What factors are significant for predicting whether credit card payments will default?

Zhuo Feng Lei (zlei5)

2/10/2020

Abstract

In order to avoid significant financial losses, banks must be able to recognize whether consumers will default on their credit card payments. Statistical machine learning methods and techniques are utilized to find an accurate model that best classify default payments.

Introduction

Credit cards are flexible tools issued by banks to consumers. It enables the cardholder to pay a merchant for goods and services based on the cardholder's promise to the card issuer to pay them for the amounts plus other agreed charges by the due date listed on the credit card statement. When a customer fails to pay back the loan by the due date, the credit card will be defaulted. If the bank is certain that they will not get their money back, the bank will try to sell the loan. If the bank are unable to sell off the loan to debt collectors, the bank will write off the loan (charge-off). As a result, the customer's credit ratings tanks while the bank suffers significant financial losses.

Risk prediction is a powerful tool that can be utilized by banks and other financial service providers. Risk predictions is essential for predicting customer's credit risks or business performances. The ability to predict which customers are most probable to default on payments will allow banks to reduce damages and uncertainty when issueing credit cards to customers. Analyzing millions of transactions and client information to make a prediction is both resource and time consuming and prone to errors. Therefore, it is more efficient to use machines to compute and analyze the data. Currently, a variety of machine learning approaches are used to predict default payments. [1] Through machine learning, the goal of this project is to implement a model that can accurately classify whether loans will default or not based on certain features.

Data

The data was accessed via Kaggle. [2] The dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005. The dataset have 30,000 observations and 25 variables. The response variable is *default.payment.next.month*, whether the payment will default next month.

The input variables are:

- LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit

Higher limit balance indicates a higher credit score. Clients with higher credit scores are deemed more reliable and trustworthy

- SEX: Gender (1=male, 2=female)

Gender wage gaps. Men may earn more money than women in some workplaces. Having high wages usually indicates that clients are able to pay back the business

- EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)

Higher education means higher income.

- MARRIAGE: Marital status (1=married, 2=single, 3=others)

Having kids cost money. Being married and having kids may affect defaulting. On the other hand, married couples may have dual income which may help pay credit card payments

- AGE: Age in years

Younger adults may still be in school or have entry level jobs that doesn't pay as much as senior positions. This may have an effect on whether payments will default or not

- PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, … 8=payment delay for eight months, 9=payment delay for nine months and above)
- PAY_2: Repayment status in August, 2005 (scale same as above)
- PAY_3: Repayment status in July, 2005 (scale same as above)
- PAY_4: Repayment status in June, 2005 (scale same as above)
- PAY_5: Repayment status in May, 2005 (scale same as above)
- PAY_6: Repayment status in April, 2005 (scale same as above)

Repayment status helps determine how clients are doing financially. Delayed payments indicate that clients are having difficulties paying back the money they owe

- BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
- BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
- BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
- BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
- BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)

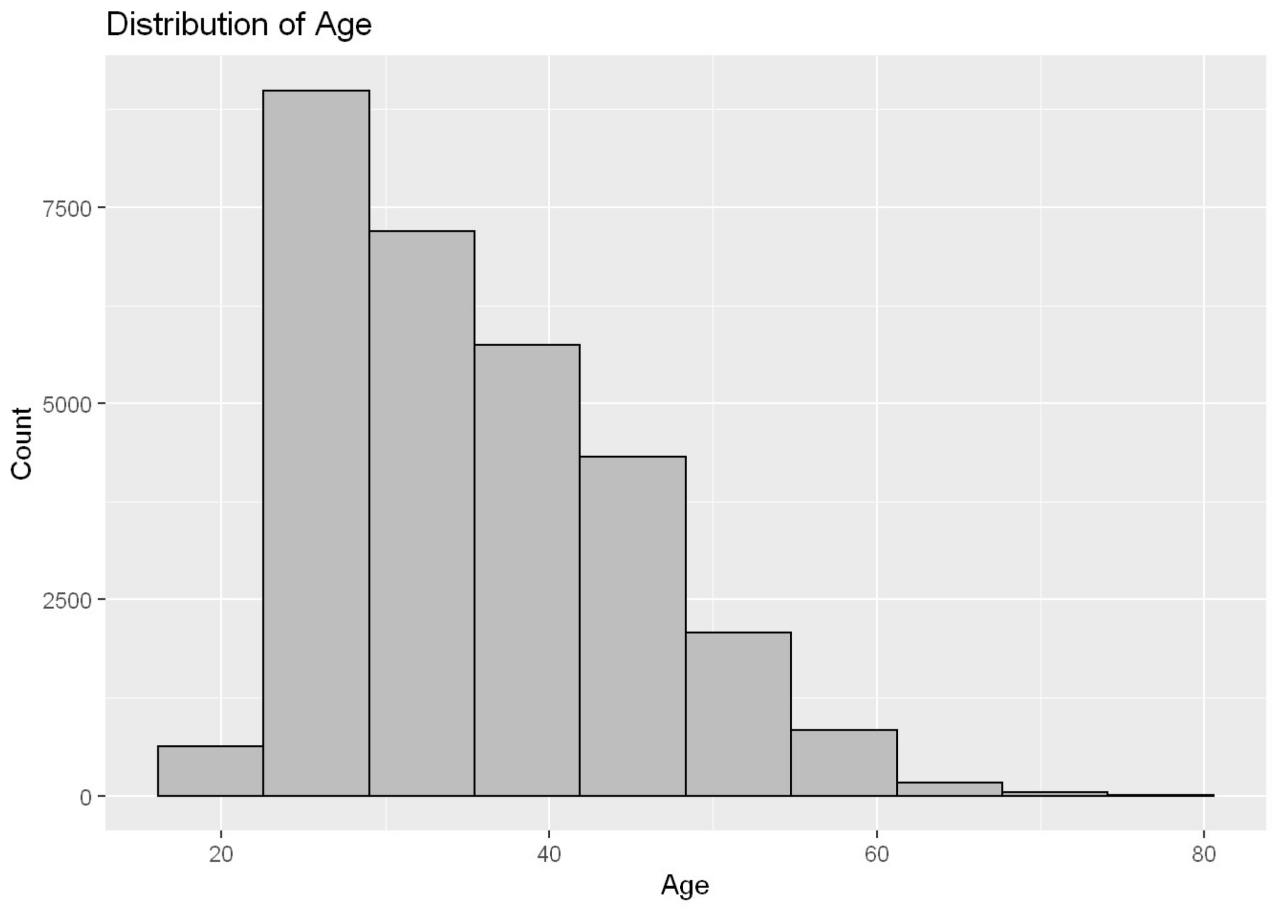
- BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)

A high bill amount may indicate bad spending habits which may have an effect on defaulting payments

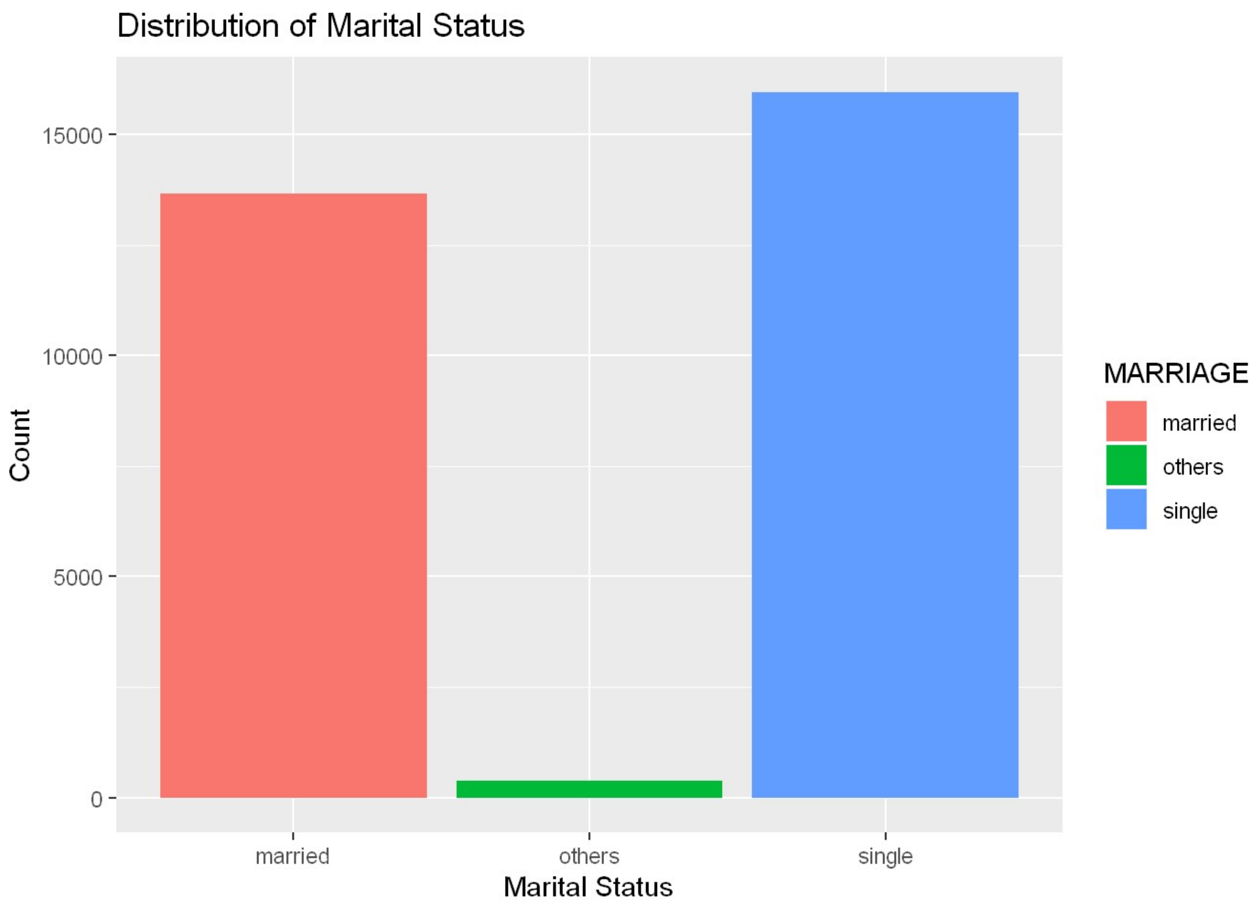
- PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
- PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
- PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
- PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
- PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
- PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)

Summary Statistics

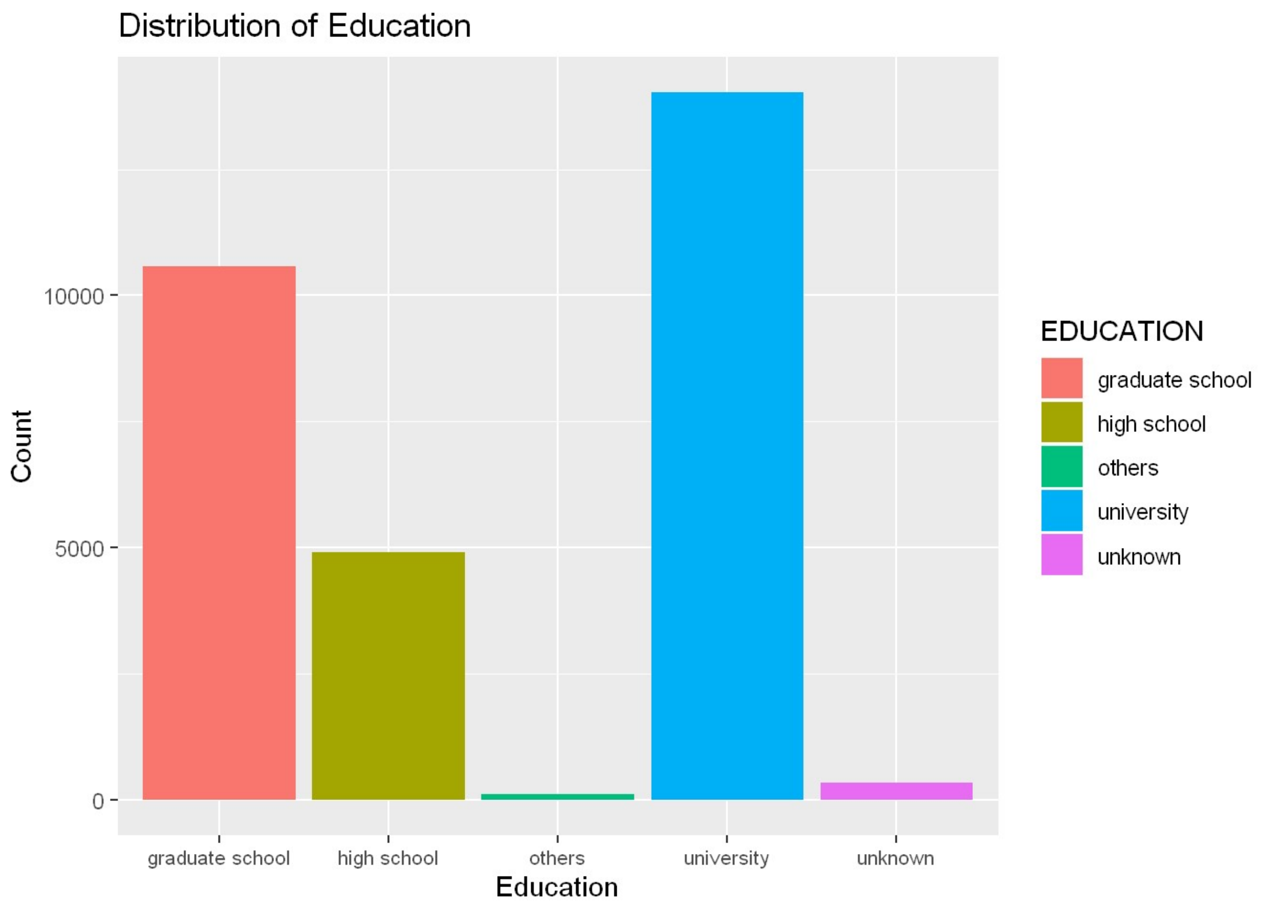
Univariate Analysis



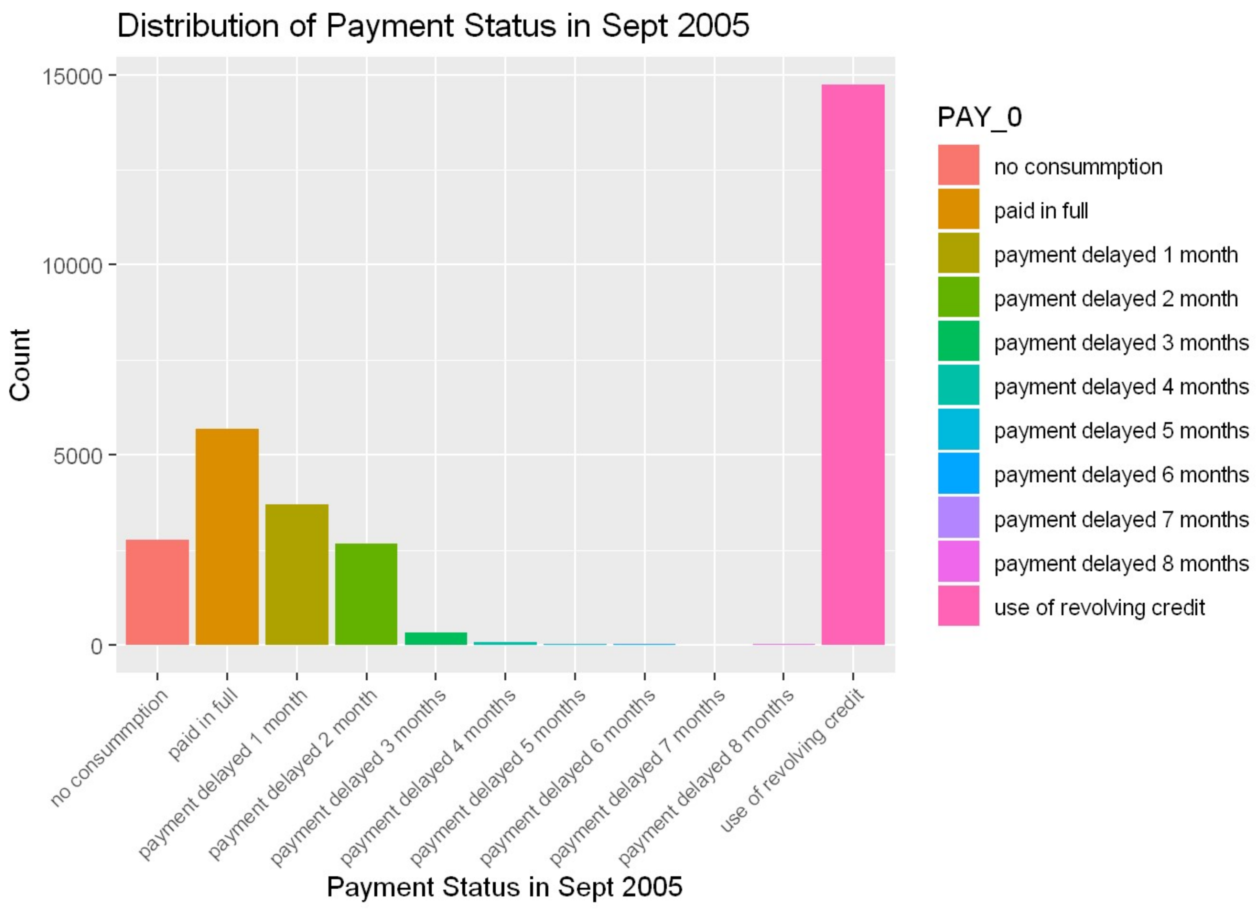
Most of the clients are in 20s and 30s. There are very little clients in the 20s. This is probably due to low credit scores in young adults.



The sample distribution of marital status are roughly even.

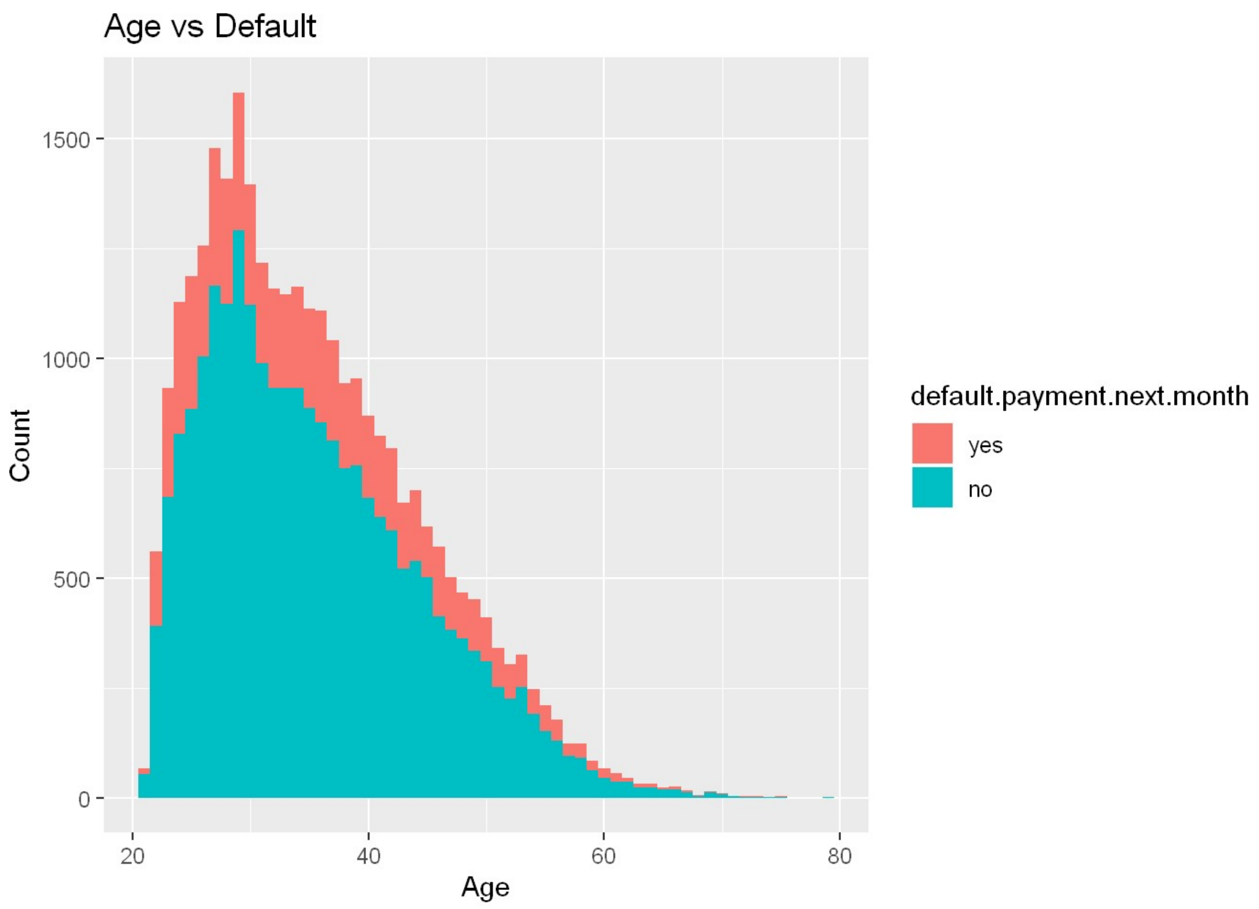


Most of the client’s either have graduate school or university as their highest form of education. The third most group is high school.

