Statistics 154 Project Report

Lending Club Loan Analysis

Group Member:
Yijun Xu (26835278),
Tianshu Zhao(3032271997, ariel1995zhao@berkeley.edu),
Jiahua Zou(26209413, jiahua.zou@berkeley.edu)

1. Introduction

Lending Club is a peer-to-peer lending company based in U.S. and headquartered in San Francisco. As the first peer-to-peer lender to register offerings as securities with the SEC, Lending Club operates an online lending platform that enables borrowers to obtain a loan, and investors to purchase notes and earn a profit depending on the risk they take. Lending club serves as a bridge between investors and borrowers, by reducing the costs of lending and borrowing for individuals with advanced data analytics. With an increasing trend of big data, Lending Club serves as an exemplary case in exploring how data analytics play an important role in providing valuable business insights.

In this project, we explore data published by Lending Club and partially organized by Wendy Kan and published on Kaggle.com[1]. The dataset primarily contains three types of loans: previously completed, late or default and current loans. In order to predict the current loans based on previous data in terms of their risk of defaulting in the future, we are going to classify the loans into good loans vs. bad loans, and use the rest of variables as predictors to find the important key metrics that influence loan rates, analyze how each important predictor affect loan rate prediction.

Due to the binary nature of our response variable, we are going to employ classification methods include: logistic regression, support vector machine, classification tree, random forest, principal component regression and partial least squares. We also use cross validation method along with the models mentioned above to adjust number of parameters used and to minimize MSE and test error rate. Upon using different machine learning methods, we hope to compare the error rate and interpretability resulted from each method, while uncover the determinants in predicting loan rates.

2. Data Reorganization

2.1 Data Cleaning

Our data contains complete information about all loans issued from 2007 to 2015 by Lending Club. It contains 887379 observations and 79 columns, other than assigned loan grades, it also exhibits additional features such as credit scores, number of finance inquiries, annual income, etc. Furthermore, our dataset contains irrelevant information (to the model) such as the

[1] https://www.kaggle.com/wendykan/lending-club-loan-data

loaner's job title, zip code and usage for money. While our project only focuses on finding predictors for loan rate, it is essential for us to clean irrelevant variables and reduce execution time.

First, we delete the variables that contain too many NAs, and set the threshold value to 90%. Second, we delete the variables with too complete separation. That is, variables with only 1 factor level, such as application type " Individuals" and so on. Next, we delete the variables that are strings, such as loans' job title and loan usage description, along with the irrelevant information like loan's member id, address, zip code, etc. The purpose of this step is to wipe off the nuisance factors that could potentially influence the final performance of our results. Most importantly, we delete the variables that are linearly dependent with each others. Those factors could lead inaccuracy of our model, therefore it is very essential to remove the dependent variables to avoid imprecision. Finally, we filter out the rows that have NA values in any of the remaining variables in order to prevent missing values.

2.2 Reclassify Data

We classify the "Fully paid" loan as good loan. "Late (16-30 days)" , "Late (31-120) days" , "charged off", "Default" loan as bad loan, " Issued" loan as current in progress loan. To make the dataset easy to manipulate,we created a new column in the original dataset we define "good loan" index as 1, "bad loan" index as 0, "current loan" index as 2. In the following part of our report, we continue to use those index in visualization and analysis process.

3. Data Visualization and Analysis

3.1 Logistic Regression

Logistic regression requires the following assumptions to hold: First, dependent variable is binary and has mutually exclusive and exhaustive category. In our case, since y is loan_rate and only have three kinds of values, and only two kinds of values, 0 (bad loan) and 1 (good loan), are used in modeling, this assumption is satisfied. Please refer to how we generate loan_rate from loan_status in above section for more detail. Second, observations are supposed to be independent and identically distributed, which, according to Kaggle and lending club's data description, seems to be the case. So this is satisfied as well. Last, any continuous independent variable is linearly related to the logit transformation of the dependent variable. After cleaning the data, we do believe the variables remaining have a linear relationship to the log odds. Thus, all distribution assumptions for logistic regression are held.

In practice, logistic regression gives a pretty good result. Most of the variables are statistically significant with p value less than 0.05. We can see from Figure 3.1.1 that int_rate (interest rate of the loan) has one of the largest coefficient among others. This seems to suggest that the higher the interest rate that the borrower applies for, the more likely the loan is going to end up being a bad loan.

```
Call:
glm(formula = loan_rate ~ ., family = binomial, data = trainset)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.0322  -0.9056   0.5950   0.7818   4.7886

Coefficients:
                              Estimate Std. Error z value Pr(>|z|)
(Intercept)                  3.367e+00  3.678e-02  91.537  < 2e-16 ***
loan_amnt                   -3.203e-04  3.972e-04  -0.806  0.42004
funded_amnt                  1.454e-03  4.605e-04   3.157  0.00159 **
funded_amnt_inv             -1.215e-03  2.312e-04  -5.256 1.47e-07 ***
int_rate                    -1.078e-01  1.663e-03 -64.820  < 2e-16 ***
installment                  1.778e-03  8.138e-05  21.850  < 2e-16 ***
dti                         -3.396e-02  8.251e-04 -41.159  < 2e-16 ***
delinq_2yrs                 -1.020e-01  7.090e-03 -14.380  < 2e-16 ***
inq_last_6mths              -2.677e-02  5.954e-03  -4.496 6.92e-06 ***
open_acc                    -3.197e-02  1.756e-03 -18.207  < 2e-16 ***
pub_rec                     -5.060e-02  1.222e-02  -4.141 3.46e-05 ***
revol_bal                   -6.037e-06  9.567e-07  -6.310 2.79e-10 ***
revol_util                  -2.765e-03  3.755e-04  -7.364 1.78e-13 ***
total_acc                    1.863e-02  7.478e-04  24.914  < 2e-16 ***
total_rec_int                8.251e-05  3.943e-06  20.927  < 2e-16 ***
total_rec_late_fee          -8.738e-02  1.705e-03 -51.263  < 2e-16 ***
collections_12_mths_ex_med -3.251e-01  5.238e-02  -6.207 5.41e-10 ***
acc_now_delinq              -1.620e-01  7.993e-02  -2.027  0.04267 *
tot_coll_amt                 5.572e-08  4.133e-07   0.135  0.89276
tot_cur_bal                  1.366e-06  5.506e-08  24.816  < 2e-16 ***
total_rev_hi_lim             6.846e-06  6.705e-07  10.210  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Figure 3.1.1

The test error rate is around 24.85%, indicating that fitting trained model on test set, 24.85% of all test cases are misclassified. Also, we notice that many variables here is not that much significant, so we can consider to remove those insignificant variables in further analysis such as tree methods and random forest in order to reduce workload. This test results also provide as a reference of significant test while assuring the whole model is accurate and meaningful. We can further test the model by running Chi-Squared test to check the P-value and plot the ROC curve.

3.2 Support Vector Machine

We also fit a support vector machine with a radial and a sigmoid kernel, respectively. Both of them are fitted with a 0.1 cost, which is determined in 10-fold cross validation. For radial kernel, our test error rate is around 25.8%, but we misclassified a significant portion of bad loan as good loan. Figure 3.2.1 we can see that radial-kernel svm only classify 3815 bad loan correctly, while missing 46669 bad loans. This could cause a problem, as companies like Lending Club would want to minimize the probability that bad loans are misclassified as good loans.

