Exercise 6 (2025) — Advanced Methods for Regression and Classification

Olesia Galynskaia 12321492

2025-11-19

Loading and observing data

```
d <- read.csv("bank.csv", sep = ";")

# y as factor
d$y <- factor(d$y, levels = c("no", "yes"))

# Quick inspection
str(d)
```

```
## 'data.frame':    4521 obs. of  17 variables:
##  $ age      : int  30 33 35 30 59 35 36 39 41 43 ...
##  $ job      : chr  "unemployed" "services" "management" "management" ...
##  $ marital  : chr  "married" "married" "single" "married" ...
##  $ education: chr  "primary" "secondary" "tertiary" "tertiary" ...
##  $ default  : chr  "no" "no" "no" "no" ...
##  $ balance  : int  1787 4789 1350 1476 0 747 307 147 221 -88 ...
##  $ housing  : chr  "no" "yes" "yes" "yes" ...
##  $ loan     : chr  "no" "yes" "no" "yes" ...
##  $ contact  : chr  "cellular" "cellular" "cellular" "unknown" ...
##  $ day      : int  19 11 16 3 5 23 14 6 14 17 ...
##  $ month    : chr  "oct" "may" "apr" "jun" ...
##  $ duration : int  79 220 185 199 226 141 341 151 57 313 ...
##  $ campaign : int  1 1 1 4 1 2 1 2 2 1 ...
##  $ pdays    : int  -1 339 330 -1 -1 176 330 -1 -1 147 ...
##  $ previous : int  0 4 1 0 0 3 2 0 0 2 ...
##  $ poutcome : chr  "unknown" "failure" "failure" "unknown" ...
##  $ y        : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
```

```
table(d$y)
```

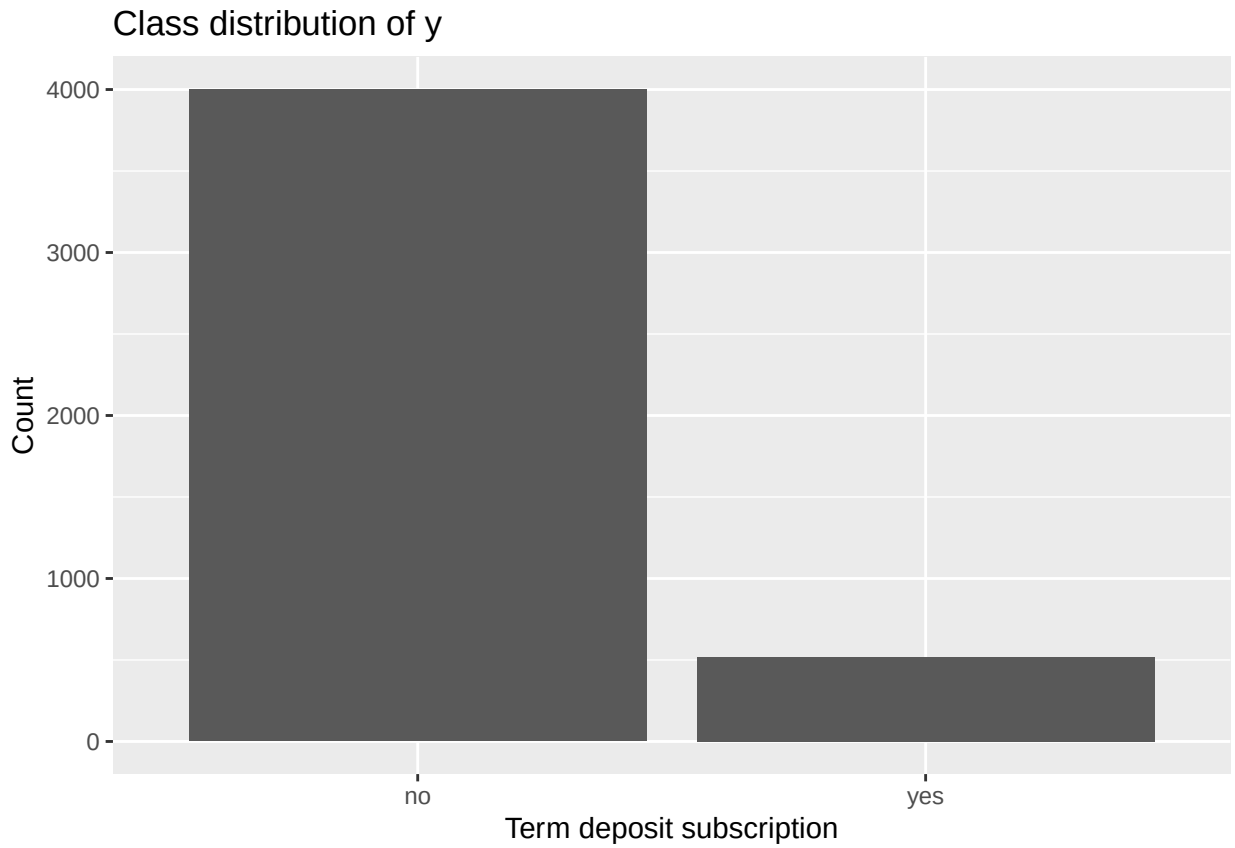
```
##
```

```
##   no  yes
## 4000  521
```

```r
prop.table(table(d$y))
```

```
##
##      no     yes
## 0.88476 0.11524
```

```r
ggplot(d, aes(x = y)) +
  geom_bar() +
  labs(
    title = "Class distribution of y",
    x = "Term deposit subscription",
    y = "Count"
  )
```



