Exercise 6

for Advanced Methods for Regression and Classification

Dzhamilia Kulikieva

26.11.2024

1. Linear Discrimant Analysis (LDA)

(a) Preprocessing and applying LDA

Just like in the previous task, we will convert the variable Status, EmpLen and Home to a numeric values to simplify modeling for LDA

```r
library(ROCit)
data(Loan)
# Convert Status to a numeric variable (0 and 1)
Loan$Status <- as.numeric(Loan$Status) - 1

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

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 : num  4 4 4 1 1 2 4 4 2 4 ...
##  $ Home   : num  3 3 1 3 1 3 3 1 1 1 ...
##  $ Income : num  126400 30900 111900 66000 71900 ...
##  $ Status : num  0 0 1 1 0 1 1 1 1 1 ...
##  $ Score  : num  201 180 162 197 203 ...
```

As column Term has only one unique value we will delete it because it doesn't contain any useful information for us.

```r
Loan$Term <- NULL
```

We should normalize the following variables: Amount, IntRate, ILR, Income, Score as they have a wide range of values and LDA is sensitive to the scale of variables.

We will use Min-Max normalization:

```r
normalize <- function(x) {
  return((x-min(x)) / (max(x) - min(x)))
}

Loan$Amount <- normalize(Loan$Amount)
Loan$IntRate <- normalize(Loan$IntRate)
```

```r
Loan$ILR <- normalize(Loan$ILR)
Loan$Income <- normalize(Loan$Income)
Loan$Score <- normalize(Loan$Score)
```

```r
# Split the data
set.seed(1234)
train_indices <- sample(1:nrow(Loan), size = 2/3*nrow(Loan))
train_data <- Loan[train_indices, ]
test_data <- Loan[-train_indices, ]
```

```r
library(MASS)

lda_model <- lda(Status ~., data = train_data)
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```r
summary(lda_model)
```

```
##         Length Class  Mode
## prior    2     -none- numeric
## counts   2     -none- numeric
## means   14     -none- numeric
## scaling  7     -none- numeric
## lev      2     -none- character
## svd      1     -none- numeric
## N        1     -none- numeric
## call     3     -none- call
## terms    3     terms  call
## xlevels  0     -none- list
```

The model warns about the collinearity of variables, which means that some variables are strongly correlated with each other.

```r
cor(train_data[, -which(names(train_data) == "Status")])
```

```
##              Amount     IntRate         ILR      EmpLen        Home      Income
## Amount   1.00000000 -0.15993023 -0.16441926  0.06589331 -0.19758941  0.48033150
## IntRate -0.15993023  1.00000000  0.98697427  0.03544263  0.10127521 -0.23009324
## ILR     -0.16441926  0.98697427  1.00000000  0.03797526  0.10221584 -0.22898904
## EmpLen   0.06589331  0.03544263  0.03797526  1.00000000 -0.18984571  0.03246153
## Home    -0.19758941  0.10127521  0.10221584 -0.18984571  1.00000000 -0.15485254
## Income   0.48033150 -0.23009324 -0.22898904  0.03246153 -0.15485254  1.00000000
## Score   -0.04583929  0.83084959  0.81874084  0.03071971  0.09912161 -0.62587370
##               Score
## Amount  -0.04583929
## IntRate  0.83084959
## ILR      0.81874084
## EmpLen   0.03071971
## Home     0.09912161
## Income  -0.62587370
## Score    1.00000000
```

For example, IntRate and ILR with a correlation of 0.9869. This can lead to problems with multicollinearity, which will make it difficult to correctly distribute weights between these features and, as a result, may degrade the performance of the model. We can simply exclude one of these features from our dataset before training the model.

```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
# Deleting IntRate
train_data <- train_data %>% select(-IntRate)
```

```r
library(MASS)

lda_model <- lda(Status ~., data = train_data)
summary(lda_model)
```

```
##         Length Class  Mode
## prior    2     -none- numeric
## counts   2     -none- numeric
## means   12     -none- numeric
## scaling  6     -none- numeric
## lev      2     -none- character
## svd      1     -none- numeric
## N        1     -none- numeric
## call     3     -none- call
## terms    3     terms  call
## xlevels  0     -none- list
```

