Exercise 6

Advanced Methods for Regression and Classification

Rita Selimi

27/11/2024

Predict Status

```r
# Load the Loan dataset
data("Loan", package = "ROCit")

# View the dataset's structure
str(Loan)
```

```
## 'data.frame':    900 obs. of  9 variables:
##  $ Amount : num  67.6 23 54 24.3 43.2 ...
##  $ Term   : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ IntRate: num  0.184 0.12 0.117 0.173 0.172 ...
##  $ ILR    : num  0.035 0.032 0.032 0.034 0.034 0.033 0.035 0.03 0.031 0.034 ...
##  $ EmpLen : Factor w/ 5 levels "A","B","C","D",..: 4 4 4 1 1 2 4 4 2 4 ...
##  $ Home   : Factor w/ 3 levels "MORTGAGE","OWN",..: 3 3 1 3 1 3 3 1 1 1 ...
##  $ Income : num  126400 30900 111900 66000 71900 ...
##  $ Status : Factor w/ 2 levels "CO","FP": 1 1 2 2 1 2 2 2 2 2 ...
##  $ Score  : num  201 180 162 197 203 ...
```

```r
head(Loan)
```

```
##   Amount Term IntRate   ILR EmpLen     Home Income Status    Score
## 1  67.57   36  0.1838 0.035      D     RENT 126400     CO 200.9581
## 2  22.97   36  0.1198 0.032      D     RENT  30900     CO 179.6058
## 3  54.05   36  0.1166 0.032      D MORTGAGE 111900     FP 161.7622
## 4  24.32   36  0.1733 0.034      A     RENT  66000     FP 196.6619
## 5  43.24   36  0.1723 0.034      A MORTGAGE  71900     CO 203.4912
## 6  16.22   36  0.1355 0.033      B     RENT  27614     FP 186.3070
```

```r
summary(Loan)
```

```
##      Amount           Term       IntRate            ILR           EmpLen
##  Min.   : 2.70   Min.   :36   Min.   :0.0830   Min.   :0.03000   A:198
##  1st Qu.:18.04   1st Qu.:36   1st Qu.:0.1219   1st Qu.:0.03200   B:198
##  Median :27.03   Median :36   Median :0.1513   Median :0.03300   C:141
##  Mean   :34.08   Mean   :36   Mean   :0.1529   Mean   :0.03353   D:305
##  3rd Qu.:43.34   3rd Qu.:36   3rd Qu.:0.1775   3rd Qu.:0.03500   U: 58
##  Max.   :94.59   Max.   :36   Max.   :0.2825   Max.   :0.04000
##        Home          Income       Status       Score
##  MORTGAGE:429   Min.   : 11900   CO:131   Min.   : -5.449
##  OWN     : 95   1st Qu.: 43900   FP:769   1st Qu.:169.358
##  RENT    :376   Median : 63000            Median :189.120
##                 Mean   : 72903            Mean   :187.440
##                 3rd Qu.: 86900            3rd Qu.:205.064
##                 Max.   :502000            Max.   :269.171
```

1. **`Amount`**: The loan amount in thousands
2. **`Term`**: The term of the loan in months
3. **`IntRate`**: The interest rate as a decimal
4. **`ILR`**: An index value representing the individual's loan rate
5. **`EmpLen`**: Employment length categorized into 5 levels
6. **`Home`**: Home ownership status categorized into 3 levels
7. **`Income`**: The individual's income in dollars
8. **`Status`**: Loan repayment status with two levels
9. **`Score`**: A score associated with the individual's loan profile

1. Linear Discriminant Analysis (LDA)

(a) Apply lda()

LDA (Linear Discriminant Analysis) is a classification method that finds the linear combination of predictors that best separates two or more classes. It assumes the data for each class is normally distributed and uses these linear combinations to assign new observations to one of the classes.

```r
# Ensure the response variable is a factor
Loan$Status <- as.factor(Loan$Status)

# Remove the constant variable 'Term'
Loan <- Loan[, !(names(Loan) %in% "Term")]
```

Preprocessing The response variable Status must be a factor because discriminant analysis methods (lda(), qda(), etc.) require categorical group labels for classification tasks.

The column Term has the same value for all observations within at least one group of Status. Since it doesn't vary, it provides no information to distinguish between classes. Constant variables cause an error in lda() because they make it impossible to calculate the covariance matrix.

Check for Missing Values

```r
# Check for missing values
sum(is.na(Loan))
```

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

Check for skewness and log transform highly skewed variables

```
# Load the necessary package
library(moments)

# Select numeric variables from the dataset
numeric_vars <- Loan[, sapply(Loan, is.numeric)]

# Compute skewness for each numeric variable
skewness_values <- sapply(numeric_vars, skewness)

# Display skewness values
skewness_values
```

```
##     Amount    IntRate        ILR     Income      Score
##  1.0948672  0.4465685  0.5098025  3.1582876 -0.6537420
```

```
Loan$Amount <- log(Loan$Amount + 1)
Loan$Income <- log(Loan$Income + 1)  # Add 1 to avoid issues with zero values
```

