HW9: Discriminant Analysis

Stat 154, Fall 2019

Introduction

In this assignment you 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.

LDA and QDA take a generative modeling perspective; that is, these two methods model the class-conditional probability $Pr(X = x|Y = k)$ 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 k-th 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* assigns an observation to the class for which the posterior probability $P(Y = k|X = x)$ is the largest; that is,

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

Note that

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

1) LDA

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

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

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 class the function $\delta_k(x)$ 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

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

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 $\hat{\delta}_k(x)$.

1.1) Function `lda_fit()`

Implement a function called `lda_fit()` that computes the necessary estimates for LDA.

The function `lda_fit()` 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

`lda_fit()` 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

1.2) Function `lda_predict()`

Implement a function called `lda_predict()` that generates predictions based on the output from `lda_fit()`. The function should contain two arguments

- `fit`: the output from `lda()`
- `newdata`: a $m \times p$ matrix of new observations

`lda_predict()` 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.)

1.3) Classification with LDA

Use `lda_fit()` to train your LDA on observations `training <- c(1:47, 51:97, 101:146)` in the dataset `iris`, and then use `lda_predict()` to obtain the predicted class on observations `testing <- c(48:50, 98:100, 147:150)`. Display the output of each function.

---

2) QDA

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

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

To estimate π_k, μ_k, Σ_k, we use

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

2.1) Function `qda_fit()`

Implement a function called `qda_fit()` 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

`qda_fit()` 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

2.2) Function `predict_qda()`

Implement a function called `predict_qda()` that generates predictions based on the output from `qda_fit()`. The function should contain two arguments

- `fit`: the output from `qda()`
- `newdata`: a $m \times p$ matrix of new observations

`predict_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.)

2.3) Classification with QDA

Use `qda_fit()` to train your LDA on observations `training <- c(1:47, 51:97, 101:146)` in the dataset `iris`, and then use `qda_predict()` to obtain the predicted class on observations `testing <- c(48:50, 98:100, 147:150)`. Display the output of each function.

---

3) k-Nearest Neighbors

The idea of k-NN is to classify a new observation based on the majority vote of the k nearest observations in the training set. Let $S_k(x)$ be the set of k training observations that are closest to x. If all the predictors are real-valued, the typical choice for the distance measure is the Euclidean distance: given $x, w \in \mathbb{R}^p$,

$$d(x, w) = \sqrt{\sum_{j=1}^{p} (x_j - w_j)^2}.$$

Once $S_k(x)$ has been identified, we can compute the estimated "conditional probability" that an observation with predictor x belongs to class c as

$$\hat{P}(Y = c|x) = \frac{1}{k} \sum_{i \in S_k(x)} I(y_i = c),$$

for each $c \in \{1, ..., K\}$, where y_i is the class of the ith observation.

k-NN classifies an observation with predictor vector x to class c^* if $c^* \in \arg\max_c \hat{P}(Y = c|x)$. Whenever a tie arises, c^* is randomly chosen from the set of maximizers.

3.1) Function `knn_predict()`

Implement a function called `knn_predict()` that generates k-NN predictions. The function `knn_fit()` should accept the following inputs:

- `X_train`: an $n_{\text{train}} \times p$ predictor matrix
- `X_test`: an $n_{\text{test}} \times p$ predictor matrix
- `y_train`: the response vector for the training set
- `k`: the number of neighbors

`knn_predict()` should return a factor vector containing the k-NN predictions for `y`.

3.2) Classification with k-NN

Use `knn_predict()` with training observations `training <- c(1:47, 51:97, 101:146)` and test set observations `testing <- c(48:50, 98:100, 147:150)`.

```
train_set <- iris[training, ]
test_set <- iris[testing, ]

pred_knn <- knn_predict(train_set[, -5], test_set[, -5], train_set$Species, k=1)
```

3.3) k-NN CV

The number of neighbors k is a hyperparameter that has to be tuned. Similar to other statistical learning algorithms, cross-validation can be used.

Use `knn_predict()` to implement a function called `find_kcv()` that finds the optimal k based on CV-misclassification rate. `find_kcv()` should accept the following inputs:

- `X_train`: an $n_{\text{train}} \times p$ predictor matrix
- `Y_train`: the response vector for the training set

- `k`: an integer vector, each of which is a candidate for the number of neighbors, defaulted to `1:10`
- `nfold`: a scalar indicating the number of folds

`find_k_CV()` should return a scalar, which is the optimal number of neighbors (in terms of minimizing the CV-misclassification rate)

The output of your function should be like the following:

```r
find_kcv(train_set[ , -5], train_set[ , 5])
```

```
## [1] 7
```

---

4) 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: 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, ]
```

- Train LDA based on `train_set`, and generate LDA predictions on `test_set`.
- Train QDA based on `train_set`, and generate QDA predictions on `test_set`.
- Find the number k of neighbors based on `train_set`, and then use it to generate k-NN predictions of `test_set` with `knn_predict()`.
- Compute the confusion matrix for each method. (Hint: You might find the function `table()` useful.)
- Based on the confusion matrices, compute the test error rate for each method.
- Comment on the comparison of LDA, QDA, and k-NN.