ST 563 Project

2025-04-03

Introduction:

The dataset used for this project is the Default of Credit Card Clients dataset from the UCI Machine Learning Repository. The dataset contains 30,000 observations of credit card holders, with 23 predictor variables related to their demographics, credit history, and past repayment behavior. The goal is to predict whether an individual will default on their payment in the next month or not (default.payment.next.month).

This dataset is commonly used in financial risk modeling and expected credit loss, something I'm interested in as a career. Financial institutions and wholesale banking companies rely on similar data to evaluate loan portfolios and predict the likelihood of borrower default using frameworks such as CCAR and CECL.

The goal is to build classification models to determine the probability of default. I'm going to do this by using models such as kNN, Logistic Regression, etc.

Importing the data locally. We're going to skip reading the first line in because in the raw data we have predictors X1 - X23 and Y (response). The second line actually has the real variable names.

```
library(readxl)
default_data <- read_excel("~/NCSU/ST 563/default of credit card clients.xls", skip = 1)
default_data
```

```
## # A tibble: 30,000 x 25
##       ID LIMIT_BAL   SEX EDUCATION MARRIAGE   AGE PAY_0 PAY_2 PAY_3 PAY_4 PAY_5
##    <dbl>     <dbl> <dbl>     <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1     1     20000     2         2        1    24     2     2    -1    -1    -2
##  2     2    120000     2         2        2    26    -1     2     0     0     0
##  3     3     90000     2         2        2    34     0     0     0     0     0
##  4     4     50000     2         2        1    37     0     0     0     0     0
##  5     5     50000     1         2        1    57    -1     0    -1     0     0
##  6     6     50000     1         1        2    37     0     0     0     0     0
##  7     7    500000     1         1        2    29     0     0     0     0     0
##  8     8    100000     2         2        2    23     0    -1    -1     0     0
##  9     9    140000     2         3        1    28     0     0     2     0     0
## 10    10     20000     1         3        2    35    -2    -2    -2    -2    -1
## # i 29,990 more rows
## # i 14 more variables: PAY_6 <dbl>, BILL_AMT1 <dbl>, BILL_AMT2 <dbl>,
## #   BILL_AMT3 <dbl>, BILL_AMT4 <dbl>, BILL_AMT5 <dbl>, BILL_AMT6 <dbl>,
## #   PAY_AMT1 <dbl>, PAY_AMT2 <dbl>, PAY_AMT3 <dbl>, PAY_AMT4 <dbl>,
## #   PAY_AMT5 <dbl>, PAY_AMT6 <dbl>, `default payment next month` <dbl>
```

Data Cleaning

Checking the variable type of each of the variables and whether or not there are any missing values. All the variables are numeric which isn't the case so we need to convert sex, education, marriage, PAY_X, and default payment next month to factor variables. The rest of the variables are numeric. There aren't any missing values since 0s in binary variables are significant.

PAY_0 - PAY_6: Indicate whether the client was late in their payments for the past six months (April–September 2005). -2, -1, or 0: Paid on time or in advance; 1, 2, 3,. . . : Late by that number of months

BILL_AMT and PAY_AMT: 0 values may be valid because a client may have no bill or made no payment. If a large percentage of rows contain all 0s, then we can handle the values differently.

I'm also going to check the summary of some of these variables to make sure there aren't any extreme values or outliers. Even though there's not missing data, it may be useful to check rows that all have 0s for BILL_AMT and PAY_AMT.

There were 795 rows that had 0s across BILL_AMT and PAY_AMT. However, after taking a glance at the raw data, some individuals with all 0s defaulted while others didn't. It may be useful to keep them in our dataset because it could affect the model.

I ran the table function on MARRIAGE and EDUCATION because according to the UCI Machine Learning Repository, those should have the following levels:

X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others). X4: Marital status (1 = married; 2 = single; 3 = others).