Scale Numerical Variables

```
numeric_vars <- sapply(Loan, is.numeric)
Loan[, numeric_vars] <- scale(Loan[, numeric_vars])

# Confirm scaling
summary(Loan)
```

```
##      Amount            IntRate               ILR           EmpLen        Home
##  Min.   :-3.2180   Min.   :-1.74047   Min.   :-1.8857   A:198   MORTGAGE:429
##  1st Qu.:-0.6568   1st Qu.:-0.77234   1st Qu.:-0.8167   B:198   OWN     : 95
##  Median :-0.0524   Median :-0.04065   Median :-0.2821   C:141   RENT    :376
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000   D:305
##  3rd Qu.: 0.6646   3rd Qu.: 0.61140   3rd Qu.: 0.7870   U: 58
##  Max.   : 1.8654   Max.   : 3.22460   Max.   : 3.4597
##      Income            Status       Score
##  Min.   :-3.161776   CO:131   Min.   :-6.58864
##  1st Qu.:-0.689563   FP:769   1st Qu.:-0.61764
##  Median :-0.005441            Median : 0.05737
##  Mean   : 0.000000            Mean   : 0.00000
##  3rd Qu.: 0.603695            3rd Qu.: 0.60199
##  Max.   : 3.925383            Max.   : 2.79173
```

LDA uses a covariance matrix, which is sensitive to the scale of the predictors. If one variable has a much larger range than another (e.g., Income vs. IntRate), it will dominate the covariance calculation. Scaling ensures that all variables contribute equally.

Encoding Categorical Variables

```
Loan$EmpLen <- as.numeric(Loan$EmpLen)
Loan$Home <- as.numeric(Loan$Home)
```

Checking Correlation

```r
library(caret)
```

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

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

```r
# Exclude `Status` from correlation analysis
numeric_vars <- Loan[, sapply(Loan, is.numeric) & names(Loan) != "Status"]

# Compute correlation matrix for numeric predictors
cor_matrix <- cor(numeric_vars)

# Find highly correlated features (cutoff = 0.9)
high_corr <- findCorrelation(cor_matrix, cutoff = 0.9)

# Remove highly correlated features from Loan dataset
Loan <- Loan[, -high_corr]
```

We are removing highly correlated predictors like `IntRate` to improve the stability of the LDA model by avoiding redundancy in the covariance matrix. The response variable `Status` is excluded from correlation analysis to ensure only predictors are evaluated.

```r


# Set a seed for reproducibility
set.seed(12332281)

# Split data into training and test sets
train_indices <- sample(1:nrow(Loan), size = 2/3 * nrow(Loan))
train_data <- Loan[train_indices, ]
test_data <- Loan[-train_indices, ]
```

Split the data

```r


# Load required library
library(MASS)

# Apply Linear Discriminant Analysis
lda_model <- lda(Status ~ ., data = train_data)

# Print LDA model details
lda_model
```

Apply lda()

```
## Call:
## lda(Status ~ ., data = train_data)
##
```

```
## Prior probabilities of groups:
##        CO        FP
## 0.1283333 0.8716667
##
## Group means:
##         Amount         ILR   EmpLen     Home      Income       Score
## CO -0.21474546  0.56481412 2.779221 2.064935 -0.38204786  0.55394112
## FP  0.04553681 -0.06135218 2.780115 1.959847  0.03682005 -0.04289813
##
## Coefficients of linear discriminants:
##                 LD1
## Amount -0.011518080
## ILR    -0.853284995
## EmpLen  0.049471596
## Home    0.126255364
## Income  0.474424482
## Score   0.009468245
```

1. **Prior Probabilities:**
   The model sees that the `FP` group dominates the training data, with about 87% of observations, while `CO` makes up only 13%.
2. **Group Means:**
   These are the average values of the predictors (`Amount`, `ILR`, etc.) for each class (`CO` and `FP`). For example:
   - `ILR` is higher for `CO` (0.564) compared to `FP` (-0.061), which may help the model differentiate between the two groups.
3. **Coefficients of Linear Discriminants:**
   These coefficients show how much each predictor contributes to separating the two groups. For example:
   - `ILR` has the largest coefficient (-0.853), making it the most important predictor for distinguishing `CO` from `FP`.
   - Other predictors like `Income` (0.474) also contribute but to a lesser extent.
   - Predictors with smaller coefficients, like `Amount` (-0.011), have little influence on classification.

b) Compute the evaluation measures

```r
# Predict on the training data
lda_predictions <- predict(lda_model, train_data)$class

# Create a confusion matrix
confusion_matrix <- table(Predicted = lda_predictions, Actual = train_data$Status)
print(confusion_matrix)
```

```
##          Actual
## Predicted  CO  FP
##        CO   2   1
##        FP  75 522
```

```r
# Calculate misclassification rate
misclassified <- sum(confusion_matrix) - sum(diag(confusion_matrix))  # FP + FN
total <- sum(confusion_matrix)  # FP + TN + FN + TP
misclassification_rate <- misclassified/total
cat("Misclassification Rate:", round(misclassification_rate, 4), "\n")
```

```
## Misclassification Rate: 0.1267
```

```r
# Compute TPR (True Positive Rate) for each class
true_positives_fp <- confusion_matrix["FP", "FP"]
true_positives_co <- confusion_matrix["CO", "CO"]
actual_fp <- sum(confusion_matrix[, "FP"])
actual_co <- sum(confusion_matrix[, "CO"])

tpr_fp <- true_positives_fp/actual_fp  # TPR for FP
tpr_co <- true_positives_co/actual_co  # TPR for CO

# Compute TNR (True Negative Rate) for each class
false_positives_fp <- confusion_matrix["FP", "CO"]
false_positives_co <- confusion_matrix["CO", "FP"]
true_negatives_fp <- confusion_matrix["CO", "CO"]
true_negatives_co <- confusion_matrix["FP", "FP"]

tnr_fp <- true_negatives_fp/(true_negatives_fp + false_positives_co)  # TNR for FP
tnr_co <- true_negatives_co/(true_negatives_co + false_positives_fp)  # TNR for CO

# Compute balanced accuracy
balanced_accuracy <- (tpr_fp + tnr_fp)/2
cat("Balanced Accuracy:", round(balanced_accuracy, 4), "\n")
```

```
## Balanced Accuracy: 0.8324
```

Confusion Matrix: `CO` is rarely predicted correctly (only **2 out of 77** true `CO` instances are classified as `CO`). `FP` is predicted correctly almost perfectly (**522 out of 523**).

