In-Class Computing Task: Day 12

Math 253: Statistical Computing & Machine Learning

Thursday March 3, 2016

Today, you're going to generate simulated data to test out LDA and QDA.

You'll make two data sets, each of which will have two classes of outcomes: `"red"`, and `"blue"`

- `SameSigma` where cases in the two classes have the same covariance matrix for the predictor variables.
- `DifferentSigma` where the two classes have different 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_1` 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_2`, 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.
- Create an matrix `red` that is `one` times the Cholesky decomposition of `covar_1`. The `red` matrix will contain correlated random variables with a covariance of approximately `covar_1`.
- Similarly, create a matrix `green` which will be `two` times the Cholesky decomposition of `covar_1`.
- Also create a matrix `blue` which will be `three` times the Cholesky decomposition of the other covariance matrix, `covar_2`.
- 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.

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.

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.

```
Sim_one <- rbind(Red, Green)
Sim_two <- rbind(Red, Blue)
```

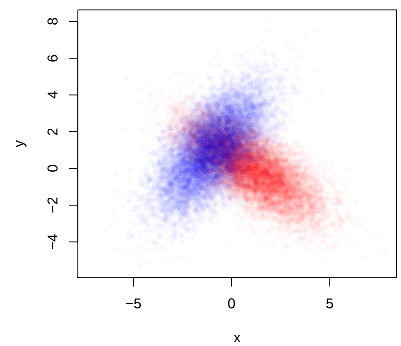

Figure 1: The two classes of cases, red and blue, in `Sim_two`.

LDA and QDA

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

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

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 = Sim_one)
```

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(Sim_one$class, test_LDA_one$class)
```

- Compare the confusion matrix from LDA on `Sim_one` to that from QDA on `Sim_one`.[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 `Sim_one$class`.