Write-up Math 154 Kaggle Competition

Aser Atawya, Bryan McNair, David Wong, Xiaoxing Yu

2022-11-22

Introduction

In this write up we are analyzing the likelihood someone experiences financial distress over the next two years given certain parameters. With the data, there are a total of 11 parameters which include:

1. SeriousDlqin2yrs: Person experienced 90 days past due delinquency or worse after 2 years
2. RevolvingUtilizationOfUnsecuredLines: Total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits
3. Age: Age of borrower in years
4. NumberOfTime30-59DaysPastDueNotWorse: Number of times borrower has been 30-59 days past due but no worse in the last 2 years.
5. DebtRatio: Monthly debt payments, alimony,living costs divided by monthly gross income
6. MonthlyIncome: Monthly income
7. NumberOfOpenCreditLinesAndLoans: Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g. credit cards)
8. NumberOfTimes90DaysLate: Number of times borrower has been 90 days or more past due.
9. NumberRealEstateLoansOrLines: Number of mortgage and real estate loans including home equity lines of credit
10. NumberOfTime60-89DaysPastDueNotWorse: Number of times borrower has been 60-89 days past due but no worse in the last 2 years.
11. NumberOfDependents: Number of dependents in family excluding themselves (spouse, children etc.)

Using these different parameters, our goal is to create the best learner we can using the evaluation metric of area under the curve (AUC) of the receiver operating characteristic (ROC) curve.

A couple of early issues arise even from a quick look of the data set. We see that there are missing/NA values in the MonthlyIncome and number of NumberOfDependents columns. We also want to make sure that we remove any outliers that might heavily screw the data. What we decide to do is use Outlier Analysis to determine which values to remove from our data set and the imputation method SMOTE to fill in our missing values.

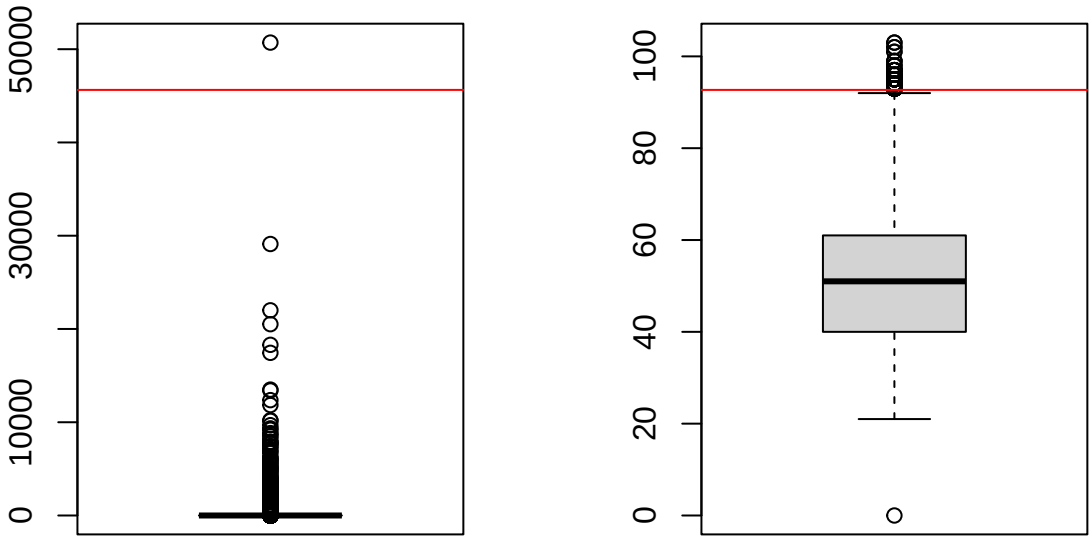
Outlier Analysis

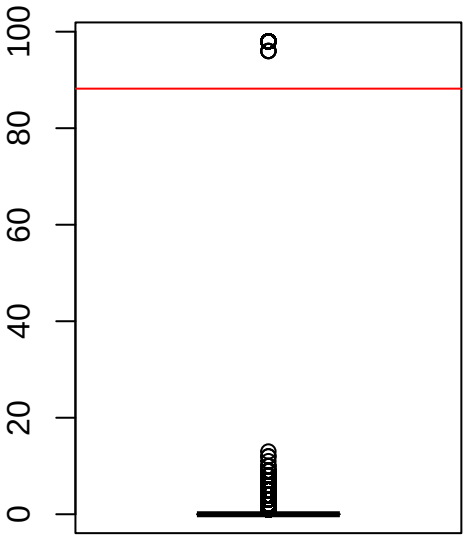
A boxplot of the data immediately shows that the data has many outliers. Additionally, it shows that the naïve approach of throwing out any data more than 1.5 IQRs from the first and third quartiles will not work for many of the variables, which consist of a large number of zero observations and a small number of positive observations.

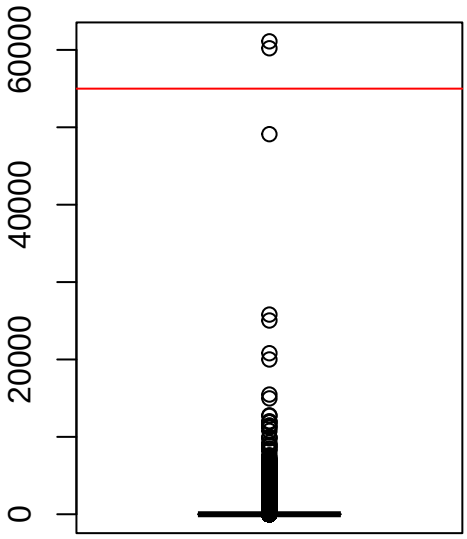
```r
# import data
require(readr)
```

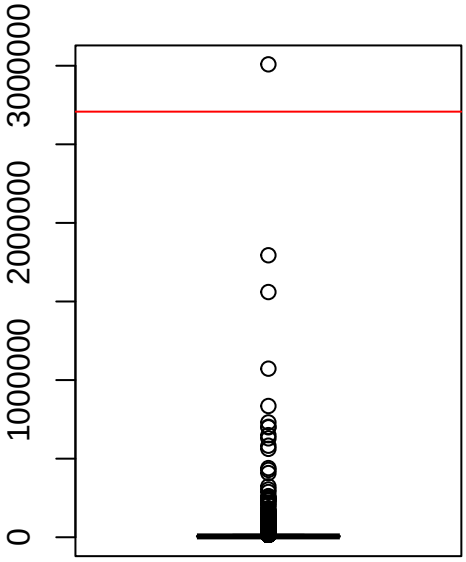
```
## Loading required package: readr
```