3. **Misclassification Rate:**
   - Calculates the proportion of misclassified observations:

$$\text{Misclassification Rate} = \frac{\text{FP} + \text{FN}}{\text{Total Observations}}$$

   The model misclassifies about 12.67% of the observations, indicating it performs well overall but struggles with minority class `CO`.

4. **Balanced Accuracy:**
   - First, it computes:
     - **TPR:** The proportion of correctly classified instances of a given class.
     - **TNR:** The proportion of correctly identified negatives for a given class.
   - Then, averages these rates:

$$\text{Balanced Accuracy} = \frac{\text{TPR} + \text{TNR}}{2}$$

   This value reflects the model's poor performance on the minority class (`CO`), as it heavily favors the majority class (`FP`).

The model performs well overall, with a low misclassification rate of 12.67%. The balanced accuracy of 83.24% indicates that the model performs reasonably well, though it still struggles with the minority class (`CO`). While it correctly identifies nearly all `FP` instances, only 2 out of 77 `CO` instances are correctly classified. To further improve the classification of the minority class, techniques like oversampling, undersampling, or SMOTE could be considered.

c) Predict for test data

```r
# Predict on the test data
test_predictions <- predict(lda_model, test_data)$class

# Create a confusion matrix
test_confusion_matrix <- table(Predicted = test_predictions, Actual = test_data$Status)
print(test_confusion_matrix)
```

```
##          Actual
## Predicted  CO  FP
##        CO   0   2
##        FP  54 244
```

```r
# Calculate misclassification rate
test_misclassification_rate <- (test_confusion_matrix["CO", "FP"] + test_confusion_matrix["FP",
    "CO"])/sum(test_confusion_matrix)
cat("Misclassification Rate (Test):", round(test_misclassification_rate, 4), "\n")
```

```
## Misclassification Rate (Test): 0.1867
```

```r
# Compute TPR (True Positive Rate) for each class
true_positives_fp_test <- test_confusion_matrix["FP", "FP"]
true_positives_co_test <- test_confusion_matrix["CO", "CO"]
actual_fp_test <- sum(test_confusion_matrix[, "FP"])
actual_co_test <- sum(test_confusion_matrix[, "CO"])

tpr_fp_test <- true_positives_fp_test/actual_fp_test
tpr_co_test <- true_positives_co_test/actual_co_test

# Compute TNR (True Negative Rate) for each class
false_positives_fp_test <- test_confusion_matrix["FP", "CO"]
false_positives_co_test <- test_confusion_matrix["CO", "FP"]
tnr_co_test <- test_confusion_matrix["FP", "FP"]/(test_confusion_matrix["FP", "FP"] +
    false_positives_fp_test)
tnr_fp_test <- test_confusion_matrix["CO", "CO"]/(test_confusion_matrix["CO", "CO"] +
    false_positives_co_test)

# Compute balanced accuracy
balanced_accuracy_test <- (tpr_co_test + tnr_co_test)/2
cat("Balanced Accuracy (Test):", round(balanced_accuracy_test, 4), "\n")
```

```
## Balanced Accuracy (Test): 0.4094
```

The model performs poorly on the test data, with a misclassification rate of 18.67% and a low balanced accuracy of 40.94%. While it correctly classifies most `FP` instances (244 out of 246), it completely fails to classify the minority class `CO`, misclassifying all 54 `CO` instances as `FP`. This indicates significant bias toward the majority class and poor generalization for the minority class.

2. Maximize the balanced accuracy

a) Undersampling

Reduces the number of observations in the majority class by randomly selecting a smaller subset, so it matches the size of the minority class.

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:MASS':
##
##     select

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
# Count observations in each group
table(train_data$Status)
```

```
##
##  CO  FP
##  77 523
```

```r
# Identify the smaller group size
min_count <- min(table(train_data$Status))

# Perform undersampling
set.seed(12332281)  # For reproducibility
train_data_undersampled <- train_data %>%
    group_by(Status) %>%
    sample_n(min_count) %>%
    ungroup()

# Check the new class balance
table(train_data_undersampled$Status)
```

```
##
## CO FP
## 77 77
```

```r
# Apply Linear Discriminant Analysis
lda_model_undersampling <- lda(Status ~ ., data = train_data_undersampled)

# Print LDA model details
lda_model_undersampling
```

Apply lda()

```
## Call:
## lda(Status ~ ., data = train_data_undersampled)
##
## Prior probabilities of groups:
##  CO  FP
## 0.5 0.5
##
## Group means:
##         Amount          ILR   EmpLen     Home      Income      Score
## CO -0.21474546  0.564814117 2.779221 2.064935 -0.38204786 0.55394112
## FP  0.01116044 -0.004435208 2.701299 2.025974 -0.05650066 0.04410618
##
## Coefficients of linear discriminants:
##               LD1
## Amount  0.2366446
## ILR    -0.6001907
## EmpLen  0.0761196
## Home    0.2199860
## Income  0.2389983
## Score  -0.3510495
```