However, Education had 7 levels instead of 4 and Marital status had 4 levels instead of 3. So, I'm going to add all the levels that aren't specified from the UCI repository into the "others" category to clean them.

```
str(default_data)
```

```
## tibble [30,000 x 25] (S3: tbl_df/tbl/data.frame)
##  $ ID                        : num [1:30000] 1 2 3 4 5 6 7 8 9 10 ...
##  $ LIMIT_BAL                 : num [1:30000] 20000 120000 90000 50000 50000 50000 500000 100000 1400
##  $ SEX                       : num [1:30000] 2 2 2 2 1 1 1 2 2 1 ...
##  $ EDUCATION                 : num [1:30000] 2 2 2 2 2 1 1 2 3 3 ...
##  $ MARRIAGE                  : num [1:30000] 1 2 2 1 1 2 2 2 1 2 ...
##  $ AGE                       : num [1:30000] 24 26 34 37 57 37 29 23 28 35 ...
##  $ PAY_0                     : num [1:30000] 2 -1 0 0 -1 0 0 0 0 -2 ...
##  $ PAY_2                     : num [1:30000] 2 2 0 0 0 0 0 -1 0 -2 ...
##  $ PAY_3                     : num [1:30000] -1 0 0 0 -1 0 0 -1 2 -2 ...
##  $ PAY_4                     : num [1:30000] -1 0 0 0 0 0 0 0 0 -2 ...
##  $ PAY_5                     : num [1:30000] -2 0 0 0 0 0 0 0 0 -1 ...
##  $ PAY_6                     : num [1:30000] -2 2 0 0 0 0 0 -1 0 -1 ...
##  $ BILL_AMT1                 : num [1:30000] 3913 2682 29239 46990 8617 ...
##  $ BILL_AMT2                 : num [1:30000] 3102 1725 14027 48233 5670 ...
##  $ BILL_AMT3                 : num [1:30000] 689 2682 13559 49291 35835 ...
##  $ BILL_AMT4                 : num [1:30000] 0 3272 14331 28314 20940 ...
##  $ BILL_AMT5                 : num [1:30000] 0 3455 14948 28959 19146 ...
##  $ BILL_AMT6                 : num [1:30000] 0 3261 15549 29547 19131 ...
##  $ PAY_AMT1                  : num [1:30000] 0 0 1518 2000 2000 ...
##  $ PAY_AMT2                  : num [1:30000] 689 1000 1500 2019 36681 ...
##  $ PAY_AMT3                  : num [1:30000] 0 1000 1000 1200 10000 657 38000 0 432 0 ...
##  $ PAY_AMT4                  : num [1:30000] 0 1000 1000 1100 9000 ...
##  $ PAY_AMT5                  : num [1:30000] 0 0 1000 1069 689 ...
##  $ PAY_AMT6                  : num [1:30000] 0 2000 5000 1000 679 ...
##  $ default payment next month: num [1:30000] 1 1 0 0 0 0 0 0 0 0 ...
```

```
colSums(is.na(default_data))
```

```
##                         ID                  LIMIT_BAL
##                          0                          0
##                        SEX                  EDUCATION
##                          0                          0
##                   MARRIAGE                        AGE
##                          0                          0
##                      PAY_0                      PAY_2
```

```
##                         0                         0
##                     PAY_3                     PAY_4
##                         0                         0
##                     PAY_5                     PAY_6
##                         0                         0
##                  BILL_AMT1                 BILL_AMT2
##                         0                         0
##                  BILL_AMT3                 BILL_AMT4
##                         0                         0
##                  BILL_AMT5                 BILL_AMT6
##                         0                         0
##                   PAY_AMT1                  PAY_AMT2
##                         0                         0
##                   PAY_AMT3                  PAY_AMT4
##                         0                         0
##                   PAY_AMT5                  PAY_AMT6
##                         0                         0
## default payment next month
##                         0
```

```
summary(default_data[, c("PAY_0", "PAY_2", "BILL_AMT1", "BILL_AMT2", "PAY_AMT1", "PAY_AMT2")])
```

```
##      PAY_0              PAY_2            BILL_AMT1         BILL_AMT2
##  Min.   :-2.0000   Min.   :-2.0000   Min.   :-165580   Min.   :-69777
##  1st Qu.:-1.0000   1st Qu.:-1.0000   1st Qu.:   3559   1st Qu.:  2985
##  Median : 0.0000   Median : 0.0000   Median :  22382   Median : 21200
##  Mean   :-0.0167   Mean   :-0.1338   Mean   :  51223   Mean   : 49179
##  3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.:  67091   3rd Qu.: 64006
##  Max.   : 8.0000   Max.   : 8.0000   Max.   : 964511   Max.   :983931
##     PAY_AMT1          PAY_AMT2
##  Min.   :     0   Min.   :     0
##  1st Qu.:  1000   1st Qu.:   833
##  Median :  2100   Median :  2009
##  Mean   :  5664   Mean   :  5921
##  3rd Qu.:  5006   3rd Qu.:  5000
##  Max.   :873552   Max.   :1684259
```

```
inactive_clients <- default_data[rowSums(default_data[, grep("BILL_AMT|PAY_AMT", colnames(default_data))

nrow(inactive_clients)
```

```
## [1] 795
```

```
default_data$SEX <- as.factor(default_data$SEX)
default_data$EDUCATION <- as.factor(default_data$EDUCATION)
default_data$MARRIAGE <- as.factor(default_data$MARRIAGE)

default_data[, c("PAY_0", "PAY_2", "PAY_3", "PAY_4", "PAY_5", "PAY_6")] <-  lapply(default_data[, c("PA

default_data$`default payment next month` <- as.factor(default_data$`default payment next month`)

table(default_data$EDUCATION)
```

```
##
##     0     1     2     3     4     5     6
##    14 10585 14030  4917   123   280    51
```

```r
table(default_data$MARRIAGE)
```

```
##
##     0     1     2     3
##    54 13659 15964   323
```

```r
default_data$EDUCATION[default_data$EDUCATION %in% c(0, 5, 6)] <- 4

default_data$EDUCATION <- factor(default_data$EDUCATION,
levels = c(1, 2, 3, 4), labels = c("Graduate School", "University", "High School", "Other"))

default_data$MARRIAGE[default_data$MARRIAGE == 0] <- 3

default_data$MARRIAGE <- factor(default_data$MARRIAGE,
levels = c(1, 2, 3), labels = c("Married", "Single", "Other"))

#checking the structure one more time after data cleaning
str(default_data)
```

```
## tibble [30,000 x 25] (S3: tbl_df/tbl/data.frame)
##  $ ID                        : num [1:30000] 1 2 3 4 5 6 7 8 9 10 ...
##  $ LIMIT_BAL                 : num [1:30000] 20000 120000 90000 50000 50000 50000 500000 100000 1400
##  $ SEX                       : Factor w/ 2 levels "1","2": 2 2 2 2 1 1 1 2 2 1 ...
##  $ EDUCATION                 : Factor w/ 4 levels "Graduate School",..: 2 2 2 2 2 1 1 2 3 3 ...
##  $ MARRIAGE                  : Factor w/ 3 levels "Married","Single",..: 1 2 2 1 1 2 2 2 1 2 ...
##  $ AGE                       : num [1:30000] 24 26 34 37 57 37 29 23 28 35 ...
##  $ PAY_0                     : Factor w/ 11 levels "-2","-1","0",..: 5 2 3 3 2 3 3 3 3 1 ...
##  $ PAY_2                     : Factor w/ 11 levels "-2","-1","0",..: 5 5 3 3 3 3 3 2 3 1 ...
##  $ PAY_3                     : Factor w/ 11 levels "-2","-1","0",..: 2 3 3 3 2 3 3 2 5 1 ...
##  $ PAY_4                     : Factor w/ 11 levels "-2","-1","0",..: 2 3 3 3 3 3 3 3 3 1 ...
##  $ PAY_5                     : Factor w/ 10 levels "-2","-1","0",..: 1 3 3 3 3 3 3 3 3 2 ...
##  $ PAY_6                     : Factor w/ 10 levels "-2","-1","0",..: 1 4 3 3 3 3 3 2 3 2 ...
##  $ BILL_AMT1                 : num [1:30000] 3913 2682 29239 46990 8617 ...
##  $ BILL_AMT2                 : num [1:30000] 3102 1725 14027 48233 5670 ...
##  $ BILL_AMT3                 : num [1:30000] 689 2682 13559 49291 35835 ...
##  $ BILL_AMT4                 : num [1:30000] 0 3272 14331 28314 20940 ...
##  $ BILL_AMT5                 : num [1:30000] 0 3455 14948 28959 19146 ...
##  $ BILL_AMT6                 : num [1:30000] 0 3261 15549 29547 19131 ...
##  $ PAY_AMT1                  : num [1:30000] 0 0 1518 2000 2000 ...
##  $ PAY_AMT2                  : num [1:30000] 689 1000 1500 2019 36681 ...
##  $ PAY_AMT3                  : num [1:30000] 0 1000 1000 1200 10000 657 38000 0 432 0 ...
##  $ PAY_AMT4                  : num [1:30000] 0 1000 1000 1100 9000 ...
##  $ PAY_AMT5                  : num [1:30000] 0 0 1000 1069 689 ...
##  $ PAY_AMT6                  : num [1:30000] 0 2000 5000 1000 679 ...
##  $ default payment next month: Factor w/ 2 levels "0","1": 2 2 1 1 1 1 1 1 1 1 ...
```

Split the data into a train and test set (70/30). Rather than doing a random sample, I'm going to do createDataPartition function instead because we have an imbalance between default probability (0: 77.88%, 1: 22.12%). We have a much lower probability of default so we're going to ensure there's an even split between the train and test set.

```r
library(caret)
```

```
## Loading required package: ggplot2
```

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

```r
set.seed(51)

index <- createDataPartition(default_data$`default payment next month`,
                             p = 0.7,
                             list = FALSE)

train <- default_data[index, ]
test <- default_data[-index, ]

id_col <- "ID"
train_data <- train[, !names(train) %in% id_col]
test_data <- test[, !names(test) %in% id_col]

dim(train_data)
```

```
## [1] 21001    24
```

```r
dim(test_data)
```

```
## [1] 8999   24
```

1) kNN: This is a non-parametric model with one tuning parameter (k). The k (nearest neighbors) determines how many of the closest data points are considered when making a prediction. We can also use measures such as Euclidean distance (distance of measures). We can't use it for inference because it's a predictive model; we don't have numeric coefficients we can use to perform inference. kNN does not perform variable selection either. We would have to do some sort of standardization because if we're using the closest points to make a prediction, we need to make sure our predictors that are really large or really small are on the same scale.

I had to standardize the variables so I looked up some syntax online (Datacamp.com) and it showed me how to scale the variables so I used some of that syntax to scale the variables up to kgrid object.

```r
preProcValues <- preProcess(train_data[, -which(names(train_data) == "default payment next month")], me

train_scaled <- predict(preProcValues, train_data)
test_scaled <- predict(preProcValues, test_data)

train_scaled$`default payment next month` <- train_data$`default payment next month`
test_scaled$`default payment next month` <- test_data$`default payment next month`

kgrid <- expand.grid(k = seq(1, 21, by = 2))

tr <- trainControl(method = "cv", number = 5)

knn_fit <- train(
  `default payment next month` ~ .,
  data = train_scaled,
  method = "knn",
  tuneGrid = kgrid,
  trControl = tr
)

best_k <- knn_fit$bestTune$k
print(best_k)
```

```
## [1] 13
```

```r
print(knn_fit)
```

```
## k-Nearest Neighbors
##
## 21001 samples
##    23 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 16801, 16801, 16801, 16801, 16800
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.7328219  0.2187847
##    3  0.7797245  0.2865522
##    5  0.7964382  0.3138996
##    7  0.8065808  0.3307815
##    9  0.8106757  0.3353745
##   11  0.8125328  0.3361003
##   13  0.8140566  0.3375990
##   15  0.8132948  0.3314516
##   17  0.8125329  0.3255159
##   19  0.8135328  0.3278582
##   21  0.8124376  0.3227929
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 13.
```

```r
#used knn3 here instead of train here because the train function was taking too long to run so I found
knn_final <- knn3(
  `default payment next month` ~ .,
  data = train_scaled,
  k = 21
)

#had to convert them to class labels instead and add a threshold of 0.5; the reason for this is because
pred_probs <- predict(knn_final, test_scaled, type = "prob")
preds <- ifelse(pred_probs[,2] > 0.5, 1, 0)
preds <- as.factor(preds)

confusionMatrix(preds, test_scaled$`default payment next month`)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6713 1404
##          1  296  586
##
##                Accuracy : 0.8111
##                  95% CI : (0.8028, 0.8191)
##     No Information Rate : 0.7789
##     P-Value [Acc > NIR] : 3.479e-14
```

```
##
##                  Kappa : 0.315
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9578
##             Specificity : 0.2945
##          Pos Pred Value : 0.8270
##          Neg Pred Value : 0.6644
##              Prevalence : 0.7789
##          Detection Rate : 0.7460
##    Detection Prevalence : 0.9020
##       Balanced Accuracy : 0.6261
##
##        'Positive' Class : 0
##
```

The accuracy is 81.11%, which is pretty decent. The sensitivity is 95.78% which means the model does well to predict the no default cases since this is for class 0. However, the specificity is 29.45% which means that we're not doing a good job of predicting for default cases. We're heavily favoring the majority class over the minority class because of a data imbalance so we may have to use a different model.