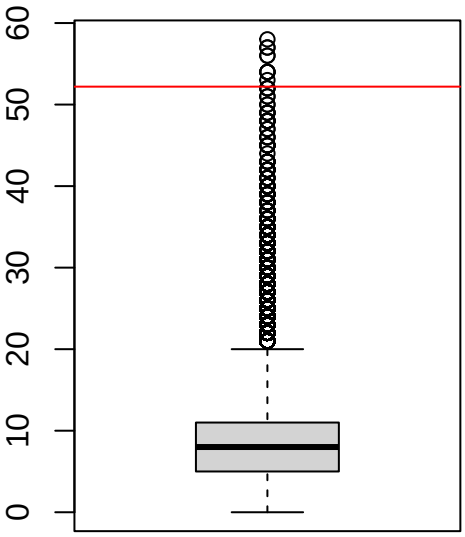
```r
train <- read.csv("cs-training.csv")
colnames(train) <- c("", "SeriousDlqin2yrs", "RevolvingUtilizationOfUnsecuredLines", "age", "NumberOfTi
train <- train[-1]

par(mfrow = c(1,2))
X <- train[-1] # remove response variable
X <- na.exclude(X)
lined.boxplot <- function(list, labels, color){ # couldn't figure out how to label the boxplots in the
  if(missing(color)){
    color <- 'red'
  }
  boxplot(list, names=labels)
  abline(h = 0.1*min(list) + 0.9*max(list), col=color)
}
invisible(lapply(X, lined.boxplot))
```

