Classification

Arnab Maity

NCSU Statistics ~ 5240 SAS Hall ~ amaity[at]ncsu.edu

Contents

Introduction

The problems of separating two or more groups/classes, and allocating new objects in previously defined classes are called discrimination and classification.

- **Discrimination:** finding the features that *separate* known groups in a multivariate sample. This can be either done graphically or algebraically. We try to find *discriminants* (features) whose numeric values can separate the classes as much as possible.
- **Classification:** developing a rule to *allocate* a new object into one of a number of known groups. We use such classification rules to classify objects into pre-defined classes. Here the emphasis is on defining the rule to optimally assigning objects to classes.

A classification rule is based on the features that separate the groups, so the two goals often (but not always) overlap. For example, KNN classifier gives us a rule to allocate objects to classes, but does not give us any discriminants.

Recall that we have learned about the Bayes rule/classifier before. Suppose that we have a classification problem with K classes. Here Y denotes the class label, taking possible value among $1, \ldots, K$, and $\mathbf{X} = (X_1, \ldots, X_p)^T$ denotes the set of predictors. The *Bayes classifier*, predicts a new observation $\boldsymbol{x}_0$ by $\widehat{Y}$ such that

> $\widehat{Y} = k$ if $P(Y = k|\mathbf{X} = \boldsymbol{x}_0)$ is maximum among $P(Y = 1|\mathbf{X} = \boldsymbol{x}_0), \ldots, P(Y = K|\mathbf{X} = \boldsymbol{x}_0)$.

We also know that the Bayes classifier minimizes[1] the expected prediction error for classification,

$$E[I(Y \neq \widehat{Y})].$$

[1] Interested readers can consult *Elements of Statistical Learning* by Hastie et al. (2017).

The misclassification error rate of the Bayes classifier is called the *Bayes error rate*. For a given x_0, Bayes error rate is

$$1 - max\{P(Y = 1|\mathbf{X} = \boldsymbol{x}_0), \ldots, P(Y = K|\mathbf{X} = \boldsymbol{x}_0)\}$$

The overall Bayes error rate is

$$1 - E[max\{P(Y = 1|\mathbf{X} = \boldsymbol{x}_0), \ldots, P(Y = K|\mathbf{X} = \boldsymbol{x}_0)\}].$$

The Bayes rate is analogous to the irreducible error that we encountered in the regression setting.

Thus, it is natural to try to estimate/model the conditional probabilities $P(Y = k|\mathbf{X})$ using the data, and use them to create classifiers. In this chapter, we discuss two approaches of obtaining estimates of $P(Y = k|\mathbf{X})$:

- *Directly estimating/modeling* $P(Y = k|\mathbf{X})$: An example of direct estimation of $P(Y = k|\mathbf{X})$ is the KNN classification technique, where the conditional probability is estimated by taking a majority vote from K nearest point to $\boldsymbol{x}_0$. Another example is *logistic* regression model, where the conditional probability is modeled using transformations of linear combinations of $\mathbf{X}$ of the form:[2]

$$P(Y = k|\mathbf{X}) = \frac{e^{\beta_0+X_1\beta_1+\ldots+X_p\beta_p}}{1+e^{\beta_0+X_1\beta_1+\ldots+X_p\beta_p}}.$$

  Therefore it is sufficient to estimate the coefficients $\beta_0, \beta_1, \ldots, \beta_p$ to obtain estimates of $P(Y = k|\mathbf{X})$.

- *Generative models*: In this approach, we model the distribution of $\mathbf{X}|Y = k$ for $k = 1, \ldots, K$, that is, we model how the input features are distributed/generated in each class. Then we apply *Bayes theorem*[3] to obtain expression for $P(Y = k|\mathbf{X})$. This approach is the basis of many *discriminant analysis* methods.

In what follows, we start our discussion with generative models, and then explore logistic regression.

[2] In general, we can use $g(\beta_0 + X_1\beta_1 + \ldots + X_p\beta_p)$ where $g(\cdot)$ is a known function, called a *link function*. Logistic regression uses $g(t) = e^t/(1+e^t)$. Choosing a different $g(\cdot)$ gives rise to other regression technique such as *probit* regression which uses the CDF of a standard normal distribution as $g(\cdot)$.

[3] Not to be confused with Bayes rule/-classifier. Given two events A and B, with $P(B) > 0$, Bayes theorem states:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

Generative Models

In general, we have the following setup:

- For i-th item, we observe predictors $\mathbf{X}_i = (X_{i1}, \ldots, X_{ip})^T$, and a class label Y_i (taking values in $1, 2, \ldots, K$).[4]
- The conditional density function of $\mathbf{X}_i|Y_i = k$ is $f_k(\cdot)$, $k = 1, 2, \ldots, K$.
- $\mathrm{P}(Y_i = k) = p_k$, such that $p_1 + \ldots + p_K = 1$. These are the *prior probabilities* of the classes. In other words, probability that a randomly chosen observation comes from the prior k-th class is p_k.

[4] Even though we specify the groups by "1" and "2", they are not numbers. Y is actually a categorical variable.

Assume that the sample size of the k-th group is n_k. Denote $n = n_1 + \ldots + n_K$.

A **classification rule** must give an prediction of group membership for any $\mathbf{X}$. By *Bayes theorem*, we obtain

$$P(Y_i = k|\mathbf{X}_i = \boldsymbol{x}) = \frac{p_k f_k(\boldsymbol{x})}{p_1 f_1(\boldsymbol{x}) + \ldots + p_K f_K(\boldsymbol{x})}$$

Thus, given a data vector $\boldsymbol{x}$, we can compute the posterior probability of being classified into class "k" using the formula above. Thus a sample will be classified to a class that has the *highest posterior probability*.[5]

[5] Recall that this is the Bayes rule.

Notice that the denominator in the expression of $P(Y_i = k|\mathbf{X}_i = \boldsymbol{x})$ is same for any value of k. Thus $P(Y_i = k|\mathbf{X}_i = \boldsymbol{x})$ is highest if the

numerator $p_k f_k(\boldsymbol{x})$ is highest. The the classification rule above is equivalent to assigning $\boldsymbol{x}$ to class k if $p_k f_k(\boldsymbol{x})$ is maximum.

However, the density functions $f_k(\boldsymbol{x})$ and the prior probabilities p_k are unknown. We can estimate p_k by relative proportion of the sample size of the k-th class compared to total sample size,

$$\widehat{p}_k = n_k/n.$$

For the unknown densities, we need to estimate them from the data.

Example: one predictor

To illustrate the basic ideas, consider the `wines` data set available at the UCI machine learning repository.[6]. The dataset contains quantities of 13 constituents found in each of the three types (cultivars) of wines.

