction.depth=4, bag.fraction=0.5 at the beginning of making the model . The results as follows

```
gbm_MM <- gbm(loan_status~.,
            data = lcdfTrn, distribution = "bernoulli",
            n.trees=500, shrinkage = 0.3,
            interaction.depth=4, bag.fraction=0.5,
            cv.folds=2, n.cores=16)
print(gbm_MM)
summary(gbm_MM, cbars=TRUE)
bestIter <- gbm.perf(gbm_MM, method='cv')
print(bestIter)
```

Fig. 6: Variable importance graph

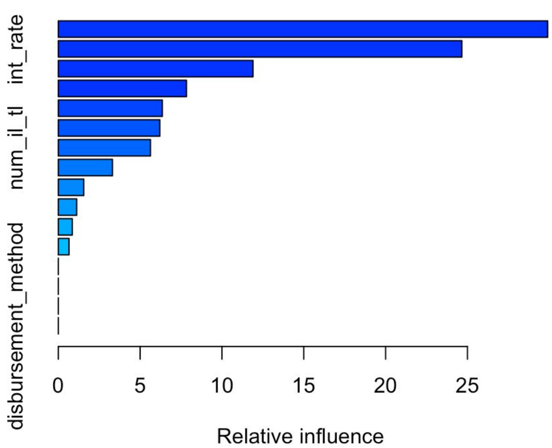


Fig. 7: Summary of the gbm model which gives variable importance

```
> summary(gbm_MM, cbars=TRUE)
                                        var    rel.inf
sub_grade                         sub_grade 29.9029916
int_rate                           int_rate 24.6624115
loan_amnt                         loan_amnt 11.8970976
purpose                             purpose  7.8330029
num_rev_accts                 num_rev_accts  6.3588657
num_sats                           num_sats  6.2054945
num_il_tl                         num_il_tl  5.6345303
home_ownership               home_ownership  3.3127434
verification_status     verification_status  1.5580894
tax_liens                         tax_liens  1.1255248
pub_rec_bankruptcies pub_rec_bankruptcies  0.8542676
initial_list_status     initial_list_status  0.6549807
funded_amnt                     funded_amnt  0.0000000
term                                   term  0.0000000
grade                                 grade  0.0000000
disbursement_method     disbursement_method  0.0000000
```

> print(bestIter) # bestIter means best iteration or n.tree = 492

So, based on the summary above the following are the 5 top important variables (relative influence):

1. Sub_grade
2. Int_rate
3. Loan_amnt
4. Purpose
5. num_rev_accts

Fig. 8: Graph of number of iterations vs Bernoulli deviance
bestIter <- gbm.perf(gbm_MM, method='cv')
print(bestIter)
> print(bestIter) # bestIter means best iteration or n.tree = 492
(Best tree = 492)

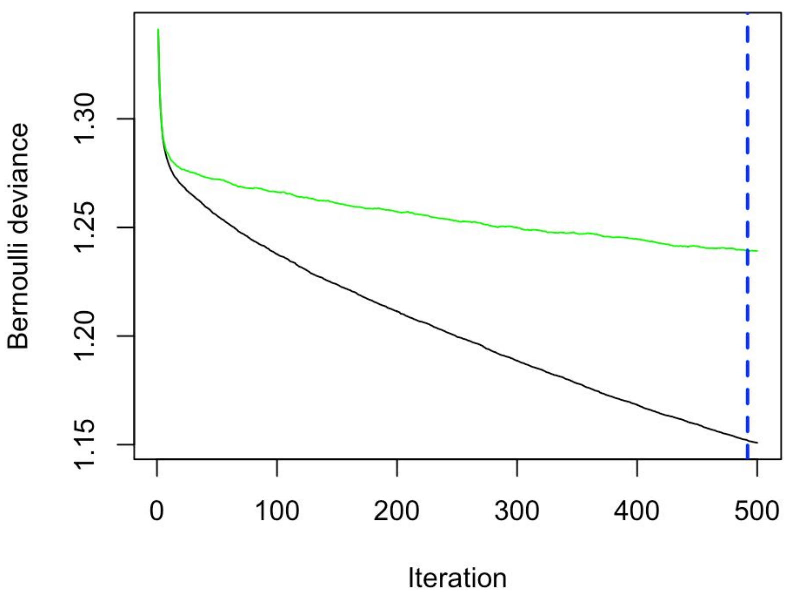
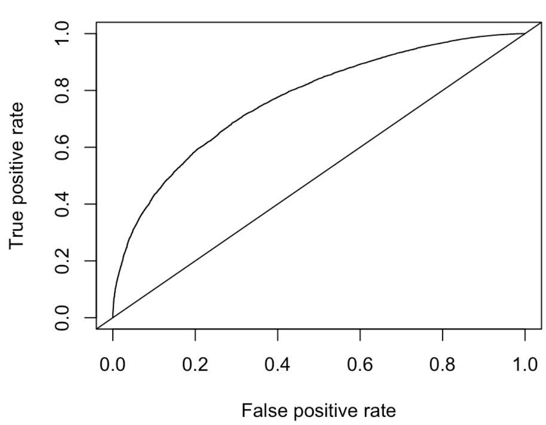
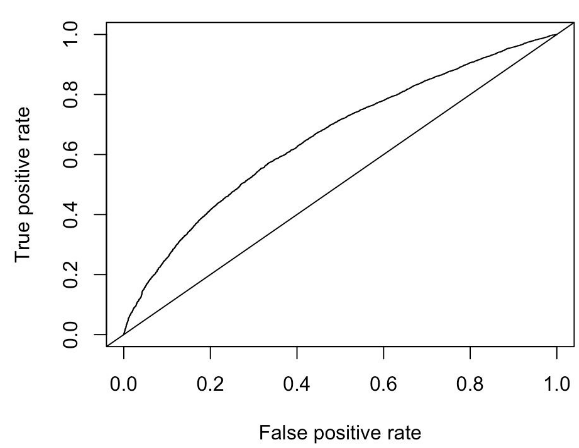


Fig. 9: ROC Curve - lcdfTrn , AUC = 0.7694294	**Fig. 10:** ROC Curve - lcdfTst , AUC = 0.656974
	

From above AUC graphs we learn that the AUC value of the train dataset is higher than the test dataset**.**

Another example of experimenting different values of parameters:

```
gbm_MM_k <- gbm(loan_status~.,
           data = lcdfTrn, distribution = "bernoulli",
           n.trees=1000, shrinkage = 0.01,
           interaction.depth=5, bag.fraction=0.7,
           cv.folds=4, n.cores=NULL)
```

Fig. 11:

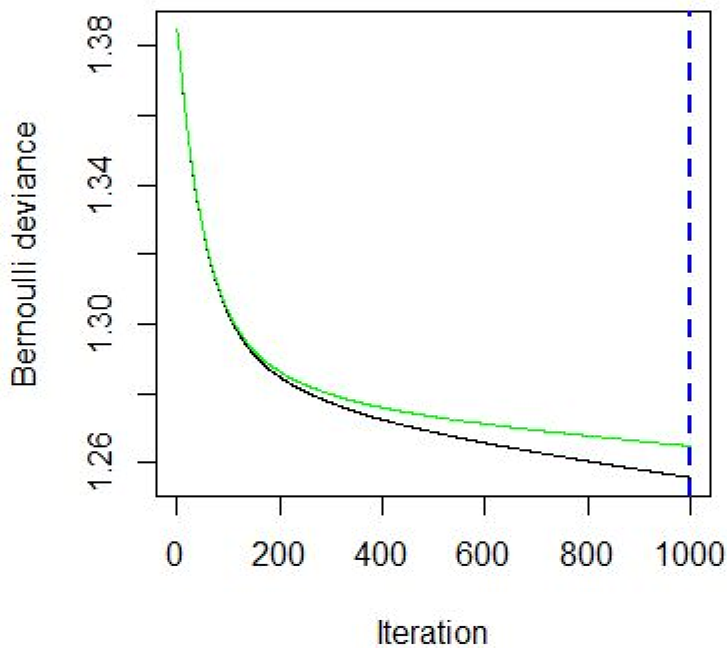


Fig. 12:

```
> summary(gbm_MM_k)
                                      var     rel.inf
int_rate                         int_rate 61.69067757
sub_grade                       sub_grade 18.05198490
home_ownership             home_ownership  5.22487889
loan_amnt                       loan_amnt  3.81558221
purpose                           purpose  2.65647722
num_sats                         num_sats  2.57888803
num_rev_accts               num_rev_accts  1.96918917
num_il_tl                       num_il_tl  1.19784860
verification_status   verification_status  1.17382406
grade                               grade  0.71108439
pub_rec_bankruptcies pub_rec_bankruptcies  0.61284666
tax_liens                       tax_liens  0.29417126
initial_list_status   initial_list_status  0.02254707
funded_amnt                   funded_amnt  0.00000000
term                                 term  0.00000000
disbursement_method   disbursement_method  0.00000000
```

Fig. 13:
ROC Curve - lcdfTrn , AUC = 0.6989405:

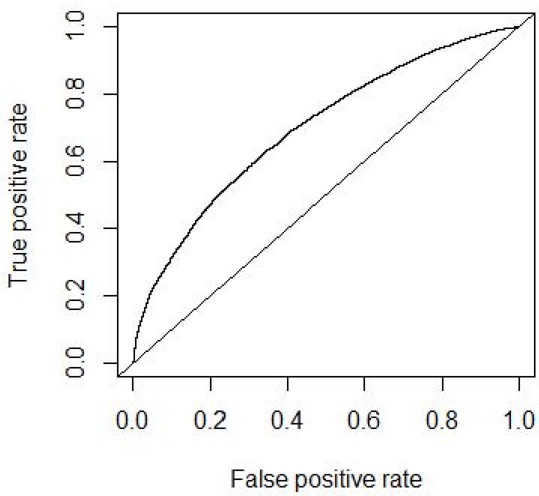
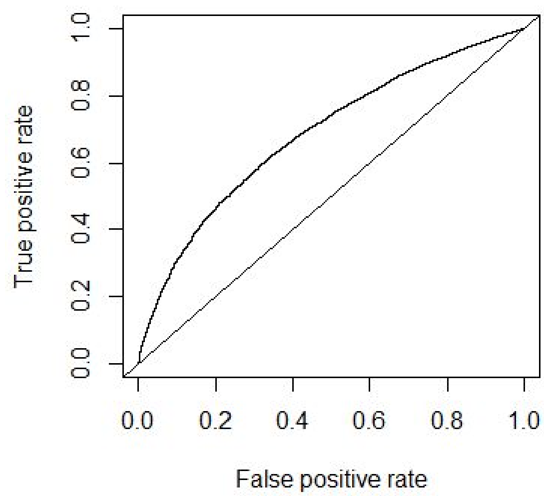


Fig. 14:
ROC Curve - lcdfTst , AUC = 0.686057



After experimenting with different parameter values in Gradient Boosted Model we obtained the highest accuracy for following model:

```
gbm_MM_k <- gbm(loan_status~.,
        data = lcdfTrn, distribution = "bernoulli",
        n.trees=1000, shrinkage = 0.01,
        interaction.depth=5, bag.fraction=0.7,
        cv.folds=4, n.cores=NULL)
```

The ROC curve for Test dataset is shown in Fig 14. Above.
AUC = 0.686057

Therefore, changing the values of parameters such as shrinkage, depth of the trees, number of trees, etc. doesn't much affect the accuracy of the gbm model. However, increasing the number of trees and reducing the shrinkage makes the model a little better in predicting the unseen test dataset.

1. (a2) For the gbm model you develop, what is the loss function, and corresponding gradient in the method you use? (Write the expression for these, and briefly describe).

The Bernoulli loss function that we use in our gbm model. The Bernoulli distribution is one of the options available under the GBM (generalized boosted regression modeling). If nothing else is specified using gbm and the response only has 2 unique values, Bernoulli is always assumed first. Bernoulli Distribution is a logistic regression equation for outcomes that have a discrete distribution between 0 and 1. The Bernoulli Distribution equation is straightforward and makes the loss function easier to understand. The probability of event occurring is 1, then the probability of p not happening is a probability of zero, or the opposite of it happening which is 1-p. We use this formula for Logistic loss: L(y, F) = ln(1 + exp(-yF)). Drawn out we see the formula as follows:

$$\Psi(y, f)_{\mathrm{Bern}} = \log(1 + \exp(-\bar{y}\, f))$$

For the gbm model the corresponding gradient in the method is the Stochastic Gradient Descent. During every boosting iteration uses training data without replacement. This means that once a piece of data is used it is not placed back into the training data set that is being used for that model. With SGD we are applying using small step sizes (“n” in our example below”). Through small iterations we work our way down the slope of the objective function to find the optimal parameter “w” (with respect to each parameter/feature).

$$\frac{dL}{dw} = 0 = \sum_t x^t \mu^t - x^t y^t = \sum_t x^t (\mu^t - y^t)$$

$$w^{new} = w + \Delta w$$

$$where$$

$$\Delta w = -\eta \sum_t x^t (\mu^t - y^t)$$

$$\eta : \ \ small \ \ step \ \ size$$

1 (b1) Develop linear (glm) models to predict loan_status. Experiment with different parameter values, and identify which gives 'best' performance. How do 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.

We developed linear regression models using cv.glmnet() to predict loan_status.
Following are the results of experimenting with different values of alpha parameters.
Alpha = 1 is for Lasso regression
Alpha = 0.5, 0.7 i.e. values between 0 and 1 is for Elastic net
Alpha = 0 is for Ridge regression.

cv.glmnet

```
set.seed(1789)
cvglmnet_1 <- cv.glmnet(data.matrix(xD), yD,
                        family = "binomial",
                        nfolds = 5,
                        alpha = 1)
```

Alpha = 1 (Lasso)

Fig. 15:
#confusion_matrix for lcdfTrn
Accuracy : 0.6255

```
Confusion Matrix and Statistics

          Reference
Prediction     1     0
         1 22809 13699
         0 12971 21729

               Accuracy : 0.6255
                 95% CI : (0.6219, 0.629)
    No Information Rate : 0.5025
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.2508

 Mcnemar's Test P-Value : 8.521e-06

            Sensitivity : 0.6375
            Specificity : 0.6133
         Pos Pred Value : 0.6248
         Neg Pred Value : 0.6262
             Prevalence : 0.5025
         Detection Rate : 0.3203
   Detection Prevalence : 0.5127
      Balanced Accuracy : 0.6254

       'Positive' Class : 1
```

Fig. 16:
#confusion_matrix for lcdfTst
Accuracy : 0.6373

```
Confusion Matrix and Statistics

          Reference
Prediction     1     0
         1 16586  1665
         0  9404  2863

               Accuracy : 0.6373
                 95% CI : (0.6319, 0.6427)
    No Information Rate : 0.8516
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1586

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.6382
            Specificity : 0.6323
         Pos Pred Value : 0.9088
         Neg Pred Value : 0.2334
             Prevalence : 0.8516
         Detection Rate : 0.5435
   Detection Prevalence : 0.5980
      Balanced Accuracy : 0.6352

       'Positive' Class : 1
```

Alpha = 0.5

Fig. 17:
#confusion_matrix for lcdfTrn
Accuracy : 0.6272

```
Confusion Matrix and Statistics

          Reference
Prediction     1     0
         1 22962 13728
         0 12818 21700

               Accuracy : 0.6272
                 95% CI : (0.6236, 0.6308)
    No Information Rate : 0.5025
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.2543

 Mcnemar's Test P-Value : 2.418e-08

            Sensitivity : 0.6418
            Specificity : 0.6125
         Pos Pred Value : 0.6258
         Neg Pred Value : 0.6287
             Prevalence : 0.5025
         Detection Rate : 0.3225
   Detection Prevalence : 0.5153
      Balanced Accuracy : 0.6271

       'Positive' Class : 1
```

Fig. 18:
#confusion_matrix for lcdfTst
Accuracy : 0.6401

```
Confusion Matrix and Statistics

          Reference
Prediction     1     0
         1 16685  1678
         0  9305  2850

               Accuracy : 0.6401
                 95% CI : (0.6347, 0.6455)
    No Information Rate : 0.8516
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1601

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.6420
            Specificity : 0.6294
         Pos Pred Value : 0.9086
         Neg Pred Value : 0.2345
             Prevalence : 0.8516
         Detection Rate : 0.5467
   Detection Prevalence : 0.6017
      Balanced Accuracy : 0.6357

       'Positive' Class : 1
```

Alpha = 0.7

Fig. 19:
#confusion_matrix for lcdfTrn
Accuracy : 0.6268

```
Confusion Matrix and Statistics

          Reference
Prediction     1     0
         1 22867 13661
         0 12913 21767

               Accuracy : 0.6268
                 95% CI : (0.6232, 0.6304)
    No Information Rate : 0.5025
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.2535

 Mcnemar's Test P-Value : 4.597e-06

            Sensitivity : 0.6391
            Specificity : 0.6144
         Pos Pred Value : 0.6260
         Neg Pred Value : 0.6277
             Prevalence : 0.5025
         Detection Rate : 0.3211
   Detection Prevalence : 0.5130
      Balanced Accuracy : 0.6268

       'Positive' Class : 1
```

Fig. 20:
#confusion_matrix for lcdfTst
Accuracy : 0.6384

```
Confusion Matrix and Statistics

          Reference
Prediction     1     0
         1 16626  1671
         0  9364  2857

               Accuracy : 0.6384
                 95% CI : (0.633, 0.6438)
    No Information Rate : 0.8516
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1591

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.6397
            Specificity : 0.6310
         Pos Pred Value : 0.9087
         Neg Pred Value : 0.2338
             Prevalence : 0.8516
         Detection Rate : 0.5448
   Detection Prevalence : 0.5995
      Balanced Accuracy : 0.6353

       'Positive' Class : 1
```

Alpha = 0 (Ridge)

Fig. 21:
#confusion_matrix for lcdfTrn
Accuracy : 0.6276

```
Confusion Matrix and Statistics

          Reference
Prediction     1     0
         1 22904 13644
         0 12876 21784

               Accuracy : 0.6276
                 95% CI : (0.624, 0.6311)
    No Information Rate : 0.5025
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.255

 Mcnemar's Test P-Value : 2.479e-06

            Sensitivity : 0.6401
            Specificity : 0.6149
         Pos Pred Value : 0.6267
         Neg Pred Value : 0.6285
             Prevalence : 0.5025
         Detection Rate : 0.3216
   Detection Prevalence : 0.5133
      Balanced Accuracy : 0.6275

       'Positive' Class : 1
```

Fig. 22:
#confusion_matrix for lcdfTst
Accuracy : 0.6392

```
Confusion Matrix and Statistics

          Reference
Prediction     1     0
         1 16641  1662
         0  9349  2866

               Accuracy : 0.6392
                 95% CI : (0.6338, 0.6446)
    No Information Rate : 0.8516
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1606

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.6403
            Specificity : 0.6330
         Pos Pred Value : 0.9092
         Neg Pred Value : 0.2346
             Prevalence : 0.8516
         Detection Rate : 0.5453
   Detection Prevalence : 0.5997
      Balanced Accuracy : 0.6366

       'Positive' Class : 1
```

Summarize Table for glmnet model

alpha	Accuracy on Test dataset
1 (Lasso regression)	**0.6373**
0.5 (Elastic net)	**0.6401**
0.7 (Elastic net)	**0.6384**
0 (Ridge regression)	**0.6392**

Based on the experiments with different alpha values the accuracy of the model differed slightly. For alpha = 0.5, i.e., Elastic net regression, the highest accuracy on Test dataset was 0.64. **Therefore, glm model with elastic net gives better accuracy than glm model with Ridge or Lasso regression.**

1 (b2) For the linear model, what is the loss function, and link function you use ? (Write the expression for these, and briefly describe).

