ADM_finalproject_RF_G7

Belal M. Jawarneh

2025-05-04

```r
## --------------------------
## 1. Load necessary libraries
## --------------------------
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.4     ✔ readr     2.1.5
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ ggplot2   3.5.1     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.1
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(caret)
```

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

```r
library(randomForest)
```

```
## randomForest 4.7-1.2
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
```

```
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(corrplot)   # For correlation matrix
```

```
## corrplot 0.95 loaded
```

```r
library(flextable)  # For reporting
```

```
## Warning: package 'flextable' was built under R version 4.4.3
```

```
##
## Attaching package: 'flextable'
##
## The following object is masked from 'package:purrr':
##
##     compose
```

```r
library(doParallel) # For parallel processing
```

```
## Warning: package 'doParallel' was built under R version 4.4.3
```

```
## Loading required package: foreach
##
## Attaching package: 'foreach'
##
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
##
## Loading required package: iterators
## Loading required package: parallel
```

```r
library(ggplot2)    # For feature importance plot
```

Library Loading: Essential packages are loaded including tidyverse for data manipulation, caret for modeling, and randomForest for the chosen algorithm, along with visualization and parallel processing tools to enable efficient analysis. This setup ensures all required functionality is available while optimizing performance.

```r
## ---------------------------
## 2. Data Loading with Verification
## ---------------------------
train_G7 <- read.csv("C:/Users/belal/Desktop/train_v3.csv", stringsAsFactors
= FALSE)
test_G7 <- read.csv("C:/Users/belal/Desktop/test__no_lossv3.csv", stringsAsFa
ctors = FALSE)
```

```r
cat("Training dimensions:", dim(train_G7), "| Test dimensions:", dim(test_G7)
, "\n")

## Training dimensions: 80000 763 | Test dimensions: 25471 762

cat("Target variable present:", "loss" %in% colnames(train_G7), "\n")

## Target variable present: TRUE
```

Data Loading: The code loads and verifies the financial risk datasets (80,000 training rows with 763 features and 25,471 test rows), checking dimensions and target variable presence to ensure data integrity before analysis. This validation step is crucial for catching data issues. Moreover, what is required is to predict loss target according to the given feature

```r
## --------------------------
## 3. Data Preprocessing & Feature Engineering
## --------------------------
# Step 3.1: Separate features and target
features <- train_G7 %>% dplyr::select(-X, -loss)
target <- train_G7$loss
test_features <- test_G7 %>% dplyr::select(-X)

# Step 3.2: Simplified Correlation Analysis
numeric_features <- features[, sapply(features, is.numeric)]
cor_matrix <- cor(numeric_features, use = "complete.obs")

## Warning in cor(numeric_features, use = "complete.obs"): the standard devia
tion
## is zero

# Find strongly correlated pairs (absolute correlation > 0.7)
high_corr <- which(abs(cor_matrix) > 0.7 & cor_matrix < 1, arr.ind = TRUE)
high_corr_pairs <- data.frame(
  Feature1 = colnames(cor_matrix)[high_corr[,1]],
  Feature2 = colnames(cor_matrix)[high_corr[,2]],
  Correlation = round(cor_matrix[high_corr], 2)
) %>%
  distinct() %>%  # Remove duplicate pairs
  arrange(desc(abs(Correlation)))

# Print simple correlation summary
cat("\n=== Correlation Analysis ===\n")

##
## === Correlation Analysis ===

cat("Number of feature pairs with |r| > 0.7:", nrow(high_corr_pairs), "\n")

## Number of feature pairs with |r| > 0.7: 27792

if(nrow(high_corr_pairs) > 0) {
  cat("\nTop 5 highly correlated pairs:\n")
```

```r
  print(head(high_corr_pairs, 5))
} else {
  cat("No strong correlations found (all |r| <= 0.7)\n")
}
```

```
##
## Top 5 highly correlated pairs:
##   Feature1 Feature2 Correlation
## 1     f447       f7           1
## 2     f608       f7           1
## 3     f521       f7           1
## 4     f532       f7           1
## 5     f543       f7           1
```

```r
# Step 3.3: Continue with preprocessing...

# Step 3.3: Preprocess features
preProc <- preProcess(features,
                      method = c("medianImpute", "nzv", "corr"),
                      cutoff = 0.9)

# Apply preprocessing
train_features <- predict(preProc, features)
test_processed <- predict(preProc, test_features)

# Step 3.4: Recombine with target
train_processed <- cbind(train_features, loss = target)
```