Train/test split

```r
set.seed(12321492)

train_idx <- createDataPartition(d$y, p = 2/3, list = FALSE)

train <- d[train_idx, ]
test  <- d[-train_idx, ]

nrow(train)
```

```
## [1] 3015
```

```r
nrow(test)
```

```
## [1] 1506
```

```r
prop.table(table(train$y))
```

```
##
##        no       yes
## 0.8845771 0.1154229
```

```r
prop.table(table(test$y))
```

```
##
##        no       yes
## 0.8851262 0.1148738
```

Evaluation metrics (misclassification + balanced accuracy)

```r
eval_measures <- function(actual, predicted) {

  cm <- table(Actual = actual, Predicted = predicted)

  # Ensure all cells exist (prevent missing categories)
  levs <- c("no", "yes")
  for (a in levs) {
    for (p in levs) {
      if (!(a %in% rownames(cm) && p %in% colnames(cm))) {
        cm[a, p] <- 0
      }
    }
  }
```

```
  cm <- cm[levs, levs]  # reorder matrix

  TN <- cm["no",  "no"]
  FP <- cm["no",  "yes"]
  FN <- cm["yes", "no"]
  TP <- cm["yes", "yes"]

  miscl <- (FP + FN) / sum(cm)

  TPR <- ifelse(TP + FN > 0, TP / (TP + FN), NA)  # sensitivity
  TNR <- ifelse(TN + FP > 0, TN / (TN + FP), NA)  # specificity

  bal_acc <- (TPR + TNR) / 2

  data.frame(
    Misclassification = miscl,
    Sensitivity = TPR,
    Specificity = TNR,
    BalancedAccuracy = bal_acc
  )
}
```

Comment

The data describe customers who were contacted during a marketing campaign for term deposits. Most variables are simple demographic or financial characteristics like age, job type, marital status, education, and balance on their account.

The main issue is the outcome variable y: almost everyone said no.
Only about 11 percent of customers actually subscribed, so the dataset is strongly imbalanced. Models that focus only on accuracy will mostly learn to predict "no" for everyone and look "good," even though they completely fail on the minority class.