2) Logistic Regression: This is a parametric model because we have a linear combination of the predictors with the logistic function. We don't have any tuning parameters but we do have Lasso and Ridge Regression to help with regularization and feature selection at times. We can also use it for inference because we get coefficients; these are used to infer on log-odds, odds, and probabilities. If we use Lasso, then we shrink some of the coefficients to 0 and do variable selection but the standard logistic regression model doesn't do variable selection. Since we're modeling probability here, we don't need to standardize the predictors.

```
default_glm <- glm(`default payment next month` ~ ., family = "binomial", data = train_data)

summary(default_glm)
```

```
##
## Call:
## glm(formula = `default payment next month` ~ ., family = "binomial",
##     data = train_data)
##
## Coefficients: (1 not defined because of singularities)
##                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -1.300e+00  1.343e-01  -9.677  < 2e-16 ***
## LIMIT_BAL            -1.701e-06  2.091e-07  -8.137 4.06e-16 ***
## SEX2                 -1.481e-01  3.880e-02  -3.817 0.000135 ***
## EDUCATIONUniversity   5.565e-02  4.479e-02   1.243 0.214031
## EDUCATIONHigh School -3.137e-02  6.019e-02  -0.521 0.602208
## EDUCATIONOther       -1.197e+00  2.389e-01  -5.010 5.43e-07 ***
## MARRIAGESingle       -1.456e-01  4.361e-02  -3.340 0.000839 ***
## MARRIAGEOther        -7.457e-02  1.706e-01  -0.437 0.662097
## AGE                   4.223e-03  2.356e-03   1.793 0.073010 .
## PAY_0-1               4.943e-01  1.307e-01   3.782 0.000156 ***
## PAY_00               -2.005e-01  1.405e-01  -1.428 0.153383
## PAY_01                8.155e-01  1.022e-01   7.979 1.47e-15 ***
## PAY_02                2.113e+00  1.278e-01  16.532  < 2e-16 ***
## PAY_03                2.237e+00  2.099e-01  10.654  < 2e-16 ***
## PAY_04                1.786e+00  3.675e-01   4.860 1.17e-06 ***
```

```
## PAY_05              1.604e+00  5.789e-01   2.772 0.005578 **
## PAY_06              4.755e-01  1.123e+00   0.423 0.671955
## PAY_07              1.047e+00  1.536e+00   0.681 0.495560
## PAY_08             -1.278e+01  5.354e+02  -0.024 0.980950
## PAY_2-1            -1.644e-01  1.379e-01  -1.192 0.233395
## PAY_20              8.454e-02  1.673e-01   0.505 0.613410
## PAY_21             -7.869e-01  6.586e-01  -1.195 0.232205
## PAY_22              9.868e-02  1.427e-01   0.692 0.489209
## PAY_23              1.188e-01  2.216e-01   0.536 0.591877
## PAY_24             -7.754e-01  3.990e-01  -1.943 0.052003 .
## PAY_25              5.429e-01  9.398e-01   0.578 0.563489
## PAY_26              5.656e-01  1.546e+00   0.366 0.714577
## PAY_27             -3.393e-01  7.572e+02   0.000 0.999642
## PAY_28             -1.384e+01  9.274e+02  -0.015 0.988093
## PAY_3-1             4.593e-03  1.307e-01   0.035 0.971959
## PAY_30              1.479e-01  1.509e-01   0.980 0.327226
## PAY_31             -1.154e+01  3.777e+02  -0.031 0.975631
## PAY_32              4.409e-01  1.536e-01   2.871 0.004091 **
## PAY_33              5.096e-01  2.700e-01   1.888 0.059078 .
## PAY_34             -1.711e-01  4.984e-01  -0.343 0.731326
## PAY_35             -5.607e-01  9.622e-01  -0.583 0.560088
## PAY_36              1.532e+01  5.354e+02   0.029 0.977180
## PAY_37             -4.801e-02  9.070e-01  -0.053 0.957786
## PAY_38              1.279e+01  5.354e+02   0.024 0.980937
## PAY_4-1            -1.543e-01  1.302e-01  -1.186 0.235737
## PAY_40             -2.534e-01  1.459e-01  -1.736 0.082554 .
## PAY_41              1.341e+01  3.777e+02   0.036 0.971677
## PAY_42              5.434e-02  1.568e-01   0.346 0.728993
## PAY_43             -2.070e-01  3.028e-01  -0.684 0.494265
## PAY_44              5.378e-01  5.319e-01   1.011 0.311943
## PAY_45             -1.629e+00  9.937e-01  -1.640 0.101067
## PAY_46              6.024e-01  7.572e+02   0.001 0.999365
## PAY_47             -2.522e+00  5.774e+02  -0.004 0.996515
## PAY_48             -2.853e+01  7.572e+02  -0.038 0.969941
## PAY_5-1            -7.892e-02  1.283e-01  -0.615 0.538365
## PAY_50             -1.879e-03  1.434e-01  -0.013 0.989549
## PAY_52              2.995e-01  1.602e-01   1.870 0.061482 .
## PAY_53              1.772e-01  2.972e-01   0.596 0.550987
## PAY_54             -2.738e-01  5.303e-01  -0.516 0.605671
## PAY_55              3.708e-01  1.016e+00   0.365 0.715196
## PAY_56              2.376e+00  5.774e+02   0.004 0.996717
## PAY_57              1.356e+01  5.354e+02   0.025 0.979790
## PAY_58              2.824e+01  1.071e+03   0.026 0.978960
## PAY_6-1            -7.127e-02  1.006e-01  -0.708 0.478800
## PAY_60             -2.816e-01  1.091e-01  -2.580 0.009871 **
## PAY_62              9.522e-02  1.259e-01   0.756 0.449588
## PAY_63              7.499e-01  2.941e-01   2.550 0.010782 *
## PAY_64             -8.455e-02  5.225e-01  -0.162 0.871452
## PAY_65             -1.116e-01  9.116e-01  -0.122 0.902577
## PAY_66              1.521e+00  1.361e+00   1.117 0.263781
## PAY_67             -1.056e+01  2.162e+02  -0.049 0.961044
## PAY_68                     NA         NA      NA       NA
## BILL_AMT1          -2.088e-06  1.307e-06  -1.598 0.110041
## BILL_AMT2           2.718e-06  1.726e-06   1.575 0.115193
```

```
## BILL_AMT3             1.711e-06  1.539e-06   1.112 0.266205
## BILL_AMT4             3.665e-07  1.610e-06   0.228 0.819909
## BILL_AMT5             1.484e-06  1.784e-06   0.832 0.405653
## BILL_AMT6            -1.741e-06  1.363e-06  -1.277 0.201697
## PAY_AMT1             -1.154e-05  2.682e-06  -4.301 1.70e-05 ***
## PAY_AMT2             -7.332e-06  2.403e-06  -3.052 0.002277 **
## PAY_AMT3             -3.708e-06  2.210e-06  -1.678 0.093400 .
## PAY_AMT4             -4.233e-06  2.291e-06  -1.847 0.064691 .
## PAY_AMT5             -2.130e-06  2.135e-06  -0.997 0.318558
## PAY_AMT6             -1.826e-06  1.532e-06  -1.192 0.233305
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 22196  on 21000  degrees of freedom
## Residual deviance: 18117  on 20923  degrees of freedom
## AIC: 18273
##
## Number of Fisher Scoring iterations: 12
```