```
In [45]: true2=df2[-train, "loan_rate"]
         pred2=predict(svmfit, newdata=df2[-train,])
         table(true2, pred2)
```

```
      pred2
true2      0      1
    0   3815  46669
    1    932 134896
```

Figure 3.2.1

On the other hand, for sigmoid kernel, we have higher error rate but lower bad loan misclassification. From figure 3.2.2, we see that It classifies 110444 good loans correctly, which is significantly lower than radial-kernel svm's 134896. However, 15881 out of (15881 + 34604), which is 31% of bad loans are classified correctly, which is significantly higher than radial-kernel svm's 3815, which is 7.5%.

```
In [52]: svmfit3 = svm(loan_rate~., data=df4[train,], kernel = "sigmoid", cost = 0.1)
         y_pred3 = predict(svmfit3, newdata=df4[-train,])
         true3=df2[-train, "loan_rate"]
         pred3=predict(svmfit3, newdata=df2[-train,])
         table(true3, pred3)
         1-(sum(true3 == pred3)/nrow(df2[-train,]))
```

```
      pred3
true3      0      1
    0  15881  34603
    1  25384 110444
```

```
0.321970672849843
```

Figure 3.2.2

A support vector machine plot with loan_amnt (loan amount) and delinq_2yrs (delinquencies in 2 years) as predictors and a sigmoid kernel are presented below (Figure 3.2.3), noted that blue area is region where observations are classified as bad loan while pink area is good loan region. Here we notice that there is a clear boundary between two types of loans, though misclassification rate is still high. The 'X' are support vectors that have been misclassified, while 'O' do not affect our model.

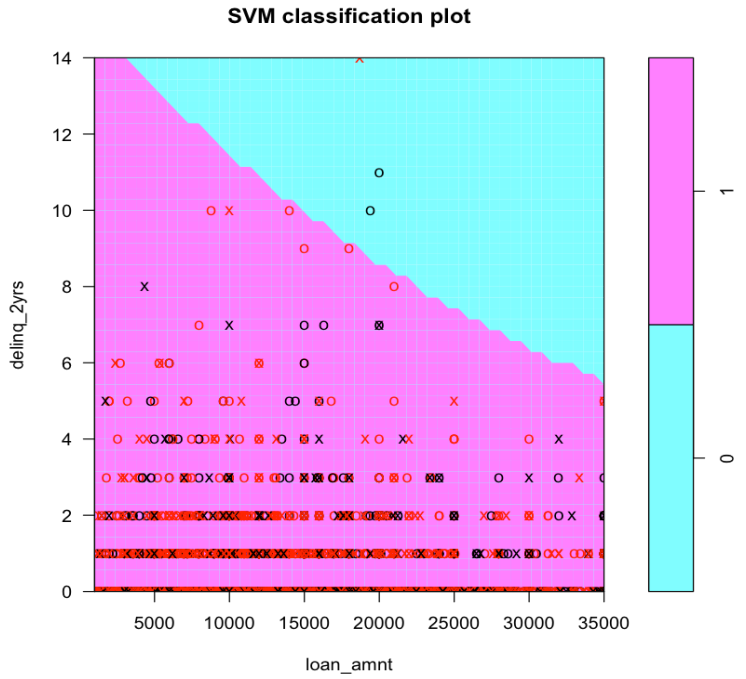


Figure 3.2.3

3.3 Classification Tree

For a response variable with class “good loan” with index 1 and “bad loan” with index 0, we want to organize the dataset into groups by the response variable ---classification tree. When our response variable is instead numeric or continuous we wish to use the data to predict the outcome, and will use classification trees in this situation.Essentially, a classification tree splits the data based on homogeneity, where categorizing based on similar data.

Given factors represents different variables in our dataset (such as loan amount, recoveries rate..), X_1, $X_2 \ldots . X_n$ we want to predict the outcome of interest , Y, which is the loan status good, bad in here. Figure 3.3.1 provides a simplified visualization of this process(since our dataset is too large, this process could take very long time,so here we provide a part of the full tree graph).Figure 3.3.2 provides the full process of tree partitioning.

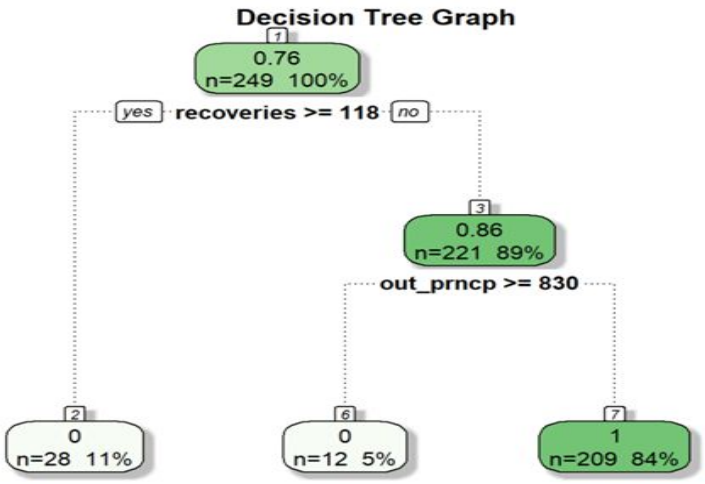


Figure 3.3.1

From Figure 3.3.1, we can see, there are total of 249 sample size, the tree first partitions of the data based on the variable Recoveries (which turns the continuous other variable into a binary variable) “Yes” or “No” with 100% observations. In the second partitions 221 sample with 89% of the total observations remain, the criteria is out_prncp(which is remaining outstanding principal for total amount funded). Based on the criterion above, we classify 16% (11% + 5% = 16%, which is 28+12 = 40 observations)into bad loan, and 89% (which is 209 observations) into good loan.