[6] `https://archive.ics.uci.edu/ml/datasets/wine`; also available with the textbook Applied Multivariate Statistics with R by Zelterman

```
# Read the data
wines <- read.table("data/Wines.txt", header = TRUE)
# classes of wine
table(wines$Class)
```

```
##
##  1  2  3
## 59 71 48
```

A pairs-plot of a few variables of the wine data is shown in Figure 1. To fix our ideas, consider the `wine` data with only $K = 2$ classes (1 and 2) and with only the `Alcohol` variable.

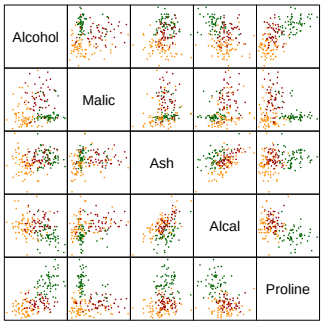

Figure 1: Pairs-plot of a few variables of the wine data.

```
# Alcohol for classes 1 and 2
alc <- wines$Alcohol[wines$Class == 1 | wines$Class == 2]
newclass <- wines$Class[wines$Class == 1 | wines$Class == 2]
# new data set
wine_small <- data.frame(Alcohol = alc,
                         Class = newclass)
```

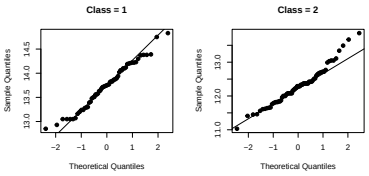

Figure 2: QQ plot for Alcohol for the two groups.

The Q-Q plots of `Alcohol` for the two groups (Figure 2) show fairly linear pattern (except may be only a few points). It is not unreasonable to assume that the data from both classes follow normal distributions with possibly different means and variances. Thus in this case we have:

- Two classes: $K = 2$
- One predictor: $X_i =$ Alcohol content of a wine sample

- The conditional density function of $X_i|Y_i = k$ is normal:

$$X_i|Y_i = 1 \sim N(\mu_1, \sigma_1^2), \quad X_i|Y_i = 2 \sim N(\mu_2, \sigma_2^2)$$

where we have possibly different means, μ_k, and variance, σ_k^2. The normal density function has the form

$$f_k(x) = \frac{1}{\sqrt{2\pi\sigma_k^2}} exp\left\{-\frac{1}{2\sigma_k^2}(x-\mu_k)^2\right\}.$$

Here the sample sizes are $n_1 = 59$ and $n_2 = 71$, with $n = 130$.

To estimate the densities, we need to estimate the unknown mean and variance parameters:

$$\widehat{\mu}_k = \text{sample mean of } X_i\text{'s from the } k\text{-th group}$$

$$\widehat{\sigma}_k^2 = \text{sample variance of } X_i\text{'s from the } k\text{-th group}$$

```
# sample means
mean_class <- aggregate(Alcohol ~ Class, mean, data = wine_small)
mean_class
```

```
##   Class  Alcohol
## 1     1 13.74475
## 2     2 12.27873
```

```
# sample variances
var_class <- aggregate(Alcohol ~ Class, var, data = wine_small)
var_class
```

```
##   Class   Alcohol
## 1     1 0.2135598
## 2     2 0.2894055
```

With the assumption of normality, the estimated density functions are

$$\widehat{f}_k(x) = \frac{1}{\sqrt{2\pi\widehat{\sigma}_k^2}} exp\left\{-\frac{1}{2\widehat{\sigma}_k^2}(x-\widehat{\mu}_k)^2\right\},$$

which are shown in Figure 3.

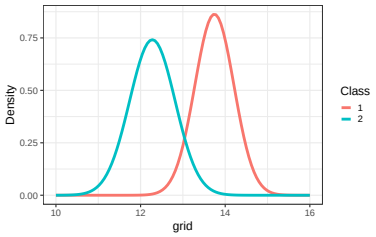

Figure 3: Estimated distribution of Alcohol for the two groups.

Finally, the estimated prior probabilities, $\widehat{p}_k$, are as follows.

```
p <- table(wine_small$Class)/nrow(wine_small)
p
```

```
##
##         1         2
## 0.4538462 0.5461538
```

Thus for a given wine sample with Alcohol value, x, we will classify the sample to group 1 if $\widehat{p}_1\widehat{f}_1(x) > \widehat{p}_2\widehat{f}_2(x)$, assign to class 2 otherwise. For example, suppose we have new data $x = 12$.

```
# new data
x <- 12
# density evaluated at x
f <- dnorm(x,
           mean = mean_class$Alcohol,
           sd = sqrt(var_class$Alcohol))
# p_k * f_k
pf <- p * f
round(pf, 3)
```

```
##
##     1     2
## 0.000 0.354
```

We see that $\widehat{p}_2\widehat{f}_2(x) > \widehat{p}_1\widehat{f}_1(x)$. Thus the sample with $x = 12$ will be assigned to class 2. The posterior probabilities are as follows.

```
post_prob <- pf / sum(pf)
round(post_prob,3)
```

```
##
##     1     2
## 0.001 0.999
```

Thus $\widehat{P}(Y = 1|X = 12) = 0.001$, and $\widehat{P}(Y = 2|X = 12) = 0.999$. We can see from Figure 3 that $x = 12$ lies in the middle of the domain of the density function of class 2 (blue). Thus it is much more likely that the new data if generated from the blue density compared to the red density.

The idea discussed above can be seen clearly if we compute the posterior probability of class "1" (or "2") for a range of values of x, see Figure 4. It seems that there is value c so that the classification rule presented before can be translated as:

Assign x to class 1 if $x > c$, assign to class 2 otherwise.

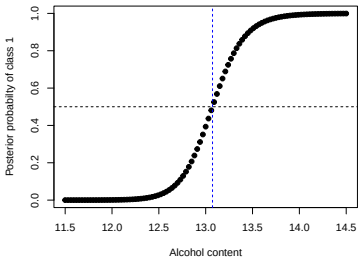

Figure 4: Estimated posterior probability of class 1 for a range of values of alcohol content.

