Fraud Classification

Business Question

Can we predict whether a transaction is fraud?

Setup and Initialization

```
# Setup
knitr::opts_chunk$set(echo = TRUE)
```

Data Import

```
# Import Data
data_original <- read.csv('./Fraud.csv')
```

```
# Copy Data
data_copy <- data_original
```

Data Cleaning and Preprocessing

```
# See Structure of Data and Summary for any issues
str(data_copy)
```

```
## 'data.frame':    6362620 obs. of  11 variables:
##  $ step          : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ type          : chr  "PAYMENT" "PAYMENT" "TRANSFER" "CASH_OUT" ...
##  $ amount        : num  9840 1864 181 181 11668 ...
##  $ nameOrig      : chr  "C1231006815" "C1666544295" "C1305486145" "C840083671" ...
##  $ oldbalanceOrg : num  170136 21249 181 181 41554 ...
##  $ newbalanceOrig: num  160296 19385 0 0 29886 ...
##  $ nameDest      : chr  "M1979787155" "M2044282225" "C553264065" "C38997010" ...
##  $ oldbalanceDest: num  0 0 0 21182 0 ...
##  $ newbalanceDest: num  0 0 0 0 0 ...
##  $ isFraud       : int  0 0 1 1 0 0 0 0 0 0 ...
##  $ isFlaggedFraud: int  0 0 0 0 0 0 0 0 0 0 ...
```

```
summary(data_copy)
```

```
##       step           type               amount          nameOrig
##  Min.   :  1.0   Length:6362620     Min.   :       0   Length:6362620
##  1st Qu.:156.0   Class :character   1st Qu.:   13390   Class :character
##  Median :239.0   Mode  :character   Median :   74872   Mode  :character
##  Mean   :243.4                      Mean   :  179862
##  3rd Qu.:335.0                      3rd Qu.:  208721
##  Max.   :743.0                      Max.   :92445517
##  oldbalanceOrg      newbalanceOrig       nameDest          oldbalanceDest
##  Min.   :       0   Min.   :       0   Length:6362620     Min.   :        0
##  1st Qu.:       0   1st Qu.:       0   Class :character   1st Qu.:        0
##  Median :   14208   Median :       0   Mode  :character   Median :   132706
##  Mean   :  833883   Mean   :  855114                      Mean   :  1100702
##  3rd Qu.:  107315   3rd Qu.:  144258                      3rd Qu.:   943037
##  Max.   :59585040   Max.   :49585040                      Max.   :356015889
##  newbalanceDest         isFraud          isFlaggedFraud
##  Min.   :        0   Min.   :0.000000   Min.   :0.0e+00
##  1st Qu.:        0   1st Qu.:0.000000   1st Qu.:0.0e+00
##  Median :   214661   Median :0.000000   Median :0.0e+00
##  Mean   :  1224996   Mean   :0.001291   Mean   :2.5e-06
##  3rd Qu.:  1111909   3rd Qu.:0.000000   3rd Qu.:0.0e+00
##  Max.   :356179279   Max.   :1.000000   Max.   :1.0e+00
```

```
# Check Number of NA values in data
sum(is.na(data_copy))
```

```
## [1] 0
```

```
# Preprocess Data
data_copy$merchant <- ifelse(substr(data_copy$nameDest, 1, 1) == 'M', TRUE, FALSE)
data_copy$isFraud <- as.logical(data_copy$isFraud)
data_copy$isFlaggedFraud <- NULL
data_copy$nameOrig <- NULL
data_copy$nameDest <- NULL
data_copy$type <- as.integer(as.factor(data_copy$type))
summary(data_copy)
```

```
##       step            type           amount          oldbalanceOrg
##  Min.   :  1.0   Min.   :1.000   Min.   :       0   Min.   :       0
##  1st Qu.:156.0   1st Qu.:2.000   1st Qu.:   13390   1st Qu.:       0
##  Median :239.0   Median :2.000   Median :   74872   Median :   14208
##  Mean   :243.4   Mean   :2.714   Mean   :  179862   Mean   :  833883
##  3rd Qu.:335.0   3rd Qu.:4.000   3rd Qu.:  208721   3rd Qu.:  107315
##  Max.   :743.0   Max.   :5.000   Max.   :92445517   Max.   :59585040
##  newbalanceOrig     oldbalanceDest       newbalanceDest       isFraud
##  Min.   :       0   Min.   :        0   Min.   :        0   Mode :logical
##  1st Qu.:       0   1st Qu.:        0   1st Qu.:        0   FALSE:6354407
##  Median :       0   Median :   132706   Median :   214661   TRUE :8213
##  Mean   :  855114   Mean   :  1100702   Mean   :  1224996
##  3rd Qu.:  144258   3rd Qu.:   943037   3rd Qu.:  1111909
##  Max.   :49585040   Max.   :356015889   Max.   :356179279
##   merchant
##  Mode :logical
##  FALSE:4211125
```

```
##  TRUE :2151495
##
##
##
```

