Discriminant Analysis

Ruoqing Zhu

October 06, 2024

Contents

Classification Basics

When we model the probability of Y given X, such as using a logistic regression, the approach is often called a soft classification, meaning that we do not directly produce the class label for prediction. However, we can also view the task as finding a function, with 0/1 as the output. In this case, the function is called a **classifier**:

$$f : \mathbb{R}^p \longrightarrow \{0, 1\}$$

In this case, we can directly evaluate the prediction error, which is calculated from the **0-1 loss**:

$$L\big(f(\mathbf{x}), y\big) = \begin{cases} 0 & \text{if} \quad y = f(\mathbf{x}) \\ 1 & \text{o.w.} \end{cases}$$

The goal is to minimize the overall **risk**, the integrated loss:

$$\text{R}(f) = \text{E}_{X,Y} \left[L\big(f(X), Y\big) \right]$$

Continuing the notation from the logistic regression, with $\eta(\mathbf{x}) = \text{P}(Y = 1 | X = \mathbf{x})$, we can easily see the decision rule to minimize the risk is to take the dominate label for any given $\mathbf{x}$, this leads to the **Bayes rule**:

$$f_B(\mathbf{x}) = \arg\min_f \mathrm{R}(f) = \begin{cases} 1 & \text{if} \quad \eta(\mathbf{x}) \geq 1/2 \\ 0 & \text{if} \quad \eta(\mathbf{x}) < 1/2. \end{cases} \tag{1}$$

Note that it doesn't matter when $\eta(\mathbf{x}) = 1/2$ since we will make 50% mistake anyway. The risk associated with this optimal rule is called the **Bayes risk**, which is the best risk we could achieve with a classification model with 0/1 loss.

The Bayes Rule

The essential idea of Discriminant Analysis is to estimate the densities functions of each class, and compare the densities at any given target point to perform classification. Let's construct the optimal rule from the Bayes prospective:

$$\mathrm{P}(Y = 1 | X = \mathbf{x}) = \frac{\mathrm{P}(X = \mathbf{x} | Y = 1)\mathrm{P}(Y = 1)}{\mathrm{P}(X = \mathbf{x})} \tag{2}$$

$$\mathrm{P}(Y = 0 | X = \mathbf{x}) = \frac{\mathrm{P}(X = \mathbf{x} | Y = 0)\mathrm{P}(Y = 0)}{\mathrm{P}(X = \mathbf{x})} \tag{3}$$

Lets further define marginal probabilities (**prior**) $\pi_1 = P(Y = 1)$ and $\pi_0 = 1 - \pi_1 = P(Y = 0)$, then, denote the conditional densities of X as

$$f_1 = \mathrm{P}(X = \mathbf{x} | Y = 1) \tag{4}$$

$$f_0 = \mathrm{P}(X = \mathbf{x} | Y = 0) \tag{5}$$

$$\tag{6}$$

Note that the Bayes rule suggests to make the decision 1 when $\eta(\mathbf{x}) \geq 1/2$, this is equivalent to $\pi_1 > \pi_0$. Utilizing the **Bayes Theorem**, we have

$$f_B(\mathbf{x}) = \arg\min_f \mathrm{R}(f) = \begin{cases} 1 & \text{if} \quad \pi_1 f_1(\mathbf{x}) \geq \pi_0 f_0(\mathbf{x}) \\ 0 & \text{if} \quad \pi_1 f_1(\mathbf{x}) < \pi_0 f_0(\mathbf{x}). \end{cases} \tag{7}$$

This suggests that we can model the conditional density of X given Y instead of modeling $P(Y|X)$ to make the decision.

Example: Linear Discriminant Analysis (LDA)

We create two density functions that use the **same covariance matrix**: $X_1 \sim \mathcal{N}(\mu_\infty, \pm)$ and $X_2 \sim \mathcal{N}(\mu_\in, \pm)$, with $\mu_1 = (0.5, -1)^\mathrm{T}$, $\mu_2 = (-0.5, 0.5)^\mathrm{T}$, and $\Sigma_{2\times 2}$ has diagonal elements 1 and off diagonal elements 0.5. Let's first generate some observations.

```
  library(mvtnorm)
## Warning: package 'mvtnorm' was built under R version 4.3.3
  set.seed(1)
```