To see this, we see that classification rule: "assign to class 1 if $\widehat{p}_1\widehat{f}_1(x) > \widehat{p}_2\widehat{f}_2(x)$" is equivalent to assigning x to class 1 if

$$log(\widehat{p}_1) + log(\widehat{f}_1(x)) > log(\widehat{p}_2) + log(\widehat{f}_2(x)).$$

Substituting the functional form of $\widehat{f}_k(x)$, we can rewrite the condition above as

$$log(\widehat{p}_1) - log(\widehat{\sigma}_1) - \frac{(x-\widehat{\mu}_1)^2}{2\widehat{\sigma}_1^2} > log(\widehat{p}_2) - log(\widehat{\sigma}_2) - \frac{(x-\widehat{\mu}_2)^2}{2\sigma_2^2}.$$

Define the functions

$$\widehat{\delta}_k(x) = log(\widehat{p}_k) - log(\widehat{\sigma}_k) - \frac{(x-\widehat{\mu}_k)^2}{2\sigma_k^2}, k = 1, 2.$$

Thus our classification rule is equivalent to assigning x to class 1 if

$$\widehat{\delta}_1(x) > \widehat{\delta}_2(x).$$

The functions $\widehat{\delta}_k(x)$ are called *discriminant functions*. Now we can determine the decision boundary of the classifier by determining the value c so that

$$\widehat{\delta}_1(c) = \widehat{\delta}_2(c).$$

Simple algebra[7] shows that we need to solve the quadratic equation

[7] We are solving a quadratic equation

$$-\left(\frac{1}{2\widehat{\sigma}_1^2} - \frac{1}{2\widehat{\sigma}_2^2}\right)c^2 + \left(\frac{\widehat{\mu}_1}{\widehat{\sigma}_1^2} - \frac{\widehat{\mu}_2}{\widehat{\sigma}_2^2}\right)c - \left(\frac{\widehat{\mu}_1^2}{2\widehat{\sigma}_1^2} - \frac{\widehat{\mu}_2^2}{2\widehat{\sigma}_2^2} + log(\widehat{p}_1/\widehat{p}_2) - log(\widehat{\sigma}_1/\widehat{\sigma}_2)\right) = 0$$

For this particular example, the two solutions are $c = 13.0729556$ and $c = 22.6722904$. However, the second solution is outside the range of the observed data, and the density values of at $c = 22.6722904$ are extremely small, 7.875424×10^{-82} and $6.5443664 \times 10^{-82}$ for class 1 and 2, respectively. Thus our actual classification boundary is given by the first solution $c = 13.0729556$. Thus. we have the classification rule:

> Assign x to class 1 if $x > 13.0729556$, assign to class 2 otherwise.

The classification method we have discovered during our exploration so far is called the *Quadratic Discriminant Analysis (QDA)*. The name is due to the fact that the discriminant functions, $\widehat{\delta}_k(x)$ are quadratic polynomials of x.

QDA can be applied to more than two classes as well. The idea remain the same – for a general value of K, we will have K discriminant functions, $\widehat{\delta}_k(x), k = 1, \ldots, K$. We classify an observation x to class k if $\delta_k(x)$ is maximum among $\delta_1(x), \ldots, \delta_K(x)$. Equivalently, $\widehat{p}_k\widehat{f}_k(x)$ is maximum. The posterior probabilities are computed as before. As an example, consider the full `wine` data with three classes. Estimates of class means and variances are shown below. The estimated density functions are shown in Figure 5.

```
# means
xbar <- aggregate(Alcohol ~ Class, mean, data = wines)
xbar <- xbar$Alcohol
xbar
```

```
## [1] 13.74475 12.27873 13.15375
```

```
# variances
vars <- aggregate(Alcohol ~ Class, var, data = wines)
vars <- vars$Alcohol
vars
```

```
## [1] 0.2135598 0.2894055 0.2811559
```

Suppose that the new data has $x = 13$. The estimated posterior probabilities are as follows.

```
x <- 13
p <- table(wines$Class)/nrow(wines)
f <- dnorm(x, mean = xbar, sd = sqrt(vars))
post_prob <- p*f / sum(p*f)
round(post_prob, 3)
```

```
##
##     1     2     3
## 0.199 0.306 0.495
```

From the results above, the new sample will be classified into class 3.

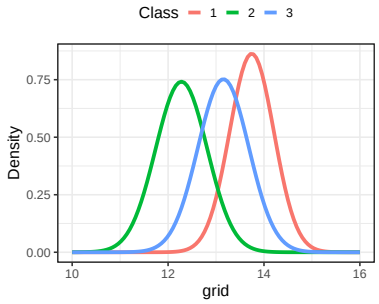

Figure 5: Estimated densities of Alcohol in the three classes in wines data, assuming normality.

Example: multiple predictors

Now let us consider the case where we have more than one predictors. For simplicity, let us start with two predictors ($p = 2$), X_{i1} and X_{i2}. To extend QDA to this setting, we need to generalize the normal distribution to two-dimensions. We call such a distribution a *bivariate normal distribution*.

In this situation, denote $\mathbf{X}_i = (X_{i1}, X_{i2})^T$. We say $\mathbf{X}_i$ is a *random vector*. We can define the mean, $\boldsymbol{\mu}$, of a random vector, $\mathbf{X}_i$, to be the vector of means of individual predictors:

$$\boldsymbol{\mu} = E(\mathbf{X}_i) = (E(X_{i1}), E(X_{i2}))^T = (\mu_1, \mu_2)^T.$$

Defining the analogous term for variance, however, requires care. For one predictor, variance can be quantified by one parameter. For two predictors, we need to look at their individual variances, $\sigma_1^2 = var(X_{i1})$ and $\sigma_2^2 = var(X_{i2})$, as well as their covariance, $\sigma_{12} = cov(X_{i1}, X_{i2})$. In general, we use the *variance-covariance matrix* of $\mathbf{X}_i$ to summarize the variability of $\mathbf{X}_i$:

$$\boldsymbol{\Sigma} = cov(\mathbf{X}_i) = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix},$$

where the diagonal entries are variances of individual predictors, and the off-diagonal entry is the covariance between the two predictors.

The random vector $\mathbf{X}_{2\times 1} = (X_1, X_2)^T$ follows a bivariate normal (Gaussian) distribution with mean vector $\boldsymbol{\mu} = (\mu_1, \mu_2)^T$ and variance-covariance (positive definite) matrix $\boldsymbol{\Sigma}$ and denoted as $\mathbf{X} \sim N_2(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ if its probability density function is[8]

$$f(\boldsymbol{x}) = (2\pi)^{-1}|\boldsymbol{\Sigma}|^{-1/2}\exp\{-(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})/2\}.$$

[8] Recall the PDF of univariate normal distribution, $N(\mu, \sigma^2)$ is

$$f_Y(y;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\{-\frac{1}{2\sigma^2}(y-\mu)^2\}$$

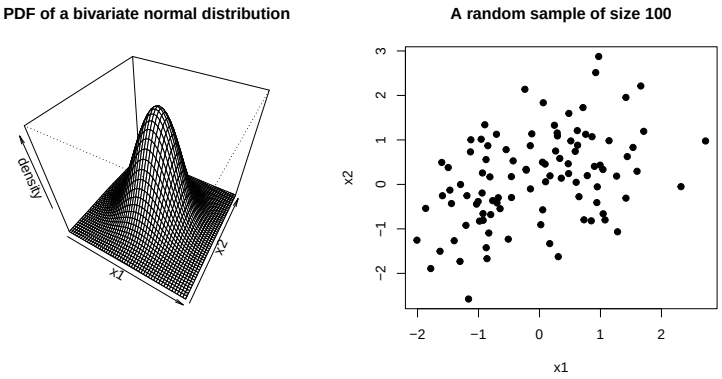

Figure 6: Density function of a bivariate normal distribution (left) and a scatterplot of a random sample of size 100 from a bivariate normal distribution.