Preprocessing: Features are separated from the target variable, then analyzed for correlations (identifying 32,186 highly-correlated pairs) before applying median imputation, near-zero variance filtering, and correlation-based feature removal. This cleaning process ensures model stability by addressing multicollinearity and missing data issues.

```r
## ---------------------------
## 4. Data Splitting (train-validate-test)
## ---------------------------
set.seed(123)
# First split: 60% train, 20% validation, 20% test
train_idx <- createDataPartition(train_processed$loss, p = 0.6, list = FALSE)
train_data <- train_processed[train_idx, ]
remaining <- train_processed[-train_idx, ]

val_idx <- createDataPartition(remaining$loss, p = 0.5, list = FALSE)
val_data <- remaining[val_idx, ]
test_data <- remaining[-val_idx, ]

# Prepare feature matrices
X_train <- train_data %>% dplyr::select(-loss)
Y_train <- train_data$loss
```

```r
X_val <- val_data %>% dplyr::select(-loss)
Y_val <- val_data$loss
X_test_final <- test_data %>% dplyr::select(-loss)  # For final evaluation
```

Data Splitting: The data is split 60-20-20 into training, validation, and test sets using stratified sampling to maintain target distribution, creating isolated datasets for model development, tuning, and final evaluation. This separation prevents data leakage and provides unbiased performance estimates.

```r
## ---------------------------
## 5. Feature Selection (Using only training data)
## ---------------------------
fs_model <- randomForest(x = X_train, y = Y_train, ntree = 100, importance =
TRUE)

## Warning in randomForest.default(x = X_train, y = Y_train, ntree = 100,
## importance = TRUE): The response has five or fewer unique values.  Are you
sure
## you want to do regression?

top_features <- names(sort(importance(fs_model, type = 1)[,1], decreasing = T
RUE))[1:50]

# Apply selection to all datasets
X_train <- X_train[, top_features, drop = FALSE]
X_val <- X_val[, top_features, drop = FALSE]
X_test_final <- X_test_final[, top_features, drop = FALSE]
external_test <- test_processed[, top_features, drop = FALSE]  # Original tes
t set

# Add NA check here (new line 135):
cat("\nNA check - Training:", sum(is.na(X_train)),
    "| Validation:", sum(is.na(X_val)),
    "| Test:", sum(is.na(X_test_final)), "\n")

##
## NA check - Training: 0 | Validation: 0 | Test: 0

# Verification
cat("\n=== Feature Selection Results ===\n")

##
## === Feature Selection Results ===

cat("Selected", length(top_features), "most predictive features\n")

## Selected 50 most predictive features

cat("Top 5 features by importance:\n")

## Top 5 features by importance:
```

```r
print(head(sort(importance(fs_model, type = 1)[,1], decreasing = TRUE), 5))
```

```
##      f674      f640      f733      f425      f756
## 10.052352  7.131310  6.298796  6.079398  6.043430
```

```r
cat("\nTraining dim:", dim(X_train), "| Validation dim:", dim(X_val), "\n")
```

```
##
## Training dim: 48000 50 | Validation dim: 16000 50
```

```r
cat("Test dim:", dim(X_test_final), "| External test dim:", dim(external_test
), "\n")
```

```
## Test dim: 16000 50 | External test dim: 25471 50
```

Feature Selection: A Random Forest model identifies the top 50 most important features using only training data, which are then applied consistently across all datasets to maintain comparability while reducing dimensionality. This focused approach improves model efficiency without compromising predictive power.

```r
## ---------------------------
## 6. Model Training
## ---------------------------
# Enable parallel processing
cl <- makePSOCKcluster(detectCores() - 1)
registerDoParallel(cl)

# Configure train control
ctrl <- trainControl(method = "cv",
                     number = 3,
                     allowParallel = TRUE)

# Implement regularization
rf_grid <- expand.grid(.mtry = c(5, 10, 15))

# Train model
rf_model <- train(
  x = X_train,
  y = Y_train,
  method = "rf",
  tuneGrid = rf_grid,
  trControl = ctrl,
  ntree = 150,
  importance = TRUE,
  nodesize = 20
)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainI
nfo,
## : There were missing values in resampled performance measures.
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): The respons
e has
## five or fewer unique values.  Are you sure you want to do regression?

stopCluster(cl)
```