We have way too many predictors here so we're going to use Lasso to do variable selection. We still have to exclude ID here and then create the design matrix without the ID variable. We're going to standardize our predictors and then fit the lasso model.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
train_no_id <- train[, !names(train) %in% "ID"]

X_train <- as.matrix(train[, !names(train) %in% c("ID", "default payment next month")])
y_train <- train$`default payment next month`

X_test <- as.matrix(test[, !names(test) %in% c("ID", "default payment next month")])
y_test <- test$`default payment next month`

cv_lasso <- cv.glmnet(X_train, y_train, family = "binomial", alpha = 1)

lasso_model <- glmnet(X_train, y_train, family = "binomial", alpha = 1, lambda = cv_lasso$lambda.1se)

coef(lasso_model)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept) -1.244291e+00
## LIMIT_BAL   -5.180399e-07
## SEX          .
## EDUCATION    .
## MARRIAGE     .
## AGE          .
## PAY_0        5.541741e-01
## PAY_2        6.678497e-02
## PAY_3        5.506472e-02
```

```
## PAY_4         .
## PAY_5         1.214714e-02
## PAY_6         .
## BILL_AMT1    -4.884355e-07
## BILL_AMT2     .
## BILL_AMT3     .
## BILL_AMT4     .
## BILL_AMT5     .
## BILL_AMT6     .
## PAY_AMT1     -4.514959e-07
## PAY_AMT2      .
## PAY_AMT3      .
## PAY_AMT4      .
## PAY_AMT5      .
## PAY_AMT6      .
```

```
lasso_predictions <- predict(lasso_model, newx = X_test, s = cv_lasso$lambda.1se, type = "response")

lasso_pred_class <- ifelse(lasso_predictions > 0.5, 1, 0)

confusionMatrix(factor(lasso_pred_class, levels = c(0, 1)), factor(y_test, levels = c(0, 1)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6872 1665
##          1  137  325
##
##                Accuracy : 0.7998
##                  95% CI : (0.7913, 0.808)
##     No Information Rate : 0.7789
##     P-Value [Acc > NIR] : 7.31e-07
##
##                   Kappa : 0.1983
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9805
##             Specificity : 0.1633
##          Pos Pred Value : 0.8050
##          Neg Pred Value : 0.7035
##              Prevalence : 0.7789
##          Detection Rate : 0.7636
##    Detection Prevalence : 0.9487
##       Balanced Accuracy : 0.5719
##
##        'Positive' Class : 0
##
```

Setting the threshold at 0.5 shows us that the model does incredibly well to predict the defaults (98.05%). However, the model doesn't do the best job at predicting the no default class (16.33%). If our goal is to just predict the probability of defaulting, then this would be a good model to use. The accuracy is also 80% which means 80% of the predictions are correct.

What we can also do is tune the threshold to get a better balance of predicting both classes. If our goal would

be predict both classes as best as possible, then we can lower the threshold to 0.35 to get 92% sensitivity and 41% specificity and 81% accuracy still.

3) GAMs: Natural Cubic Splines (with a logistic regression model). The spline function itself is non-parametric and we do have degrees of freedom as a tuning parameter (we can also manually pick this). We could use these for inference but we'd have to plot the results to see the effects. This model does not do variable selection and we don't need to standardize our predictors but it could be helpful in certain situations.

I was having trouble figuring out how to plot both LIMIT_BAL and AGE since I couldn't find how we plotted two predictors in one plot function from the notes. Instead, I chose to keep age constant by getting the median of AGE and plotted LIMIT_BAL instead.

```
library(ggplot2)
library(caret)
library(splines)
library(gam)
```

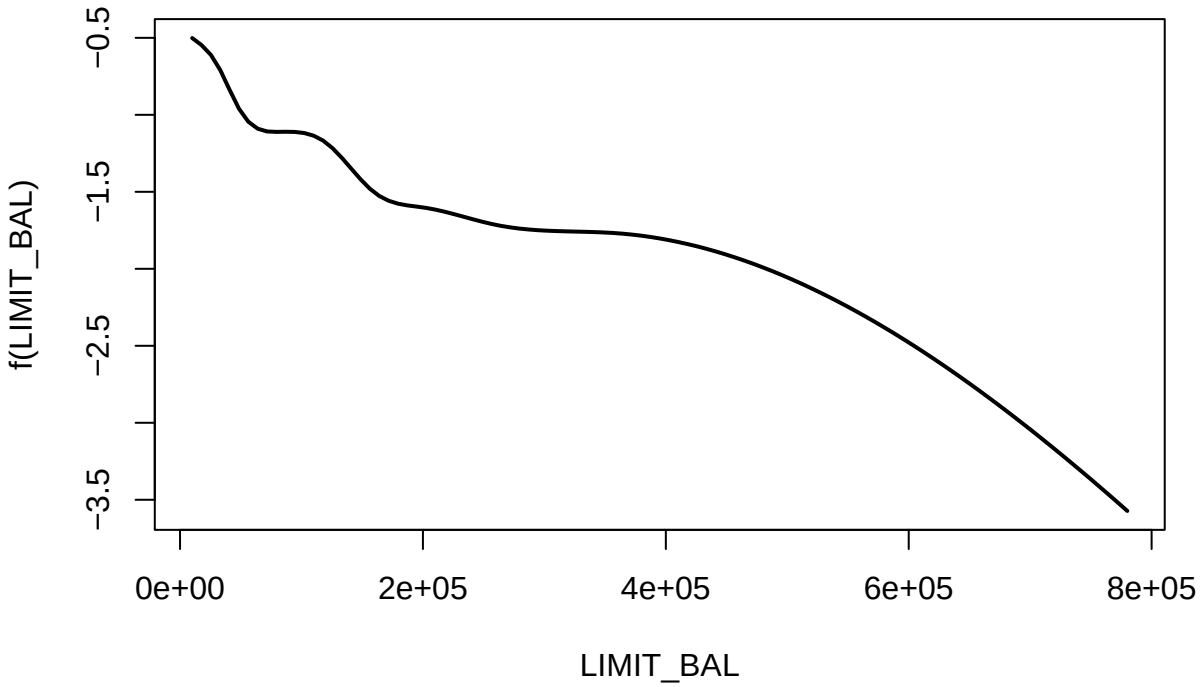
```
## Loading required package: foreach
```

```
## Loaded gam 1.22-5
```

```
gam_model <- glm(`default payment next month` ~ ns(LIMIT_BAL, df = 9) + s(AGE, df = 4), data = train_da



summary(gam_model)
```

```
##
## Call:
## glm(formula = `default payment next month` ~ ns(LIMIT_BAL, df = 9) +
##     s(AGE, df = 4), family = binomial, data = train_data)
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)             -0.836201   0.111266  -7.515 5.68e-14 ***
## ns(LIMIT_BAL, df = 9)1 -0.604816   0.105154  -5.752 8.83e-09 ***
## ns(LIMIT_BAL, df = 9)2 -0.614743   0.148892  -4.129 3.65e-05 ***
## ns(LIMIT_BAL, df = 9)3 -0.591360   0.147450  -4.011 6.06e-05 ***
## ns(LIMIT_BAL, df = 9)4 -1.086222   0.145878  -7.446 9.61e-14 ***
## ns(LIMIT_BAL, df = 9)5 -1.083694   0.133022  -8.147 3.74e-16 ***
## ns(LIMIT_BAL, df = 9)6 -1.278449   0.140047  -9.129  < 2e-16 ***
## ns(LIMIT_BAL, df = 9)7 -1.085394   0.337071  -3.220  0.00128 **
## ns(LIMIT_BAL, df = 9)8 -3.156311   0.797817  -3.956 7.62e-05 ***
## ns(LIMIT_BAL, df = 9)9 -4.679118   1.658707  -2.821  0.00479 **
## s(AGE, df = 4)          0.009851   0.001787   5.512 3.54e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 22196  on 21000  degrees of freedom
## Residual deviance: 21563  on 20990  degrees of freedom
## AIC: 21585
##
## Number of Fisher Scoring iterations: 5
```

```r
xgrid <- seq(min(test_data$LIMIT_BAL), max(test_data$LIMIT_BAL), len = 101)

newage <- median(test_data$AGE)

pred <- predict(gam_model, newdata = data.frame(LIMIT_BAL = xgrid, AGE = newage))

plot(xgrid, pred, type = "l", lwd = 2,
     xlab = "LIMIT_BAL", ylab = "f(LIMIT_BAL)")
```



What this plot is showing is that as the credit limit (LIMIT_BAL) increases, the probability of default decreases because the log-odds are going down. The x-axis shows values of the credit limit going all the way up to near 800,000. Higher credit limits are associated with a lower likelihood of defaulting.

4) Single tree model: Classification Tree. These are non-parametric and we can use them for limited inference. We get the splits based on the data but we don't have any coefficients or p-values that we can use to do hypothesis testing. The variable selection is built in to the algorithm and we don't need to scale our predictors. It chooses variables based on the split at each node.

I fit the model using rpart() and printed out the parameter table. This helped show us which pruning level minimizes the cv error. I also extracted the complexity parameter table to get the cv error rates for the different pruning levels and selected some specific levels and plotted the pruned tree using rpart.plot(). I used it to make predictions on the training data and compute the error matrix as well. I did this for for the 1-SE rule and the min cross cv error. I then evaluated the performance on the test set and printed out the confusion matrix to see certain metrics.