The shape of the PDF (and that of the scatterplot of a random sample generated from the distribution) is determined by $\boldsymbol{\Sigma}$, the variance-covariance matrix of $\mathbf{X}$. An easy was to visualize the PDF of a bivariate distribution is to plot the constant probability density contours.

Constant probability density contours

We define the constant probability density contour (also called constant-density contour) of a bivariate normal PDF to be the set of vectors $\boldsymbol{x}$ such that $f(\boldsymbol{x})$ is constant, These sets are ellipses that are centered around $\boldsymbol{\mu}$.

Figures 7 – 9 show three examples of bivariate normal distribution with different variance-covariance patterns. The shape of the density function, and equivalently the contours, depend on the variance-covariance structure of X_{i1} and X_{i2}. If the two variables are uncorrelated, the major and minor axes of the elliptical contours will be parallel to the x- and y-axis. In presence of correlation, the ellipses will be oriented according the sign/magnitude of the correlation.

More generally, a random vector $\mathbf{X}_i = (X_{i1}, \ldots, X_{ip})^T$ is said for follow a multivariate normal distribution $N_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}$ is a $p \times 1$ vector and $\boldsymbol{\Sigma}$ is positive definite matrix, if the PDF of $\mathbf{X}$ is

$$f(\boldsymbol{x}) = (2\pi)^{-p/2}|\boldsymbol{\Sigma}|^{-1/2}\exp\{-(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})/2\}.$$

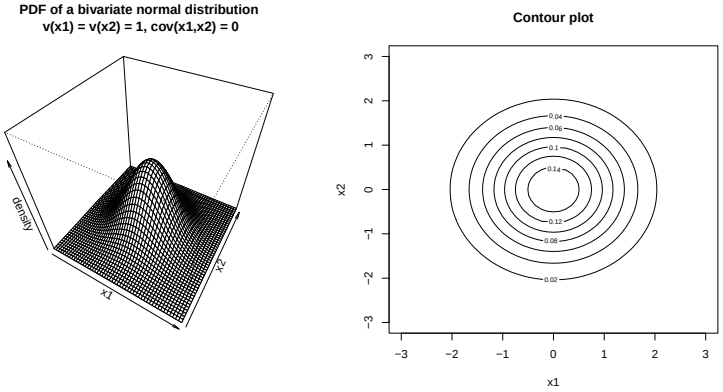

Figure 7: PDF and contours of a bivariate normal distribution with $var(X_1) = var(X_2) = 1, cov(X_1, X_2) = 0$. The contours are concentric circles since X_1 and X_2 are uncorrelated, and have the same variance.

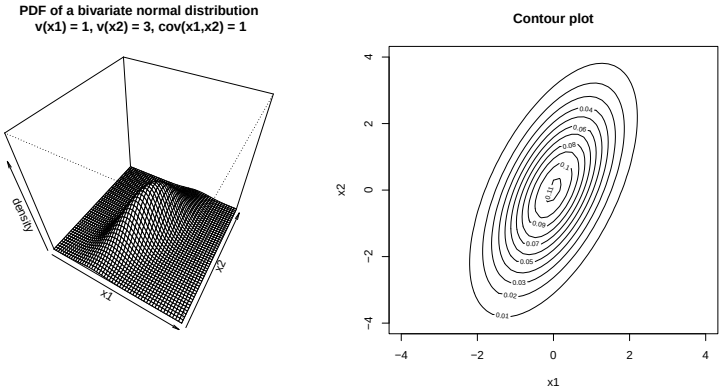

Figure 8: PDF and contours of a bivariate normal distribution with $var(X_1) = 1, var(X_2) = 3, cov(X_1, X_2) = 1$. The countours are oriented accoring to the positive correlation between X_1 and X_2. Also, the contours are narrower along X_1 axis comapred to X_2 due to $var(X_1)$ being more that $var(X_2)$.

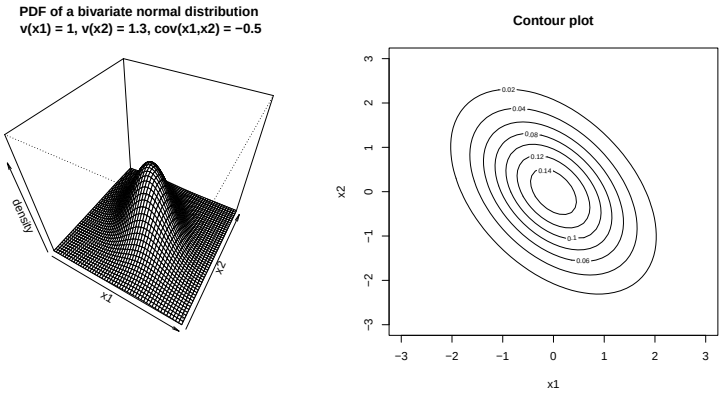

Figure 9: PDF and contours of a bivariate normal distribution with v(x1) = 1, v(x2) = 1.3, cov(x1,x2) = -0.5

We can show that $E(\mathbf{X}) = \boldsymbol{\mu}$ and that $cov(\mathbf{X}) = \boldsymbol{\Sigma}$.

To develop a classifier with p predictors, we again define the random vector $\mathbf{X}_i$ containing the p predictors for the i-th observation. we assume that

$$\mathbf{X}_i | Y = k \sim N(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \quad k = 1, \ldots, K,$$

where $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are the mean vector and variance-covariance matrix corresponding to class k.

In practice, the true values of $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are unknown, we estimate these parameters as

$$\hat{\boldsymbol{\mu}}_k = \text{ sample mean of group } k \text{ ,}$$

$$\hat{\boldsymbol{\Sigma}} = \text{ sample variance-covariance matrix of group } k.$$

The rest of the process is exactly as before: we classify a new observation with data $\boldsymbol{x}$ to class k if the estimated posterior probability of class k is highest, or equivalently, if $\widehat{p}_k \widehat{f}_k(\boldsymbol{x})$ is highest.

The discriminant functions are[9]

$$\widehat{\delta}_k(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x} - \widehat{\boldsymbol{\mu}}_k)^T \widehat{\boldsymbol{\Sigma}}_k^{-1} (\boldsymbol{x} - \widehat{\boldsymbol{\mu}}_k) - \frac{1}{2}|\widehat{\boldsymbol{\Sigma}}_k| + log(\widehat{p}_k).$$

[9] This is a quadratic function of each of the predictors in $\boldsymbol{x}$.