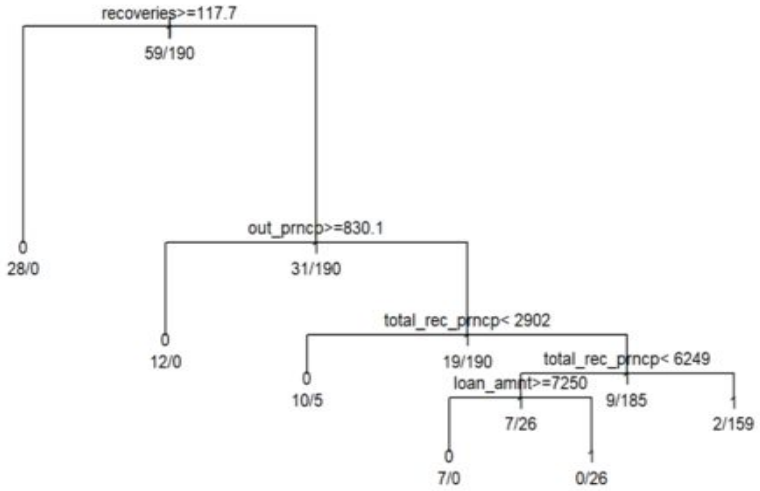


Figure 3.3.2

Since the sample size is 80,000 dataset, which is hard for our laptop to process, here we randomly select 500 rows out of 80,000 to show the result. There is a total of 190 observations, and 59 root node error. For the first partition , we have 59/ 190 = 0.3105 error rate for the first single node tree. Here, 0 , 1 in the bottom is the predicted class for nodes. The first partition split the node error into 28 observations in the left side, and 31 observations in the right side. Performed the splitting process few more time, we get the results above.

```
## Call:
## rpart(formula = loan_rate ~ ., data = non_current_df, method = "class")
##   n= 249
##
##          CP nsplit rel error    xerror       xstd
## 1 0.47457627      0 1.0000000 1.0000000 0.11372374
## 2 0.20338983      1 0.5254237 0.5254237 0.08829933
## 3 0.08474576      2 0.3220339 0.4067797 0.07893056
## 4 0.05932203      3 0.2372881 0.3389831 0.07269107
## 5 0.01000000      5 0.1186441 0.2542373 0.06363585
##
## Variable importance
## collection_recovery_fee             recoveries        total_rec_prncp
##                      16                     16                      8
##             out_prncp          out_prncp_inv            funded_amnt
##                       8                      8                      7
##        funded_amnt_inv              loan_amnt           total_pymnt
##                       7                      7                      7
##            installment        total_pymnt_inv          total_rec_int
##                       5                      5                      4
##
## Node number 1: 249 observations,    complexity param=0.4745763
##   predicted class=1  expected loss=0.2369478  P(node) =1
##     class counts:    59   190
##    probabilities: 0.237 0.763
##   left son=2 (28 obs) right son=3 (221 obs)
##   Primary splits:
##       recoveries             < 117.685  to the right, improve=36.73699, (0 missing)
##       collection_recovery_fee < 2.7377   to the right, improve=36.73699, (0 missing)
##       total_rec_prncp        < 6394.785 to the left,  improve=36.24565, (0 missing)
##       total_pymnt            < 8698.835 to the left,  improve=18.02811, (0 missing)
##       total_pymnt_inv        < 8698.835 to the left,  improve=18.02811, (0 missing)
##   Surrogate splits:
##       collection_recovery_fee < 2.7377   to the right, agree=1.000, adj=1.000, (0 split)
##       total_rec_prncp        < 737.08   to the left,  agree=0.908, adj=0.179, (0 split)
##
```

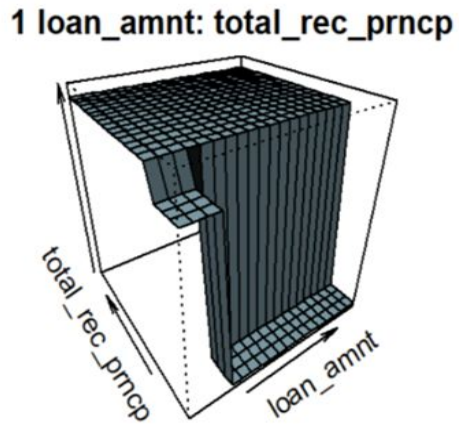
```
Node number 2: 28 observations
  predicted class=0  expected loss=0  P(node) =0.1124498
    class counts:    28     0
   probabilities: 1.000 0.000

