In-class Programming Activity 12

Math 253: Statistical Computing & Machine Learning

Testing LDA and QDA

Today, you're going to generate simulated data to test out LDA and QDA. The data set you make will have two quantitative variables `x` and `y` to use as inputs, and a output `color` with levels `"red"` and `"blue"`. Actually, you'll be creating two different data tables:

- `Same_sigma` where cases in the two classes have the same (x, y) covariance matrix for the predictor variables.
- `Different_sigma` where the two classes have different (x, y) covariance matrices.

Generating simulated data

- Create an object `n_cases` with the value 100.
- Create an object `red_mean` with the value `c(1, 0)`.
- Create an object `green_mean` with value `c(0, -1)`.
- Create an object `blue_mean` with value `c(-1, 1)`.
- Create an object `covar_red` which is a 2 x 2 matrix with 3 and 1 on the diagonal and -1.7 on the off-diagonal. It should be structured as a covariance matrix.
- Similarly, create an object `covar_blue`, a covariance matrix with 2 and 3 on the diagonal and 1.5 on the off-diagonal.
- Create three matrices, `One`, `Two`, and `Three`. Each should have the appropriate shape for data from `n_cases` cases and two variables. The variables should each be `n_cases` random draws from a $\mathrm{N}(0, 1^2)$ distribution, that is, a normal distribution with mean zero and standard deviation 1.

In specifying the normal distribution, one needs to decide whether to report the standard deviation or the variance. R uses `sd=`. To help to eliminate ambiguity in mathematical notation, the form 1^2 is used simply as a reminder that the quantity is a variance.

- Now you're going to transform `One`, `Two` and `Three` to create correlations between the two columns.
  - Create an matrix `Red` that is `One` times the Cholesky decomposition of `covar_red`. The `Red` matrix will contain correlated random variables with a covariance of approximately `covar_red`.
  - Similarly, create a matrix `Green` which will be `Two` times the Cholesky decomposition of `covar_red`. That is, `Green` and `Red` will have the same correlation structure.
  - Now create a matrix `Blue` which will be `Three` times the Cholesky decomposition of the other covariance matrix, `covar_blue`.
- Modify the `Red`, `Green` and `Blue` matrices by adding to each column a value for the mean drawn from `red_mean`, `green_mean`, and `blue_mean` respectively. That is, for `Red`, add 1 to the first column and 0 to the second.

Hint: It won't work to do the obvious, simple thing, e.g. add `red_mean` to `Red`. There are many ways to construct a statement that works. Among others, there's a way using `outer()`, a way using `matrix()`, and even a way using `t()` twice.

- Create three data frames, each with variables `x`, `y`, and `class`.
  - `Red` will have `x` as the first column of `Red`, `y` as the second column of `Red`, and `class` set equal to the string `"red"`.
  - `Blue` will be the same thing but using the columns of `blue` and the `class` set to `"blue"`
  - `Green` is similar, using the columns of `Green` and the class `"green"`.

You'll use `data.frame()` to construct the data frames. Make sure to give the optional argument `stringsAsFactors = FALSE`. This will let the class be stored as straightforward character strings that can be used in plotting to specify the color.

- Last step in generating the simulated data: make two data frames each of which combines "data" from two classes.

```
Same_sigma <- rbind(Red, Green)
Different_sigma <- rbind(Red, Blue)
```

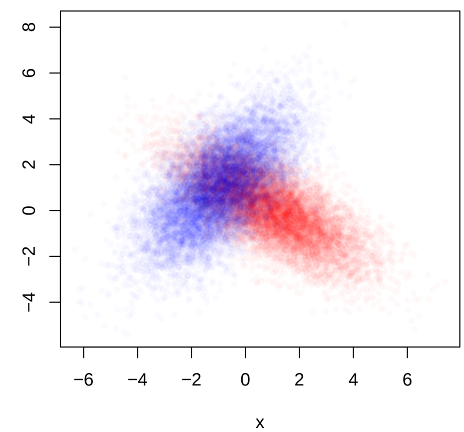

Figure 1: The two classes of cases, red and blue, in `Sim_two`.

LDA and QDA

Fit a linear discriminant model `class ~ x + y` to the data in `Same_sigma`. Call the model `mod_LDA_one`.

```
library(MASS)
mod_LDA_one <- lda(class ~ x + y, data = Same_sigma)
```

Then use the model to test the model on the same training data to which it was fit. Store the result in `test_LDA_one`.

```
test_LDA_one <- predict(mod_LDA_one, newdata = Same_sigma)
```

The resulting object, `test_LDA_one`, is a list of three items. Make sure you understand what each of them is.

QDA works in the same way: the function is `qda()`.

Confusion matrices

The confusion matrix compares the actual class to the predicted class from the model. It's straightforward to compute:

```
table(Same_sigma$class, test_LDA_one$class)
```

- Compare the confusion matrix from LDA on `Same_sigma` to that from QDA based on `Same_sigma`.[1] Which one shows better performance?

- Fit both LDA and QDA models to `Sim_two`. Which one performs better?

[1] Use the names `mod_QDA_one` and `test_QDA_one` to store the fitted model and the test results from `predict()` respectively.

Bigger n

The difference in performance of LDA and QDA in these examples is not so large that it's evident in a sample with 100 cases of each class. Go back and set `n_cases` to be 10000, and re-evaluate the confusion matrices.

Above and beyond

Calculate the log likelihood for `mod_LDA_one` against the observations `Different_sigma$class`.