Because of this imbalance, we can't rely only on plain misclassification error.
We will need balanced accuracy and similar metrics to judge models fairly, especially for the minority class yes.

Apart from that, the data structure is straightforward, and after removing duration (as required), nothing looks broken or suspicious.

Ex-1

1(a) Apply lda() & preprocessing

```
# Convert character predictors to factors for LDA
train_lda <- train %>%
```

```r
  mutate(across(where(is.character), factor))

# Fit LDA model on the training set
lda_fit <- lda(y ~ ., data = train_lda)

lda_fit
```

```
## Call:
## lda(y ~ ., data = train_lda)
##
## Prior probabilities of groups:
##        no       yes
## 0.8845771 0.1154229
##
## Group means:
##          age jobblue-collar jobentrepreneur jobhousemaid jobmanagement
## no  41.05512      0.2167229      0.03937008   0.02212223     0.2118485
## yes 42.48276      0.1235632      0.02298851   0.03448276     0.2385057
##     jobretired jobself-employed jobservices jobstudent jobtechnician
## no  0.04499438       0.03974503  0.09111361 0.01612298     0.1739783
## yes 0.10344828       0.04597701  0.07758621 0.03448276     0.1551724
##     jobunemployed  jobunknown maritalmarried maritalsingle educationsecondary
## no     0.02999625 0.008998875      0.6261717     0.2617173          0.5031871
## yes    0.03160920 0.011494253      0.5344828     0.3304598          0.4770115
##     educationtertiary educationunknown defaultyes  balance housingyes
## no          0.2950881       0.04461942 0.01499813 1397.302  0.5759280
## yes         0.3505747       0.04022989 0.02011494 1540.833  0.4022989
##        loanyes contacttelephone contactunknown      day monthaug    monthdec
## no  0.15785527       0.06411699      0.3142107 15.97188 0.143607 0.003374578
## yes 0.08045977       0.08045977      0.1178161 16.03448 0.137931 0.020114943
##       monthfeb   monthjan  monthjul  monthjun    monthmar  monthmay   monthnov
## no  0.04424447 0.03037120 0.1552306 0.1166104 0.007124109 0.3307087 0.09261342
## yes 0.06609195 0.03735632 0.1321839 0.1120690 0.028735632 0.1752874 0.08045977
##       monthoct    monthsep duration campaign    pdays  previous poutcomeother
## no  0.01124859 0.007124109 224.3817 2.875891 35.72178 0.4653168    0.04236970
## yes 0.07758621 0.025862069 554.4626 2.218391 69.64080 1.1580460    0.06609195
##     poutcomesuccess poutcomeunknown
## no       0.01124859       0.8413948
## yes      0.17241379       0.6436782
##
## Coefficients of linear discriminants:
##                                LD1
## age                  6.567598e-04
## jobblue-collar      -3.370153e-01
## jobentrepreneur     -1.344880e-01
## jobhousemaid         1.677860e-02
## jobmanagement       -1.257619e-01
```

```
## jobretired          2.352180e-01
## jobself-employed   -4.870173e-02
## jobservices        -1.272030e-01
## jobstudent          2.200090e-01
## jobtechnician      -2.494380e-01
## jobunemployed      -3.883720e-01
## jobunknown          1.012738e-02
## maritalmarried     -1.651360e-01
## maritalsingle      -5.769808e-02
## educationsecondary -2.077238e-02
## educationtertiary   6.000869e-02
## educationunknown   -1.998275e-01
## defaultyes          5.765374e-01
## balance            -6.425332e-06
## housingyes         -1.420268e-01
## loanyes            -2.342584e-01
## contacttelephone   -6.758520e-02
## contactunknown     -5.163122e-01
## day                 1.214150e-02
## monthaug           -3.473662e-01
## monthdec            3.187771e-01
## monthfeb           -5.988805e-02
## monthjan           -6.775372e-01
## monthjul           -4.167854e-01
## monthjun            2.095437e-01
## monthmar            1.099868e+00
## monthmay           -3.177159e-01
## monthnov           -5.355858e-01
## monthoct            1.385682e+00
## monthsep            5.720164e-01
## duration            3.355053e-03
## campaign           -1.226182e-02
## pdays               1.084726e-04
## previous            2.324568e-02
## poutcomeother       2.145591e-01
## poutcomesuccess     3.053775e+00
## poutcomeunknown    -3.885807e-02
```

```
str(train_lda)
```

```
## 'data.frame':    3015 obs. of  17 variables:
##  $ age      : int  35 36 41 43 43 36 40 56 37 25 ...
##  $ job      : Factor w/ 12 levels "admin.","blue-collar",..: 5 7 3 8 1 10 5 10 1 2 ...
##  $ marital  : Factor w/ 3 levels "divorced","married",..: 3 2 2 2 2 2 2 2 3 3 ...
##  $ education: Factor w/ 4 levels "primary","secondary",..: 3 3 3 1 2 3 3 2 3 1 ...
##  $ default  : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ balance  : int  1350 307 221 -88 264 1109 194 4073 2317 -221 ...
```

```
##  $ housing  : Factor w/ 2 levels "no","yes": 2 2 2 2 2 1 1 1 2 2 ...
##  $ loan     : Factor w/ 2 levels "no","yes": 1 1 1 2 1 1 2 1 1 1 ...
##  $ contact  : Factor w/ 3 levels "cellular","telephone",..: 1 1 3 1 1 1 1 1 1 3 ...
##  $ day      : int  16 14 14 17 17 13 29 27 20 23 ...
##  $ month    : Factor w/ 12 levels "apr","aug","dec",..: 1 9 9 1 1 2 2 2 1 9 ...
##  $ duration : int  185 341 57 313 113 328 189 239 114 250 ...
##  $ campaign : int  1 1 2 1 2 2 2 5 1 1 ...
##  $ pdays    : int  330 330 -1 147 -1 -1 -1 -1 152 -1 ...
##  $ previous : int  1 2 0 2 0 0 0 0 2 0 ...
##  $ poutcome : Factor w/ 4 levels "failure","other",..: 1 2 4 1 4 4 4 4 1 4 ...
##  $ y        : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
```

