XGBoost Binary Regression Model Practice Using Basic Loan Data

Daniel Tetrick dantetrickk@gmail.com

Date: 2019-03-07

Introduction

This script is a created to display my skills using machine learning techniques. I will take historical bank lending data found in the Revolution R package and train an XGBoost binary regresssion model to predict future default loans. The data set consists of 100,000 rows of bank mortage data from the years 2000-2009. The columns available are: <creditScore, houseAge, yearsEmploy, ccDebt, year, default>

Set Project Parameters

Set scientific notation to off and clean garbage.

```
# Set Project Parameters
gc()
```

```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  520031 27.8    1177781   63   616042 33.0
## Vcells 1014411  7.8    8388608   64  1636404 12.5
```

```
options(scipen = 999)
```

Load all necessary packages

If the output shows a "FALSE" value, use the "install.packages()" to load that package and re-run script.

```
# Load all packages
sapply(c("dplyr", "data.table", "plotROC","car","caret", "glmnet", "xgboost"), function(x) {
  require(x, character.only = T)
  })
```

```
##      dplyr data.table    plotROC        car      caret     glmnet
##       TRUE       TRUE       TRUE       TRUE       TRUE       TRUE
##    xgboost
##       TRUE
```

Set Paths and Create Directories

```
## [1] FALSE FALSE FALSE FALSE
```

Load all csv's in data folder and create full data set

Load all data from Data folder as lists of data frames. Bind all dataframes into single data frame and randomize rows. Finally remove year varibale because we won't be using it in the first analysis. In the future, we can add the time element if desired.

```
# Load data
filesToLoad <- lapply(list.files(DataPath, pattern = ".csv", full.names = T), function(x) {fread(x)})

# Bind All data and mix up
```

```
set.seed(8675309)
dt <- rbindlist(filesToLoad) %>%
  sample_frac(1) %>%
  select(-year)

data.table(dt)
```

```
##         creditScore houseAge yearsEmploy ccDebt default
##      1:         687       26           6   9637       0
##      2:         729       20           7   5664       0
##      3:         619       20           5     99       0
##      4:         599       21           2   4001       0
##      5:         774       24           6   4653       0
##     ---
##  99996:         628       23           6   3026       0
##  99997:         758        6           7   5230       0
##  99998:         639       25           3   6483       0
##  99999:         692       18           5   2399       0
## 100000:         696       20           2   6037       0
```

It is important to note that the data is 'shuffled' here to control for biased sampling later.

Create a Default Data set

Isolate all default loans into a single data set in prep for analysis. Display.

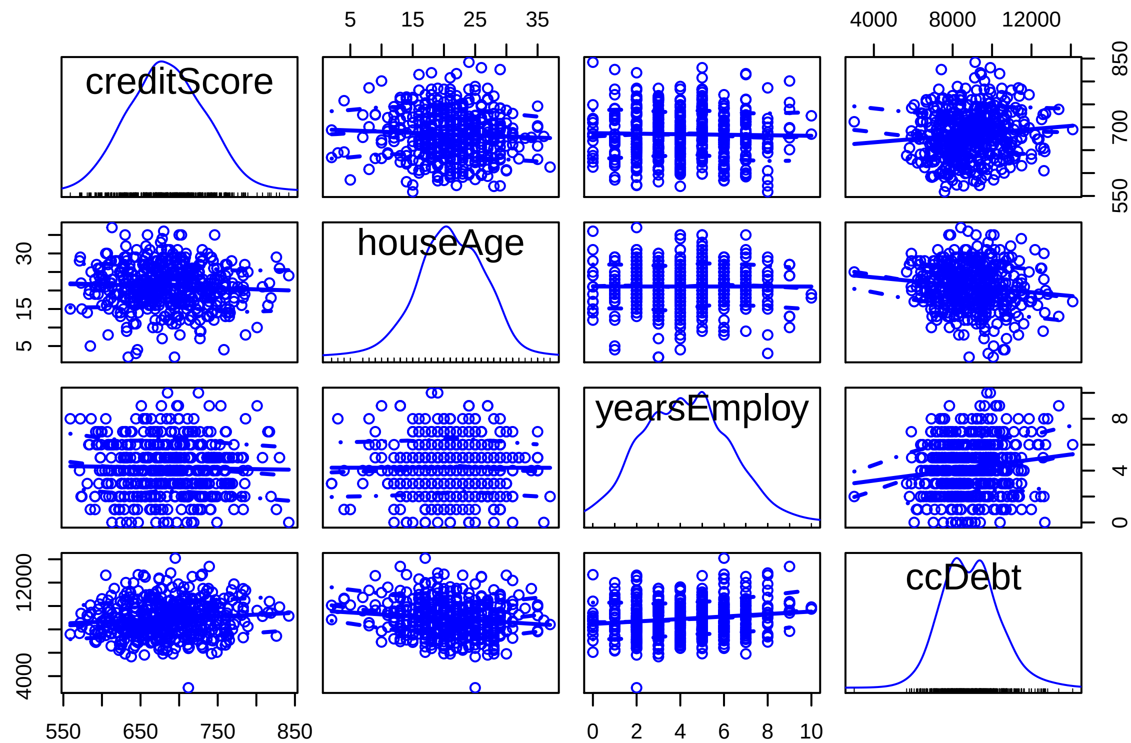
```
# Separate Defaults and create a test and train data sets
dt_default <- dt[default == 1, ]

data.table(dt_default)
```

```
##      creditScore houseAge yearsEmploy ccDebt default
##   1:         689       19           8  12823       1
##   2:         783       20           4  11482       1
##   3:         712       20           7   8851       1
##   4:         669       12           1  10714       1
##   5:         776       21           5   9924       1
##  ---
## 467:         749       14           4   8409       1
## 468:         662       18           1  10979       1
## 469:         713       12           6   9011       1
## 470:         727        7           4  10735       1
## 471:         677       12           4   9022       1
```