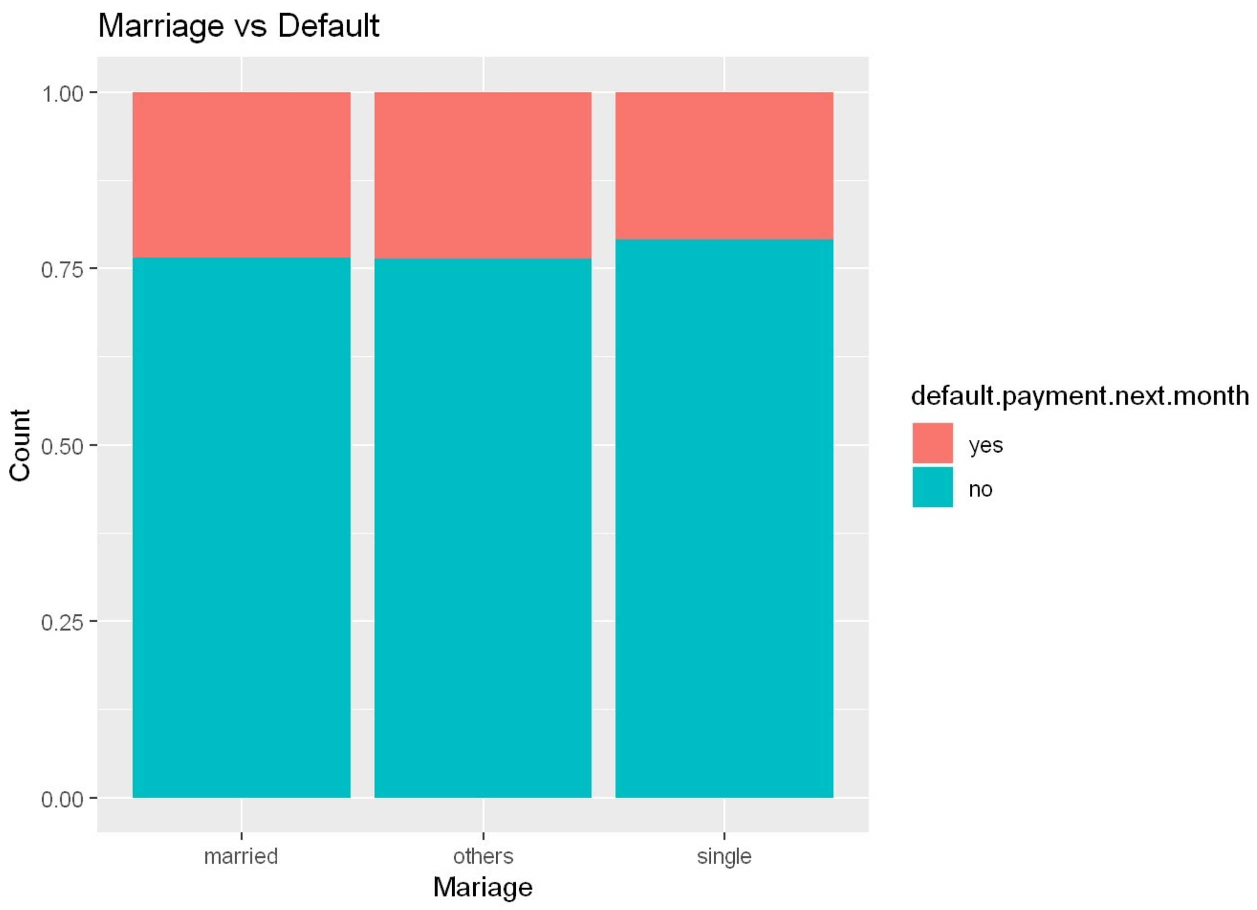


It appears that clients mostly use revolving credit for their payments in Sept 2005. Stagnant wages coupled with the rising cost of living may cause people to live paycheck to paycheck. As a result, many people may not be able to pay off all their credit card bill and use revolving credit. The second most frequent status is clients paying off all their balance.

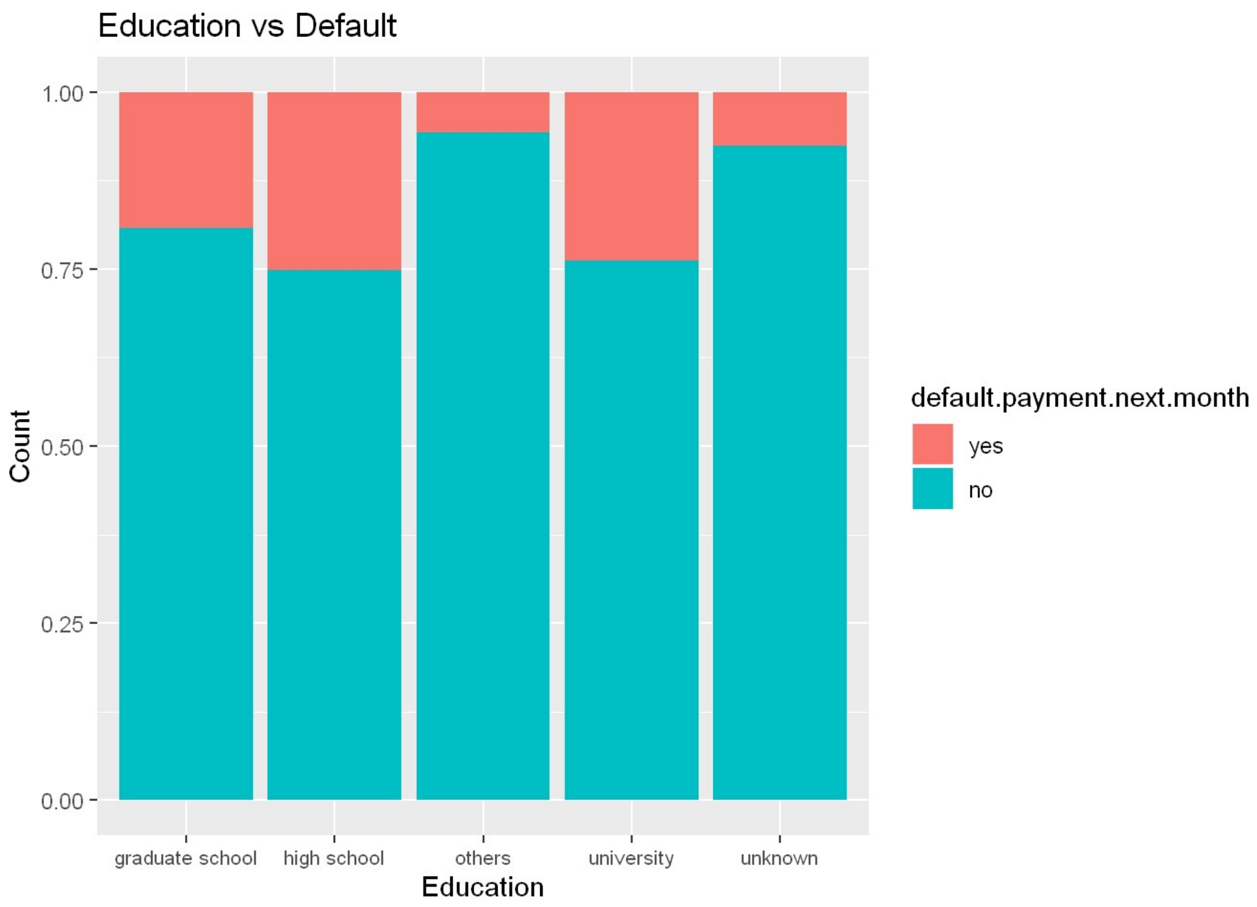
Bi-variate Analysis



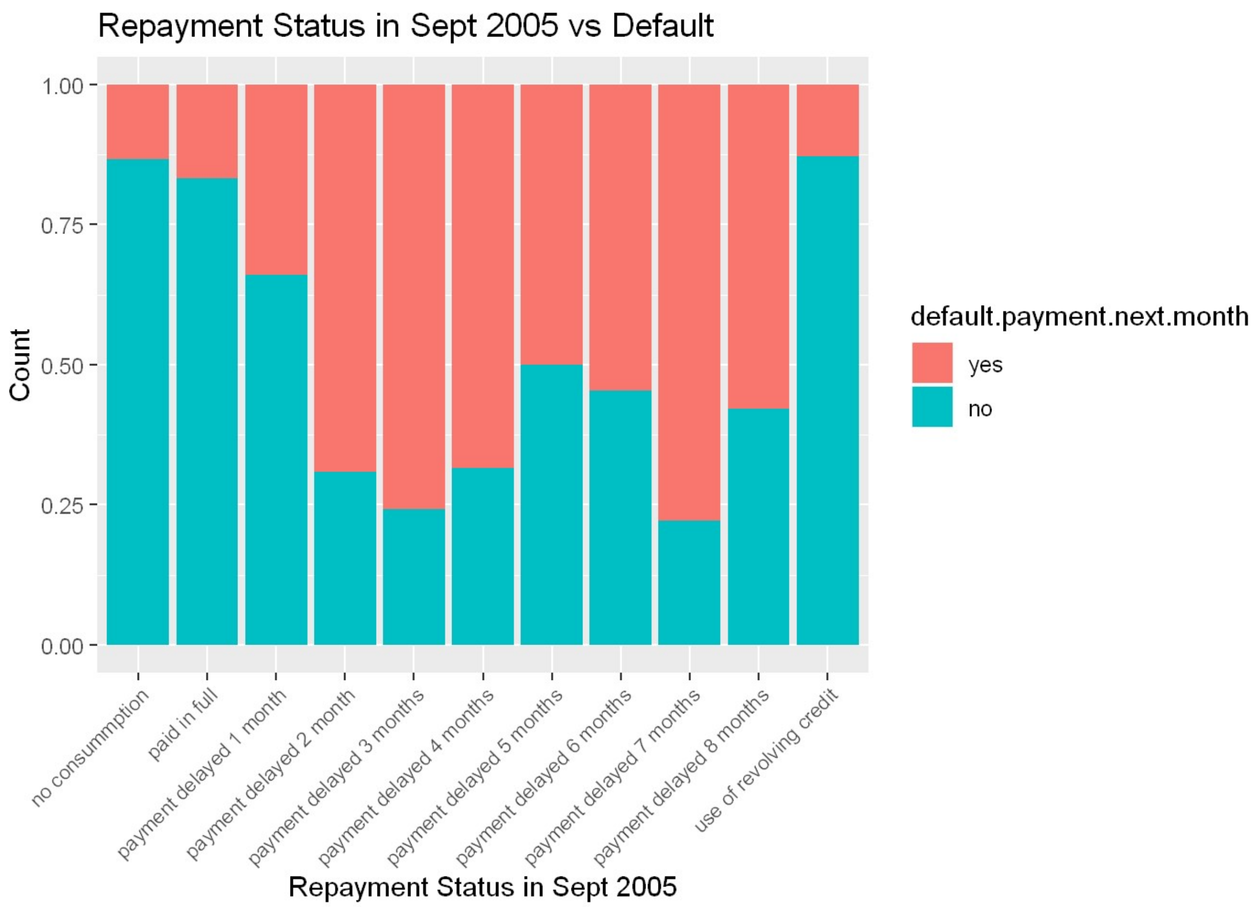
The age distribution for default payments are more skewed than the age for non-default payments. Many people that default in their next month payments are in their 20s to 40s. As age increases, the number of default payments seem to decrease. It appears that age hae a significant effect on whether payments will default, but additional analysis is needed to determine if this is truly the case.



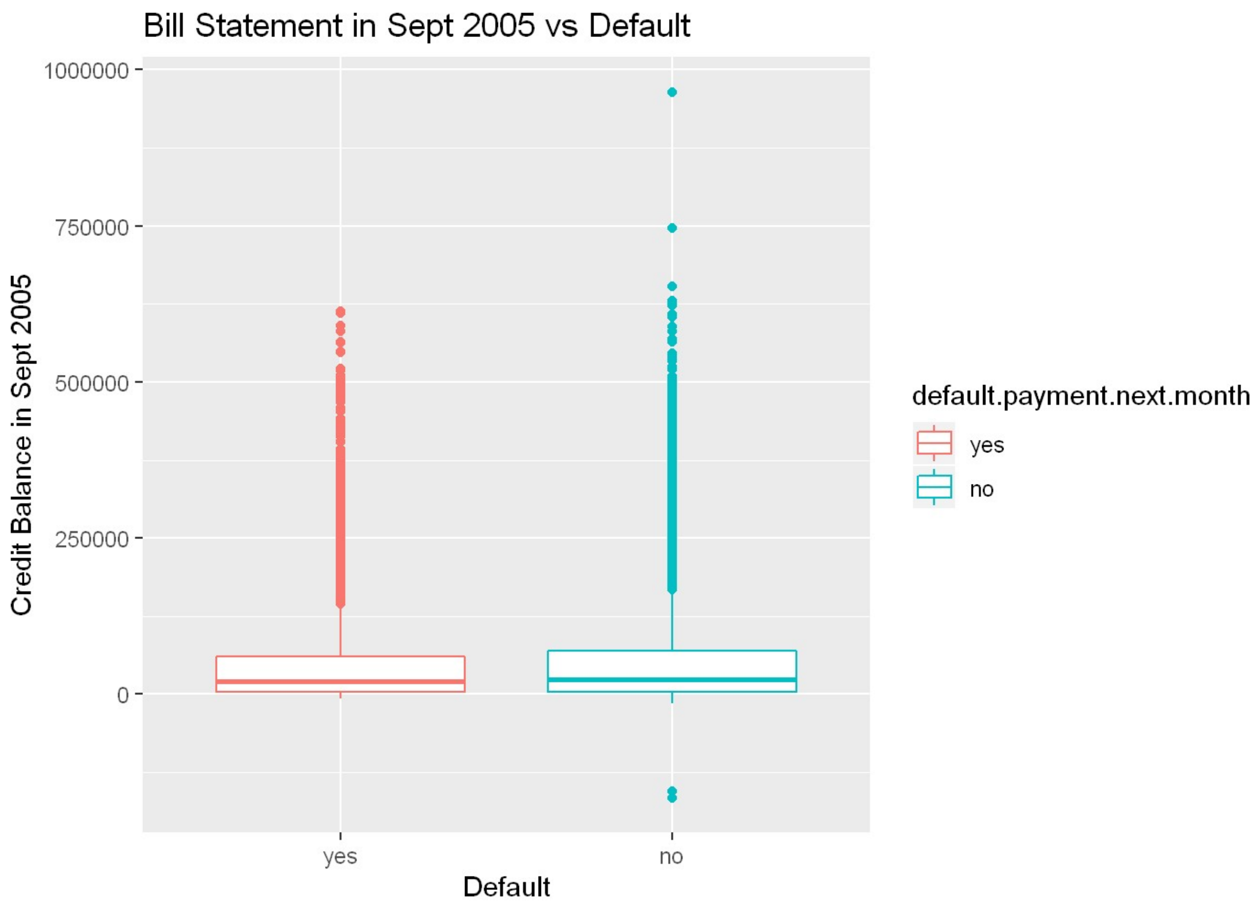
It appears that marital status does not have significantly affect whether payments will default since all three groups have roughly the same ratio of default payments in their respective groups. Additional analysis is needed to to determine whether marital status have any significant effects on default payments.



Clients whose highest education was university or highschool have the largest ratio of default payments. In the case of university, this may be due to student loans or field of studies. In the case of highschool, clients may not have high income due to the lack of higher education. Additional analysis is needed to determine if education have any significant effect on whether payments will default.



It appears that the payments that are delayed for 2,3,4 or 7 months have the highest ratio of default payments in their respective populations. Additional analysis is needed to determine if repayment status have any significant effects on the outcome.



Since the confidence intervals for both boxplot overlap, the difference between the two groups seems insignificant. It does not seem that credit balance in Sept 2005 have a significant effect on whether the payment next month will default or not. Additional analysis is required.

Modeling

In order to build a classifier to predict whether payments will default, different modeling techniques were considered: K's Nearest Neighbor, Logistic Regression, and Probit Regression.

Logistic Regression

Model	Accuracy
Logistic Regression Model 1	0.8214776
Logistic Regression Model 2	0.7971429
Logistic Regression Model 3	0.8200486
Logistic Regression Model 4	0.8210481

Model	Accuracy
Logistic Regression Model 5	0.8210489

The best logistic model is the one with all the predictors except ID. Independent features with p-value less than .05 have a significant effect on the dependent feature (whether payments will default or not in the next month) and their coefficients are significantly different from zero. Features with lower p-values have a larger effect on the outcome. According to the summary of the logistic regression mod with all the features, some of the significant features are the balance limit on the account, the male gender, unknown education status, how much the client paid in the first and second month, and repayment status in the first month.

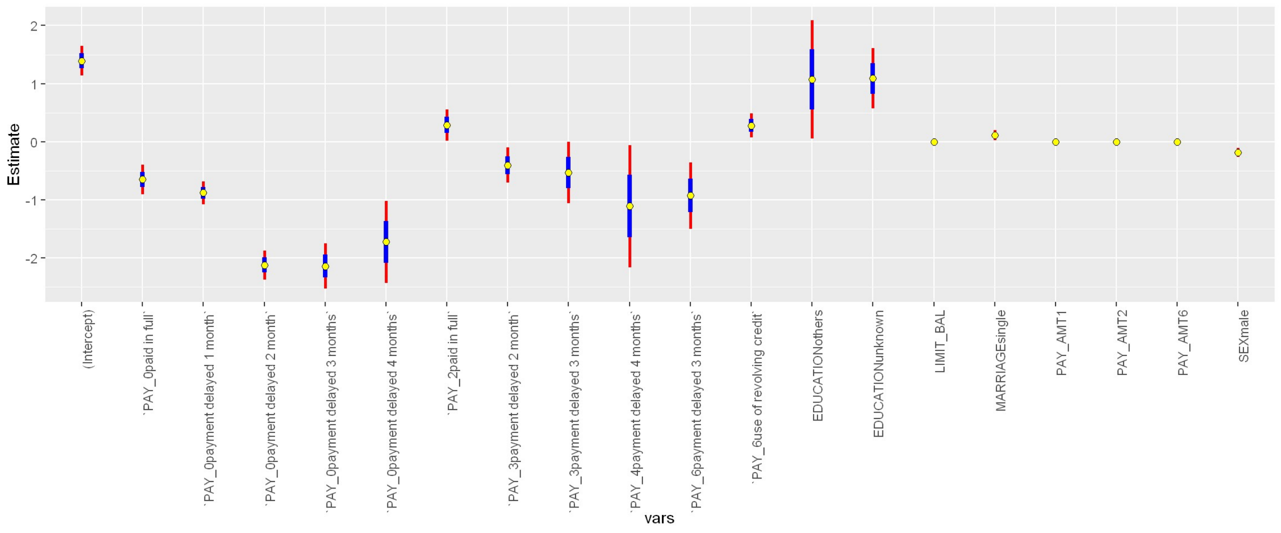
```
##                        LIMIT_BAL                         SEXmale 
##                     1.852798e-06                   -1.883685e-01 
##          `EDUCATIONhigh school`             EDUCATIONuniversity 
##                     6.561628e-02                   -1.418319e-02 
##                   MARRIAGEothers                             AGE 
##                     8.667289e-02                   -3.416999e-03 
##  `PAY_0payment delayed 1 month`  `PAY_0payment delayed 2 month` 
##                    -8.810109e-01                   -2.123550e+00 
## `PAY_0payment delayed 3 months` `PAY_0payment delayed 4 months` 
##                    -2.143407e+00                   -1.726956e+00 
## `PAY_0payment delayed 5 months`  `PAY_2payment delayed 1 month` 
##                    -6.706212e-01                    3.719730e-01 
## `PAY_3payment delayed 3 months` `PAY_3payment delayed 4 months` 
##                    -5.319631e-01                    8.325471e-02 
## `PAY_4payment delayed 5 months` `PAY_6payment delayed 4 months` 
##                     4.563092e-01                    3.679199e-02 
##                        BILL_AMT1                        PAY_AMT2 
##                     1.258920e-06                    9.862795e-06 
##                         PAY_AMT3 
##                     4.270652e-07
```

The table above displays a list of all significant variables and their respective coefficients.