```r
library(rpart)
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.3.3
```

```r
library(klaR)
```

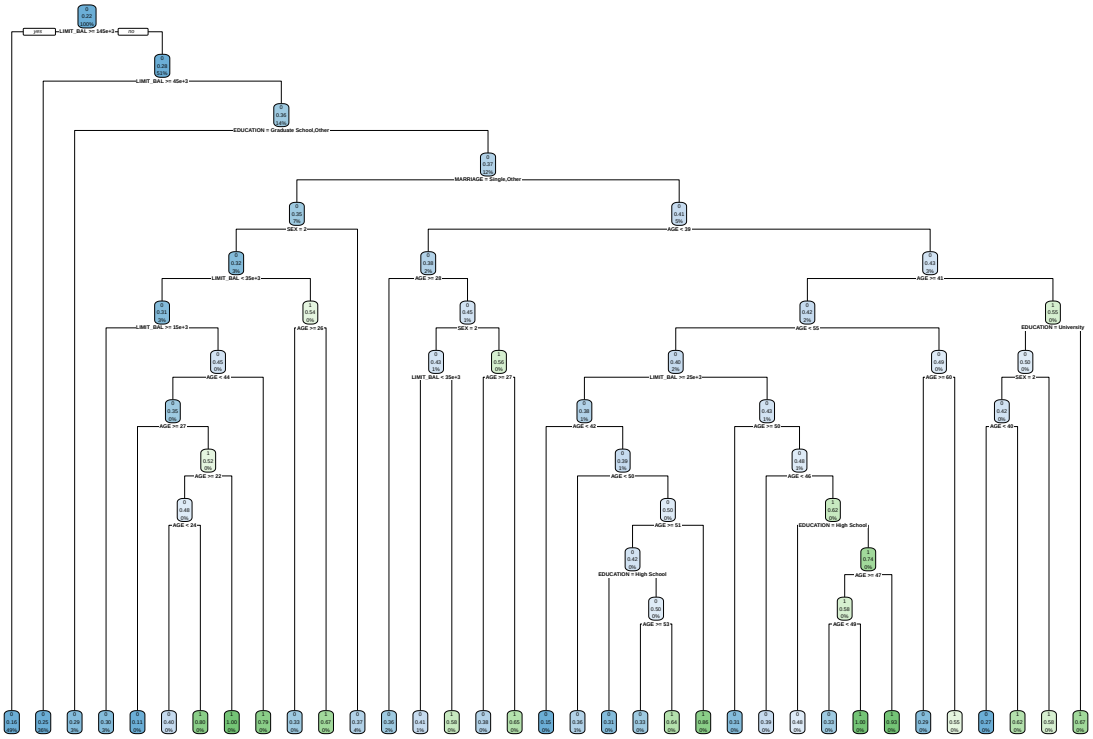
```
## Loading required package: MASS
```

```r
library(caret)
library(knitr)

set.seed(1001)

tree_model <- rpart(`default payment next month` ~ LIMIT_BAL + AGE + SEX + EDUCATION + MARRIAGE, data =

printcp(tree_model)
```

```
##
## Classification tree:
## rpart(formula = `default payment next month` ~ LIMIT_BAL + AGE +
##     SEX + EDUCATION + MARRIAGE, data = train_data, method = "class",
##     parms = list(split = "information"), control = rpart.control(xval = 10,
##         minbucket = 2, cp = 0))
##
## Variables actually used in tree construction:
## [1] AGE       EDUCATION LIMIT_BAL MARRIAGE  SEX
##
## Root node error: 4646/21001 = 0.22123
##
## n= 21001
##
##            CP nsplit rel error xerror     xstd
## 1  3.4438e-04      0   1.00000 1.0000 0.012947
## 2  3.2286e-04     25   0.98838 1.0043 0.012967
## 3  2.8699e-04     34   0.98515 1.0050 0.012970
## 4  2.7674e-04     49   0.98084 1.0071 0.012980
## 5  2.5111e-04     56   0.97891 1.0093 0.012990
## 6  2.4599e-04     62   0.97740 1.0093 0.012990
## 7  2.1524e-04     69   0.97568 1.0250 0.013061
## 8  1.9132e-04    109   0.96707 1.0351 0.013106
## 9  1.8213e-04    122   0.96449 1.0502 0.013173
## 10 1.7219e-04    194   0.94813 1.0590 0.013211
## 11 1.6741e-04    225   0.94210 1.0626 0.013227
## 12 1.6143e-04    234   0.94059 1.0654 0.013239
## 13 1.4349e-04    263   0.93543 1.0747 0.013279
## 14 1.3837e-04    294   0.93091 1.0818 0.013309
## 15 1.3452e-04    347   0.92079 1.0874 0.013332
## 16 1.2914e-04    372   0.91649 1.0906 0.013346
## 17 1.2299e-04    402   0.91240 1.0943 0.013361
## 18 1.1958e-04    423   0.90981 1.0941 0.013360
## 19 1.0762e-04    459   0.90486 1.1375 0.013536
## 20 1.0129e-04    632   0.88442 1.1436 0.013560
## 21 9.5662e-05    649   0.88269 1.1517 0.013592
## 22 9.2245e-05    770   0.86849 1.1601 0.013624
## 23 8.9683e-05    791   0.86655 1.1612 0.013628
## 24 8.6096e-05    803   0.86548 1.1638 0.013638
## 25 8.0715e-05    874   0.85923 1.1679 0.013654
## 26 7.1746e-05    884   0.85837 1.1965 0.013761
## 27 6.6227e-05    986   0.85084 1.2000 0.013774
## 28 6.4572e-05   1004   0.84933 1.2025 0.013783
## 29 6.1497e-05   1023   0.84783 1.2032 0.013785
## 30 5.8702e-05   1078   0.84417 1.2032 0.013785
```

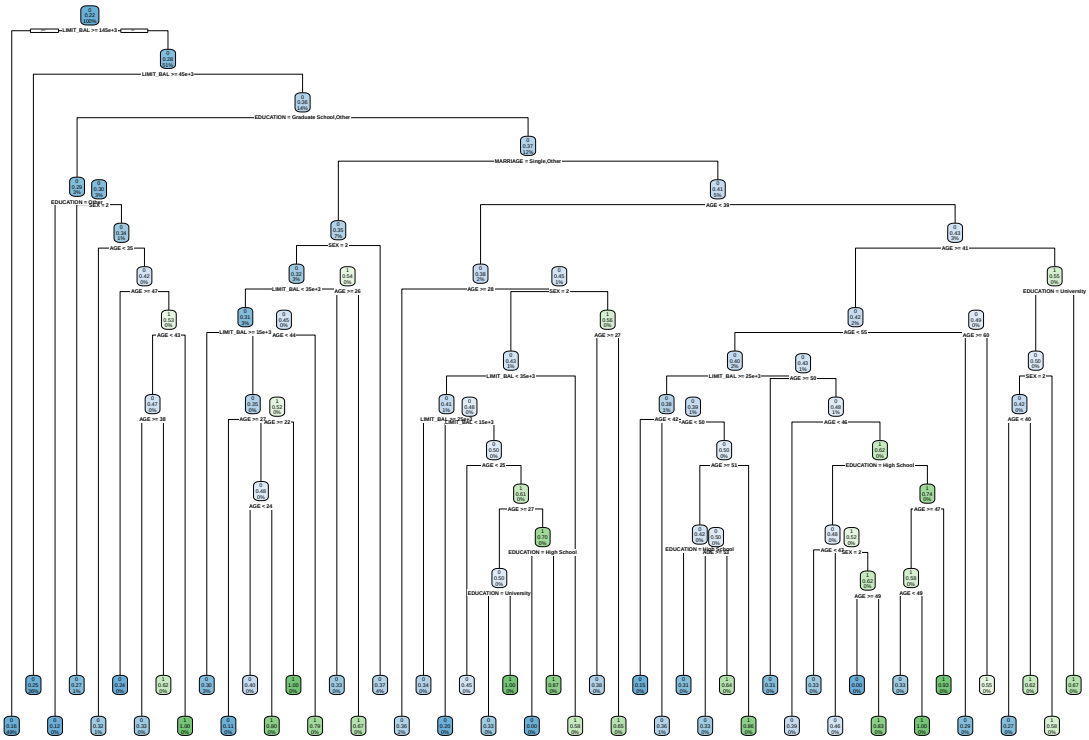
```
## 31 5.3810e-05   1092   0.84331 1.2183 0.013840
## 32 4.9671e-05   1255   0.83384 1.2195 0.013845
## 33 4.7831e-05   1268   0.83319 1.2195 0.013845
## 34 4.3048e-05   1277   0.83276 1.2266 0.013870
## 35 3.9134e-05   1312   0.83125 1.2269 0.013871
## 36 3.5873e-05   1323   0.83082 1.2316 0.013887
## 37 3.0748e-05   1355   0.82953 1.2337 0.013895
## 38 2.6905e-05   1397   0.82824 1.2344 0.013897
## 39 0.0000e+00   1421   0.82759 1.2357 0.013902
```

