STAT5003_14_Credit_Card_Analysis

STAT5003 group project 14

2023-10-24

OVERVIEW

- **Problem statement:** Commercial banks need to continually observe the impact of customer's work and life stability on credit card defaults, which is costly to maintain (Li et. al., 2019). Traditional regression functions on demographic variables and credit card features yield limited explanatory power compared to other factors such as attitude variables and personality variables (Wang et. al, 2011). Previous attempts at applying machine learning to classify card applicants into different credit limits has achieved acceptable results (82% predictive accuracy of credit card defaulters with neural networkst) (Leong et. all., 2019).
- **Goal:** The goal of this exercise is to apply machine learning methods to predict potential credit defaulters based on a set of data available at point of application. The dataset can be accessed here.
- **The dataset is split into two files**: **credit_record** (3 features x 1+ million observations) which tracks customer credit performance overtime and **application_record** (18 features x 438+ thousand observations) which contains collected information at the initial credit card application such as ID, gender, occupation, number of children, marital status, etc. The total data points Both files are connected by the feature **ID**.

```
# load dataset(s)
# source: https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction

#get and set path to current working directory
path = getwd()
setwd(path)

#read csv
application_record <- read.csv('application_record.csv', header = TRUE)
credit_record <- read.csv('credit_record.csv', header = TRUE)

# rename features for code simplicity
application_record <- application_record %>%
  rename (gender = CODE_GENDER,
          car_flag = FLAG_OWN_CAR,
          realty_flag = FLAG_OWN_REALTY,
          children_count= CNT_CHILDREN,
          income_total = AMT_INCOME_TOTAL,
          income_type = NAME_INCOME_TYPE,
          education_type = NAME_EDUCATION_TYPE,
          family_status = NAME_FAMILY_STATUS,
          housing_type = NAME_HOUSING_TYPE,
          days_age = DAYS_BIRTH,
```

```
          days_employed = DAYS_EMPLOYED,
          mobilephone_flag=FLAG_MOBIL,
          workphone_flag = FLAG_WORK_PHONE,
          phone_flag = FLAG_PHONE,
          email_flag = FLAG_EMAIL,
          occupation_type = OCCUPATION_TYPE,
          familymembers_count = CNT_FAM_MEMBERS
          )

credit_record <- credit_record %>%
  rename (months_balance = MONTHS_BALANCE,
          credit_status = STATUS
          )

#print dimensions
#cat("Dimensions of application_record: Rows =", dim(application_record)[1], "Columns =", dim(applicati
#cat("Dimensions of credit_record: Rows =", dim(credit_record)[1], "Columns =", dim(credit_record)[2],
```

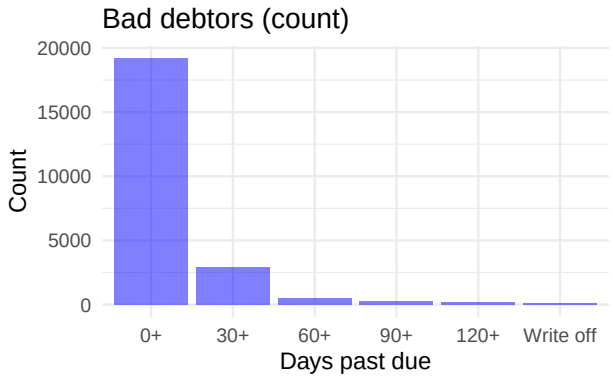
FEATURE ENGINEERING

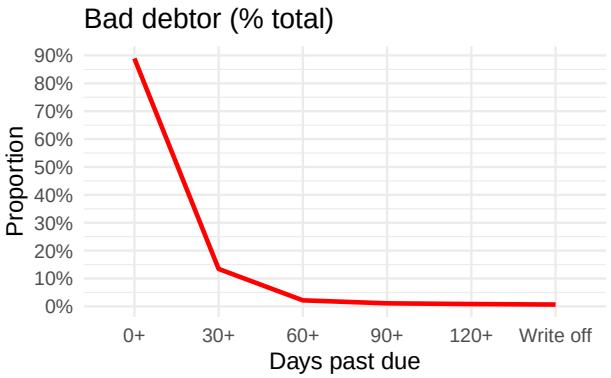
- **Data analysis and cleaning:** the dataset was screened for nulls, blanks, and duplicates and further trimmed to only include IDs with 20+ months banking history. As displayed below, 88.9% of debtors have more than 0 days of delinquency which could be an honest mistake and not representative of their ability to service their payments. Thus we defined **30+ days of delinquency as our target feature, denoted as class 1. The resulting dataset contained 21,575 customers of which 2,895 customers (c.13%) are in class 1**.

```
# check and remove NAs --> no NAs found
na_rows_application <- application_record[!complete.cases(application_record), ]
na_rows_credit <- credit_record[!complete.cases(credit_record), ]
nrow_app <- nrow(na_rows_application) # no rows with NA
nrow_cred <- nrow(na_rows_credit) # no rows with NA

#cat('Number of rows with null values in application_record:',nrow_app,'rows','\n')
#cat('Number of rows with null values in credit_record:',nrow_cred,'rows','\n')

#OCCUPATION_TYPE has blank values (''), fill with 'Pensioner' and 'Unknown' accordingly

pensioner_count <- 0
unknown_count <- 0

for (i in 1:length(application_record$occupation_type)) {
  if (application_record$occupation_type[i] == "" && application_record$income_type[i] == "Pensioner") {
    application_record$occupation_type[i] <- "Pensioner"
    pensioner_count <- pensioner_count + 1
  }
}

for (i in 1:length(application_record$occupation_type)) {
  if (application_record$occupation_type[i] == "" ) {
    application_record$occupation_type[i] <- "Unknown"
    unknown_count <- unknown_count + 1
```

```
  }
}

#cat("Number of blank occupation_type filled with 'Pensioner' (inferred from 'income_type'):", pensione
#cat("Number of blank occupation_type filled with 'Unknown':", unknown_count,"cells", "\n")

# check and remove duplicates in application_record
duplicate_rows <- application_record %>% group_by(ID) %>%filter(n() > 1)
nrow_dup <- nrow(duplicate_rows)
application_record <- application_record %>% anti_join(duplicate_rows, by = "ID")

#cat("Number of duplicate IDs dropped:", nrow_dup,"rows", "\n")

# Trim both dataset by common IDs
common_ids <- intersect(application_record$ID, credit_record$ID)
filtered_application_records <- application_record %>% filter(ID %in% common_ids) #36k x 18 fetures
filtered_credit_records <- credit_record %>% filter(ID %in% common_ids) #777k x 3 features

# We can only judge credit quality if customer has already banked with us for some time
# setting a cut off for customers that has banked with us for more than 'threshold' months

threshold <- -20 #this is a negative number since 0 = now

filtered_credit_records_months <- filtered_credit_records  %>% filter(months_balance <= threshold)
length_ids <- unique(filtered_credit_records_months$ID)

#cat("There are", length(length_ids),"unique IDs with more than",-threshold,"months of transaction hist

# trim both datasets to to only include those that meets the threshold history
length_filtered_application_records <- filtered_application_records %>%
                                       filter(ID %in% length_ids)
length_filtered_credit_records <- filtered_credit_records %>%
                                       filter(ID %in% length_ids)

#dimension of filtered records

#cat("Dimension of trimmed application_record:", dim(length_filtered_application_records)[1], "rows x",

#cat("Dimension of trimmed credit_record:", dim(length_filtered_credit_records)[1], "rows x",dim(length


# currently, STATUS is filled with the following values:
    # X : No loan for the month
    # C : paid off for the month
    # 0 : 0-29 days past due
    # 1 : 30-59 days past due
    # 2 : 60-89 days past due
    # 3 : 90-119 days past due
    # 4 : 120-149 days past due
    # 5 : bad debt / write off

filters <- list(
  c('0','1','2','3','4','5'),
```

```
  c('1','2','3','4','5'),
  c('2','3','4','5'),
  c('3','4','5'),
  c('4','5'),
  c('5')
)

category_counts <- numeric(length(filters))
category_ids <- vector("list", length(filters))

for (i in seq_along(filters)) {
  filtered_records <- length_filtered_credit_records %>% filter(credit_status %in% filters[[i]])
  category_ids[[i]] <- unique(filtered_records$ID)
  category_counts[i] <- length(category_ids[[i]])
}

df <- data.frame(
  debt_category = c('0+',
                    '30+',
                    '60+',
                    '90+',
                    '120+',
                    'Write off'),
  category_count = category_counts
)

# Create a factor with the desired order for debt_category
df$debt_category <- factor(df$debt_category, levels = df$debt_category)

#Calculate proportion to total
df$proportion <- df$category_count / length(length_ids)

# print(df) #unhash to run

bar_plot <- ggplot(df, aes(x = debt_category)) +
  geom_bar(aes(y = category_count), stat = 'identity', fill = 'blue', alpha = 0.5) +
  labs(title = 'Bad debtors (count)', x = 'Days past due', y = 'Count') +
  theme_minimal()

line_plot <- ggplot(df, aes(x = debt_category)) +
  geom_line(aes(y = proportion, group = 1), color = 'red', size = 1) +
  scale_y_continuous(breaks = seq(0, 1, by = 0.1), labels = percent) +  # Set breaks at 5% increments a
  labs(title = 'Bad debtor (% total)', x = 'Days past due', y = 'Proportion') +
  theme_minimal()


grid.arrange(bar_plot, line_plot, ncol = 2)
```