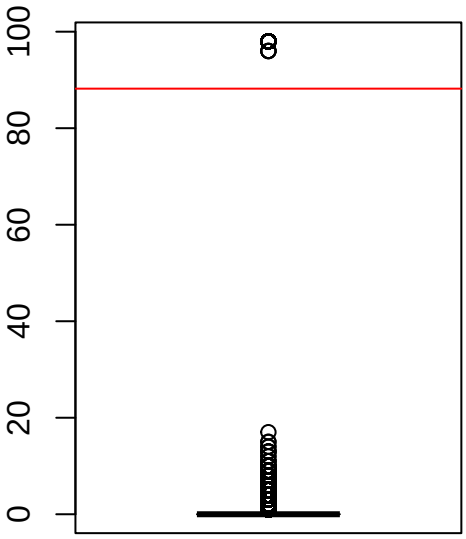


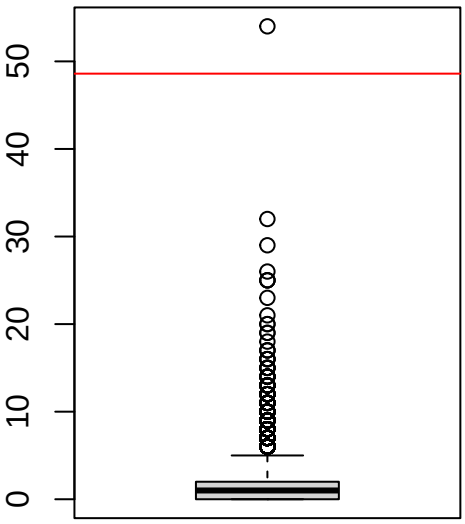

0
20
40
60
80
100

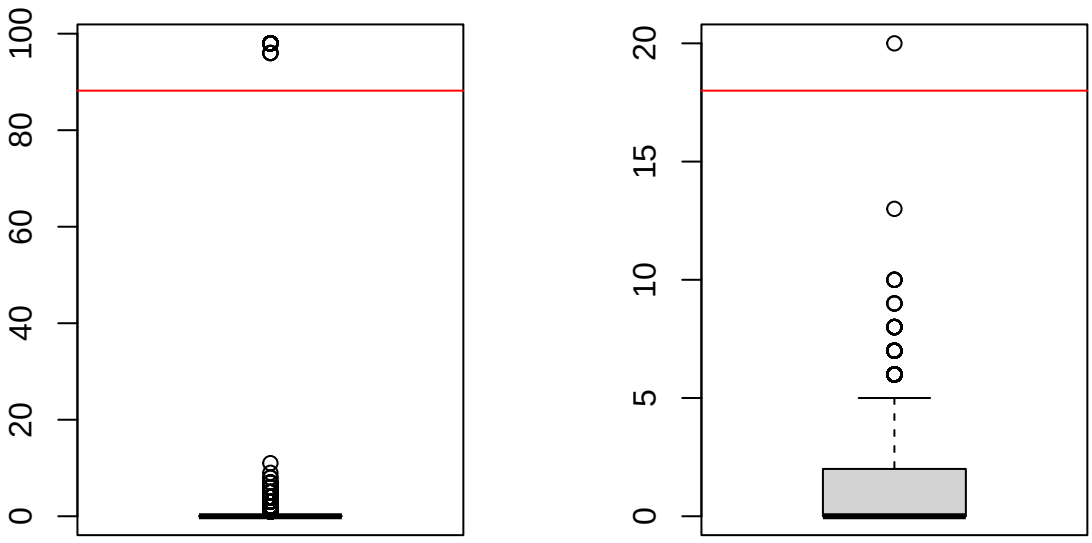

0
20000
40000
60000


0
1000000
2000000
3000000


0
10
20
30
40
50
60

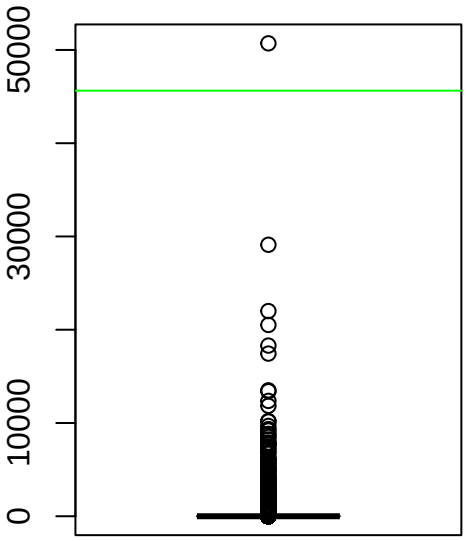

0
20
40
60
80
100

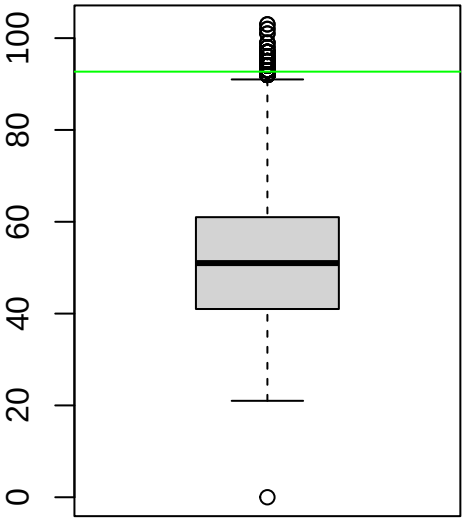

0
10
20
30
40
50

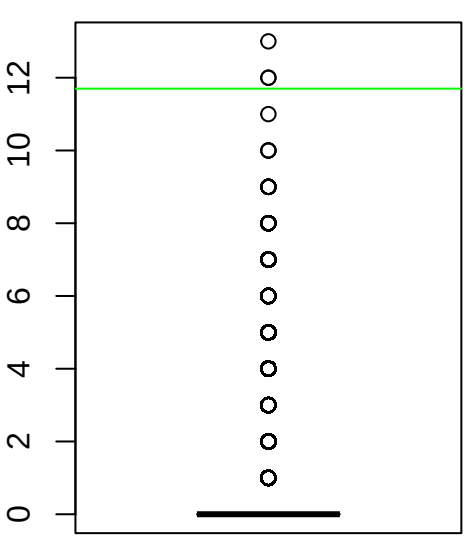


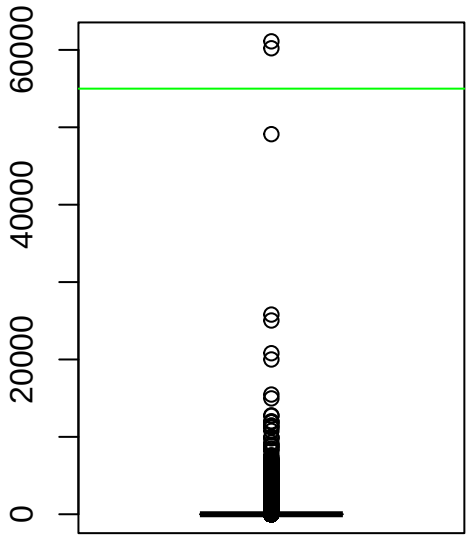
Instead, based on visual examination of the boxplots, we opted to treat the data in the top 10% of the range as outliers for the variables "NumberOfTime30-59DaysPastDueNotWorse", "NumberOfTimes90DaysLate", "NumberRealEstateLoansOrLines", "NumberOfTime60-89DaysPastDueNotWorse", and "NumberOfDependents". The resulting data appears somewhat better.

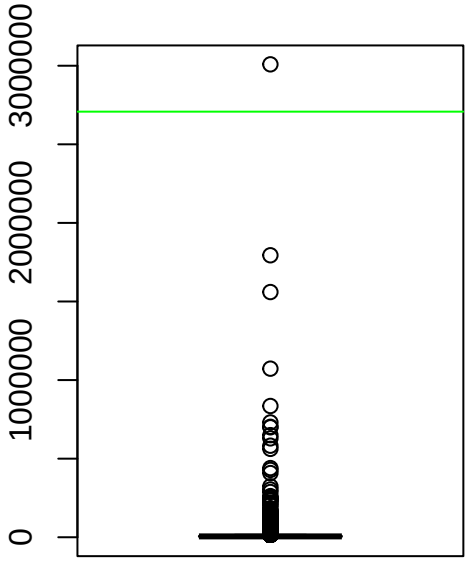
```
for(i in c(4, 8, 9, 10, 11)){
  train[which(train[i] > 0.9*max(train[i]) + 0.1*min(train[i])),i] <- NA
}

X <- train[-1]
X <- na.exclude(X)
par(mfrow = c(1,2))
invisible(lapply(X, lined.boxplot, color='green'))
```

