Data cleaning, preprocessing, pipeline and model creation

Group F

2024-10-30

```r
# Define the Mode function
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
```

Load dataset (CustData2.csv)

```r
# Read 'CustData2.csv' file into data frame 'customers'
custData <- read.csv("CustData2.csv")

# Feature engineering -> create the eligible attribute based on the annual salary
custData$Eligible <- ifelse(custData$Annual.Salary > 50000, 1, 0)
```

Generate and save importand variables for use in prediction

```r
# Calculate % eligible of baseline
numEligibleOriginal <- sum(custData$Eligible == 1, na.rm = TRUE)
totalCustomersOriginal <- length(custData$Eligible)
eligiblePercantageOriginal <- (numEligibleOriginal /
                                 totalCustomersOriginal) * 100
cat("Percentage of Eligible Customers in baseline model: ",
    round(eligiblePercantageOriginal, 2))
```

```
## Percentage of Eligible Customers in baseline model:  64.43
```

```r
# Calculate frequency tables with column names aligned to match new_record
title_frequency <- table(custData$Title)
department_frequency <- table(custData$Department.Name)
custData$Country_id <- as.character(custData$Country_id)
country_frequency <- table(custData$Country_id)
```

```r
# Write the frequency encoded values to csv files for use in the predictions
write.csv(data.frame(title_frequency = title_frequency),
          "title_frequency.csv", row.names = FALSE)
write.csv(data.frame(department_frequency = department_frequency),
          "department_frequency.csv", row.names = FALSE)
write.csv(data.frame(country_frequency = country_frequency),
          "country_frequency.csv", row.names = FALSE)
```

Pipeline

Create pipeline function

```
# Define preprocessing pipeline function
create_preprocessing_pipeline <- function(data, title_freq,
                                          dept_freq, country_freq) {
  # Step 1: Feature Engineering
  data$Age <- as.integer(year(today()) - data$year_of_birth)

  # Remove unnecessary columns
  keepColumns <- c("Title", "Department.Name", "Annual.Salary",
                   "Gross.Pay.Last.Paycheck", "Gross.Year.To.Date",
                   "Gross.Year.To.Date...FRS.Contribution",
                   "Age", "marital_status", "Country_id", "Education",
                   "Occupation", "household_size", "yrs_residence", "Eligible")
  data <- data[keepColumns]

  # Step 2: Data Cleaning
  data$marital_status <- tolower(data$marital_status)
  data$marital_status <- recode(data$marital_status, "married" = "married",
                                "mar-af" = "married", "neverm" = "single",
                                "mabsent" = "single", "divorc." = "divorced",
                                "separ." = "divorced", "widow" = "widowed",
                                "widowed" = "widowed")

  # Fill missing values for marital status with mode
  data$marital_status[is.na(data$marital_status) | data$marital_status == ""] <-
    Mode(data$marital_status)

  # Remove empty cells for all columns/attributes
  data <- data[!(is.na(data$Title) | data$Title == "" |
                     is.na(data$Department.Name)  |
                     data$Department.Name == ""  |
                     is.na(data$Annual.Salary)  |
                     data$Annual.Salary == ""  |
                     is.na(data$Gross.Pay.Last.Paycheck)  |
                     data$Gross.Pay.Last.Paycheck == ""  |
                     is.na(data$Gross.Year.To.Date)  |
                     data$Gross.Year.To.Date == ""  |
                     is.na(data$Gross.Year.To.Date...FRS.Contribution)  |
                     data$Gross.Year.To.Date...FRS.Contribution == ""), ]

  # Step 3: Outlier Treatment
  cap_outliers <- function(column) {
    lower_cap <- quantile(column, 0.01, na.rm = TRUE)
    upper_cap <- quantile(column, 0.99, na.rm = TRUE)
    column[column < lower_cap] <- lower_cap
    column[column > upper_cap] <- upper_cap
    return(column)
  }
  num_vars <- c("Annual.Salary", "Gross.Pay.Last.Paycheck",
                "Gross.Year.To.Date", "Gross.Year.To.Date...FRS.Contribution")
  data[num_vars] <- lapply(data[num_vars], cap_outliers)

  # Step 4: Frequency Encoding (frequency encoding using pre-calculated tables)
  data$Frequency_Title <- ifelse(data$Title %in% names(title_freq),
                                 title_freq[data$Title], 0)
  data$Frequency_Department <-
    ifelse(data$Department.Name %in% names(dept_freq),
           dept_freq[data$Department.Name], 0)
  data$Frequency_Country_ID <-
    ifelse(data$Country_id %in% names(country_freq),
           country_freq[data$Country_id], 0)

  # Step 5: One-Hot Encoding
  # One-hot encode marital status
  data$Marital_Status_married <-
    ifelse(data$marital_status == "married", 1, 0)
  data$Marital_Status_single <-
    ifelse(data$marital_status == "single", 1, 0)
  data$Marital_Status_divorced <-
    ifelse(data$marital_status == "divorced", 1, 0)
  data$Marital_Status_widowed <-
    ifelse(data$marital_status == "widowed", 1, 0)

  # One-hot encode education
  data$Education_Bach <- ifelse(data$Education == "Bach.", 1, 0)
  data$Education_Masters <- ifelse(data$Education == "Masters", 1, 0)
  data$Education_HS <- ifelse(data$Education == "HS-grad", 1, 0)

  # One-hot encode occupation
  data$Occupation_Cleric <- ifelse(data$Occupation == "Cleric.", 1, 0)
  data$Occupation_Prof <- ifelse(data$Occupation == "Prof.", 1, 0)
  data$Occupation_Exec <- ifelse(data$Occupation == "Exec.", 1, 0)
  data$Occupation_Sales <- ifelse(data$Occupation == "Sales", 1, 0)

  # Remove irrelevant attributes that are not longer needed
  data <- data %>% select(-Title, -Department.Name, -Country_id,
                          -marital_status, -Education, -Occupation)

  # Step 6: Create preprocessing pipeline
  preprocess_pipeline <- preProcess(data, method = c("center",
                                                     "scale", "BoxCox"))

  return(list(data = data, pipeline = preprocess_pipeline))
}
```