For the GLM (Generalized Linear Model) we use a series of different loss functions, each one with their own loss functions. We do this as a comparison to see which experiments work better within their respective equations. We figured out through our model that elastic net gives us better accuracy over that of ridge or lasso regression. This makes logic since net elastic is both the combination of lasso and ridge regression. The lasso alpha is default at 1 and the ridge alpha is equal to 0. The reason we use elastic net is because we are controlling lamba (λ, tuning parameter) to

We can see from the breakdown

$$\min_{\beta_0,\beta} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda \left[(1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1 \right]$$

`glmnet` solves the following problem

$$\min_{\beta_0,\beta} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda \left[(1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1 \right],$$

over a grid of values of λ covering the entire range. Here $l(y, \eta)$ is the negative log-likelihood contribution for observation i; e.g. for the Gaussian case it is $\frac{1}{2}(y-\eta)^2$. The *elastic-net* penalty is controlled by α, and bridges the gap between lasso ($\alpha = 1$, the default) and ridge ($\alpha = 0$). The tuning parameter λ controls the overall strength of the penalty.

1 (c) Compare performance of models with that of random forests (which you did in your last assignment).

We developed a ranger model to predict loan_status.

Following are the results of experimenting with different values of no. of trees.
num.trees = 500, 1000

Random Forest with 500 trees

Fig. 23:
#Variable Importance

	x
int_rate	0.160203820
sub_grade	0.126674262
funded_amnt	0.126387315
loan_amnt	0.126342922
num_sats	0.114424004
num_rev_accts	0.110800962
home_ownership	0.105098850
num_il_tl	0.097811824
grade	0.074494190
verification_status	0.073872229
purpose	0.051261374
initial_list_status	0.048003667
pub_rec_bankruptcies	0.023507816
tax_liens	0.007359747
term	0.000000000
disbursement_method	0.000000000

Fig. 24:
#plot ROC lcdfTrn
#auc lcdfTrn = 0.999846

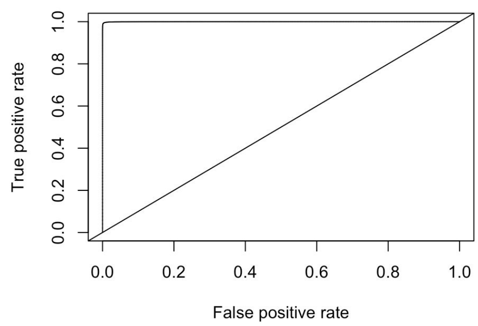
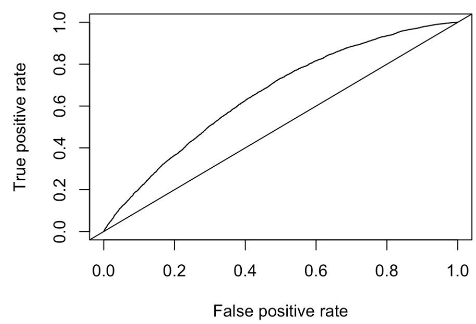


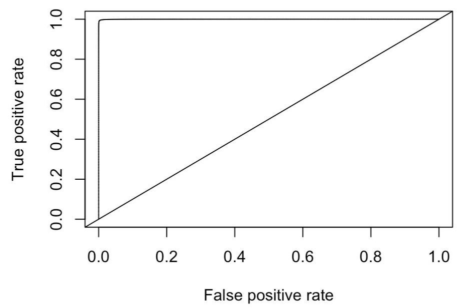
Fig. 25:
#plot ROC lcdfTst
#auc lcdfTst = 0.6567078

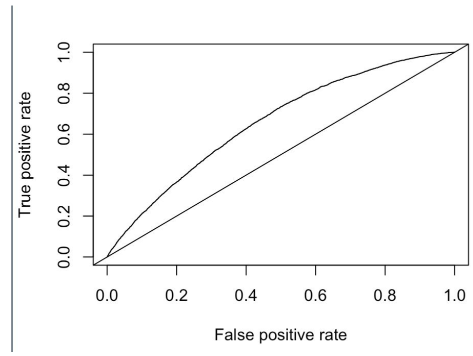


#Random Forest, Trees = 1000
#Variable importance

Fig. 26:
#Variable Importance
	Fig. 27:
	#plot ROC lcdfTrn
	AUC = 0.9998481
		Fig. 28:
		#plot ROC lcdfTst
		AUC = 0.6574085

	x
int_rate	0.15807998
sub_grade	0.12634435
funded_amnt	0.12632383
loan_amnt	0.12607116
num_sats	0.11460710
num_rev_accts	0.11111126
home_ownership	0.10490847
num_il_tl	0.09763780
grade	0.07815351
verification_status	0.07415688
purpose	0.05199239
initial_list_status	0.04702194
pub_rec_bankruptcies	0.02368985
tax_liens	0.00742447
term	0.00000000
disbursement_method	0.00000000





1 (c) Compare performance of models with that of random forests (which you did in your last assignment).

<u>Summary Table of 3 different models (GMB, GLM, RF)</u>

Model	Accuracy on Test dataset
GBM (with 1500 trees, shrinkage 0.01)	**0.6861** 0.6815637
GLM model (alpha = 0.5 i.e. Elastic net regression)	**0.6401**
Random Forest model	**0.6574**

Comparing the results from gbm, glm, and random forest models, we learn that the **gbm model** gives **the best accuracy** on the Test dataset.

1 (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?

Fig. 29: Variable importance in gbm model

```
> summary(gbm_MM_k)
                                      var     rel.inf
int_rate                         int_rate 61.69067757
sub_grade                       sub_grade 18.05198490
home_ownership             home_ownership  5.22487889
loan_amnt                       loan_amnt  3.81558221
purpose                           purpose  2.65647722
num_sats                         num_sats  2.57888803
num_rev_accts               num_rev_accts  1.96918917
num_il_tl                       num_il_tl  1.19784860
verification_status   verification_status  1.17382406
grade                               grade  0.71108439
pub_rec_bankruptcies pub_rec_bankruptcies  0.61284666
tax_liens                       tax_liens  0.29417126
initial_list_status   initial_list_status  0.02254707
funded_amnt                   funded_amnt  0.00000000
term                                 term  0.00000000
disbursement_method   disbursement_method  0.00000000
```

Fig. 30: Variable importance in cv.glmnet model

```
# A tibble: 16 x 3
   Variable             Importance Sign
   <chr>                     <dbl> <chr>
 1 purpose                 0.00552 POS
 2 disbursement_method     0       NEG
 3 funded_amnt             0       NEG
 4 initial_list_status     0       NEG
 5 loan_amnt               0       NEG
 6 num_il_tl               0       NEG
 7 num_rev_accts           0       NEG
 8 tax_liens               0       NEG
 9 term                    0       NEG
10 num_sats               -0.00500 NEG
11 pub_rec_bankruptcies   -0.0224  NEG
12 verification_status    -0.0356  NEG
13 sub_grade              -0.0436  NEG
14 int_rate               -0.0607  NEG
15 grade                  -0.0674  NEG
16 home_ownership         -0.135   NEG
```

Fig. 31: Variable importance in random forest model with 1000 trees

	x
int_rate	0.15807998
sub_grade	0.12634435
funded_amnt	0.12632383
loan_amnt	0.12607116
num_sats	0.11460710
num_rev_accts	0.11111126
home_ownership	0.10490847
num_il_tl	0.09763780
grade	0.07815351
verification_status	0.07415688
purpose	0.05199239
initial_list_status	0.04702194
pub_rec_bankruptcies	0.02368985
tax_liens	0.00742447
term	0.00000000
disbursement_method	0.00000000

Summary table of Variable importance of each model

Model	**Top important variables as per model**
Gbm with 1500 trees and 0.01 shrinkage	Int_rate, sub_grade, loan_amnt, purpose and home_ownership
Glmnet model with alpha = 0.5	Purpose (disbursement_method, funded_amnt, initial_list_status, loan_amnt with importance = 0)
Random forest with 1000 trees	Int_rate, sub_grade, funded_amnt, loan_amnt

- The variable importance for different models are different. There is some similarity in variable importance between gbm and random forest models. They both give high importance to “int_rate” and “sub_grade” variables.
- However, for glmnet models, the variable importance is way different from the above 2 models. Surprisingly, “purpose” is given variable importance in the cv.glmnet model.
- Another interesting observation is that variable importance varies even with variation in parameters within the same model. For example, Fig. 2 shows high variable importance to sub_grade in the gbm model with 500 trees.

1 (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?

In earlier models we experimented with larger training samples, for example, 70:30 split.
Here we experimented with 50:50 split on Gbm model, thus a smaller training dataset.

```
> TRG_PCT=0.5
> set.seed(1789)
> trnIndex = sample(1:nr, size = round(TRG_PCT*nr), replace=FALSE)
> nrow(lcdfTrn)
[1] 50863
> nrow(lcdfTst)
[1] 50863
```

```
> library(gbm)
> set.seed(1789)
> gbm_MM <- gbm(loan_status~.,
+             data = lcdfTrn, distribution = "bernoulli",
+             n.trees=500, shrinkage = 0.01,
+             interaction.depth=4, bag.fraction=0.5,
+             cv.folds=2, n.cores=16)
```

500 iterations were performed.
The best cross-validation iteration was 500.