```r
cp <- tree_model$cptable

final_tree1se <- prune(tree_model, cp = cp[3, 1])

rpart.plot(final_tree1se)
```



```r
final_tree_mincv <- prune(tree_model, cp = cp[4, 1])

rpart.plot(final_tree_mincv)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```
pred_train1se <- predict(final_tree1se, newdata = train_data, type = "class")

train_error1se <- klaR::errormatrix(true = train_data$`default payment next month`,  predicted = pred_t

train_error1se |> kable()
```

	0	1	-SUM-
0	0.9959645	0.0040355	0.0040355
1	0.9709427	0.0290573	0.9709427
-SUM-	0.9855801	0.0144199	0.2179420

```
pred_train_mincv <- predict(final_tree_mincv, newdata = train_data, type = "class")

train_error_mincv <- klaR::errormatrix(true = train_data$`default payment next month`, predicted = pred

train_error_mincv |> kable()
```

	0	1	-SUM-
0	0.9954754	0.0045246	0.0045246
1	0.9649161	0.0350839	0.9649161
-SUM-	0.9837612	0.0162388	0.2169897

```
pred_test1se <- predict(final_tree1se, newdata = test_data, type = "class")

confusionMatrix(pred_test1se, test_data$`default payment next month`)
```

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

```
##           Reference
## Prediction    0    1
##          0 6970 1950
##          1   39   40
##
##                Accuracy : 0.779
##                  95% CI : (0.7703, 0.7875)
##     No Information Rate : 0.7789
##     P-Value [Acc > NIR] : 0.4959
##
##                   Kappa : 0.0222
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9944
##             Specificity : 0.0201
##          Pos Pred Value : 0.7814
##          Neg Pred Value : 0.5063
##              Prevalence : 0.7789
##          Detection Rate : 0.7745
##    Detection Prevalence : 0.9912
##       Balanced Accuracy : 0.5073
##
##        'Positive' Class : 0
##
```

```r
pred_test_mincv <- predict(final_tree_mincv, newdata = test_data, type = "class")

confusionMatrix(pred_test_mincv, test_data$`default payment next month`)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6955 1943
##          1   54   47
##
##                Accuracy : 0.7781
##                  95% CI : (0.7694, 0.7866)
##     No Information Rate : 0.7789
##     P-Value [Acc > NIR] : 0.5764
##
##                   Kappa : 0.0241
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99230
##             Specificity : 0.02362
##          Pos Pred Value : 0.78164
##          Neg Pred Value : 0.46535
##              Prevalence : 0.77886
##          Detection Rate : 0.77286
##    Detection Prevalence : 0.98878
##       Balanced Accuracy : 0.50796
##
```

```
##        'Positive' Class : 0
##
```

Like before with the logistic regression model, we're doing a really strong job of identifying the default class but the model doesn't do as well identifying the non-default class since we have a low specificity. This is again due to an imbalance in our data so we'd have to use other techniques such sampling balancing (oversampling the minority class or undersampling the majority class).

5) Ensemble Tree Models - Boosting Regression Trees: This is a non-parametric model since it learns from the patterns in the data. There are tuning parameters such as the best number of trees. We can't really use them for inference since they're used for predicting and we don't get coefficients or p-values to do hypothesis testing on. They do variable selection as well because we pick trees based on importance when splitting them. We do not need to standardize our predictors.

I had to convert the response to numeric to fit the boosted model and when I went to predict on the new data, I turned the predictions back into classes with a 0.5 threshold. After fitting all the predictors and looking at the relative influence, the most important predictors were: PAY_0, PAY_2, BILL_AMT1, PAY_3, and LIMIT_BAL. So, instead of fitting all the predictors I only chose those predictors to fit the model.

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.3.3
```

```
## Loaded gbm 2.2.2
```

```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

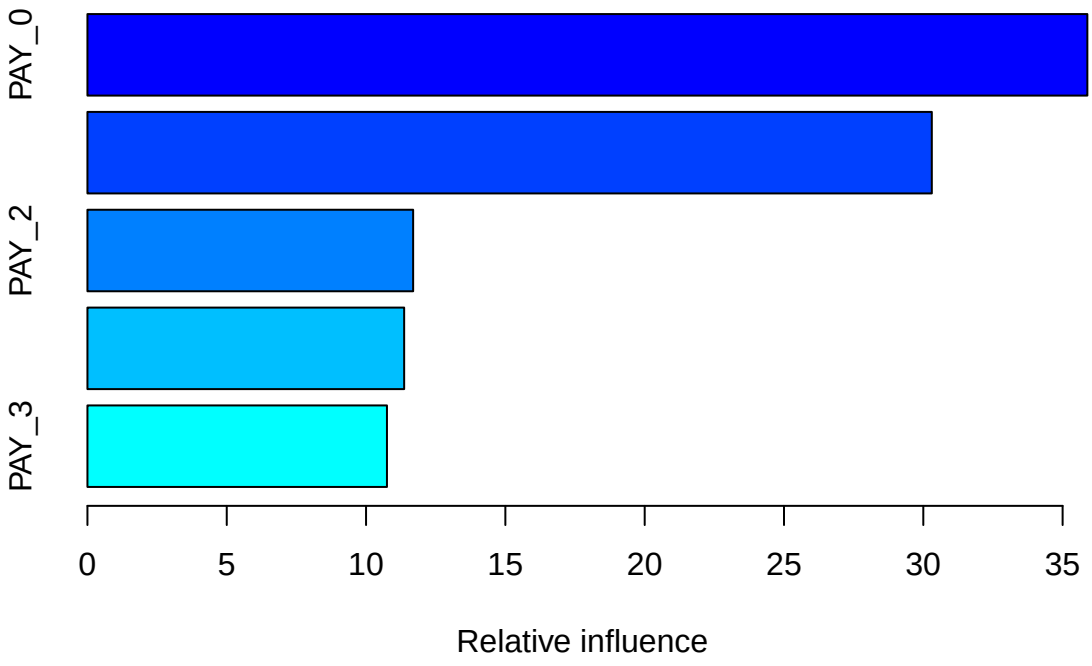
```
train_data$`default payment next month` <- as.numeric(as.character(train_data$`default payment next mon

set.seed(1001)

boosted_default <- gbm(
  formula = `default payment next month` ~ PAY_0 + PAY_2 + PAY_3 + BILL_AMT1 + LIMIT_BAL,
  data = train_data,
  distribution = "bernoulli",
  n.trees = 1000,
  shrinkage = 0.1*2,
  interaction.depth = 3,
  n.minobsinnode = 10,
  cv.folds = 5
)

print(boosted_default)
```

```
## gbm(formula = `default payment next month` ~ PAY_0 + PAY_2 +
##     PAY_3 + BILL_AMT1 + LIMIT_BAL, distribution = "bernoulli",
##     data = train_data, n.trees = 1000, interaction.depth = 3,
##     n.minobsinnode = 10, shrinkage = 0.1 * 2, cv.folds = 5)
## A gradient boosted model with bernoulli loss function.
## 1000 iterations were performed.
## The best cross-validation iteration was 59.
## There were 5 predictors of which 5 had non-zero influence.
```