Training and Testing Sets Creation

```r
# Create Training and Testing Sets
set.seed(10)
smp_size <- floor(0.75 * nrow(data_copy))
print(paste("Sample size: ", smp_size))
```

```
## [1] "Sample size:  4771965"
```

```r
set.seed(100)
train_ind <- sample(1:nrow(data_copy), size = smp_size)

train <- data_copy[train_ind, ]
test <- data_copy[-train_ind, ]
```

Avoiding Complexity: Feature (Model) Selection

We want to avoid complexity by tuning the classification model complexity to the classification task. We want to ensure there are not too many features which can lead to overfitting while too few and the model can not learn good rules. The Regularization method will be used here with a Lasso (L1) penalty function.

This is because L1 tends to shrink coefficients to zero whereas Ridge (L2) tends to shrink coefficients evenly. L1 is therefore useful for feature selection, as we can drop any variables associated with coefficients that go to zero.

Ref: https://explained.ai/regularization/L1vsL2.html
Ref: https://www.statology.org/lasso-regression-in-r/

First, we need to define the response variable and all the features as a matrix

```r
# lets define the response variable
y <- train$isFraud

# lets define a matrix of all features except for the columns related to the response variable
all_column_names <- colnames(train)
all_columns_noFraud <- all_column_names[all_column_names != "isFraud"]
x <- data.matrix(train[, all_columns_noFraud])
```

Then, we need to determine the optimal lambda value to use which will be done using cross-validation (CV)

```r
library(glmnet)
```

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

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

```r
library(Matrix)
library(doParallel)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

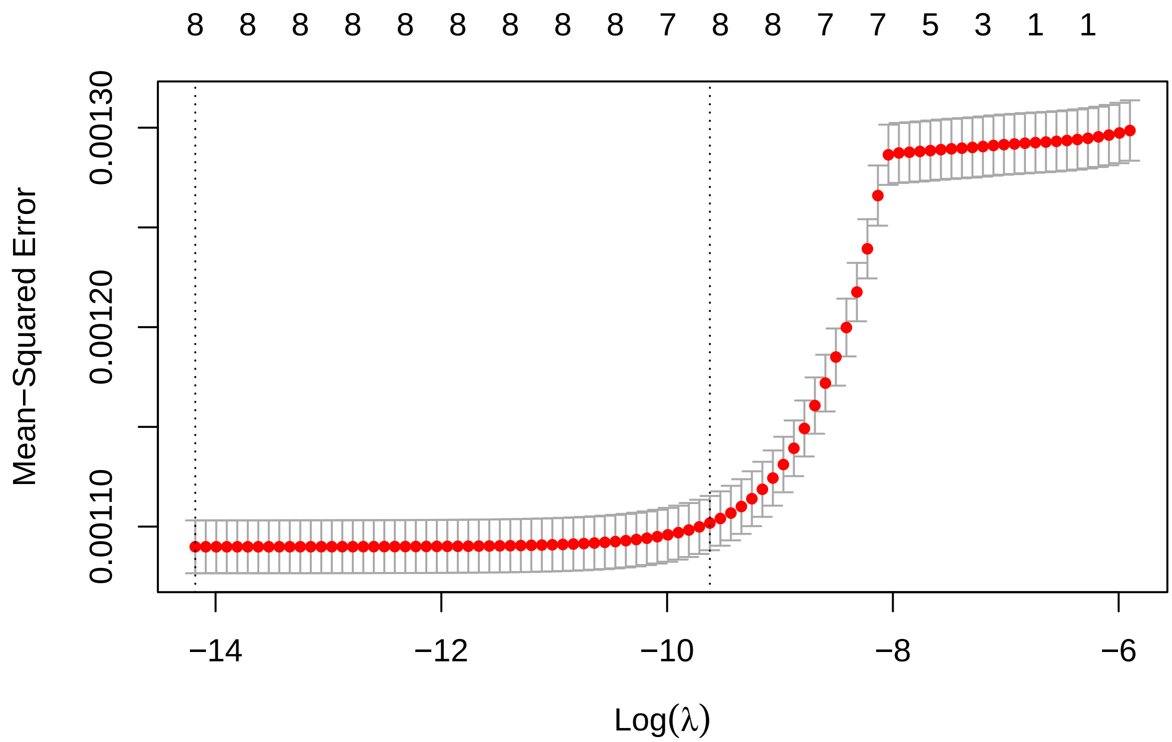
```r
if (file.exists("./cv_model.RData")) {
  # load cv_model
  load("./cv_model.RData")