```r
# Predict on the training data
lda_predictions_undersampling <- predict(lda_model_undersampling, train_data_undersampled)$class

# Create a confusion matrix
confusion_matrix <- table(Predicted = lda_predictions_undersampling, Actual = train_data_undersampled$S
print(confusion_matrix)
```

Compute the evaluation measures

```
##          Actual
## Predicted CO FP
##        CO 46 24
##        FP 31 53
```

```r
# Calculate misclassification rate
misclassified <- sum(confusion_matrix) - sum(diag(confusion_matrix))  # FP + FN
total <- sum(confusion_matrix)  # FP + TN + FN + TP
misclassification_rate <- misclassified/total
cat("Misclassification Rate:", round(misclassification_rate, 4), "\n")
```

```
## Misclassification Rate: 0.3571
```

```
# Compute TPR (True Positive Rate) for each class
true_positives_fp <- confusion_matrix["FP", "FP"]
true_positives_co <- confusion_matrix["CO", "CO"]
actual_fp <- sum(confusion_matrix[, "FP"])
actual_co <- sum(confusion_matrix[, "CO"])

tpr_fp <- true_positives_fp/actual_fp  # TPR for FP
tpr_co <- true_positives_co/actual_co  # TPR for CO

# Compute TNR (True Negative Rate) for each class
false_positives_fp <- confusion_matrix["FP", "CO"]
false_positives_co <- confusion_matrix["CO", "FP"]
true_negatives_fp <- confusion_matrix["CO", "CO"]
true_negatives_co <- confusion_matrix["FP", "FP"]

tnr_fp <- true_negatives_fp/(true_negatives_fp + false_positives_co)  # TNR for FP
tnr_co <- true_negatives_co/(true_negatives_co + false_positives_fp)  # TNR for CO

# Compute balanced accuracy
balanced_accuracy <- (tpr_fp + tnr_fp)/2
cat("Balanced Accuracy:", round(balanced_accuracy, 4), "\n")
```

```
## Balanced Accuracy: 0.6727
```

```


# Predict on the test data
test_predictions_undersampling <- predict(lda_model_undersampling, test_data)$class

# Create a confusion matrix
test_confusion_matrix <- table(Predicted = test_predictions_undersampling, Actual = test_data$Status)
print(test_confusion_matrix)
```

Predict for test data

```
##          Actual
## Predicted  CO  FP
##        CO  27  85
##        FP  27 161
```

```
# Calculate misclassification rate
test_misclassification_rate <- (test_confusion_matrix["CO", "FP"] + test_confusion_matrix["FP",
    "CO"])/sum(test_confusion_matrix)
cat("Misclassification Rate (Test):", round(test_misclassification_rate, 4), "\n")
```

```
## Misclassification Rate (Test): 0.3733
```

```r
# Compute TPR (True Positive Rate) for each class
true_positives_fp_test <- test_confusion_matrix["FP", "FP"]
true_positives_co_test <- test_confusion_matrix["CO", "CO"]
actual_fp_test <- sum(test_confusion_matrix[, "FP"])
actual_co_test <- sum(test_confusion_matrix[, "CO"])

tpr_fp_test <- true_positives_fp_test/actual_fp_test
tpr_co_test <- true_positives_co_test/actual_co_test

# Compute TNR (True Negative Rate) for each class
false_positives_fp_test <- test_confusion_matrix["FP", "CO"]
false_positives_co_test <- test_confusion_matrix["CO", "FP"]
tnr_co_test <- test_confusion_matrix["FP", "FP"]/(test_confusion_matrix["FP", "FP"] +
    false_positives_fp_test)
tnr_fp_test <- test_confusion_matrix["CO", "CO"]/(test_confusion_matrix["CO", "CO"] +
    false_positives_co_test)

# Compute balanced accuracy
balanced_accuracy_test <- (tpr_co_test + tnr_co_test)/2
cat("Balanced Accuracy (Test):", round(balanced_accuracy_test, 4), "\n")
```

```
## Balanced Accuracy (Test): 0.6782
```

b) Oversampling

Increases the number of observations in the minority class by duplicating samples (with replacement), so it matches the size of the majority class.

```r
# Count observations in each group
table(train_data$Status)
```

```
##
##  CO  FP
##  77 523
```

```r
# Identify the larger group size
max_count <- max(table(train_data$Status))

# Perform oversampling
train_data_oversampling <- train_data %>%
    group_by(Status) %>%
    sample_n(max_count, replace = TRUE) %>%
    ungroup()

# Check the new class balance
table(train_data_oversampling$Status)
```

```
##
##  CO  FP
## 523 523
```

```r
# Apply Linear Discriminant Analysis
lda_model_oversampling <- lda(Status ~ ., data = train_data_oversampling)

# Print LDA model details
lda_model_oversampling
```

Apply lda()

```
## Call:
## lda(Status ~ ., data = train_data_oversampling)
##
## Prior probabilities of groups:
##  CO  FP
## 0.5 0.5
##
## Group means:
##        Amount        ILR   EmpLen     Home      Income      Score
## CO -0.2431118  0.5948123 2.797323 2.032505 -0.40584088  0.5733961
## FP  0.0293435 -0.1359628 2.833652 1.952199  0.04629917 -0.1238304
##
## Coefficients of linear discriminants:
##               LD1
## Amount  0.1521201
## ILR    -0.4585204
## EmpLen  0.1162207
## Home    0.1453752
## Income  0.1943125
## Score  -0.4894667
```