Run pipeline

```r
# Apply preprocessing pipeline
processed_data <- create_preprocessing_pipeline(custData,
                    title_frequency, department_frequency,
                    country_frequency)
custData <- predict(processed_data$pipeline, newdata = processed_data$data)
```

```r
# Ensure there are no missing values in the target variable 'Eligible'
custData <- custData[!is.na(custData$Eligible), ]
custData$Eligible <- as.factor(custData$Eligible)
```

Generate model

Split data into training and testing sets

```r
# Split data into training (80%) and testing sets (20%)
set.seed(123)
train_index <- createDataPartition(custData$Eligible, p = 0.8, list = FALSE)
train_data <- custData[train_index, ]
test_data <- custData[-train_index, ]
```

```r
# Remove rows with missing values in train_data
train_data <- na.omit(train_data)
```

Train Random Forest Model

```r
# Train Random Forest Model using pipeline
train_data$Eligible <- as.factor(train_data$Eligible)
randomForest_model <- randomForest(Eligible ~ . -Annual.Salary,
                                   data = train_data, ntree = 100,
                                   mtry = 3, importance = TRUE)
```

Predict using the Random Forest Model

```r
# Predictions using the model for new data
randomForest_predictions <- predict(randomForest_model, newdata = test_data)
```

Evaluate Model

Create confusion matrix

```r
# Confusion Matrix for Random Forest
randomForest_cm <- confusionMatrix(as.factor(randomForest_predictions), as.factor(test_data$Eligible))#
randomForest_matrix <- randomForest_cm$table
```

```r
# Extract TruePositive, TrueNegative, FalsePositive
# and FalseNegative for confusion matrix
randomForest_truePositive <- randomForest_matrix[1, 1]
randomForest_trueNegative <- randomForest_matrix[2, 2]
randomForest_falsePositive <- randomForest_matrix[1, 2]
randomForest_falseNegative <- randomForest_matrix[2, 1]
```

Calculate evaluation metrics

```r
# Calculate Evaluation Metrics
randomForest_accuracy <-
  round(((sum(diag(randomForest_matrix)) / sum(randomForest_matrix))) * 100, 2)
randomForest_precision <-
  round((randomForest_truePositive / (randomForest_truePositive +
                                        randomForest_falsePositive)) * 100, 2)
randomForest_recall <-
  round((randomForest_truePositive / (randomForest_truePositive +
                                        randomForest_falseNegative)) * 100, 2)
randomForest_f1_score <-
  round(2 * (randomForest_precision * randomForest_recall) /
          (randomForest_precision + randomForest_recall), 2)

cat("Random Forest Accuracy:", randomForest_accuracy, "% \n")
```

```
## Random Forest Accuracy: 96.09 %
```

```r
cat("Random Forest Precision:", randomForest_precision, "% \n")
```

```
## Random Forest Precision: 97.14 %
```

```r
cat("Random Forest Recall:", randomForest_recall, "% \n")
```

```
## Random Forest Recall: 91.7 %
```

```r
cat("Random Forest F1-score:", randomForest_f1_score, "% \n")
```

```
## Random Forest F1-score: 94.34 %
```

New percentage eligible customers

```r
# Count eligible customers
num_eligible_customers <- sum(randomForest_predictions == 0.743001202264081)

# Total number of customers
total_customers <- length(randomForest_predictions)

# Calculate the eligibility percentage
eligibility_percentage <- (num_eligible_customers / total_customers) * 100

cat("Percentage of eligible customers:",
    round(eligibility_percentage, 2), "%\n")
```

```
## Percentage of eligible customers: 66.42 %
```

Write evaluation metrics to "csv" file

```r
# Create a dataframe to store/save the evaluation metrics
accuracy_metrics <- data.frame(Accuracy = randomForest_accuracy,
                               Precision = randomForest_precision,
                               Recall = randomForest_recall,
                               F1_Score = randomForest_f1_score,
                               original_eligibility_rate = eligiblePercantageOriginal,
                               improved_eligibility_rate = eligibility_percentage )

# Write evaluation metrics dataframe to a csv file
write.csv(accuracy_metrics, "accuracy_metrics.csv")
```

Save model and pipeline for use in predictions

```r
# Save the model and preprocessing pipeline
saveRDS(randomForest_model, file = "random_forest_model.rds")
saveRDS(processed_data$pipeline, file = "preprocessing_pipeline.rds")
```