Node number 3: 221 observations,    complexity param=0.2033898
  predicted class=1  expected loss=0.1402715  P(node) =0.8875502
    class counts:    31   190
   probabilities: 0.140 0.860
  left son=6 (12 obs) right son=7 (209 obs)
  Primary splits:
      out_prncp       < 830.105  to the right, improve=18.757710, (0 missing)
      out_prncp_inv   < 830.105  to the right, improve=18.757710, (0 missing)
      total_rec_prncp < 7496.215 to the left,  improve=13.424080, (0 missing)
      total_pymnt     < 7948.86  to the left,  improve= 7.837983, (0 missing)
      total_pymnt_inv < 7929.3   to the left,  improve= 7.837983, (0 missing)
  Surrogate splits:
      out_prncp_inv < 830.105  to the right, agree=1, adj=1, (0 split)

Node number 6: 12 observations
  predicted class=0  expected loss=0  P(node) =0.04819277
    class counts:    12     0
   probabilities: 1.000 0.000

Node number 7: 209 observations,    complexity param=0.08474576
  predicted class=1  expected loss=0.09090909  P(node) =0.8393574
    class counts:    19   190
   probabilities: 0.091 0.909
  left son=14 (15 obs) right son=15 (194 obs)
  Primary splits:
      total_rec_prncp < 2901.725 to the left,  improve=10.7138400, (0 missing)
      total_pymnt     < 3785.56  to the left,  improve= 6.3262110, (0 missing)
      total_pymnt_inv < 3768.725 to the left,  improve= 6.3262110, (0 missing)
      dti             < 19.275   to the right, improve= 0.9858538, (0 missing)
      annual_inc      < 65781    to the left,  improve= 0.9814050, (0 missing)
  Surrogate splits:
      total_pymnt     < 3490.614 to the left,  agree=0.986, adj=0.8, (0 split)
      total_pymnt_inv < 3490.615 to the left,  agree=0.986, adj=0.8, (0 split)
      loan_amnt       < 2900     to the left,  agree=0.957, adj=0.4, (0 split)
```

Figure 3.3.3

Further analysis the results of classification tree as Figure 3.3.3 shows above, The overall proportion of bad and good is (0.23,0.763) in the original data. Then it partitions further based on other variables, such as recoveries and so on. The deviation is calculated using the Gini Index. We see the decrease in deviance as we get more precise in partitioning.

The complexity table in Figure 3.3.3 provides information regarding to all of the trees considered for the final model. The above figure shows tree complexity parameter, the number of splits, the resubstitution error rate, the cross-validated error rate, and the associated standard error. We used complexity parameter to control the size of the decision tree and to select the optimal tree size. If the cost of adding another variable to the decision tree from the current node is above the value of cp, then tree building does not continue. The resubstitution rate decreases as you go down the list of trees. The largest tree will always yield the lowest resubstitution error rate. However, choosing the tree with the lowest resubstitution rate is not the optimal choice, as

this tree will have a bias. Larger tree size will put random variation in the predictions as they overfit outliers.




```
##
## Classification tree:
## rpart(formula = loan_rate ~ ., data = non_current_df, method = "class")
##
## Variables actually used in tree construction:
## [1] loan_amnt       out_prncp       recoveries      total_rec_prncp
##
## Root node error: 59/249 = 0.23695
##
## n= 249
##
##         CP nsplit rel error  xerror     xstd
## 1 0.474576      0   1.00000 1.00000 0.113724
## 2 0.203390      1   0.52542 0.52542 0.088299
## 3 0.084746      2   0.32203 0.40678 0.078931
## 4 0.059322      3   0.23729 0.33898 0.072691
## 5 0.010000      5   0.11864 0.25424 0.063636
```

Figure 3.3.4

Figure 3.3.5

Figure 3.3.4 shows the most significant relationships between our variables in 3D plot. The 3D plot allows us to explore the structure of the loan dataset, while developing easy way to visualize decision rules. It also provides the set of splitting rules used to segment the predictor spaces. Figure 3.3.5 indicates that the overall root node error rate is 23.695%

3.4 Random Forest

Random Forest can balance errors in datasets where the classes are imbalanced. Most importantly, it can handle massive datasets with large dimensionality which is a very good choice for large dataset like ours. But we also facing overfitting problem with random forest.

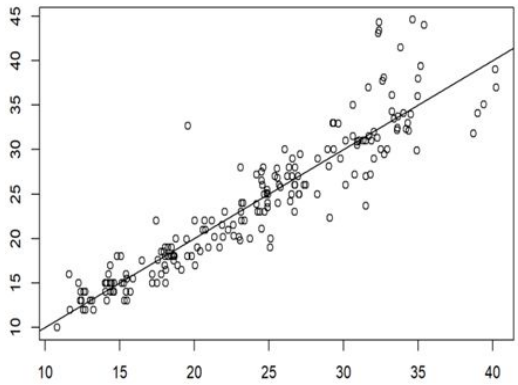


Figure 3.4.1

Looking at the relationship in figure 3.4,1 between predicted and actual test set loan status class, our classification forest with 4 variables is pretty well at predicting the class. Most of the points lie closely around the y = x line, and on the line means the prediction is perfect. We observe there a few outliers, and the model is not very accuracy in predicting the lowest and highest ranges of loans using the random forest.