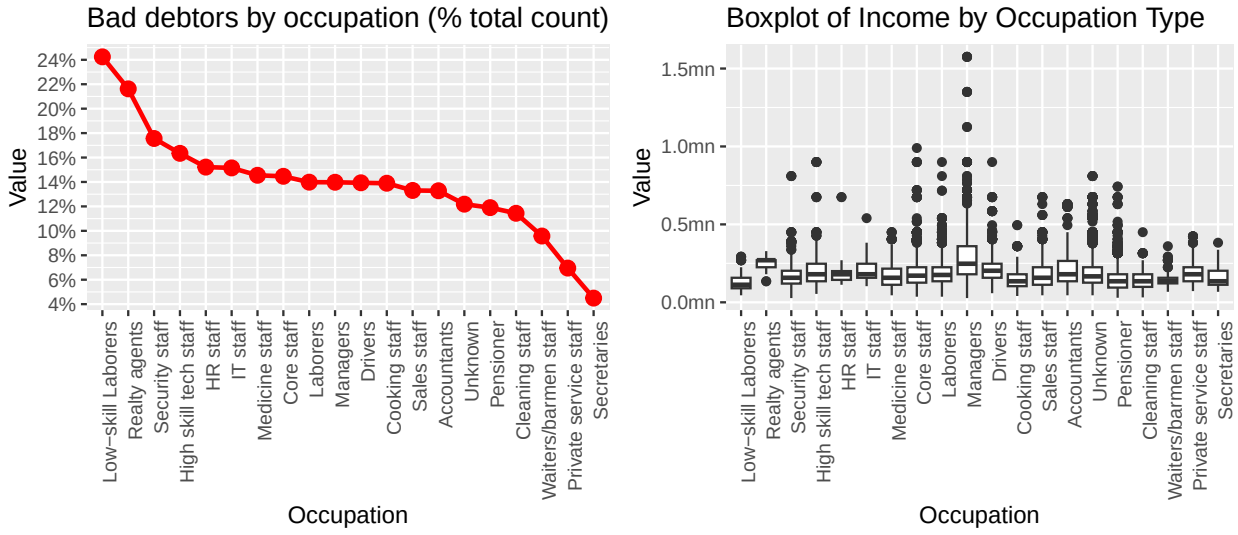



```r
# start a new copy of the trimmed application dataset
application_clean <- length_filtered_application_records

# add relationship_length into dataset
relationship_length_values <- credit_record %>% group_by(ID) %>%
  summarize(relationship_length = -min(months_balance) +1)

application_clean <- application_clean %>% left_join(relationship_length_values, by = "ID")

# add column debt_quality
application_clean$debt_quality <- 0

# depending on how hard we want to make this, we can choose how many flags we want, below we choose 2:
application_clean$debt_quality[application_clean$ID %in% category_ids[[2]]] <- 1
#application_clean$debt_quality[application_clean$ID %in% category_ids[[3]]] <- 2
#application_clean$debt_quality[application_clean$ID %in% category_ids[[4]]] <- 3
#application_clean$debt_quality[application_clean$ID %in% category_ids[[5]]] <- 4
#application_clean$debt_quality[application_clean$ID %in% category_ids[[6]]] <- 5

# Create a table with 'debt_quality' and 'count'
debt_quality_counts <- table(application_clean$debt_quality)
debt_quality_table <- data.frame(debt_quality = names(debt_quality_counts), count = as.vector(debt_quali

# Print the table
#debt_quality_table
```

- Delinquency rate by occupation type shown below. We can see that delinquency rate does not necessarily increase with lower wages and some occupation has large variability in total income.

```r
# Count bad debt by occupation type, descending order
debt_count <- application_clean %>%
  filter(debt_quality >0) %>%
  group_by(occupation_type) %>%
  summarise(debt_count=n()) %>%
  arrange(desc(debt_count))

occupation_count <- application_clean %>%
  group_by(occupation_type) %>%
  summarise(occupation_count=n())
```

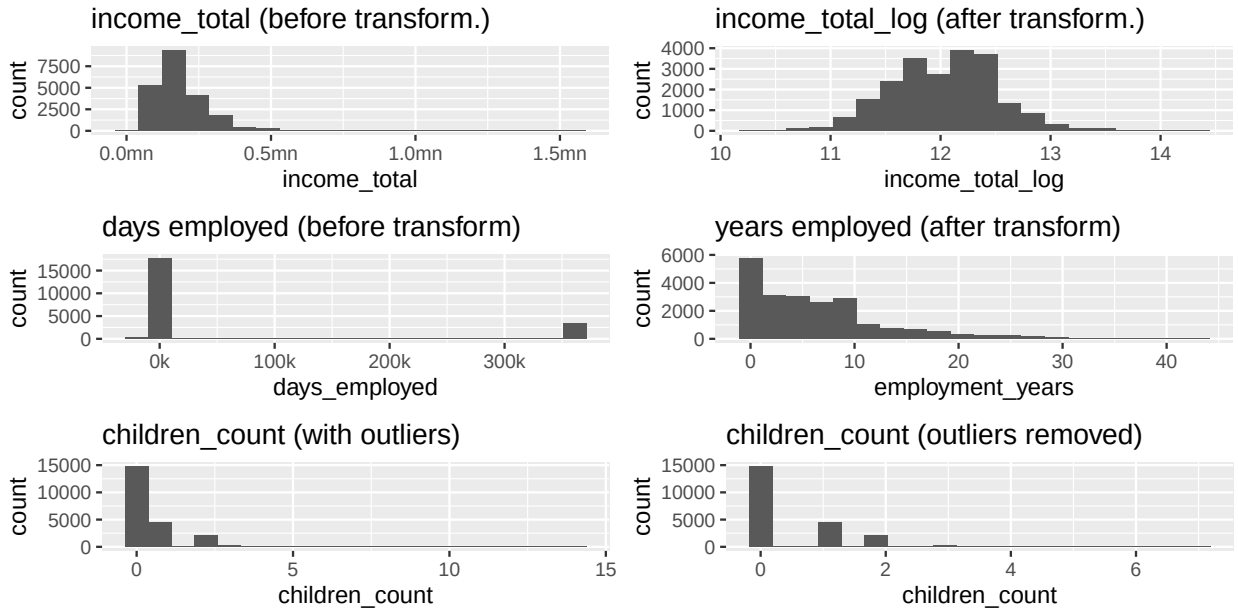
```r
line.result <- left_join(occupation_count, debt_count, by = "occupation_type") %>%
  mutate(ratio = debt_count / occupation_count) %>%  arrange(desc(ratio))

line.result$occupation_type <- factor(line.result$occupation_type, levels = line.result$occupation_type

#line_plot <- ggplot(line.result, aes(x = occupation_type)) +
#  geom_line(aes(y = ratio, group = 1), color = 'red', size = 1) +
#  geom_text_repel(aes(y = ratio, label = percent(ratio, accuracy = 0.1)), nudge_x = 0.0, nudge_y = 0.0
#  labs(title = 'Bad debtors by occupation (% total count)', x = 'Occupation', y = 'Value') +
#  theme(axis.text.x = element_text(angle = 90, hjust = 1))

line_plot <- ggplot(line.result, aes(x = occupation_type)) +
  geom_line(aes(y = ratio, group = 1), color = 'red', size = 1) +
  geom_point(aes(y = ratio), color = 'red', size = 3) +  # Add dots for each data point
  scale_y_continuous(breaks = seq(0, 1, by = 0.02),  labels = percent_format(scale = 100)) +     # Add t
  labs(title = 'Bad debtors by occupation (% total count)', x = 'Occupation', y = 'Value') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))


#Boxplot, x-axis matches line plot
box_plot <- ggplot(application_clean, aes(x = factor(occupation_type, levels = levels(line.result$occupa
  geom_boxplot() +
  labs(title = 'Boxplot of Income by Occupation Type', x = 'Occupation', y = 'Value') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_y_continuous(labels = scales::number_format(scale = 1e-6, suffix = "mn"))

#print them out
#print(line_plot)
#print(box_plot)
grid.arrange(line_plot, box_plot, ncol = 2)
```



- Selected feature distributions shown below to show the impact of data cleaning/transformation.