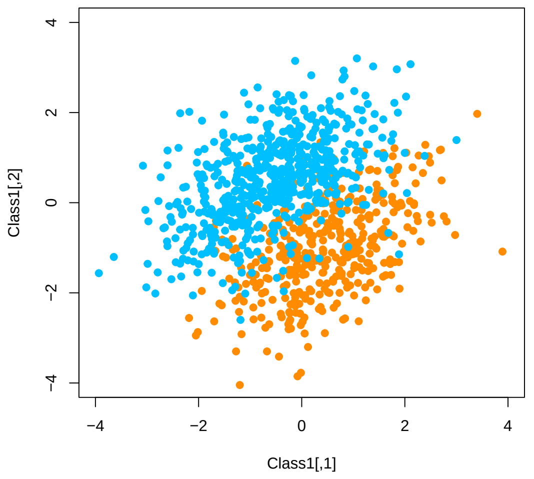
```r
# generate two sets of samples
Sigma = matrix(c(1, 0.5, 0.5, 1), 2, 2)
mu1 = c(0.5, -1)
mu2 = c(-0.5, 0.5)

# define prior
p1 = 0.4
p2 = 0.6

n = 1000

Class1 = rmvnorm(n*p1, mean = mu1, sigma = Sigma)
Class2 = rmvnorm(n*p2, mean = mu2, sigma = Sigma)

plot(Class1, pch = 19, col = "darkorange", xlim = c(-4, 4), ylim = c(-4, 4))
points(Class2, pch = 19, col = "deepskyblue")
```



Linear Discriminant Analysis

As we demonstrated earlier using the Bayes rule, the conditional probability can be formulated using Bayes Theorem. For this time, we will assume in generate that there are K classes instead of just two. However, the notation are similar to the previous case:

$$\begin{aligned} \text{P}(Y = k | X = \mathbf{x}) &= \frac{\text{P}(X = \mathbf{x} | Y = k)\text{P}(Y = k)}{\text{P}(X = \mathbf{x})} \qquad (8) \\ &= \frac{\text{P}(X = \mathbf{x} | Y = k)\text{P}(Y = k)}{\sum_{l=1}^K \text{P}(X = \mathbf{x} | Y = l)\text{P}(Y = l)} \qquad (9) \\ &= \frac{\pi_k f_k(\mathbf{x})}{\sum_{l=1}^K \pi_l f_l(\mathbf{x})} \qquad (10) \end{aligned}$$

Given any target point $\mathbf{x}$, the best prediction is simply picking the one that maximizes the posterior

$$\underset{k}{\arg\max}\ \pi_k f_k(x)$$

Both LDA and QDA model f_k as a normal density function. Suppose we model each class density as multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu}_k, \Sigma_k)$, and **assume that the covariance matrices are the same across all k, i.e.,** $\Sigma_k = \Sigma$. Then

$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^{\mathrm{T}}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right].$$

The log-likelihood function for the conditional distribution is

$$\begin{aligned}\log f_k(\mathbf{x}) =& -\log\left((2\pi)^{p/2}|\Sigma|^{1/2}\right) - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^{\mathrm{T}}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) && (11)\\ =& -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^{\mathrm{T}}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) + \text{Constant} && (12)\end{aligned}$$

The **maximum a posteriori** probability (MAP) estimate is simply

$$\begin{aligned}\widehat{f}(\mathbf{x}) &= \underset{k}{\arg\max}\ \log\left(\pi_k f_k(\mathbf{x})\right) && (13)\\ &= \underset{k}{\arg\max}\ -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^{\mathrm{T}}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) + \log(\pi_k) && (14)\end{aligned}$$

Estimating Parameters in LDA

Estimating the parameters in LDA is very simple:

- Prior probabilities: $\widehat{\pi}_k = n_k/n = n^{-1}\sum_k \mathbf{1}\{y_i = k\}$, where n_k is the number of observations in class k.
- Centroids: $\widehat{\boldsymbol{\mu}}_k = n_k^{-1}\sum_{i:\, y_i=k} x_i$
- Pooled covariance matrix:

$$\widehat{\Sigma} = \frac{1}{n-K}\sum_{k=1}^{K}\sum_{i:\, y_i=k}(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_k)(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_k)^{\mathrm{T}}$$

Let's perform this using our simulated data previously:

```r
# calculate the centers
mu1 = colMeans(Class1)
mu2 = colMeans(Class2)

# the centered data
C1_centered = scale(Class1, center = TRUE, scale = FALSE)
C2_centered = scale(Class2, center = TRUE, scale = FALSE)

# pooled covariance matrix
Sigma = ( t(C1_centered) %*% C1_centered + t(C2_centered) %*% C2_centered ) / (n- 2)

# the prior proportions
pi1 = nrow(Class1) / n
```