- For every unit increase in LIMIT_BAL (balance limit on the account), the log odds of defaulting increases by 1.852798e-06.
- If the client is male, the log odds of defaulting decreases by 1.883685e-01.
- If education is in the others category, then the log odds of defaulting increases by 1.070740e+00.
- If education is unknown, the log odds of defaulting increases by 1.091142e+00.
- If the client is single, the log odds of defaulting increase by 1.093061e-01.
- If client paid in full during their first month, the log odds of defaulting decreases by 6.517546e-01.
- If the payment is delayed for 1 month during the first payment, then the log odds of defaulting decreases by 8.810109e-01.
- If first payment is delayed for 2 months, the log odds of defaulting decreases by 2.123550e+00.

- If first payment is delayed for 3 months, then the log odds of defaulting decreases by 2.143407e+00.
- If first payment is delayed for 4 months, then the log odds of defaulting decreases by 1.726956e+00.
- If second payment is paid in full, the log odds of defaulting increases by 2.882975e-01.
- If third payment is delayed for 2 months, then the log odds of defaulting decreases by 4.027158e-01.
- If fourth payment is delayed for 4 months, then log odds of defaulting decreases by 1.112698e+00.
- If sixth payment is delayed for 3 months, then the log odds of defaulting decreases by 9.270053e-01.
- If sixth payment is paid using revolving credit, then the log odds of defaulting increases by 2.789789e-01.
- For every unit increase in the first payment amount, the log odds of defaulting increases by 1.260881e-05.
- For every unit increase in the second payment amount, the log odds of defaulting increases by 9.862795e-06.
- For every unit increase in the sixth payment amount, the log odds of defaulting increases by 5.201935e-06.

Some of the coefficients from the logistic regression model seems illogical in a real life scenario. I agree that if client's owe more to the card companies, the odds of defaulting increases. People with high payment amounts are usually bad with money management and live a lifestyle they cannot sustain. However, the model also says that delayed payments reduce the chance of defaulting. It would make more sense if delayed payments increase the chance of defaulting since it indicates that the client cannot financiially pay back the amount owed. It is also interesting to note that being single increases the chances of defaulting. This may be due to the fact that people usually wait until they are financially stable before starting a family (raising kids cost a lot). Being male also lowers the chances of defaulting. This may be caused by the gender wage gaps. Additional modeling and research is needed to verify whether these relationships are true or not.



If range of the standard error includes 0, then the independent variable may not be reliable in predicting whether credit card payments will default or not. The range of the standard errors significant variables list previously do not include zeros which means the variables have a significant effect on the dependent variable.

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction  yes   no
##        yes  8.0  3.7
##        no  14.2 74.1
##
##  Accuracy (average) : 0.8215
```

The logistic regression model accurately classified ~82.15% of the data. The true positive rate is 8% while the false positive rate is 3..7%. Out of 22.2% clients that defaulted, the model only accurately classified 8% of them. The logistic regression model should not be used for classifying whether payments will default because it missed 14.2% of the clients that defaulted. In a business context, this would mean significant financial losses as the model failed to catch a lot of the clients that will default on their card payments.

Probit Regression

```
## Generalized Linear Model
##
## 21000 samples
##    24 predictor
##     2 classes: 'yes', 'no'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 18900, 18901, 18900, 18899, 18901, 18900, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8217621  0.3794524
```

The probit model correctly classified ~82.18% of the testing data. This is slightly higher than the accuracy of the logistic regression model.

```
##                        LIMIT_BAL                          SEXmale 
##                     1.048554e-06                    -1.054614e-01 
##          `EDUCATIONhigh school`             EDUCATIONuniversity 
##                     2.953039e-02                    -1.394896e-02 
##                  MARRIAGEothers                              AGE 
##                     4.691508e-02                    -1.975981e-03 
##  `PAYAY_0payment delayed 1 month`  `PAY_0payment delayed 2 month` 
##                    -5.044629e-01                    -1.264947e+00 
## `PAY_0payment delayed 3 months` `PAY_0payment delayed 4 months` 
##                    -1.288808e+00                    -1.020902e+00 
## `PAY_0payment delayed 5 months`  `PAY_2payment delayed 1 month` 
##                    -3.505612e-01                     2.616484e-01 
## `PAY_3payment delayed 3 months` `PAY_3payment delayed 4 months` 
##                    -3.158608e-01                     5.221333e-02 
## `PAY_6payment delayed 4 months`                        BILL_AMT1 
##                     1.703023e-02                     2.184918e-07 
##                         PAY_AMT2                         PAY_AMT3 
##                     4.032940e-06                     4.830022e-07
```

According to the probit regression, the features above have a significant effect on whether crediti card payments will default. Similar to the logistic regression model, some of the probit regression results also does not make sense in a business context. It is illogical that delayed payments decrease the chances of defaulting. The probit regression model returned similar results to the logistic regression model.

K’s Nearest Neighbors

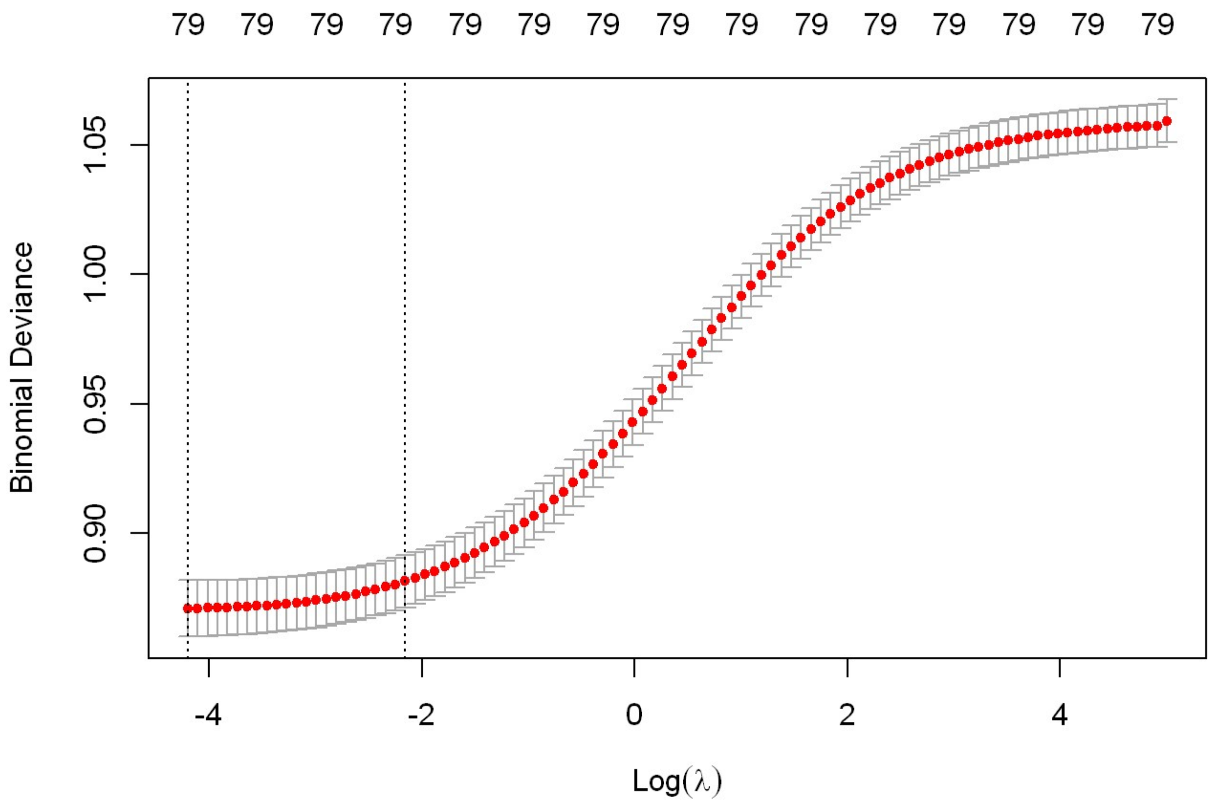
```
## k-Nearest Neighbors 
## 
## 21000 samples
##    24 predictor
##     2 classes: 'yes', 'no' 
## 
## No pre-processing
## Resampling: Cross-Validated (10 fold) 
## Summary of sample sizes: 18899, 18900, 18900, 18899, 18901, 18899, ... 
## Resampling results across tuning parameters:
## 
##   k  Accuracy   Kappa    
##   5  0.7540948  0.1237021
##   7  0.7593810  0.1082840
##   9  0.7649999  0.1099473
## 
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

The KNN model with a k = 9 accurately predicted ~76.5% of the data while the logistic regression managed to accuratel predict 82.15% of the data. This may be due to the fact that KNN tends to perform badly with high dimensional data. The model's performance indicats that the KNN model should not be used to classify whether credit card payments will default or not.

Ridge Regression

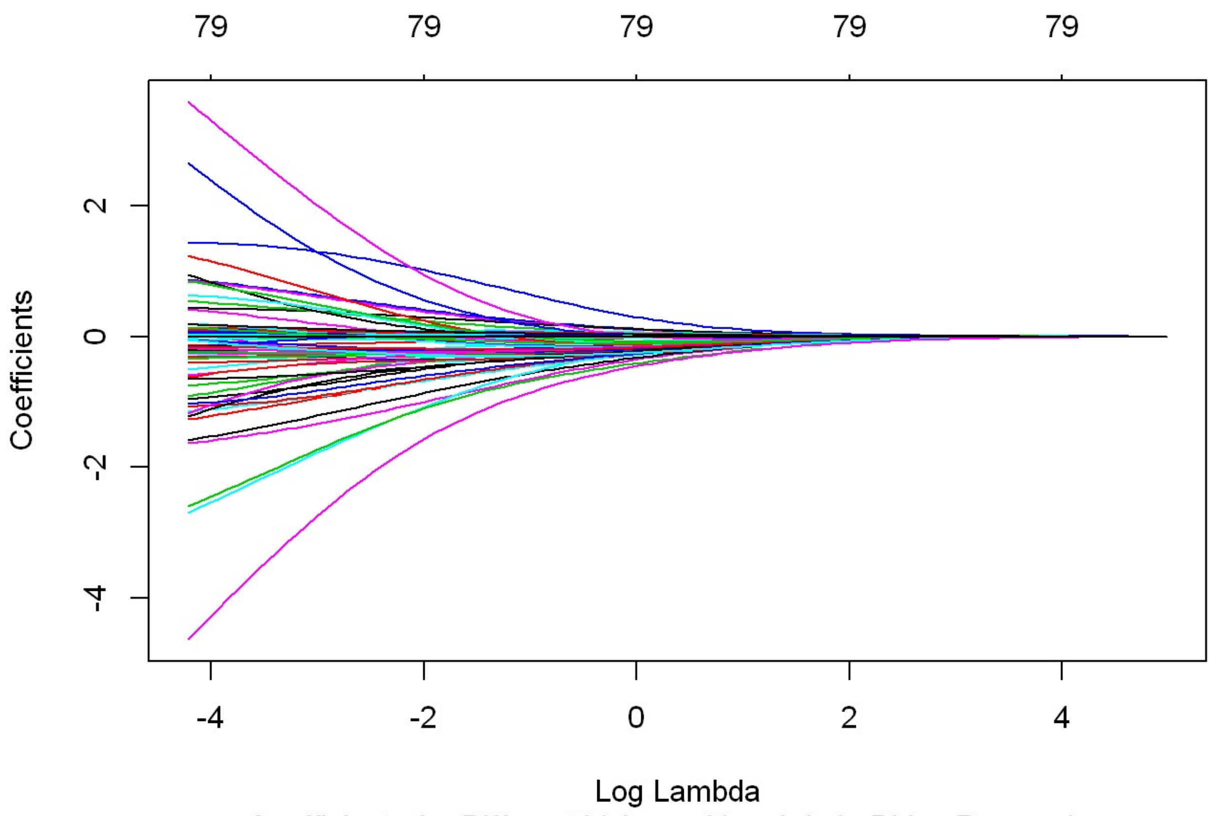
```
##
## Call:  cv.glmnet(x = cc_trn_x, y = cc_trn$default.payment.next.month,      alpha =
0, family = "binomial")
##
## Measure: Binomial Deviance
##
##      Lambda Measure      SE Nonzero
## min 0.01495  0.8709 0.01085      79
## 1se 0.11576  0.8814 0.01001      79
```

```
## [1] "The lambda within one standard error of the minimum lambda is 0.01495143015730
17"
```



Training MSE as a Function of Lambda in Ridge Regression

As lambda increases, deviance increases. This is related to the tradeoff between bias and deviance. As deviance increases, we have less bias.



Coefficients for Different Values of Lambda in Ridge Regression

The ridge model does not have feature selection. Therefore, variables slowly converges to zero whereas the variables in lasso may sharlpy decrease to 0 as lambda increases. As penalty increases, more variables converge to zero.

```
##                        LIMIT_BAL                          SEXmale
##                     1.661062e-06                    -1.361582e-01
##              EDUCATIONhigh school                  EDUCATIONothers
##                     1.756536e-03                     8.742573e-01
##              EDUCATIONuniversity                 EDUCATIONunknown
##                    -4.037247e-02                     8.606922e-01
##                   MARRIAGEothers                   MARRIAGEsingle
##                     6.456639e-02                     1.286576e-01
##                              AGE                PAY_0paid in full
##                    -3.507986e-03                    -1.630770e-01
##   PAY_0payment delayed 1 month   PAY_0payment delayed 2 month
##                    -4.969004e-01                    -1.628793e+00
## PAY_0payment delayed 3 months PAY_0payment delayed 4 months
##                    -1.577554e+00                    -1.261570e+00
## PAY_0payment delayed 5 months PAY_0payment delayed 6 months
##                    -9.069509e-01                    -2.232731e-01
## PAY_0payment delayed 7 months PAY_0payment delayed 8 months
##                    -1.147306e+00                     1.055774e-01
##   PAY_0use of revolving credit                PAY_2paid in full
##                     4.274054e-01                     6.560073e-02
##   PAY_2payment delayed 1 month   PAY_2payment delayed 2 month
##                     5.365009e-01                    -2.116594e-01
## PAY_2payment delayed 3 months PAY_2payment delayed 4 months
##                    -2.624525e-01                     4.183582e-01
## PAY_2payment delayed 5 months PAY_2payment delayed 6 months
##                    -9.474963e-01                    -1.066636e+00
## PAY_2payment delayed 7 months PAY_2payment delayed 8 months
##                    -7.485074e-01                     1.431973e+00
##   PAY_2use of revolving credit                PAY_3paid in full
##                    -5.506756e-02                     6.232664e-02
##   PAY_3payment delayed 1 month   PAY_3payment delayed 2 month
##                     9.411059e-01                    -3.189696e-01
## PAY_3payment delayed 3 months PAY_3payment delayed 4 months
##                    -3.378665e-01                     1.185642e-01
## PAY_3payment delayed 5 months PAY_3payment delayed 6 months
##                     6.307373e-01                    -1.164618e+00
## PAY_3payment delayed 7 months PAY_3payment delayed 8 months
##                    -2.122372e-01                     1.226202e+00
##   PAY_3use of revolving credit                PAY_4paid in full
##                    -9.955529e-03                     7.372505e-02
##   PAY_4payment delayed 1 month   PAY_4payment delayed 2 month
##                    -2.685573e+00                    -2.598629e-01
## PAY_4payment delayed 3 months PAY_4payment delayed 4 months
##                    -1.439501e-01                    -4.033678e-01
## PAY_4payment delayed 5 months PAY_4payment delayed 6 months
##                     8.453430e-01                     2.641790e+00
## PAY_4payment delayed 7 months PAY_4payment delayed 8 months
##                    -3.274304e-02                     3.564582e+00
##  PAY_4use of revolving credit             PAY_5paid in full
##                 2.586693e-02                  9.995637e-02
##  PAY_5payment delayed 2 month PAY_5payment delayed 3 months
##                -2.389425e-01                 -6.122996e-02
## PAY_5payment delayed 4 months PAY_5payment delayed 5 months
##                 1.919235e-01                 -5.895302e-01
## PAY_5payment delayed 6 months PAY_5payment delayed 7 months
##                -1.206964e+00                 -6.188929e-01
## PAY_5payment delayed 8 months  PAY_5use of revolving credit
##                -2.585703e+00                  3.933022e-02
##             PAY_6paid in full  PAY_6payment delayed 2 month
##                 3.059653e-02                 -1.813728e-01
## PAY_6payment delayed 3 months PAY_6payment delayed 4 months
##                -6.517594e-01                 -1.591981e-01
## PAY_6payment delayed 5 months PAY_6payment delayed 6 months
##                 1.383168e-01                 -1.028214e+00
## PAY_6payment delayed 7 months PAY_6payment delayed 8 months
##                -5.470686e-02                 -4.602558e+00
##  PAY_6use of revolving credit                     BILL_AMT1
##                 1.848355e-01                 -3.037793e-07
##                     BILL_AMT2                     BILL_AMT3
##                -8.418378e-07                 -6.464863e-07
##                     BILL_AMT4                     BILL_AMT5
##                -2.187557e-07                  1.389749e-08
##                     BILL_AMT6                      PAY_AMT1
##                 1.533201e-07                  8.129814e-06
##                      PAY_AMT2                      PAY_AMT3
##                 5.500552e-06                  2.300467e-06
##                      PAY_AMT4                      PAY_AMT5
##                 2.823587e-06                  3.132620e-06
##                      PAY_AMT6
##                 2.750934e-06
```

Ridge regression does not perform variable selection, therefore none of the coefficients are exactly 0. However, independent variables with coefficients close to zero have less importance or effect on the dependent variable. According to the ridge model, features like repayment status, education, and marital status affect whether clients will default and have higher coefficients compared to other variables like payment amount and billing amount. The coefficient estimates seem to make sense in real life scenarios. The ridge models indicates that single clients are less likely to default. People with families might find it harder to pay off their card balances since they also have a family to support. Repayment status helps determine the financial risks. Clients with delayed payments are often deemed as less trustworthy and involves more risk. Education is correlated with income. People with higher education are able to find better jobs and have more job opportunities. The estimates of the ridge model makes sense from an economic perspective.