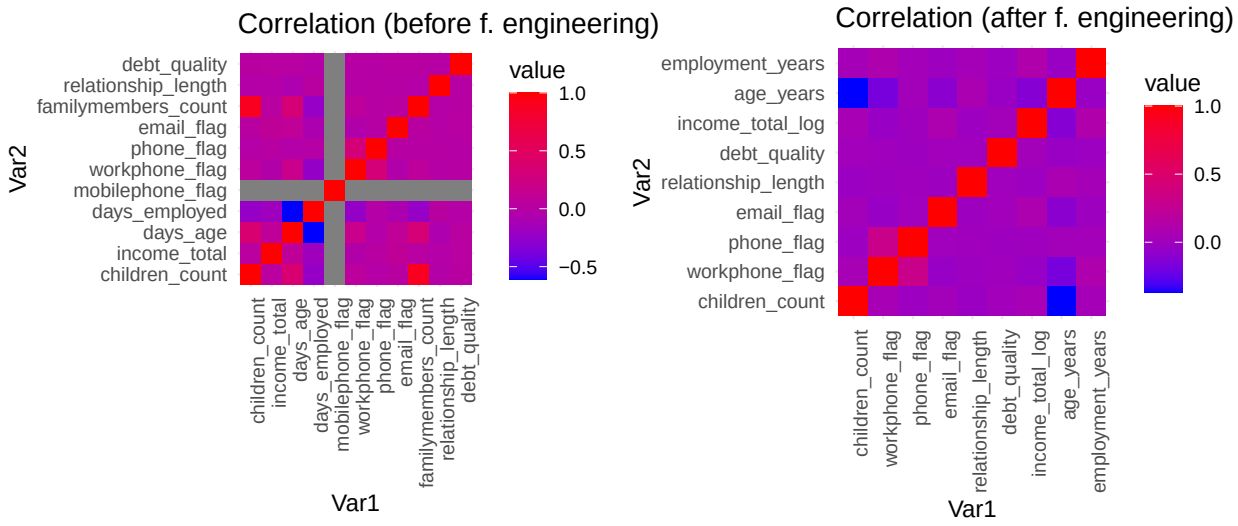
```r
# Plot 1: correlation before cleaning
application_data_numerical <- application_clean %>% select_if(is.numeric)
application_data_numerical$ID <- NULL

correlation_matrix <- cor(application_data_numerical)

corr_1_plot <- ggplot(data = melt(correlation_matrix), aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient(low = "blue", high = "red") +
  theme_minimal() +
  labs(title = 'Correlation (before f. engineering)')+
  coord_fixed(ratio = 1) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

#add column income_total_log due to long tail of income_total
application_clean$income_total_log <- log(application_clean$income_total)

# Plot 3 and 4: income_total and income_total_log
bin_count <- 20

p_income_total <-ggplot(application_clean, aes(x = income_total)) +
    geom_histogram(bins = bin_count) +
    labs(title = 'income_total (before transform.)') +
  scale_x_continuous(labels = scales::unit_format(scale = 1e-6, suffix = "mn"))

p_income_total_log<-ggplot(application_clean, aes(x = income_total_log)) +
    geom_histogram(bins = bin_count) +
    labs(title = 'income_total_log (after transform.)')

#transform days_age into age_years which is now a positive integer
application_clean$age_years = as.integer(round(-application_clean$days_age / 365))

#transform days_employed into employment_years which is now a positive integer, set minimum to 0
application_clean$employment_years = as.integer(round(-application_clean$days_employed / 365))
application_clean$employment_years[application_clean$employment_years < 0] <- 0

# Plot 5 and 6: days_employed and employment_years
p_days_employed <-ggplot(application_clean, aes(x = days_employed)) +
    geom_histogram(bins = bin_count) +
    labs(title = 'days employed (before transform)') +
  scale_x_continuous(labels = scales::unit_format(scale = 1e-3, suffix = "k"))

p_employment_years<-ggplot(application_clean, aes(x = employment_years)) +
    geom_histogram(bins = bin_count) +
    labs(title = 'years employed (after transform)')


#drop familymembers_count as it highly correlates with children_count + spouse (spouse if married)
application_clean$familymembers_count <- NULL

#drop income_total
application_clean$income_total <- NULL
```

```r
#mobilephone_flag only has 1 value = 1, drop
application_clean$mobilephone_flag <- NULL

# drop days_age
application_clean$days_age <- NULL

# drop days_employed
application_clean$days_employed <- NULL

# drop ID
application_clean$ID <- NULL

# Plot 7 and 8: children_count before and after outliers removal
p_children_1 <- ggplot(application_data_numerical, aes(x = children_count)) +
    geom_histogram(bins = bin_count) +
    labs(title = 'children_count (with outliers)')

# remove outliers in children_count
application_clean <- subset(application_clean, children_count <= 10) # 2 rows removed
application_data_numerical <- application_clean %>% select_if(is.numeric)
application_data_numerical$ID <- NULL

p_children_2 <- ggplot(application_data_numerical, aes(x = children_count)) +
        geom_histogram(bins = bin_count) +
        labs(title = 'children_count (outliers removed)')



#Plot 2: correlation after cleaning
correlation_matrix <- cor(application_data_numerical)

corr_2_plot <- ggplot(data = melt(correlation_matrix), aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient(low = "blue", high = "red") +
  theme_minimal() +
  labs(title = 'Correlation (after f. engineering)')+
  coord_fixed(ratio = 1) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

#feature engineering plots
grid.arrange(p_income_total,
             p_income_total_log,
             p_days_employed,
             p_employment_years,
             p_children_1,
             p_children_2,
             ncol = 2)
```



- Correlation plots shown below to show the impact of data cleaning/transformation. Improvement in correlation between features can be seen by fewer red boxes on the right.

```
layout_matrix <- rbind(c(1, 2))
grid.arrange(corr_1_plot,
             corr_2_plot,
             layout_matrix = layout_matrix)
```



PRE-PROCESSING

Pre-processing used for this analysis are described below:

- **Min-max scaling:** applied to numerical data (excluding the target) to remove bias from large values.
- **One-hot-encode:** applied to categorical data (excluding the target) to create binary representation of each categorical value. This is done as some machine learning models require numerical input.
- **Synthetic Minority Over-sampling Technique (SMOTE):** applied to the training set to prevent bias towards majority class due to data imbalance. SMOTE works by creating synthetic data points between randomly-chosen k-nearest neighbor at random distances in the feature space. In practice, SMOTE should only be used on training data to prevent leakage.
- **Up-sampling:** applied to the training set to address significant imbalances between classes. upSample operates by increasing the number of instances of the minority class through duplication of random instances.
- **Principal Component Analysis (PCA)**: applied to the training set to reduce data dimensionality used in Support Vector Machine (SVM). As discussed below, SVM works by building a decision boundary/hyperplane that best separates features. This process can be overly complicated on overly complex dataset.

```
# PRE PROCESSING FUNCTIONS

# FUNCTION: minmax scaler, to be used on numerical data in the below function

minmax_scale <- function(x) {
  (x - min(x, na.rm = TRUE)) / (max(x, na.rm = TRUE) - min(x, na.rm = TRUE))
}

# FUNCTION: apply minmax to numerical data and one hot encode to categorical data.

preprocess_data <- function(dataframe, target_col) {
  #split dataframe into 3 parts for processing
  target_data <- dataframe %>% select({{ target_col }})
  dataframe <- dataframe %>% select(-{{ target_col }}) #take target_data out so it's not accidentally p
  numerical_data <- dataframe %>% select_if(is.numeric)
  categorical_data <- dataframe %>% select_if(is.character)

  #minmax on numerical_data
  numerical_cols <- names(numerical_data)
  scaled_numerical_data <- numerical_data

  for (col_name in numerical_cols) {
    scaled_col <- minmax_scale(numerical_data[[col_name]])
    scaled_numerical_data[[col_name]] <- scaled_col
  }

  #one hot encode on categorical_data
  formula <- dummyVars(~ ., data = categorical_data)
  encoded_data <- predict(formula, newdata = categorical_data)

  #transform target_col as.factor, needed for random forest
  target_data[[target_col]] <- as.factor(target_data[[target_col]])

  #bind the three dataframes together
  merged_data <- cbind(scaled_numerical_data, encoded_data, target_data)

  return(merged_data)
  }
```

```r
#FUNCTION: rename column names (SMOTE seems to have issues with spaces and other special chars)
rename_columns <- function(dataframe, replacements) {
  for (replacement in replacements) {
    for (col in names(dataframe)) {
      new_col <- gsub(replacement$old, replacement$new, col)
      names(dataframe)[names(dataframe) == col] <- new_col
    }
  }
  return(dataframe)
}

# FUNCTION: apply SMOTE and set output as.factor to allow for randomforest

apply_smote <- function(dataframe, target_col){
  set.seed(1)
  #target_col_data <- dataframe[,target_col]

  smote_output <- SMOTE(dataframe[, -which(names(dataframe) == target_col)],
                        dataframe[target_col])

  smote_data <- smote_output$data
  smote_data[[target_col]] <- as.factor(smote_data$class)
  smote_data <- smote_data[, -which(names(smote_data) == "class")]
  #smote_data <- cbind(smote_data, target_col_data)
  return (smote_data)
}
```

CLASSIFICATION METHODS

Five algorithms are used in this exercise and discussed below. Please see testing scores in the 'RESULTS' section.

```r
# create empty dataframe to store results

# Specify column names
col_names <- c("RF", "LogRes", "LDA", "AdaBoost", "SVM")

# Specify row names (index)
row_names <- c("accuracy","precision","sensitivity","specificity","balanced.accuracy","f1.score" )

# Create an empty data frame with specified column and row names
overall.results.df <- data.frame(matrix(NA, nrow = length(row_names), ncol = length(col_names)))
rownames(overall.results.df) <- row_names
colnames(overall.results.df) <- col_names


#how do assign values: row col
# overall.results.df['accuracy','random_forest'] <- 0.85
```

1. Random Forest

- **Description:** Random forest is an ensemble method which combines multiple decision trees to create predictions. Random forest is known for its robustness against overfitting as it takes random features in each of the individual decision trees and takes an average/vote of these trees for a combined result.
- **Design:** (i) Data pre-processing: min-max scaling, one-hot-encoding and SMOTE were applied to the dataset. (ii) Hyperparameter tuning: grid search using train and validation sets were applied on **'ntree'** (number of trees) and **'mtry'** (number of features considered for splitting) to identify the model with the best **f1-score**. (iii) Rationale for metric selection: f1-score gives a balance between precision (minimizing false positives) and recall (minimizing false negatives). (iv) Key feature identification: the top-10 most important features were identified to understand the drivers of bad debt.
- **The best model parameters by f1-score shown below:**