} else {
  # create cv_model

  num_cores <- 4
  cl <- makeCluster(num_cores)
  registerDoParallel(cl)

  # perform k-fold cross-validation to find optimal lambda value (using the default k=10 folds)
  # use alpha=1 as we want to fit the lasso regression model
  cv_model <- cv.glmnet(x, y, alpha = 1, parallel = TRUE)

  stopCluster(cl)

  # save cv_model
  save(cv_model, file="./cv_model.RData")

}

# determine optimal lambda value that minimizes test mean squared error (MSE)
optimal_lambda <- cv_model$lambda.min
print("The Optimal Lambda is:")
```

```
## [1] "The Optimal Lambda is:"
```

```r
optimal_lambda
```

```
## [1] 6.946304e-07
```

```r
# produce plot of test mean squared error (MSE) by lambda value
plot(cv_model)
```



Finally, we can use the optimal lambda value to determine coefficient estimates for each variable

```
if (file.exists("./regularized_model.RData")) {
  # load regularized_model
  load("./regularized_model.RData")

} else {
  # create regularized_model

  num_cores <- 4
  cl <- makeCluster(num_cores)
  registerDoParallel(cl)


  # determine coefficient estimates for each variable of the regularized model
  regularized_model <- glmnet(x, y, alpha = 1, lambda = optimal_lambda)

  stopCluster(cl)

  # save regularized_model
  save(regularized_model, file="./regularized_model.RData")

}

# create coefficient matrix
coef_matrix <- coef(regularized_model)
```

```
coef_matrix
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                           s0
## (Intercept)     6.833142e-03
## step            6.519135e-06
## type           -2.556640e-03
## amount          1.335611e-08
## oldbalanceOrg   1.010215e-07
## newbalanceOrig -1.003548e-07
## oldbalanceDest  8.990134e-09
## newbalanceDest -9.162193e-09
## merchant        9.398258e-04
```

Let's construct a formula with only the significant features.

```
# drop the Intercept column as it is not needed
row_index <- which(rownames(coef_matrix) == "(Intercept)")
coef_matrix_noIntercept <- coef_matrix[-row_index, , drop = FALSE]

# store only the feature names that don't have a coefficient of 0
ideal_features <- rownames(coef_matrix_noIntercept)[coef_matrix_noIntercept[, 1] != 0][]
ideal_features
```

```
## [1] "step"           "type"           "amount"         "oldbalanceOrg"
## [5] "newbalanceOrig" "oldbalanceDest" "newbalanceDest" "merchant"
```

```
# construct the formula as a string
formula_string <- paste("isFraud ~", paste(ideal_features, collapse = " + "))
formula_string
```

```
## [1] "isFraud ~ step + type + amount + oldbalanceOrg + newbalanceOrig + oldbalanceDest + newbalanceDes
```

```
# convert the formula string to a formula object
formula_object <- as.formula(formula_string)
print("Final relation:")
```

```
## [1] "Final relation:"
```

```
print(formula_object)
```

```
## isFraud ~ step + type + amount + oldbalanceOrg + newbalanceOrig +
##     oldbalanceDest + newbalanceDest + merchant
```

Model 1: Logistic Regression

For the first model, we generate a logistic regression classifier for the formula defined above. We train the model on the previously defined training set and evaluate this model using the testing set. We use a cut value of 0.5 since this is a widely accepted value and fits our case.