Comment

The main preprocessing step is to convert all categorical variables that are stored as character (job, marital, education, contact, month, poutcome) into factors.

The variables are already on reasonable scales,so no further transformation seems necessary.

1(b) Evaluation on the training data

```r
# Predict on the training set
pred_train <- predict(lda_fit, train_lda)$class

# Compute evaluation measures
eval_train <- eval_measures(train_lda$y, pred_train)
eval_train
```

```
##   Misclassification Sensitivity Specificity BalancedAccuracy
## 1        0.09253731   0.4425287    0.968129        0.7053289
```

Comment

The LDA model looks good only at first glance. The overall misclassification is low because the model predicts "no" very confidently, and most people in the data actually said no.

But when you look at the two classes separately, the picture is different. The model has very high specificity (it almost always gets "no" right) but only moderate sensitivity. It catches less than half of the actual "yes" cases.

The balanced accuracy shows this clearly: it is much lower than the plain accuracy. So LDA learns the majority class well but is still weak at detecting the minority group.

1(c) Evaluation on the test data

```r
# Convert characters to factors also in the test set
test_lda <- test %>%
  mutate(across(where(is.character), factor))

# Predict on the test set
pred_test <- predict(lda_fit, test_lda)$class

# Compute evaluation measures
eval_test <- eval_measures(test_lda$y, pred_test)
eval_test
```

```
##   Misclassification Sensitivity Specificity BalancedAccuracy
## 1         0.1055777   0.3583815    0.963991        0.6611863
```

Comment

On the test data the LDA model behaves the same way as on the training set.
It predicts the majority class "no" very reliably, which keeps the overall misclassification low.
But the sensitivity drops even more here, meaning the model misses many of the actual "yes" cases.