Create a Default Data set

Use a scatterplotMatrix from the Car package to look for bi-variate interacations withinn scatter plots and histograms of all variables.

```
# Scatterplot
defaultScatters <- scatterplotMatrix(dt_default %>% select(creditScore, houseAge, yearsEmploy, ccDebt))
```



Create a Test and Train dataset of the 'default' values

Create a test data set of 50 rows of data from the 471 overall default loans found. These values will be held aside until we predict our final model. The remaining 421 data points will be used in the training data to create the final model.

```
# Keep all but 50 random default for training data sets
dt_default_train <- dt_default[1:(nrow(dt_default)-50), ] %>%
  select(creditScore, houseAge, yearsEmploy, ccDebt, default)

# Keep th 50 random default for test data sets
dt_default_test <- dt_default[((nrow(dt_default)-49):nrow(dt_default)), ] %>%
  select(creditScore, houseAge, yearsEmploy, ccDebt, default)
```

Create Non-default data set and a 200 point sample for testing

Create a dataset of all non-default mortage data and take 200 values for a test data set.

```
# Separate the nondefault data from default
dt_nondefault <- dt[default == 0, ]

# Keep 4x as many non default points for testing later
dt_nondefault_test <- dt_nondefault[1:200,]
```

Create Full Test data set by unioning the test default and non-default data sets.

Create a dataset of all non-default mortage data and take 200 values for a test data set.

```
# Separate the nondefault data from default
dt_nondefault <- dt[default == 0, ]

# Keep 4x as many non default points for testing later
dt_nondefault_test <- dt_nondefault[1:200,]
```

Union default and non-default test data sets

Create a 250 row table to be used in prediction later. Dataset consists of 200 non-default and 50 default values.

```
# Create full test data for later
dt_test <- bind_rows(dt_default_test, dt_nondefault_test) %>%
  select(creditScore, houseAge, yearsEmploy, ccDebt, default) %>%
  sample_frac(1) %>%
  as.matrix()

data.table(dt_test)
```

```
##      creditScore houseAge yearsEmploy ccDebt default
##   1:         749       13           5   8763       0
##   2:         634       16           6   5519       0
##   3:         737       30           7   9973       1
##   4:         734       26           8   6517       0
##   5:         667       23           5   5741       0
##  ---
## 246:         728       20           3   9487       0
## 247:         676       39           6   3221       0
## 248:         697       23           5   1477       0
## 249:         578       17           6   5326       0
## 250:         658       20           8   9989       1
```

Create xgboost Matrix of test data

Create a dataset of all non-default mortage data and take 200 values for a test data set.

```
xgbMatrix_test <- xgb.DMatrix(data = dt_test[,1:4]
                              , label = dt_test[,5]
                              )

xgbMatrix_test
```

```
## xgb.DMatrix  dim: 250 x 4  info: label  colnames: yes
```

Create Non-Default training data set.

Instead of using the entire 99K+ data set with the 421 default rows, we will take a sample of the data and use it to train the model. Remember to exclude the top 200 rows that have been designated as the test data set to be used in prediction. After top 200 rows are removed, we take a random sample 4x the size of the total number of 'default' values found. The sample size is arbitrary for now, but it is a decent fold size in my experience to get a model working and ready to iterate on.

```
# Remove all testing points from nondefault to create overall training set
dt_nondefault_train <- dt_nondefault[201:nrow(dt_nondefault),]

# Create sample data set that is 4x larger than total number of default loand found.
```