```r
if (file.exists("./logReg.RData")) {
  # load classifier
  load("./logReg.RData")
} else {
  # create classifier
  cls <- glm(formula_object, family='binomial',data=train)

  # save classifier
  save(cls, file="./logReg.RData")
}
```

Training and Testing Error Assessment

Now, let's evaluate the training and testing error for the Logistic Regression Classifier

```r
# Set the cut to 0.5
cut=0.5

# Calculate Training error
yhat_tr = (predict(cls,train,type="response")>cut)
tr.err = mean(train$isFraud != yhat_tr)


# Calculate testing error
yhat_te = (predict(cls,test,type="response")>cut)
te.err = mean(test$isFraud != yhat_te)


print("Training Error")
```

```
## [1] "Training Error"
```

```r
print(tr.err)
```

```
## [1] 0.0007340791
```

```r
print("Testing Error")
```

```
## [1] "Testing Error"
```

```r
print(te.err)
```

```
## [1] 0.0007141712
```

In this case, we see a very low error rate for the training set, which means the model is not underfit. In addition, the testing error is very similar to the training error, meaning that the model is not overfit.

Model 2: Decision Tree Classifier Model

Classifier Generation

We can generate a decision tree classifier trained on the training set and evaluated on the testing set. Since complexity parameter (cp) controls the improvement threshold to make a split, we can compare different thresholds.

```r
# define and train each of the classifiers
library(rpart)
```

```r
if (file.exists("./tree1.RData")) {
  # load tree1
  load("./tree1.RData")

} else {
  # create tree1
  tree1 <- rpart(formula_object,method="class", data=train, cp=0.01)

  # save tree1
  save(tree1, file="./tree1.RData")

}
```

```r
if (file.exists("./tree2.RData")) {
  # load tree2
  load("./tree2.RData")

} else {
  # create tree2
  tree2 <- rpart(formula_object,method="class", data=train, cp=0.001)

  # save tree2
  save(tree2, file="./tree2.RData")

}
```

```r
if (file.exists("./tree3.RData")) {
  # load tree3
  load("./tree3.RData")

} else {
  # create tree3
  tree3 <- rpart(formula_object,method="class", data=train, cp=0.00001)

  # save tree3
  save(tree3, file="./tree3.RData")

}
```

Training and Testing Error Assessment

Now, lets compare the training and testing error for cp=0.01, cp=0.001, and cp=0.00001

```r
treeErr <- function(tree, test_dataset, train_dataset)
{
  test_pred = predict(tree,test_dataset,type="class")
  test_err = mean(test_dataset$isFraud != test_pred)


  train_pred = predict(tree,train_dataset,type="class")
  train_err = mean(train_dataset$isFraud != train_pred)

  test_acc = 1 - test_err

  conf_matrix <- table(Actual = test_dataset$isFraud, Predicted = test_pred)

  result_df <- data.frame(
    Metric = c("Train Error", "Test Error", "Accuracy"),
    Value = c(train_err, test_err, test_acc)
  )

  return(list(result_df, Confusion_Matrix = conf_matrix))

}
```

```r
# Errors for tree1
tree1Err <- treeErr(tree1, test, train)
print(tree1Err)
```

```
## [[1]]
##        Metric        Value
## 1 Train Error 0.0004293829
## 2  Test Error 0.0004086367
## 3    Accuracy 0.9995913633
##
## $Confusion_Matrix
##        Predicted
## Actual    FALSE    TRUE
##   FALSE 1588582      65
##   TRUE      585    1423
```