```
In [31]: importance(rf_model)
```

	MeanDecreaseGini
loan_amnt	3323.0490
funded_amnt	3317.6038
funded_amnt_inv	3640.2764
int_rate	8423.2912
installment	5588.9964
dti	7516.2359
delinq_2yrs	1183.2540
inq_last_6mths	1908.2250
open_acc	4056.0094
pub_rec	844.6215
revol_bal	5978.1019
revol_util	6251.2847
total_acc	4906.5944

Figure 3.4.2

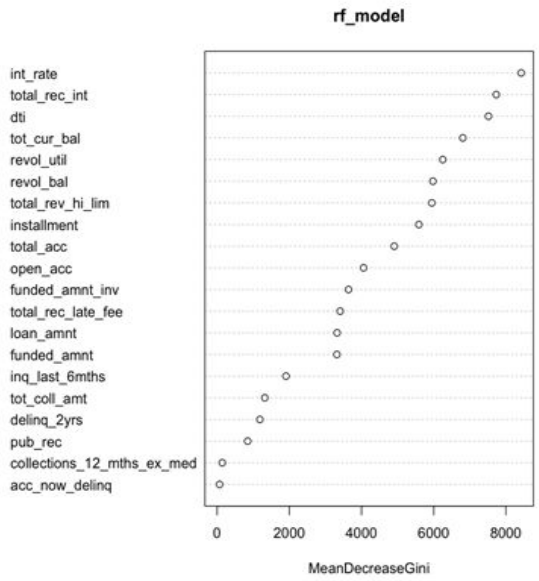


Figure 3.4.3

```
yhat_rf
      0     1
0  2430  8812
1  1145 29675

0.236721981836337
```

Figure 3.4.4

```
Call:
 randomForest(formula = loan_rate ~ ., data = trainset2,
              Type of random forest: classification
                    Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 24.03%
Confusion matrix:
     0      1 class.error
0 9856  35278  0.78162804
1 5157 117959  0.04188733
```

Figure 3.4.5

Figure 3.4.2 is the mean decrease in Gini coefficient is a measure of how each variable contributes to the homogeneity of the nodes and leaves in the resulting random forest, and figure 3.4.3 shows the importance of our variables. From figure 3.4.5, we see that our OOB out of bag error rate turns out to be 24.03%. Confusion matrix is what we used to calculate true positive, true negative, false positives, false negatives rate. Class Error rate is calculated as the number of all incorrect predictions divided by the total number of the dataset. The class error rate for 0 class is very high, 0.7816, but the class error rate for 1 is only 0.04, which is relatively low. Thus, our next step could be trying to minimize the 1 class error rate as much as we can while keeping the balance between data precision and accuracy.

3.5 Principal Component Regression

Principal Component Regression creates components to explain the observed variability in the predictor variables. Due to the fact that we have one response variable loan_rate, and up to 20 predictor variables, we believe that PCR would be a great instrument to reduce the dimension and complexity in predictor variables, while maximizing variance explained in response variable.

The assumptions of PCR are the same as those used in regular multiple regression: linearity, constant variance, and independence. Since PCR does not provide confidence limits, normality assumption can be eliminated here. By reasoning and observing the data distribution, we believe these assumptions are satisfied.

By setting scale = TRUE prior to generating the principle components, we make sure that the scale on which each variable is measured does not affect how PCR generates components. We then set validation method as “CV”, causing PCR to compute ten-fold cross-validation error for each possible number of principal components used. Figure 3.5.1 exhibits the resulting fit. From the figure we are able to observe the cross validation score ranging from using M=0 to

M=20, by using 20 principle components, about 11.32% of variance in predictor variables are explained.

```
TRAINING: % variance explained
           1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
X          25.6182   36.461   44.286   51.299   56.936   62.340   67.341
loan_rate   0.3573    2.352    5.592    6.909    7.011    7.047    7.047
           8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
X           72.281   77.190    81.841    86.310    90.082    93.149
loan_rate    7.049    8.521     8.808     8.966     9.561     9.577
           14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
X            95.710     97.73     99.27     99.69     100.0    100.00
loan_rate     9.619     10.87     10.93     10.94      11.3     11.31
           20 comps
X            100.00
loan_rate     11.32
```

Figure 3.5.1

We also plot the cross-validation scores using validationplot() function in R. Figure 3.5.2 shows all cross-validation MSE and number of components used. As seen from the plot, MSE reaches minimum when using 19 principle components. However, since 15 components are already able to explain a similar level of variance and result in low MSE, we fit PCR on the testing set with M=15, result in a test MSE of 25.04%.

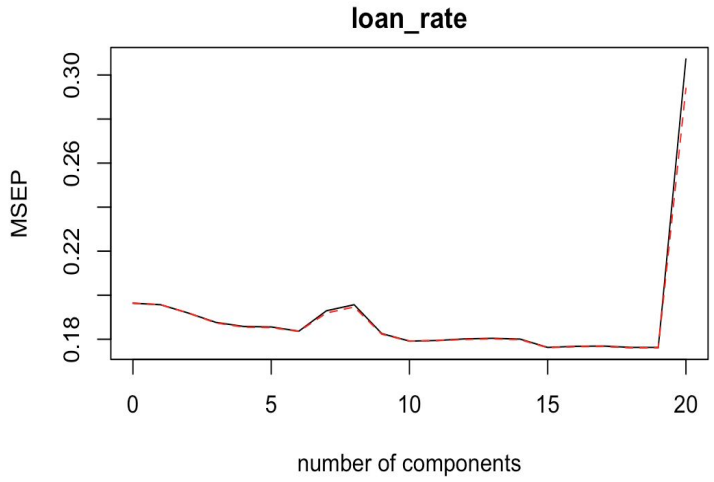


Figure 3.5.2

3.6 Partial Least Squares

While PCR creates components to explain variability in predictor variables, without considering the response variable at all, PLS does take loan_rate into account, and therefore we would expect using PLS method leading to models that are able to fit the response variable with fewer components. Similar to PCR, the assumptions of PCR also include linearity, constant variance, and independence, and these assumptions are satisfied.

We implement Partial Least Squares using plsr() function with similar syntax as pcr(). We can see from figure 3.6.1 that, PLS is also able to explain 11.32% of variance with 20 components, which is same as using PCR, yet PLS is able to reach up to 10.72% of variance with only 3 components, which is much higher than that of PCR. After using 3 components, each additional component adds little incremental impact on the percentage of variance explained.