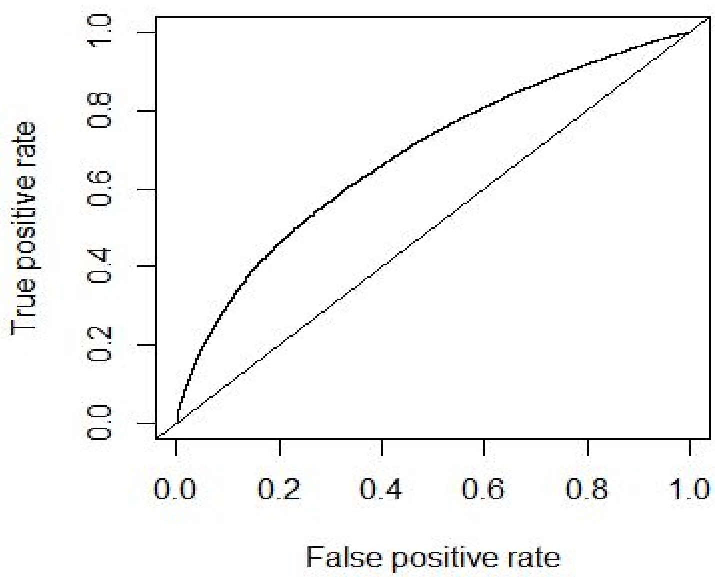
Fig. 32: Following fig. gives the information about variable importance

```
> summary(gbm_MM, cbars=TRUE)
                                      var     rel.inf
int_rate                         int_rate 62.30786803
sub_grade                       sub_grade 20.94464066
home_ownership             home_ownership  5.06118037
loan_amnt                       loan_amnt  4.03106605
num_sats                         num_sats  2.20865998
purpose                           purpose  1.94936542
num_rev_accts               num_rev_accts  1.22277416
verification_status   verification_status  0.80355107
num_il_tl                       num_il_tl  0.48197467
pub_rec_bankruptcies pub_rec_bankruptcies  0.40489091
grade                               grade  0.35687471
tax_liens                       tax_liens  0.20576756
initial_list_status   initial_list_status  0.02138641
funded_amnt                   funded_amnt  0.00000000
term                                 term  0.00000000
disbursement_method   disbursement_method  0.00000000
```

Based on the above fig. Following are the top 5 important variables

1. Int_rate
2. Sub_grade
3. home_ownership
4. Loan_amnt
5. num_stats

Fig. 33: ROC Curve - lcdfTst , AUC = 0.6839434



The accuracy of the model on test data has hardly any change compared to the ROC curve accuracy given in Fig. 9.
Therefore, taking smaller or larger training samples doesn't make a significant difference on the accuracy. Thus, changing the size of the training dataset may not ensure better models.

2. 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.

#Create: annRet, actualTerm, actualReturn (nCol=145+3)
#annRet
lcdf$annRet <- (lcdf$total_pymnt -lcdf$funded_amnt)/lcdf$funded_amnt*(12/36)*100
lcdf$last_pymnt_d<-paste(lcdf$last_pymnt_d, "-01", sep = "")
lcdf$last_pymnt_d<-parse_date_time(lcdf$last_pymnt_d, "myd")
#actualTerm
lcdf$actualTerm <- ifelse(lcdf$loan_status == "Fully Paid", as.duration(lcdf$issue_d %--%lcdf$last_pymnt_d)/dyears(1),3)
#actualReturn
lcdf$actualReturn <- ifelse(lcdf$actualTerm>0,((lcdf$total_pymnt-lcdf$funded_amnt)/lcdf$funded_amnt)*(1/lcdf$actualTerm),0)
dim(lcdf)
#Noted: nCol=145+3 = 148

From the first assignment we had to create new calculations for the annual return, actual term, and actual return. These numbers vary based on how Lending Club measures their return on investments. First we calculated the annual return by subtracting the total payment by the funded amount, then dividing it by the funded amount. This would give us our total annual return number, we then break that number down into a percentage by months in the given term. We do that because some of the term lengths are different lengths of time in months. We have to make sure that the numbers we are using are all normalized to the months given per individual term. After creating the numbers for annual return, actual term, and actual return, we turn to Lending Club's policy in regards to the service costs included in the loan.

Lending Club's service cost is "one percent (1%) of the amount of any borrower payment received by the payment due date or during applicable grace periods"[1]. Therefore, the payments received by Lending Club are associated fees that are found within the total payment amount variable before we do any math to deduce the funded amount from the total payments to come up with our annual returns. In the dataset we use there is not a specific column for the amount, we are using information on the Net Annualized Return policy given to us by LendingClub.com. In the definition provided on the site we see that the Annualized Return policy is based on "on actual borrower payments received each month, net of fees, charge-offs, and recoveries. NAR assumes all loans that are not charged-off will be paid in full, regardless of their current or delinquent status."[2]. We get further clarification from this statement that "net of fees" is directly found in the annual return amount. That is assuming all loans will be paid off in full, which is the model we are following in our problem.

References:

1) https://help.lendingclub.com/hc/en-us/articles/215480768

2) https://blog.lendingclub.com/why-lendingclub-investors-should-focus-net-return#:~:text=NAR%20is%20an%20annualized%20measure,charge%2Doffs%2C%20and%20recoveries.

Show how you systematically experiment with different parameters to find the best models. Compare model performance. Do you find larger training sets to give better models ?

Xgboost : Actual Return Prediction

Step 1 : Finding the best parameters for the model

First, trial the parameters max_depth= 2, 5 ,eta =0.001, 0.01, 0.1 on xgb.cv model

We select the xgb.cv model for training data to ensure that our model expose to the whole data in our Training dataset

```
xgbParamGrid <- expand.grid(max_depth= c(2,5),
                            eta = c(0.1, 0.01,0.001)
```

```
> xgbParamGrid
  max_depth   eta bestTree  bestPerf
1         2 0.100       67 0.0855832
2         5 0.100       51 0.0857738
3         2 0.010      500 0.0856386
4         5 0.010      500 0.0857788
5         2 0.001      500 0.2881452
6         5 0.001      500 0.2881512
```

Performance stands for RMSE value (the least the RMSE value the better the performance)

Xgboost model with max_depth= 2, eta = 0.1 gives the best performance at 0.0856

Step 2 : Applying the best parameter to the model

Once we got the best performance parameters, we applied to the model xgboost.

The variable importances for Training and testing as follows

Variable importances for Training

The final train-rmse:0.085412

	Feature	Gain	Cover	Frequency
1	int_rate	0.470	0.261	0.288
2	home_ownership.MORTGAGE	0.137	0.073	0.076
3	loan_amnt	0.106	0.147	0.116
4	home_ownership.RENT	0.098	0.066	0.066
5	num_sats	0.066	0.156	0.121
6	sub_grade.F4	0.023	0.026	0.030
7	num_il_tl	0.020	0.071	0.071
8	verification_status.Not Verified	0.014	0.037	0.030
9	purpose.other	0.013	0.039	0.030
10	purpose.credit_card	0.008	0.030	0.020
11	pub_rec_bankruptcies	0.008	0.001	0.015
12	sub_grade.D4	0.006	0.006	0.020
13	num_rev_accts	0.006	0.012	0.025
14	tax_liens	0.005	0.014	0.025
15	sub_grade.G5	0.003	0.022	0.015
16	sub_grade.C5	0.003	0.015	0.010
17	purpose.small_business	0.002	0.000	0.005
18	sub_grade.F3	0.002	0.000	0.005
19	purpose.house	0.002	0.007	0.005
20	sub_grade.D3	0.002	0.001	0.005
21	verification_status.Verified	0.001	0.002	0.005
22	sub_grade.E3	0.001	0.004	0.005
23	sub_grade.E5	0.001	0.002	0.005
24	sub_grade.G4	0.001	0.007	0.005

Variable importances for Testing

The final train-rmse:0.083508

	Feature	Gain	Cover	Frequency
1	int_rate	0.441	0.205	0.183
2	loan_amnt	0.097	0.133	0.132
3	num_sats	0.094	0.159	0.132
4	home_ownership.RENT	0.064	0.051	0.051
5	home_ownership.MORTGAGE	0.055	0.068	0.056
6	sub_grade.C3	0.035	0.066	0.046
7	num_rev_accts	0.027	0.069	0.086
8	purpose.house	0.025	0.060	0.041
9	sub_grade.E4	0.022	0.000	0.030
10	num_il_tl	0.020	0.017	0.036
11	purpose.renewable_energy	0.015	0.004	0.015
12	home_ownership.OWN	0.014	0.000	0.020
13	sub_grade.E1	0.013	0.008	0.015
14	purpose.credit_card	0.013	0.038	0.025
15	sub_grade.F2	0.010	0.015	0.015
16	sub_grade.F4	0.008	0.015	0.015
17	sub_grade.F5	0.007	0.007	0.010
18	verification_status.Not Verified	0.007	0.029	0.020
19	tax_liens	0.006	0.003	0.015
20	grade.D	0.005	0.003	0.005
21	sub_grade.D1	0.005	0.023	0.015
22	sub_grade.E2	0.004	0.003	0.005
23	sub_grade.C5	0.003	0.006	0.005
24	purpose.medical	0.003	0.004	0.005
25	verification_status.Verified	0.002	0.007	0.005
26	sub_grade.C1	0.002	0.007	0.005
27	purpose.other	0.001	0.000	0.005
28	purpose.small_business	0.001	0.002	0.005

Step 3 : Comparing the RMSE from Training and Testing dataset

The RMSE from Training and Testing dataset are in acceptable numbers at 8%.