```r
# Errors for tree2
tree2Err <- treeErr(tree2, test, train)
print(tree2Err)
```

```
## [[1]]
##        Metric        Value
## 1 Train Error 0.0003105639
## 2  Test Error 0.0003287954
## 3    Accuracy 0.9996712046
##
## $Confusion_Matrix
##        Predicted
## Actual    FALSE    TRUE
##   FALSE 1588574      73
##   TRUE      450    1558
```

```r
# Errors for tree3
tree3Err <- treeErr(tree3, test, train)
print(tree3Err)
```

```
## [[1]]
##        Metric        Value
## 1 Train Error 0.0001921640
## 2  Test Error 0.0003143359
## 3    Accuracy 0.9996856641
##
## $Confusion_Matrix
##        Predicted
## Actual    FALSE    TRUE
##   FALSE 1588438     209
##   TRUE      291    1717
```

It is clear that for cp=0.01 the tree is underfit as it does not split and defaults to classifying all transactions as fraud. For cp=0.001 the training error and testing error improve but only slightly. For cp=0.000001, the tree is much more complex and we see evidence of overfitting.

Model 3: Support Vector Machine (SVM) Model

Since SVM takes a very long time to run for large dataset, we will use the data reduction strategy to randomly select a subset of our data for training and testing

```r
library(e1071)
library(caret)
```

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

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

```r
# Training / testing data for SVM
set.seed(100) # Ensure reproducibility
sample_size <- floor(0.1 * nrow(data_copy)) # Calculate 10% of the data size
sample_indices <- sample(1:nrow(data_copy), size = sample_size) # Get random sample indices
data_subset <- data_copy[sample_indices, ] # Extract 10% of the data

set.seed(100)
train_size <- floor(0.7 * nrow(data_subset)) # Calculate 70% of the subset size for training
train_indices <- sample(1:nrow(data_subset), size = train_size) # Get random sample indices for trainin

train_svm <- data_subset[train_indices, ] # Create the training set from the subset
test_svm <- data_subset[-train_indices, ] # Create the testing set from the subset


train_svm$isFraud <- as.factor(train_svm$isFraud) #The target variable should be a factor
test_svm$isFraud <- as.factor(test_svm$isFraud)

if (file.exists("./svm.RData")) {
  # load svm
```

```r
  load("./svm.RData")

} else {
  # create svm

  # Define the number of cores to use
num_cores <- 4

# Register parallel backend
cl <- makeCluster(num_cores)
registerDoParallel(cl)
  svm_model <- svm(formula = formula_object,
                   data = train,
                   kernel = "radial",
                   cost = 1,
                   scale = TRUE) # Automatic feature scaling

  # save svm
  save(svm_model, file="./svm.RData")

}



# Predictions on the training set
train_predictions <- predict(svm_model, newdata = train_svm)

# Predictions on the testing set
test_predictions <- predict(svm_model, newdata = test_svm)

# Calculating Training Error
training_error <- mean(train_predictions != train_svm$isFraud)
print(paste("Training Error: ", training_error))
```

```
## [1] "Training Error:  0.000749916364118074"
```

```r
# Calculating Testing Error
testing_error <- mean(test_predictions != test_svm$isFraud)
print(paste("Testing Error: ", testing_error))
```

```
## [1] "Testing Error:  0.000722971096872888"
```

```r
# Confusion Matrix


# Now, compute confusion matrix
conf_matrix <- table(test_predictions, test_svm$isFraud)
print(conf_matrix)
```

```
##
## test_predictions  FALSE   TRUE
##            FALSE 190639    138
##            TRUE       0    102
```

Model 4: K-Nearest Neighbours (KNN) Model

```r
#KNN

library(FNN)  # Faster k-nearest neighbor algorithm implementation
library(doParallel)  # For parallel computation if needed

# Example parallelization setup
# Adjust the number of cores as per your machine's configuration

if(file.exists("./knn_model.rds")){
  test_pred  <- readRDS("knn_model.rds")
} else {

  cores <- 4
  registerDoParallel(cores=cores)

  class_labels <- train$isFraud  # Assuming "isFraud" is the target variable
  train_features <- train[, -which(names(train) == "isFraud")]  # Exclude the target variable
  test_features <- test[, -which(names(test) == "isFraud")]  # Exclude the target variable