The specificity stays very high, so the model clearly prefers predicting "no".
Because of this imbalance in performance, the balanced accuracy is noticeably lower than the plain accuracy.

In short, the model generalizes consistently, but it still struggles with the minority class and is not good at finding the "yes" customers.

Ex-2

2(a) Undersampling

```r
set.seed(12321492)

# Count class sizes in the original training set
n_no  <- sum(train$y == "no")
n_yes <- sum(train$y == "yes")

# Smallest group size
min_n <- min(n_no, n_yes)

# Undersample: take min_n samples from each class
undersampled <- train %>%
  group_by(y) %>%
  slice_sample(n = min_n) %>%
  ungroup()
```

```r
# Check new class balance
table(undersampled$y)
```

```
##
##  no yes
## 348 348
```

```r
# Convert character variables to factors
undersampled_lda <- undersampled %>%
  mutate(across(where(is.character), factor))

# Fit LDA on the undersampled training set
lda_under <- lda(y ~ ., data = undersampled_lda)

# Evaluation on undersampled TRAIN set
pred_train_under <- predict(lda_under, undersampled_lda)$class
eval_under_train <- eval_measures(undersampled_lda$y, pred_train_under)
eval_under_train
```

```
##   Misclassification Sensitivity Specificity BalancedAccuracy
## 1         0.1767241   0.7729885   0.8735632        0.8232759
```

```r
# Evaluation on the unchanged TEST set
test_lda <- test %>%
  mutate(across(where(is.character), factor))

pred_test_under <- predict(lda_under, test_lda)$class
eval_under_test <- eval_measures(test_lda$y, pred_test_under)
eval_under_test
```

```
##   Misclassification Sensitivity Specificity BalancedAccuracy
## 1         0.1660027   0.7456647   0.8454614        0.7955631
```

Comment

To balance the classes, we took the same number of "no" and "yes" samples from the training data. Since "yes" is the smaller group, we had to throw away most of the "no" cases and kept only 348 from each class. This makes the training set perfectly balanced, but much smaller than before.

After training LDA on this reduced dataset, the model finally started paying attention to the "yes" class. The sensitivity increased a lot, both on the training and on the test set. This means the model is catching many more positive cases than the original LDA.

The downside is that specificity dropped compared to the earlier model, because we removed most of the "no" cases and the classifier has less information about them. Still, the balanced accuracy is clearly higher than before, which shows that the classifier handles the two classes more evenly.

2(b) Oversampling

```r
set.seed(12321492)

# Class sizes in the original training set
n_no  <- sum(train$y == "no")
n_yes <- sum(train$y == "yes")

max_n <- max(n_no, n_yes)

# Separate majority and minority classes
train_no  <- train %>% filter(y == "no")
train_yes <- train %>% filter(y == "yes")

# Oversample: keep max_n from each class
# - majority class: sample without replacement (essentially keep almost everything)
# - minority class: sample with replacement to reach max_n
over_no  <- train_no  %>% slice_sample(n = max_n, replace = FALSE)
over_yes <- train_yes %>% slice_sample(n = max_n, replace = TRUE)

# Combine into oversampled training set
oversampled <- bind_rows(over_no, over_yes)

# Check new class balance
table(oversampled$y)
```

```
##
##   no  yes
## 2667 2667
```

```r
# Convert character variables to factors
oversampled_lda <- oversampled %>%
  mutate(across(where(is.character), factor))

# Fit LDA on the oversampled training set
lda_over <- lda(y ~ ., data = oversampled_lda)

# Evaluation on oversampled TRAIN set
pred_train_over <- predict(lda_over, oversampled_lda)$class
eval_over_train <- eval_measures(oversampled_lda$y, pred_train_over)
eval_over_train
```

```
##   Misclassification Sensitivity Specificity BalancedAccuracy
## 1         0.1867267   0.7615298   0.8650169        0.8132733
```

```r
# Evaluation on the unchanged TEST set
test_lda <- test %>%
  mutate(across(where(is.character), factor))

pred_test_over <- predict(lda_over, test_lda)$class
eval_over_test <- eval_measures(test_lda$y, pred_test_over)
eval_over_test
```

```
##   Misclassification Sensitivity Specificity BalancedAccuracy
## 1         0.1633466    0.734104   0.8499625        0.7920333
```