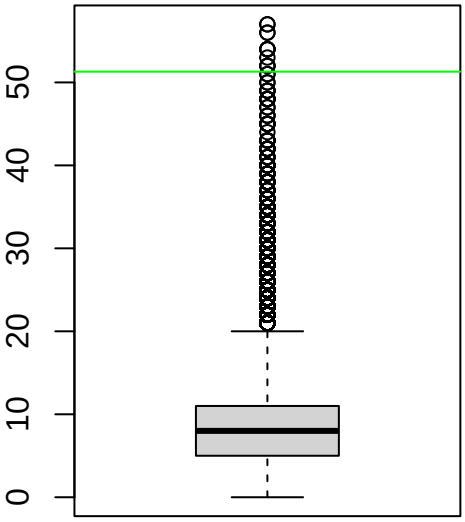

0
10000
30000
50000

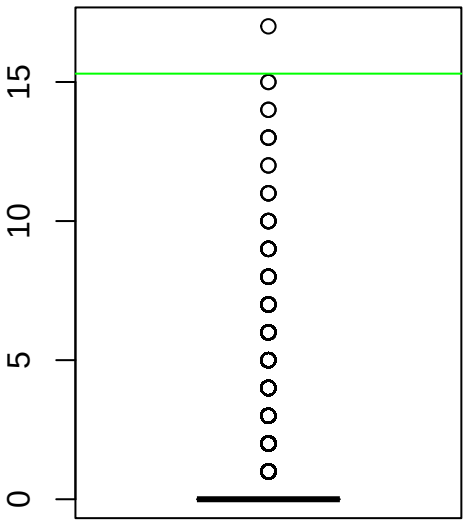

0
20
40
60
80
100

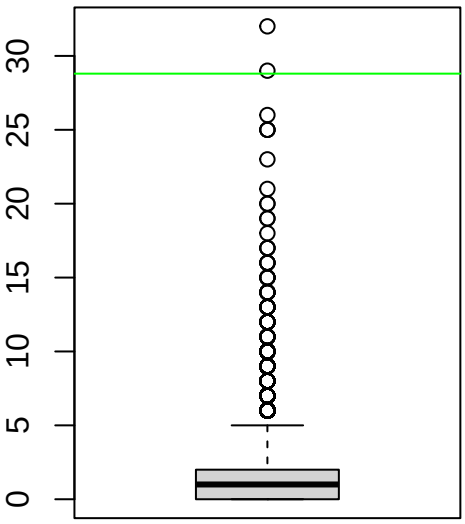

0
2
4
6
8
10
12

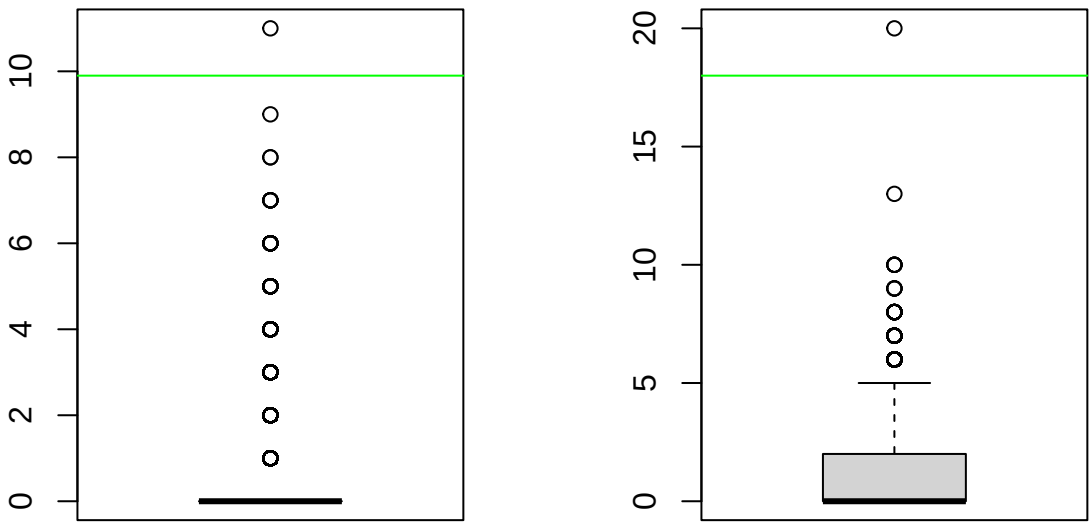

0
20000
40000
60000


0
1000000
2000000
3000000


0
10
20
30
40
50


0
5
10
15


0
5
10
15
20
25
30



Imputation

```
remotes::install_github("cran/DMwR")
```

```
## Skipping install of 'DMwR' from a github remote, the SHA1 (6fd4f0cd) has not changed since last insta
##   Use `force = TRUE` to force installation
```

```
library(DMwR)
```

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

```
## Loading required package: grid
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
original_data <- read.csv("cs-training.csv")
table(original_data$SeriousDlqin2yrs)
```

```
##
##      0      1
## 139974  10026
```

```
data_imputate <- train
data_imputate$SeriousDlqin2yrs <- as.factor(data_imputate$SeriousDlqin2yrs)
data_imputate <- SMOTE(SeriousDlqin2yrs ~., data_imputate, perc.over = 1000, perc.under = 100)
```

```
table(data_imputate$SeriousDlqin2yrs)
```

```
##
##      0      1
## 100260 110286
```

```
head(data_imputate)
```

```
##        SeriousDlqin2yrs RevolvingUtilizationOfUnsecuredLines age
## 133791                0                           0.003539397  54
## 44888                 0                           0.749375019  60
## 133823                0                           0.041272487  39
## 36732                 0                           0.003229946  67
## 147474                0                           0.032217293  33
## 22841                 0                           0.225940198  60
##        NumberOfTime30-59DaysPastDueNotWorse   DebtRatio MonthlyIncome
## 133791                                    0   0.2811283          9500
## 44888                                     2   0.7241862          5713
## 133823                                    0   0.1650378          4501
## 36732                                     0 693.0000000            NA
## 147474                                    0   0.1931228          2500
## 22841                                     0   0.2098017          7916
##        NumberOfOpenCreditLinesAndLoans NumberOfTimes90DaysLate
## 133791                              10                       0
## 44888                               28                       0
## 133823                               6                       0
## 36732                                7                       0
## 147474                              10                       0
## 22841                                6                       1
##        NumberRealEstateLoansOrLines NumberOfTime60-89DaysPastDueNotWorse
## 133791                            2                                    0
## 44888                             4                                    2
## 133823                            0                                    0
## 36732                             1                                    0
## 147474                            1                                    0
## 22841                             1                                    0
##        NumberOfDependents
## 133791                  1
## 44888                   0
## 133823                  3
## 36732                   0
## 147474                  0
## 22841                   0
```

```
data_imputate[is.na(data_imputate)] = 0
sample <- sample(c(TRUE, FALSE), nrow(data_imputate), replace=TRUE, prob=c(0.7,0.3))
data_imputate_training <- data_imputate[sample,] #70 percent of the data is training
data_imputate_test <- data_imputate[!sample,] # 30 percent of the data is test
```

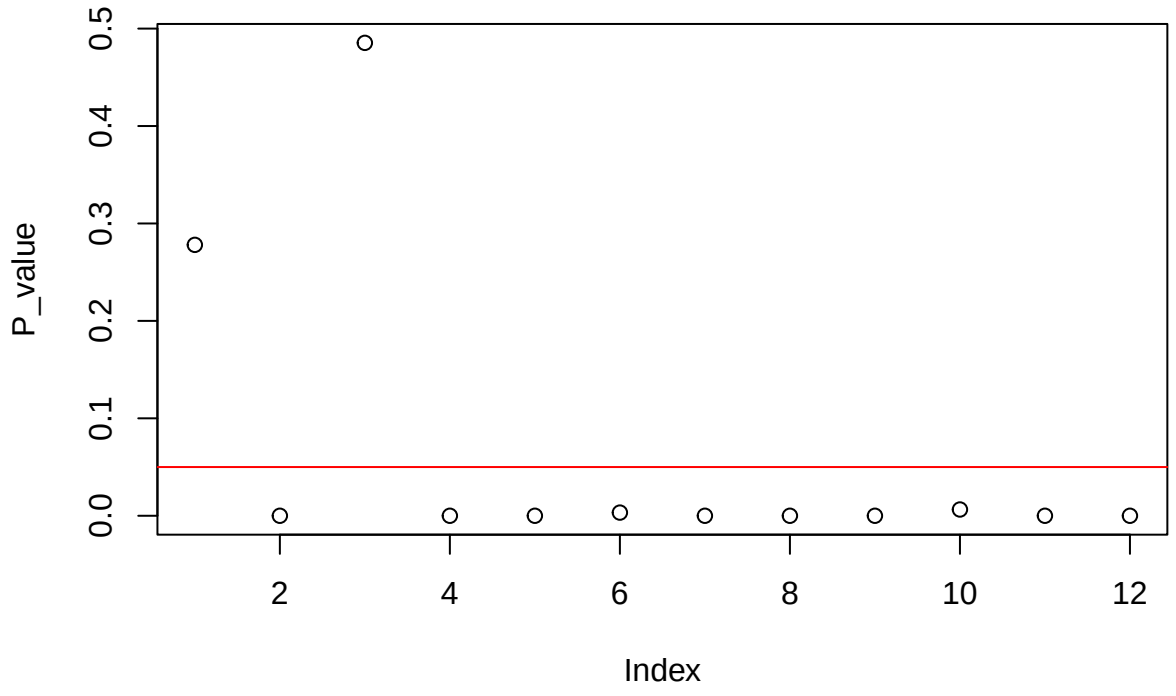
The imputation with SMOTE is quite essential to work with an unbalanced dataset like this one. We can see the distribution of the SeriousDlqin2yrs in the original data to have 139,974 0's and only 10,026 1's. This unbalanced dataset will cause many problems while working with the K-NN model as we will show. After imputation, we can see the distribution of SeriousDlqin2yrs changed signifcantly to have 100260 0's and 110286 1's.