```r
# Predict on the training data
lda_predictions_oversampling <- predict(lda_model_oversampling, train_data_oversampling)$class

# Create a confusion matrix
confusion_matrix <- table(Predicted = lda_predictions_oversampling, Actual = train_data_oversampling$Sta
print(confusion_matrix)
```

Compute the evaluation measures

```
##          Actual
## Predicted  CO  FP
##        CO 329 171
##        FP 194 352
```

```r
# Calculate misclassification rate
misclassified <- sum(confusion_matrix) - sum(diag(confusion_matrix))  # FP + FN
total <- sum(confusion_matrix)  # FP + TN + FN + TP
misclassification_rate <- misclassified/total
cat("Misclassification Rate:", round(misclassification_rate, 4), "\n")
```

```
## Misclassification Rate: 0.3489
```

```
# Compute TPR (True Positive Rate) for each class
true_positives_fp <- confusion_matrix["FP", "FP"]
true_positives_co <- confusion_matrix["CO", "CO"]
actual_fp <- sum(confusion_matrix[, "FP"])
actual_co <- sum(confusion_matrix[, "CO"])

tpr_fp <- true_positives_fp/actual_fp  # TPR for FP
tpr_co <- true_positives_co/actual_co  # TPR for CO

# Compute TNR (True Negative Rate) for each class
false_positives_fp <- confusion_matrix["FP", "CO"]
false_positives_co <- confusion_matrix["CO", "FP"]
true_negatives_fp <- confusion_matrix["CO", "CO"]
true_negatives_co <- confusion_matrix["FP", "FP"]

tnr_fp <- true_negatives_fp/(true_negatives_fp + false_positives_co)  # TNR for FP
tnr_co <- true_negatives_co/(true_negatives_co + false_positives_fp)  # TNR for CO

# Compute balanced accuracy
balanced_accuracy <- (tpr_fp + tnr_fp)/2
cat("Balanced Accuracy:", round(balanced_accuracy, 4), "\n")
```

```
## Balanced Accuracy: 0.6655
```

```


# Predict on the test data
test_predictions_oversampling <- predict(lda_model_oversampling, test_data)$class

# Create a confusion matrix
test_confusion_matrix <- table(Predicted = test_predictions_oversampling, Actual = test_data$Status)
print(test_confusion_matrix)
```

Predict for test data

```
##          Actual
## Predicted  CO  FP
##        CO  28  87
##        FP  26 159
```

```
# Calculate misclassification rate
test_misclassification_rate <- (test_confusion_matrix["CO", "FP"] + test_confusion_matrix["FP",
    "CO"])/sum(test_confusion_matrix)
cat("Misclassification Rate (Test):", round(test_misclassification_rate, 4), "\n")
```

```
## Misclassification Rate (Test): 0.3767
```

```r
# Compute TPR (True Positive Rate) for each class
true_positives_fp_test <- test_confusion_matrix["FP", "FP"]
true_positives_co_test <- test_confusion_matrix["CO", "CO"]
actual_fp_test <- sum(test_confusion_matrix[, "FP"])
actual_co_test <- sum(test_confusion_matrix[, "CO"])

tpr_fp_test <- true_positives_fp_test/actual_fp_test
tpr_co_test <- true_positives_co_test/actual_co_test

# Compute TNR (True Negative Rate) for each class
false_positives_fp_test <- test_confusion_matrix["FP", "CO"]
false_positives_co_test <- test_confusion_matrix["CO", "FP"]
tnr_co_test <- test_confusion_matrix["FP", "FP"]/(test_confusion_matrix["FP", "FP"] +
    false_positives_fp_test)
tnr_fp_test <- test_confusion_matrix["CO", "CO"]/(test_confusion_matrix["CO", "CO"] +
    false_positives_co_test)

# Compute balanced accuracy
balanced_accuracy_test <- (tpr_co_test + tnr_co_test)/2
cat("Balanced Accuracy (Test):", round(balanced_accuracy_test, 4), "\n")
```

```
## Balanced Accuracy (Test): 0.689
```

Training Data

Metric	**Undersampling**	**Oversampling**	**Better Strategy**
Misclassification Rate	35.71%	37.09%	**Undersampling**
Balanced Accuracy	67.27%	64.24%	**Undersampling**

- **Undersampling:** Achieves a slightly better misclassification rate and higher balanced accuracy on the training data.

Test Data

Metric	**Undersampling**	**Oversampling**	**Better Strategy**
Misclassification Rate	37.33%	40.67%	**Undersampling**
Balanced Accuracy	67.82%	61.86%	**Undersampling**

- **Undersampling:** Performs better on test data, with both a lower misclassification rate and higher balanced accuracy.

Undersampling is the more successful strategy. It achieves better results on both the training and test datasets, with a higher balanced accuracy and a lower misclassification rate. While undersampling risks losing information from the majority class, it avoids overfitting, making it more effective for this dataset. Oversampling, on the other hand, struggles with overfitting due to repeated samples in the minority class, leading to poorer generalization on the test data.

3. Quadratic Discriminant Analysis (QDA)

Quadratic Discriminant Analysis (QDA) is a classification method that models each class with its own covariance matrix, allowing for more flexibility than LDA by capturing nonlinear decision boundaries.

a) Undersampling

```r
library(MASS)

