Lab 3: LDA, QDA, and ROC curves

PSTAT131-231

Week 3

Objectives

- Fit LDA and QDA classifiers in R
- Construct ROC curves using `ROCR` package functions

Throughout the lab, we'll work with the `Default` dataset from the `ISLR` package. Mostly, you'll be reconstructing the examples from the second set of classification lectures.

Linear discriminant analysis

The LDA classifier assumes that the features in each class are multivariate normal with equal covariance matrices:

$$(\mathbf{x} \mid Y = k) \sim N_p(\mu_k, \Sigma)$$

Training the classifier consists in estimating:

- μ_k from $\{\mathbf{x}_i\}_{i:Y_i=k}$;
- Σ from $\mathbf{X}$;
- $\pi_k = P(Y_i = k)$ by $n^{-1} \sum_i I(Y_i = k)$.

Estimates of the posterior probability $p_{ki} = P(Y_i = k \mid \mathbf{x})$ are then found via Bayes' rule, and the maximum a posteriori ('MAP') estimate of the class label is obtained from the *linear* discriminant functions

$$\hat{Y}_i = \arg\max_k \hat{\delta}_k(\mathbf{x})$$

where

$$\hat{\delta}_k(\mathbf{x}_i) = \log \hat{p}_{ki} + C$$

The `lda()` function from the `MASS` package produces all of these computations.

```
# fit lda model
lda_fit <- lda(default ~ balance + income,
               method = 'mle',
               data = Default)
# view output
lda_fit
```

The default estimates of prior probabilities are the proportions of observations in each class. Check that the following gives the same result:

```
# explicitly compute prior probability estimates (pi_k)
prior_est <- Default %>%
  count(default) %>%
  mutate(prop = n/sum(n)) %>%
  pull(prop)

lda(default ~ balance + income,
```

```
    prior = prior_est,
    method = 'mle',
    data = Default)
```

Your turn (1) Check the documentation for `lda` and examine the available methods for estimating μ_k and Σ. Try each option. Does the output of `lda()` change much with the choice of method? Be sure to change the `eval = F` option to `eval = T` when you complete this portion.

```
# check documentation
help(lda)

# try alternative methods
lda(default ~ balance + income,
    method = 'moment',
    data = Default)

lda(default ~ balance + income,
    method = 'mve',
    data = Default)

lda(default ~ balance + income,
    method = 't',
    data = Default)
```

For estimated class labels and posterior probabilities, use `predict()`:

```
# compute estimated classes
lda_preds <- predict(lda_fit, Default)

# view output
str(lda_preds)
```

Notice that `lda_preds` is a named list. The class labels are named `class`:

```
# get class labels
lda_preds$class %>% str()
```

Your turn (2) Use `table` to cross-tabulate the true and estimated classes. Divide by the sample sizes in each class to obtain classification error rates. Be sure to change the `eval = F` option to `eval = T` when you complete this portion.

```
# cross-tabulate estimated and true classes
errors_lda <- table(class = Default$default,
                    pred = lda_preds$class)
errors_lda
```

The posterior probabilities are named `posterior` and stored as a matrix.

```
# get posterior probabilities
lda_preds$posterior %>% head()
```

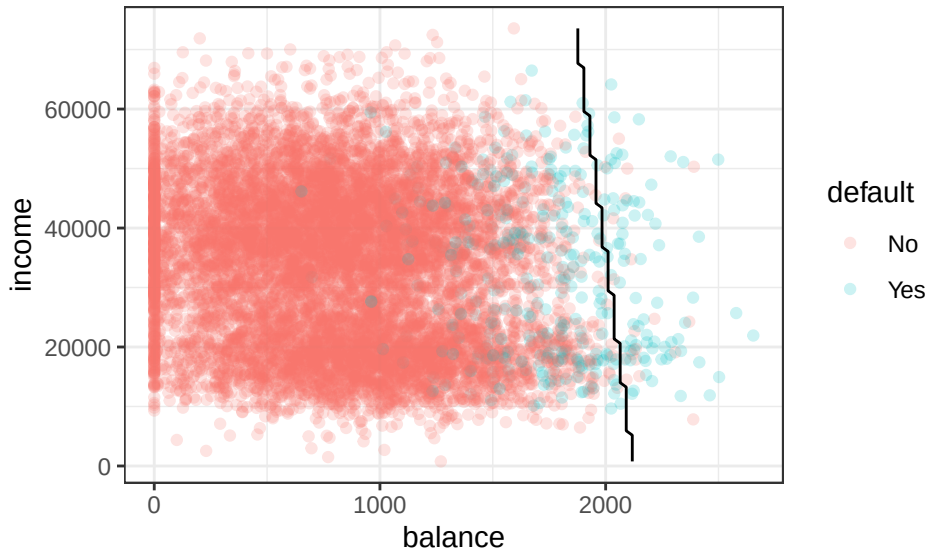
To construct a plot of the decision boundary and estimated class probabilities, generate a grid in the feature space and compute predictions using `lda_fit`:

```
# grid for drawing decision boundary
pred_df <- Default %>%
  data_grid(balance = seq_range(balance, n = 100),
```

```
            income = seq_range(income, n = 100))

# predictions on grid
preds_grid_lda <- predict(lda_fit, pred_df)
probs_grid_lda <- preds_grid_lda$posterior[ , 2] # column 2 is p(yes)

# bind posterior probabilities as a new column to `pred_df`
pred_df <- pred_df %>%
  bind_cols(probs_lda = probs_grid_lda)
```