ANOVA test

```
data1 <- read.csv("cs-training.csv")
```

ANOVA is a statistical test used to determine whether a particular classification of the data is useful in understanding the variation of an outcome i.e. whether the feature you are using in classifying data is efficient. Let's assume we divide the data into two classes based on whether someone will face serious delinquency in two years. The total variation in the dependent variable can be decomposed into the variation between classes and the variation within classes. When the within-class variation is small relative to the between-class variation, the classification is efficient and helps in understanding the difference between the two groups; members of each class behave similarly to one another, but people from different clusters behave distinctively i.e. people who will face serious delinquency in two years have distinct features compared to those who won't face serious delinquency in two years. The statistical test has a null hypothesis and an alternative hypothesis: Null Hypothesis (H0) : No difference in means i.e. no relation between variables of interest Alternative Hypothesis (H1): There exist a relation between variables of interest Reject (H0) if p-value < 0.05

```
P_value <- c() # will save the p-value of each test
for(i in 1:12){ # finds the p-value of the 12 columns
one.way <- aov( data1[,i] ~ as.factor(SeriousDlqin2yrs), data = data1)
summary(one.way) #the result of the test
P_value [i] <- summary(one.way)[[1]][["Pr(>F)"]][1]  # save the p-value of each column in the p-value v
}
plot(P_value )
abline(h = 0.05, col = "red") #the cutoff p-value is 0.05
```



We didn't reject the null hypothesis (H0) only for columns X and RevolvingUtilizationOfUnsecuredLines. Columns X is just a counter of the rows, and so it has nothing to do with whether someone will face a serious

delinquency in two years. The RevolvingUtilizationOfUnsecuredLines is the total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits and the values are in percentage. We expected this to be a relevant feature, but we have to reject it based on the results of the ANOVA test.

Classification

##K_NN Model # choosing K

```
install.packages(c("ggplot2", "ggpubr", "tidyverse", "broom", "AICcmodavg", "class", "caret"))
```

```
library(ggplot2)
library(ggpubr)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v tibble  3.1.8     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.1
## v purrr   0.3.4     v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(broom)
```

```
##
## Attaching package: 'broom'
##
## The following object is masked from 'package:DMwR':
##
##     bootstrap
```

```
library(AICcmodavg)
library(class)
library(caret)
```

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

```
data_train <- read.csv("cs-training.csv")
data_train[is.na(data_train)] = 0
head(data_train)
```

```
##   X SeriousDlqin2yrs RevolvingUtilizationOfUnsecuredLines age
## 1 1                1                              0.7661266  45
## 2 2                0                              0.9571510  40
## 3 3                0                              0.6581801  38
## 4 4                0                              0.2338098  30
## 5 5                0                              0.9072394  49
## 6 6                0                              0.2131787  74
##   NumberOfTime30.59DaysPastDueNotWorse  DebtRatio MonthlyIncome
## 1                                    2 0.80298213          9120
```

```
## 2                                    0 0.12187620          2600
## 3                                    1 0.08511338          3042
## 4                                    0 0.03604968          3300
## 5                                    1 0.02492570         63588
## 6                                    0 0.37560697          3500
##   NumberOfOpenCreditLinesAndLoans NumberOfTimes90DaysLate
## 1                              13                       0
## 2                               4                       0
## 3                               2                       1
## 4                               5                       0
## 5                               7                       0
## 6                               3                       0
##   NumberRealEstateLoansOrLines NumberOfTime60.89DaysPastDueNotWorse
## 1                            6                                    0
## 2                            0                                    0
## 3                            0                                    0
## 4                            0                                    0
## 5                            1                                    0
## 6                            1                                    0
##   NumberOfDependents
## 1                  2
## 2                  1
## 3                  0
## 4                  0
## 5                  0
## 6                  1
```

It's computationally expensive to test different k's for 150,000 rows, so I will just work with 15,000, and the results should be fairly similar to the models working with the original dataset.

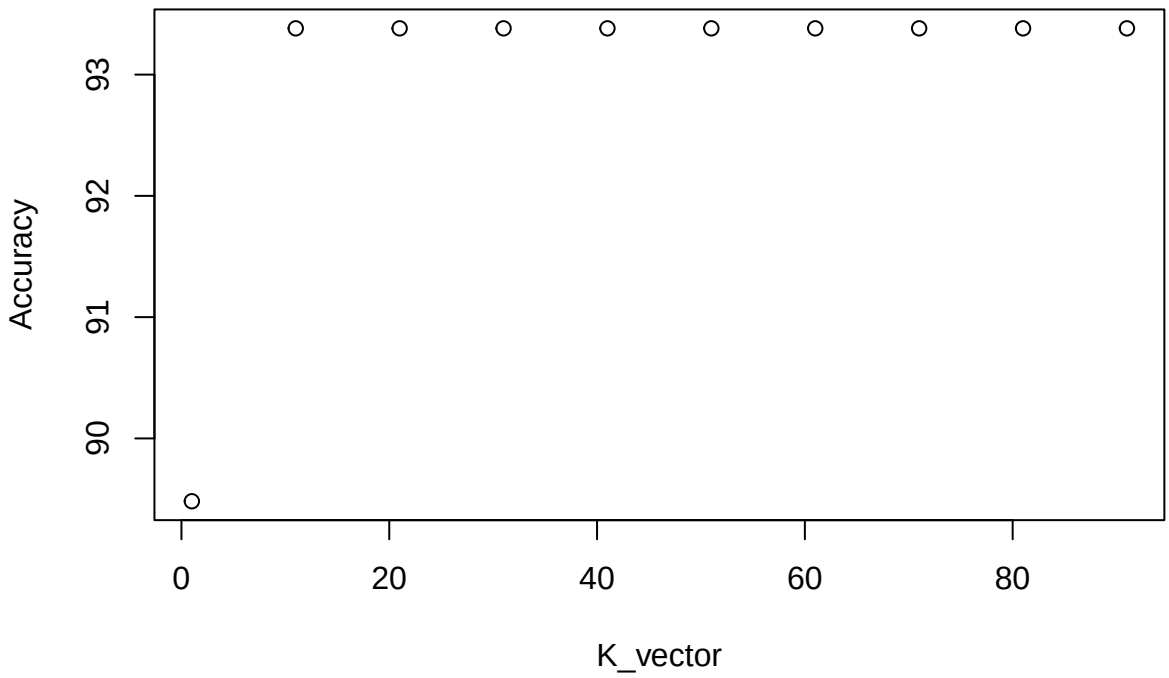
```
Accuracy <- c() # this vector will include the accuracies of different K-NN models based on k
j = 1;
K_vector <- c() # this vector will include the k's we tested
for(i in 1: 10)
{
knn <- class::knn(train= data_train[1:10000, c(2,4:12)], test = data_train[10000:15000, c(2,4:12)], cl=
modelAccuracy <- 100 * sum(data_train[10000:15000, 2]== knn)/NROW(data_train[10000:15000, 2]) # the acc
Accuracy[i] <- modelAccuracy # save the accuracy of the model into the vector
K_vector[i] <- j
j = j + 10 # We try 10 different k's starting from 1 to 91 increasing 10 each time to get a sense of wh
}
summary(Accuracy) # summarize the most important stats about the different accuracies
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   89.48   93.38   93.38   92.99   93.38   93.38
```

```
K_vector[which.max(Accuracy)] # this line will print which k produce the best accuracy
```

```
## [1] 11
```

```
plot(x = K_vector, y = Accuracy) # plot the data
```