As before, an equivalent classification rule can be constructed using $\widehat{\delta}_k(\boldsymbol{x})$: assign $\boldsymbol{x}$ to class k is $\widehat{\delta}_k(\boldsymbol{x})$ is largest among $\widehat{\delta}_1(\boldsymbol{x}), \ldots, \widehat{\delta}_K(\boldsymbol{x})$. This is the form of QDA for multiple predictors.

Let us consider the `wines` data again with all three classes, and two predictors, `Alcohol` and `Proline`. Figure 10 shows the data for the three classes, overlayed with bivariate normal density contours for each class. The estimated mean vectors and variance-covariance matrices for each class are shown below.

```
X <- cbind(wines$Alcohol, wines$Proline)
mu <- vector("list")
Sigma <- vector("list")
for(ii in 1:3){
  mu[[ii]] <- colMeans(X[wines$Class == ii, ])
  Sigma[[ii]] <- cov(X[wines$Class == ii, ])
}
mu
```

```
## [[1]]
## [1]   13.74475 1115.71186
##
## [[2]]
## [1]  12.27873 519.50704
##
## [[3]]
## [1]  13.15375 629.89583
```

```
Sigma
```

```
## [[1]]
##            [,1]        [,2]
## [1,]  0.2135598    36.91949
## [2,] 36.9194944 49071.45003
##
## [[2]]
##           [,1]         [,2]
## [1,] 0.2894055     3.651366
## [2,] 3.6513662 24715.367807
##
## [[3]]
##            [,1]         [,2]
## [1,]  0.2811559    -5.434707
## [2,] -5.4347074 13247.329344
```

As before, for a given data point $\boldsymbol{x}$, we can compute $\widehat{p}_k \widehat{f}_k(\boldsymbol{x})$, and the associated posterior probabilities.

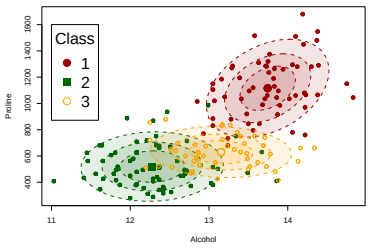

Figure 10: Three class wines data overlayed with normal density contours.

```
# For multivariate normal density
library(mnormt)
# new data
newx <- data.frame(Alcohol = 13, Proline = 600)
# p-hat
p <- table(wines$Class)/nrow(wines)
# f-hat
f <- c()
for(ii in 1:3){
  f[ii] <- dmnorm(newx, mean = mu[[ii]], varcov = Sigma[[ii]])
}
# Posterior prob
post_prob <- p*f / sum(p*f)
round(post_prob, 3)
```

```
##
##     1     2     3
## 0.027 0.290 0.683
```

In general, we can use the `qda()`[10] function in `MASS` package to build QDA models.

[10] See `?qda` for details.

```
wine_qda <- qda(Class ~ Alcohol + Proline, data = wines)
wine_qda
```

```
## Call:
## qda(Class ~ Alcohol + Proline, data = wines)
##
## Prior probabilities of groups:
##         1         2         3
## 0.3314607 0.3988764 0.2696629
##
## Group means:
##    Alcohol   Proline
## 1 13.74475 1115.7119
## 2 12.27873  519.5070
## 3 13.15375  629.8958
```

```
# prediction of new x
pred <- predict(wine_qda, newdata = newx)
pred
```

```
## $class
## [1] 3
## Levels: 1 2 3
##
## $posterior
##          1         2         3
## 1 0.027429 0.2898352 0.6827358
```

The decision boundaries for QDA are shown in Figure 11. Here the boundaries are quadratic functions of `Alcohol` and `Proline` since the discriminant functions are quadratic.

We can also estimate the test error rate of QDA when applied to wine data using data splitting methods such as CV or holdout. We can use `caret` to do so.

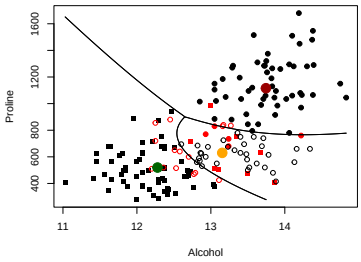

Figure 11: Decision boundary of QDA when applied to wines data.

```
set.seed(1001)
caret_qda <- train(as.factor(Class) ~ Alcohol + Proline,
                   data = wines,
                   method = "qda",
                   trControl = trainControl(method = "CV",
                                            number = 10))
caret_qda$results
```

```
##   parameter  Accuracy     Kappa AccuracySD   KappaSD
## 1      none 0.8291667 0.7402777  0.1446417 0.2199071
```

Thus the estimated test error for QDA is $1 - \text{Accuracy} = 0.171$.

The `qda()` function also has the option to perform leave-one-out cross-validation.

```
qda_cv <- qda(Class ~ Proline + Alcohol, data = wines,
              CV = TRUE)
err <- confusionMatrix(reference = as.factor(wines$Class),
                   data = qda_cv$class)
err
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 54  1  3
##          2  0 63 14
##          3  5  7 31
##
## Overall Statistics
##
##                Accuracy : 0.8315
##                  95% CI : (0.7682, 0.8833)
##     No Information Rate : 0.3989
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7424
##
##  Mcnemar's Test P-Value : 0.28
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.9153   0.8873   0.6458
## Specificity            0.9664   0.8692   0.9077
## Pos Pred Value         0.9310   0.8182   0.7209
## Neg Pred Value         0.9583   0.9208   0.8741
## Prevalence             0.3315   0.3989   0.2697
## Detection Rate         0.3034   0.3539   0.1742
## Detection Prevalence   0.3258   0.4326   0.2416
## Balanced Accuracy      0.9408   0.8782   0.7768
```

Different clasification methods

In the discussion presented in the previous section, we made the assumption that the predictors have multivariate normal density with different mean vectors and different variance-covariance matrices in each class. As a result, we developed QDA method. If we specify the densities in a different way, or put other assumptions on the

parameters, we will get different classification methods. For example, we might assume that the multivariate normal distributions have the *same variance-covariance matrix*, $\mathbf{\Sigma}$, in each class. Then it can be shown that the discriminant functions are *linear* in $\boldsymbol{x}$. Specifically,

$$\widehat{\delta}_k(\boldsymbol{x}) = \boldsymbol{x}^T \widehat{\mathbf{\Sigma}}^{-1} \widehat{\boldsymbol{\mu}}_k - \frac{1}{2} \widehat{\boldsymbol{\mu}}_k^T \widehat{\mathbf{\Sigma}}^{-1} \widehat{\boldsymbol{\mu}}_k + log(\widehat{p}_k)$$

The common variance-covariance matrix, $\mathbf{\Sigma}$, can be estimated by a "pooled" estimator.[11] The corresponding classification method is known as *Linear Discriminant Analysis (LDA)*, and the decision boundaries are linear.