Your turn (3) Modify the plot below by adding the decision boundary and filled contours of the estimated class probability using `pred_df`. (Follow the example toward the end of lab 2.) Be sure to change the `eval = F` option to `eval = T` when you complete this portion.

```
# plot of decision boundary
Default %>%
  ggplot(aes(x = balance, y = income)) +
  geom_point(aes(color = default), alpha = 0.2) +
  geom_contour(aes(z = as.numeric(preds_grid_lda$class)), data = pred_df,
               bins = 1, show.legend = F, color = "black") + theme_bw()
```



Quadratic discriminant analysis

QDA is exactly the same as LDA except it is assumed that the feature covariances differ between classes:

$$(\mathbf{x} \mid Y = k) \sim N_p(\mu_k, \Sigma_k)$$

Training the QDA classifier consists in estimating μ_k, Σ_k, π_k and calculating resulting estimates of the class labels and probabilities. This is accomplished in R using `qda()` from the `MASS` package:

```
# fit qda model
qda_fit <- qda(default ~ balance + income, data = Default)

# view output
str(qda_fit)
```

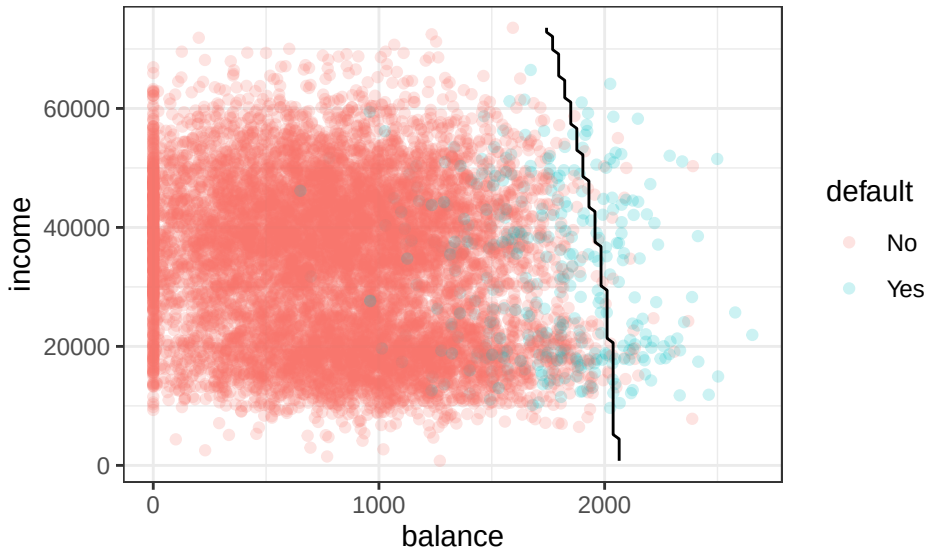
The syntax and output format matches that of `lda()` usage.

Your turn (4) Follow the `lda()` example above and produce the following:

i) Cross-classification table of error rates (not counts).
ii) A plot of the decision boundary.