The best accuracy is found around k = 11. Next, I will try k from 5 to 15 to make sure I pick the best k

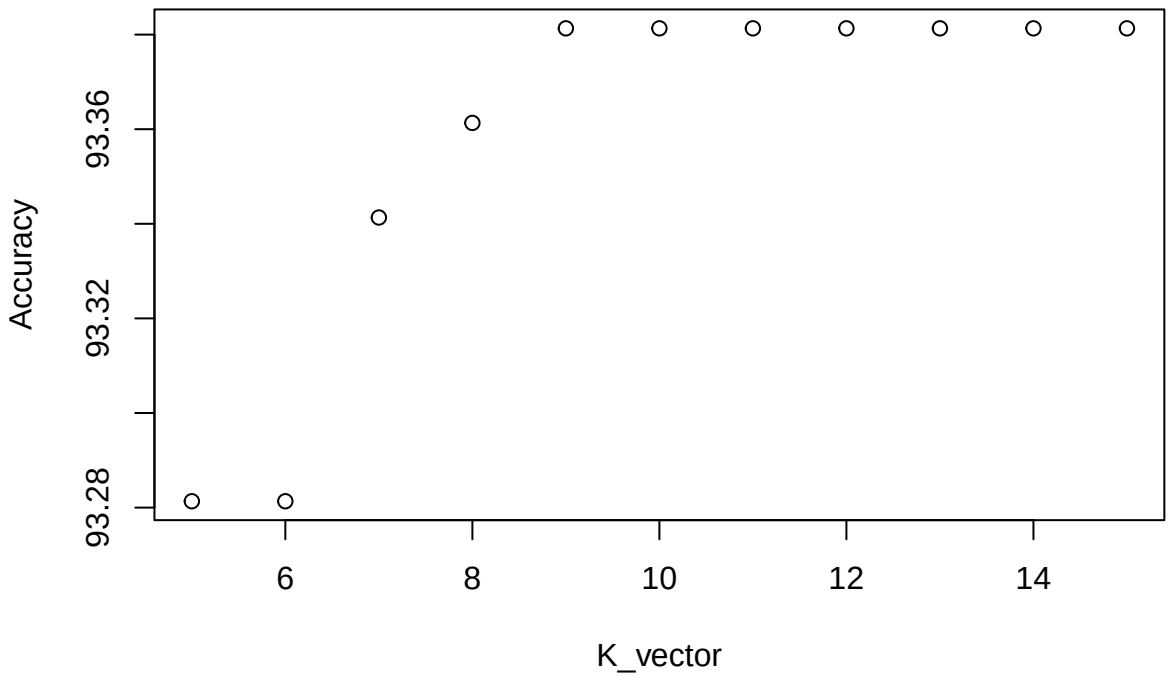
```
Accuracy <- c() # this vector will include the accuracies of different K-NN models based on k
j = 5;
K_vector <- c() # this vector will include the k's we tested
for(i in 1: 11)
{
knn <- class::knn(train= data_train[1:10000, c(2,4:12)], test = data_train[10000:15000, c(2,4:12)], cl=
modelAccuracy <- 100 * sum(data_train[10000:15000, 2]== knn)/NROW(data_train[10000:15000, 2]) # the acc
Accuracy[i] <- modelAccuracy # save the accuracy of the model into the vector
K_vector[i] <- j
j = j + 1 # We try 10 different k's starting from 5 to 15 increasing 1 each time to pick the best k
}
summary(Accuracy) # summarize the most important stats about the different accuracies
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   93.28   93.35   93.38   93.36   93.38   93.38
```

```
K_vector[which.max(Accuracy)] # this line will print which k produce the best accuracy
```

```
## [1] 9
```

```
plot(x = K_vector, y = Accuracy) # plot the data
```



9 is the best K, and at which the accuracy is 93.38% for the dataset of 15,000 rows. However, k = 9 is very low compared to a dataset of 15,000 rows, which imply that k = 9 allows the noise to influence the results and wouldn't be ideal for generalization. Generally a k = $\sqrt{n}$ should be better in terms of generalization.

```
Nine_NN <- class::knn(train= data_train[1:100000, c(2,4:12)], test = data_train[100000:150000, c(2,4:12)
Nine_NNmodelAccuracy <- 100 * sum(data_train[100000:150000, 2]== Nine_NN)/NROW(data_train[100000:150000
Nine_NNmodelAccuracy
```

```
## [1] 93.33213
```

```
Sqrt_NN <- class::knn(train= data_train[1:100000, c(2,4:12)], test = data_train[100000:150000, c(2,4:12)
Sqrt_NNmodelAccuracy <- 100 * sum(data_train[100000:150000, 2]== Sqrt_NN )/NROW(data_train[100000:150000
Sqrt_NNmodelAccuracy
```

```
## [1] 93.28013
```

To compare the accuracy of the 9-NN model with the $\sqrt{n}$-NN model, we need to run both models for the original dataset of 150,000 rows. As expected the model with k = $\sqrt{n}$ has lower accuracy: 93.28 compared to 93:33 for the model with k = 9. Back to the bias-variance trade off. We can't achieve a model with low bias and low variance. The model with k = 9 had very low bias but high variance, and the model with k = $\sqrt{n}$ has higher bias than 9-NN and lower variance than 9-NN. However, in this case, the difference in accuracy is negligible; it's only 0.05%. Hence, the $\sqrt{n}$-NN model may be better because it will avoid over-fitting. Despite being less over-fitting, the 387-NN model is computationally expensive in contrast to 9-NN model.