Plus, there is a slight difference of RMSE between training and testing models.

Step 4 : Evaluating the results

predXgbRet_Trn,   RMSE = 0.08590409

	tile	count	avgPredRet	numDefaults	avgActRet	minRet	maxRet	avgTer	totA	totB	totC	totD	totE	totF
	<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	1	7121	0.0636	1310	0.0686	-0.333	0.397	2.18	0	347	2850	3162	644	88
2	2	7121	0.0569	1226	0.0592	-0.311	0.341	2.20	0	1774	3456	1360	468	54
3	3	7121	0.0536	1278	0.0522	-0.333	0.334	2.23	0	2586	3443	720	335	30
4	4	7121	0.0497	1353	0.0503	-0.333	0.307	2.28	11	2315	2864	1489	395	44
5	5	7121	0.0466	1273	0.0469	-0.321	0.283	2.27	128	3087	2956	741	181	23
6	6	7121	0.0438	1155	0.0442	-0.333	0.319	2.27	884	3536	1813	708	156	22
7	7	7121	0.0418	1135	0.0388	-0.322	0.313	2.28	1985	2766	1598	646	112	13
8	8	7121	0.0399	902	0.0366	-0.323	0.412	2.27	3610	2172	816	267	226	30
9	9	7120	0.0378	571	0.0360	-0.323	0.287	2.28	5321	1533	58	23	162	20
10	10	7120	0.0343	610	0.0313	-0.333	0.312	2.34	6124	920	5	6	33	30

predXgbRet_Tst, RMSE = 0.13122

	tile	count	avgPredRet	numDefaults	avgActRet	minRet	maxRet	avgTer	totA	totB	totC	totD	totE	totF
	<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	1	3052	0.0634	614	0.0653	-0.322	0.318	2.18	0	135	1222	1369	281	33
2	2	3052	0.0568	559	0.0560	-0.333	0.320	2.19	0	840	1446	556	187	23
3	3	3052	0.0535	528	0.0541	-0.322	0.367	2.22	2	1094	1491	316	133	13
4	4	3052	0.0496	594	0.0510	-0.322	0.366	2.26	5	1030	1213	611	165	26
5	5	3052	0.0466	548	0.0480	-0.322	0.277	2.29	49	1304	1246	356	85	12
6	6	3052	0.0440	483	0.0465	-0.323	0.341	2.28	346	1517	813	306	67	3
7	7	3052	0.0419	437	0.0430	-0.323	0.311	2.25	894	1187	678	243	46	4
8	8	3052	0.0400	326	0.0426	-0.312	0.314	2.27	1510	925	368	145	90	14
9	9	3051	0.0380	208	0.0383	-0.333	0.240	2.29	2268	645	30	12	86	10
10	10	3051	0.0343	231	0.0344	-0.333	0.350	2.34	2614	397	3	4	14	18

glmnet : Actual Return Prediction

Step 1 : Finding the best parameters for the model. Therefore, we experiment with different parameters while developing the model.

The trial the parameters with different max.depth= 2, 5 and eta = 0.1, 0.01, 0.001.

	max_depth	eta	bestTree	bestPerf
1	2	0.100	67	0.0855832
2	5	0.100	51	0.0857738
3	2	0.010	500	0.0856386
4	5	0.010	500	0.0857788
5	2	0.001	500	0.2881452
6	5	0.001	500	0.2881512

Step 2 : Applying the best parameter to the model

The best set of parameters gave the best performance has the parameters value as follows: nrounds= 67, max.depth= 2, eta= 0.1, objective="reg: squared error"

Step 3 : Comparing the RMSE from Training and Testing dataset

Step 4 : Evaluating the results

predGlmnetTrn, RMSE = 0.0859347

	tile	count	avgPredRet	numDefaults	avgActRet	minRet	maxRet	avgTer	totA	totB	totC	totD	totE	totF
	<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	1	7121	0.0518	2052	0.0558	-0.333	0.412	2.29	0	0	0	3993	2712	354
2	2	7121	0.0495	1622	0.0562	-0.333	0.299	2.24	0	0	2732	4389	0	0
3	3	7121	0.0483	1413	0.0537	-0.322	0.397	2.23	0	0	6382	739	0	0
4	4	7121	0.0474	1246	0.0511	-0.323	0.237	2.24	0	2031	5090	0	0	0
5	5	7121	0.0467	1159	0.0492	-0.323	0.251	2.26	0	2989	4132	0	0	0
6	6	7121	0.0459	1026	0.0465	-0.333	0.243	2.26	0	5601	1520	0	0	0
7	7	7121	0.0449	782	0.0428	-0.325	0.189	2.26	762	6359	0	0	0	0
8	8	7121	0.0441	611	0.0392	-0.323	0.164	2.24	4522	2599	0	0	0	0
9	9	7120	0.0433	511	0.0361	-0.333	0.145	2.28	5669	1451	0	0	0	0
10	10	7120	0.0423	391	0.0336	-0.313	0.153	2.29	7110	6	3	1	0	0

predGlmnetTst, RMSE =0.0840432

	tile	count	avgPredRet	numDefaults	avgActRet	minRet	maxRet	avgTer	totA	totB	totC	totD	totE	totF
	<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	1	3052	0.0517	902	0.0557	-0.322	0.367	2.29	0	0	0	1724	1154	156
2	2	3052	0.0495	732	0.0543	-0.333	0.296	2.25	0	0	1131	1921	0	0
3	3	3052	0.0483	602	0.0555	-0.322	0.252	2.23	0	0	2782	270	0	0
4	4	3052	0.0474	485	0.0561	-0.323	0.304	2.18	0	884	2168	0	0	0
5	5	3052	0.0467	495	0.0503	-0.300	0.242	2.25	0	1313	1739	0	0	0
6	6	3052	0.0459	424	0.0490	-0.323	0.232	2.27	0	2364	688	0	0	0
7	7	3052	0.0449	289	0.0457	-0.310	0.202	2.26	323	2729	0	0	0	0
8	8	3052	0.0441	244	0.0407	-0.333	0.161	2.28	1878	1174	0	0	0	0
9	9	3051	0.0433	203	0.0372	-0.323	0.153	2.25	2443	608	0	0	0	0
10	10	3051	0.0423	152	0.0347	-0.333	0.127	2.32	3044	2	2	3	0	0

Random Forest : Actual Return Prediction

Step 1 : Finding the best parameters for the model. Therefore, we experiment with different parameters while developing the model.

First, trial the parameters with different mtry= 3,5 and num.trees = 100,200

	mtry	num.trees	minRMSE
1	3	100	0.007657401
2	5	100	0.007745478
3	3	200	0.007577299
4	5	200	0.007659333

The best set of parameters is mtry = 3, num.trees = 200 which gave the minimum RMSE

Step 2 : Applying the best parameter to the model

Step 3 : Comparing the RMSE from Training and Testing dataset

Step 4 : Evaluating the results

predRfRet_Trn RMSE = 0.04855733

A tibble: 10 x 14

	tile	count	avgPredRet	numDefaults	avgActRet	minRet	maxRet	avgTer	totA	totB	totC	totD	totE	totF
	<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	1	7121	0.101	10	0.139	0.0708	0.412	1.24	1	253	2467	3079	1120	164
2	2	7121	0.0779	28	0.103	0.0303	0.238	1.79	9	1258	3676	1791	355	28
3	3	7121	0.0678	46	0.0879	0	0.176	2.10	45	2368	3595	918	182	13
4	4	7121	0.0602	67	0.0776	0.0186	0.164	2.25	247	3419	2839	505	105	6
5	5	7121	0.0535	127	0.0689	-0.00941	0.154	2.23	1134	3857	1771	296	55	8
6	6	7121	0.0472	142	0.0593	-0.0613	0.162	2.21	2847	3146	949	146	33	0
7	7	7121	0.0419	166	0.0506	-0.0389	0.136	2.38	4398	2100	497	112	14	0
8	8	7121	0.0366	268	0.0426	-0.0552	0.158	2.62	5323	1291	378	101	23	4
9	9	7120	0.0211	2844	0.00912	-0.230	0.136	2.78	3376	1746	1190	610	175	20
10	10	7120	-0.0534	7115	-0.174	-0.333	0.103	3.00	683	1598	2497	1564	650	111

predRfRet_Tst RMSE = 0.08529379

A tibble: 10 x 14

	tile	count	avgPredRet	numDefaults	avgActRet	minRet	maxRet	avgTer	totA	totB	totC	totD	totE	totF
	<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	1	3052	0.0762	667	0.0562	-0.322	0.301	2.23	0	277	1287	1148	287	44
2	2	3052	0.0620	542	0.0544	-0.322	0.304	2.23	1	943	1447	528	121	11
3	3	3052	0.0548	503	0.0532	-0.322	0.366	2.23	38	1284	1210	403	99	17
4	4	3052	0.0493	425	0.0524	-0.322	0.320	2.23	379	1319	960	304	84	6
5	5	3052	0.0449	431	0.0469	-0.323	0.277	2.24	882	1125	733	238	65	7
6	6	3052	0.0412	343	0.0448	-0.313	0.296	2.26	1329	959	535	174	50	5
7	7	3052	0.0377	319	0.0420	-0.299	0.367	2.28	1600	767	472	153	54	5
8	8	3052	0.0338	324	0.0416	-0.333	0.277	2.29	1581	745	487	181	51	7
9	9	3051	0.0279	418	0.0423	-0.333	0.250	2.27	1252	858	587	257	90	6
10	10	3051	0.0111	556	0.0455	-0.333	0.350	2.32	626	797	792	532	253	48

Compare model performance.
RMSE is the indicator of model performance.