Be sure to change the `eval = F` option to `eval = T` when you complete this portion.

```
# compute estimated classes
qda_preds <- predict(qda_fit, Default)

# cross-tabulate estimated and true classes
errors_qda <- table(class = Default$default,
                    pred = qda_preds$class)
errors_qda

# compute grid predictions
preds_grid_qda <- predict(qda_fit, pred_df)
probs_grid_qda <- preds_grid_qda$posterior[ , 2]

# add to `pred_df`
pred_df <- pred_df %>%
  bind_cols(probs_qda = probs_grid_qda)

# plot decision boundary
Default %>%
  ggplot(aes(x = balance, y = income)) +
  geom_point(aes(color = default), alpha = 0.2) +
  geom_contour(aes(z = as.numeric(preds_grid_qda$class)), data = pred_df,
               bins = 1, show.legend = F, color = "black") + theme_bw()
```



ROC curves

Receiver operating characteristic curves help with selecting a probability threshold for better control of class-wise error rates in binary classification. Let $p = P(Y = 1)$, denote by n_0, n_1 the numbers of observations in each class, and consider a generic threshold rule:

$$\hat{Y}_i = I(\hat{p}_i > c)$$

The value of c modulates the false positive and true positive rates:

$$FPR(c) = \frac{FP(c)}{N} = \frac{\sum_{i:Y_i=0} I(\hat{p}_i > c)}{n_0} TPR(c) = \frac{TP(c)}{P} = \frac{\sum_{i:Y_i=1} I(\hat{p}_i > c)}{n_1}$$

The ROC curve is a plot of $TPR(c)$ against $FPR(c)$ generated for a grid of $c \in (0, 1)$. In other words, one generates $0 < c_1 < c_2 < \cdots < c_J < 1$ and plots a path through the points $(FPR(c_j), TPR(c_j))$ for $j = 1, \ldots, J$.

The `ROCR` package handily automates these computations from just the class labels $Y_1, \ldots, Y_n$ and any set of estimated probabilities $\hat{p}_1, \ldots, \hat{p}_n$. First, create a `prediction` object to store the labels and probabilities:

```
# create prediction object
prediction_lda <- prediction(predictions = lda_preds$posterior[, 2],
                             labels = Default$default)
```

Then, use `performance()` to automatically generate the grid $\{c_j\}$ and compute the error rates.

```
# compute error rates as a function of probability threshhold
perf_lda <- performance(prediction.obj = prediction_lda, 'tpr', 'fpr')
perf_lda
```

Examining the structure shows that `performance` objects are of a package-associated class with the data structures and names stored in 'slots'. These can be extracted either using the `slot()` function or the `@` operator:

```
# examine structure
str(perf_lda)

# two ways to get a slot
slot(perf_lda, 'alpha.name')
perf_lda@alpha.name
```

The `x.values` slot stores the *second* requested error rate (FPR), and the `y.values` slot stores the *first* requested error rate (TPR). If you forget, you can always check the names (`x.name` and `y.name` slots). The threshold values $\{c_j\}$ are stored in the `alpha.values` slot.

Your turn (5) Using `perf_lda`, create a tibble with three columns: `fpr`, `tpr`, and `thresh`. Be sure to change the `eval = F` option to `eval = T` when you complete this portion.

```
# extract rates and threshold from `perf_lda` as a tibble
rates_lda <- tibble(fpr = perf_lda@x.values,
                    tpr = perf_lda@y.values,
                    thresh = perf_lda@alpha.values) %>%
  unnest(everything())
rates_lda
```

Now, to find the optimal threshold, we'll need to compute Youden's statistic, which is defined as

$$\text{Youden}(c) = TPR(c) - FPR(c)$$

Your turn (6) Use `mutate()` to compute Youden's statistic for the threshold values stored in `rates_lda`. Be sure to change the `eval = F` option to `eval = T` when you complete this portion.

```
# compute youden's statistic
rates_lda <- rates_lda %>%
  mutate(youden = (tpr - fpr))
rates_lda
```

The rates can be checked for any threshold stored in `rates_lda` using a `filter()` command. The `near()` function is a useful helper. For example, we can look at the TPR and FPR around the usual threshold ($\hat{p} > 0.5$). Be sure to change the `eval = F` option to `eval = T` when you arrive at this portion.