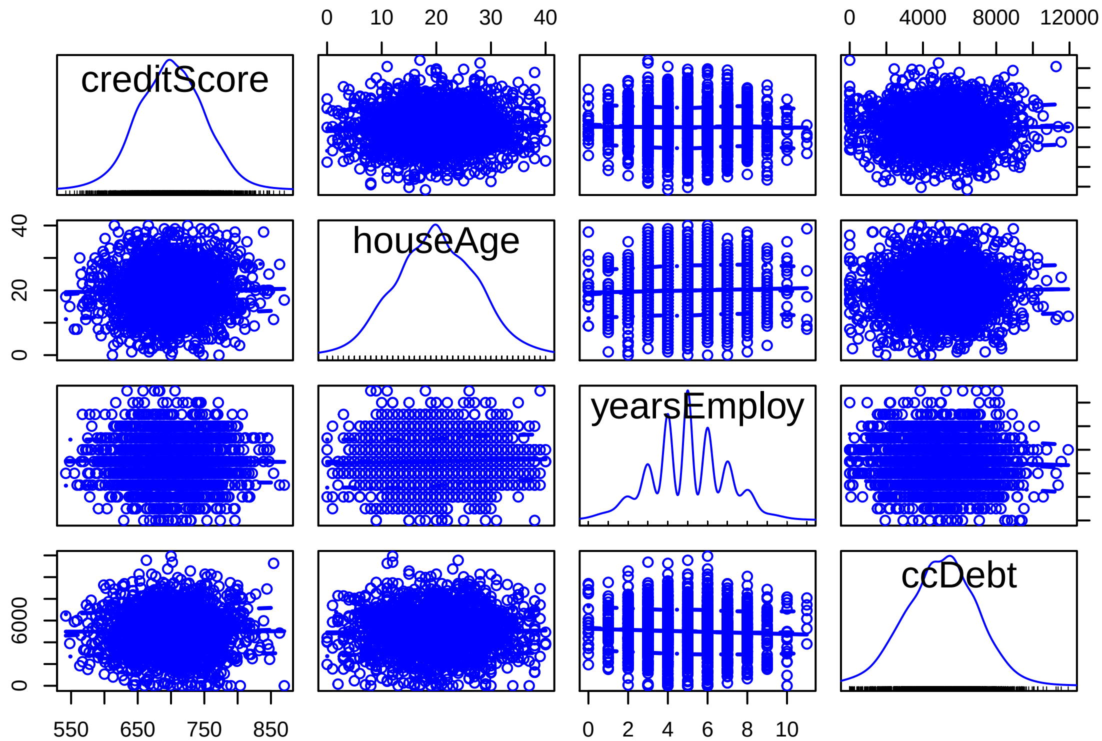
```r
set.seed(8675309)
dt_nondefault_samp <- dt_nondefault_train %>%
  sample_n(nrow(dt_default)*4) %>%
  select(creditScore, houseAge, yearsEmploy, ccDebt, default)

data.table(dt_nondefault_samp)
```

```
##       creditScore houseAge yearsEmploy ccDebt default
##    1:         779       28           6   4917       0
##    2:         795       29           6   7384       0
##    3:         718       27           6   4623       0
##    4:         667       25           5   8650       0
##    5:         660       27          10   2177       0
##   ---
## 1880:         674       26           8   4775       0
## 1881:         694       14           3   2861       0
## 1882:         770       17           3   2855       0
## 1883:         675       34           5   6066       0
## 1884:         721       10           3   4061       0
```

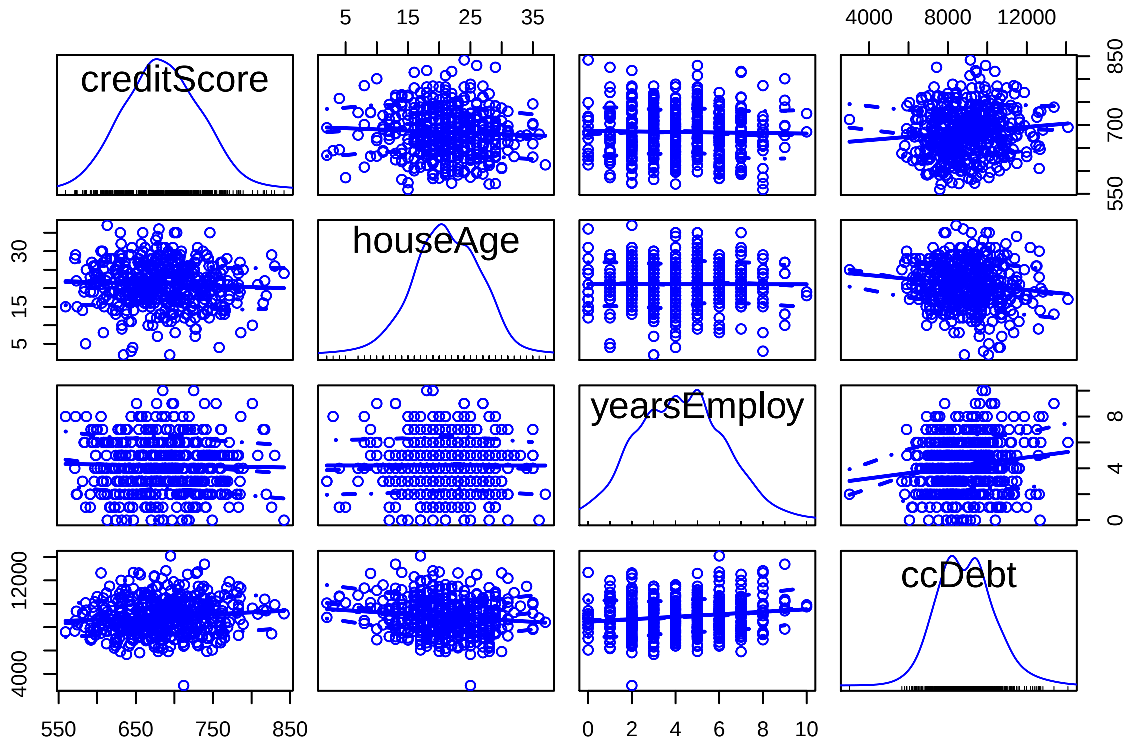
Create Scatterplot for Non-Default training data