Model and Parameters	**RMSE of Testing Data**
Xgboost : max_depth= 2, eta = 0.1	0.13122
Glmnet : nrounds= 67, max.depth= 2, eta= 0.1	0.08404
Random Forest : mtry = 3, num.trees = 200	0.08529

Comparing the results from gbm, glm, and random forest models, we learn that the glmnet model gives the best accuracy on the Test dataset.
However, when we are looking at samples of data and range of predicted values of glmnet, we found that glmnet gave the narrow range.

Glmnet range is between (0.041,0.058) with mean of 0.046
Random forest is between (-0.17,0.06) with mean of 0.046
Compared to actual data which range is between (-0.33,0.06) with mean of 0.46

Therefore, **random fores**t is the **best accuracy** for annRet prediction.
These are examples of predicted actual returns**.** (using function predict to predict the values from the best model). We skim the data first 20 rows from both Training and Testing dataset. We found that Prd_annRet_rf are closer to actualReturn than the value from glmnet (Prd_annRet_rf)

Training Dataset

	grade	loan_status	actualTerm	actualReturn	Prd_annRet_rf	Prd_annRet_glm	int_rate	Prd_loan_status
1	D	Fully Paid	2.16290212	0.12579769	0.07713388	0.05120637	17.86	0.4572602
2	A	Fully Paid	3.00068446	0.03123179	0.03348379	0.04286415	5.93	0.6401974
3	B	Fully Paid	3.00068446	0.06231854	0.04312106	0.04640963	11.53	0.2662332
4	C	Fully Paid	0.58590007	0.13212114	0.08980002	0.04807582	14.65	0.5560874
5	A	Fully Paid	3.00068446	0.04193758	0.03705662	0.04295323	7.89	0.5291126
6	A	Fully Paid	3.00068446	0.03360981	0.02952794	0.04181656	6.39	0.4719124
7	C	Charged Off	3.00000000	-0.07146854	-0.02362993	0.04681791	12.99	0.5393276
8	B	Charged Off	3.00000000	0.05780243	0.06318853	0.04606013	11.99	0.3406623
9	C	Fully Paid	3.00068446	0.06776602	0.05440024	0.04768362	12.29	0.2792305
10	C	Fully Paid	1.41273101	0.10638889	0.06263331	0.04728865	12.69	0.5476520
11	B	Charged Off	3.00000000	-0.12971736	-0.04798672	0.04669851	10.99	0.3933894
12	C	Fully Paid	2.07802875	0.09297532	0.06357937	0.04821407	12.99	0.8524442
13	C	Fully Paid	2.24777550	0.08413180	0.06150782	0.04636324	12.39	0.8479876
14	A	Fully Paid	0.75290897	0.07336117	0.05240442	0.04295323	7.89	0.4504871
15	C	Fully Paid	0.16700890	0.16057827	0.09189710	0.04897185	13.99	0.5305197
16	C	Fully Paid	3.00068446	0.06665658	0.06586929	0.04768362	12.29	0.7286432
17	B	Fully Paid	3.00068446	0.05946097	0.04893596	0.04669851	10.99	0.8781974
18	A	Fully Paid	1.42094456	0.05885952	0.05036121	0.04387199	7.26	0.3640675
19	C	Fully Paid	1.25393566	0.12342097	0.06541992	0.04807582	14.65	0.7062856
20	C	Fully Paid	0.33401780	0.15420136	0.08806076	0.04847171	13.33	0.5276764

Testing Dataset

	grade	loan_status	actualTerm	actualReturn	Prd_annRet_rf	Prd_annRet_glm	int_rate	Prd_loan_status
1	D	Fully Paid	2.1629021	0.12579769	0.07713388	0.05120637	17.86	0.4572602
2	A	Fully Paid	3.0006845	0.03123179	0.03348379	0.04286415	5.93	0.6401974
3	B	Fully Paid	3.0006845	0.06231854	0.04312106	0.04640963	11.53	0.2662332
4	C	Fully Paid	0.5859001	0.13212114	0.08980002	0.04807582	14.65	0.5560874
5	A	Fully Paid	3.0006845	0.04193758	0.03705662	0.04295323	7.89	0.5291126
6	A	Fully Paid	3.0006845	0.03360981	0.02952794	0.04181656	6.39	0.4719124
7	C	Charged Off	3.0000000	-0.07146854	-0.02362993	0.04681791	12.99	0.5393276
8	B	Charged Off	3.0000000	0.05780243	0.06318853	0.04606013	11.99	0.3406623
9	C	Fully Paid	3.0006845	0.06776602	0.05440024	0.04768362	12.29	0.2792305
10	C	Fully Paid	1.4127310	0.10638889	0.06263331	0.04728865	12.69	0.5476520
11	B	Charged Off	3.0000000	-0.12971736	-0.04798672	0.04669851	10.99	0.3933894
12	C	Fully Paid	2.0780287	0.09297532	0.06357937	0.04821407	12.99	0.8524442
13	C	Fully Paid	2.2477755	0.08413180	0.06150782	0.04636324	12.39	0.8479876
14	A	Fully Paid	0.7529090	0.07336117	0.05240442	0.04295323	7.89	0.4504871
15	C	Fully Paid	0.1670089	0.16057827	0.09189710	0.04897185	13.99	0.5305197
16	C	Fully Paid	3.0006845	0.06665658	0.06586929	0.04768362	12.29	0.7286432
17	B	Fully Paid	3.0006845	0.05946097	0.04893596	0.04669851	10.99	0.8781974
18	A	Fully Paid	1.4209446	0.05885952	0.05036121	0.04387199	7.26	0.3640675
19	C	Fully Paid	1.2539357	0.12342097	0.06541992	0.04807582	14.65	0.7062856
20	C	Fully Paid	0.3340178	0.15420136	0.08806076	0.04847171	13.33	0.5276764

Do you find larger training sets to give better models ?

Due to the minimum RMSE given from the Glmnet, we're trying on Glmnet with Training Dataset : Testing Dataset of 50:50

Training, RMSE = 0.08604398

	tile	count	avgPredRet	numDefaults	avgActRet	minRet	maxRet	avgTer	totA	totB	totC	totD	totE	totF
	<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	1	5087	0.0483	1567	0.0509	-0.333	0.412	2.33	0	0	0	2845	1945	258
2	2	5087	0.0475	1150	0.0578	-0.322	0.334	2.25	0	0	1405	3682	0	0
3	3	5087	0.0471	1042	0.0532	-0.323	0.285	2.24	0	0	5087	0	0	0
4	4	5086	0.0468	937	0.0497	-0.323	0.237	2.24	0	0	5086	0	0	0
5	5	5086	0.0465	828	0.0489	-0.333	0.251	2.24	0	2546	2540	0	0	0
6	6	5086	0.0462	657	0.0503	-0.301	0.197	2.24	0	5086	0	0	0	0
7	7	5086	0.0458	572	0.0429	-0.323	0.187	2.26	0	5086	0	0	0	0
8	8	5086	0.0454	456	0.0390	-0.313	0.164	2.26	2751	2335	0	0	0	0
9	9	5086	0.0452	370	0.0360	-0.333	0.127	2.25	5086	0	0	0	0	0
10	10	5086	0.0449	186	0.0352	-0.303	0.122	2.26	5080	4	2	0	0	0

Tresting, RMSE = 0.08493892

	tile	count	avgPredRet	numDefaults	avgActRet	minRet	maxRet	avgTer	totA	totB	totC	totD	totE	totF
	<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	1	5087	0.0483	1508	0.0547	-0.322	0.367	2.31	0	0	0	2873	1921	252
2	2	5087	0.0475	1203	0.0557	-0.333	0.397	2.24	0	0	1451	3636	0	0
3	3	5087	0.0471	1069	0.0521	-0.323	0.277	2.26	0	0	5087	0	0	0
4	4	5086	0.0468	892	0.0524	-0.322	0.304	2.24	0	0	5086	0	0	0
5	5	5086	0.0465	795	0.0506	-0.323	0.232	2.25	0	2464	2622	0	0	0
6	6	5086	0.0463	649	0.0513	-0.325	0.218	2.24	0	5086	0	0	0	0
7	7	5086	0.0458	532	0.0445	-0.312	0.202	2.26	0	5086	0	0	0	0
8	8	5086	0.0454	413	0.0401	-0.333	0.153	2.27	2673	2413	0	0	0	0
9	9	5086	0.0452	310	0.0376	-0.323	0.161	2.27	5086	0	0	0	0	0
10	10	5086	0.0449	205	0.0345	-0.333	0.111	2.29	5075	4	3	4	0	0

The RMSE of the model on test data is slightly higher than separating data into 70:30. Therefore, taking smaller or larger training samples doesn't make a significant difference on the RMSE. However, the higher data in training dataset gives higher accuracy on testing data because the model learns better with higher exposure to the data.

3. 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?

According to loan status prediction and annRet prediction model in previous part.
GBM is our best loan_status prediction model and Random forest is the best for annRet prediction.

Our strategy for client selection is to consider the overall profit that we could possibly make on this model. We consider 2 points Revenue and Expense of this lending club.
Revenue is basically from the int_rate that the lending club can collect from their clients.
Expense is the amount of money that possibly gets default in the future loans.

expReturn = p("Fully Paid") * (total_paymnt - funded_amoint)- p("Default") * funded_amount
expReturn = p("Fully Paid") * (annRet * loan_amnt)- p("Default") * funded_amount
expReturn = Revenue -Expense

Revenue Calculation

From question 1, we build the loan_status prediction model which GBM is the best prediction model from our assumption. We got the score (probability of being "Fully paid") for every observation which indicates the probability of getting paid back. For example, customer no.1 gets a score of 0.6 on his loan status prediction. This means this customer has potentially not going to pay back for 40%.

