

LDA

Statistical Learning, BGC - 2011

Niels Richard Hansen
September 18, 2011

The `lda` function is in the MASS package. In this example we will use the ggplot2 package.

```
> library(MASS)
> library(ggplot2)
```

Reading the data.

```
> vowels <- read.table("http://www-stat-
class.stanford.edu/%7Etibs/ElemStatLearn/datasets/vowel.train",
+                       row.names = 1,
+                       head = TRUE,
+                       sep = ",")
> vowelsY <- vowels[, 1]
> vowelsX <- as.matrix(vowels[, -1])
> xbar <- colMeans(vowelsX)
```

Computations of the group means. Note that the 11 groups are all of equal size (48 observations per group), which makes all the prior class probabilities equal. If the prior class probabilities are not taken equal, the final classifier will have to be modified accordingly by a class specific additive term.

```
> A <- model.matrix(~ y - 1, data = data.frame(y = factor(vowelsY)))
> M <- solve(t(A) %*% A, t(A) %*% vowelsX)
> AM <- A %*% M
> n <- dim(A)[1] ## For later use
> ng <- dim(A)[2] ## For later use
```

We can then proceed as outlined in ESL, page 114, to compute a coordinate system for projection and classification. The computations of the W_0 ($W_0^T W_0 = W^{-1}$) and V^* matrices are most easily achieved by using the *singular value decomposition* to be dealt with later in the course.

```
> svdX <- svd(vowelsX - AM)
> W0 <- svdX$v %*% diag(1/svdX$d)
> Vstar <- svd(scale(M, center = xbar, scale = FALSE) %*% W0)$v
> scalings <- W0 %*% Vstar
```

The resulting *scaling* variables can be used for computing the classifications. Note that using

the centering variable to be `xbar` above, instead of the mean of `M`, makes no difference if the class priors are equal as in this case. In general, it does make a difference. If we center with any affine combination of the rows in `M` we will always get a V^* matrix with columns that span the same space, but their exact form will depend on the chosen centering.

```
> predictors <- vowelsX %*% scalings
> means <- M %*% scalings
> dist <- matrix(0, n, ng)
> for(i in 1:ng)
+   dist[, i] <- rowSums(scale(predictors, means[i, ], scale = FALSE)^2)
> prior <- colMeans(A)
> ## Predicted values on training data
> yHat <- apply(scale(dist, center = 2*log(prior)/(n-ng), scale = FALSE),
+               1, which.min)
```

The predicted values above are based on computing the distance to the nearest class mean in the transformed basis where the inner product is proportional to the standard inner product. When $K < p$ this is, in general, a good idea. Note that it is in the final computation above that the prior class probabilities have to enter, if they differ. In the alternative, we can compute the nearest class mean in the original basis but with an inner product based on the empirical covariance matrix.

```
> Wm1 = solve(crossprod((vowelsX - AM)))
> dist2 <- matrix(0, n, ng)
> for(i in 1:ng) {
+   XmM <- scale(vowelsX, M[i, ], scale = FALSE)
+   dist2[, i] <- rowSums(XmM * (XmM %*% Wm1))
+ }
> max(abs(dist2 - dist))
```

```
[1] 2.220446e-16
```

Why all the mess with the scalings if the above code can just as easily produce the distances to the class means? One point is that once the scalings are computed we only need to compute K inner products (scales like Kp) and K standard norms (scales like Kp) to make one classification. Using the standard basis we need to compute a matrix product (scales like p^2), which is slower. Another point is for visualization where the projection onto the first few *scalings* can provide some insight into the separation of the classes in the p -dimensional space.

Note that *any* orthonormal basis for the column space of the centered M will work as the V^* matrix for computation of distances and for classification, but that the specific choices of *scalings* above (in that particular order) are known as the *canonical variates* and projection onto this basis gives canonical coordinates.

The MASS package provides an implementation of LDA that is pretty easy to use, and which hides most of the messy stuff.

```
> vowelsLda <- lda(y ~ ., data = vowels)
> sqrt(n - ng)*scalings/vowelsLda$scaling
```

	LD1	LD2	LD3	LD4	LD5	LD6	LD7	LD8	LD9	LD10
x.1	1	1	1	1	1	1	1	1	-1	-1
x.2	1	1	1	1	1	1	1	1	-1	-1
x.3	1	1	1	1	1	1	1	1	-1	-1
x.4	1	1	1	1	1	1	1	1	-1	-1
x.5	1	1	1	1	1	1	1	1	-1	-1
x.6	1	1	1	1	1	1	1	1	-1	-1
x.7	1	1	1	1	1	1	1	1	-1	-1
x.8	1	1	1	1	1	1	1	1	-1	-1
x.9	1	1	1	1	1	1	1	1	-1	-1
x.10	1	1	1	1	1	1	1	1	-1	-1

In this example, the *scalings* produced above and from `lda` agree up to the sign (the v -vectors produced by the singular value decomposition do not have a unique sign). The predictions also agree.