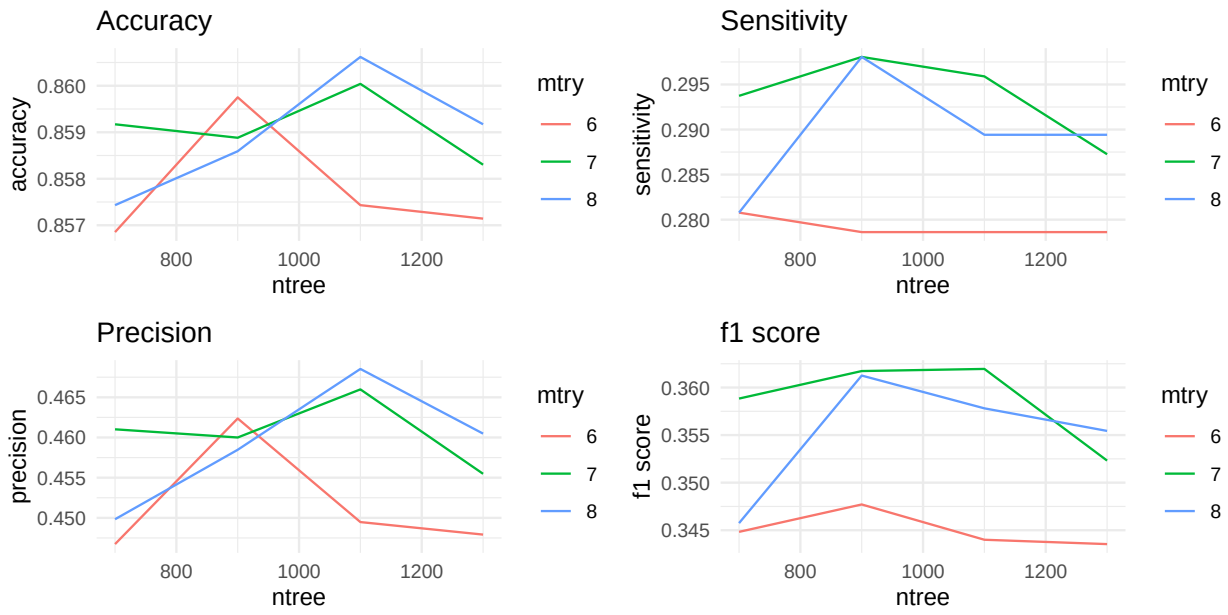
```
#PRE-PROCESSING

# 1. Apply minmax and one hot encode to data
processed_data <- preprocess_data(application_clean,target_col='debt_quality')

# 2. Rename columns since it triggers error on SMOTE
replacements <- list(
  list(old = " ", new = "_"),
  list(old = "/", new = "_"),
  list(old = "-", new = "_")
)

processed_data <- rename_columns(processed_data,replacements)

#3. Apply train test split, createDataPartition automatically stratifies classes
set.seed(1)
inTrain <- createDataPartition(processed_data[['debt_quality']],p=0.8)[[1]]
processed_data.train.full <- processed_data[inTrain,]
processed_data.test <- processed_data[-inTrain,]
actual.test <- processed_data.test$debt_quality

set.seed(1)
inTrain <- createDataPartition(processed_data.train.full[['debt_quality']],p=0.8)[[1]]
processed_data.train <- processed_data.train.full[inTrain,]
processed_data.val <- processed_data.train.full[-inTrain,]
actual.val <- processed_data.val$debt_quality

#apply SMOTE to training data only
smote_processed_data <- apply_smote(processed_data.train,target_col='debt_quality')

# Model and grid search

#initiate results grid
rf.results.df <- data.frame(ntree = integer(),
                            mtry= integer(),
                            accuracy = numeric(),
                            precision = numeric(),
                            sensitivity = numeric(),
                            specificity = numeric(),
                            balanced.accuracy = numeric(),
```

```
                        f1.score = numeric())

#hyperparameter grid
ntree_vals <- c(700,900,1100,1300)
mtry_vals <- c(6,7,8)

#grid search loop
for (ntree_val in ntree_vals){
  for (mtry_val in mtry_vals){
    #model and val
    rf.model <- randomForest(debt_quality~., data = smote_processed_data, ntree=ntree_val, mtry=mtry_val
    rf.val <- predict(rf.model, processed_data.val[, -which(names(processed_data.val) == 'debt_quality'
    rf.cm <- confusionMatrix(rf.val, actual.val, positive="1")

    #add result to dataframe
    rf.results.df <- rbind(rf.results.df,data.frame(ntree = ntree_val, mtry = mtry_val,
                                                    accuracy = rf.cm$overall['Accuracy'],
                                                    precision = rf.cm$byClass['Precision'],
                                                    sensitivity = rf.cm$byClass['Sensitivity'],
                                                    specificity = rf.cm$byClass['Specificity'],
                                                    balanced.accuracy = (rf.cm$byClass['Sensitivity'] + rf.cm
                                                    f1.score = 2 * rf.cm$byClass['Precision'] * rf.cm$byClass
  }
}

#remove row names
rownames(rf.results.df) <- NULL

#rf.results.df

# Find index of max sensitivity and get values of the associated ntree and mtry
max_row_index <- which.max(rf.results.df$f1.score)
ntree_max <- rf.results.df$ntree[max_row_index]
mtry_max <- rf.results.df$mtry[max_row_index]
best_accuracy <- rf.results.df$accuracy[max_row_index]
best_precision <- rf.results.df$precision[max_row_index]
best_sensitivity <- rf.results.df$sensitivity[max_row_index]
best_specificity <- rf.results.df$specificity[max_row_index]
best_balanced.accuracy <- rf.results.df$balanced.accuracy[max_row_index]
best_f1.score <- rf.results.df$f1.score[max_row_index]

# Print the values
cat('Best model parameters (maximizing f1 score): ntree: ', ntree_max, ' mtry: ',mtry_max,'\n')
cat('\n')
cat("Training scores for the best model:\n")
cat('- accuracy: ', best_accuracy, '    - precision: ',best_precision, '         - sensitivity: ', best_
cat('- specificity: ', best_specificity, ' - balanced accuracy: ',best_balanced.accuracy,' - f1 score:
```

```
## Best model parameters (maximizing f1 score): ntree:  1100  mtry:  7
##
## Training scores for the best model:
## - accuracy:  0.8600406     - precision:  0.4659864          - sensitivity:  0.2958963
## - specificity:  0.9474565  - balanced accuracy:  0.6216764  - f1 score:  0.3619551
```

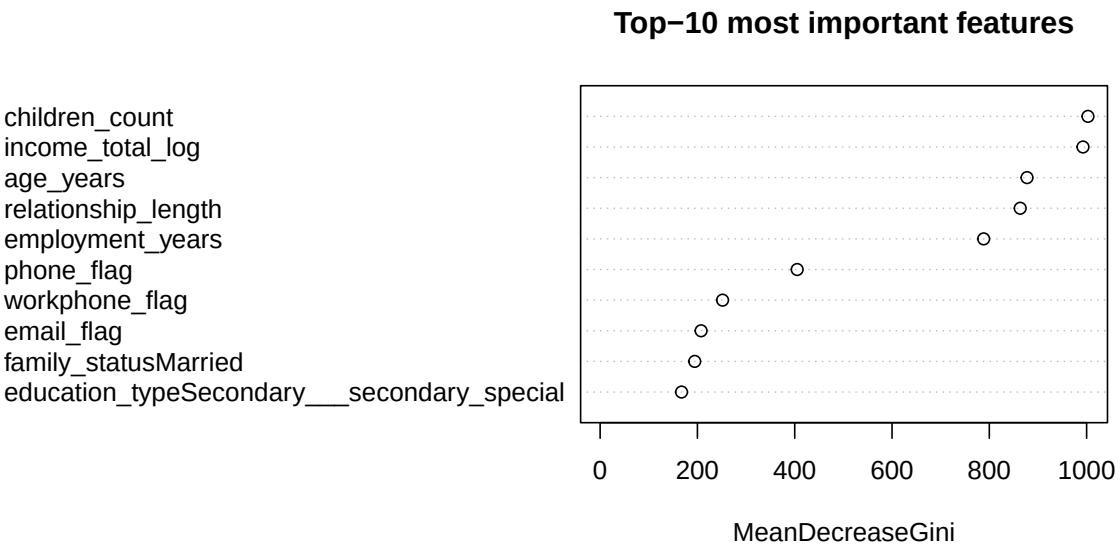
- Low sensitivity, precision, and f1-score implies the model has high proportion of false negatives and false positives and is unreliable. Overall grid search results shown below.

```
#plot gridsearch results
rf.accuracy.plot <- ggplot(rf.results.df, aes(x = ntree, y = accuracy, color = factor(mtry))) +
  geom_line() +
  labs(x = "ntree", y = "accuracy", color = "mtry") +
  scale_color_discrete(name = "mtry") +
  ggtitle('Accuracy') +
  theme_minimal()

rf.precision.plot <- ggplot(rf.results.df, aes(x = ntree, y = precision, color = factor(mtry))) +
  geom_line() +
  labs(x = "ntree", y = "precision", color = "mtry") +
  scale_color_discrete(name = "mtry") +
  ggtitle('Precision') +
  theme_minimal()

rf.sensitivity.plot <- ggplot(rf.results.df, aes(x = ntree, y = sensitivity, color = factor(mtry))) +
  geom_line() +
  labs(x = "ntree", y = "sensitivity", color = "mtry") +
  scale_color_discrete(name = "mtry") +
  ggtitle('Sensitivity') +
  theme_minimal()

rf.f1.score.plot <- ggplot(rf.results.df, aes(x = ntree, y = f1.score, color = factor(mtry))) +
  geom_line() +
  labs(x = "ntree", y = "f1 score", color = "mtry") +
  scale_color_discrete(name = "mtry") +
  ggtitle('f1 score') +
  theme_minimal()

grid.arrange(rf.accuracy.plot, rf.sensitivity.plot, rf.precision.plot, rf.f1.score.plot, ncol = 2)
```



- Key features: the top-5 features, which are children_count, income_total_log, age_years, relationship_length, and employment_years, show significant gini decrease values.

```
#random forest test score
set.seed(1)

#model
rf.best.model <- randomForest(debt_quality ~., data = smote_processed_data, ntree=ntree_max, mtry = mtry
rf.test.pred <- predict (rf.best.model, processed_data.test[, -which(names(processed_data.test) == 'debt
rf.cm.best <- confusionMatrix(rf.test.pred, actual.test)

#plot top-10 feature importance
varImpPlot(rf.best.model, n.var=10, main ='Top-10 most important features')
```




```
# store results of rf best model in overall.results.df
overall.results.df['accuracy','RF'] <- rf.cm.best$overall['Accuracy']
overall.results.df['precision','RF'] <- rf.cm.best$byClass['Precision']
overall.results.df['sensitivity','RF'] <- rf.cm.best$byClass['Sensitivity']
overall.results.df['specificity','RF'] <- rf.cm.best$byClass['Specificity']
overall.results.df['balanced.accuracy','RF'] <- (rf.cm.best$byClass['Sensitivity'] + rf.cm.best$byClass
overall.results.df['f1.score','RF'] <- 2*(rf.cm.best$byClass['Precision'] * rf.cm.best$byClass['Sensiti

#cat('confusion table:\n')
#rf.cm.best
```

2. Logistic Regression

- **Description:** Logistic Regression is a statistical model used for predicting the probability of occurrence of an event by fitting data to a logistic curve. It is particularly adept for binary classification problems. One of the inherent strengths of logistic regression is its ability to estimate the probability of an event, allowing for a nuanced decision-making process. When combined with regularization techniques such

as L1 (Lasso) and L2 (Ridge), the model can mitigate overfitting, especially in scenarios with a large feature set.