From question 2, we build the annual return prediction model which cv.glmnet is the best prediction model. What we got is the actual return prediction (IRR for this business model).

If we combine the results from status prediction and predicted actual return together by multiplying these values together. This will reflect the real annual return or "expReturn" that we already accompany by the probability of getting default into this value.

#exp_Rev = p("Fully Paid") * actualRetFP+ p("Default") * actualRetDef
exp_Rev = p("Fully Paid") * predicted actual return
In this case we cannot collect any actualRetDef from customers who are getting default.

However, this is going to be our “Revenue” because we only look at the interest rate that we will collect from custumers. In order to calculate the “Profit” , we need to incorporate the loss from being unable to collect the remaining outstanding for each customer.

Customer A has who gets the predicted loan status of 0.6 and predicted annRet of 0.04.
His expReturn = 0.04*0.6 = 0.024.
This value represents the actual percentage that we can make money from this customer. (2.4% instead of 4% because it incorporated the loss of being default)

If we combine both results from the prediction models together. The results are going to be more comprehensible which reflect truly revenue you are going to make from this lending club model.

Expense calculation

Normally, when we evaluate the profit that we’re going to make from the project that we construct we should look at the expected loss on the portfolio as well.

According to the revenue calculation, we didn’t include the expected loss (EL) on it.
Basically, when we calculate the expected loss on this portfolio, we will include the probability default (PD) in our calculation which can be calculated from results from the status prediction model.
The probability shows the potential of being paid for this loan (“Fully paid”).
Therefore, we can compute the probability default (PD)

PD = 1 - p(“Fully paid”)

The formula of Expected Loss (EL) should be calculated from Exposure at default (EAD) combined with probability of getting default (PD) and unrecovered of the underlying asset of each loan (LGD)

- EAD we will assume that the total money we can lose for each person is equal to the amount of loan at the beginning of the loan. (loan amount)
- We wouldn’t include the LGD term in our calculation

Expert =pred*actret
expReturn = p(“Fully Paid”) * (total_paymnt - funded_amount)- p(“Default”) * funded_amount
which is same as
expReturn = p(“Fully Paid”) * (annRet * loan_amnt)- p(“Default”) * funded_amount
In our case we have used Actual_returns instead of annual returns
expReturn = Revenue - Expense

How does performance here compare with use of single models?

> mean(expected_single$EXP_rf)
[1] -4015.968
> mean(exp_combined$EXP_comb)
[1] -5405.717

> sd(expected_single$EXP_r)
[1] 3580.073
> sd(exp_combined$EXP_comb)
[1] 4151.294

Expected return from Single model is higher than Expected return from combined model

If we take a look at the default rate (actual proportion of getting default) compared to the predDefaultRate that we gain from gbm status prediction model. The value shifts from each grade very much which I think that our prediction model might not accurately reflect the correct prediction result. However, predicting annual returns from random forests is close to reality.

Training

	grade	count	numDefaults	ActDefaultRate	PredDefaultRate	avgActRet	avgPredRet_rf
	<fct>	<int>	<int>	<dbl>	<dbl>	<dbl>	<dbl>
1	A	18063	1069	0.0592	0.503	0.0364	0.0359
2	B	21036	2593	0.123	0.502	0.0456	0.0444
3	C	19859	3769	0.190	0.502	0.0517	0.0504
4	D	9122	2349	0.258	0.503	0.0550	0.0538
5	E	2712	862	0.318	0.498	0.0507	0.0490
6	F	354	150	0.424	0.502	0.0462	0.0438
7	G	62	21	0.339	0.491	0.0754	0.0681

Testing

grade	count	numDefaults	ActDefaultRate	PredDefaultRate	avgPredRet	avgActRet	avgPredRet_rf
<fct>	<int>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
A	7688	417	0.0542	0.755	0.755	0.0375	0.0356
B	9074	1064	0.117	0.575	0.575	0.0473	0.0431
C	8510	1575	0.185	0.447	0.447	0.0538	0.0485
D	3918	1026	0.262	0.356	0.356	0.0548	0.0513
E	1154	376	0.326	0.304	0.304	0.0542	0.0462
F	156	64	0.410	0.275	0.275	0.0530	0.0408
G	18	6	0.333	0.342	0.342	0.0895	0.0629

how would you select loans for investment?

	tile	count	avgPredRet_rf	numDefaults	avgActRet	avgTer	totA	totB	totC	totD	totE	totF
	<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	1	3052	0.0636	329	0.0542	2.18	774	1132	806	269	61	7
2	2	3052	0.0528	279	0.0493	2.20	1385	885	543	202	33	3
3	3	3052	0.0505	344	0.0456	2.27	1251	985	572	208	32	4
4	4	3052	0.0495	395	0.0451	2.26	1056	1020	686	248	36	5
5	5	3052	0.0477	397	0.0472	2.24	882	1011	833	276	49	1
6	6	3052	0.0458	461	0.0462	2.27	743	1017	882	323	79	6
7	7	3052	0.0434	465	0.0487	2.28	567	911	1053	408	97	13
8	8	3052	0.0392	554	0.0467	2.28	486	828	1105	479	131	21
9	9	3051	0.0330	602	0.0477	2.28	324	727	1098	656	216	27
10	10	3051	0.0133	702	0.0485	2.32	220	558	932	849	420	69

D5 the average term is less than others which mean customers take less time to pay back loans. Plus, the average actual return in D5 is higher than overall average on this portfolio.

The average actual return in D7 has little peak compared to neighboring D6,D8. Plus, the average actual return in D7 is higher than overall average on this portfolio. These could be targeted customers in future.

For more details of information, we should zoom in to each decile and map with the existing data in order to get the characteristics of customers that we are going to target on.

According to variable importance that we found in the GBM model (status prediction) the ranks of importance are accordingly; Sub_grade, Int_rate, Loan_amnt, Purpose.

Above data could tell us only the details of customers in grades perspective. To be more specific on customers' characteristics, we should map the information back to existing data by referring to the variable importance which we already got from the status prediction model. For example, D5. We can compute the average Int_rate, Loan_amnt and count number of customers by purpose. Therefore, we will have general ideas for customer selection.

4. 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. Compare performance of models from different methods (glm, gbm, rf). Can this provide a useful approach for investment? Compare performance with that in Question 3.

Definitely, focussing on a particular group of customers will provide you a higher return.
For example, lower grade loans (C and below)

We initially removed Grade A and Grade B.
Then we checked the default rate, actual returns and other details for remaining grades.

	grade	nLoans	defaults	defaultRate	avgInterst	avgLoanAmt	avgRet	avgActualRet	avgActualTerm	minActualRet	maxActualRet
1	C	28369	5344	18.83746	13.34365	11738.75	2.3298103	5.237076	2.236987	-32.26708	39.72585
2	D	13040	3375	25.88190	16.61591	12445.83	2.0506539	5.495385	2.279187	-33.33333	36.05423
3	E	3866	1238	32.02276	19.30809	13559.79	1.3926051	5.175158	2.324470	-33.33333	41.19355
4	F	510	214	41.96078	23.89739	11109.61	0.6186133	4.829171	2.417675	-32.06218	36.56101
5	G	80	27	33.75000	25.90050	9875.00	2.3601434	7.857836	2.268215	-24.71439	30.87502

From the table observation it is clear that # Grade F has a high default rate of 41.96078 % and average actual returns are also less compared to Grades C,D,E and G
Therefore, we can remove F grade.

Now lets develop a gbm model
set.seed(1789)
```
> gbm_MM_LG <- gbm(loan_status~.,
+              data = lcdfTrn_LG, distribution = "bernoulli",
+              n.trees=1500, shrinkage = 0.01,
+              interaction.depth=10, bag.fraction=1,
+              cv.folds=5, n.cores=16)
```

The best cross-validation iteration was 1500.

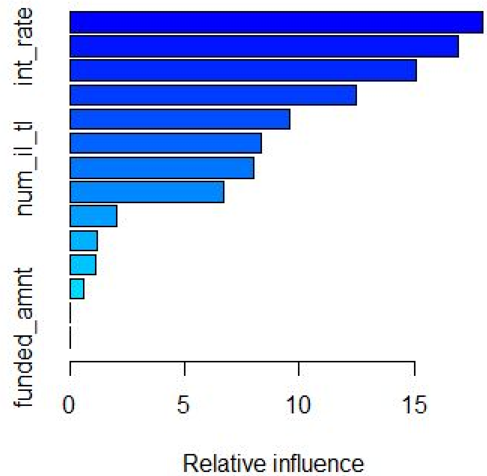
Following fig. gives the variable importance

```
> summary(gbm_MM_LG, cbars=TRUE)
                                       var      rel.inf
sub_grade                        sub_grade 17.98044194
int_rate                          int_rate 16.89793926
loan_amnt                        loan_amnt 15.09246897
purpose                            purpose 12.45060796
num_sats                          num_sats  9.54463724
num_rev_accts                num_rev_accts  8.35643734
num_il_tl                        num_il_tl  7.99317742
home_ownership              home_ownership  6.69497757
verification_status    verification_status  2.04046518
tax_liens                        tax_liens  1.16292827
pub_rec_bankruptcies pub_rec_bankruptcies  1.13415242
initial_list_status    initial_list_status  0.61914876
grade                                grade  0.03261765
funded_amnt                    funded_amnt  0.00000000
```