(b) Computation of the evaluation measures for the training data

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


# Confusion matrix
conf_matrix_train <- table(Predicted = train_predictions, Actual = train_data$Status)
print(conf_matrix_train)
```

```
##          Actual
## Predicted   0   1
##         0   3   1
##         1  83 513
```

```r
# Compute evaluation metrics
TP <- conf_matrix_train[2, 2]
TN <- conf_matrix_train[1, 1]
FP <- conf_matrix_train[2, 1]
FN <- conf_matrix_train[1, 2]
```

```
misclassification_rate <- (FP + FN) / sum(conf_matrix_train)
TPR <- TP / (TP + FN) # Sensitivity
TNR <- TN / (TN + FP) # Specificity
balanced_accuracy <- (TPR + TNR) / 2

list(
  MisclassificationRate = misclassification_rate,
  BalancedAccuracy = balanced_accuracy
)
```

```
## $MisclassificationRate
## [1] 0.14
##
## $BalancedAccuracy
## [1] 0.5164691
```

Misclassification Rate (0.14) indicates that the model has an error in 14% of predictions, which is a reasonably good result in most cases.

Balanced Accuracy (0.516) suggests that the model is somewhat performing well, but there is room for improvement. If the classes are highly imbalanced, this metric reflects that the model might not be doing well with the minority class.

(c) Prediction of the group membership for the test data and the evaluation measures.

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

# Confusion matrix for test data
conf_matrix_test <- table(Predicted = test_predictions, Actual = test_data$Status)
print(conf_matrix_test)
```

```
##          Actual
## Predicted   0   1
##         0   1   2
##         1  44 253
```

```
# Compute evaluation metrics
TP <- conf_matrix_test[2, 2]
TN <- conf_matrix_test[1, 1]
FP <- conf_matrix_test[2, 1]
FN <- conf_matrix_test[1, 2]

misclassification_rate_test <- (FP + FN) / sum(conf_matrix_test)
TPR <- TP / (TP + FN) # Sensitivity
TNR <- TN / (TN + FP) # Specificity
balanced_accuracy_test <- (TPR + TNR) / 2

list(
  MisclassificationRate = misclassification_rate_test,
  BalancedAccuracy = balanced_accuracy_test
)
```

```
## $MisclassificationRate
## [1] 0.1533333
##
## $BalancedAccuracy
## [1] 0.5071895
```

Misclassification Rate on the test data (0.1533) is slightly higher than on the training data (0.14), which is common and could indicate overfitting. This suggests that the model is better tailored to the training data and may not generalize as well to new data.

Balanced Accuracy on the test data (0.5072) is a bit lower than on the training data (0.5165), which also points to some performance drop on test data. However, the results are still fairly close. It's important to note that balanced accuracy is less sensitive to class imbalance than regular accuracy.

2. Comparison of Data Balancing Methods: Undersampling vs. Oversampling

(a) Undersampling

Let's identify the number of samples in each group (n1, n2).

```r
n1 <- sum(train_data$Status == 0)
n2 <- sum(train_data$Status == 1)
min_n <- min(n1, n2)

# Randomly sample min(n1, n2) observations from each group
set.seed(123) # For reproducibility
group_0 <- train_data[train_data$Status == 0, ]
group_1 <- train_data[train_data$Status == 1, ]

undersampled_0 <- group_0[sample(nrow(group_0), min_n), ]
undersampled_1 <- group_1[sample(nrow(group_1), min_n), ]

balanced_train <- rbind(undersampled_0, undersampled_1)
```