```
best <- which.min(boosted_default$cv.error)
best
```

```
## [1] 59
```

```
summary(boosted_default)
```

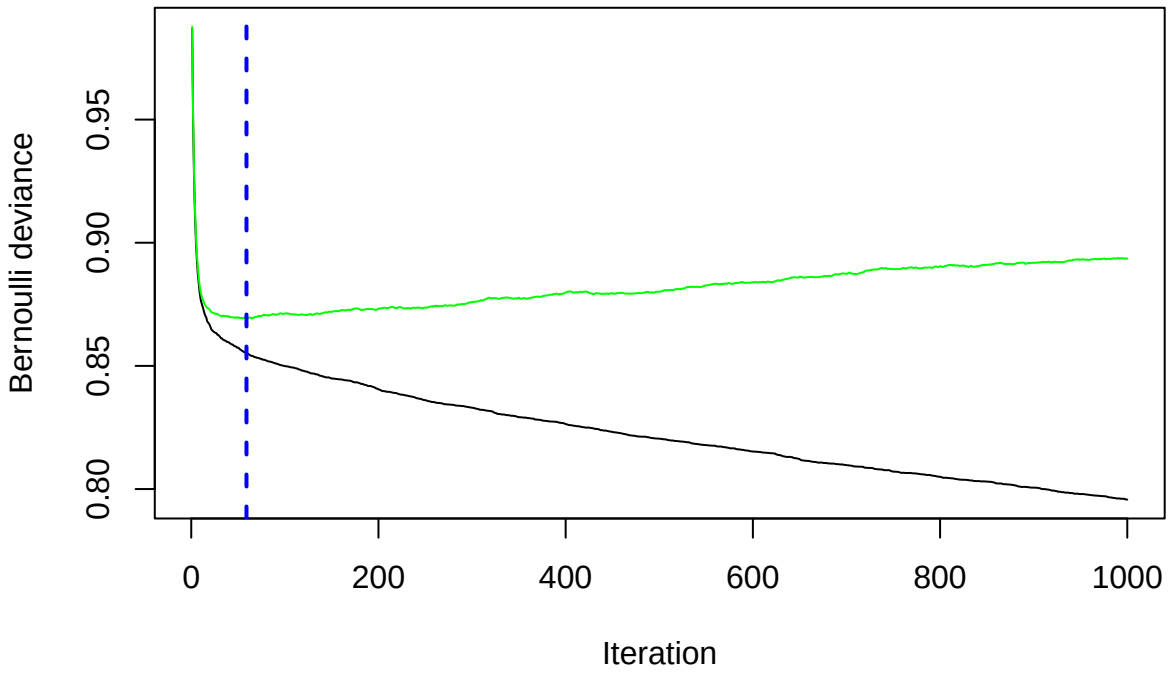



```
##                 var  rel.inf
## PAY_0         PAY_0 35.88801
## BILL_AMT1 BILL_AMT1 30.30502
## PAY_2         PAY_2 11.69019
## LIMIT_BAL LIMIT_BAL 11.36708
## PAY_3         PAY_3 10.74971
```

```
boosted_default$cv.error[best]
```

```
## [1] 0.8692321
```

```
gbm.perf(boosted_default, method = "cv")
```




```
## [1] 59
```

```
pred <- predict(boosted_default, newdata = test_data, n.trees = best, type = "response")

pred_class <- ifelse(pred > 0.35, 1, 0)

conf_matrix <- confusionMatrix(factor(pred_class), factor(test_data$`default payment next month`))
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6400 1114
##          1  609  876
##
##                Accuracy : 0.8085
##                  95% CI : (0.8003, 0.8166)
##     No Information Rate : 0.7789
##     P-Value [Acc > NIR] : 2.936e-12
##
##                   Kappa : 0.3886
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9131
##             Specificity : 0.4402
##          Pos Pred Value : 0.8517
##          Neg Pred Value : 0.5899
##              Prevalence : 0.7789
##          Detection Rate : 0.7112
##    Detection Prevalence : 0.8350
##       Balanced Accuracy : 0.6767
```

```
##
##        'Positive' Class : 0
##
```

It looks like the optimal number of trees is 59 (lowest cross validation error). The minimum cv error is 0.869 which is too high because this means the model's log loss at the best iteration is around 0.869. When we fit the model to unseen data with a threshold of 0.5, we get an accuracy of about 81%, similar to the logistic regression model. The sensitivity of 95% tells us that the model does a really strong job of predicting the default class but a 31% of specificity shows us that it's not doing that good of a job predicting the non-default class.

However, setting the threshold to 0.35 on the test set when predicting, we get a better balance between the sensitivity and specificity levels (91% and 44%), which may be a better threshold. Again, we'd probably want to perform oversampling and undersampling in the minority and majority classes, respectively.

6) Support Vector Machines: SVM is a non-parametric model and it does have tuning parameters such as cost and kernel type can also be a tuning parameter. We don't get coefficients or p-values since this is a classification task so we cannot make inference. It doesn't do feature selection directly but it does focus on providing features that define the support vectors. We do have to standardize the predictors because we're basing our predictions on certain distances.

I had to standardize the predictors separately because when I tried using the argument scale = TRUE, I got an error saying x must be numeric. My predictors are factors so I first selected only my numeric variables, scale them, and make sure the factor predictors aren't changed.

set.seed(1001) tr <- trainControl(method = "repeatedcv", number = 3, repeats = 5) tune_grid <- expand.grid(cost = exp(seq(-2,2,len=10))) sv_caret <- train(`default payment next month` ~ PAY_0 + PAY_2 + PAY_AMT1 + BILL_AMT1 + LIMIT_BAL + AGE, data = scaled_train, method = "svmLinear2", tuneGrid = tune_grid, trControl = tr)

I tried using the code above but I kept reaching the max number of iterations because of the size of the dataset. This lead to the dataset not converging. I chose to tune manually instead. I created a for loop that looped over the different values of the cost tuning parameter and that ran a bit faster compared to before.

```
library(e1071)
library(caret)
library(klaR)
library(knitr)


train_data$`default payment next month` <- as.factor(train_data$`default payment next month`)

test_data$`default payment next month` <- as.factor(test_data$`default payment next month`)

scaled_train <- train_data
scaled_test <- test_data

responsevar <- "default payment next month"

numeric_colmns <- sapply(scaled_train, is.numeric)
numeric_colmns[responsevar] <- FALSE

scaled_train[, numeric_colmns] <- scale(scaled_train[, numeric_colmns])

library(e1071)

svm_default <- svm(
```

```
  `default payment next month` ~ PAY_0 + PAY_2 + PAY_AMT1 + BILL_AMT1 + LIMIT_BAL + AGE,
  data = scaled_train,
  type = "C-classification",
  kernel = "linear",
  cost = 1
)