Scatterplot matrix for the non-default data set, juxtapose this with the default data scatterplot earlier. There is a noticeable difference between ccDebt and its interactions with other variables when comparing the two scatter plot matrices.

```r
# Create Scatters
nondefaultScatters_samp <- scatterplotMatrix(dt_nondefault_samp %>%
                                               select(-default)
                                             )
```




```
# Scatterplot
defaultScatters <- scatterplotMatrix(dt_default %>%
                                        select(creditScore, houseAge, yearsEmploy, ccDebt))
```



Create Scatterplot for Non-Default training data

Create the model training data set by unioning the non-default sample and the default training data sets. Shuffle the data and create a matrix.

```
set.seed(8675309)
dt_train <- bind_rows(dt_nondefault_samp, dt_default_train) %>%
  sample_frac(1) %>%
  as.matrix()
```

Create Xgboost Matrix for training

Create the Xgboost matrix for the training data set.

```
xgbMatrix_train <- xgb.DMatrix(dt_train[, 1:4], label = dt_train[,5])

xgbMatrix_train
```

```
## xgb.DMatrix  dim: 2305 x 4  info: label  colnames: yes
```

Create Xgboost Matrix for training

Create a Cross-validated xgboosted binary logistic model. A thousand rounds using a slow and deep learning rate of .01. Using 4 nfolds to create a 75/25 cv. Max depth is 4 but preliminary work showed that 3 and 4 are relatively equal. More work can be done to train the model, but this is a good display, and pretty good. CV example output displayed next code chunk.

```
cv <- xgb.cv(data = xgbMatrix_train
             , nrounds = 1000
             , nfold = 4
             , verbose = F
             , metrics = list("rmse","auc")
             , max_depth = 4
             , eta = .01
             , objective = "binary:logistic"
             )
```

Pick Best CV from the XGBoost evaluation log

Pick best number of rounds of XGboosting by selecting the highest AUC value from the CV matrix. Return the NROUNDS

```
# Pick the best CV
NROUNDS <- cv$evaluation_log[, which(cv$evaluation_log$test_auc_mean == max(cv$evaluation_log$test_auc_r

NROUNDS
```

```
## [1] 675
```

Pick Best CV from the XGBoost evaluation log

Create the final data model by using the NROUNDS chosen by CV. Use same exact parameters used in CV.

```
XGBoost_Model <-   xgboost(data = xgbMatrix_train
                           , nfold = 4
                           , max.depth = 4
                           , nrounds = NROUNDS
                           , nthread = 4
                           , metrics = list("rmse","auc")
                           , eta = .01
                           , verbose = F
                           , objective = "binary:logistic")
```

Look at importance matrix

Imporance matrix displays a ranking of each variables importance in the model.

```
data.table(xgb.importance(model = XGBoost_Model))
```

```
##        Feature       Gain     Cover Frequency
## 1:      ccDebt 0.86474781 0.6592355 0.4696318
## 2: creditScore 0.06138678 0.1298754 0.2231787
## 3:    houseAge 0.04585960 0.1354629 0.1971943
## 4: yearsEmploy 0.02800581 0.0754262 0.1099952
```

Predict the xgboost test matrix data using the Xgboost model

Use the predict function on the xgboost test data and project the values onto the dt_test data set created earlier.

```
# Predict Model
dt_test <- dt_test %>%
  tbl_df() %>%
  mutate(prediction = predict(XGBoost_Model
```