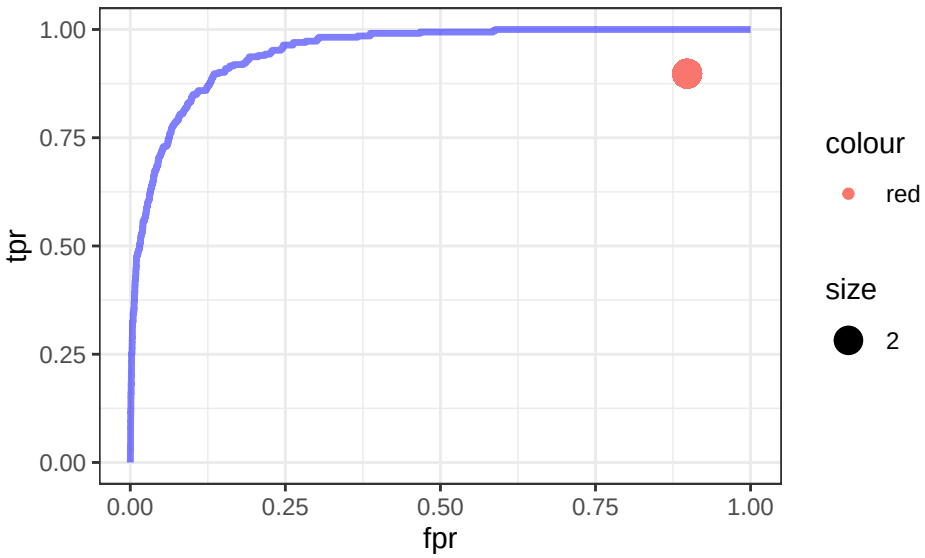
```
# check rates for threshold values within 0.01 of 0.5
rates_lda %>%
  filter(near(thresh, 0.5, tol = 0.01))
```

The optimal threshold can be found quickly using `slice_max(df, var)`, which finds the row of `df` having the larges value of variable `var`:

```
# find optimal threshold
optimal_thresh <- rates_lda %>%
  slice_max(youden)
```

Your turn (7) Using `rates_lda`, construct the ROC curve: a plot of TPR against FPR. Add in the point corresponding to the optimal threshold in red. Be sure to change the `eval = F` option to `eval = T` when you complete this portion.

```
# roc curve
rates_lda %>%
  ggplot(aes(x = fpr, y = tpr)) +
  geom_line(color = "blue", alpha = 0.5, lwd = 1.2) +
  geom_point(aes(x = optimal_thresh$tpr,
                 y = optimal_thresh$tpr,
                 color = "red",
                 size = 2)) + theme_bw()
```



The optimal c can be used to directly compute new estimated class probabilities by creating a factor. Be sure to change the `eval = F` option to `eval = T` when you arrive at this portion.

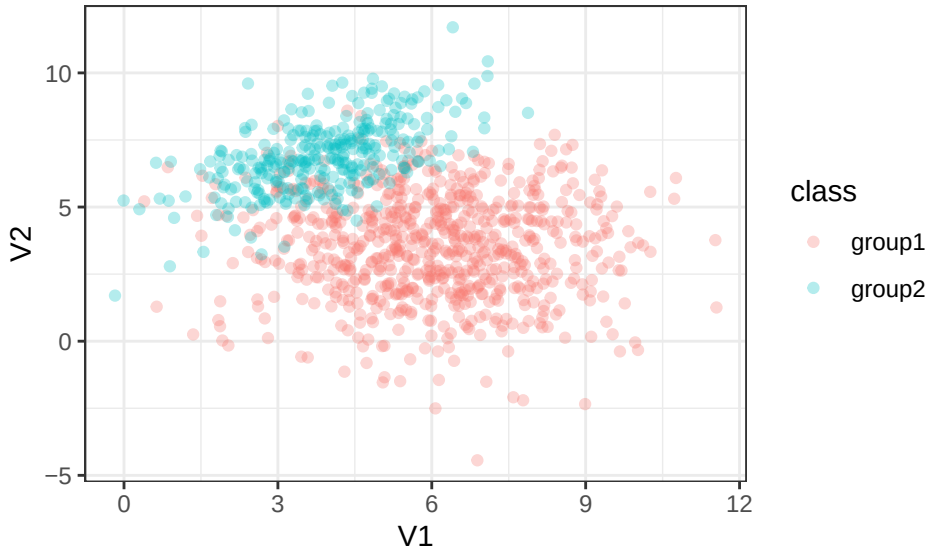
```
# recalibrate qda with different probability threshold
lda_preds_adj <- factor(lda_preds$posterior[, 2] > optimal_thresh$thresh,
                        labels = c('No', 'Yes'))
```