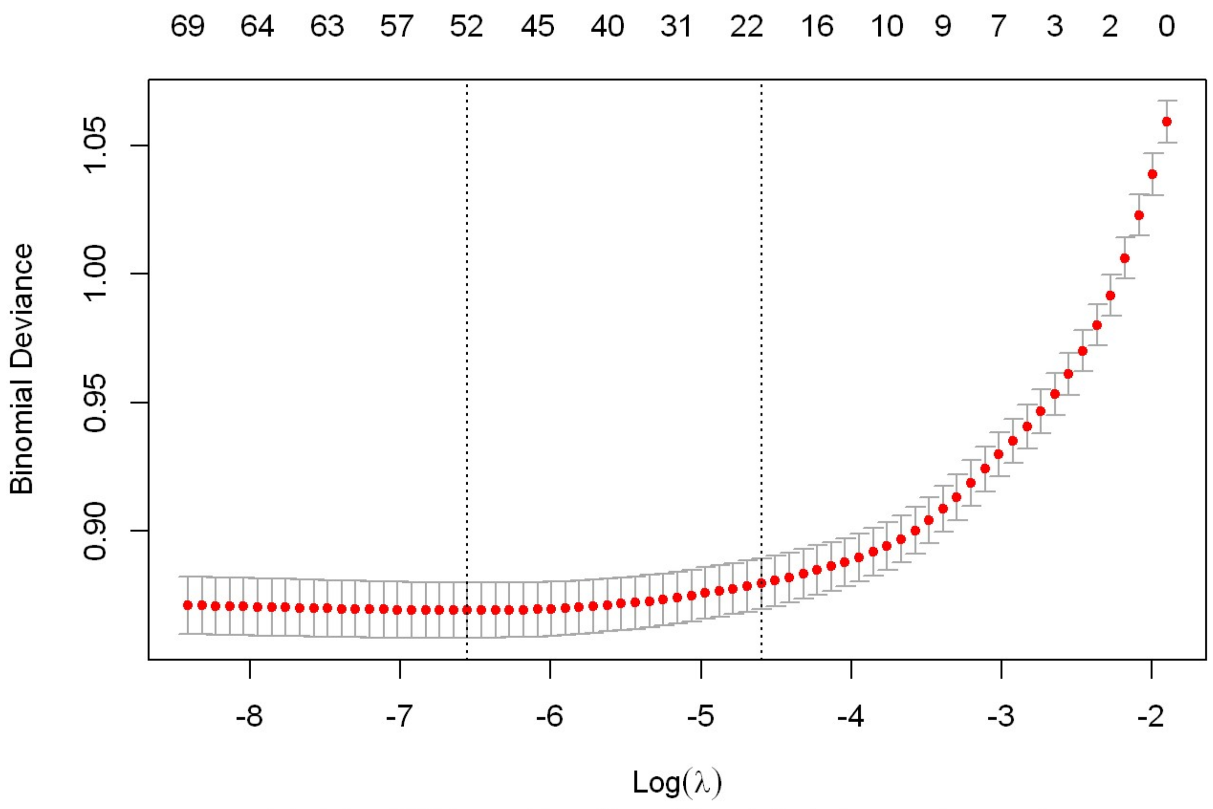
```
## [1] "Ridge regression accurately classified 81.6555555555556% of the data."
```

Although the ridge model coefficient estimates make sense, it still undeperformed when compared to the logit and probit models. The logistic and probit models both correctly classified ~82% of the data while the ridge model only correctly classified ~81.66%. However, the ridge model slightly outperforms the lasso model which correctly classified ~81.63% of the data. On the other hand, the KNN model still remains to have the highest error rate, only managing to correctly classify ~76.5% of the data.

Lasso Regression

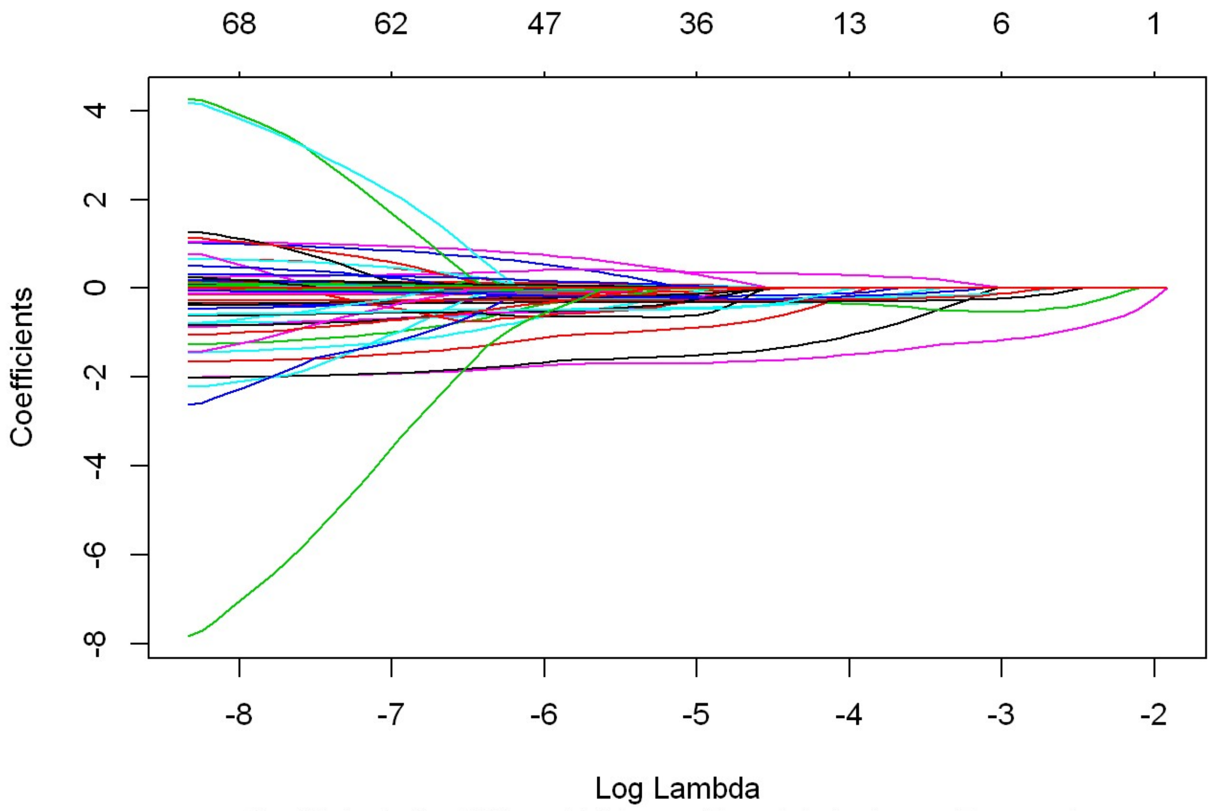
```
##
## Call:  cv.glmnet(x = cc_trn_x, y = cc_trn$default.payment.next.month,      alpha =
1, family = "binomial")
##
## Measure: Binomial Deviance
##
##       Lambda Measure       SE Nonzero
## min 0.001427  0.8693 0.010760      52
## 1se 0.010069  0.8796 0.009889      19
```

```
## [1] "The lambda within one standard error of the minimum lambda is 0.00142718645209
44"
```



Training MSE as a Function of Lambda in Lasso Regression

As the penalty increases, the deviance also increases. This is related to the tradeoff between bias and deviance. As deviance increases, we have less bias.



Coefficients for Different Values of Lambda in Lasso Regression

As stated previously, the lasso model has built in feature selecion.The lasso picks out non-important variables and set their coefficients to zero. This may cause sharp decreases to zero where as the variables in the ridge model slowly converges to zero. As the penalty increases, more non-important variables are selected.

```
## 80 x 1 sparse Matrix of class "dgCMatrix"
##                                          1
## (Intercept)                   1.282934e+00
## LIMIT_BAL                     1.833262e-06
## SEXmale                      -1.317952e-01
## EDUCATIONhigh school           .
## EDUCATIONothers               7.413715e-01
## EDUCATIONuniversity          -2.144408e-02
## EDUCATIONunknown              8.786389e-01
## MARRIAGEothers                 .
## MARRIAGEsingle                1.287915e-01
## AGE                          -2.965077e-03
## PAY_0paid in full            -2.430045e-01
## PAY_0payment delayed 1 month  -6.328563e-01
## PAY_0payment delayed 2 month  -1.860616e+00
## PAY_0payment delayed 3 months -1.828286e+00
## PAY_0payment delayed 4 months -1.346042e+00
## PAY_0payment delayed 5 months -7.638013e-01
## PAY_0payment delayed 6 months -2.897274e-01
## PAY_0payment delayed 7 months -1.003132e+00
## PAY_0payment delayed 8 months  .
## PAY_0use of revolving credit   3.398330e-01
## PAY_2paid in full              9.599316e-02
## PAY_2payment delayed 1 month   2.962033e-01
## PAY_2payment delayed 2 month  -8.232628e-02
## PAY_2payment delayed 3 months -1.134548e-01
## PAY_2payment delayed 4 months  3.009446e-01
## PAY_2payment delayed 5 months -5.942216e-01
## PAY_2payment delayed 6 months -5.451947e-01
## PAY_2payment delayed 7 months -7.221375e-01
## PAY_2payment delayed 8 months  1.413121e-01
## PAY_2use of revolving credit   .
## PAY_3paid in full              5.198394e-02
## PAY_3payment delayed 1 month   .
## PAY_3payment delayed 2 month  -3.307121e-01
## PAY_3payment delayed 3 months -2.605054e-01
## PAY_3payment delayed 4 months  .
## PAY_3payment delayed 5 months  1.286879e-02
## PAY_3payment delayed 6 months -3.155071e-01
## PAY_3payment delayed 7 months -5.862294e-02
## PAY_3payment delayed 8 months  .
## PAY_3use of revolving credit   .
## PAY_4paid in full              6.409409e-02
## PAY_4payment delayed 1 month  -7.185503e-01
## PAY_4payment delayed 2 month  -2.700061e-01
## PAY_4payment delayed 3 months -1.067376e-01
## PAY_4payment delayed 4 months -2.494520e-01
## PAY_4payment delayed 5 months  2.305335e-01
## PAY_4payment delayed 6 months  4.396583e-01
## PAY_4payment delayed 7 months -2.762189e-02
## PAY_4payment delayed 8 months  1.054769e+00
## PAY_4use of revolving credit   .
## PAY_5paid in full              7.997094e-02
## PAY_5payment delayed 2 month  -2.749492e-01
## PAY_5payment delayed 3 months -5.816983e-02
## PAY_5payment delayed 4 months  .
## PAY_5payment delayed 5 months  .
## PAY_5payment delayed 6 months -7.204482e-02
## PAY_5payment delayed 7 months -5.126269e-01
## PAY_5payment delayed 8 months  .
## PAY_5use of revolving credit   .
## PAY_6paid in full              4.931109e-02
## PAY_6payment delayed 2 month  -1.164090e-01
## PAY_6payment delayed 3 months -4.914677e-01
## PAY_6payment delayed 4 months  .
## PAY_6payment delayed 5 months  .
## PAY_6payment delayed 6 months -5.608047e-01
## PAY_6payment delayed 7 months  .
## PAY_6payment delayed 8 months -1.914082e+00
## PAY_6use of revolving credit   2.357446e-01
## BILL_AMT1                      .
## BILL_AMT2                     -1.373813e-06
## BILL_AMT3                     -4.907284e-07
## BILL_AMT4                      .
## BILL_AMT5                      .
## BILL_AMT6                      .
## PAY_AMT1                       9.530053e-06
## PAY_AMT2                       5.986068e-06
## PAY_AMT3                       1.241629e-06
## PAY_AMT4                       1.772342e-06
## PAY_AMT5                       2.451019e-06
## PAY_AMT6                       1.890882e-06
```

Unlike the ridge model, the lasso model has built-in feature selection. The coefficient of non-important variables are set to zero. Based on the estimates, 20 of the 80 coefficient estimates are exactly zero. This means that those variables have no effect on whether credit card payments will default. According to the lasso model, variables like payment amount, repayment status, education, and age all affect whether clients default on their credit card payments. This makes sense in a business context. Younger adults tend to have entry level jobs so they do not make as much money as their seniors. Having a higher education allows clients to have more job opportunities. Repayment status and payment amount helps determine the credit of the client. Paying very little and delaying payments are usually indicators of financial trouble. However, the lasso model is also a bit illogical. The lasso model only found bill amount in certain months to be useful in predicting whether client’s default or not instead of finding all the months important. This issue also applies to repayment status in certain months as well.

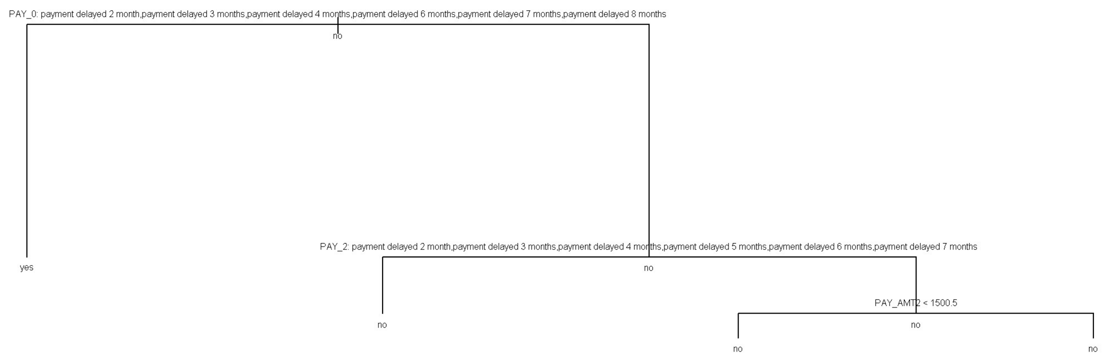
```
## [1] "Lasso regression accurately classified 81.6333333333333% of the data."
```

The lasso model is slightly outperformed by the ridge model. The ridge model managed to correctly classified .03% more data than the lasso model. The lasso model also have a higher error rate than the logistic and probit models. On the other hand, the lasso model outperforms the KNN model due to the curse of high dimensionality.

Classification Tree

```
##
## Classification tree:
## tree(formula = default.payment.next.month ~ ., data = cc_trn)
## Variables actually used in tree construction:
## [1] "PAY_0"    "PAY_2"    "PAY_AMT2"
## Number of terminal nodes:  4
## Residual mean deviance:  0.8862 = 18610 / 21000
## Misclassification error rate: 0.1792 = 3764 / 21000
```

The classification tree have 4 terminal nodes and 3 important variables. It considers the variables *PAY_0*, *PAY_2*, and *PAY_AMT2* important at classifying whether credit card payments will default or not. The classification tree managed to correctly classified 82.08% of the in the training dataset.

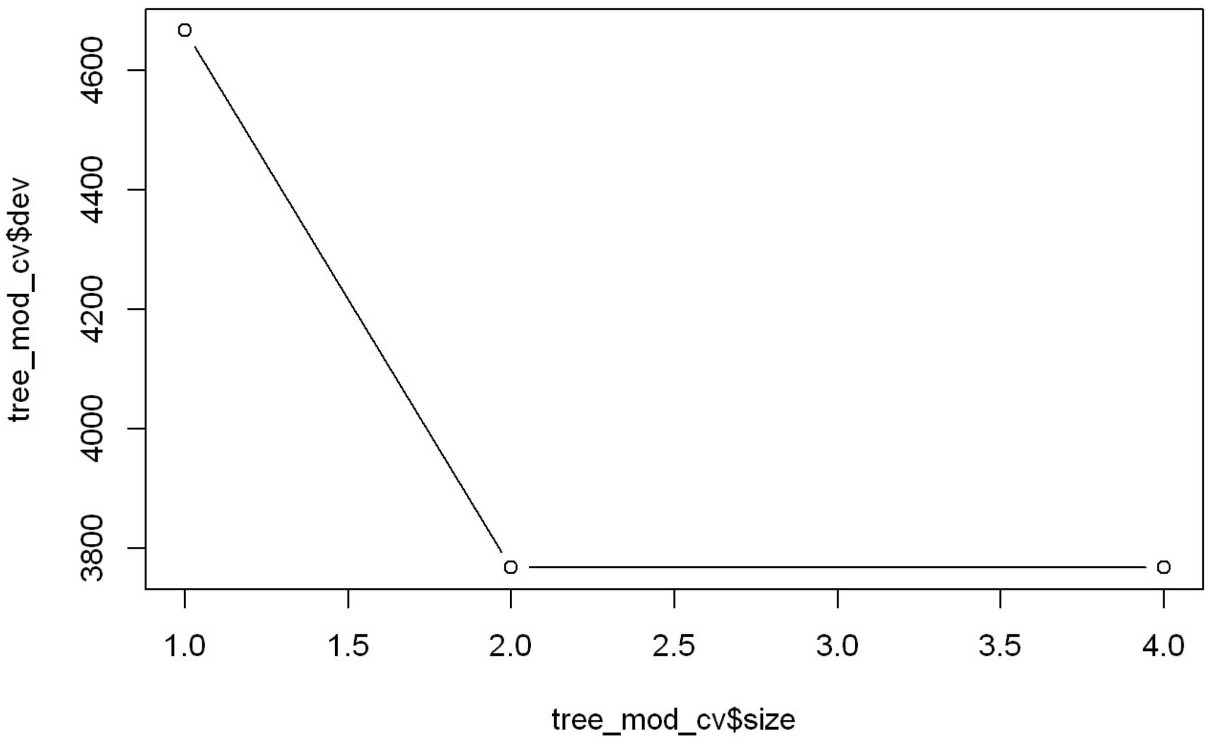

PAY_0: payment delayed 2 month,payment delayed 3 months,payment delayed 4 months,payment delayed 6 months,payment delayed 7 months,payment delayed 8 months
no
yes
PAY_2: payment delayed 2 month,payment delayed 3 months,payment delayed 4 months,payment delayed 5 months,payment delayed 6 months,payment delayed 7 months
no
no
PAY_AMT2 < 1500.5
no
no
no

The most important indicator of whether credit card payments will default appears to be the repayment status in September, 2005. The first branch of the classification tree differentiates whether clients had no consumption, paid in full, used revolving credit, or delayed the payment for 1-8 months. The next most important indicator is the repayment status in August 2005. It also differentiates whether clients had no consumption, paid all their balance, used revolving credit, or delayed payment for 1-8 months. The third most important indicator is how much the client paid in the second month. It differentiates whether clients paid more than $1500.5. However, it appears that whether if the payment are above or below $1500.5, both groups are classified to not default on their credit card payments.

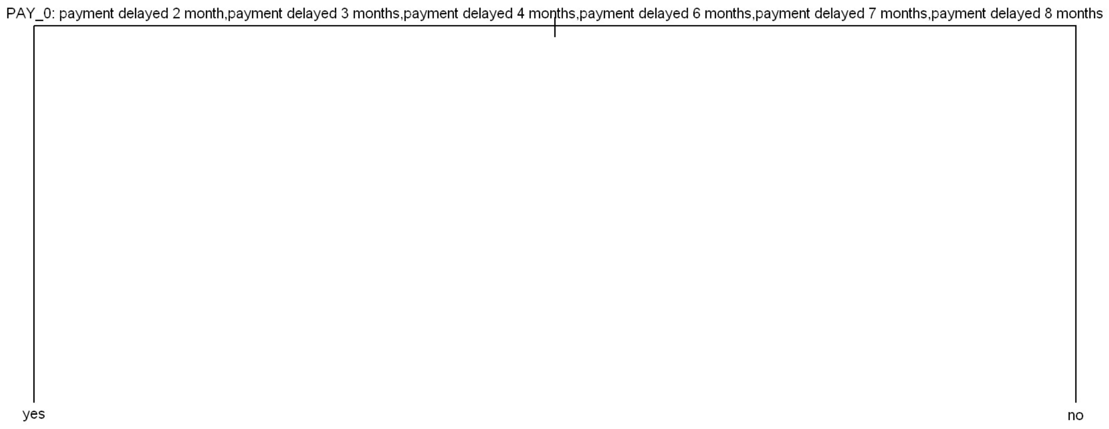
```
## [1] "Classification tree accurately classified 81.6888888888889% of the data."
```

The classification tree managed to accurately classified ~81.69% of the data. This is still not ideal in a business scenario. A model having an error rate of ~19% will cost significant losses for the credit card companies. Pruning the classification tree may improve the performance of the model.

```
## $size
## [1] 4 2 1
##
## $dev
## [1] 3767 3767 4666
##
## $k
## [1] -Inf    0  902
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```



We see from this plot that the tree with 2 terminal nodes results in the lowest cross-validation error rate, with 3769 cross-validation errors. The classification tree may perform better if we only use 2 terminal nodes.


PAY_0: payment delayed 2 month,payment delayed 3 months,payment delayed 4 months,payment delayed 6 months,payment delayed 7 months,payment delayed 8 months
yes
no

After pruning the classification tree, only one variable is left. The pruned classification tree model considers *PAY_0* to be the only feature needed to predict whether credit card payments will default. It differentiates whether clients had no consumption, paid in full, used revolving credits, or delayed the payment for 1-8 months.

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
## 1) root 21000 22250 no ( 0.2222 0.7778 )
##   2) PAY_0: payment delayed 2 month,payment delayed 3 months,payment delayed 4 mont
hs,payment delayed 6 months,payment delayed 7 months,payment delayed 8 months 2206  26
78 yes ( 0.7044 0.2956 ) *
##   3) PAY_0: no consummption,paid in full,payment delayed 1 month,payment delayed 5
months,use of revolving credit 18794 16870 no ( 0.1656 0.8344 ) *
```

The pruned classification tree only have one branch. If the clients had no consumption, paid all their balances, delayed their payments for 1 or 5 months, or used revolving credit in September 2005, then they are classified to not default on their credit card payments. On the other hand, if the clients had their payments delayed for 2,3,4,6,7, or 8 months, then the clients are classified to default on their card payments. This result does not really make sense in a business scenario. Clients that delayed their payments for 5 months are classified to not default on their payments while clients that delayed their payments for 2 months are. It would make more sense if clients that delayed their payments for 5 months are classified to default on their payments.