Train the LDA model on the balanced training set and compute evaluation metrics on the unchanged test set:

```r
library(MASS)
lda_model_undersampled <- lda(Status ~ ., data = balanced_train)

# Predictions on training set
train_pred <- predict(lda_model_undersampled, newdata = balanced_train)$class



# Calculate metrics



# Confusion matrix
conf_train <- table(Predicted = train_pred, Actual = balanced_train$Status)
print(conf_train)
```

```
##          Actual
## Predicted  0  1
##         0 58 30
##         1 28 56
```

```r
# Compute evaluation metrics
TP <- conf_train[2, 2]
TN <- conf_train[1, 1]
FP <- conf_train[2, 1]
FN <- conf_train[1, 2]

Train_misclassification_rate <- (FP + FN) / sum(conf_train)
TPR <- TP / (TP + FN) # Sensitivity
TNR <- TN / (TN + FP) # Specificity
Train_balanced_accuracy <- (TPR + TNR) / 2

list(
  MisclassificationRate = Train_misclassification_rate,
  BalancedAccuracy = Train_balanced_accuracy
)
```

```
## $MisclassificationRate
## [1] 0.3372093
##
## $BalancedAccuracy
## [1] 0.6627907
```

```r
# Predictions on test set
test_pred <- predict(lda_model_undersampled, newdata = test_data)$class


# Confusion matrix for test data
conf_test <- table(Predicted = test_pred, Actual = test_data$Status)
print(conf_test)
```

```
##          Actual
## Predicted   0   1
##         0  26 106
##         1  19 149
```

```r
# Compute evaluation metrics
TP <- conf_test[2, 2] # True Positives
TN <- conf_test[1, 1] # True Negatives
FP <- conf_test[2, 1] # False Positives
FN <- conf_test[1, 2] # False Negatives

misclassification_rate_test <- (FP + FN) / sum(conf_test) # Misclassification Rate
TPR <- TP / (TP + FN)  # Sensitivity or True Positive Rate
TNR <- TN / (TN + FP)  # Specificity or True Negative Rate
balanced_accuracy_test <- (TPR + TNR) / 2 # Balanced Accuracy

# Print the results
list(
  MisclassificationRate = misclassification_rate_test,
  BalancedAccuracy = balanced_accuracy_test
)
```

```
## $MisclassificationRate
## [1] 0.4166667
##
## $BalancedAccuracy
## [1] 0.5810458
```

(b) Oversampling

Let's identify the number of samples in each group (n1, n2)

```
max_n <- max(n1, n2)

# Randomly sample with replacement from the smaller group to increase its size to max(n1, n2)
set.seed(123) # For reproducibility
oversampled_0 <- group_0[sample(nrow(group_0), max_n, replace = TRUE), ]
oversampled_1 <- group_1[sample(nrow(group_1), max_n, replace = TRUE), ]

balanced_train2 <- rbind(oversampled_0, oversampled_1)
```

Train the LDA model on the balanced training set and compute evaluation metrics on the unchanged test set (same as in (a)).

```
lda_model_oversampled <- lda(Status ~ ., data = balanced_train2)

# Predictions on training set
train_pred2 <- predict(lda_model_oversampled, newdata = balanced_train2)$class


# Calculate metrics
# Confusion matrix
conf_train2 <- table(Predicted = train_pred2, Actual = balanced_train2$Status)
print(conf_train2)
```

```
##          Actual
## Predicted   0   1
##         0 336 199
##         1 178 315
```

```
# Compute evaluation metrics
TP <- conf_train2[2, 2]
TN <- conf_train2[1, 1]
FP <- conf_train2[2, 1]
FN <- conf_train2[1, 2]

Train_misclassification_rate <- (FP + FN) / sum(conf_train2)
TPR <- TP / (TP + FN) # Sensitivity
TNR <- TN / (TN + FP) # Specificity
Train_balanced_accuracy <- (TPR + TNR) / 2

list(
  MisclassificationRate = Train_misclassification_rate,
  BalancedAccuracy = Train_balanced_accuracy
)
```

```
## $MisclassificationRate
## [1] 0.3667315
##
## $BalancedAccuracy
## [1] 0.6332685
```

```r
# Predictions on test set
test_pred2 <- predict(lda_model_oversampled, newdata = test_data)$class


# Confusion matrix for test data
conf_test2 <- table(Predicted = test_pred2, Actual = test_data$Status)
print(conf_test)
```

```
##          Actual
## Predicted   0   1
##         0  26 106
##         1  19 149
```

```r
# Compute evaluation metrics
TP <- conf_test2[2, 2] # True Positives
TN <- conf_test2[1, 1] # True Negatives
FP <- conf_test2[2, 1] # False Positives
FN <- conf_test2[1, 2] # False Negatives

misclassification_rate_test <- (FP + FN) / sum(conf_test2) # Misclassification Rate
TPR <- TP / (TP + FN)  # Sensitivity or True Positive Rate
TNR <- TN / (TN + FP)  # Specificity or True Negative Rate
balanced_accuracy_test <- (TPR + TNR) / 2 # Balanced Accuracy

# Print the results
list(
  MisclassificationRate = misclassification_rate_test,
  BalancedAccuracy = balanced_accuracy_test
)
```

```
## $MisclassificationRate
## [1] 0.4233333
##
## $BalancedAccuracy
## [1] 0.5496732
```