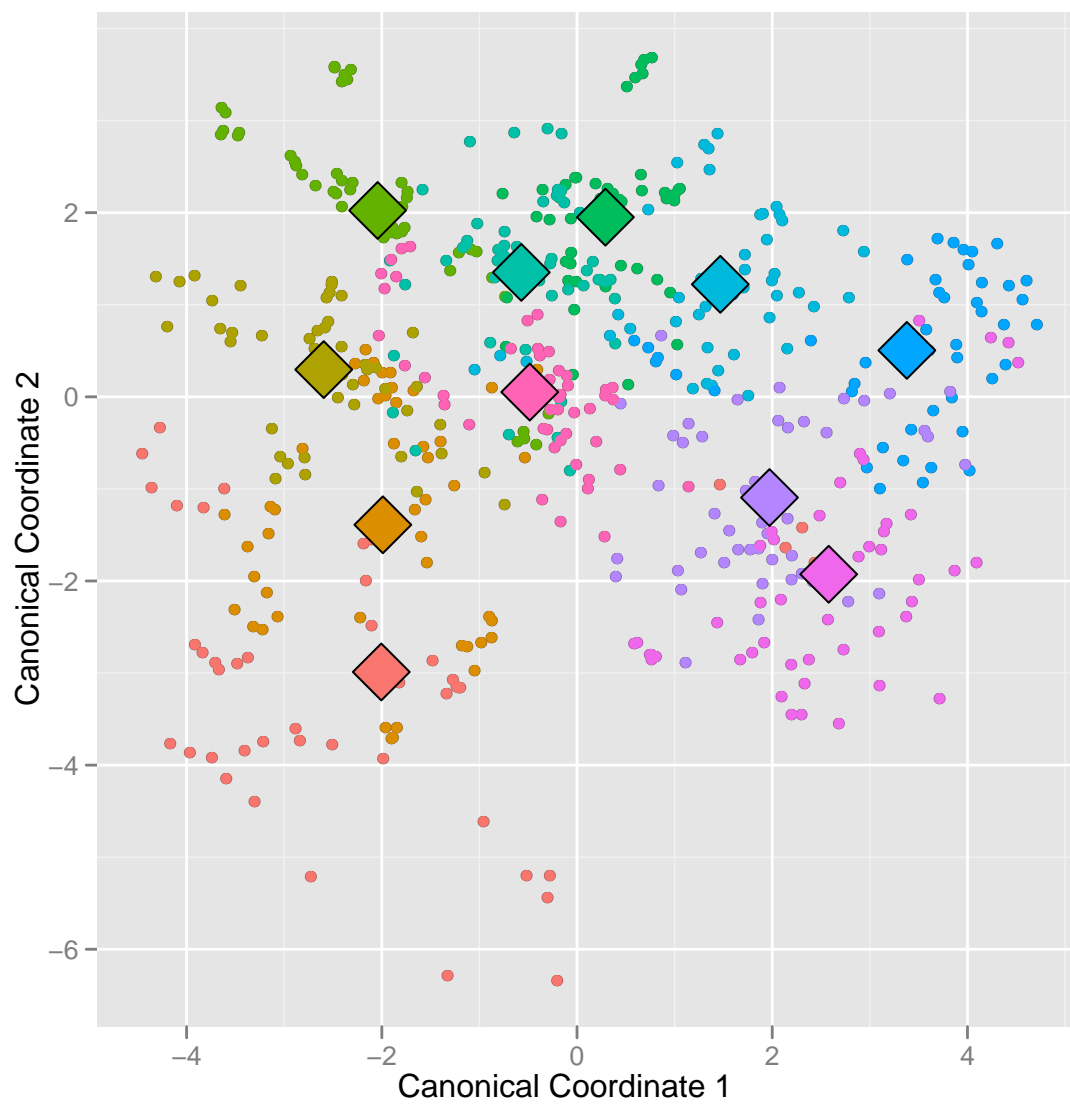
```
> all.equal(predict(vowelsLda, vowels)$class, as.factor(yHat))
```

```
[1] TRUE
```

In general, when the class priors are not equal, the *scalings* that `lda` returns are based on a different V^* matrix, which is computed from a weighted covariance matrix of M . The row weights are the square roots of the class priors. It's just a slightly different basis that is chosen to emphasize large groups over smaller small groups.

Plot of the projection onto the canonical coordinates using the results computed by the `lda` function. The data and the group means have been centered before the projection. This does not change the figure except for a translation. It seems to be common when plotting projections onto the canonical coordinates to always plot the projections of the centered values.

```
> vowelsXCentered <- scale(vowelsX, scale = FALSE)
> ldaMeans <- scale(vowelsLda$mean, scale = FALSE) %*%
+   vowelsLda$scaling[, c(1, 2)]
> p <- qplot(vowelsXCentered %*% vowelsLda$scaling[, 1],
+           vowelsXCentered %*% vowelsLda$scaling[, 2]) +
+   geom_point(aes(colour = factor(vowelsY))) +
+   geom_point(aes(x = ldaMeans[, 1], y = ldaMeans[, 2], fill =
+ factor(1:11)),
+             shape=23, size=8) +
+   scale_colour_discrete(legend = FALSE) +
+   scale_fill_discrete(legend = FALSE) +
+   xlab("Canonical Coordinate 1") +
+   ylab("Canonical Coordinate 2")
```



A less fancy version of the same figure.

```
> plot(vowelsLda, dimen = 2)
> points(ldaMeans[, 1], ldaMeans[, 2], cex = 5, pch=20)
```