The pruned classification tree seems illogical in a business context. Based on prior and background knowledge, payment amount cannot be the sole indicator of whether credit card payments will default or not. There should be more factors that affect whether the dependent variable but the classification tree aren’t considering those.

```
## [1] "Classification tree accurately classified 81.6888888888889% of the data."
```

It appears that pruning the model did not improve the performance of the classification tree. The classification tree before pruning and after pruning both have identical error rates. The classification tree performed better than the lasso and ridge models. The lasso model accurately predicted ~81.63% of the data while the ridge model accurately classified ~81.66% of the data. However, the classification tree performed worse than the logitistic model and the probit model. The logitistic model with all covariates correctly classified ~82.15% of the data and the probit model correctly classified ~81.18% of the data. The classification tree outperforms the knn model, but that is to be expected since knn performs badly with high dimensionality data. Out of all the models I ran so far, probit regression have the lowest error rate. The logitistic regression model has the second lowest error rate..

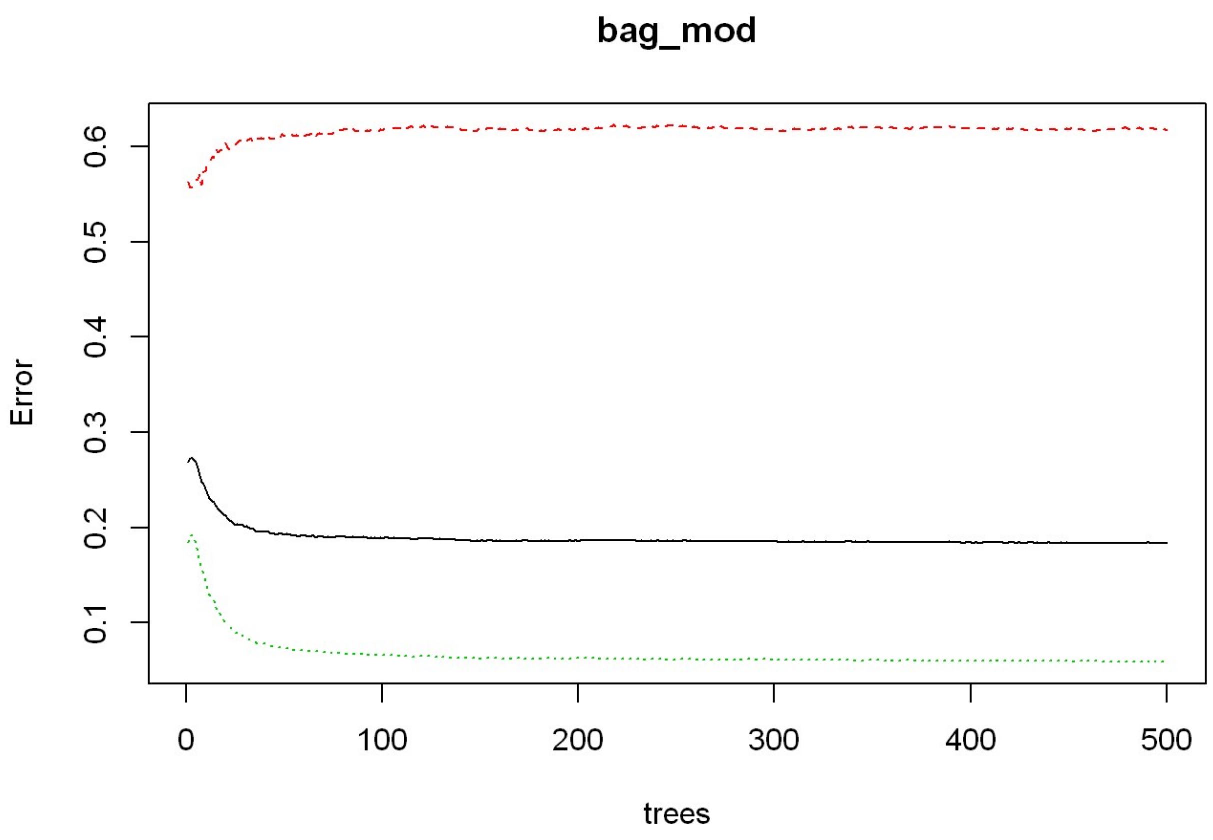
Bootstraping

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = cc_trn, statistic = boot_fn, R = 100, formula = default.payment.next.mo
nth ~
##     .)
##
##
## Bootstrap Statistics :
##      original        bias     std. error
## t1* 0.8168889 -7.777778e-06 0.0003810759
```

Decision trees are not very robust and suffer from having high variance. The prediction can change drastically with a small change in the sample. Therefore, bootstrapping should be used to to reduce the variance by averaging over multiple trees. The bootstrapped model accurately classified ~81.69% of the data with a std error of .00038 and a bias of -7.78 e^-06. The bootstrapped model have a less bias and variance. The bootstrapped model has an identical performance to the classification tree.

Bagging

```
##
## Call:
##  randomForest(formula = default.payment.next.month ~ . - ID, data = cc_trn,      mt
ry = ncol(cc_trn) - 1, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 23
##
##         OOB estimate of  error rate: 18.33%
## Confusion matrix:
##      yes    no class.error
## yes 1784  2882  0.61765967
## no   968 15366  0.05926289
```



As the number of trees increase, the error rate starts to decrease and levels off. Variance decreases as we increase the number of trees.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  yes   no
##        yes  723  428
##        no  1247 6602
##
##                Accuracy : 0.8139
##                  95% CI : (0.8057, 0.8219)
##     No Information Rate : 0.7811
##     P-Value [Acc > NIR] : 1.001e-14
##
##                   Kappa : 0.36
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.36701
##             Specificity : 0.93912
##          Pos Pred Value : 0.62815
##          Neg Pred Value : 0.84113
##              Prevalence : 0.21889
##          Detection Rate : 0.08033
##    Detection Prevalence : 0.12789
##       Balanced Accuracy : 0.65306
##
##        'Positive' Class : yes
##
```

```
## [1] "Bagging Classification tree accurately classified 81.3888888888889% of the dat
a."
```

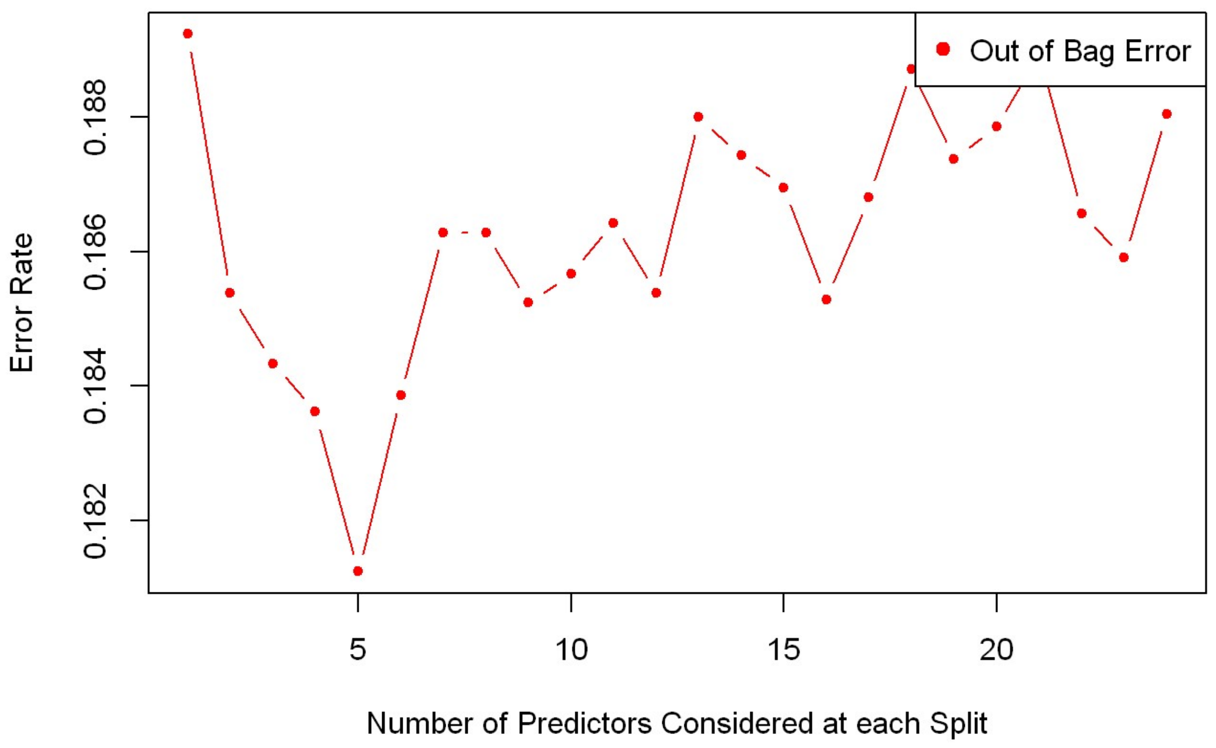
Bagging model accurately classified ~81.39% of the testing data. Out of 1970 clients that defaulted on their credit card payment, the bagging model only caught 723 cases. The bagging model has a false negative rate of ~63.3%. This is really high. Utilizing this model in real life would cause credit card companies to suffer a lot of losses as the model is bad at predicting positive cases. On the other hand, the model has a false positive rate of ~6.1%. The model performs decently at predicting negative cases. This means that the model will occassionally turn away good clients for credit card applications.

The bagging model have a higher error rate than the Lasso/Ridge model. The Ridge model misclassified 18.34% of the testing data while the Lasso model misclassified 18.36% of the testing data. The bagging model misclassified 18.61% of the data, slightly higher than the error rate of the Lasso and Ridge models. The bagging model also have a higher error rate than the logistic model, the probit model, the classification tree model, and the boostrap tree model. However, it did manage to perform better than the KNN model which had an error rate of 23.5%.

```
##                 yes         no MeanDecreaseAccuracy MeanDecreaseGini
## LIMIT_BAL  19.182476  21.769037            28.891229        450.64390
## SEX        -1.924070   5.047149             3.407947         59.22184
## EDUCATION  -2.097800   7.978057             5.730147        140.95399
## MARRIAGE   -5.607034  18.567454            14.650514         76.74884
## AGE         2.049547  29.033931            27.742907        566.06804
## PAY_0     122.649122 107.575994           177.452561       1210.79138
## PAY_2      10.393669  52.093478            63.562702        264.66679
## PAY_3      11.914752  31.755646            38.836298         86.64990
## PAY_4       5.611669  32.139219            37.862323         70.66421
## PAY_5       5.933912  30.614477            37.092647         72.63727
## PAY_6      23.664795  32.777273            48.764134         92.60930
## BILL_AMT1  22.324313  19.110551            29.591640        460.98722
## BILL_AMT2 -14.815029  45.457815            48.229967        327.06956
## BILL_AMT3 -14.115849  45.819416            48.287510        314.37423
## BILL_AMT4 -18.045527  50.522630            53.044367        304.95185
## BILL_AMT5  -5.916044  41.264169            46.513121        299.52241
## BILL_AMT6   8.218083  30.039235            39.090316        326.92237
## PAY_AMT1   -7.561595  38.189996            40.256303        341.45676
## PAY_AMT2    2.565895  30.990050            35.132920        385.78308
## PAY_AMT3   10.279756  30.654742            39.223096        369.05215
## PAY_AMT4    9.426616  29.278779            36.016264        316.62259
## PAY_AMT5    3.076627  32.911603            38.254631        327.03450
## PAY_AMT6   14.792542  31.013857            40.094221        375.98113
```

The variable *Pay_0* have the largest magnitude value by far. This indicates that the most recent payment amount made by the client is the most important variable at determining whether the payment will default or not. This makes sense. If a client doesn't pay or pays a small fraction of the amount owed, it is very likely that the client is unable to pay back what they owed. This makes the client highly at risk for defaulting. All of the other variables also have some form of effect on whether payments will default as their importantce matrix coefficients are not 0. Some examples of other notable predictors are the most recent bill amount, the balance limit on the account, and the first payment amount. These variables have a magnitude in the 20s, The balance limit, payment amount, and bill amount are all reasonable predictors for predicting whether payments will default. Higher bill amounts are harder to pay off, a low balance limit indicates lower income or credit score, and a low payment amount may indicate finacial inability to pay back what was owed.

Random Forest



Based on the graph, the error rate is the lowest when the model considers 5 predictors at each split. Therefore, we should tune the number of predictors considered at each split to be 5 in the random forest model.

mtry	OOB.Error
1	0.1892381
2	0.1853810
3	0.1843333
4	0.1836190
5	0.1812381
6	0.1838571
7	0.1862857
8	0.1862857

mtry	OOB.Error
9	0.1852381
10	0.1856667
11	0.1864286
12	0.1853810
13	0.1880000
14	0.1874286
15	0.1869524
16	0.1852857
17	0.1868095
18	0.1887143
19	0.1873810
20	0.1878571
21	0.1889524
22	0.1865714
23	0.1859048
24	0.1880476

```
## [1] "The mtry with the lowest out of bag error rate is 5"
```

```
## ntree      OOB      1      2
##   100:  18.41% 62.22%  5.90%
##   200:  18.27% 62.52%  5.63%
##   300:  18.15% 62.47%  5.49%
##   400:  18.08% 62.41%  5.41%
##   500:  18.09% 62.15%  5.50%
```

```
##
## Call:
##  randomForest(formula = default.payment.next.month ~ . - ID, data = cc_trn,      mt
ry = 5, importance = TRUE, do.trace = 100)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 18.09%
## Confusion matrix:
##      yes    no class.error
## yes 1766  2900  0.62151736
## no   899 15435  0.05503857
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  yes   no
##        yes  725  408
##        no  1245 6622
##
##                Accuracy : 0.8163
##                  95% CI : (0.8082, 0.8243)
##     No Information Rate : 0.7811
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3659
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.36802
##             Specificity : 0.94196
##          Pos Pred Value : 0.63989
##          Neg Pred Value : 0.84174
##              Prevalence : 0.21889
##          Detection Rate : 0.08056
##    Detection Prevalence : 0.12589
##       Balanced Accuracy : 0.65499
##
##        'Positive' Class : yes
##
```

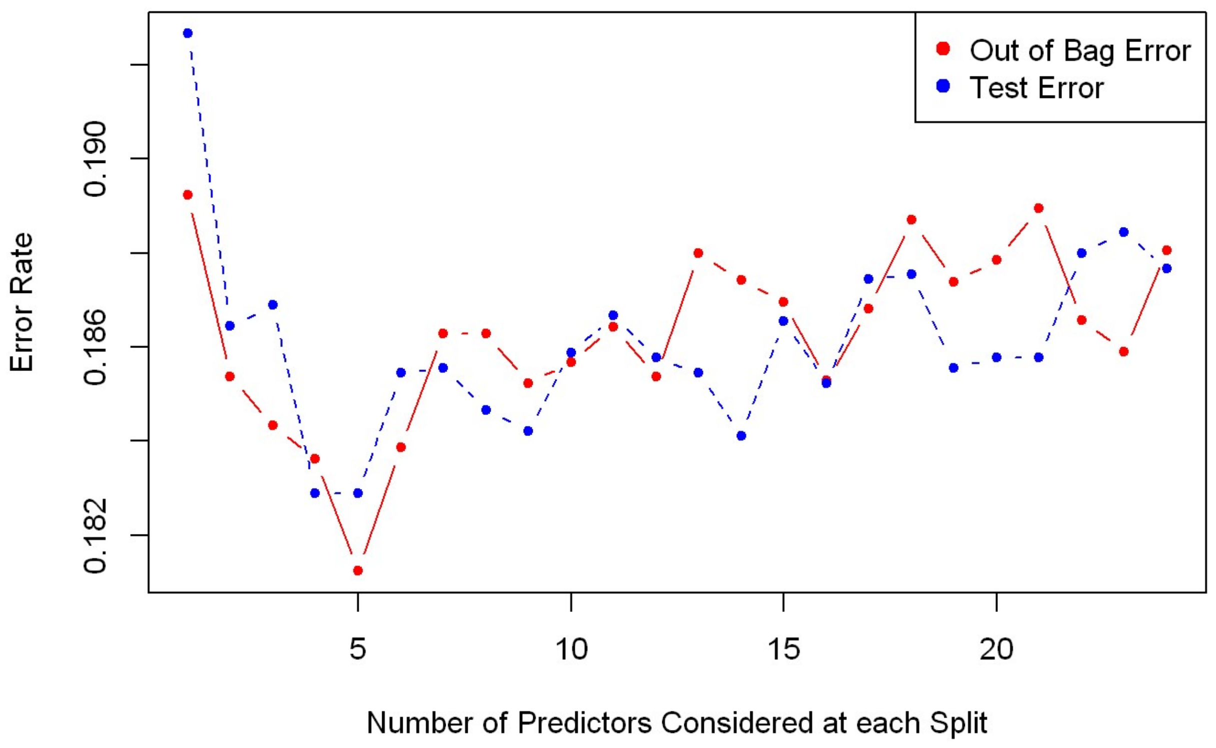
```
## [1] "Random forest model accurately classified 81.6333333333333% of the data."
```

Random Forest model accurately classified ~81.73% of the testing data. Out of 1970 clients that defaulted on their credit card payment, the Random Forest model only caught 724 cases. The Random Forest model has a false negative rate of ~63.25%. This is really high. Similar to the bagging model, utilizing the Random Forest model in real life would cause credit card companies to suffer huge losses as the model is bad at accurately predicting positive cases. On the other hand, the model has a false positive rate of ~5.7%. The model performs decently at predicting negative cases. This means that the model will occassionally turn away good clients for credit card applications. Like the bagging model, the Random Forest is bad at predicting positive cases accurately but predicts the negative cases quite well.

The Random Forest model performed better than Lasso, Ridge, and Bagging models. The Ridge model accurately classified ~81.66 percent of the test data while the lasso accurately classified ~81.63%. The bagging model accurately classified ~81.39% of the testing data. The Random Forest model accurately classified ~81.73 percent of the testing data which is higher than the correct classification of the models listed previously. However, the Random Forest model's performance does not exceed the logistic and probit regression models which correctly classified ~82% of the testing data.

```
##                   yes        no MeanDecreaseAccuracy MeanDecreaseGini
## LIMIT_BAL  19.048573 18.164629            25.133451        377.32416
## SEX         1.063040  5.551208             5.457726         66.07302
## EDUCATION  -1.808091  4.206595             2.733439        134.53591
## MARRIAGE   -2.785210 13.964114            10.914620         80.22669
## AGE         5.392862 19.735666            20.683092        425.49711
## PAY_0     103.912997 75.587913           129.723502        782.09900
## PAY_2      23.150866 28.130808            44.014582        317.19962
## PAY_3      15.870843 28.249104            38.955594        214.21010
## PAY_4      11.963017 26.702784            34.309408        166.77636
## PAY_5      10.712658 25.823804            34.345785        151.18392
## PAY_6      17.454556 26.402356            37.122580        139.00681
## BILL_AMT1  20.778633 21.220066            35.643596        433.79039
## BILL_AMT2  -5.388692 40.888934            45.442897        381.48234
## BILL_AMT3  -4.033535 40.897509            45.546185        367.16818
## BILL_AMT4  -7.299518 42.303178            49.360381        355.13977
## BILL_AMT5   1.722944 38.177107            44.981289        354.15409
## BILL_AMT6   5.070563 30.441358            38.552123        351.32450
## PAY_AMT1    5.623695 28.896515            33.309876        359.07839
## PAY_AMT2    7.386301 29.902001            36.570264        350.33819
## PAY_AMT3    7.818866 31.791731            40.467335        332.42509
## PAY_AMT4    6.025205 32.049914            38.446073        308.65597
## PAY_AMT5    4.250880 27.320017            32.390339        307.50921
## PAY_AMT6   11.655749 26.209279            32.302279        333.23011
```

The Random Forest model and the bagging model have similar importance matrices. Both consider the variable *Pay_0* to be the most important since it have the largest magnitude value. If a client doesn't pay or pays a small fraction of the amount owed, it is very likely that the client is unable to pay back what they owed. This makes the client highly at risk for defaulting. All of the other variables also have some effect on whether payments will default as their importantce matrix coefficients are not 0. Similarly, both models found variables like bill amount, balance limit, and payment amount to be significant at predicting whether credit card payments will default.