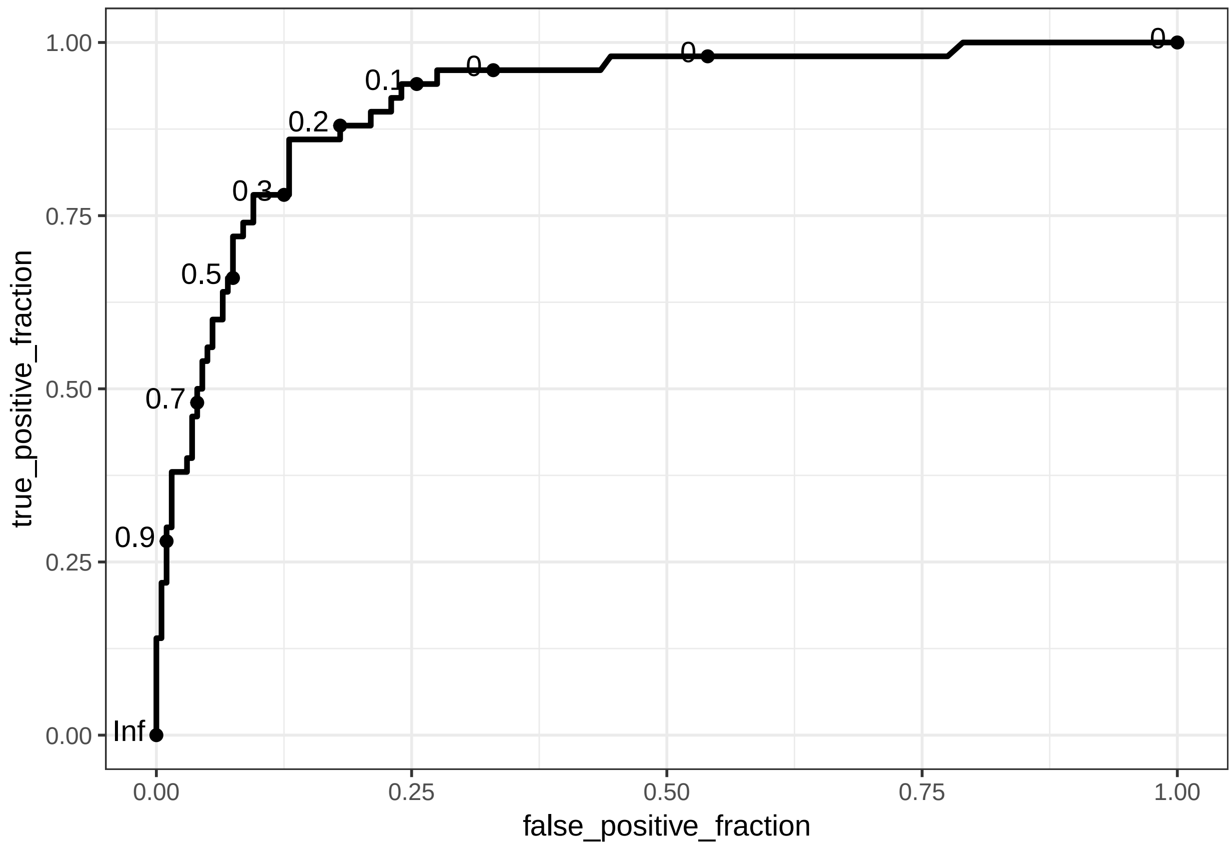
```r
                              , newdata = xgbMatrix_test
                              ))

data.table(dt_test)
```

```
##      creditScore houseAge yearsEmploy ccDebt default  prediction
##   1:         749       13           5   8763       0 0.401021361
##   2:         634       16           6   5519       0 0.001301979
##   3:         737       30           7   9973       1 0.752586603
##   4:         734       26           8   6517       0 0.050652627
##   5:         667       23           5   5741       0 0.001161993
##  ---
## 246:         728       20           3   9487       0 0.782866001
## 247:         676       39           6   3221       0 0.001361457
## 248:         697       23           5   1477       0 0.001266424
## 249:         578       17           6   5326       0 0.001301979
## 250:         658       20           8   9989       1 0.852065504
```

Create ROC curve of the test

Use the predict function on the xgboost test data and project the values onto the dt_test data set created earlier.

```r
ggplot(dt_test, aes(d = default, m = prediction)) +
  geom_roc() +
  theme_bw()
```



Calculate AUC of prediction

Use the AUC function on the predicted data to calculate the area-under-the-curve of above ROC curve.

```
auc(dt_test$default, dt_test$prediction)
```

```
## [1] 0.91675
```

Create a 101 Sequence Grid Search of all possible confusion matrices values

Sequence between 0 and 1 by .01 to create a grid search value string to use to find best cut line on ROC.

```
Sequences = seq(0,1, by = .01)

Sequences
```

```
##   [1] 0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13
##  [15] 0.14 0.15 0.16 0.17 0.18 0.19 0.20 0.21 0.22 0.23 0.24 0.25 0.26 0.27
##  [29] 0.28 0.29 0.30 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.40 0.41
##  [43] 0.42 0.43 0.44 0.45 0.46 0.47 0.48 0.49 0.50 0.51 0.52 0.53 0.54 0.55
##  [57] 0.56 0.57 0.58 0.59 0.60 0.61 0.62 0.63 0.64 0.65 0.66 0.67 0.68 0.69
##  [71] 0.70 0.71 0.72 0.73 0.74 0.75 0.76 0.77 0.78 0.79 0.80 0.81 0.82 0.83
##  [85] 0.84 0.85 0.86 0.87 0.88 0.89 0.90 0.91 0.92 0.93 0.94 0.95 0.96 0.97
##  [99] 0.98 0.99 1.00
```

Custom Confusion Matrix Calcluator (GridConfuser)

Calculates all Correct predictions, False Positives, and False Negatives across all values in the grid search. From that calculation, create a confusion matrix for each each grid search. Finally, create a bar plot of each confustion matrix and return all elements in the output.