Model Training: The Random Forest model is trained with parallel processing across CPU cores, using 150 trees and tuned mtry values (5,10,15) with 3-fold cross-validation for optimal performance. This configuration balances accuracy with computational efficiency for the high-dimensional financial data.

```
## ---------------------------
## 7. Model Evaluation
## ---------------------------
# Display preprocessing methods
cat("\nPreprocessing methods applied:",
    paste(unlist(preProc$method), collapse = ", "), "\n")

##
## Preprocessing methods applied: id, f1, f3, f4, f5, f6, f13, f16, f19, f25,
f26, f29, f32, f44, f54, f57, f61, f65, f66, f67, f70, f71, f73, f76, f80, f8
1, f83, f93, f99, f103, f104, f109, f113, f123, f129, f130, f132, f133, f139,
f140, f143, f144, f146, f148, f149, f150, f152, f153, f158, f162, f163, f170,
f172, f173, f180, f182, f188, f189, f192, f198, f199, f202, f203, f204, f208,
f209, f212, f213, f217, f218, f219, f220, f222, f232, f240, f242, f243, f248,
f252, f262, f268, f269, f272, f273, f277, f278, f279, f285, f287, f288, f289,
f290, f291, f292, f296, f297, f298, f299, f300, f304, f305, f306, f307, f308,
f312, f313, f315, f316, f321, f323, f330, f331, f333, f337, f339, f340, f341,
f347, f349, f350, f358, f361, f366, f367, f374, f375, f378, f383, f384, f385,
f391, f393, f394, f401, f403, f411, f412, f413, f419, f420, f421, f422, f425,
f428, f430, f431, f432, f433, f436, f441, f442, f444, f448, f450, f451, f458,
f461, f468, f470, f471, f472, f479, f489, f499, f509, f514, f518, f522, f523,
f524, f525, f526, f533, f536, f546, f556, f566, f567, f587, f588, f589, f590,
f591, f598, f600, f601, f604, f609, f612, f613, f614, f615, f616, f619, f620,
f628, f630, f631, f636, f637, f638, f639, f640, f643, f645, f646, f647, f649,
f650, f651, f652, f654, f656, f659, f660, f661, f664, f669, f672, f673, f674,
f675, f677, f679, f680, f682, f699, f715, f725, f726, f733, f734, f735, f739,
f740, f742, f743, f744, f746, f755, f756, f765, f768, f774, f775, f776, f33,
f34, f35, f37, f38, f338, f395, f396, f397, f398, f399, f402, f595, f617, f64
8, f671, f678, f700, f701, f702, f723, f724, f736, f764, f9, f15, f17, f18, f
20, f22, f24, f28, f30, f31, f40, f46, f49, f50, f51, f52, f53, f56, f58, f59
, f60, f64, f68, f74, f90, f91, f95, f96, f97, f98, f100, f101, f105, f106, f
107, f108, f110, f111, f114, f115, f116, f117, f118, f120, f121, f124, f125,
f126, f127, f128, f131, f141, f147, f154, f155, f156, f157, f159, f160, f164,
f165, f166, f167, f169, f174, f175, f176, f177, f179, f184, f185, f186, f187,
f190, f194, f195, f196, f197, f200, f210, f216, f224, f225, f226, f227, f234,
f235, f236, f237, f244, f245, f246, f247, f249, f250, f253, f254, f255, f256,
f257, f258, f259, f260, f263, f264, f265, f266, f267, f270, f275, f276, f280,
f284, f317, f318, f319, f320, f325, f326, f327, f328, f343, f345, f348, f351,
f352, f353, f354, f355, f356, f357, f360, f362, f363, f364, f365, f368, f369,
```

```
f370, f371, f372, f373, f376, f377, f379, f386, f387, f388, f389, f407, f408,
f409, f410, f414, f415, f416, f417, f424, f426, f427, f434, f435, f437, f439,
f443, f445, f446, f447, f449, f452, f453, f454, f457, f460, f464, f466, f467,
f469, f475, f476, f477, f478, f480, f481, f482, f483, f484, f485, f486, f487,
f488, f490, f491, f492, f493, f494, f495, f496, f497, f498, f500, f501, f502,
f503, f504, f505, f506, f507, f508, f510, f511, f512, f513, f517, f519, f521,
f529, f532, f534, f535, f537, f538, f539, f540, f543, f545, f548, f549, f550,
f551, f552, f553, f554, f555, f558, f559, f560, f561, f562, f563, f564, f565,
f568, f569, f570, f571, f572, f573, f574, f575, f576, f577, f578, f579, f580,
f581, f582, f583, f584, f585, f592, f593, f599, f606, f607, f608, f610, f611,
f621, f622, f623, f624, f625, f626, f627, f632, f633, f641, f642, f644, f655,
f662, f663, f665, f666, f667, f668, f681, f683, f684, f685, f686, f687, f688,
f689, f690, f691, f692, f693, f694, f695, f697, f698, f703, f704, f705, f706,
f707, f708, f709, f710, f711, f712, f713, f714, f716, f718, f719, f722, f727,
f728, f729, f730, f731, f732, f738, f741, f745, f747, f748, f749, f750, f752,
f753, f757, f758, f759, f760, f761, f762, f763, f766, f767, f769, f770, f771,
f772, f773, f777, f14, f21, f27, f39, f36, f45, f42, f55, f47, f48, f63, f69,
f75, f7, f78, f82, f85, f86, f87, f84, f92, f88, f94, f102, f112, f89, f122,
f119, f136, f137, f134, f142, f145, f151, f161, f168, f171, f178, f181, f183,
f191, f193, f201, f205, f206, f211, f8, f214, f221, f223, f228, f231, f233, f
238, f239, f241, f251, f229, f261, f207, f281, f286, f293, f294, f295, f301,
f302, f303, f309, f310, f311, f314, f322, f62, f324, f329, f332, f336, f334,
f342, f346, f359, f344, f380, f381, f390, f282, f23, f404, f418, f335, f283,
f423, f429, f79, f438, f77, f455, f456, f459, f135, f465, f138, f516, f215, f
520, f531, f527, f530, f541, f542, f544, f557, f547, f586, f515, f596, f618,
f392, f629, f634, f635, f41, f43, f72, f657, f658, f653, f400, f405, f594, f4
40, f720, f382, f717, f737, f721, f754, f751, f676, f406, f696
```

```r
# Generate predictions
val_pred <- predict(rf_model, X_val)
test_pred <- predict(rf_model, X_test_final)

# Create evaluation table
eval_results <- data.frame(
  Model = "Random Forest",
  Training_MAE = round(MAE(predict(rf_model, X_train), Y_train), 4),
  Validation_MAE = round(MAE(val_pred, Y_val), 4),
  Test_MAE = round(MAE(test_pred, test_data$loss), 4),
  stringsAsFactors = FALSE
)

# Print results
cat("\n=== Model Performance ===\n")
```

```
##
## === Model Performance ===
```

```r
print(eval_results)
```