The test error follows a similar pattern to the out of bag error, therefore it is reasonable to tune the number of predictors considered at each split using the out of bag error for the Random Forest model. Both the test error and the out of bag error are the lowest when mtry, number of predictors considered at each split, equals to 5.

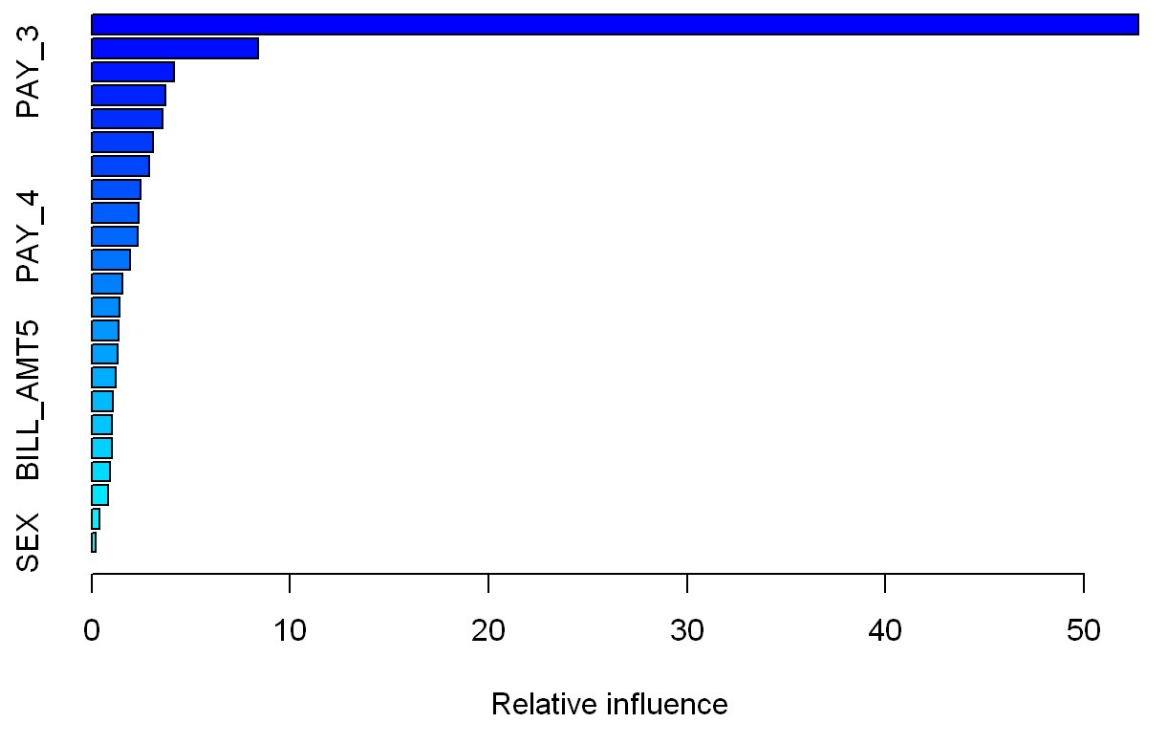
Boosting

```
## gbm(formula = default.payment.next.month ~ . - ID, distribution = "multinomial",
##     data = cc_df, n.trees = 100, interaction.depth = 4)
## A gradient boosted model with multinomial loss function.
## 100 iterations were performed.
## There were 23 predictors of which 23 had non-zero influence.
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  6689  341
##        yes 1252  718
##
##                Accuracy : 0.823
##                  95% CI : (0.815, 0.8308)
##     No Information Rate : 0.8823
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.379
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.67800
##             Specificity : 0.84234
##          Pos Pred Value : 0.36447
##          Neg Pred Value : 0.95149
##              Prevalence : 0.11767
##          Detection Rate : 0.07978
##    Detection Prevalence : 0.21889
##       Balanced Accuracy : 0.76017
##
##        'Positive' Class : yes
##
```

```
## [1] "Boosting Classification tree accurately classified 82.3% of the data."
```

Boosting model accurately classified ~82.26% of the testing data. Out of 1065 clients that defaulted on their credit card payment, the boosting model missed 346 cases. The boosting model has a false negative rate of ~32.49%. While this is not as high as the bagging/Random Forest model, it would still induce losses from the defaulting clients that the model missed. On the other hand, the model has a false positive rate of ~15.77%. Unlike the random forest and bagging models, the boosting model is slightly worse at predicting negative cases. This means that the model will sometimes turn away good clients for credit card applications. When compared to the Random Forest model, it appears that the boosting model is slightly better at predicting positive cases but slightly worse at predicting negative cases.

The boosting model have the lowest error rate out of all the models ran so far. The boosting model accurately classified 82.26% of the data. The logistic model accurately predicted ~82.15% and probit model accurately predicted ~82.18%. The lasso model and ridge model both accurately predicted ~81.6% of the data. The random forest correctly classified ~81.73% of th testing data and the bagging model correctly classified ~81.33%. The boosting model classified most testing cases correctly and have the lowest error rate thus far.



	var <fctr>	rel.inf <dbl>
PAY_0	PAY_0	52.7461878
PAY_2	PAY_2	8.3637947
PAY_3	PAY_3	4.1688143
LIMIT_BAL	LIMIT_BAL	3.7268911
PAY_5	PAY_5	3.5658320
BILL_AMT1	BILL_AMT1	3.1099675
PAY_6	PAY_6	2.8997372
PAY_AMT3	PAY_AMT3	2.4803079
PAY_AMT1	PAY_AMT1	2.3649345
PAY_4	PAY_4	2.3277529

1-10 of 23 rows   Previous **1** 2 3 Next

Similar to the bagging and random forest model, the boosting model also considers *PAY_0* to be the most important variable. *PAY_0* have the highest relative influence out of all the predictors. In addition, the boosting model also consider all other predictors to have an effect on whether credit card payments will default. However, some variables have less of an effect when compared to others. An example of this would be the variables *MARRIAGE* with a relative influence of 0.7498291 and *SEX* with a relative influence of 0.4501672 . The relative influence of these variables are close to 0 which indicates that the effect of these independent variables does not have a strong impact on the dependent variable. This seems reasonable. As society gets more and more progressive, we see gender equality in more places. More and more women are paid about the same as their male counterparts. Marriage also does not have a strong impact on defaulting. This may be due to different tax brackets for singles and couples or other factors. However, that is another issue.

For the most part, Variables like payment amount, bill amount, limit balance, age have similar relative influences. All of these are pretty useful for predicting whether clients will default on their credit card payments. As stated previously, these variables do make sense in real life scenarios.

XGboost

```
## [1]  train-error:0.170667 
## [11] train-error:0.160286 
## [21] train-error:0.153238 
## [31] train-error:0.148000 
## [41] train-error:0.142762 
## [51] train-error:0.136190 
## [61] train-error:0.131381 
## [71] train-error:0.125810 
## [81] train-error:0.121667 
## [91] train-error:0.113286 
## [100]    train-error:0.107667
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  689  417
##          1 1281 6613
##
##                Accuracy : 0.8113
##                  95% CI : (0.8031, 0.8194)
##     No Information Rate : 0.7811
##     P-Value [Acc > NIR] : 9.506e-13
##
##                   Kappa : 0.3449
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.34975
##             Specificity : 0.94068
##          Pos Pred Value : 0.62297
##          Neg Pred Value : 0.83772
##              Prevalence : 0.21889
##          Detection Rate : 0.07656
##    Detection Prevalence : 0.12289
##       Balanced Accuracy : 0.64521
##
##        'Positive' Class : 0
##
```

```
## [1] "XGBoost model accurately classified 81.1333333333333% of the data."
```

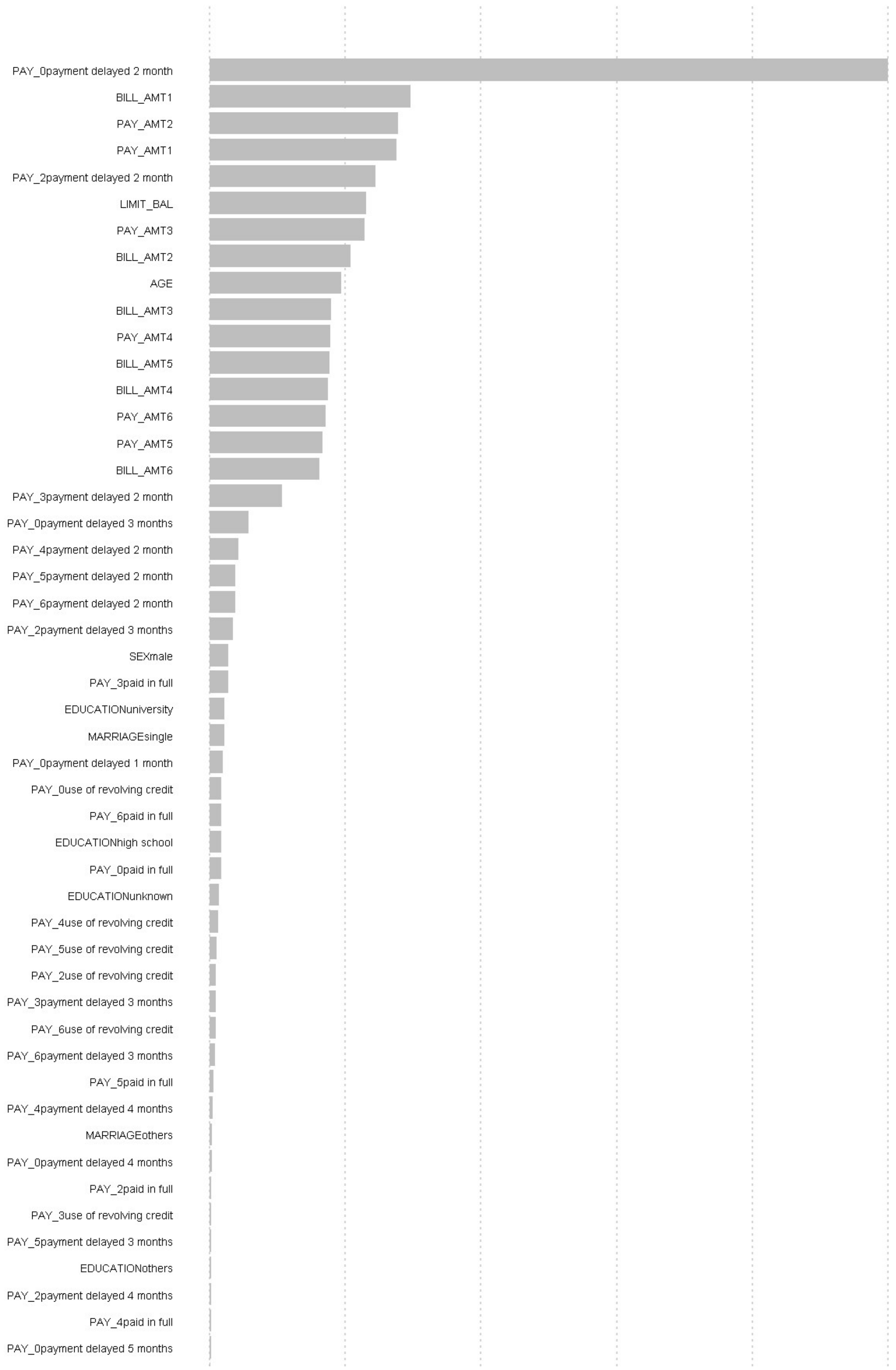
XGBoost model accurately classified ~81.13% of the testing data. Out of 1970 clients that defaulted on their credit card payment, the bagging model only caught 689 cases. The XGBoost model has a false negative rate of ~65%. This is really high. Utilizing this model in real life would cause credit card companies to suffer huge losses as the model is bad at predicting positive cases. On the other hand, the model has a false positive rate of ~5.9%. The model performs decently at predicting negative cases. This means that the model will occassionally turn away good clients for credit card applications.

The XGBoost have a higher error rate than the Lasso, Ridge, Bagging, boosting, and random forest models. The bagging and XGboost models both have similar test error rates. Both correctly classified ~81.13% of the data. The boosting model accurately classified 82.26% of the data. The logistic model accurately predicted ~82.15% and probit model accurately predicted ~82.18%. The lasso model and ridge model both accurately predicted ~81.6% of the data. The random forest correctly classified ~81.73% of th testing data.

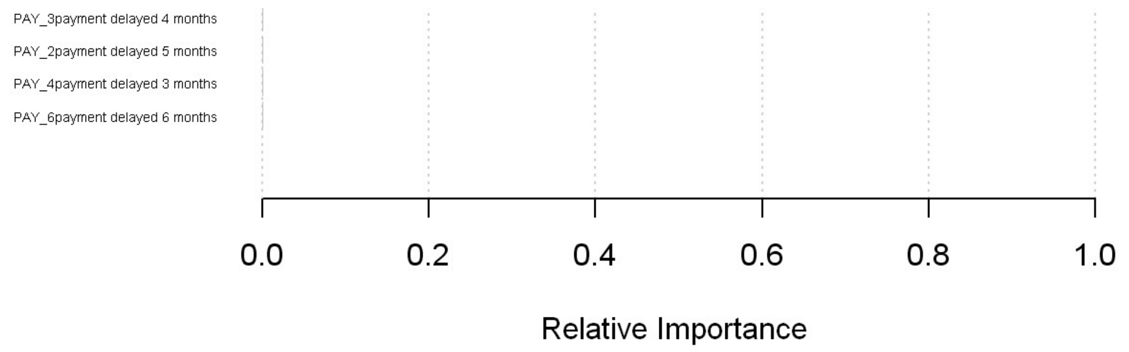
Feature	Gain	Cover	Frequency
<chr>	<dbl>	<dbl>	<dbl>

Feature	Gain	Cover	Frequency
<chr>	<dbl>	<dbl>	<dbl>
PAY_0payment delayed 2 month	0.1963003233	3.076860e-02	0.0060814384
BILL_AMT1	0.0581180306	6.146263e-02	0.0695399260
PAY_AMT2	0.0544573373	5.675712e-02	0.0608143839
PAY_AMT1	0.0540706557	7.011576e-02	0.0613432047
PAY_2payment delayed 2 month	0.0480671988	8.614280e-03	0.0031729244
LIMIT_BAL	0.0452079397	3.589075e-02	0.0515600212
PAY_AMT3	0.0447751944	5.441775e-02	0.0618720254
BILL_AMT2	0.0407288235	8.821172e-02	0.0542041248
AGE	0.0380825701	1.949846e-02	0.0605499736
BILL_AMT3	0.0352121531	5.987515e-02	0.0520888419

1-10 of 53 rows Previous **1** 2 3 4 5 6 Next



PAY_0payment delayed 2 month
BILL_AMT1
PAY_AMT2
PAY_AMT1
PAY_2payment delayed 2 month
LIMIT_BAL
PAY_AMT3
BILL_AMT2
AGE
BILL_AMT3
PAY_AMT4
BILL_AMT5
BILL_AMT4
PAY_AMT6
PAY_AMT5
BILL_AMT6
PAY_3payment delayed 2 month
PAY_0payment delayed 3 months
PAY_4payment delayed 2 month
PAY_5payment delayed 2 month
PAY_6payment delayed 2 month
PAY_2payment delayed 3 months
SEXmale
PAY_3paid in full
EDUCATIONuniversity
MARRIAGEsingle
PAY_0payment delayed 1 month
PAY_0use of revolving credit
PAY_6paid in full
EDUCATIONhigh school
PAY_0paid in full
EDUCATIONunknown
PAY_4use of revolving credit
PAY_5use of revolving credit
PAY_2use of revolving credit
PAY_3payment delayed 3 months
PAY_6use of revolving credit
PAY_6payment delayed 3 months
PAY_5paid in full
PAY_4payment delayed 4 months
MARRIAGEothers
PAY_0payment delayed 4 months
PAY_2paid in full
PAY_3use of revolving credit
PAY_5payment delayed 3 months
EDUCATIONothers
PAY_2payment delayed 4 months
PAY_4paid in full
PAY_0payment delayed 5 months



The higher the gain, the more effect the predictor has on the dependent variable. Like the previous models, the XGboost model also found the variable *PAY_0* to be the most important variable for predicting whether credit card payments will default or not. Variables with relative importance close to 0 does not impact the dependent variable significantly. The XGBoost model found the older payment amount variables and older bill amounts to be not as important as the more recent counterparts. This seems reasonable since recent information are more up to date and reflects the client's situation better than information in the past. According to the XGBoost model, recent payment amounts, bill amounts, limit balance, and age are good indicators of credit card payment defaults.