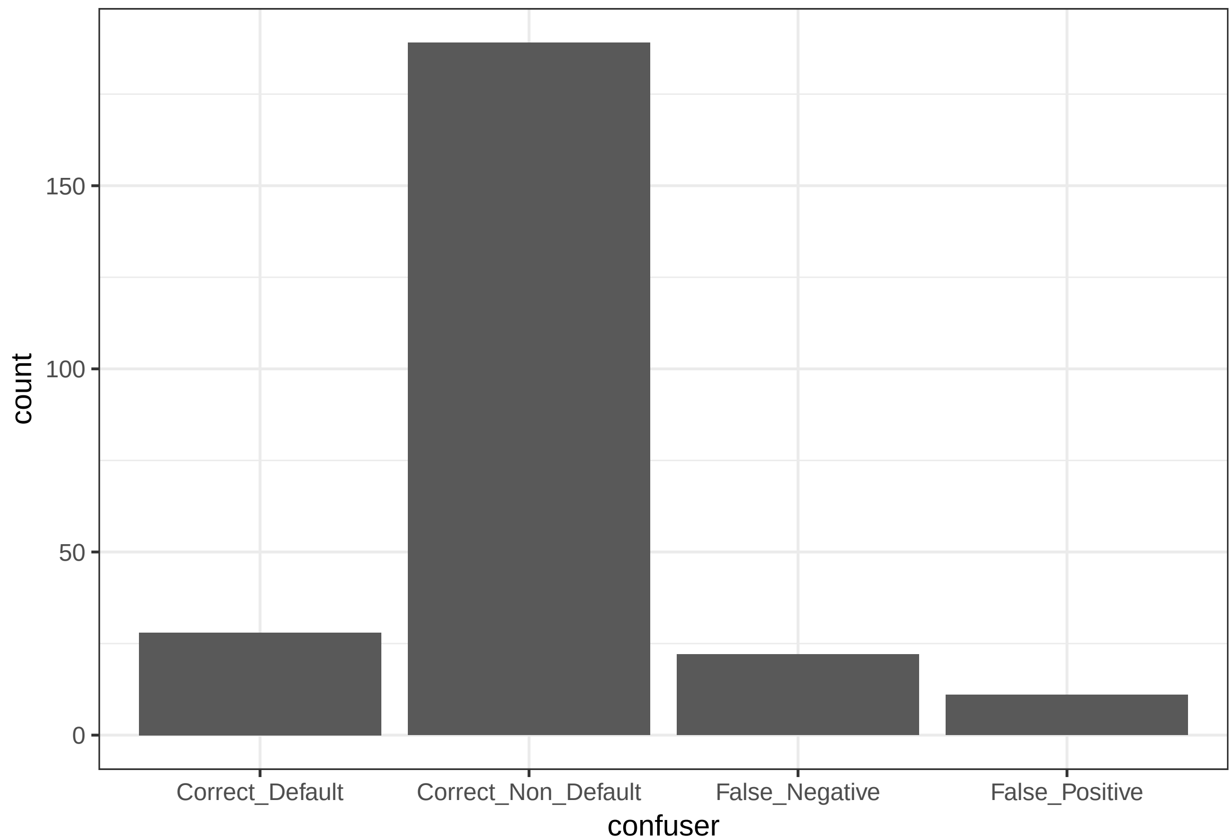
```
GridConfuser <- lapply(Sequences, function(x) {


  dt <- dt_test %>%
    mutate(test = ifelse(prediction >= x, 1, 0)
           , confuser = ifelse(test == 1 & default == 1, "Correct_Default",
                               ifelse(test == 0 & default == 0, "Correct_Non_Default",
                                      ifelse(test == 1 & default == 0, "False_Positive",
                                             ifelse(test == 0 & default == 1, "False_Negative",NA))))
    )

  ConfusionMatrix <- dt %>%
    group_by(confuser) %>%
    summarize(Count = n()) %>%
    t() %>%
    data.frame(., row.names = NULL)

    names(ConfusionMatrix) <- as.character(unlist(ConfusionMatrix[1,]))

    ConfusionMatrix <- ConfusionMatrix[-1, ] %>%
    mutate(Sequence = x)


plotme <- ggplot(dt, aes(x = confuser)) +
  geom_bar() +
  theme_bw()
```

```
output <- list(data = dt
               , plot = plotme
, table = ConfusionMatrix)

})
```