# Train QDA on the undersampled training data
qda_model_undersampling <- qda(Status ~ ., data = train_data_undersampled)

# Print QDA model details
qda_model_undersampling
```

Apply qda()

```
## Call:
## qda(Status ~ ., data = train_data_undersampled)
##
## Prior probabilities of groups:
##  CO  FP
## 0.5 0.5
##
## Group means:
##         Amount          ILR   EmpLen     Home      Income      Score
## CO -0.21474546  0.564814117 2.779221 2.064935 -0.38204786 0.55394112
## FP  0.01116044 -0.004435208 2.701299 2.025974 -0.05650066 0.04410618
```

```r
# Predict on the training data
qda_predictions_undersampling <- predict(qda_model_undersampling, train_data_undersampled)$class

# Create a confusion matrix
confusion_matrix <- table(Predicted = qda_predictions_undersampling, Actual = train_data_undersampled$S
print(confusion_matrix)
```

Compute the evaluation measures

```
##          Actual
## Predicted CO FP
##        CO 48 24
##        FP 29 53
```

```r
# Calculate misclassification rate
misclassified <- sum(confusion_matrix) - sum(diag(confusion_matrix))  # FP + FN
total <- sum(confusion_matrix)  # FP + TN + FN + TP
misclassification_rate <- misclassified/total
cat("Misclassification Rate:", round(misclassification_rate, 4), "\n")
```

```
## Misclassification Rate: 0.3442
```

```r
# Compute TPR (True Positive Rate) for each class
true_positives_fp <- confusion_matrix["FP", "FP"]
true_positives_co <- confusion_matrix["CO", "CO"]
actual_fp <- sum(confusion_matrix[, "FP"])
actual_co <- sum(confusion_matrix[, "CO"])

tpr_fp <- true_positives_fp/actual_fp  # TPR for FP
tpr_co <- true_positives_co/actual_co  # TPR for CO

# Compute TNR (True Negative Rate) for each class
false_positives_fp <- confusion_matrix["FP", "CO"]
false_positives_co <- confusion_matrix["CO", "FP"]
true_negatives_fp <- confusion_matrix["CO", "CO"]
true_negatives_co <- confusion_matrix["FP", "FP"]

tnr_fp <- true_negatives_fp/(true_negatives_fp + false_positives_co)  # TNR for FP
tnr_co <- true_negatives_co/(true_negatives_co + false_positives_fp)  # TNR for CO

# Compute balanced accuracy
balanced_accuracy <- (tpr_fp + tnr_fp)/2
cat("Balanced Accuracy:", round(balanced_accuracy, 4), "\n")
```

```
## Balanced Accuracy: 0.6775
```

```r


# Predict on the test data
test_predictions_undercampling <- predict(qda_model_undersampling, test_data)$class

# Create a confusion matrix
test_confusion_matrix <- table(Predicted = test_predictions_undercampling, Actual = test_data$Status)
print(test_confusion_matrix)
```

Predict for test data

```
##          Actual
## Predicted  CO  FP
##        CO  27  98
##        FP  27 148
```

```r
# Calculate misclassification rate
test_misclassification_rate <- (test_confusion_matrix["CO", "FP"] + test_confusion_matrix["FP",
    "CO"])/sum(test_confusion_matrix)
cat("Misclassification Rate (Test):", round(test_misclassification_rate, 4), "\n")
```

```
## Misclassification Rate (Test): 0.4167
```

```r
# Compute TPR (True Positive Rate) for each class
true_positives_fp_test <- test_confusion_matrix["FP", "FP"]
true_positives_co_test <- test_confusion_matrix["CO", "CO"]
actual_fp_test <- sum(test_confusion_matrix[, "FP"])
```

```r
actual_co_test <- sum(test_confusion_matrix[, "CO"])

tpr_fp_test <- true_positives_fp_test/actual_fp_test
tpr_co_test <- true_positives_co_test/actual_co_test

# Compute TNR (True Negative Rate) for each class
false_positives_fp_test <- test_confusion_matrix["FP", "CO"]
false_positives_co_test <- test_confusion_matrix["CO", "FP"]
tnr_co_test <- test_confusion_matrix["FP", "FP"]/(test_confusion_matrix["FP", "FP"] +
    false_positives_fp_test)
tnr_fp_test <- test_confusion_matrix["CO", "CO"]/(test_confusion_matrix["CO", "CO"] +
    false_positives_co_test)

# Compute balanced accuracy
balanced_accuracy_test <- (tpr_co_test + tnr_co_test)/2
cat("Balanced Accuracy (Test):", round(balanced_accuracy_test, 4), "\n")
```

```
## Balanced Accuracy (Test): 0.6729
```

b) Oversampling

```r
# Train QDA on the oversampling training data
qda_model_oversampling <- qda(Status ~ ., data = train_data_oversampling)

# Print QDA model details
qda_model_oversampling
```

Apply qda()

```
## Call:
## qda(Status ~ ., data = train_data_oversampling)
##
## Prior probabilities of groups:
##  CO  FP
## 0.5 0.5
##
## Group means:
##        Amount        ILR   EmpLen     Home      Income      Score
## CO -0.2431118  0.5948123 2.797323 2.032505 -0.40584088  0.5733961
## FP  0.0293435 -0.1359628 2.833652 1.952199  0.04629917 -0.1238304
```

```r
# Predict on the training data
qda_predictions_oversampling <- predict(qda_model_oversampling, train_data_oversampling)$class