Comment

For oversampling we simply duplicated the minority class "yes" until it had the same size as the "no" class.
So nothing was removed; we only added repeated examples of the smaller group to make the training set balanced.

The model trained on this oversampled data learns to notice the "yes" class much better.
On the training set both sensitivity and specificity are quite high, and the balanced accuracy is strong.

On the test set the results stay stable: the model still does a much better job detecting "yes" compared to the original LDA, and its specificity remains good.

The balanced accuracy on the test set is clearly better than before.

Overall, oversampling helps the classifier pay attention to both classes without throwing away useful data. The performance is more even and the model handles the minority class more reliably than the original version.

Which strategy is more successful

Both undersampling and oversampling improve the model compared to training on the original imbalanced data, because the classifier finally sees both classes in a more equal way.
However, oversampling is the more successful strategy here.

With undersampling we had to throw away most of the "no" cases, which makes the training set much smaller.
The model becomes more sensitive to the "yes" class, but it loses information about the majority group and becomes less stable.

Oversampling keeps all original data and only repeats the minority class.
This leads to a more balanced model without reducing the size of the training set.
The test balanced accuracy with oversampling is slightly higher, and the overall performance is more stable.

Ex-3 Quadratic Discrimant Analysis (QDA)

QDA with Undersampling

```
set.seed(12321492)

# class counts
n_no  <- sum(train$y == "no")
n_yes <- sum(train$y == "yes")
min_n <- min(n_no, n_yes)

# undersample each class
under_qda_train <- train %>%
  group_by(y) %>%
  slice_sample(n = min_n) %>%
  ungroup() %>%
  mutate(across(where(is.character), factor))

# fit QDA
qda_under <- qda(y ~ ., data = under_qda_train)

# test set (same as before)
test_qda <- test %>%
  mutate(across(where(is.character), factor))

# predictions on test set
pred_qda_under <- predict(qda_under, test_qda)$class

# evaluation
eval_qda_under <- eval_measures(test_qda$y, pred_qda_under)
eval_qda_under
```

```
##   Misclassification Sensitivity Specificity BalancedAccuracy
## 1         0.1932271   0.5549133   0.8394599        0.6971866
```

QDA with Oversampling

```
set.seed(12321492)

n_no  <- sum(train$y == "no")
n_yes <- sum(train$y == "yes")
max_n <- max(n_no, n_yes)

train_no  <- train %>% filter(y == "no")
```

```r
train_yes <- train %>% filter(y == "yes")

# oversample: minority with replacement
over_no  <- train_no  %>% slice_sample(n = max_n, replace = FALSE)
over_yes <- train_yes %>% slice_sample(n = max_n, replace = TRUE)

over_qda_train <- bind_rows(over_no, over_yes) %>%
  mutate(across(where(is.character), factor))

# fit QDA
qda_over <- qda(y ~ ., data = over_qda_train)

# predict on test
pred_qda_over <- predict(qda_over, test_qda)$class

# evaluation
eval_qda_over <- eval_measures(test_qda$y, pred_qda_over)
eval_qda_over
```

```
##   Misclassification Sensitivity Specificity BalancedAccuracy
## 1         0.1540505   0.4971098   0.8912228        0.6941663
```

Comment

With undersampling, QDA gets a big improvement in sensitivity compared to the original model, but the specificity drops. The balanced accuracy on the test set (about 0.70) is decent, but the model is clearly unstable because we removed most of the data.