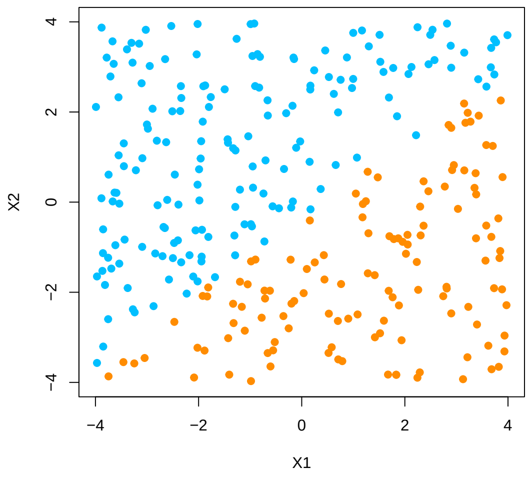
```r
pi2 = nrow(Class2) / n

# generate some new data
testdata = matrix(runif(600, -4, 4), 300, 2)

# center the testing data using mu1 or mu2
test1 = sweep(testdata, MARGIN = 2, STATS = mu1, FUN = "-")
test2 = sweep(testdata, MARGIN = 2, STATS = mu2, FUN = "-")

# calculate and compare the posteriori probability
f1 = - 0.5 * rowSums(test1 %*% solve(Sigma) * test1) + log(pi1)
f2 = - 0.5 * rowSums(test2 %*% solve(Sigma) * test2) + log(pi2)

# plot the decisions
plot(testdata, pch = 19, xlab = "X1", ylab = "X2",
     col = ifelse(f1 > f2, "darkorange", "deepskyblue"),
     xlim = c(-4, 4), ylim = c(-4, 4))
```



The Linear Decision Rule

The term $(\mathbf{x} - \boldsymbol{\mu}_k)^{\mathrm{T}}\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)$ is simply the **Mahalanobis distance** between x and the centroid $\boldsymbol{\mu}_k$ for class k. Hence, this is essentially classifying x to the class label with the closest centroid, by incorporating the covariance matrix and adjusting the for prior.

The decision boundary of LDA, as its name suggests, is a linear function of $\mathbf{x}$. To see this, let's look at the terms in the MAP. Note that anything that does not depends on the class index k is irrelevant to the decision.

$$
\begin{aligned}
& -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^{\mathrm{T}}\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) + \log(\pi_k) && (15)\\
=& \ \mathbf{x}^{\mathrm{T}}\Sigma^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^{\mathrm{T}}\Sigma^{-1}\boldsymbol{\mu}_k + \log(\pi_k) + \text{irrelevant terms} && (16)\\
=& \ \mathbf{x}^{\mathrm{T}}\mathbf{w}_k + b_k + \text{irrelevant terms} && (17)
\end{aligned}
$$

Then, the decision boundary between two classes, k and l is

$$\begin{aligned} & \mathbf{x}^{\mathrm{T}}\mathbf{w}_k + b_k = \mathbf{x}^{\mathrm{T}}\mathbf{w}_l + b_l && (18) \\ \Leftrightarrow \quad & \mathbf{x}^{\mathrm{T}}(\mathbf{w}_k - \mathbf{w}_l) + (b_k - b_l) = 0, && (19) \\ & && (20) \end{aligned}$$

which is a linear function of $\mathbf{x}$. The following density plot show exactly this effect by using the same true covariance matrix and the true centroid to calculate the densities.

LDA Example Continued

We can instead get $\mathbf{w}_k$ and b_k for each class. This gives us the same decision rule. And it is clearly linear.

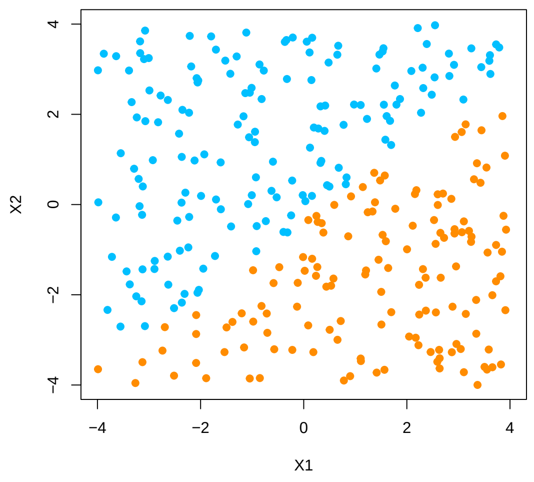
```
# calculating Wk
w1 = solve(Sigma) %*% mu1
w2 = solve(Sigma) %*% mu2

# calculating bk
b1 = - 0.5 * t(mu1) %*% solve(Sigma) %*% mu1 + log(pi1)
b2 = - 0.5 * t(mu2) %*% solve(Sigma) %*% mu2 + log(pi2)

# predicting new data
testdata = matrix(runif(600, -4, 4), 300, 2)

# calculate and compare the posteriori probability
f1 = testdata %*% w1 + as.numeric(b1)
f2 = testdata %*% w2 + as.numeric(b2)

# plot the decisions
plot(testdata, pch = 19, xlab = "X1", ylab = "X2",
     col = ifelse(f1 > f2, "darkorange", "deepskyblue"),
     xlim = c(-4, 4), ylim = c(-4, 4))
```