The results of the 387-NN model represented in a confusion matrix.

```
caret::confusionMatrix(Sqrt_NN, as.factor(data_train[100000:150000, 2]))
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction     0     1
##          0 46641  3360
##          1     0     0
##
##                Accuracy : 0.9328
##                  95% CI : (0.9306, 0.935)
##     No Information Rate : 0.9328
##     P-Value [Acc > NIR] : 0.5046
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9328
##          Neg Pred Value :    NaN
##              Prevalence : 0.9328
##          Detection Rate : 0.9328
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

The K-NN model failed to predict any 1's at k = 387, which is a serious error. This result is explained by the fact that the majority of the dataset is 0, so the voting process always yielded 0 at a considerably high k = 387.

```
caret::confusionMatrix(Nine_NN, as.factor(data_train[100000:150000, 2])) # see the confusion matrix of
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 46623  3316
##          1    18    44
##
##                Accuracy : 0.9333
##                  95% CI : (0.9311, 0.9355)
##     No Information Rate : 0.9328
##     P-Value [Acc > NIR] : 0.3251
##
##                   Kappa : 0.0233
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9996
##             Specificity : 0.0131
##          Pos Pred Value : 0.9336
##          Neg Pred Value : 0.7097
##              Prevalence : 0.9328
##          Detection Rate : 0.9324
##    Detection Prevalence : 0.9988
```

```
##       Balanced Accuracy : 0.5064
##
##        'Positive' Class : 0
##
```

At k = 9, the model predicted very few 1's, which is still not enough but is better than the 387-NN model.

Trying the K-NN models after imputating the data should let us see whether the fact that the majority of the dataset is 0 is the main reason of the error of not predicitng 1's or not.

```
remotes::install_github("cran/DMwR")
```

```
## Skipping install of 'DMwR' from a github remote, the SHA1 (6fd4f0cd) has not changed since last insta
##   Use `force = TRUE` to force installation
```

```
library(DMwR)
```

```
data_imputate <- read.csv("cs-training.csv")
data_imputate$SeriousDlqin2yrs <- as.factor(data_imputate$SeriousDlqin2yrs)
data_imputate <- SMOTE(SeriousDlqin2yrs ~., data_imputate, perc.over = 1000, perc.under = 100)
table(data_imputate$SeriousDlqin2yrs)
```

```
##
##      0      1
## 100260 110286
```

```
data_imputate[is.na(data_imputate)] = 0
sample <- sample(c(TRUE, FALSE), nrow(data_imputate), replace=TRUE, prob=c(0.7,0.3))
data_imputate_training <- data_imputate[sample,]
data_imputate_test <- data_imputate[!sample,]
head(data_imputate)
```

```
##             X SeriousDlqin2yrs RevolvingUtilizationOfUnsecuredLines age
## 52133   52133                0                            0.71713147  40
## 115633 115633                0                            0.00000000  73
## 86561   86561                0                            0.99999990  33
## 116097 116097                0                            0.00000000  65
## 43874   43874                0                            0.00000000  41
## 31939   31939                0                            0.02651408  59
##        NumberOfTime30.59DaysPastDueNotWorse    DebtRatio MonthlyIncome
## 52133                                     0 3.749639e-03          3466
## 115633                                    0 2.218371e-01          7500
## 86561                                     0 1.442268e-01          2398
## 116097                                    0 1.834000e+03             0
## 43874                                     0 9.564293e-02          3763
## 31939                                     0 2.209877e-01         18000
##        NumberOfOpenCreditLinesAndLoans NumberOfTimes90DaysLate
## 52133                                1                       1
## 115633                               3                       0
## 86561                                4                       0
## 116097                               9                       0
## 43874                                4                       0
## 31939                               16                       0
##        NumberRealEstateLoansOrLines NumberOfTime60.89DaysPastDueNotWorse
## 52133                             0                                    0
## 115633                            1                                    0
## 86561                             0                                    0
```

```
## 116097                            3                                     0
## 43874                             0                                     0
## 31939                             3                                     0
##        NumberOfDependents
## 52133                   0
## 115633                  0
## 86561                   0
## 116097                  0
## 43874                   2
## 31939                   0
```

```
knn_imputate_387 <- class::knn(train= data_imputate_training[, c(4:12)], test = data_imputate_test[, c(4
accuracy_knn_imputate_387 <- 100 * sum(data_imputate_test[, 2]== knn_imputate_387 )/NROW(data_imputate_t
```

```
accuracy_knn_imputate_387
```

```
## [1] 68.75129
```

```
# caret::confusionMatrix(as.factor(accuracy_knn_imputate_387), as.factor(data_imputate_test[, 2]))
```

```
knn_imputate_9 <- class::knn(train= data_imputate_training[, c(4:12)], test = data_imputate_test[, c(4:1
accuracy_knn_imputate_9 <- 100 * sum(data_imputate_test[, 2]== knn_imputate_9 )/NROW(data_imputate_test
```

```
accuracy_knn_imputate_9
```

```
## [1] 78.33621
```

```
# caret::confusionMatrix(as.factor(knn_imputate_9), as.factor(data_imputate_test[, 2]))
```

After imputing the data, we see the K-NN model accuracy decreased significantly because it was no longer the case that the majority are zeros. Also, we can see the bias-variance trade-off manifesting. The accuracy increased significantly from 68.7% to 78.1% when the k decreased from 387 to 9. (The sampling of the test and training data are random, so these numbers may vary when the code runs again.). When we used the 9-NN model with the test data on Kaggle, the public score was 0.5051, and the private score was 0.506. And with the 387-NN model, both the public and the private score were 0.5.

In conclusion, there isn't a single ideal k to pick. Instead, we should pick a k that suits the purpose of our model. If we are looking for accuracy and less computational expenses, we should pick k = 9; if we are looking for more generalization, we should pick k = 387. For the data without imputation, I'd pick a 9-NN model because the bank is supposed to use this data and this model to predict whether someone will face a serious delinquency in the next two years, and if all the predictions are 0, then the model is pointless. Also, imputation proved to be essential because it showed the true accuracy of the model and underscored the error resulting from having an unbalanced dataset.

SVM

#Setup The svm models took a long time to run due to the fact that the svm model is computationally intense, so the code here will not be ran.