# Create a confusion matrix
confusion_matrix <- table(Predicted = qda_predictions_oversampling, Actual = train_data_oversampling$Sta
print(confusion_matrix)
```

Compute the evaluation measures

```
##          Actual
## Predicted  CO  FP
##        CO 365 184
##        FP 158 339
```

```r
# Calculate misclassification rate
misclassified <- sum(confusion_matrix) - sum(diag(confusion_matrix))  # FP + FN
total <- sum(confusion_matrix)  # FP + TN + FN + TP
misclassification_rate <- misclassified/total
cat("Misclassification Rate:", round(misclassification_rate, 4), "\n")
```

```
## Misclassification Rate: 0.327
```

```r
# Compute TPR (True Positive Rate) for each class
true_positives_fp <- confusion_matrix["FP", "FP"]
true_positives_co <- confusion_matrix["CO", "CO"]
actual_fp <- sum(confusion_matrix[, "FP"])
actual_co <- sum(confusion_matrix[, "CO"])

tpr_fp <- true_positives_fp/actual_fp  # TPR for FP
tpr_co <- true_positives_co/actual_co  # TPR for CO

# Compute TNR (True Negative Rate) for each class
false_positives_fp <- confusion_matrix["FP", "CO"]
false_positives_co <- confusion_matrix["CO", "FP"]
true_negatives_fp <- confusion_matrix["CO", "CO"]
true_negatives_co <- confusion_matrix["FP", "FP"]

tnr_fp <- true_negatives_fp/(true_negatives_fp + false_positives_co)  # TNR for FP
tnr_co <- true_negatives_co/(true_negatives_co + false_positives_fp)  # TNR for CO

# Compute balanced accuracy
balanced_accuracy <- (tpr_fp + tnr_fp)/2
cat("Balanced Accuracy:", round(balanced_accuracy, 4), "\n")
```

```
## Balanced Accuracy: 0.6565
```

```r
# Predict on the test data
test_predictions_oversampling <- predict(qda_model_oversampling, test_data)$class

# Create a confusion matrix
test_confusion_matrix <- table(Predicted = test_predictions_oversampling, Actual = test_data$Status)
print(test_confusion_matrix)
```

Predict for test data

```
##          Actual
## Predicted  CO  FP
##        CO  28  96
##        FP  26 150
```

```r
# Calculate misclassification rate
test_misclassification_rate <- (test_confusion_matrix["CO", "FP"] + test_confusion_matrix["FP",
    "CO"])/sum(test_confusion_matrix)
cat("Misclassification Rate (Test):", round(test_misclassification_rate, 4), "\n")
```

```
## Misclassification Rate (Test): 0.4067
```

```r
# Compute TPR (True Positive Rate) for each class
true_positives_fp_test <- test_confusion_matrix["FP", "FP"]
true_positives_co_test <- test_confusion_matrix["CO", "CO"]
actual_fp_test <- sum(test_confusion_matrix[, "FP"])
actual_co_test <- sum(test_confusion_matrix[, "CO"])

tpr_fp_test <- true_positives_fp_test/actual_fp_test
tpr_co_test <- true_positives_co_test/actual_co_test

# Compute TNR (True Negative Rate) for each class
false_positives_fp_test <- test_confusion_matrix["FP", "CO"]
false_positives_co_test <- test_confusion_matrix["CO", "FP"]
tnr_co_test <- test_confusion_matrix["FP", "FP"]/(test_confusion_matrix["FP", "FP"] +
    false_positives_fp_test)
tnr_fp_test <- test_confusion_matrix["CO", "CO"]/(test_confusion_matrix["CO", "CO"] +
    false_positives_co_test)

# Compute balanced accuracy
balanced_accuracy_test <- (tpr_co_test + tnr_co_test)/2
cat("Balanced Accuracy (Test):", round(balanced_accuracy_test, 4), "\n")
```

```
## Balanced Accuracy (Test): 0.6854
```

Test Data

Metric	**Undersampling**	**Oversampling**	**Better Strategy**
Misclassification Rate	41.67%	**40.67%**	**Oversampling**
Balanced Accuracy	67.29%	**68.54%**	**Oversampling**

- **Oversampling:** Performs slightly better on the test set, with a lower misclassification rate and higher balanced accuracy. It generalizes better than undersampling in this case.

For QDA, **oversampling** performs slightly better than undersampling on the test set, achieving a lower misclassification rate (40.67% vs. 41.67%) and a higher balanced accuracy (68.54% vs. 67.29%). This suggests that oversampling helps the model generalize better to unseen data by providing more examples of the minority class.

4. Regularized Discrimant Analysis (RDA)

Regularized Discriminant Analysis (RDA) is a classification method that combines Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) by introducing regularization parameters to balance between the two, improving flexibility and stability for datasets with multicollinearity or small sample sizes.

a) Undersampling

```
library(klaR)

# Train RDA on the undersampled training data
rda_model_undersampling <- rda(Status ~ ., data = train_data_undersampled)

rda_model_undersampling
```

Apply rda()

```
## Call:
## rda(formula = Status ~ ., data = train_data_undersampled)
##
## Regularization parameters:
##      gamma     lambda
## 0.22584746 0.07727248
##
## Prior probabilities of groups:
##  CO  FP
## 0.5 0.5
##
## Misclassification rate:
##        apparent: 35.714 %
## cross-validated: 38.085 %
```

```
# Predict on the test data
test_predictions_undercampling <- predict(rda_model_undersampling, test_data)$class