Example: Quadratic Discriminant Analysis (QDA)

When we assume that each class has its own covariance structure, the decision boundary may not be linear anymore. Let's visualize this by creating two density functions that use different covariance matrices. We will skip the detailed derivation of the QDA, they are available at our textbook.

Example: South Africa Heart Data

We use the South Africa heart disease data as an example. The data contains two classes.

```
    library(MASS)
##
## Attaching package: 'MASS'
## The following object is masked from 'package:plotly':
##
##     select
    library(ElemStatLearn)
    data(SAheart)
    dim(SAheart)
## [1] 462  10

    # fit lda
    heart.lda = lda(chd ~ ., data = SAheart)

    # calculate the predicted value
    heart.fitted = predict(heart.lda, data = SAheart)

    # we will not perform cross validation, but just use the fitted class
    table(heart.fitted$class, SAheart$chd)
##
##       0   1
##   0 258  73
##   1  44  87
```

The in-sample classification accuracy is 0.7467532. We can also fit the QDA model:

```
    # fit qda
    heart.qda = qda(chd ~ ., data = SAheart)

    # calculate the predicted value
    heart.fitted = predict(heart.qda, data = SAheart)

    # we will not perform cross validation, but just use the fitted class
    table(heart.fitted$class, SAheart$chd)
##
##       0   1
##   0 257  67
##   1  45  93
```

The in-sample classification error for QDA is 0.7575758.

Practice Question

The `iris` data is a classical example for classification. It contains three classes: `setosa`, `versicolor` and `virginica`, and four variables. Use these variables to perform QDA. What is the in-sample accuracy?

```r
    data("iris")

    # fit qda
    iris.qda = qda(Species ~ ., data = iris)
    qda.pred = predict(iris.qda, iris)
    table(qda.pred$class, iris$Species)

    # accuracy
    sum(diag(table(qda.pred$class, iris$Species)))/nrow(iris)
```

Example: the Hand Written Digit Data

We first sample 100 data from both the training and testing sets.

```r
    # a plot of some samples
    findRows <- function(zip, n) {
        # Find n (random) rows with zip representing 0,1,2,...,9
        res <- vector(length=10, mode="list")
        names(res) <- 0:9
        ind <- zip[,1]
        for (j in 0:9) {
        res[[j+1]] <- sample( which(ind==j), n ) }
        return(res)
    }

    set.seed(1)

    # find 100 samples for each digit for both the training and testing data
    train.id <- findRows(zip.train, 100)
    train.id = unlist(train.id)

    test.id <- findRows(zip.test, 100)
    test.id = unlist(test.id)

    X = zip.train[train.id, -1]
    Y = zip.train[train.id, 1]
    dim(X)
## [1] 1000  256

    Xtest = zip.test[test.id, -1]
    Ytest = zip.test[test.id, 1]
    dim(Xtest)
## [1] 1000  256
```

We can then fit LDA and predict.

```
    # fit LDA
    dig.lda=lda(X,Y)

    # This can be used to predict new observations
    Ytest.pred = predict(dig.lda, Xtest)
    table(Ytest, Ytest.pred$class)
##
## Ytest  0  1  2  3  4  5  6  7  8  9
##     0 92  0  2  2  0  0  1  0  3  0
##     1  0 94  0  0  4  0  2  0  0  0
##     2  2  2 66  7  5  2  4  2 10  0
##     3  2  0  3 75  2  8  0  3  6  1
##     4  0  4  2  1 76  1  3  2  2  9
##     5  2  0  3 10  0 79  0  0  3  3
##     6  0  0  4  1  3  4 86  0  1  1
##     7  0  0  0  2  5  0  0 87  0  6
##     8  2  0  4  5  6  7  1  0 72  3
##     9  0  0  0  1  4  0  0  5  0 90
    mean(Ytest != Ytest.pred$class)
## [1] 0.183
```

Practice Question

Fit QDA to this data.

```
    dig.qda = qda(X, Y) # error message

    # does not work in this case. Why did this happen?
```