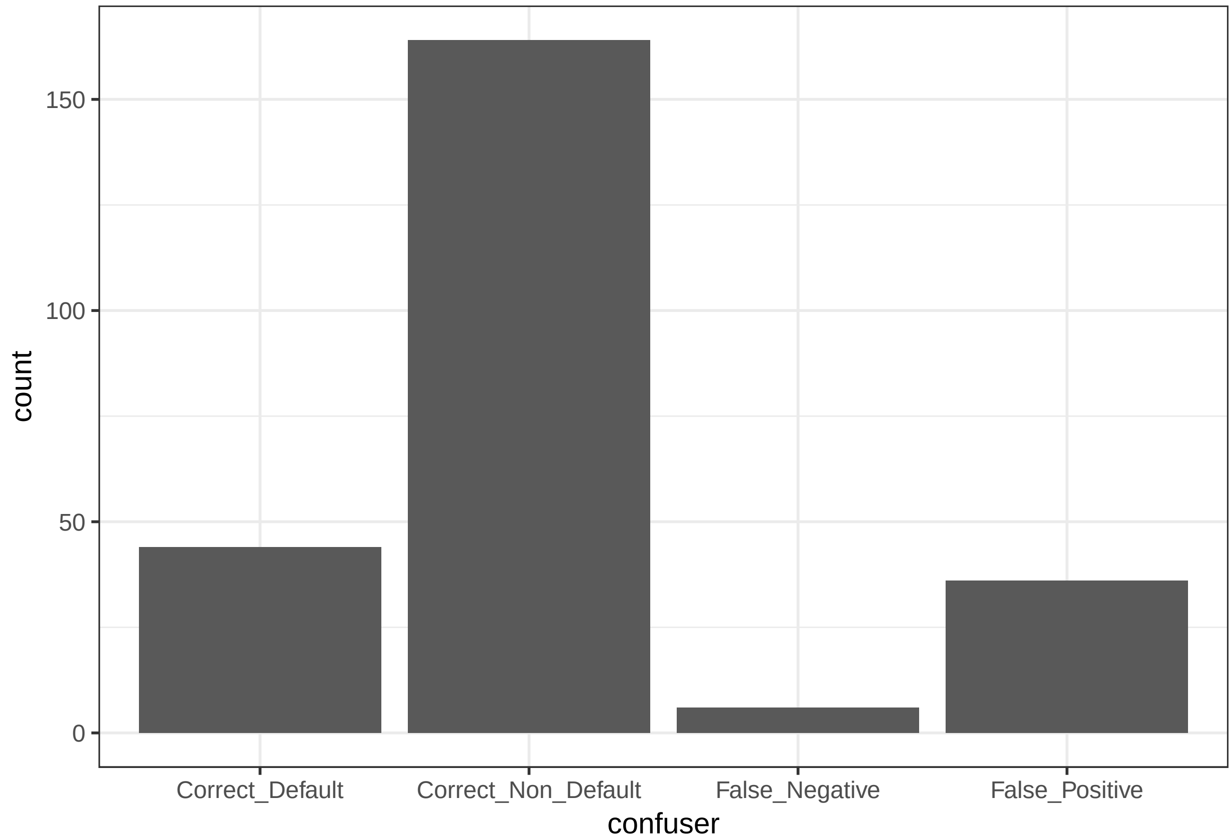
Display Confusion Matrix Bar Plots

Bar Plot outputs from above Confusion can be displayed at any level of the grid search. A manual search can be done of these outputs.

```
GridConfuser[[61]]$plot
```




```
GridConfuser[[16]]$plot
```



Bind All Grid Confusion matrices into a data frame and calculate errors

Bind all the confusion matrices (GridConfuser output) into a data table to compare the accuracy of each based on the model outputs. In the next iteration, I would weight each False_Positive and False_Negative based on its actual impact on the business. In this case, a false negative seems more detrimental than a false positive because a default loan is more costly than not granting a loan that would have not been defaulted on. But… banking is weird so I am sure there is a point where a default loan is a positive if the payor paid long enough.

The "Errors" value calculated is a sum of Type I and Type II errors. They are equally weighted for now, weighting can be done here to more 'smartly' select the best cutline.

```
dt_ConfusionGrid <- rbindlist(lapply(GridConfuser, function(x){x[[3]]}), use.names = T, fill = T) %>%
  tbl_df() %>%
  mutate_all(funs(ifelse(is.na(.), 0, as.numeric(as.character(.))))) %>%
  mutate(Errors = (as.numeric(False_Positive) + as.numeric(False_Negative)) ) %>%
  select(Correct_Default, Correct_Non_Default, False_Positive, False_Negative, Errors, Sequence) %>%
  arrange(Errors)

data.table(dt_ConfusionGrid)
```