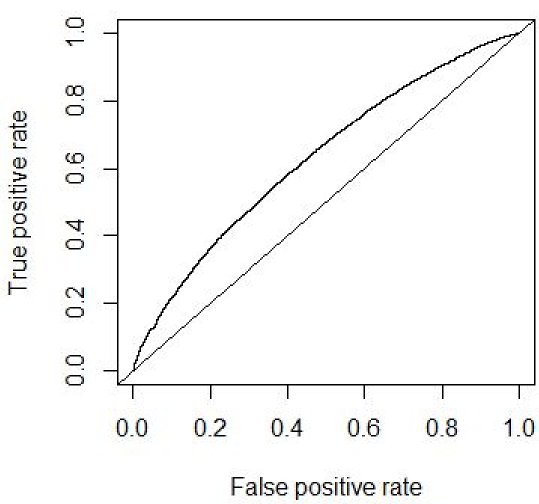
Following variables come out to be the most important variables:

1. sub_grade
2. Int_rate
3. Loan_amnt
4. Purpose

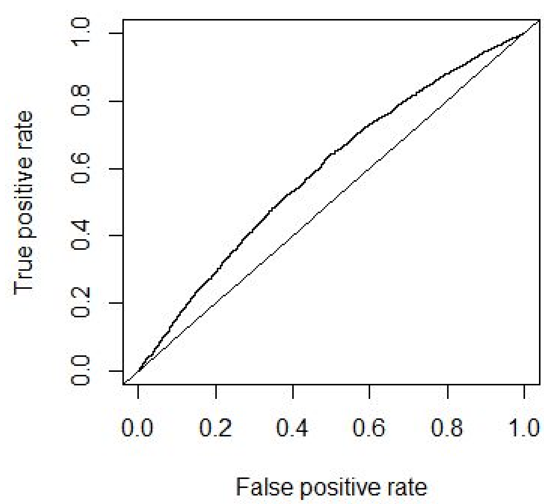
Following fig. gives the variable importance



ROC Curve Train dataset AUC value: 0.6305248



ROC Curve for Test dataset, AUC value = 0.5906961



We deleted loan grade A,B and performed the Random Forest model.

```
set.seed(1789)
rf_act = ranger(actualReturn ~ .,
            data = lcdfTrn_rf,
            mtry = 3,
            num.trees = 200,
            importance ='impurity')
```

Random forest Trn: RMSE = 0.06035093

	tile	count	avgPredRet	numDefaults	avgActRet	minRet	maxRet	avgTer	totA	totB	totC	totD	totE	totF
	<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	1	3211	0.113	3	0.159	0.0872	0.397	0.980	0	0	698	1625	736	122
2	2	3211	0.0910	6	0.123	0.0704	0.222	1.52	0	0	1484	1328	366	31
3	3	3211	0.0817	15	0.108	0.00592	0.190	1.85	0	0	1859	1106	225	20
4	4	3211	0.0749	20	0.0993	0.0333	0.206	2.09	0	0	2192	834	175	10
5	5	3211	0.0690	34	0.0908	-0.00871	0.181	2.34	0	0	2413	666	124	8
6	6	3210	0.0632	44	0.0848	0.0101	0.182	2.51	0	0	2601	518	85	6
7	7	3210	0.0564	105	0.0803	-0.00605	0.180	2.62	0	0	2659	459	84	6
8	8	3210	0.0430	675	0.0639	-0.127	0.151	2.70	0	0	2552	514	130	13
9	9	3210	-0.00824	3038	-0.0793	-0.266	0.142	2.96	0	0	1884	981	296	47
10	10	3210	-0.0712	3210	-0.203	-0.333	-0.0665	3	0	0	1538	1108	468	84

Random forest Tst: RMSE = 0.108995

A tibble: 10 x 14

	tile	count	avgPredRet	numDefaults	avgActRet	minRet	maxRet	avgTer	totA	totB	totC	totD	totE	totF
	<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	1	1376	0.0844	330	0.0598	-0.310	0.328	2.25	0	0	460	672	207	26
2	2	1376	0.0704	297	0.0543	-0.311	0.277	2.21	0	0	845	421	98	9
3	3	1376	0.0638	268	0.0584	-0.310	0.314	2.25	0	0	881	386	100	8
4	4	1376	0.0586	281	0.0580	-0.312	0.287	2.24	0	0	930	342	97	7
5	5	1376	0.0537	280	0.0545	-0.322	0.354	2.25	0	0	964	316	83	12
6	6	1376	0.0488	301	0.0513	-0.311	0.367	2.28	0	0	947	333	79	15
7	7	1376	0.0433	299	0.0510	-0.323	0.263	2.27	0	0	974	312	79	8
8	8	1376	0.0368	306	0.0548	-0.323	0.318	2.28	0	0	908	356	96	15
9	9	1376	0.0277	325	0.0517	-0.333	0.333	2.25	0	0	866	368	118	21
10	10	1376	0.00694	361	0.0446	-0.328	0.412	2.35	0	0	714	395	220	42

The table below shows the results of the Random forest model considering loan grade of C-F.

	tile	count	avgPredRet	numDefaults	ActualDefaultRate	avgActRet	minRet	maxRet	avgTer	totA	totB	totC	totD	totE	totF
	<int>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	1	1376	0.0844	330	0.240	0.0598	-0.310	0.328	2.25	0	0	460	672	207	26
2	2	1376	0.0704	297	0.216	0.0543	-0.311	0.277	2.21	0	0	845	421	98	9
3	3	1376	0.0638	268	0.195	0.0584	-0.310	0.314	2.25	0	0	881	386	100	8
4	4	1376	0.0586	281	0.204	0.0580	-0.312	0.287	2.24	0	0	930	342	97	7
5	5	1376	0.0537	280	0.203	0.0545	-0.322	0.354	2.25	0	0	964	316	83	12
6	6	1376	0.0488	301	0.219	0.0513	-0.311	0.367	2.28	0	0	947	333	79	15
7	7	1376	0.0433	299	0.217	0.0510	-0.323	0.263	2.27	0	0	974	312	79	8
8	8	1376	0.0368	306	0.222	0.0548	-0.323	0.318	2.28	0	0	908	356	96	15
9	9	1376	0.0277	325	0.236	0.0517	-0.333	0.333	2.25	0	0	866	368	118	21
10	10	1376	0.00694	361	0.262	0.0446	-0.328	0.412	2.35	0	0	714	395	220	42

Range of predicted annual return from Random Forest model is (-0.0897,0.129) with average of 0.0439

We ranked the table by predicted annual return. Then we evaluate the result by taking a look at the 3rd,4th decile which have higher return and lower default number on these deciles compared to others.

Recall the variable importance that we found in the GBM model (status prediction) the ranks of importance are accordingly; Sub_grade, Int_rate, Loan_amnt, Purpose.

The table below presents the customers in D3,4 group by purpose and their grades details.

```
d34 <- pred_rf_Ret_Tst %>% filter(tile>=3,tile<=4) #Select tile 3,4
d34 %>% group_by(purpose) %>% summarise(count=n(),
                                numDefaults=sum(loan_status=="Charged Off"),
                                ActualDefaultRate = sum(loan_status=="Charged Off")/n(),
                                avgActRet=mean(actualReturn),
                                avgPredRet=mean(predicted_Ret_Tst),
                                avgInt=mean(int_rate),
                                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"))
```

	purpose	count	numDefaults	ActualDefaultRate	avgActRet	avgPredRet	avgInt	avgTer	totA	totB	totC	totD	totE	totF
	<fct>	<int>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>	<int>
1	car	13	3	0.231	0.0367	0.0611	15.6	2.44	0	0	6	4	3	0
2	credit_card	336	64	0.190	0.0595	0.0605	14.2	2.25	0	0	247	71	17	1
3	debt_consolidation	1809	366	0.202	0.0569	0.0613	14.6	2.22	0	0	1207	471	126	5
4	home_improvement	159	28	0.176	0.0634	0.0614	14.6	2.31	0	0	110	35	13	1
5	house	22	2	0.0909	0.107	0.0617	16.2	1.45	0	0	11	5	4	1
6	major_purchase	37	10	0.270	0.0522	0.0609	15.0	2.32	0	0	20	16	1	0
7	medical	39	8	0.205	0.0564	0.0603	15.1	2.50	0	0	23	13	3	0
8	moving	43	10	0.233	0.0440	0.0610	15.8	2.43	0	0	19	20	3	1
9	other	225	44	0.196	0.0609	0.0611	15.0	2.34	0	0	133	72	16	4
10	renewable_energy	4	1	0.25	0.0896	0.0630	18.2	1.66	0	0	1	1	2	0
11	small_business	35	9	0.257	0.0540	0.0617	16.9	2.57	0	0	12	15	6	2
12	vacation	30	4	0.133	0.0759	0.0611	14.3	2.26	0	0	22	5	3	0

There are 4 interesting points on this table

1) The red arrow points to the largest proportion of customers in D3,4 whose loan purpose is for debt consolidation.
2) The yellow arrow points to the second largest proportion of customers in D3,4 whose loan purpose is for credit cards.
3) The green arrow points to the lowest average length of term. Customers who have intention to buy houses and accommodations tend to pay back to Lending club earlier than other purposes. Plus, they contribute a large return percentage to Lending club and have very low default rates among other purposes.

4) The blue arrow points to customers who have a purpose on renewable energy. This is the second highest actual return rate. It’s also interesting to promote this group of customers to our next portfolio.

1), 2) which have purposes for debt consolidation and credit cards will be our target customer for the next lending project. (large numbers customers, higher possibility to find customers with these characteristic)
3),4) which have purposes for housing and renewable energy will be our target customer for the next lending project. (high return, should focus and advertise these customers in order to attend our Lending club)