Lab 9: Linear Disciminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA)

Johnny Hong

Stat 154, Fall 2017

Introduction

In this lab we will explore linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA). Recall that in a classification problem the response variable Y is categorical, which can be formulated as $Y \in \{1, ..., K\}$ with $K \geq 2$ being the number of classes; the predictor X is a p-dimensional vector. Logistic regression takes a discriminative modeling perspective, meaning that it directly models $Pr(Y = k|X = x)$. On the other hand, both LDA and QDA take a generative modeling perspective; that is, these two methods model $Pr(X = x|Y = k)$ first, and then use Bayes theorem to deduce $Pr(Y = k|X = x)$. Specifically, LDA assumes that $X|Y = k \sim N(\mu_k, \Sigma)$ and QDA assumes that $X|Y = k \sim N(\mu_k, \Sigma_k)$ for $k = 1, ..., K$. The key difference between LDA and QDA is that LDA assumes all the classes are based upon the same covariance matrix while QDA allows the covariance matrices to be different for different classes.

Let's consider the general generative modeling perspective. Let

- $\pi_k = Pr(Y = k)$ is the prior probability that a randomly chosen observation comes from the kth class
- $f_k(x) = Pr(X = x|Y = k)$.

Using Bayes Theorem,

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}.$$

The *Bayes classifer* classifies an observation to the class for which the posterior probability $P(Y = k|X = x)$ is the largest; that is,

$$\hat{k}_{Bayes}(x) = \arg \max_{k=1,...,K} Pr(Y = k|X = x).$$

Note that

$$\begin{aligned} \arg \max_{k=1,...,K} Pr(Y = k|X = x) &= \arg \max_{k=1,...,K} \frac{Pr(Y = k)Pr(X = x|Y = k)}{Pr(X = x)} \\ &= \arg \max_{k=1,...,K} Pr(Y = k)Pr(X = x|Y = k) \\ &= \arg \max_{k=1,...,K} \left[\log \pi_k + \log f_k(x)\right]. \end{aligned}$$

LDA

Assuming that $X|Y = k \sim N(\mu_k, \Sigma)$, we have

$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right), x \in \mathbb{R}.$$

Given $X = x$, the Bayes classifier suggests that we compute

$$\delta_k(x) = \log \pi_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \mu_k^T\Sigma^{-1}x$$

for $k = 1, ..., K$ and see for which k $\delta_k(x)$ is the largest. However, π_k, μ_k, and Σ are all unknown and hence must be estimated from the data. Given data $\{(x_i, y_i)\}_{i=1}^n$, these parameters can be estimated via

$$\hat{\pi}_k = \frac{n_k}{n}, \hat{\mu}_k = \frac{1}{n_k}\sum_{i:y_i=k} x_i, \hat{\Sigma} = \frac{1}{n-K}\sum_{k=1}^K \sum_{i:y_i=k}(x_i-\hat{\mu}_k)(x_i-\hat{\mu}_k)^T,$$

where n_k is the number of observations in class k. The LDA classifier can be viewed as an estimated version of the Bayes classifier; it assigns an observation $X = x$ to the class for which

$$\hat{\delta}_k(x) = \log \hat{\pi}_k - \frac{1}{2}\hat{\mu}_k^T\hat{\Sigma}^{-1}\hat{\mu}_k + \hat{\mu}_k^T\hat{\Sigma}^{-1}x$$

is maximized. For the sake of matching the output from `lda()` in the `MASS` package, we would compute the estimated posterior probabilities instead of $\hat{\delta}_k(x)$.

Your turn

- Implement a function called `my_lda()` that computes the necessary estimates for LDA. The function should contain two arguments
  - `X`: the predictor matrix, which is an $n \times p$ matrix
  - `y`: the response vector, which is a factor vector of length n
- `my_lda()` should return a list of objects:
  - `pi_hat`: the prior probability vector, which is a vector of length K
  - `mu_hat`: a $K \times p$ matrix in which each row contains the mean of the group
  - `sigma_hat`: the $p \times p$ covariance matrix of the predictors
- Implement a function called `predict_my_lda()` that generates predictions based on the output from `my_lda()`. The function should contain two arguments
  - `fit`: the output from `my_lda()`
  - `newdata`: a $m \times p$ matrix of new observations
- `predict_my_lda()` should return a list of objects:
  - `class`: a length-m factor vector; each of its elements indicate the predicted class of an observation
  - `posterior`: a $m \times K$ matrix of posterior probabilities (Hint: You might find `dmvnorm()` in the `mvtnorm` package useful.)
- Train your LDA on the first 140 observations in the dataset `iris` and predict the last 10 observations.
- Check your answer with `lda()` in the `MASS` package.

QDA

Assuming that $X|Y = k \sim N(\mu_k, \Sigma_k)$, we have

$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right), x \in \mathbb{R}.$$

To estimate π_k, μ_k, Σ_k, we use

$$\hat{\pi}_k = \frac{n_k}{n}, \hat{\mu}_k = \frac{1}{n_k}\sum_{i:y_i=k} x_i, \hat{\Sigma}_k = \frac{1}{n_k-1}\sum_{i:y_i=k}(x_i-\hat{\mu}_k)(x_i-\hat{\mu}_k)^T.$$

Your turn

- Implement a function called `my_qda()` that computes the necessary estimates for QDA. The function should contain two arguments
  - `X`: the predictor matrix, which is an $n \times p$ matrix
  - `y`: the response vector, which is a factor vector of length n
- `my_qda()` should return a list of objects:
  - `pi_hat`: the prior probability vector, which is a vector of length K
  - `mu_hat`: a $K \times p$ matrix in which each row contains the mean of the group
  - `sigma_hat`: a $p \times p \times K$ array, where `sigma_hat[␣,k]` contains the covariance matrix of the predictors in class k
- Implement a function called `predict_my_qda()` that generates predictions based on the output from `my_qda()`. The function should contain two arguments
  - `fit`: the output from `my_qda()`
  - `newdata`: a $m \times p$ matrix of new observations
- `predict_my_qda()` should return a list of objects:
  - `class`: a length-m factor vector; each of its elements indicate the predicted class of an observation
  - `posterior`: a $m \times K$ matrix of posterior probabilities (Hint: You might find `dmvnorm()` in the `mvtnorm` package useful.)
- Train your QDA on the first 140 observations in the dataset `iris` and predict the last 10 observations.
- Check your answer with `qda()` in the `MASS` package.