- **Design:** (i) Hyperparameter Tuning with Cross-Validation: A comprehensive grid search, coupled with 5-fold cross-validation, was conducted over hyperparameters alpha (indicating the type of regularization) and lambda (denoting the regularization strength). Cross-validation ensures a robust estimation of model performance across different subsets of the training data, preventing overfitting and providing a more generalized model.(ii) Model Training and Evaluation: The logistic regression model was trained using the best hyperparameters derived from the grid search. Performance metrics, including accuracy, precision, sensitivity, specificity, and F1 score, were evaluated on both the training and test datasets to understand the model's predictive power. (iii) Rationale for Metric Selection: Accuracy was chosen as the primary metric for hyperparameter tuning and model evaluation due to several factors: the pre-processing steps ensured a balanced representation of both classes, making accuracy a relevant and reliable performance measure. Logistic regression's output of probabilities necessitates a threshold for classification. With a balanced dataset, accuracy provides a clear and direct measure of the model's correctness in predictions. Among various metrics, accuracy is intuitive, straightforward to interpret, and computationally efficient, making it a practical choice for evaluating model performance.
- **The best model parameters by f1 score shown below:**

```
# Processing variables, create dummy variables, separate categorical variables and numerical variables.
target_data <- application_clean %>% select( 'debt_quality' )
tmpdata <- application_clean %>% select(- 'debt_quality' )

class_data <- tmpdata %>% select_if(is.character)
formula <- dummyVars(~ ., data = class_data)
encoded_data <- predict(formula, newdata = class_data)

num_data <- tmpdata %>% select_if(is.numeric)

merged_data <- cbind(num_data, encoded_data,target_data)

merged_data[['debt_quality']] <- as.factor(merged_data[['debt_quality']])

# Imbalanced dataset, hoping for balance.
data_label1 = subset(merged_data, merged_data$debt_quality == 1)
data_label0 = subset(merged_data, merged_data$debt_quality == 0)

#To calculate the number of observations in two subsets.

count_label1 <- nrow(data_label1)
count_label0 <- nrow(data_label0)

# Select the label with fewer observations and then randomly sample from that dataset to make it as many
set.seed(2)
if (count_label1 < count_label0) {
  data_label0 <- data_label0[sample(nrow(data_label0), count_label1), ]
} else {
  data_label1 <- data_label1[sample(nrow(data_label1), count_label0), ]
}

# Merge datasets
balanced_data <- rbind(data_label0, data_label1)

# Split the dataset into training set and test set
```

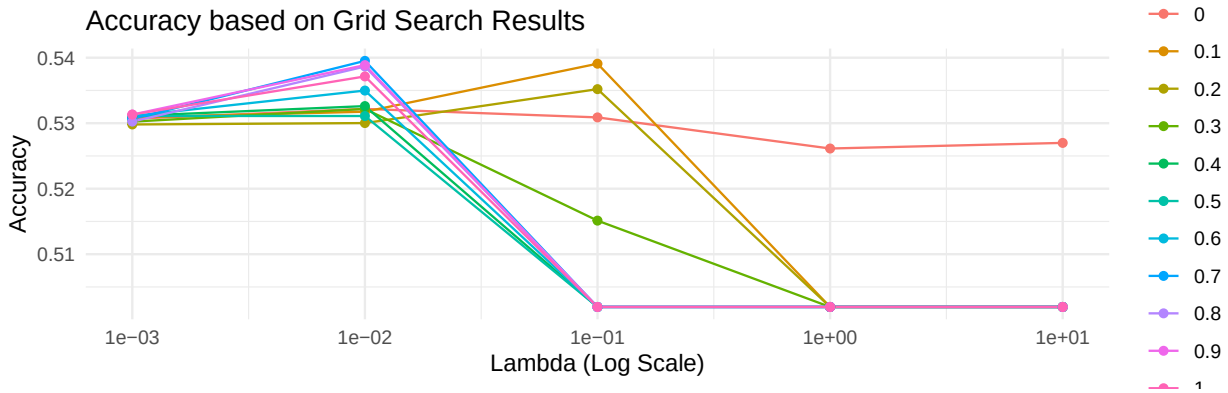
```r
train_index = sample(1:nrow(balanced_data), 0.8 * nrow(balanced_data))
train_data = balanced_data[train_index, ]
test_data = balanced_data[-train_index, ]

# Grid search for logistic regression
# Define the grid
hyper_grid <- expand.grid(
  alpha = seq(0, 1, by=0.1), # from ridge (0) to lasso (1)
  lambda = c(0.001, 0.01, 0.1, 1, 10)
)

# Define the control for training
train_control <- trainControl(
  method = "cv",
  number = 5,
  search = "grid"
)

# Train the model using the grid search
grid_search <- train(
  debt_quality ~ .,
  data = train_data,
  method = "glmnet",
  trControl = train_control,
  tuneGrid = hyper_grid
)

# Extract the best hyperparameters from the grid search
best_alpha <- grid_search$bestTune$alpha
best_lambda <- grid_search$bestTune$lambda

# Train the glmnet model using the best hyperparameters
set.seed(123) # for reproducibility
fit <- glmnet(as.matrix(train_data[, -ncol(train_data)]), # predictors
              train_data$debt_quality, # response
              alpha = best_alpha,
              lambda = best_lambda,
              family = "binomial")

# # Predict using the best model on the training data
train_probs <- predict(fit, newx = as.matrix(train_data[, -ncol(train_data)]), type = "response")
train_pred <- ifelse(train_probs > 0.5, 1, 0)

# Evaluate the model's performance on the training data using confusionMatrix
confusion_matrix_train <- confusionMatrix(as.factor(train_pred), as.factor(train_data$debt_quality))

# Extract evaluation metrics
LogRes.train_results.table = confusion_matrix_train$table
LogRes.train_results.accuracy = confusion_matrix_train$overall["Accuracy"]
LogRes.train_results.precision = confusion_matrix_train$byClass["Precision"]
LogRes.train_results.sensitivity = confusion_matrix_train$byClass["Sensitivity"]
LogRes.train_results.specificity = confusion_matrix_train$byClass["Specificity"]
LogRes.train_results.balanced.accuracy = (LogRes.train_results.sensitivity + LogRes.train_results.speci
```

```
LogRes.train_results.f1 = confusion_matrix_train$byClass["F1"]

# Print out the training results
#LogRes.train_results.table

cat('Best model parameters (maximizing accuracy score): alpha: ', best_alpha, ' lambda: ',best_lambda,'
cat('\n')
cat("Training scores for the best model:\n")
cat('- accuracy: ', LogRes.train_results.accuracy, '    - precision: ',LogRes.train_results.precision,
cat('- specificity: ', LogRes.train_results.specificity, ' - balanced accuracy: ',LogRes.train_results.

# Predict using the best model on the test data
test_probs <- predict(fit, newx = as.matrix(test_data[, -ncol(test_data)]), type = "response")
test_pred <- ifelse(test_probs > 0.5, 1, 0)

# Evaluate the model's performance on the test data using confusionMatrix
confusion_matrix_updated <- confusionMatrix(as.factor(test_pred), as.factor(test_data$debt_quality))

# Extract evaluation metrics
#LogRes.results.table = confusion_matrix_updated$table
LogRes.results.accuracy = confusion_matrix_updated$overall["Accuracy"]
LogRes.results.precision = confusion_matrix_updated$byClass["Precision"]
LogRes.results.sensitivity = confusion_matrix_updated$byClass["Sensitivity"]
LogRes.results.specificity = confusion_matrix_updated$byClass["Specificity"]
LogRes.results.balanced.accuracy = (LogRes.results.sensitivity + LogRes.results.specificity)/2
LogRes.results.f1 = confusion_matrix_updated$byClass["F1"]

# Print out the results
#LogRes.results.table
#cat('The testing set accuracy is: ',LogRes.results.accuracy,'\n')
#cat('The testing set precision is: ',LogRes.results.precision,'\n')
#cat('The testing set sensitivity is: ',LogRes.results.sensitivity,'\n')
#cat('The testing set specificity is: ',LogRes.results.specificity,'\n')
#cat('The testing set balance accuracy is: ',LogRes.results.balanced.accuracy,'\n')
#cat('The testing set f1 score is: ',LogRes.results.f1,'\n')


# store results of logres in overall.results.df
overall.results.df['accuracy','LogRes'] <- LogRes.results.accuracy
overall.results.df['precision','LogRes'] <- LogRes.results.precision
overall.results.df['sensitivity','LogRes'] <- LogRes.results.sensitivity
overall.results.df['specificity','LogRes'] <- LogRes.results.specificity
overall.results.df['balanced.accuracy','LogRes'] <- LogRes.results.balanced.accuracy
overall.results.df['f1.score','LogRes'] <- LogRes.results.f1

results <- grid_search$results
```

```
## Best model parameters (maximizing accuracy score): alpha:  0.7  lambda:  0.01
##
## Training scores for the best model:
## - accuracy:  0.5514039     - precision:  0.5445352          - sensitivity:  0.6071119
## - specificity:  0.4961274  - balanced accuracy:  0.5516196  - f1 score:  0.5741234
```