[11] For K classes,

$$\hat{\mathbf{\Sigma}} = \frac{(n_1 - 1)\boldsymbol{S}_1 + \ldots + (n_K - 1)\boldsymbol{S}_K}{n_1 + \ldots + n_K - K},$$

where $\boldsymbol{S}_1, \ldots, \boldsymbol{S}_K$ are sample covariance matrices of $\mathbf{X}$'s from class $1, \ldots, K$, respectively.

In general, various classification methods specify or estimate the densities in different ways, giving rise to different classification rules. Some of these methods are shown below:

- **Linear discriminant analysis** (LDA):[12] use Gaussian densities with different means but *same* covariance matrix for each class
- **Quadratic discriminant analysis** (QDA):[13] use Gaussian densities with different means and *different* covariance matrices for each class;
- **Naive Bayes Classifier**:[14] uses estimated density assuming that the inputs are conditionally independent in each class;
- **Regularized Discriminant Analysis**:[15] using regularized group covariance matrices that are robust against multicollinearity in the data;
- **Flexible discriminant analysis**:[16] regerssion based classifier, captures nonlinear features of the covariates;
- **Mixture discriminant analysis**:[17] density of each class is modeled using a *mixture* (weighted sum) of normal densities, can model multimodal densities;
- **Kernel Density Classification**:[18] densities are estimated nonparametrically using kernel density estimation.

[12] `lda` function in `MASS` library

[13] `qda` function in `MASS` library

[14] `NaiveBayes` in `klaR` library

[15] `rda` function in `klaR` library

[16] `fda` function in `mda` library

[17] `mda` function in `mda` library

[18] `kda` function in `ks` library

Figure 12 shows the decision boundaries of a few classification methods applied to wines data. Keep in mind that there are *many* more discrimination analysis methods available in literature and in various R packages.

LDA vs. QDA

Even though QDA can be considered more general method that LDA (due to the restrictive assumption made in LDA that each class has

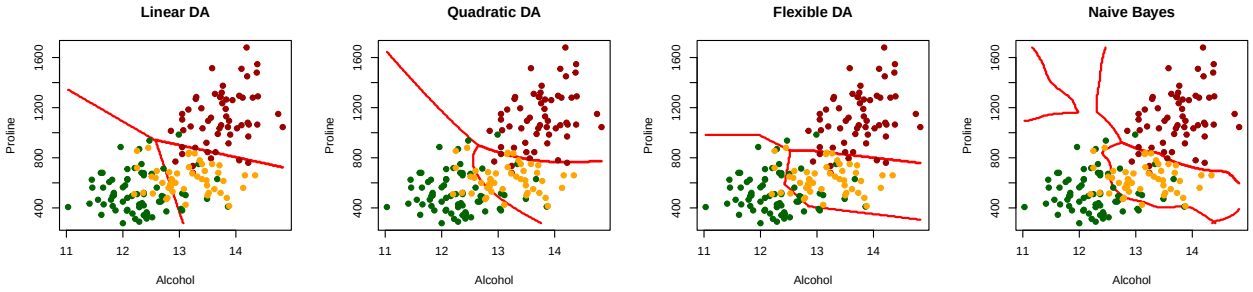

Figure 12: Classification boundaries for four classifiers.

the same variance-covariance matrix), we might still prefer LDA in some situations over QDA.

This is because QDA requires estimation of a larger number of parameters than LDA. In our wine data example, suppose we use all $p = 13$ predictors, with $K = 3$ classes. QDA estimates K variance-covariance matrix with size $p \times p$. Thus QDA estimates a total of $Kp(p+1)/2 = 273$ parameters. In comparison, LDA requires estimation of only one common variance-covariance matrix. Since LDA is a less flexible model, it may have more bias but less variance. Thus sometimes LDA might have better prediction performance than QDA.

However, we should keep in mind that if LDA's assumption of a common variance-covariance matrix of the K classes is badly violated, then LDA can suffer from high bias. Roughly speaking, LDA tends to be a better choice than QDA if training sample size is small and so reducing variance is crucial. In contrast, QDA is recommended if the training set is very large, so that the variance of the classifier is not a major concern, or if the assumption of a common covariance matrix for the K classes is clearly untenable.

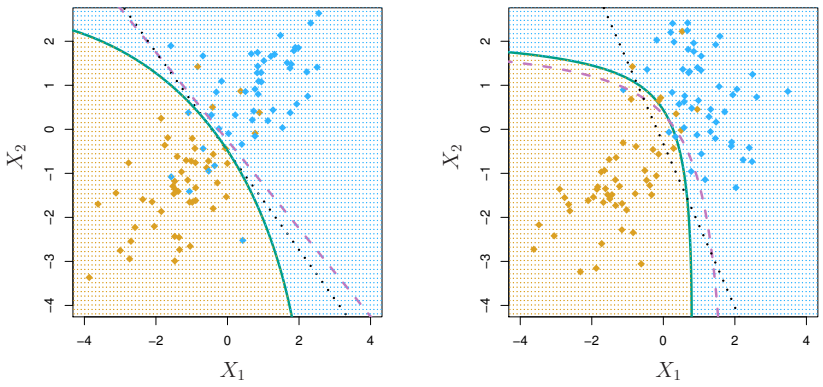

Figure 13: LDA vs QDA in two simulated example. Left panel shows a data set where each class has the same covariance matrix. The right panel shows a data set with different covariance matrices for each class. The Bayes (purple dashed), LDA (black dotted), and QDA (green solid) decision boundaries are displayed. The shading indicates the QDA decision rule

Figure 13 shows examples of simulated data sets – figure taken from *Introduction to Statistical Learning*. Each of the data sets contains two variables X_1 and X_2 and two classes. The left panel shows the data with $cor(X_1, X_2) = 0.7$ for both the classes. Thus the Bayes decision boundary is linear in this case, and LDA performs better than QDA, because QDA suffers from higher variance without a corresponding decrease in bias. The right panel shows a data set with $cor(X_1, X_2) = 0.7$ in one class and $cor(X_1, X_2) = -0.7$ in the other. Here, the assumption os a common variance-covariance matrix is not apppropriate and thus, LDA suffers from high bias but QDA performs better.

A possible disadvantage of QDA/LDA is the assumption of multivariate normality of predictors in each class. When this assumotion is unreasonable LDA/QDA can perfrom badly. However, assumption of normality is more crucial for QDA than LDA since another formulation of LDA[19] does not require normality of the predictors.