```
install.packages(c("e1071", "caret"))
```

```
library("caret")
library("e1071")
```

```
data_train <- read.csv("cs-training.csv")
```

```
data_train <- na.omit(data_train[-1, names(data_train) != "X"])
```

First, we removed all the nas from the training data, removed the first column, which was the row indices, and removed the first row, which was the column names. Since we were running a classification svm model, we also wanted to ensure all of the other parameters in the SVM were numerical.

```
data.train.2 < - data_train
names(data.train.2) <- c("V2", "V3", "V4", "V5", "V6", "V7", "V8", "V9", "V10", "V11", "V12")
data.train.2$V2 <- as.numeric(data.train.2$V2)
data.train.2$V3 <- as.numeric(data.train.2$V3)
data.train.2$V4 <- as.numeric(data.train.2$V4)
data.train.2$V5 <- as.numeric(data.train.2$V5)
data.train.2$V6 <- as.numeric(data.train.2$V6)
data.train.2$V7 <- as.numeric(data.train.2$V7)
data.train.2$V8 <- as.numeric(data.train.2$V8)
data.train.2$V9 <- as.numeric(data.train.2$V9)
data.train.2$V10 <- as.numeric(data.train.2$V10)
data.train.2$V11 <- as.numeric(data.train.2$V11)
data.train.2$V12 <- as.numeric(data.train.2$V12)
```

#Choosing Gamma and Cost

```
svm.fit <- tune(svm, factor(V2)~., data=data.train.2[1:25000,], ranges = list(gamma = 2^(-10:10), cost =
```

Then, we wanted to determine what would be a good gamma and cost value to fit the data, so we ran the tune function with a subset of the training data and ranges of gamma equal to 2^i with i from -10 to 10, and ranges of cost equal to 2^i with i from 1 to 10, where we ended up getting a best value of cost equal to 2 and gamma equal to 1.

After that, we decided to train our model with the training data with the nas removed. This training model had a gamma value of 1 and a cost of 2.

```
svm.fit = svm(factor(V2) ~ ., data = data.train.2, cost=2, gamma=1, probabilities=TRUE)
```

Now, we focused on preparing the test data.

```
data_test <- read.csv("cs-test.csv")
data_test <- data_test[-1, names(data_test) != "X"]
names(data_test) <- c("V2", "V3", "V4", "V5", "V6", "V7", "V8", "V9", "V10", "V11", "V12")
data_test <- data_test[, names(data_test) != "V2"]

data_test$V3 <- as.numeric(data_test$V3)
data_test$V4 <- as.numeric(data_test$V4)
data_test$V5 <- as.numeric(data_test$V5)
data_test$V6 <- as.numeric(data_test$V6)
data_test$V7 <- as.numeric(data_test$V7)
data_test$V8 <- as.numeric(data_test$V8)
data_test$V9 <- as.numeric(data_test$V9)
data_test$V10 <- as.numeric(data_test$V10)
data_test$V11 <- as.numeric(data_test$V11)
data_test$V12 <- as.numeric(data_test$V12)
```

With the test data, we wanted to fill in the NA values, so we split up the data into values of NA and values without NA. With these two pieces of data, we used the values without NA to create SVMs that would predict the NA values.

```
data_test.na.12 <- data_test[rowSums(is.na(data_test)) > 0,]
data_test.na.12 <- data_test.na.12[, names(data_test.na.12) != "V7"]
data_test.na.12 <- data_test.na.12[, names(data_test.na.12) != "V12"]
```

```
data_test.no.na.12 <- data_test[!is.na(data_test$V7),]
data_test.no.na.12 <- data_test.no.na.12[, names(data_test.no.na.12) != "V7"]

svm.12 = svm(V12 ~ ., data = data_test.no.na.12, cost=2)

data.na$V12 <- predict(svm.12, newdata = data_test.na.12)
```

```
data_test.with.all.12 <- rbind(data_test.no.na.12, data_test.na.12)
data_test.with.all.12$V7 <- data_test$V7

data_test.with.all.12$V3 <- as.numeric(data_test.with.all.12$V3)
data_test.with.all.12$V4 <- as.numeric(data_test.with.all.12$V4)
data_test.with.all.12$V5 <- as.numeric(data_test.with.all.12$V5)
data_test.with.all.12$V6 <- as.numeric(data_test.with.all.12$V6)
data_test.with.all.12$V7 <- as.numeric(data_test.with.all.12$V7)
data_test.with.all.12$V8 <- as.numeric(data_test.with.all.12$V8)
data_test.with.all.12$V9 <- as.numeric(data_test.with.all.12$V9)
data_test.with.all.12$V10 <- as.numeric(data_test.with.all.12$V10)
data_test.with.all.12$V11 <- as.numeric(data_test.with.all.12$V11)
data_test.with.all.12$V12 <- as.numeric(data_test.with.all.12$V12)
```

This process was repeated to fill in the NA values for Monthly Income.

```
data_test.na.7 <- data_test.with.all.12[rowSums(is.na(data_test.with.all.12)) > 0,]
data_test.na.7 <- data_test.na.7[, names(data_test.na.7) != "V7"]

data_test.no.na.7 <- data_test.with.all.12[!is.na(data_test.with.all.12$V7),]

svm.7 = svm(V7 ~ ., data = data_test.no.na.7, cost=2)

data_test.na.7$V7 <- predict(svm.7, data_test.na.7)

finished <- rbind(data_test.no.na.7, data_test.na.7)
prob.finshed <- predict(svmfit.com, newdata=finished, probability=TRUE)
predictions <- as.data.frame(attr(prob.finshed, "probabilities")[,1])
names(predictions) <- c("prob")
write.csv(predictions, "math154KaggleSVM.csv")
```

After submitting the results of the SVM to Kaggle, the results we got were a private score of 0.62361 and a public score of 0.62704.

Results and Conclusion

As expected, directly applying a technique such as k-NN resulted in a poor prediction of credit scores, even with the help of techniques such as outlier analysis, ANOVA, and imputation. Both public and private scores were in the vicinity of 0.5, which is comparable to random guessing. The SVM returned better results, with private and public scores hovering around 0.62, but is still lacking in predictive power. It is apparent that more sophisticated approaches are required for this relatively difficult problem.