beta_hat <- coef(svm_default)
beta_hat
```

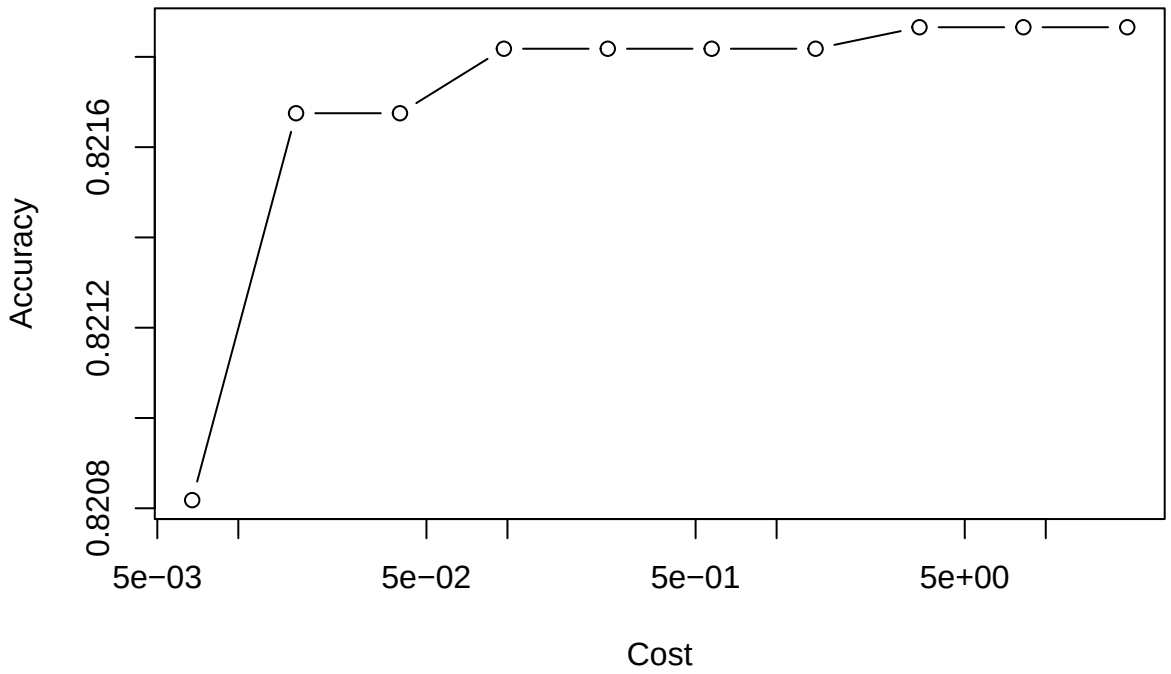
```
##   (Intercept)         PAY_0         PAY_2      PAY_AMT1     BILL_AMT1
## -2.546293e-01 -7.452654e-01 -7.454135e-01 -7.454439e-01 -7.454453e-01
##     LIMIT_BAL           AGE          <NA>          <NA>          <NA>
##  1.254405e+00  1.254504e+00  1.254195e+00  4.184935e-01 -7.455096e-01
##          <NA>          <NA>          <NA>          <NA>          <NA>
##  4.180975e-01 -8.726175e-01  1.367406e-04  1.417648e-04  1.900929e-04
##          <NA>          <NA>          <NA>          <NA>          <NA>
##  1.299587e-04  3.372966e-04  2.369401e-04  1.959101e-04  8.365910e-01
##          <NA>          <NA>          <NA>          <NA>          <NA>
##  1.273825e-01  0.000000e+00 -4.913244e-06 -1.685308e-05 -9.645066e-05
##          <NA>
##  6.409417e-06
```

```
cost_values <- exp(seq(-5, 3, length.out = 10))


results <- data.frame(cost = cost_values, accuracy = NA)

for (i in 1:length(cost_values)) {

svm_model <- svm(
    `default payment next month` ~ PAY_0 + PAY_2 + PAY_AMT1 + BILL_AMT1 + LIMIT_BAL + AGE,
    data = scaled_train,
    type = "C-classification",
    kernel = "linear",
    cost = cost_values[i],
    scale = FALSE
  )

pred <- predict(svm_model, scaled_train)


accuracy <- mean(pred == scaled_train$`default payment next month`)


results$accuracy[i] <- accuracy
}

# Print results
print(results)
```

```
##           cost  accuracy
## 1   0.006737947 0.8208181
## 2   0.016389554 0.8216752
```

```
## 3   0.039866368 0.8216752
## 4   0.096971968 0.8218180
## 5   0.235877083 0.8218180
## 6   0.573753421 0.8218180
## 7   1.395612425 0.8218180
## 8   3.394723187 0.8218656
## 9   8.257411091 0.8218656
## 10 20.085536923 0.8218656
```

```r
plot(results$cost, results$accuracy, type = "b", xlab = "Cost", ylab = "Accuracy", log = "x")
```




```r
final_svm <- svm(
  `default payment next month` ~ PAY_0 + PAY_2 + PAY_AMT1 + BILL_AMT1 + LIMIT_BAL + AGE,
  data = scaled_train,
  type = "C-classification",
  kernel = "linear",
  cost = 3.39,
  scale = FALSE
)

summary(final_svm)
```

```
##
## Call:
## svm(formula = `default payment next month` ~ PAY_0 + PAY_2 + PAY_AMT1 +
##     BILL_AMT1 + LIMIT_BAL + AGE, data = scaled_train, type = "C-classification",
##     kernel = "linear", cost = 3.39, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  3.39
##
```

```
## Number of Support Vectors:  7893
##
##  ( 3811 4082 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
pred_class <- predict(final_svm, newdata = scaled_test)

mean(pred_class == scaled_test$`default payment next month`)
```

```
## [1] 0.2646961
```

It seems like the model's accuracy on the test set is about 0.5703, which is a bit low. This tells us that there may be some overfitting, where the model performs well on the training data but struggles to generalize to the test data. We also may have to use sampling techniques here to account for the data imbalance (oversampling and undersampling).

Which model do I like best?

The best model for this data would be the logistic regression model. The reason I say that is because the main goal is for us to predict the actual loan defaults. With a threshold of 0.5, we had a really high sensitivity which meant the model predicted really well for the default class. If we decrease the threshold to 0.35, then we had a better balance between sensitivity and specificity. So, overall, the logistic regression model would be the best model to fit here due to its predictive ability which is our main goal here (predicting who would default based on the given predictors).

Refitting the logistic model to the entire data set

I combined both training and testing sets using ribnd and created a dataframe calling it X_combined and scaled it like I did before in the original model.

```
combined_data <- rbind(train_data, test_data)

X_combined <- combined_data[, !names(combined_data) %in% c("ID", "default payment next month")]

X_combined <- data.frame(lapply(X_combined, as.numeric))
y_combined <- combined_data$`default payment next month`

X_combined <- scale(X_combined)

cv_lasso_full <- cv.glmnet(X_combined, y_combined, family = "binomial", alpha = 1)

lasso_model_full <- glmnet(X_combined, y_combined, family = "binomial", alpha = 1, lambda = cv_lasso_ful

coef(lasso_model_full)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) -1.408146721
## LIMIT_BAL   -0.092759125
## SEX         -0.003928558
## EDUCATION    .
## MARRIAGE    -0.038816274
## AGE          0.014116789
```

```
## PAY_0        0.631269319
## PAY_2        0.090686081
## PAY_3        0.086696980
## PAY_4        0.031590184
## PAY_5        .
## PAY_6        .
## BILL_AMT1   -0.093400315
## BILL_AMT2    .
## BILL_AMT3    .
## BILL_AMT4    .
## BILL_AMT5    .
## BILL_AMT6    .
## PAY_AMT1    -0.072918938
## PAY_AMT2    -0.024648431
## PAY_AMT3     .
## PAY_AMT4    -0.005064395
## PAY_AMT5    -0.003921781
## PAY_AMT6     .
```

```
lasso_predictions_full <- predict(lasso_model_full, newx = X_combined, s = cv_lasso_full$lambda.1se, ty

threshold <- 0.35
lasso_pred_class_full <- ifelse(lasso_predictions_full > threshold, 1, 0)

confusionMatrix(factor(lasso_pred_class_full, levels = c(0, 1)), factor(y_combined, levels = c(0, 1)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 21604  3830
##          1  1760  2806
##
##                Accuracy : 0.8137
##                  95% CI : (0.8092, 0.8181)
##     No Information Rate : 0.7788
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3912
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9247
##             Specificity : 0.4228
##          Pos Pred Value : 0.8494
##          Neg Pred Value : 0.6145
##              Prevalence : 0.7788
##          Detection Rate : 0.7201
##    Detection Prevalence : 0.8478
##       Balanced Accuracy : 0.6738
##
##        'Positive' Class : 0
##
```