```
TRAINING: % variance explained
           1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
X           13.628   33.051    40.97    46.81    51.03    55.74    58.83
loan_rate    8.157    9.971    10.72    11.00    11.11    11.17    11.27
           8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
X            62.37    66.09     69.48      72.9     76.17     77.71
loan_rate    11.30    11.30     11.30      11.3     11.30     11.30
           14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
X             81.97     85.52     90.06     90.22     95.04    100.00
loan_rate     11.30     11.30     11.30     11.31     11.31     11.31
           20 comps
X            100.00
loan_rate     11.32
```

Figure 3.6.1

It is also clear from cross-validation plot, figure 3.6.1, that the lowest cross-validation error occurs when only M=2 partial least squares directions are used. We evaluate the corresponding test error rate using only 2 components, and obtained 25.29%, which is slightly higher than using PCR with 15 components.

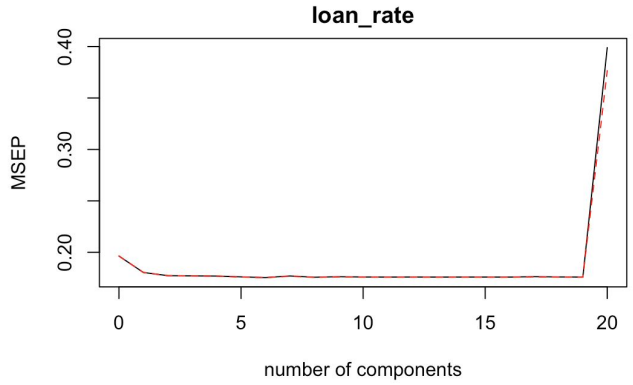


Figure 3.6.2

Comparing to PCR, PLS is able to explain a similar level of variance and reach similar test error rate, and it leads to models that are able to fit the response variable with much fewer components. Though both principal component regression and partial least regression provide a reasonable level of test error rate, the interpretability is a huge concern to us. While PLS is able to shrink predictor variables to only 2 components, we are not able to identity the most important predictors from the result.

4. Conclusion

Overall, Logistic regression, random forest, and classification tree provide the best performance in terms of test error rate. Support vector machine, principal component regression and partial least regression do not provide as satisfying results comparing to the other models in terms of error rate and interpretability. Figure 4.1.1 shows the error rates in different method. Support vector machine with sigmoid kernel, though, is able to correctly classify a relevently larger portion of bad loans, which can be valuable from a risk management perspective.

1. Logistic Regression (24.8%)
2. Support Vector Machine (25% ~ 32%)
3. Classification tree(23.7%)
4. Random Forest (23.7%)
5. PCR (25.3%)

Figure 4.1.1

That being said, logistic regression stands out to us because it is particularly easy to interpret and takes least time to train. Note that though classification tree and random forest result in relatively low MSE and error rates, we were not able to train models with the whole training set due to execution time constraint, and ended up with using randomly selected 500 observations. It is worth debating weather the extra 1% reduction in test error rate by using tree and random forest is worth this excessive execution time.

On the other hand, both the feature importance function of random forest model and the result summary of logistic regression indicate the a similar set of variables that are important in both measures. Both models illustrate that the 3 most important predictors for loan rate are: interest rate on the loan, interest received to date, and borrower's monthly debt payment divided by self-reported monthly income. In comparison, variables such as number of collections in the past 12 month does not have much effect on loan rate. It is to our surprise that the number of past-due incidents of delinquency in borrower's credit file is not one of the most important predictors, this is probably due to the fact that part of past-due incidents is already factored into credit score.

This study of Lending Club loan rate is far from complete, the conclusions presented here are limited in various aspects. For future improvement, we may consider adjusting our matrix to include more complex indicator other than test error rate. Indicator such as bad loan misclassification rate should be weighted more heavily, since for the company and lender, if a bad loan is misclassified as good, the consequence could be far worse than if a good loan is classified as bad.

5. Appendix

* * All the code and result below can also be found in our github repository:
https://github.berkeley.edu/jiahua-zou/lending-club-loan-project.git

Clean data:

```
library(dplyr)
library(tidyr)
library(ggplot2)
df <- read.csv("lending-club-loan-data/loan.csv")
head(df)
bcolnames(df)
summary(is.na(df))
too_many_na <- function(x) sum(is.na(x)) < 100000
df2 <- df %>% select_if(too_many_na)
head(df2)
summary(is.na(df2))
df2 <- df2 %>% drop_na()
summary(is.na(df2))
df2 <- df2 %>% filter(application_type == 'INDIVIDUAL')
nrow(df2)/nrow(df)

colnames(df2)
unique(df2$loan_status)
df2$loan_status <- as.character(df2$loan_status)
head(df2$loan_status, 20)
print(unique(df2$loan_status))
df2$loan_rate <- ifelse(df2$loan_status == "Current" | df2$loan_status == "Issued", 2,
                        ifelse(df2$loan_status == "Fully Paid", 1, 0))
head(df2$loan_rate, 20)
write.csv(df2, "cleaned_loan_data.csv")
sum(df2$loan_rate == 0)
```

```
sum(df2$loan_rate == 1)
sum(df2$loan_rate == 2)
nrow(df2) == sum(df2$loan_rate == 0) + sum(df2$loan_rate == 1) + sum(df2$loan_rate == 2)
```

Logistic Regression & Support Vector Machine:

```
library(dplyr)
library(tidyr)
library(ISLR)
library(MASS)
library(ggplot2)
library(e1071)
df <- read.csv('cleaned_loan_data.csv')
head(df$loan_rate, 20)
df2 <- df %>% filter(loan_rate != 2) %>% dplyr::select(-c(policy_code, id, X, member_id, term,
 out_prncp_inv)) %>% select_if(is.numeric)
df2$loan_rate <- as.factor(df2$loan_rate)
head(df2$loan_rate)
useful_var = c("loan_amnt", "funded_amnt", "funded_amnt_inv",
               "int_rate", "installment", "dti", "delinq_2yrs", "inq_last_6mths",
               "open_acc", "pub_rec", "revol_bal", "revol_util", "total_acc",
               "total_rec_int", "total_rec_late_fee", "collections_12_mths_ex_med",
               "acc_now_delinq", "tot_coll_amt", "tot_cur_bal", "total_rev_hi_lim", "loan_rate")
df3 <- df2[, useful_var]
colnames(df2)
set.seed(1)
train=sample(nrow(df3), round(0.75 * nrow(df3)))
trainset=df3[train,]
testset=df3[-train,]
glm.fit2 = glm(loan_rate~., data = trainset, family = binomial)
summary(glm.fit2)

glm.probs=predict(glm.fit2, testset, type="response")
glm.pred=rep(0, nrow(testset))
glm.pred[glm.probs>0.5]=1
table(glm.pred, testset$loan_rate)
1-mean(glm.pred==testset$loan_rate)
colnames(testset)
testset$loan_rate <- as.numeric(as.character(testset$loan_rate))
fit = glm(loan_rate ~ int_rate, data=trainset, family=binomial)
newdat <- data.frame(int_rate=seq(min(testset$int_rate), max(testset$int_rate),len=100))
newdat$loan_rate = predict(fit, newdata=newdat, type="response")
plot(loan_rate~int_rate, data=testset, col="red4")
```

```
lines(loan_rate~int_rate, newdat, col="green4", lwd=2)
ggplot(testset, aes(x=int_rate, y=loan_rate)) + geom_point() +
   geom_smooth(method = "glm", method.args = list(family = "binomial"), se = FALSE)
hist(df3$numeric_loan_rate)
temp <- sample(nrow(df3), 30000)
df4 <- df3[temp, ]
train = sample(nrow(df4), round(0.8*nrow(df4)))
length(train)
#set.seed(1)
#tune.out=tune(svm, loan_rate~., data=df4[train,], kernel = "sigmoid",
          #ranges=list(cost=c(0.1, 1, 10, 100, 1000)))
#summary(tune.out)
svmfit=svm(loan_rate~., data=df4[train,], kernel = "radial", cost = 0.1)
y_pred2 = predict(svmfit, newdata=df4[-train,])
true=df4[-train, "loan_rate"]
table(true, y_pred2)
1-(sum(true == y_pred2)/nrow(df4[-train,]))
temp <- df4[-train,]
head(temp)
plot(svmfit, temp, delinq_2yrs~loan_amnt)
true2=df2[-train, "loan_rate"]
pred2=predict(svmfit, newdata=df2[-train,])
table(true2, pred2)
1-(sum(true2 == pred2)/nrow(df2[-train,]))
svmfit3 = svm(loan_rate~., data=df4[train,], kernel = "sigmoid", cost = 0.1)
y_pred3 = predict(svmfit3, newdata=df4[-train,])
true3=df2[-train, "loan_rate"]
pred3=predict(svmfit3, newdata=df2[-train,])
table(true3, pred3)
1-(sum(true3 == pred3)/nrow(df2[-train,]))
1-((15881 + 110444)/(15881+34603+25384+110444))

plot(svmfit3, temp, delinq_2yrs~loan_amnt)
colnames(df4)
train2 = sample(nrow(df3), round(0.8 * nrow(df3)))
length(train2)
trainset2 <- df3[train2,]
dim(trainset2)
testset2 <- df3[-train2, ]
dim(testset2)
```

Random Forest:

```
library(randomForest)
```

```
rf_model <- randomForest(loan_rate~.,data = trainset2, keep.forest=TRUE)
rf_model
varImpPlot(rf_model)
importance(rf_model)
yhat_rf <- predict(rf_model, newdata = testset2)
plot(yhat_rf, testset2$loan_rate)
abline(0, 1)
table(testset2$loan_rate, yhat_rf)
1-(sum(testset2$loan_rate == yhat_rf)/nrow(testset2))
```

Classification Tree :

```
Data Implement
df2 <read.csv("C:/Users/ariel/Dropbox/Stat154/Project/data/newdata21.csv")
#0 is bad loan, 1 is good loan, 2 is current loan status
Perform tree on the data set:
library("dplyr")
random <sample(80000,500)
new.df <df2[random,]
Find the non current loan
non_current_df <new.df[new.df$loan_rate<2,]
library("tree")
set.seed(1)
bad <ifelse(non_current_df$loan_rate <= 0,"Bad","Good")
non_current_df$term <NULL
non_current_df$grade <NULL
non_current_df$sub_grade <NULL
non_current_df$addr_state <NULL
non_current_df$initial_list_status <NULL
library("rpart")
loan.tree <rpart(loan_rate~.,data = non_current_df,method = "class")
loan.tree
## n= 249
##
## node), split, n, loss, yval, (yprob)
## * denotes terminal node
##
## 1) root 249 59 1 (0.23694779 0.76305221)
## 2) recoveries>=117.685 28 0 0 (1.00000000 0.00000000) *
## 3) recoveries< 117.685 221 31 1 (0.14027149 0.85972851)
## 6) out_prncp>=830.105 12 0 0 (1.00000000 0.00000000) *
## 7) out_prncp< 830.105 209 19 1 (0.09090909 0.90909091)
## 14) total_rec_prncp< 2901.725 15 5 0 (0.66666667 0.33333333) *
## 15) total_rec_prncp>=2901.725 194 9 1 (0.04639175 0.95360825)
```