- Accuracy scores for different alpha and lambda values shown below:

```
# Plot
ggplot(results, aes(x = lambda, y = Accuracy, color = as.factor(alpha))) +
  geom_line(aes(group = alpha)) +
  geom_point() +
  scale_x_log10() + # Lambda values are often plotted on a log scale for better visualization
  labs(title = "Accuracy based on Grid Search Results",
       x = "Lambda (Log Scale)",
       y = "Accuracy",
       color = "Alpha") +
  theme_minimal()
```



3. Linear discriminant analysis (LDA)

- **Description:** LDA is a dimensionality reduction technique that works by finding the linear combination of features that best separates multiple classes in data. It is very interpretable, and very robust against overfitting since it best estimates the discriminant boundaries using covariance information. LDA does not require any hyperparameter tuning, since the lda() function in the MASS package tunes parameters itself, using the covariance matrices of the input data. This means that for this technique, only cross validation has been done to ensure robustness.
- **Design:** (i) Data pre-processing: min-max scaling, one-hot-encoding and SMOTE were applied to the dataset, (ii) hyperparameter tuning: lda() was run with 5 fold cross-validation, (iii) Rationale for metric selection: best accuracy score is chosen since : 1) The SMOTE pre processing done means that the classes are not too imbalanced, hence accuracy becomes a relevant metric , 2) LDA works in such a way that it uses correlation matrices to find the best results, hence accuracy is generally the more useful metric than sensitivity or precision since this mitigates a lot of the imbalance and 3) accuracy is the easiest metric to interpret and it is the most computationally efficient to calculate.
- **The best model parameters by accuracy score shown below:**

```
set.seed(14)

# Set up the train control
ctrl <- trainControl(method = "cv", number = 5)

# Perform LDA with cross-validation
lda_model <- train(debt_quality ~ .,
                   data = smote_processed_data,
```

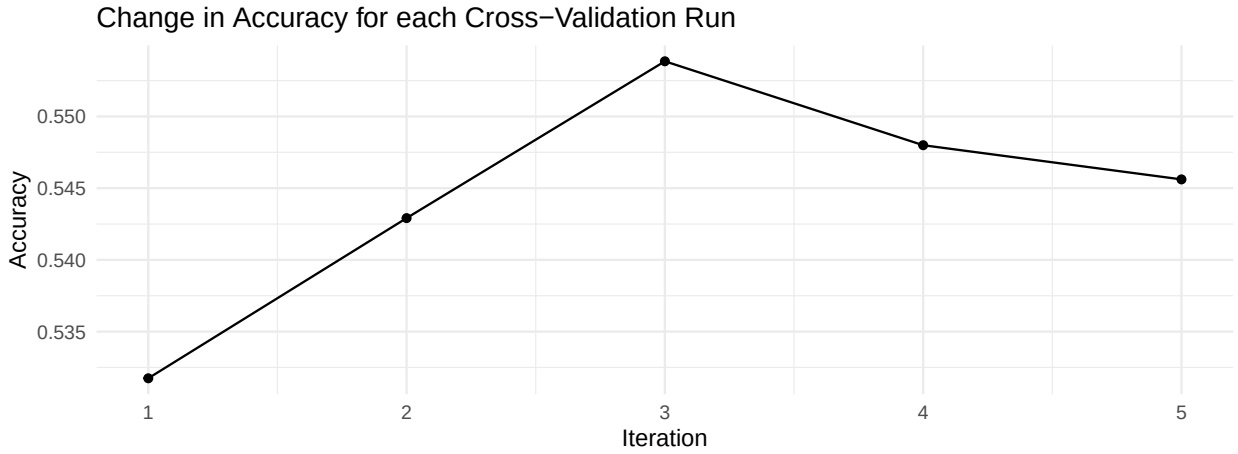
```
                    method = "lda",
                    trControl = ctrl)

# Access the accuracy values from each iteration of the cross-validation
cv_results <- data.frame(Accuracy = lda_model$resample$Accuracy, Iteration = 1:5)

# Print the accuracy on each run of the cross-validation
#for (i in 1:5) {
#  cat(paste("Accuracy on run", i, ":", cv_results$Accuracy[i], "\n"))
#}

# Extract the best model
best_lda <- lda_model$finalModel

# Make predictions on the training set
train.pred <- predict(best_lda, newdata = smote_processed_data[, -which(names(smote_processed_data) ==
cm_train <- confusionMatrix(train.pred$class, smote_processed_data$debt_quality)

lda.train.accuracy <- cm_train$overall['Accuracy']
lda.train.sensitivity <- cm_train$byClass['Sensitivity']
lda.train.precision <- cm_train$byClass['Precision']
lda.train.specificity <- cm_train$byClass['Specificity']
lda.train.balanced.accuracy <- (cm_train$byClass['Sensitivity'] + cm_train$byClass['Specificity']) / 2
lda.train.f1.score <- 2*(cm_train$byClass['Precision'] * cm_train$byClass['Sensitivity'])/(cm_train$byC

# Print the results of the best model on the training set
cat("Training scores for the best model:\n")
cat('- accuracy: ', lda.train.accuracy, '    - precision: ',lda.train.precision, '         - sensitivity
cat('- specificity: ', lda.train.specificity, ' - balanced accuracy: ',lda.train.balanced.accuracy,' - f


# Use the best model to make predictions on the test set
test.pred <- predict(best_lda, newdata = processed_data.test[, -which(names(processed_data.test) == 'del
cm_best <- confusionMatrix(test.pred$class, actual.test)

# Store results of the best model in overall.results.df
overall.results.df['accuracy', 'LDA'] <- cm_best$overall['Accuracy']
overall.results.df['sensitivity', 'LDA'] <- cm_best$byClass['Sensitivity']
overall.results.df['precision', 'LDA'] <- cm_best$byClass['Precision']
overall.results.df['specificity', 'LDA'] <- cm_best$byClass['Specificity']
overall.results.df['balanced.accuracy', 'LDA'] <- (cm_best$byClass['Sensitivity'] + cm_best$byClass['Spe
overall.results.df['f1.score', 'LDA'] <-2*(cm_best$byClass['Precision'] * cm_best$byClass['Sensitivity']
```

```
## Training scores for the best model:
## - accuracy:  0.5507064     - precision:  0.55756          - sensitivity:  0.6436935
## - specificity:  0.4507106  - balanced accuracy:  0.5472021  - f1 score:  0.5975387
```

- Accuracy scores shown below:

```
# Plot the change in accuracy for each run of the cross-validation
ggplot(cv_results, aes(x = Iteration, y = Accuracy)) +
  geom_line() +
  geom_point() +
```

```
  labs(title = "Change in Accuracy for each Cross-Validation Run", x = "Iteration", y = "Accuracy") +
  theme_minimal()
```



4. AdaBoost

- **Description:** Ada is an ensemble learning algorithm that builds a strong classifier by combining multiple weak learners, in this case decision trees. Ada works by focusing on instances of the dataset that are most difficult to classify by adaptively adjusting weights during successive training rounds.
- **Design:** (i) Data pre-processing: mix-max scaling, one-hot encoding were applied. Data was then oversampled using upSample and SMOTE. (ii) Hyperparameter tuning: grid search was run on iterator, loss functions, maxdepth values as well as the two oversampling techniques. (iii) Rationale for metric selection: balanced accuracy was chosen as it takes into account both the true positive rate (sensitivity) and true negative rate, while mitigating the impact of class imbalance.
- **The best model parameters by accuracy score shown below:**

```
#apply Upsampling to training data to balance class weights
upSampled_data <- upSample(x = processed_data.train[, -which(names(processed_data.train) == 'debt_quali
names(upSampled_data)[names(upSampled_data) == "Class"] <- "debt_quality"

#initiate results grid
ada_results.df <- data.frame(loss = character(),
                         maxdepth = integer(),
                         iter = integer(),
                         bal = character(),
                         accuracy = numeric(),
                         precision = numeric(),
                         sensitivity = numeric(),
                         specificity = numeric(),
                         balanced.accuracy = numeric(),
                         f1.score = numeric(),
                         true.positive = integer(),
                         true.negative = integer(),
                         false.positive = integer(),
                         false.negative = integer()
                         )
```

```r
#hyperparameter grid
iter_values <- c(30, 60)
loss_values <- c("exponential", "huber", "gentle")
maxdepth_values <- c(1, 5)
balance_values <- c("smote", "upSample")

#grid search loop
for (i in iter_values) {
  for (l in loss_values) {
    for (d in maxdepth_values){
      for (b in balance_values){

        #set 'maxdepth' for the 'rpart' decision trees
        control_rpart <- rpart.control(maxdepth = d)

        #apply class balancing of either upSample or SMOTE
        if(b == "upSample"){
          balanced_data <- upSampled_data
        }else{
          balanced_data <- smote_processed_data
        }

        #model and validation
        ada_model <- ada(debt_quality ~ ., data = balanced_data, iter = i, loss = l, control = control_r
        ada_val <- predict(ada_model, processed_data.val[, -which(names(processed_data.val) == 'debt_qua
        ada_cm <- confusionMatrix(ada_val, actual.val)

        #add results to the dataframe
        ada_results.df <- rbind(ada_results.df, data.frame(loss = l,
                                                           maxdepth = d,
                                                           iter = i,
                                                           bal = b,
                                                           accuracy = ada_cm$overall['Accuracy'],
                                                           precision = ada_cm$byClass['Precision'],
                                                           sensitivity = ada_cm$byClass['Sensitivity'],
                                                           specificity = ada_cm$byClass['Specificity'],
                                                           balanced.accuracy = (ada_cm$byClass['Sensitivi
                                                           f1.score = 2 * ada_cm$byClass['Precision'] * ac
                                                           true.positive = ada_cm$table[2,2],
                                                           true.negative = ada_cm$table[1,1],
                                                           false.positive = ada_cm$table[1,2],
                                                           false.negative = ada_cm$table[2,1]))
      }
    }
  }
}
```