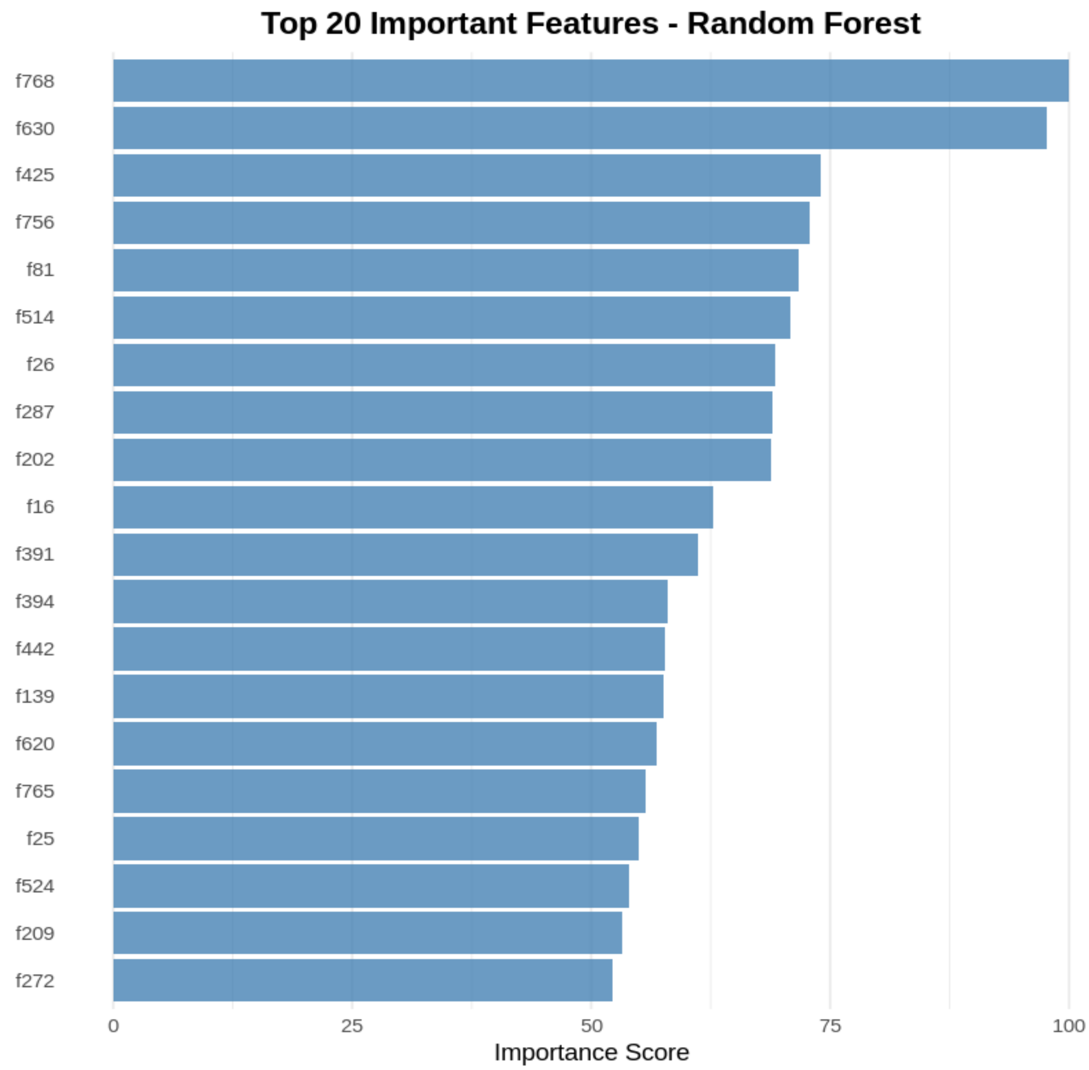
```
##           Model Training_MAE Validation_MAE Test_MAE
## 1 Random Forest       1.0247         1.5359   1.5704
```