Your turn (8) Use `lda_preds_adj` to cross-tabulate true and estimated classifications. Divide by the numbers of observations in each class to examine class-wise rates. Be sure to change the `eval = F` option to `eval = T` when you complete this portion.

```
# examine misclassifications after thresholding
errors_lda_adj <- table(class = Default$default,
                        pred = lda_preds_adj)
errors_lda_adj/rowSums(errors_lda_adj)
```

Repeating the process for simulated data

Now let's shift gears. The following are simulated data:

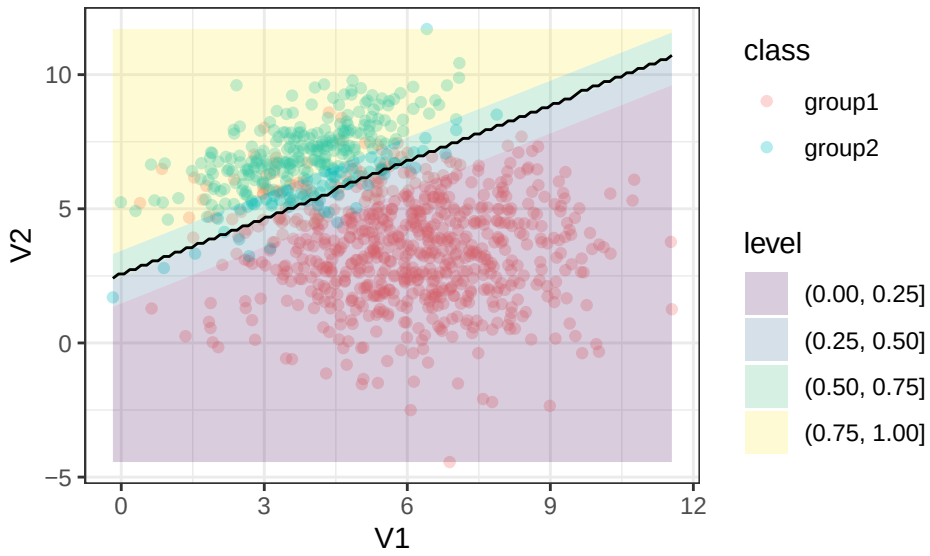


Your turn (9) (help from OH) Select any classification method that you think would be appropriate for this data. Train the classifier and select an optimal threshold by carrying out the process above for the `Default` data. Explain your choice of method in 1-2 sentences and produce:

i) an ROC curve indicating the optimal threshold;
ii) a table of the classification rates (not counts); and
iii) a plot of the decision boundary for your optimal threshold.

```
# train classifier
lda_fit <- lda(class ~ V1 + V2,
               method = 'mle',
               data = sim_data)

# compute predictions
lda_preds <- predict(lda_fit, sim_data)

# calculate TPR, FPR for a grid of threshold values
pred_df <- sim_data %>%
  data_grid(V1 = seq_range(V1, n = 100),
            V2 = seq_range(V2, n = 100))
preds_grid <- predict(lda_fit, pred_df)
probs_grid <- preds_grid$posterior[ , 2]
pred_df <- pred_df %>%
  bind_cols(probs  = probs_grid)

# plot ROC curve
pred.obj <- prediction(predictions = lda_preds$posterior[, 2],
                       labels = sim_data$class)
perf <- performance(prediction.obj = pred.obj, "tpr", "fpr")
rates <- tibble(fpr = perf@x.values,
                tpr = perf@y.values,
                thresh = perf@alpha.values) %>%
  unnest(everything()) %>%
```

```r
  mutate(youden = (tpr - fpr))

# select optimal threshold
optimal_thresh <- rates %>%
  slice_max(youden)
optimal_thresh

# compute classification rates
preds_adj <- factor(lda_preds$posterior[, 2] > optimal_thresh$thresh,
                    labels = c("group1", "group2"))
table(class = sim_data$class, pred = preds_adj)

# plot decision boundary
sim_data %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_point(aes(color = class), alpha = 0.3) +
  geom_contour_filled(aes(z = probs), data = pred_df,
                      alpha = 0.2, bins = 4) +
  geom_contour(aes(z = as.numeric(preds_grid$class)),
               data = pred_df,bins = 1,
               show.legend = F,color = "black") + theme_bw()
```



I chose the mle method because the "MLE procedures has greater efficiency and better numerical stability can often be obtained by taking advantage of the properties of the specific estimation problem." Therefore, I feel that it is applicable to various classification scenarios.