[19] Fisher, 1936

Naive Bayes classifier

Recall that LDA and QDA estimate the densities $f_k(\cdot)$ for the K classes using the multivariate normality assumption. The naive Bayes classifier makes a single assumption: the joint density of $X_{i1}, \ldots, X_{ip}$ is the product of p individual density function,

$$f_k(\boldsymbol{x}) = f_{k1}(x_1) \times \ldots \times f_{kp}(x_p).$$

In other words, within each class the p predictors are assumed to be *independent*. This is a quite strong assumption – this will also imply that there is *no relation* (linear or otherwise) between the predictors *within each class*. In most situations, this assumption is not appropriate. However, a classifier can still be constructed basec on this assumption, and it often gives good results (especially for smaller n).

With this assumption, we only need to form estimates of the *marginal density functions*, $\widehat{f}_{kj}, j = 1, \ldots, p$, to obtain an estimate of $f_k(\boldsymbol{x})$:

$$\widehat{f}_k(\boldsymbol{x}) = \widehat{f}_{k1}(x_1) \times \ldots \times \widehat{f}_{kp}(x_p).$$

The rest of the procedure is the same as before: we compute the posterior probabilites, or equivalently $\widehat{p}_k \widehat{f}_k(\boldsymbol{x})$, and classify observations accordingly.

We can form $\widehat{f}_{kj}$ by following any of the options below:

- If X_{ij} is quantitative, we can assume $X_{ij}|Y_i = k \sim N(\mu_{kj}, \sigma^2_{kj})$. Then we only need to estimate μ_{kj} and σ^2_{kj}.[20]
- Another option for quantitative predictors is to estimate the densities nonparametric methods. Examples of such estimators are

[20] This is equivalent to running QDA with a *diagonal variance-covariance matrix*.

relative frequency histogram and *kernel density estimator* – a smoothed version of histogram.

- If X_{ij} is qualitative, then we can simply take the proportion of sample observations for each value of the predictor. In other words, $\widehat{f}_{kj}$ is the estimated *probability mass function* of the j-th predictor:
$$\widehat{f}_{kj}(x_j) = \frac{1}{n_{\text{train,k}}} \sum_i I(X_{ij} = x_j),$$
where $n_{\text{train,k}}$ is the size of the training set for the k-th class.

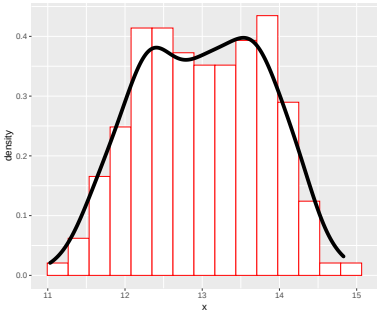

Figure 14: Relative frequancy histogram and kernel density estimator (solid line) of a sample.

In R, we can use the `NaiveBayes()` function in the `klaR` library for build a naive Bayes classifier.

```
library(klaR)
nb_wine <- NaiveBayes(as.factor(Class) ~ Alcohol + Proline,
                      data = wines,
                      usekernel = FALSE)
nb_kern <- NaiveBayes(as.factor(Class) ~ Alcohol + Proline,
                      data = wines,
                      usekernel = TRUE)
predict(nb_wine, newdata = data.frame(Alcohol = 13,
                                      Proline = 600))
```

```
## $class
## [1] 3
## Levels: 1 2 3
##
## $posterior
##                1         2         3
## [1,] 0.01007071 0.2884064 0.7015229
```

```
predict(nb_kern, newdata = data.frame(Alcohol = 13,
                                      Proline = 600))
```

```
## $class
## [1] 3
## Levels: 1 2 3
##
## $posterior
##                1         2        3
## [1,] 0.02208289 0.2097631 0.768154
```

Here we have assumed normal distribution for each predictor (`usekernel = FALSE`) in the first fit, and use kernel density estimation in the second fit (`usekernel = TRUE`). We can also use `caret` to evaluate test error with "method = nb".

Logistic Regression

One disadvantage of the classifiers discussed in the previous section is that we need to estimate the densities of the multivariate random variable $\mathbf{X}$. The QDA method for example requires multivariate normality assumption which is not reasonable if components of $\mathbf{X}$ are binary or categorical variables. The logistic regression model arises from the desire to model the posterior probabilities of each of the classes as functions of the data vector $\mathbf{X}$ without actually specifying any distribution of $\mathbf{X}$.

Usually, logistic regression models are used mostly as a tool for data analysis and inference, where the main goal is to understand the role of the predictors in explaining the outcome.

Model

For simplicity, we start with the case where we have two classes (1 and 2), and suppose for an item, we have covariate $\mathbf{X} = (X_{i1}, \ldots, X_{ip})^T$. We can model posterior probabilities of the 2 classes via linear combinations of $\mathbf{X}$, such that the probabilities sum to one. We assume that $Y|\mathbf{X}$ has a Bernoulli distribution, and model the posterior probabilities as follows:

$$P(Y = 1|\mathbf{X}) = \frac{exp(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p)}{1 + exp(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p)};$$

$$P(Y = 2|\mathbf{X}) = 1 - P(Y = 1|\mathbf{X}) = \frac{1}{1 + exp(\beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p)}.$$

Here $\beta = (\beta_1, \ldots, \beta_p)^T$ is the vector of coefficients of the covariates. Another way to write the same model is using *log-odds*:[21]

$$log\left[\frac{P(Y = 1|\mathbf{X})}{P(Y = 2|X)}\right] = \beta_0 + X_{i1}\beta_1 + \ldots + X_{ip}\beta_p.$$

[21] Thus we are modeling the log-odds of class 1 vs 2 as a linear function of $\mathbf{X}_i$.

The expression $\frac{P(Y=1|\mathbf{X})}{P(Y=2|\mathbf{X})}$ is called the *odds* of Y being 1 vs Y being 2. The parameters $\beta_0, \beta_1, \ldots, \beta_p$ quantifies the impact of the covariates to the prediction of class labels.[22] The group used in the denominator (class 2 in our formulation above) is called the *reference group*. The choice of reference group is arbitrary as the estimates of the posterior probabilities are same.

[22] We should note that the logistic regression is not just a classifier; it is a more general regression model.

Odds and log-odds

Let us understand the concepts of *odds* and *log-odds* in more detail. Let us revisit `wines` data with $K = 2$ classes (1 and 2) and only one covariate, $X =$ '*Alcohol*'. The odds are defined as

$$\frac{P(Y = 1|X)}{P(Y = 2|X)} = \frac{P(Y = 1|X)}{1 - P(Y = 1|X)}.$$

Thus, if $P(Y = 1|X) = 0.1$ leads to odds $0.1/0.9 = 1/9$. In contrast $P(Y = 1|X) = 0.9$ leads to odds $0.9/0.1 = 9$. Thus greater odds relate to higher posterior probability of class 1 (since class 2 is the reference class).