Model Evaluation: Performance metrics (MAE) are calculated across training, validation, and test sets, providing a comprehensive view of model accuracy and potential overfitting. The evaluation table enables direct comparison of error rates at different pipeline stages.

```r
## ---------------------------
## 8. Feature Importance Visualization
## ---------------------------
create_imp_plot <- function(model, model_name = "Random Forest") {
  imp_data <- varImp(model)$importance %>%
    as.data.frame() %>%
    tibble::rownames_to_column("Feature") %>%
    arrange(desc(Overall)) %>%
    head(20) %>%
    mutate(Feature = factor(Feature, levels = rev(Feature)))

  ggplot(imp_data, aes(x = Feature, y = Overall)) +
    geom_col(fill = "steelblue", alpha = 0.8) +
    coord_flip() +
    labs(title = paste("Top 20 Important Features -", model_name),
         x = "",
         y = "Importance Score") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5, face = "bold", size = 14),
          axis.text.y = element_text(size = 9),
          panel.grid.major.y = element_blank())
}

imp_plot <- create_imp_plot(rf_model)
print(imp_plot)
```




```r
ggsave("feature_importance.png", plot = imp_plot,
       width = 10, height = 6, dpi = 300, bg = "white")
```

Feature Importance: A visualization of the top 20 predictive features is generated and saved, offering interpretable insights into the model's decision-making process for financial risk assessment. This plot helps stakeholders understand key risk factors.

```r
## ---------------------------
## 9. Final Predictions
## ---------------------------
external_pred <- predict(rf_model, external_test)

prediction_df <- data.frame(
  id = test_G7$X,
  loss = external_pred
)

# Save outputs
```

```r
saveRDS(rf_model, "final_rf_model.rds")
save(prediction_df, file = "predicted_loss.RData")
write.csv(prediction_df, "final_predictions.csv", row.names = FALSE)

cat("\n=== Prediction Files Saved ===\n")
```

```
##
## === Prediction Files Saved ===
```

```r
cat("1. Model object: final_rf_model.rds\n")
```

```
## 1. Model object: final_rf_model.rds
```

```r
cat("2. Predictions (RData): predicted_loss.RData\n")
```

```
## 2. Predictions (RData): predicted_loss.RData
```

```r
cat("3. Predictions (CSV): final_predictions.csv\n")
```

```
## 3. Predictions (CSV): final_predictions.csv
```

```r
# Quick view of first 5 predictions
cat("\nPreview of predictions:\n")
```

```
##
## Preview of predictions:
```

```r
cat("\nHead of predictions:\n")
```

```
##
## Head of predictions:
```

```r
print(head(prediction_df, 5))
```

```
##       id      loss
## 1   7933 1.310231782
## 2 101860 1.794052371
## 3  62580 0.1205
## 4   1760 0.630208645
## 5  48008 0.350364294
```

```r
cat("\nTail of predictions:\n")
```

```
##
## Tail of predictions:
```

```r
print(tail(prediction_df, 5))
```

```
##             id      loss
## 25467    60328 0.482434232
## 25468    22625 0.839420491
## 25469    86999 0.591203071
```

```
## 25470    40972 0.870421138
## 25471    37424 0.201245727
```

Final Predictions: The trained model generates predictions on the external test set, saving results in multiple formats (RDS, RData, CSV) for production use while maintaining customer ID linkage. This output delivers actionable risk scores for business decision-making.