  # Example of using FNN library which might be faster
  test_pred <- knn(train_features, test = test_features, cl = class_labels, k = 10)

  # Stop parallel computation
  stopImplicitCluster()

}
```

```r
actual <- test$isFraud
cm <- table(actual,test_pred)
cm
```

```
##        test_pred
## actual     FALSE    TRUE
##   FALSE 1588513     134
##   TRUE      712    1296
```

```r
accuracy <- sum(diag(cm))/length(actual)
sprintf("KNN Accuracy: %.10f%%", accuracy*100)
```

```
## [1] "KNN Accuracy: 99.9468143626%"
```

Model 5: Neural Net (NN) Model

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
```

```
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.3     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x purrr::accumulate() masks foreach::accumulate()
## x tidyr::expand()     masks Matrix::expand()
## x dplyr::filter()     masks stats::filter()
## x dplyr::lag()        masks stats::lag()
## x purrr::lift()       masks caret::lift()
## x tidyr::pack()       masks Matrix::pack()
## x tidyr::unpack()     masks Matrix::unpack()
## x purrr::when()       masks foreach::when()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(keras)
library(tensorflow)
```

```
##
## Attaching package: 'tensorflow'
##
## The following object is masked from 'package:caret':
##
##     train
```

```
checkpoint_path <- "training_2/cp.ckpt"
checkpoint_dir <- fs::path_dir(checkpoint_path)


# Define predictor variables
predictor_vars <- c("step", "amount", "oldbalanceOrg", "newbalanceOrig", "oldbalanceDest", "newbalanceDe
formula <- as.formula(paste("isFraud ~", paste(predictor_vars, collapse = "+")))


if (file.exists("new_model.hdf5")) {

  model <- load_model_hdf5("new_model.hdf5")
  message("Loaded weights from checkpoint.")

} else {

  model <- keras_model_sequential() %>%
  layer_dense(units = 4, activation = "relu", input_shape = length(predictor_vars)) %>%
  layer_dense(units = 2, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")


# Compile the model
model %>% compile(
  loss = "binary_crossentropy",
  optimizer =  tf$keras$optimizers$legacy$Adam(),
  metrics = "accuracy"
)
```

```r
cp_callback <- callback_model_checkpoint(
  filepath = checkpoint_path,
  save_weights_only = TRUE,
  verbose = 1
)


# Train the model
history <- model %>% fit(
  x = as.matrix(train[predictor_vars]),
  y = train$isFraud,
  epochs = 10,
  batch_size = 100,
  class_weights= c(1,1000),
  validation_split = 0.3,  # Split data for validation
  callbacks = list(cp_callback, early_stopping) # Pass callback to training
)

model %>% save_model_hdf5("./new_model.hdf5")

}
```

```
## Loaded weights from checkpoint.
```

```r
# Evaluate the model
model %>% evaluate(
  x = as.matrix(test[predictor_vars]),
  y = test$isFraud
)
```

```
## 49708/49708 - 198s - loss: 108.7588 - accuracy: 0.9991 - 198s/epoch - 4ms/step
```

```
##        loss    accuracy
## 108.7587509   0.9990646
```

```r
# Predict probabilities using the model
predicted_probs <- model %>% predict(as.matrix(test[predictor_vars]))
```

```
## 49708/49708 - 77s - 77s/epoch - 2ms/step
```

```r
# Convert probabilities to binary classes based on a threshold of 0.5
predicted_classes <- ifelse(predicted_probs > 0.5, TRUE, FALSE)

# Create a data frame to store actual and predicted values
results <- data.frame(Actual = test$isFraud, Predicted = predicted_classes)

# Create confusion matrix
conf_matrix <- table(Actual = results$Actual, Predicted = results$Predicted)

# Print the confusion matrix
print(conf_matrix)
```

```
##        Predicted
## Actual    FALSE    TRUE
##   FALSE 1588366     281
##   TRUE     1207     801
```