Undersampling appears to be more successful because it achieves higher balanced accuracy on both the training and test data compared to oversampling. The lower misclassification rate in the undersampling model also suggests that the model trained on a balanced dataset performs better at distinguishing between classes, especially when evaluated on unseen data (test data).

Oversampling leads to a higher misclassification rate and slightly lower balanced accuracy, which indicates that the model might have overfitted to the oversampled training set, causing it to perform worse on the test set.

Thus, undersampling is the more successful strategy in this case, as it achieves a better balance between model performance and generalization to the test data.

3. Quadratic Discriminant Analysis (QDA)

We can use already defined functions for undersampling and oversampling in previous steps:

- `balanced_train` is the dataset after **undersampling**
- `balanced_train2` is the dataset after **oversampling**.

```r
# Train QDA model on undersampled data
qda_model_undersample <- qda(Status ~ ., data = balanced_train)

# Train QDA model on oversampled data
qda_model_oversample <- qda(Status ~ ., data = balanced_train2)

# Predictions on test set for both models
test_pred_undersample <- predict(qda_model_undersample, newdata = test_data)$class
test_pred_oversample <- predict(qda_model_oversample, newdata = test_data)$class

# Calculate confusion matrices for test set
conf_test_undersample <- table(Predicted = test_pred_undersample, Actual = test_data$Status)
conf_test_oversample <- table(Predicted = test_pred_oversample, Actual = test_data$Status)

# Compute evaluation metrics for both models
compute_metrics <- function(conf_matrix) {
  TP <- conf_matrix[2, 2]
  TN <- conf_matrix[1, 1]
  FP <- conf_matrix[2, 1]
  FN <- conf_matrix[1, 2]

  misclassification_rate <- (FP + FN) / sum(conf_matrix)
  TPR <- TP / (TP + FN)  # Sensitivity
  TNR <- TN / (TN + FP)  # Specificity
  balanced_accuracy <- (TPR + TNR) / 2  # Balanced Accuracy

  return(list(
    MisclassificationRate = misclassification_rate,
    BalancedAccuracy = balanced_accuracy
  ))
}

# Evaluate metrics for both models
metrics_undersample <- compute_metrics(conf_test_undersample)
metrics_oversample <- compute_metrics(conf_test_oversample)

# Print results
list(
  UndersamplingMetrics = metrics_undersample,
  OversamplingMetrics = metrics_oversample
)
```

```
## $UndersamplingMetrics
## $UndersamplingMetrics$MisclassificationRate
## [1] 0.49
##
## $UndersamplingMetrics$BalancedAccuracy
## [1] 0.5287582
##
##
## $OversamplingMetrics
## $OversamplingMetrics$MisclassificationRate
## [1] 0.52
##
```

```
## $OversamplingMetrics$BalancedAccuracy
## [1] 0.4745098
```

Balanced Accuracy for undersampling (0.5288) is higher than for oversampling (0.4745), which suggests that the model trained using undersampling is better at correctly classifying both classes (i.e., it's more balanced in terms of sensitivity and specificity).