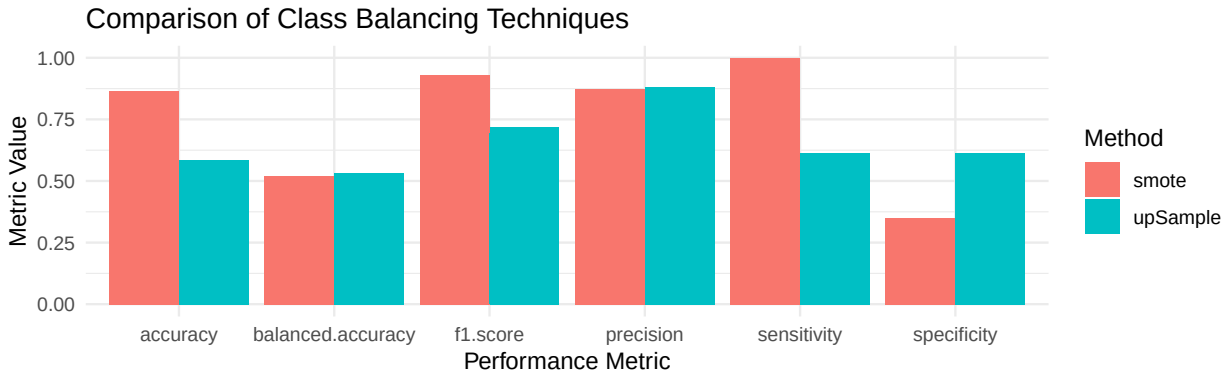
```r
#Find params with best balanced.accuracy
sorted_results <- ada_results.df[order(-ada_results.df$balanced.accuracy), ]
best_model_params <- sorted_results[1, ]

#ada_results.df
```

```r
best_loss <- best_model_params$loss
best_maxdepth <- best_model_params$maxdepth
best_iter <- best_model_params$iter
best_bal <- best_model_params$bal

control_rpart <- rpart.control(maxdepth = best_maxdepth)

max_balanced_accuracy <- best_model_params$balanced.accuracy
max_accuracy <- best_model_params$accuracy
max_precision <- best_model_params$precision
max_sensitivity <- best_model_params$sensitivity # also known as recall
max_specificity <- best_model_params$specificity
max_f1 <- best_model_params$f1.score

cat('Best model parameters (maximizing balanced accuracy score):\n')
cat('- loss function: ', best_loss, '  - max depth: ',best_maxdepth,'\n')
cat('- iterations: ',best_iter,'        - balancing technique: ', best_bal, '\n')
cat('\n')
cat("Training scores for the best model:\n")
cat('- accuracy: ', max_accuracy, '    - precision: ',max_precision, '         - sensitivity: ', max_se
cat('- specificity: ', LogRes.train_results.specificity, ' - balanced accuracy: ',max_balanced_accuracy


if(best_bal == "upSample"){
  best_balanced_data <- upSampled_data
}else{
  best_balanced_data <- smote_processed_data
}


#run the best model and predict the test data
best_ada_model <- ada(debt_quality ~ ., data = best_balanced_data, iter = best_iter, loss = best_loss,

ada_test_pred <- predict(best_ada_model, processed_data.test[, -which(names(processed_data.test) == 'de

ada_test_results <- confusionMatrix(ada_test_pred, actual.test)

#store results of best AdaBoost model in the overall.results.df
overall.results.df['accuracy','AdaBoost'] <- ada_test_results$overall['Accuracy']
overall.results.df['precision','AdaBoost'] <- ada_test_results$byClass['Precision']
overall.results.df['sensitivity','AdaBoost'] <- ada_test_results$byClass['Sensitivity']
overall.results.df['specificity','AdaBoost'] <- ada_test_results$byClass['Specificity']
overall.results.df['balanced.accuracy','AdaBoost'] <- (ada_test_results$byClass['Sensitivity'] + ada_te
overall.results.df['f1.score','AdaBoost'] <- 2 * (ada_test_results$byClass['Precision'] * ada_test_resu
```

```
## Best model parameters (maximizing balanced accuracy score):
## - loss function:  huber   - max depth:  5
## - iterations:  60         - balancing technique:  upSample
##
## Training scores for the best model:
## - accuracy:  0.5575196     - precision:  0.8790867          - sensitivity:  0.5669344
## - specificity:  0.4961274  - balanced accuracy:  0.5318473  - f1 score:  0.6893184
```

- Scores for SMOTE and up-sample shown below:

```
tidy_results <- ada_results.df %>%
  gather(key = "metric", value = "value", accuracy, precision, sensitivity, specificity, balanced.accura
  filter(bal %in% c("smote", "upSample"))

#visualizing the effect of class balancing techniques on results
ggplot(tidy_results, aes(fill=bal, y=value, x=metric)) +
  geom_bar(position="dodge", stat="identity") +
  labs(y = "Metric Value", x = "Performance Metric", title = "Comparison of Class Balancing Techniques"
  theme_minimal()
```



5. Support vector machine

- **Description:** Support Vector Machines (SVM) are a set of supervised learning methods used for classification, regression, and outliers detection. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other. It is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.
- **Design:** (i) Data pre-processing: min-max scaling and one-hot-encoding were applied to the dataset. Principal Component Analysis (PCA) was used on train and test sets for feature selection. (ii) Hyper-parameter tuning: the grid search technique was applied on **'class.weights'** to find the **best f1 score**. (iv) Rationale for metric selection: f1-score was chosen to achieve a good balance between precision (minimizing false positives) and recall (minimizing false negatives). Accuracy was not maximised as the dataset is imbalanced.
- **The best model parameters by f1 score shown below:**

```
#create train and test data

processed_data <- preprocess_data(application_clean, target_col='debt_quality')

#apply train test split, createDataPartition automatically stratifies classes
set.seed(5003)
inTrain <- createDataPartition(processed_data[['debt_quality']],p=0.8)[[1]]
processed_data.train <- processed_data[inTrain,]
processed_data.test <- processed_data[-inTrain,]
```

```r
library(e1071)

# Apply grid search for SVM
grid_search_svm <- function(train_data, test_data, class_weights) {
  #initiate results grid
  svm.results.df <- data.frame(
                          class.weight = numeric(),
                          accuracy = numeric(),
                          precision = numeric(),
                          sensitivity = numeric(),
                          specificity = numeric(),
                          balanced.accuracy = numeric(),
                          f1.score = numeric()
                          )
  for(class_weight in class_weights) {
    svm.model <- svm(debt_quality ~ ., data = train_data, kernel = "radial", class.weights = class_weigh
    svm.predict <- predict(svm.model, test_data)
    svm.cm <- confusionMatrix(svm.predict, reference = test_data$debt_quality, positive="1")

    #add result to dataframe
    svm.results.df <- rbind(svm.results.df , data.frame(class.weight = class_weight,
                                            accuracy = svm.cm$overall['Accuracy'],
                                            precision = svm.cm$byClass['Precision'],
                                            sensitivity = svm.cm$byClass['Sensitivity'],
                                            specificity = svm.cm$byClass['Specificity'],
                                            balanced.accuracy = (svm.cm$byClass['Sensitivity'] + svm.
                                            f1.score = 2 * svm.cm$byClass['Precision'] * svm.cm$byCla
  }
  return(svm.results.df)
}
```

```r
#Utilize feature selection techniques

#Use pca for selecting features
select_data <- preProcess(processed_data.train, method = "pca", pcaComp = 30)

#Desperate data for grid search
set.seed(5003)
inTrain <- createDataPartition(processed_data.train[['debt_quality']],p=0.8)[[1]]
processed_data.train.set <- processed_data.train[inTrain,]
processed_data.train.test <- processed_data.train[-inTrain,]

train_data_pca <- predict(select_data, processed_data.train.set)
train_predict_data_pca <- predict(select_data, processed_data.train.test)

train_data_pca_full <- predict(select_data, processed_data.train)
test_data_pca <- predict(select_data, processed_data.test)

#Set class_weights set, the max of the class weights should be less than the ratio of two classes of da
class_weights = list(c("0" = 1, "1" = 2),  c("0" = 1, "1" = 3), c("0" = 1, "1" = 4),c("0" = 1, "1" = 5)

#Use grid search to find the appropriate class weight parameters
svm.result.df <- grid_search_svm(train_data_pca, train_predict_data_pca, class_weights)
```