With oversampling, QDA keeps all original "no" cases and only repeats the "yes" samples. Sensitivity becomes slightly lower than in undersampling, but specificity becomes higher. The final balanced accuracy on the test set (about 0.69) is almost the same as with undersampling.

Overall, both approaches help QDA handle the minority class better than the original version. Undersampling gives higher sensitivity, oversampling gives higher specificity, and the balanced accuracy ends up very similar. Neither method is clearly superior here, and QDA does not benefit as strongly from balancing as LDA did.

Ex-4 Regularized Discrimant Analysis (RDA)

RDA with undersampling

```r
set.seed(12321492)

# Prepare undersampled training set
n_no  <- sum(train$y == "no")
```

```r
n_yes <- sum(train$y == "yes")
min_n <- min(n_no, n_yes)

under_rda_train <- train %>%
  group_by(y) %>%
  slice_sample(n = min_n) %>%
  ungroup() %>%
  mutate(across(where(is.character), factor))

# Fit RDA
rda_under <- rda(y ~ ., data = under_rda_train)

# Predict on unchanged test set
test_rda <- test %>%
  mutate(across(where(is.character), factor))

pred_rda_under <- predict(rda_under, test_rda)$class

# Evaluation
eval_rda_under <- eval_measures(test_rda$y, pred_rda_under)
eval_rda_under
```

```
##   Misclassification Sensitivity Specificity BalancedAccuracy
## 1          0.873174   0.9942197  0.01425356        0.5042366
```

```r
# Show chosen tuning parameters
rda_under$regularization
```

```
##     gamma    lambda
## 0.7473661 0.5608224
```

RDA with oversampling

```r
set.seed(12321492)

# Prepare oversampled training set
n_no  <- sum(train$y == "no")
n_yes <- sum(train$y == "yes")
max_n <- max(n_no, n_yes)

train_no  <- train %>% filter(y == "no")
train_yes <- train %>% filter(y == "yes")

over_no  <- train_no  %>% slice_sample(n = max_n, replace = FALSE)
```

```r
over_yes <- train_yes %>% slice_sample(n = max_n, replace = TRUE)

over_rda_train <- bind_rows(over_no, over_yes) %>%
  mutate(across(where(is.character), factor))

# Fit RDA
rda_over <- rda(y ~ ., data = over_rda_train)

# Predict on test set
pred_rda_over <- predict(rda_over, test_rda)$class

# Evaluation
eval_rda_over <- eval_measures(test_rda$y, pred_rda_over)
eval_rda_over
```

```
##   Misclassification Sensitivity Specificity BalancedAccuracy
## 1         0.3041169   0.7687861   0.6864216        0.7276039
```

```r
# Show tuning parameters
rda_over$regularization
```

```
##     gamma    lambda
## 0.9999569 0.9483156
```

Comment

After undersampling, RDA performs poorly on the test set. Sensitivity is almost 1, but specificity is near zero, so the model predicts nearly everyone as "yes". Balanced accuracy is about 0.50, which is basically useless.

The tuning parameters (gamma 0.75, lambda 0.56) show that the model applies moderate shrinkage and sits between QDA and LDA. With such a small undersampled training set, this regularization pushes the classifier into an overly simple decision boundary, causing the collapse toward predicting "yes" for almost all cases.

Oversampling gives a much better result: sensitivity .77, specificity 0.69, and balanced accuracy 0.73. The model becomes far more balanced and does not collapse into predicting a single class.

The tuning parameters (gamma 1, lambda 0.95) indicate very strong regularization: the covariance structure is almost pooled and almost diagonal. With the larger oversampled training set, this heavy regularization produces a stable classifier that handles both classes more reliably.

Oversampling is clearly better than undersampling for RDA here. It keeps all the majority-class information, provides enough data for the minority class, and leads to a stable model with much higher balanced accuracy.