Misclassification Rate for undersampling (0.49) is lower than for oversampling (0.52), indicating that the undersampling method also reduces the overall number of misclassifications more effectively.

Thus, undersampling is the more effective strategy for QDA in this scenario, similar to what we observed with LDA earlier. It helps the model maintain better generalization and better performance on both classes.

4. Regularized Discriminant Analysis (RDA) with Undersampling and Oversampling

Here we also can use already defined functions for undersampling and oversampling in previous steps:

- `balanced_train` is the dataset after **undersampling**
- `balanced_train2` is the dataset after **oversampling**.

```r
library(klaR)
# Train RDA model on undersampled data
rda_model_undersample <- rda(Status ~ ., data = balanced_train)

# Train RDA model on oversampled data
rda_model_oversample <- rda(Status ~ ., data = balanced_train2)

# Predictions on test set for both models
test_pred_rda_undersample <- predict(rda_model_undersample, newdata = test_data)$class
test_pred_rda_oversample <- predict(rda_model_oversample, newdata = test_data)$class

# Calculate confusion matrices for test set
conf_test_rda_undersample <- table(Predicted = test_pred_rda_undersample, Actual = test_data$Status)
conf_test_rda_oversample <- table(Predicted = test_pred_rda_oversample, Actual = test_data$Status)

# Compute evaluation metrics for both models
metrics_rda_undersample <- compute_metrics(conf_test_rda_undersample)
metrics_rda_oversample <- compute_metrics(conf_test_rda_oversample)

# Print results
list(
  UndersamplingMetricsRDA = metrics_rda_undersample,
  OversamplingMetricsRDA = metrics_rda_oversample
)
```

```
## $UndersamplingMetricsRDA
## $UndersamplingMetricsRDA$MisclassificationRate
## [1] 0.43
##
## $UndersamplingMetricsRDA$BalancedAccuracy
## [1] 0.5823529
##
##
## $OversamplingMetricsRDA
## $OversamplingMetricsRDA$MisclassificationRate
## [1] 0.4766667
##
## $OversamplingMetricsRDA$BalancedAccuracy
## [1] 0.5183007
```

Undersampling yields again a lower Misclassification Rate (0.43) and higher Balanced Accuracy (0.5824) compared to oversampling.

Undersampling is the more effective strategy for RDA, as it provides better performance in terms of both Misclassification Rate and Balanced Accuracy. It appears to strike a better balance in learning from the training data and generalizing to unseen data.

In Regularized Discriminant Analysis (RDA), the parameters gamma and lambda control the regularization and blending between LDA and QDA. When gamma = 0, RDA is equivalent to Quadratic Discriminant Analysis (QDA). When gamma = 1, RDA simplifies to Linear Discriminant Analysis (LDA). Lambda shrinks the covariance matrices toward a diagonal form. When lambda = 0, the full covariance matrix is used. When lambda = 1, only the diagonal elements of the covariance matrix are used, making the model more robust to small sample sizes or noise.

Let's extract gamma and lamda values for each model:

```r
# Gamma and Lambda for undersampling model
undersample_regularization <- rda_model_undersample$regularization
print(undersample_regularization)
```

```
##        gamma       lambda
## 0.0002820955 0.9580837481
```

```r
# Gamma and Lambda for oversampling model
oversample_regularization <- rda_model_oversample$regularization
print(oversample_regularization)
```

```
##      gamma     lambda
## 0.04459173 0.03186208
```

Model with undersampling: Gamma = 0.00028: Very close to 0, meaning the model heavily relies on QDA, offering maximum flexibility for class separation. Lambda = 0.958: High value, close to 1, indicating that the covariance matrix is highly shrunk toward a diagonal form, reducing the influence of feature interactions.

Model with oversampling: Gamma = 0.0446: Low value, meaning the model still leans towards QDA but incorporates some regularization toward LDA. Lambda = 0.0319: Very low value, meaning the model retains nearly full covariance matrices, allowing it to account for interactions between features.