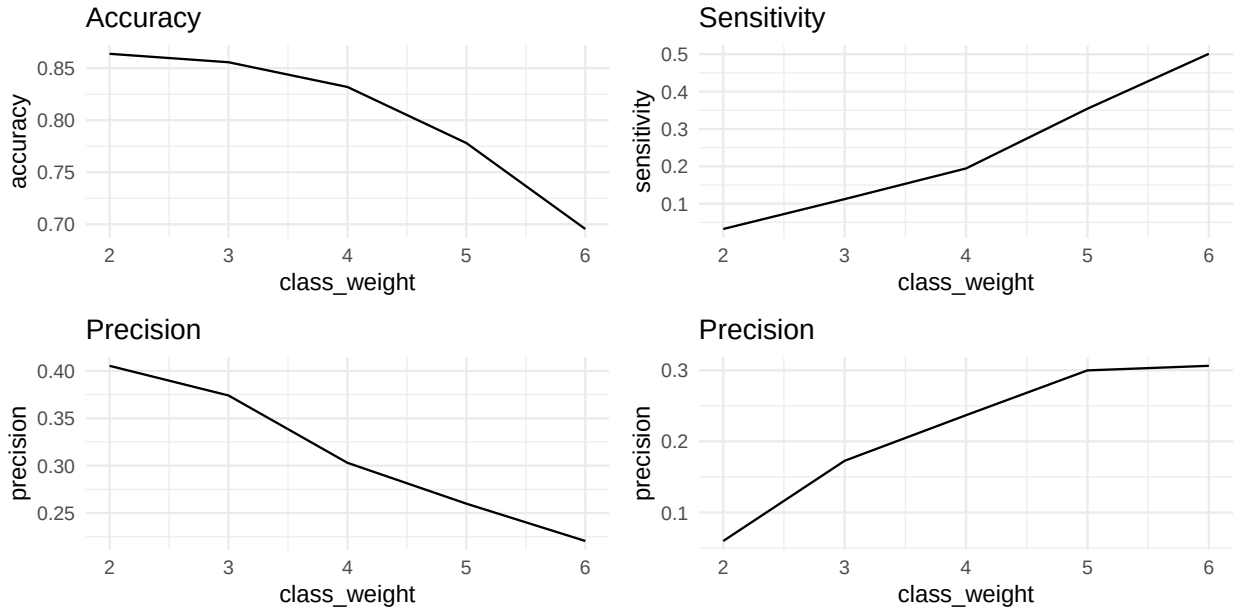
```r
svm.result.df <- svm.result.df[svm.result.df$class.weight != 1, ]

#define row index for max f1 score
max_row_index <- which.max(svm.result.df$f1.score)
best_result <- svm.result.df[max_row_index, ]
class_weight = class_weights[[max_row_index]]

# Print out best model results

cat('Best model parameters (maximizing f1 score): class weights: ', class_weight,'\n')
cat('\n')
cat("Training scores for the best model:\n")
cat('- accuracy: ', best_result$accuracy, '    - precision: ',best_result$precision, '         - sensit
cat('- specificity: ', best_result$specificity, ' - balanced accuracy: ',best_result$balanced.accuracy,


# Use the appropriate params to create model
pca.svm.model <- svm(debt_quality ~ ., data = train_data_pca_full, kernel = "radial", class.weights = cl

# Predict on the test data set using pca
pca.svm.predictions <- predict(pca.svm.model,test_data_pca)
confusion_matrix_pca <- confusionMatrix(pca.svm.predictions, reference =  test_data_pca$debt_quality)

overall.results.df['accuracy', 'SVM'] <- confusion_matrix_pca$overall["Accuracy"]
overall.results.df['sensitivity', 'SVM'] <- confusion_matrix_pca$byClass["Sensitivity"]
overall.results.df['precision', 'SVM'] <- confusion_matrix_pca$byClass["Precision"]
overall.results.df['specificity', 'SVM'] <- confusion_matrix_pca$byClass["Specificity"]
overall.results.df['balanced.accuracy', 'SVM'] <- (confusion_matrix_pca$byClass["Sensitivity"] + confusi
overall.results.df['f1.score', 'SVM'] <- 2*(confusion_matrix_pca$byClass['Precision'] * confusion_matri
```

```
## Best model parameters (maximizing f1 score): class weights:  1 6
##
## Training scores for the best model:
## - accuracy:  0.6954506     - precision:  0.2205323          - sensitivity:  0.5010799
## - specificity:  0.7255689  - balanced accuracy:  0.6133244  - f1 score:  0.3062706
```

- Plots of results shown below:

```r
#plot gridsearch results
svm.accuracy.plot <- ggplot(svm.result.df, aes(x = class.weight, y = accuracy)) +
  geom_line() +
  labs(x = "class_weight", y = "accuracy") +
  ggtitle('Accuracy') +
  theme_minimal()

svm.sensitivity.plot <- ggplot(svm.result.df, aes(x = class.weight, y = sensitivity)) +
  geom_line() +
  labs(x = "class_weight", y = "sensitivity") +
  ggtitle('Sensitivity') +
  theme_minimal()

svm.precision.plot <- ggplot(svm.result.df, aes(x = class.weight, y = precision)) +
  geom_line() +
```

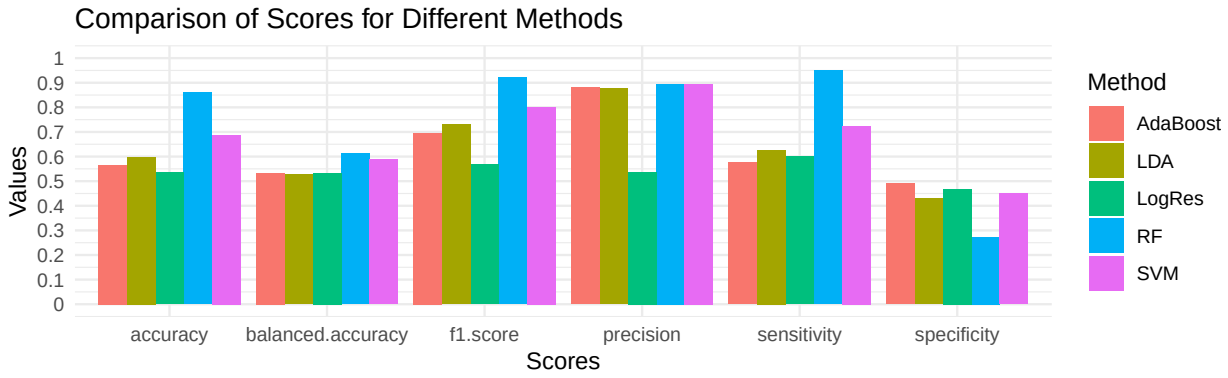
```
  labs(x = "class_weight", y = "precision") +
  ggtitle('Precision') +
  theme_minimal()

svm.f1.score.plot <- ggplot(svm.result.df, aes(x = class.weight, y = f1.score)) +
  geom_line() +
  labs(x = "class_weight", y = "precision") +
  ggtitle('Precision') +
  theme_minimal()

grid.arrange(svm.accuracy.plot, svm.sensitivity.plot, svm.precision.plot,svm.f1.score.plot,ncol = 2)
```



RESULTS

```
rounded.results.df <- round(overall.results.df,3)

# Assuming your dataframe is named rounded.results.df
library(tibble)
library(tidyr)

# Convert row names to a new column
rounded.results.df <- rownames_to_column(rounded.results.df, var = "Category")

# Reshape the dataframe from wide to long format
rounded.results.df_long <- gather(rounded.results.df, Method, Score, -Category)

# Your existing data and ggplot code
ggplot(rounded.results.df_long, aes(x = Category, y = Score, fill = Method)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Scores", y = "Values") +
```

```
  ggtitle("Comparison of Scores for Different Methods") +
  theme_minimal() +

  # Specify y-axis breaks, labels, and limits
  scale_y_continuous(breaks = seq(0, 1.0, by = 0.1),
                     labels = seq(0, 1.0, by = 0.1),
                     limits = c(0, 1.0))
```



Due to data imbalance, performance measures which accounts for true values while minimizing false positives and false negatives were analyzed:

- **Precision = (True Positives / (True Positives + False Positives))**: precision measures the accuracy of a model when it predicts a positive class. All models aside from logistic regression performed relatively well (87%+) which means they were able to correctly identify actual bad debtors and not misclassifying too many good debtors as bad debtors.
- **Sensitivity = (True Positives / (True Positives + False Negatives))**: sensitivity measures the model's ability to correctly identify positive instances from the total number of actual positive instances in the dataset. Random forest scored highest which means the model was able to correctly identify almost all bad debtors in the test set.

Final model recommendation: We recommend Random Forest (RF) for this analysis, though it still has its limitations, as shown by its low specificity score. This implies that the RF model is 'too strict', where some good debtors are also flagged as bad debtors. This limitation may be caused by data imbalance which has been minimized by SMOTE but still presents some bias. We are comfortable with this flaw as having false positives is conservative and carry lesser economic impact than having false negatives (e.g., lost revenue instead of bad debt on the banking book). The higher test scores may be caused by underfitting or spurious relationship which will be discussed below.

DISCUSSION

Potential shortcomings and future work: (i) Data imbalance: additional data points on bad debt would address the bias experienced in this dataset. (ii) Lack of features: additional features could be collected which may have higher correlation with the target feature. For example: customer credit score, criminal records, past history of indebtedness, identifier on any existing factors that would signal lower delinquency risk such as letter of guarantee, employer reference, etc. (iii) Hyperparameter tuning: a wider tuning grid could be applied to ensure that each model achieves its global optimum solution.

Conclusion: Machine learning can be utilized as a first layer screening for potential bad debtors if it is provided with a rich dataset. Our largest obstacle was data imbalance which can be solved with a larger

number (and wider spread) of customer track record and richer information at point of application. This exercise also clarified the importance of credit score in the credit card industry. Credit score accumulates credit behaviour data from sources not directly available otherwise at point of application. Credit score mayserve as a strong 'leak' of customers' tendency to default in the future.

REFERENCES

Li, Y., Li, Y., & Li, Y. (2019). What factors are influencing credit card customer's default behavior in China? A study based on survival analysis. Physica A: Statistical Mechanics and Its Applications, 526, 120861. https://doi.org/10.1016/j.physa.2019.04.097

Leong, O. J., & Jayabalan, M. (2019). A Comparative Study on Credit Card Default Risk Predictive Model. Journal of Computational and Theoretical Nanoscience, 16(8), 3591–3595. https://doi.org/10.1166/jctn.2019.8330

Wang, L., Lu, W., & Malhotra, N. K. (2011). Demographics, attitude, personality and credit card features correlate with credit card debt: A view from China. Journal of Economic Psychology, 32(1), 179–193. https://doi.org/10.1016/j.joep.2010.11.006

CONTRIBUTION STATEMENT

All members contribute equally on all aspect of this project.