Tuned Boosting

```
##     shrinkage interaction.depth n.trees
## 128      0.05                 4      80
```

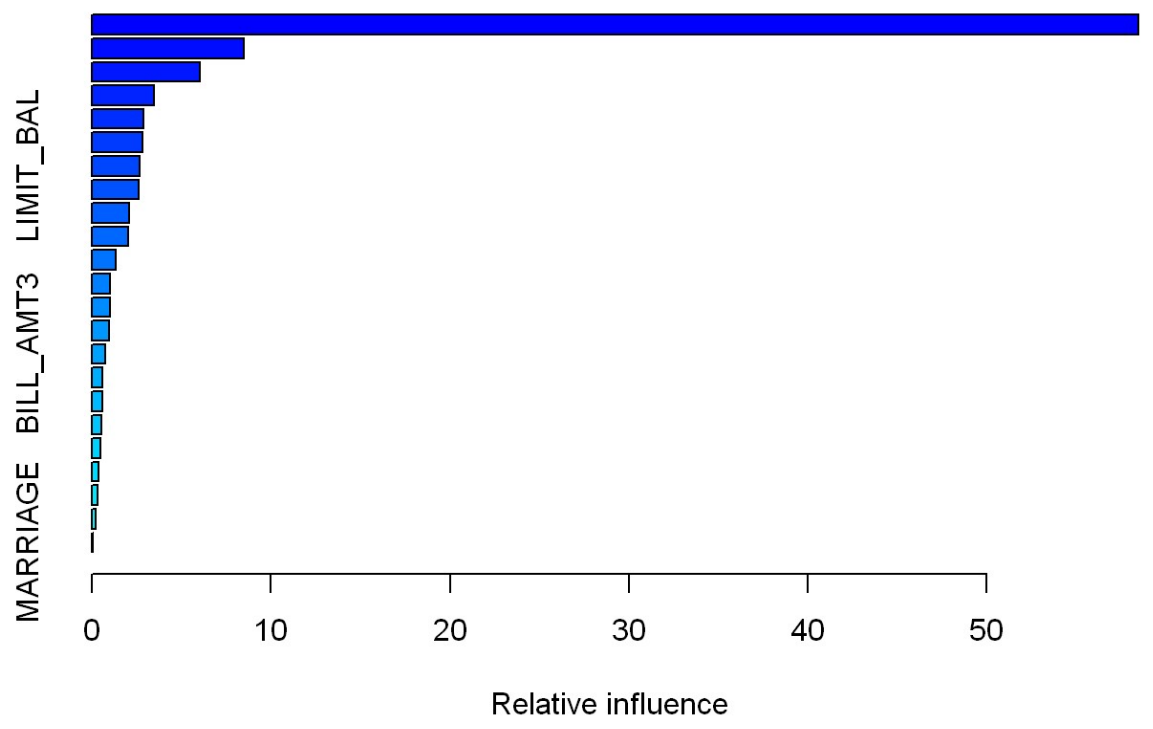
The model with the lowest test error is the 128th gbm model with .05 shrinkage, 4 interaction depth, and 80 trees.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  6673  357
##        yes 1279  691
##
##                Accuracy : 0.8182
##                  95% CI : (0.8101, 0.8261)
##     No Information Rate : 0.8836
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3607
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.65935
##             Specificity : 0.83916
##          Pos Pred Value : 0.35076
##          Neg Pred Value : 0.94922
##              Prevalence : 0.11644
##          Detection Rate : 0.07678
##    Detection Prevalence : 0.21889
##       Balanced Accuracy : 0.74926
##
##        'Positive' Class : yes
##
```

```
## [1] "Tuned boosting model accurately classified 81.8222222222222% of the data."
```

Tuned boosting model accurately classified ~81.68% of the testing data. Out of 1043 clients that defaulted on their credit card payment, the tuned boosting model only caught 391 cases. The tuned boosting model has a false negative rate of ~34.61%. The margin of error is still really high. Utilizing this model in real life would cause credit card companies to suffer huge losses as the model is bad at predicting positive cases. On the other hand, the model has a false positive rate of ~16.19%. The model also doesn’t do very well when predicting negative cases. This means that the model will decline many good clients for credit cards.

The tuned boosting model have an error rate of 18.32%. It have a lower error rate than the XGBoosting, bagging, Ridge, Lasso, and KNN models. However, the tuned boosting model have a higher error rate than the boosting, random forest, and classification tree models. The tuned boosting model have a higher error rate than the previous boosting model. This may be caused by not having enough variation in hyperparameters during the grid search. However, grid search is computationally intensive so a more sophisticated method should be used instead.



	var <fctr>	rel.inf <dbl>
PAY_0	PAY_0	58.45782012
PAY_2	PAY_2	8.50497418
PAY_3	PAY_3	6.04110847
PAY_6	PAY_6	3.46194220
BILL_AMT1	BILL_AMT1	2.87928841
PAY_5	PAY_5	2.81980807
LIMIT_BAL	LIMIT_BAL	2.69969159
PAY_AMT3	PAY_AMT3	2.62729811
PAY_AMT1	PAY_AMT1	2.07890760
PAY_AMT2	PAY_AMT2	2.04699511

1-10 of 23 rows Previous **1** 2 3 Next

Both the boosting model and the tuned version have similar importance matrices. Both found the variable *PAY_0* to be the most important variable. The amount paid by the client can reveal critical information about the client’s ability to pay back what they owe. If a client does not pay off a good chunk of their bill, it is reasonable to assume that the client is at risk for defaulting. The tuned boosting model found all variables to have an effect on default payments but some variables have a weaker effect. The variables with relative influence close to 0 have less of an impact than variables with high relative influence. Variables like *SEX*, *MARRIAGE*, and *BILL_AMT6* have a very weak influence on whether credit card payments will default or not. The model also only found recent bill amounts to be useful. It did not consider older bill amounts to be useful in predicting the dependent variable. This makes sense since bill amounts in the past does not provide a lot of relevant information in the present.

Neural Network

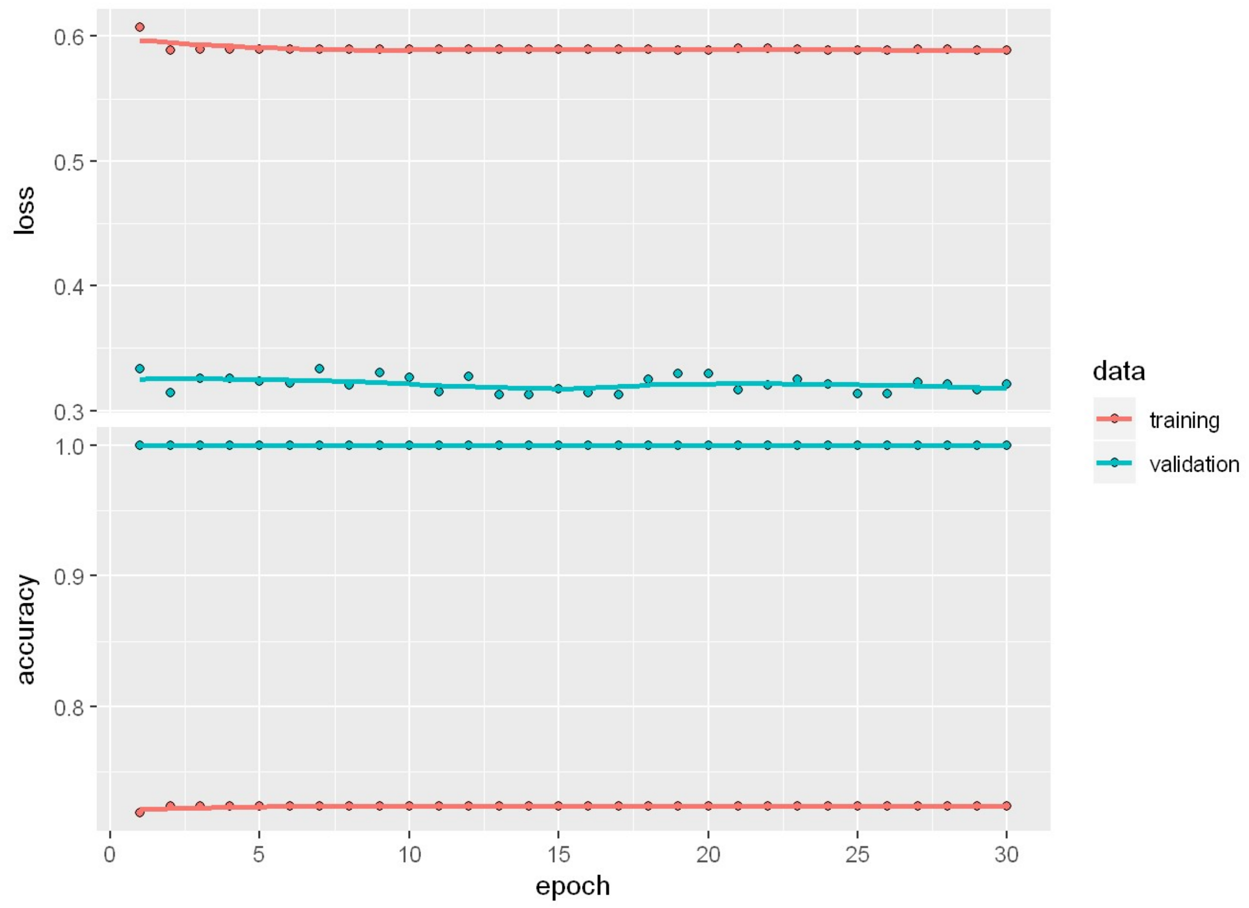
```
#neural network model
set.seed(490)
model = keras_model_sequential() %>%
  layer_dense(units = 64, activation = 'softmax',
              input_shape = dim(train_data)[2]) %>%
  layer_dense(units = 64, activation = 'softmax') %>%
  layer_dense(units = 2, activation = 'softmax')

model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
summary(model)
```

```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## dense (Dense)                       (None, 64)                      1664
## ________________________________________________________________________________
## dense_1 (Dense)                     (None, 64)                      4160
## ________________________________________________________________________________
## dense_2 (Dense)                     (None, 2)                       130
## ================================================================================
## Total params: 5,954
## Trainable params: 5,954
## Non-trainable params: 0
## ________________________________________________________________________________
```

The neural network model consists of two hidden layers. Each hidden layer have 64 nodes. The outer layer consists of 2 nodes since the output is binary. All of the layers utilizes the softmax activation function. Since the dependent variable is a binary categorical variable, I will use categorical crossentropy as the loss function. The RMSprop optimization algorithm will be used to optimize the model. The metric used to assess the model’s performance will be accuracy, how well the model can classify the dependent variables correctly.

```
set.seed(490)
early_stop = callback_early_stopping(monitor = 'val_loss', patience = 20)
epochs = 30
history_class = model %>% fit(
  train_data,
  train_labels,
  epochs = epochs,
  validation_split = .2,
  callbacks = list(early_stop)
)
```

20% of the training data is split to validate the model while the rest of the training data is used to train the model. The patience is 20 wwhich means the model only accepts increase in loss in the validation set for 20 epochs. The models stops once we reach this threshhold after 25 epochs.



The graphs depict the loss and accuracy in both the training data and the validation data. The validation accuracy is higher than the traaining accuracy. This means that we aren’t overfitting the data. However, it appears that the dataset is highly imbalanced which may affect the performance of the neural network. This may be the reason why the validation accuracy and the training accuracy have such a huge difference.

```
##    test_class
##        0
##   0 7010
##   1 1991
```

Our model misclassifies ~22.2% of the testing data. The testing dataset was randomly sampled. Due to the imbalanced proportion of classes in the whole dataset, the testing dataset have no positive cases for the model to predict on. Therefore, we do not know how well the model can predict on positive cases. Since the misclassification rate of positive cases is unknown, we should not use this model in real life. Since this dataset is imbalanced, there are more negative cases than positive cases. As a result, the model is not given enough positive samples to be trained to accurately detect whether clients will default on their credit card payments.

The neural network model slightly outperformed the KNN model.The KNN model suffers from the curse of high dimensionality and is still the worse performing model by far. The KNN model accurately classified ~76.5% of the data while the neural network only classified ~77.8% of the testing data. However, the other models from the previous section with the exception of KNN all outperformed the neural network model and have a higher accuracy than the neural network model.

Results

Model	**Best_Tune**	**Accuracy**
Logistic Regression Model	NA	0.8214776
Probit Regression Model	NA	0.8217621
K’s Nearest Neighbor	k = 9	0.7649999
Ridge Regression	lambda = 0.0149514301573017	0.8165556
Lasso Regression	lambda = 0.0014271864520944	0.8163333
Classification Tree	Termind Nodes = 2	0.8168889
Bootstrapped Tree	NA	0.8168889
Bagging Model	mtry = 24	0.8138889

Model	Best_Tune	Accuracy
Random Forest	mtry = 5	0.8163333
Boosting	number of trees = 100, and interaction depth = 4	0.8230000
XGBoost	Number of rounds = 100	0.8113333
Boosting (Tuned)	shrinkage = .05, number of trees = 80, interaction depth = 4	0.8182222
Neural Network	number of epochs = 21	0.7788023

Conclusion

Based off the models I ran so far, the logistic regression model classified most of the testing data correctly. Probit regression also performed similarly to the logistic regression. On the other hand, K's nearest neighbor performed very badly on the data. This may be due to the large amount of features in the dataset. Both logistic regression and probit regression model found features that had a significant effect on the depedent variable. Such variables include payment statuses during certain months, payment amount in the first few months, gender, education status, and marraige status. However, some of the results are illogical. The models claim that client's owe more to the card companies, the odds of defaulting increases. This makes sense since people with high payment amounts are usually bad with money management and live a lifestyle they cannot sustain. However, the model also says that delayed payments reduce the chance of defaulting. It would make more sense if delayed payments increase the chance of defaulting since it indicates that the client cannot financiially pay back the amount owed. Different models should be trained in order to find better relationships between X's and Y.

After running classification trees, ridge regression, lasso regression, and training a neural network, the KNN model still have the highest error rate. The KNN model is very likely to suffer from the curse of high dimensionality. The neural network slightly outperforms the KNN model. The neural network model classified 77.9% of the testing data accurately while the KNN model only accurately classified ~76.5% of the test data. It appears that there is not enough positive cases in the dataset to train the neural network better. The performance of the neural network on positive cases is also unknown. In addition, the neural network model also suffers from a lack of feature detection. The downside of neural networks are that is hard to comprehend in human languages. Humans do not know how the network is classifying the data while other models have some sort of feature selection or variable coefficients. Therefore, the neural network and the KNN model both does not seem to be a good fit for our dataset. On the other hand, both ridge and lasso models have identical performance with ridge classifying slightly more data correctly. The classification tree classified have a lower error rate than the ridge and lasso model. However, pruning and bootstrapping the tree did not yield a lower error rate. These new models can still not beat the logistic and probit regression model which managed to correctly classify ~82% of the data. While these models have the lowest error rate, it is important to note that some of the coefficient estimates of the models are illogical in a real world scenario. On the other hand, the coefficient estimates from the ridge and lasso model seems to be more reasonable and provides a better context in the real world. In addition, the bagging, random forest, XGBoost, and the tuned boosting model also have similar error rates. These models managed to classify ~81% of the testing data correctly. However, the untuned boosting model with 100 trees and 4 interaction depth managed to slightly outperform the logistic and probit regression models. The boosting model accurately predicted ~82.39% of the data. The boosting model classified the most cases correctly by far.

Boosting, the best performing model, found *PAY_O*, the most recent payment made by the client, to be the most important variable at predicting whether credit card payments will default. The model also found other recent payments and bill amount to have a significant effect on the dependent variable. The model did not find older payments and bill amounts to have a big impact on defaulting. This makes sense because financial information from the past does not provide as much relevant information from financial information in the present especially when things can change over time. The model also found limit balance to be significant in predicting default payments. Clients with a good credit score or a high income tend to have higher balance limits in their credit accounts. Out of the demographic variables (age, sex, marriage, and education), the model only found age and education to have a strong impact on default payments. This also makes sense. Gender wage gap is slowly decreasing and more women are working in today’s society. Therefore, both gender have access to well paying jobs. Younger people generally have more entry level positions which pays less than the senior positions or younger people might have student loans. Therefore, age does have an effect on whether credit card payments will default or not. As for education, people that have graduate degrees tend to have more opportunities for higher paying jobs. The importance matrix of the boosting model seems reasonable and makes sense in real world context.

Appendix

Libraries

```
knitr::opts_chunk$set(message = FALSE, echo = FALSE, warning = FALSE, fig.align = 'cen
ter', cache = TRUE)
library(ggplot2)
library(gridExtra)
library(dplyr)
library(caret)
library(corrplot)
library(e1071)
library(randomForest)
library(tibble)
library(kableExtra)
library(glmnet)
library(tree)
library(purrr)
library(boot)
library(gbm)
library(xgboost)
library(rpart.plot)
library(ISLR)
```

Data Cleaning

```
cc_df = read.csv("UCI_Credit_Card.csv")
cc_df$default.payment.next.month = as.factor(cc_df$default.payment.next.month)
cc_df$SEX = case_when(cc_df$SEX == 1 ~ 'male',
                      TRUE ~ 'female')
cc_df$EDUCATION = case_when(cc_df$EDUCATION == 1 ~ 'graduate school',
                            cc_df$EDUCATION == 2 ~ 'university',
                            cc_df$EDUCATION == 3 ~ 'high school',
                            cc_df$EDUCATION == 4 ~ 'others',
                            TRUE ~ 'unknown')
cc_df$MARRIAGE = case_when(cc_df$MARRIAGE == 1 ~ 'married',
                          cc_df$MARRIAGE == 2 ~ 'single',
                          TRUE ~ 'others')
cc_df$PAY_0 = case_when(cc_df$PAY_0 == -2 ~ 'no consummption',
                       cc_df$PAY_0 == -1 ~ 'paid in full',
                       cc_df$PAY_0 == 0 ~ 'use of revolving credit',
                       cc_df$PAY_0 == 1 ~ 'payment delayed 1 month',
                       cc_df$PAY_0 == 2 ~ 'payment delayed 2 month',
                       cc_df$PAY_0 == 3 ~ 'payment delayed 3 months',
                       cc_df$PAY_0 == 4 ~ 'payment delayed 4 months',
                       cc_df$PAY_0 == 5 ~ 'payment delayed 5 months',
                       cc_df$PAY_0 == 6 ~ 'payment delayed 6 months',
                       cc_df$PAY_0 == 7 ~ 'payment delayed 7 months',
                       cc_df$PAY_0 == 8 ~ 'payment delayed 8 months',
                       cc_df$PAY_0 == 9 ~ 'payment delayed 9 months or above')
cc_df$PAY_2 = case_when(cc_df$PAY_2 == -2 ~ 'no consummption',
                       cc_df$PAY_2 == -1 ~ 'paid in full',
                       cc_df$PAY_2 == 0 ~ 'use of revolving credit',
                       cc_df$PAY_2 == 1 ~ 'payment delayed 1 month',
                       cc_df$PAY_2 == 2 ~ 'payment delayed 2 month',
                       cc_df$PAY_2 == 3 ~ 'payment delayed 3 months',
                       cc_df$PAY_2 == 4 ~ 'payment delayed 4 months',
                       cc_df$PAY_2 == 5 ~ 'payment delayed 5 months',
                       cc_df$PAY_2 == 6 ~ 'payment delayed 6 months',
                       cc_df$PAY_2 == 7 ~ 'payment delayed 7 months',
                       cc_df$PAY_2 == 8 ~ 'payment delayed 8 months',
                       cc_df$PAY_2 == 9 ~ 'payment delayed 9 months or above')
cc_df$PAY_3 = case_when(cc_df$PAY_3 == -2 ~ 'no consummption',
                       cc_df$PAY_3 == -1 ~ 'paid in full',
                       cc_df$PAY_3 == 0 ~ 'use of revolving credit',
                       cc_df$PAY_3 == 1 ~ 'payment delayed 1 month',
                       cc_df$PAY_3 == 2 ~ 'payment delayed 2 month',
                       cc_df$PAY_3 == 3 ~ 'payment delayed 3 months',
                       cc_df$PAY_3 == 4 ~ 'payment delayed 4 months',
                       cc_df$PAY_3 == 5 ~ 'payment delayed 5 months',
                       cc_df$PAY_3 == 6 ~ 'payment delayed 6 months',
                       cc_df$PAY_3 == 7 ~ 'payment delayed 7 months',
                       cc_df$PAY_3 == 8 ~ 'payment delayed 8 months',
                       cc_df$PAY_3 == 9 ~ 'payment delayed 9 months or above')
```

```
cc_df$PAY_4 = case_when(cc_df$PAY_4 == -2 ~ 'no consummption',
                        cc_df$PAY_4 == -1 ~ 'paid in full',
                        cc_df$PAY_4 == 0 ~ 'use of revolving credit',
                        cc_df$PAY_4 == 1 ~ 'payment delayed 1 month',
                        cc_df$PAY_4 == 2 ~ 'payment delayed 2 month',
                        cc_df$PAY_4 == 3 ~ 'payment delayed 3 months',
                        cc_df$PAY_4 == 4 ~ 'payment delayed 4 months',
                        cc_df$PAY_4 == 5 ~ 'payment delayed 5 months',
                        cc_df$PAY_4 == 6 ~ 'payment delayed 6 months',
                        cc_df$PAY_4 == 7 ~ 'payment delayed 7 months',
                        cc_df$PAY_4 == 8 ~ 'payment delayed 8 months',
                        cc_df$PAY_4 == 9 ~ 'payment delayed 9 months or above')
cc_df$PAY_5 = case_when(cc_df$PAY_5 == -2 ~ 'no consummption',
                        cc_df$PAY_5 == -1 ~ 'paid in full',
                        cc_df$PAY_5 == 0 ~ 'use of revolving credit',
                        cc_df$PAY_5 == 1 ~ 'payment delayed 1 month',
                        cc_df$PAY_5 == 2 ~ 'payment delayed 2 month',
                        cc_df$PAY_5 == 3 ~ 'payment delayed 3 months',
                        cc_df$PAY_5 == 4 ~ 'payment delayed 4 months',
                        cc_df$PAY_5 == 5 ~ 'payment delayed 5 months',
                        cc_df$PAY_5 == 6 ~ 'payment delayed 6 months',
                        cc_df$PAY_5 == 7 ~ 'payment delayed 7 months',
                        cc_df$PAY_5 == 8 ~ 'payment delayed 8 months',
                        cc_df$PAY_5 == 9 ~ 'payment delayed 9 months or above')
cc_df$PAY_6 = case_when(cc_df$PAY_6 == -2 ~ 'no consummption',
                        cc_df$PAY_6 == -1 ~ 'paid in full',
                        cc_df$PAY_6 == 0 ~ 'use of revolving credit',
                        cc_df$PAY_6 == 1 ~ 'payment delayed 1 month',
                        cc_df$PAY_6 == 2 ~ 'payment delayed 2 month',
                        cc_df$PAY_6 == 3 ~ 'payment delayed 3 months',
                        cc_df$PAY_6 == 4 ~ 'payment delayed 4 months',
                        cc_df$PAY_6 == 5 ~ 'payment delayed 5 months',
                        cc_df$PAY_6 == 6 ~ 'payment delayed 6 months',
                        cc_df$PAY_6 == 7 ~ 'payment delayed 7 months',
                        cc_df$PAY_6 == 8 ~ 'payment delayed 8 months',
                        cc_df$PAY_6 == 9 ~ 'payment delayed 9 months or above')
cc_df$default.payment.next.month = case_when(cc_df$default.payment.next.month == 0 ~
'no',
                                             TRUE ~ 'yes')

i = sapply(cc_df, is.character)
cc_df[i] = lapply(cc_df[i], as.factor)

cc_df$default.payment.next.month = factor(cc_df$default.payment.next.month, levels=rev
(levels(cc_df$default.payment.next.month)))
```

Data Split