# Create a confusion matrix
test_confusion_matrix <- table(Predicted = test_predictions_undercampling, Actual = test_data$Status)
print(test_confusion_matrix)
```

Predict for test data

```
##          Actual
## Predicted  CO  FP
##        CO  26  73
##        FP  28 173
```

```
# Calculate misclassification rate
test_misclassification_rate <- (test_confusion_matrix["CO", "FP"] + test_confusion_matrix["FP",
    "CO"])/sum(test_confusion_matrix)
cat("Misclassification Rate (Test):", round(test_misclassification_rate, 4), "\n")
```

```
## Misclassification Rate (Test): 0.3367
```

```
# Compute TPR (True Positive Rate) for each class
true_positives_fp_test <- test_confusion_matrix["FP", "FP"]
true_positives_co_test <- test_confusion_matrix["CO", "CO"]
actual_fp_test <- sum(test_confusion_matrix[, "FP"])
actual_co_test <- sum(test_confusion_matrix[, "CO"])

tpr_fp_test <- true_positives_fp_test/actual_fp_test
tpr_co_test <- true_positives_co_test/actual_co_test

# Compute TNR (True Negative Rate) for each class
false_positives_fp_test <- test_confusion_matrix["FP", "CO"]
false_positives_co_test <- test_confusion_matrix["CO", "FP"]
tnr_co_test <- test_confusion_matrix["FP", "FP"]/(test_confusion_matrix["FP", "FP"] +
    false_positives_fp_test)
tnr_fp_test <- test_confusion_matrix["CO", "CO"]/(test_confusion_matrix["CO", "CO"] +
    false_positives_co_test)

# Compute balanced accuracy
balanced_accuracy_test <- (tpr_co_test + tnr_co_test)/2
cat("Balanced Accuracy (Test):", round(balanced_accuracy_test, 4), "\n")
```

```
## Balanced Accuracy (Test): 0.6711
```

b) Oversampling

```


rda_model_oversampling <- rda(Status ~ ., data = train_data_oversampling)

rda_model_oversampling
```

Apply rda()

```
## Call:
## rda(formula = Status ~ ., data = train_data_oversampling)
##
## Regularization parameters:
##     gamma    lambda
## 0.4231200 0.9488401
##
## Prior probabilities of groups:
##  CO  FP
## 0.5 0.5
##
```

```
## Misclassification rate:
##        apparent: 31.931 %
## cross-validated: 32.142 %
```

```r
# Predict on the test data
test_predictions_oversampling <- predict(rda_model_oversampling, test_data)$class

# Create a confusion matrix
test_confusion_matrix <- table(Predicted = test_predictions_oversampling, Actual = test_data$Status)
print(test_confusion_matrix)
```

Predict for test data

```
##          Actual
## Predicted  CO  FP
##        CO  29  86
##        FP  25 160
```

```r
# Calculate misclassification rate
test_misclassification_rate <- (test_confusion_matrix["CO", "FP"] + test_confusion_matrix["FP",
    "CO"])/sum(test_confusion_matrix)
cat("Misclassification Rate (Test):", round(test_misclassification_rate, 4), "\n")
```

```
## Misclassification Rate (Test): 0.37
```

```r
# Compute TPR (True Positive Rate) for each class
true_positives_fp_test <- test_confusion_matrix["FP", "FP"]
true_positives_co_test <- test_confusion_matrix["CO", "CO"]
actual_fp_test <- sum(test_confusion_matrix[, "FP"])
actual_co_test <- sum(test_confusion_matrix[, "CO"])

tpr_fp_test <- true_positives_fp_test/actual_fp_test
tpr_co_test <- true_positives_co_test/actual_co_test

# Compute TNR (True Negative Rate) for each class
false_positives_fp_test <- test_confusion_matrix["FP", "CO"]
false_positives_co_test <- test_confusion_matrix["CO", "FP"]
tnr_co_test <- test_confusion_matrix["FP", "FP"]/(test_confusion_matrix["FP", "FP"] +
    false_positives_fp_test)
tnr_fp_test <- test_confusion_matrix["CO", "CO"]/(test_confusion_matrix["CO", "CO"] +
    false_positives_co_test)

# Compute balanced accuracy
balanced_accuracy_test <- (tpr_co_test + tnr_co_test)/2
cat("Balanced Accuracy (Test):", round(balanced_accuracy_test, 4), "\n")
```

```
## Balanced Accuracy (Test): 0.701
```

Undersampling Results - Tuning Parameters (Gamma: 0.2258, Lambda: 0.0773):
- A **lower gamma** indicates that the model relies more on Quadratic Discriminant Analysis (QDA), allowing

for non-linear decision boundaries.
- A **lower lambda** suggests that the covariance matrices for the classes are less regularized, meaning the model assumes more flexibility in the spread of the data.

The model performs reasonably well, but the balanced accuracy suggests it struggles slightly with classifying both classes equally.

Oversampling Results - Tuning Parameters (Gamma: 0.4231, Lambda: 0.9488):
- A **higher gamma** indicates the model incorporates more Linear Discriminant Analysis (LDA), favoring linear decision boundaries.
- A **higher lambda** suggests strong regularization of the covariance matrices, making the model more stable and less sensitive to outliers or noise.

The model achieves higher balanced accuracy but has a slightly higher misclassification rate.

- **Oversampling** is preferable due to its higher balanced accuracy, which aligns with the goal of equal performance across both classes. The high lambda ensures model stability, while gamma's balance between LDA and QDA avoids overfitting to the minority class.