Confusion matrix

One way to describe the performance of a classifier is by computing the confusion matrix. For a K-class problem, the confusion matrix is a $K \times K$ matrix in which the $(i,j)th$ entry is the count of observations that actually belong to class j and are classified as class i. (Remark: This convention has not been standardized:t some confusion matrices you encountered in practice flip the role of i and j.)

```r
set.seed(100)
train_idx <- sample(nrow(iris), 90)
train_set <- iris[train_idx, ]
test_set <- iris[-train_idx, ]
```

Your turn

- Train LDA and QDA based on `train_set`.
- Generate predictions on `test_set`.
- Compute the confusion matrix for each method. (Hint: You might find the function `table()` useful.)
- Check your answer with `confusionMatrix()` in the R package `caret`. You might also have to install the package `e1071`.

Multinomial Logistic Regression

We have seen that logistic regression can be used to handle two-class classification. We are now going to explore how to do K-class classification with logistic regression. We use one of the classes as the baseline category. For conreteness, we will take this class as class K. The idea of multilogistic regression is to fit $K-1$ binary logistic regression simultaneously:

$$\log \frac{P(Y=k|X)}{P(Y=K|X)} = \beta_{0k} + \sum_{j=1}^{p} x_{ij}\beta_{jk},$$

for $k = 1, ..., K-1$, where x_{ij} is the jth predictor for the ith observation and β_{jk} is the jth coefficient for the kth logistic regression. Based on this construction,

$$P(Y=k|X) = \frac{\exp(\beta_{0k} + \sum_{j=1}^{p} x_{ij}\beta_{jk})}{1 + \sum_{m=1}^{K} \exp(\beta_{0m} + \sum_{j=1}^{p} x_{ij}\beta_{jm})}, \text{ for } k = 1, ..., K-1$$

and

$$P(Y=K|X) = \frac{1}{1 + \sum_{m=1}^{K} \exp(\beta_{0m} + \sum_{j=1}^{p} x_{ij}\beta_{jm})}.$$

The parameters are estimated via maximum likelihood estimation. The log-likelihood function for the multinomial logistic regression model is

$$l(\beta) = \sum_{i=1}^{n} \left[\sum_{k=1}^{K-1} \left(y_{ik} \sum_{j=0}^{p} x_{ij}\beta_{jk} \right) - \log\left(1 + \sum_{k=1}^{K-1} \exp\left(\sum_{j=0}^{p} x_{ij}\beta_{jk} \right) \right) \right].$$

where $x_{i0} = 1$ for all $i = 1, ..., n$, $y_{ik} = I(\text{obs } i \text{ belongs to class } k)$, and β is a $(p+1)\times(K-1)$ matrix. Recall that β_{jk} is the jth coefficient for the kth logistic regression. The maximum likelihood estimate of β is

$$\hat{\beta}_{MLE} = \arg\max_{\beta} l(\beta).$$

Your turn

- Implement a function called `find_multinom_coef()` that computes the coefficients for multinomial logistic regression. The function treats the first category as the baseline category.
- `find_multinom_coef()` takes the following inputs:
  - `X`: a $n \times p$ matrix of predictors (it does not contain a column of 1's)
  - `y`: a factor vector of length n with at least two different categories (the first category is the first element in `levels(y)`)
- `find_multinom_coef()` gives the following output:
  - `param`: a $(p+1) \times (K-1)$ matrix
- Hint: You might find the function `dummify()` written in the first (or second?) lab useful.
- Hint: You are allowed to use `optim()` for this task. Check the documentation of `optim()` to see how to use it properly. Make sure you set `method="BFGS"` (otherwise `optim()` might not converge within a reasonable number of iterations).
- Run `find_multinom_coef(X=iris[1:140, 1:4], y=iris$Species[1:140])` to obtain the coefficient matrix. Make sure the output has the same format as below.

```r
find_multinom_coef(X=iris[1:140, 1:4], y=iris$Species[1:140])
```

```
##              versicolor  virginica
## (Intercept)  17.7254637 -24.631223
## Sepal.Length -6.7005422  -9.107771
## Sepal.Width  -6.2433337 -12.869906
## Petal.Length 13.7900525  23.118285
## Petal.Width  -0.5066336  17.596108
```

- Your coefficient matrix should contain values close to the following:

```r
library(nnet)
iris_multi <- multinom(Species ~ ., data=iris[1:140, ]) # ignore the output here.
```

```
## # weights:  18 (10 variable)
## initial  value 153.805720
## iter  10 value 24.082349
## iter  20 value 6.036653
## iter  30 value 5.937954
## iter  40 value 5.930515
## iter  50 value 5.926939
## iter  60 value 5.925467
## final  value 5.923988
## converged
```

```r
t(coef(iris_multi))
```

```
##              versicolor  virginica
## (Intercept)  17.7252583 -24.630925
## Sepal.Length -6.7006986  -9.107935
## Sepal.Width  -6.2434619 -12.870044
## Petal.Length 13.7902839  23.118434
## Petal.Width  -0.5060067  17.596721
```