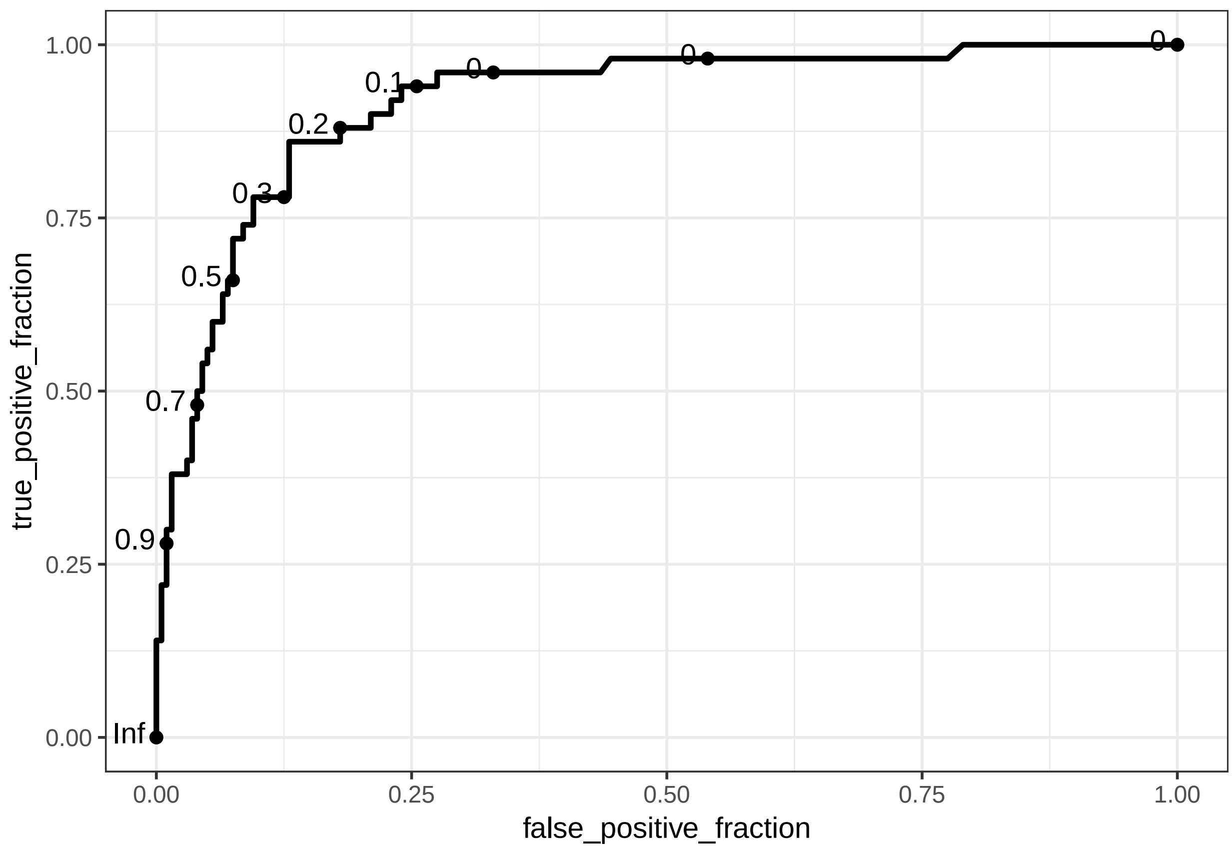
```
##     Correct_Default Correct_Non_Default False_Positive False_Negative
##  1:              39                 181             19             11
##  2:              36                 184             16             14
##  3:              35                 185             15             15
##  4:              39                 180             20             11
##  5:              34                 185             15             16
```

```
## ---
##  97:              48                   142            58              2
##  98:              48                   136            64              2
##  99:              48                   132            68              2
## 100:              48                   129            71              2
## 101:              50                     0           200              0
##      Errors Sequence
##   1:     30     0.39
##   2:     30     0.41
##   3:     30     0.42
##   4:     31     0.38
##   5:     31     0.43
## ---
##  97:     60     0.05
##  98:     66     0.03
##  99:     70     0.02
## 100:     73     0.01
## 101:    200     0.00
```

Select Best ROC cut line based on the above output

For now, the 'Sequence' corresponding to the lowest value of Errors is chosen to be the value used for the final cut line. There may be a few more elegant ways to select this based on the weighting of False_Positive and False_Negative values as mentioned above. This rough cut is easier to explain to an audience.

```r
ggplot(dt_test, aes(d = default, m = prediction)) +
  geom_roc() +
  theme_bw()
```




```r
cutLine <- as.numeric(unlist(dt_ConfusionGrid[1,"Sequence"]))

cutLine
```

```
## [1] 0.39
```

Create Final Test Predictions

Using the above cutline, make a final prediction for each test value.

```r
dt_test <- dt_test %>%
  mutate(finalPrediction = ifelse(prediction >= cutLine, "default", "non_default")
         , default = ifelse(default == 1, "default", "non_default"))

data.table(dt_test)
```

```
##      creditScore houseAge yearsEmploy ccDebt     default  prediction
##   1:         749       13           5   8763 non_default 0.401021361
##   2:         634       16           6   5519 non_default 0.001301979
##   3:         737       30           7   9973     default 0.752586603
##   4:         734       26           8   6517 non_default 0.050652627
##   5:         667       23           5   5741 non_default 0.001161993
##  ---
## 246:         728       20           3   9487 non_default 0.782866001
## 247:         676       39           6   3221 non_default 0.001361457
## 248:         697       23           5   1477 non_default 0.001266424
## 249:         578       17           6   5326 non_default 0.001301979
## 250:         658       20           8   9989     default 0.852065504
```

```
##      finalPrediction
##   1:         default
##   2:     non_default
##   3:         default
##   4:     non_default
##   5:     non_default
##  ---
## 246:         default
## 247:     non_default
## 248:     non_default
## 249:     non_default
## 250:         default
```

Create Final Model Confusion Matrix metrics

Using the above cutline, make a final prediction for each test value.

```r
xtab <- confusionMatrix(table(dt_test$default, dt_test$finalPrediction))

xtab
```

```
## Confusion Matrix and Statistics
##
##
##               default non_default
##   default          39          11
##   non_default      19         181
##
##                Accuracy : 0.88
##                  95% CI : (0.8331, 0.9176)
##     No Information Rate : 0.768
##     P-Value [Acc > NIR] : 0.000005257
##
##                   Kappa : 0.6462
##  Mcnemar's Test P-Value : 0.2012
##
##             Sensitivity : 0.6724
##             Specificity : 0.9427
##          Pos Pred Value : 0.7800
##          Neg Pred Value : 0.9050
##              Prevalence : 0.2320
##          Detection Rate : 0.1560
##    Detection Prevalence : 0.2000
##       Balanced Accuracy : 0.8076
##
##        'Positive' Class : default
##
```

Final Accuracy of model is .88. It's an effective model that can be improved a little more with deep tuning. But for this exercise, I am happy with the outcome.