```
## 30) total_rec_prncp< 6248.635 33 7 1 (0.21212121 0.78787879)
## 60) loan_amnt>=7250 7 0 0 (1.00000000 0.00000000) *
## 61) loan_amnt< 7250 26 0 1 (0.00000000 1.00000000) *
## 31) total_rec_prncp>=6248.635 161 2 1 (0.01242236 0.98757764) *

     summary(loan.tree)
## Call:
## rpart(formula = loan_rate ~ ., data = non_current_df, method = "class")
## n= 249
##
## CP nsplit rel error xerror xstd
## 1 0.47457627 0 1.0000000 1.0000000 0.11372374
## 2 0.20338983 1 0.5254237 0.5254237 0.08829933
## 3 0.08474576 2 0.3220339 0.4067797 0.07893056
## 4 0.05932203 3 0.2372881 0.3389831 0.07269107
## 5 0.01000000 5 0.1186441 0.2542373 0.06363585
##
## Variable importance
## collection_recovery_fee recoveries total_rec_prncp
## 16 16 8
## out_prncp out_prncp_inv funded_amnt
## 8 8 7
## funded_amnt_inv loan_amnt total_pymnt
## 7 7 7
## installment total_pymnt_inv total_rec_int
## 5 5 4
##
## Node number 1: 249 observations, complexity param=0.4745763
## predicted class=1 expected loss=0.2369478 P(node) =1
## class counts: 59 190
## probabilities: 0.237 0.763
## left son=2 (28 obs) right son=3 (221 obs)
## Primary splits:
## recoveries < 117.685 to the right, improve=36.73699, (0 missin
g)
## collection_recovery_fee < 2.7377 to the right, improve=36.73699, (0 missin
g)
## total_rec_prncp < 6394.785 to the left, improve=36.24565, (0 missin
g)
## total_pymnt < 8698.835 to the left, improve=18.02811, (0 missin
g)
## total_pymnt_inv < 8698.835 to the left, improve=18.02811, (0 missin
g)
```

```
## Surrogate splits:
## collection_recovery_fee < 2.7377 to the right, agree=1.000, adj=1.000, (0 s
plit)
## total_rec_prncp < 737.08 to the left, agree=0.908, adj=0.179, (0 s
plit)

    plot(loan.tree,uniform = FALSE)
    text(loan.tree,all = TRUE,cex = 0.75, splits = TRUE, use.n = TRUE,xpd = TRUE)
    set.seed(1)
    library("rpart")
    library("rpart.plot")
    loan.tree <rpart(loan_rate~.,data = non_current_df,cp = 0.2)
    rpart.plot(loan.tree)
    library("rattle")
    fancyRpartPlot(loan.tree,main = "Decision Tree Graph")
    prp(loan.tree,extra = 100,under = T,yesno = F)
    library("randomForest")
    plot(loan.rf)
```

Principal Component Regression:

```
    loan= read.csv("/Users/yijunxu/Downloads/cleaned_loan_data.csv")
    ls(loan)
    head(loan)
    library(dplyr)
    useful_var = c("loan_amnt", "funded_amnt", "funded_amnt_inv",
          "int_rate", "installment", "dti", "delinq_2yrs", "inq_last_6mths",
          "open_acc", "pub_rec", "revol_bal", "revol_util", "total_acc",
          "total_rec_int", "total_rec_late_fee", "collections_12_mths_ex_med",
          "acc_now_delinq", "tot_coll_amt", "tot_cur_bal", "total_rev_hi_lim",
          "loan_rate")
    loan2 <- loan[, useful_var]
    loan3 <-  loan2 %>% filter(loan_rate != 2)
    #split into training and testing set
    train = sample (210312, 147218)
    trainset = loan3[train,]
    testset = loan3[-train,]
    library(pls)
    pcr.fit = pcr(loan_rate ~., data = trainset, scale = TRUE, validation = "CV")
    summary(pcr.fit)
    validationplot(pcr.fit, val.type = "MSEP")
    pcr.pred = predict(pcr.fit, testset, ncomp=3)
    mean((pcr.pred-testset$loan_rate)^2)
```

```
pcr.pred = predict(pcr.fit, testset, ncomp=15)
pcr.select = rep("0", 63094)
pcr.select[pcr.pred>0.5] = "1"
table(pcr.select, testset$loan_rate)
1-mean(pcr.select == testset$loan_rate)
#PCR excluding some variables randomly
pcr.fit2 = pcr(loan_rate ~ loan_amnt,
      data = trainset, scale = TRUE, validation = "CV")
summary(pcr.fit2)
#plot validation score
validationplot(pcr.fit2, val.type = "MSEP")
pcr.pred2 = predict(pcr.fit, testset, ncomp=1)
pcr.select2 = rep("0", 63094)
pcr.select2[pcr.pred2>0.5] = "1"
table(pcr.select2, testset$loan_rate)
1-mean(pcr.select2 == testset$loan_rate)
sum(pcr.pred2 < 0.5)
```

Partial Least Square Regression:

```
set.seed(1)
pls.fit = plsr(loan_rate ~., data = trainset, scale = TRUE, validation = "CV")
summary(pls.fit)
validationplot(pls.fit, val.type = "MSEP")
pls.pred=predict(pls.fit,testset,ncomp=2)
mean((pls.pred-testset$loan_rate)^2)
pls.select = rep("0", 63094)
pls.select[pls.pred>0.5] = "1"
table(pls.select, testset$loan_rate)
1-mean(pls.select == testset$loan_rate)
```