```r
set.seed(490)
trn_idx = sample(nrow(cc_df), size = .7 * nrow(cc_df))
cc_trn = cc_df[trn_idx, ]
cc_tst = cc_df[-trn_idx, ]
```

Exploratory Data Analysis

```r
plot_a = ggplot(cc_df, aes(AGE)) +
  geom_histogram(bins = 10, color="black", fill="grey") +
  labs(title = 'Distribution of Age',
       x = 'Age', y = 'Count')
plot_a
```

```r
plot_b = ggplot(cc_df, aes(MARRIAGE, fill = MARRIAGE)) +
  geom_bar() +
  labs(title = 'Distribution of Marital Status',
       x = 'Marital Status', y = 'Count')
plot_b
```

```r
plot_c = ggplot(cc_df, aes(EDUCATION, fill = EDUCATION)) +
  geom_bar() +
  labs(title = 'Distribution of Education',
       x = 'Education', y = 'Count') +
  theme(axis.text.x = element_text(size = 8))
plot_c
```

```r
plot_d = ggplot(cc_df, aes(PAY_0, fill = PAY_0)) +
  geom_bar()+
  labs(title = 'Distribution of Payment Status in Sept 2005',
       x = 'Payment Status in Sept 2005', y = 'Count') +
  theme(axis.text.x = element_text(size = 8, angle = 45, hjust = 1))
plot_d
```

```r
#make a box boxplot
plot_1 = ggplot(cc_df, aes(AGE, fill = default.payment.next.month)) +
  geom_histogram(binwidth = 1) +
  labs(title = 'Age vs Default',
       x = 'Age', y = 'Count')

plot_1
```

```
plot_2 = ggplot(cc_df, aes(x = MARRIAGE, fill = default.payment.next.month)) +
  geom_bar(position = 'fill') +
  labs(title = 'Marriage vs Default',
       x = 'Mariage', y = 'Count')
plot_2
```

```
plot_3 = ggplot(cc_df, aes(x = EDUCATION, fill = default.payment.next.month)) +
  geom_bar(position = 'fill') +
  labs(title = 'Education vs Default', x = 'Education', y = 'Count') +
  theme(axis.text.x = element_text(size = 8))
plot_3
```

```
plot_4 = ggplot(cc_df, aes(PAY_0, fill = default.payment.next.month)) +
  geom_bar(position = 'fill') +
  labs(title = 'Repayment Status in Sept 2005 vs Default', x = 'Repayment Status in Se
pt 2005', y = 'Count') +
  theme(axis.text.x = element_text(size = 8, angle = 45, hjust = 1))
plot_4
```

```
plot_5 = ggplot(cc_df, aes(default.payment.next.month, BILL_AMT1, color = default.paym
ent.next.month)) +
  geom_boxplot() +
  labs(title = 'Bill Statement in Sept 2005 vs Default', x = 'Default', y = 'Credit Ba
lance in Sept 2005')
plot_5
```

Logistic Regression

```
set.seed(490)
lr_mod1 = train(default.payment.next.month ~ . - ID, data = cc_trn, method = 'glm', fa
mily = 'binomial', trControl = trainControl(method = 'cv', number = 10), metric = 'Acc
uracy')

lr_mod2 = train(default.payment.next.month ~ . -ID - PAY_0 - PAY_2 - AGE - PAY_3 - MAR
RIAGE,  data = cc_trn, method = 'glm', family = 'binomial', trControl = trainControl(m
ethod = 'cv', number = 10), metric = 'Accuracy')

lr_mod3 = train(default.payment.next.month ~ LIMIT_BAL + AGE + EDUCATION + PAY_0 + PAY
_2, data = cc_trn, method = 'glm', family = 'binomial', trControl = trainControl(metho
d = 'cv', number = 10), metric = 'Accuracy')

lr_mod4 = train(default.payment.next.month ~ LIMIT_BAL + EDUCATION + MARRIAGE + AGE +
PAY_0 + BILL_AMT1, data = cc_trn, method = 'glm', family = 'binomial', trControl = tra
inControl(method = 'cv', number = 10), metric = 'Accuracy')

lr_mod5 = train(default.payment.next.month  ~ LIMIT_BAL + EDUCATION + MARRIAGE + AGE +
PAY_0 + PAY_2 +PAY_3 + PAY_4 +PAY_5 + PAY_6, data = cc_trn, method = 'glm', family =
'binomial', trControl = trainControl(method = 'cv', number = 10), metric = 'Accuracy')
```

```
results = tibble(
  Model = c("Logistic Regression Model 1",
            "Logistic Regression Model 2",
            "Logistic Regression Model 3",
            "Logistic Regression Model 4",
            "Logistic Regression Model 5"),
  Accuracy = c(lr_mod1$results[,2],
               lr_mod2$results[,2],
               lr_mod3$results[,2],
               lr_mod4$results[,2],
               lr_mod5$results[,2])
)

kable(results) %>%
  kable_styling("striped", full_width = F)
```

```
lr_mod1_summary = summary(lr_mod1)
lr_mod1_summary$coefficients[-1,1][lr_mod1_summary$coefficients[ ,4] < 0.05][-20]
```

```r
# Plotting coefficient and standard error of significant features
coefs = as.data.frame(lr_mod1_summary$coefficients[lr_mod1_summary$coefficients[,4] <
0.05,1:2]) # -1 is to exclude the intercept
names(coefs)[2] = "se"
coefs$vars = rownames(coefs)
ggplot(coefs, aes(vars, Estimate)) +
geom_errorbar(aes(ymin=Estimate - 1.96*se, ymax=Estimate + 1.96*se), lwd=1, colour="re
d", width=0) +
geom_errorbar(aes(ymin=Estimate - se, ymax=Estimate + se), lwd=1.5, colour="blue", wid
th=0) +
geom_point(size=2, pch=21, fill="yellow") +
theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

Probit Regression

```r
probit_mod = train(default.payment.next.month ~ . - ID, data = cc_trn, method = 'glm',
family = 'binomial'(link="probit"), trControl = trainControl(method = 'cv', number = 1
0), metric = 'Accuracy')
probit_mod
```

```r
probit_mod_summary = summary(probit_mod)
probit_mod_summary$coefficients[-1,1][probit_mod_summary$coefficients[,4] < 0.05][-19]
```

K’s Nearest Neighbors

```r
knn_mod = train(default.payment.next.month ~ . - ID, data = cc_trn, method = "knn", tr
Control = trainControl(method = 'cv', number = 10), metric = 'Accuracy')
knn_mod
```

Ridge Regression

```r
set.seed(490)
cc_trn_x = model.matrix(default.payment.next.month ~ ., data = cc_trn)[,-1]
cc_tst_x = model.matrix(default.payment.next.month ~ ., data = cc_tst)[,-1]
ridge_mod = cv.glmnet(cc_trn_x, cc_trn$default.payment.next.month, alpha = 0, family =
'binomial')
ridge_mod
```

```r
ridge_best_lambda = ridge_mod$lambda.min
paste0('The lambda within one standard error of the minimum lambda is ',ridge_best_lam
bda)
```

```r
plot(ridge_mod, sub = 'Training MSE as a Function of Lambda in Ridge Regression')
```

```r
cc_df_x = model.matrix(default.payment.next.month ~ ., data = cc_df)[,c(-1,-2)]
ridge_mod_full_data = glmnet(cc_df_x, cc_df$default.payment.next.month, alpha = 0, fam
ily = 'binomial')
plot(ridge_mod_full_data, xvar = "lambda", sub = 'Coefficients for Different Values of
Lambda in Ridge Regression')
```

```r
ridge_coef = predict(ridge_mod_full_data, type = "coefficients", s = ridge_best_lambd
a)[-1,]
ridge_coef
```

```r
ridge_pred = predict(ridge_mod, cc_tst_x, s = ridge_best_lambda, type = 'class')
#calculate percent correctly classified
ridge_acc = length(which(ridge_pred == cc_tst$default.payment.next.month)) / nrow(cc_t
st)
paste0('Ridge regression accurately classified ', paste0(ridge_acc*100,'% of the dat
a.'))
```

Lasso Regression

```r
set.seed(490)
lasso_mod = cv.glmnet(cc_trn_x, cc_trn$default.payment.next.month, alpha = 1, family =
'binomial')
lasso_best_lambda = lasso_mod$lambda.min
lasso_mod
```

```r
paste0('The lambda within one standard error of the minimum lambda is ', lasso_best_la
mbda)
```

```r
plot(lasso_mod, sub = 'Training MSE as a Function of Lambda in Lasso Regression')
```

```r
lasso_mod_full_data = glmnet(cc_df_x, cc_df$default.payment.next.month, alpha = 1, fam
ily = 'binomial')
plot(lasso_mod_full_data, xvar = "lambda", sub = 'Coefficients for Different Values of
Lambda in Lasso Regression')
```

```r
lasso_coef = predict(lasso_mod_full_data, type = "coefficients", s = lasso_best_lambd
a)
lasso_coef
```

```
lasso_pred = predict(lasso_mod, cc_tst_x, s = lasso_best_lambda, type = 'class')
#calculate percent correctly classified
lasso_acc =length(which(lasso_pred == cc_tst$default.payment.next.month)) / nrow(cc_ts
t)
paste0('Lasso regression accurately classified ', paste0(lasso_acc*100,'% of the dat
a.'))
```

Classification Tree

```
tree_mod = tree(default.payment.next.month ~ ., data = cc_trn)
summary(tree_mod)
```

```
tree_mod_cv = cv.tree(tree_mod, FUN = prune.misclass)
tree_mod_cv
```

```
plot(tree_mod)
text(tree_mod, all = TRUE, cex = .5, pretty = 0)
```

```
tree_pred = predict(tree_mod, cc_tst, type = 'class')
#calculate percent correctly classified
tree_acc =length(which(tree_pred == cc_tst$default.payment.next.month)) / nrow(cc_tst)
paste0('Classification tree accurately classified ', paste0(tree_acc*100,'% of the dat
a.'))
```

```
plot(tree_mod_cv$size, tree_mod_cv$dev, type = "b")
```

```
tree_pruned = prune.misclass(tree_mod, best = 2)
plot(tree_pruned)
text(tree_pruned,cex = .7, pretty = 0)
```

```
tree_pruned
```

```
prune_pred = predict(tree_pruned, cc_tst, type = 'class')
#calculate percent correctly classified
pruned_acc =length(which(prune_pred == cc_tst$default.payment.next.month)) / nrow(cc_t
st)
paste0('Classification tree accurately classified ', paste0(pruned_acc*100,'% of the d
ata.'))
```

Bootstrapping

```
set.seed(490)
boot_fn = function(formula, data, indices){
  d = data[indices,]
  fit = tree(formula, data = d)
  fit_pred = predict(fit, cc_tst, type = 'class')
  fit_acc = mean(fit_pred == cc_tst$default.payment.next.month)
  return (fit_acc)
  }

result = boot(data = cc_trn, statistic = boot_fn, R = 100, formula = default.payment.n
ext.month ~.)
result
```

Bagging

```
set.seed(490)
bag_mod = randomForest(default.payment.next.month ~ . - ID, data = cc_trn, mtry = ncol
(cc_trn)-1, importance=TRUE)
bag_mod
```

```
plot(bag_mod)
```

```
bag_pred = predict(bag_mod, cc_tst, type = 'class')
bag_cm = confusionMatrix(bag_pred, cc_tst$default.payment.next.month)
bag_cm
bag_acc = mean(bag_pred == cc_tst$default.payment.next.month)
paste0('Bagging Classification tree accurately classified ', paste0(bag_acc*100,'% of
the data.'))
```

```
importance(bag_mod)
```

Random Forest

```
set.seed(490)
oob.err = double(24)
test.err = double(24)

for (mtry in 1:24){
  rf = randomForest(default.payment.next.month ~ . - ID, data = cc_trn, mtry = mtry, n
tree = 100)
  oob.err[mtry] = rf$err.rate[100]

  pred = predict(rf, cc_tst)
  test.err[mtry] = 1 - mean(pred == cc_tst$default.payment.next.month)
}

matplot(1:mtry , oob.err, pch=20 , col = "red",type = "b",ylab="Error Rate",xlab="Numb
er of Predictors Considered at each Split")
legend("topright",legend = "Out of Bag Error",pch=19, col = "red")
```

```
rf_tib = tibble(
  mtry = 1:24,
  OOB.Error = oob.err
)

kable(rf_tib) %>%
  kable_styling("striped", full_width = F)

paste0('The mtry with the lowest out of bag error rate is ', which.min(oob.err))
```

```
rf_mod = randomForest(default.payment.next.month ~ . - ID, data = cc_trn, mtry = 5, im
portance = TRUE, do.trace = 100)
rf_mod
```

```
rf_pred = predict(rf_mod, cc_tst, type = 'class')
rf_cm = confusionMatrix(rf_pred, cc_tst$default.payment.next.month)
rf_cm
rf_acc = mean(rf_pred == cc_tst$default.payment.next.month)
paste0('Random forest model accurately classified ', paste0(rf_acc*100,'% of the dat
a.'))
```

```
importance(rf_mod)
```

```
matplot(1:mtry , cbind(oob.err,test.err), pch=20 , col=c("red","blue"),type="b",ylab
="Error Rate",xlab="Number of Predictors Considered at each Split")
legend("topright",legend=c("Out of Bag Error","Test Error"),pch=19, col=c("red","blu
e"))
```

Boosting

```
boost_mod = gbm(default.payment.next.month ~ . - ID, data = cc_df, distribution = "mul
tinomial", n.trees = 100, interaction.depth = 4)
boost_mod
```

```
boost_pred = predict.gbm(boost_mod, cc_tst, n.tree = 100, type = 'response')
labels = colnames(boost_pred)[apply(boost_pred, 1, which.max)]
boost_cm = confusionMatrix(cc_tst$default.payment.next.month, as.factor(labels), posit
ive = 'yes')
boost_cm
paste0('Boosting Classification tree accurately classified ', paste0(boost_cm$overall
[1]*100,'% of the data.'))
```

```
summary(boost_mod)
```

XGboost

```
set.seed(490)
d_train = xgb.DMatrix(data = cc_trn_x, label = as.numeric(cc_trn$default.payment.next.
month) - 1)
xgb_mod = xgboost(data = d_train, nrounds = 100,print_every_n = 10,objective = "binar
y:logistic")
```

```
xgb_pred = predict(xgb_mod, cc_tst_x)
xgb_pred_class = as.numeric(xgb_pred > 0.50)
xgb_cm = confusionMatrix(as.factor(xgb_pred_class), as.factor(as.numeric(cc_tst$defaul
t.payment.next.month) - 1), positive = '0')
xgb_cm
paste0('XGBoost model accurately classified ', paste0(xgb_cm$overall[1]*100,'% of the
data.'))
```

```
importance = xgb.importance(colnames(cc_trn_x), model = xgb_mod)[-2,]
importance

xgb.plot.importance(importance, rel_to_first=TRUE, xlab="Relative Importance")
```

Tuned Boosting

```
set.seed(490)
gbm_tune = gbm(
    default.payment.next.month ~ . - ID,
    distribution = 'multinomial',
    data = cc_trn,
    n.trees = 80,
    interaction.depth = 4,
    shrinkage = .05,
    verbose = FALSE
  )
  tune_pred = predict.gbm(gbm_tune, cc_tst, n.trees = 80, type = 'response')
  labels = colnames(tune_pred)[apply(tune_pred, 1, which.max)]
  tune_cm = confusionMatrix(cc_tst$default.payment.next.month, as.factor(labels), posi
tive = 'yes')
  tune_cm
  paste0('Tuned boosting model accurately classified ', paste0(tune_cm$overall[1]*10
0,'% of the data.'))
```

```
summary(gbm_tune)
```

Neural Network

```
library(tensorflow)
library(keras)
set.seed(490)
cc_copy = read.csv("UCI_Credit_Card.csv")
default = subset(cc_copy, default.payment.next.month == '1')
default_idx = sample(nrow(default), size = .7 * nrow(default))
nondefault = subset(cc_copy, default.payment.next.month == '0')
nondefault_idx = sample(nrow(nondefault), size = .7 * nrow(nondefault))
X_train = default[default_idx,]
X_train = rbind(X_train, nondefault[nondefault_idx,])
X_test = default[-default_idx,]
X_test = rbind(X_test, nondefault[-nondefault_idx,])
y_train = X_train$default.payment.next.month
y_test = X_test$default.payment.next.month
#X_train = X_train[, -which(names(X_train) %in% c('ID', 'default.payment.next.mont
h'))]
#X_test = X_test[, -which(names(X_test) %in% c('ID', 'default.payment.next.month'))]

train_labels = to_categorical(y_train,2)
train_data = as.matrix(X_train)
test_labels = to_categorical(y_test,2)
test_data = as.matrix(X_test)
```

```
#neural network model
set.seed(490)
model = keras_model_sequential() %>%
  layer_dense(units = 64, activation = 'softmax',
              input_shape = dim(train_data)[2]) %>%
  layer_dense(units = 64, activation = 'softmax') %>%
  layer_dense(units = 2, activation = 'softmax')

model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
summary(model)
```

```
set.seed(490)
early_stop = callback_early_stopping(monitor = 'val_loss', patience = 20)
epochs = 30
history_class = model %>% fit(
  train_data,
  train_labels,
  epochs = epochs,
  validation_split = .2,
  callbacks = list(early_stop)
)
```

```
plot(history_class)
```

```
set.seed(490)
test_predictions = model %>% predict(test_data)
test_class = model %>% predict_classes(test_data)
table(test_labels[,2], test_class)
```

Results

```
final_results = tibble(
  Model = c('Logistic Regression Model',
            'Probit Regression Model',
            "K's Nearest Neighbor",
            'Ridge Regression',
            'Lasso Regression',
            'Classification Tree',
            'Bootstrapped Tree',
            'Bagging Model',
            'Random Forest',
            'Boosting',
            'XGBoost',
            'Boosting (Tuned)',
            'Neural Network'
            ),
  Best_Tune = c('NA',
                'NA',
                paste0('k = ',knn_mod$bestTune[,1]),
                paste0('lambda = ', ridge_best_lambda),
                paste0('lambda = ', lasso_best_lambda),
                paste0('Termind Nodes = ', 2),
                paste0('NA'),
                paste0('mtry = ', ncol(cc_trn) - 1),
                paste0('mtry = 5'),
                paste0('number of trees = 100, and interaction depth = 4'),
                paste0('Number of rounds = 100'),
                paste0('shrinkage = .05, number of trees = 80, interaction depth =
4'),
                paste0('number of epochs = 21')
                ),
  Accuracy = c(lr_mod1$results[,2],
               probit_mod$results[,2],
               knn_mod$results[knn_mod$results[,1] == knn_mod$bestTune[,1], 2],
               ridge_acc,
               lasso_acc,
               pruned_acc,
               result$t0,
               bag_acc,
               rf_acc,
               boost_cm$overall[1],
               xgb_cm$overall[1],
               tune_cm$overall[1],
               0.7788023
               )
)

kable(final_results) %>%
  kable_styling("striped", full_width = F)
```

1. Credit Default Mining Using Combined Machine Learning and Heuristic Approach (https://arxiv.org/ftp/arxiv/papers/1807/1807.01176.pdf)↩
2. Default of Credit Card Clients Dataset (https://www.kaggle.com/uciml/default-of-credit-card-clients-dataset)↩