In terms of log-odds, we have the model

$$log\left[\frac{P(Y = 1|X)}{P(Y = 2|X)}\right] = \beta_0 + X_i\beta_1.$$

This model implies that with one unit increase in X (Alcohol), the log-odds will change by β_1 units. This change in log-odds does *not* depend on the value of X, that is, whether X goes from 10 to 11, or from 13 to 14, the change in log-odds stays the same, β_1.[23] Equivalently, due to one unit increase in X, the *odds* gets multiplied by e^{β_1}. Specifically,

$$\frac{P(Y = 1|X = x + 1)}{P(Y = 2|X = x + 1)} = e^{\beta_1}\frac{P(Y = 1|X = x)}{P(Y = 2|X = x)}.$$

[23] This phenomenon is similar to linear regression, where change in $E(Y)$ depends only on *change* in X, but not on the actual starting value of X.

In terms of probabilities $P(Y = 1|X)$, we note that the relation between the posterior probabilities and odds (and X) is *not* linear:

$$P(Y = 1|X) = \frac{odds}{1 + odds}.$$

Thus one unit increase in X, or equivalently β_1 unit change in odds, does not result in a constant amount of change in $P(Y = 1|X)$. The actual change in $P(Y = 1|X)$ depends on both the starting value of X as well as the change in X. For example, with $\beta_0 = 40$ and $\beta_1 = -3$, Figure 15 shows the plots of log-odds, odds, and $P(Y = 1|X)$ over a grid of values of X.

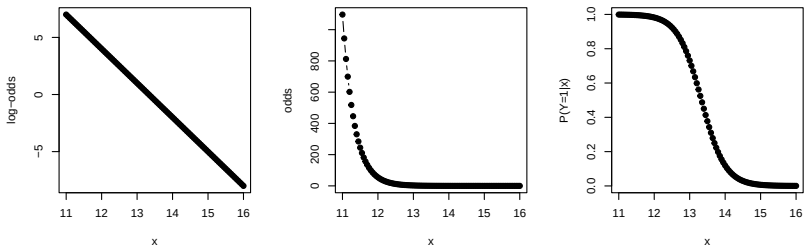

Figure 15: Plots of plots of log-odds, odds, and $P(Y = 1|X)$ over a grid of values of X.

Classification

The model parameters can be estimated directly by maximum likelihood, solution is obtained numerically by iteratively reweighted least

squares.[24] It follows that the posterior probabilities can be estimated by

$$\widehat{P}(Y=1|\mathbf{X}=\boldsymbol{x}) = \frac{exp(\widehat{\beta}_0 + x_1\widehat{\beta}_1 + \ldots x_p\widehat{\beta}_p)}{1+exp(\widehat{\beta}_0 + x_1\widehat{\beta}_1 + \ldots x_p\widehat{\beta}_p)};$$
$$\widehat{P}(Y=2|\mathbf{X}=\boldsymbol{x}) = 1 - \widehat{P}(Y=1|\boldsymbol{x}),$$

where $\widehat{\beta}_j$'s are the estimates of the regression coefficients. We can predict the class for a item with covariate $\boldsymbol{x}$ using the estimated probability that $Y=1$ as follows:

> The item is classified in group 1 if $\widehat{P}(Y=1|\boldsymbol{x}) \geq \widehat{P}(Y=2|\boldsymbol{x})$, otherwise in group 2.

Logistic regression can be performed using the `glm()` function in base R. For demonstration purposes, let us consider only two classes (1 and 2), and two covariates `Alcohol` and `Proline`. We create the small wines data – the only change is that now we explicitly change `Class` to a factor.

```
# wine data for classes 1 and 2
wine_new <- wines[wines$Class == 1 | wines$Class == 2, ]
wine_new$Class <- as.factor(wine_new$Class)
wine_new$Class <- relevel(wine_new$Class, ref = 2)
```

Note that we called the function `relevel()` with the argument `"ref = 2"`. This is done to set class 2 as the reference group. Now we can perform logistic regression using the `glm()` function.

```
# Logistic regression
wine_glm =  glm(Class ~ Proline + Alcohol,
                family = binomial(),
                data = wine_new)
```

The first part `Class ~ Proline + Alcohol` is specifying `Class` as response and `Proline` and `Alcohol` as covariates. The statement `family = binomial()` is used to perform logistic regression.[25]

The estimated coefficients are as follows.

[24] Essentially using a Newton–Raphson algorithm; see for example Hastie, Tibshirani and Friedman (2009), The Elements of Statistical Learning.

[25] It will performs linear regression using least squares without the "family = binomial()" statement.

```
wine_coef <- wine_glm$coefficients
wine_coef
```

```
##  (Intercept)      Proline      Alcohol
## -69.02763830   0.01369522   4.45310922
```

We can interpret $\widehat{\beta}_0$ as the *log-odds* when both Proline and Alcohol levels are zero. Thus we have the corresponding odds of

$1.0511812 \times 10^{-30}$. Keep in mind that in our dataset, zero values for Alcohol and Proline are not present, so such an interpretation is purely mechanical.

The estimated value of β_1 can be interpreted as the amount log-odds will *change* due to one unit *increase* in `Proline` while *keeping the `Alcohol` level fixed*. Thus keeping `Alcohol` level fixed, one unit increase in `Proline` level is associated with 0.0136952 unit change in log-odds. Equivalently, odds will change by a multiplicative factor of 1.0137894 (in other words, increase by 1.379 percent). Similar interpretation can be given for $\widehat{\beta}_2$.

Suppose we have a new sample with `Proline = 600` and `Alcohol = 13`. So here $x_1 = 600$ and $x_2 = 13$. We can compute the estimated posterior probabilities as

$$\widehat{P}(Y = 1|\boldsymbol{x} = (600, 13)) = \frac{e^{(-69.0276383+0.0136952*600+4.4531092*13)}}{1 + e^{(-69.0276383+0.0136952*600+4.4531092*13)}} = \frac{exp(-2.9200835)}{1 + exp(-2.9200835)} = 0.0511696,$$

$$\widehat{P}(Y = 2|\boldsymbol{x} = (600, 13)) = 1 - \widehat{P}(Y = 1|x_1 = 600, x_2 = 13) = 0.9488304.$$

Thus the new sample will be classified to class 2.

In R, we can simply use the `predict()` function to compute the probabilities shown above. By default, `predict()` gives the probability of *non-reference class*, class 1 in our example.[26]

[26] See `?predict.glm()` for more details.

```
newx <- data.frame(Proline = 600,
                   Alcohol = 13)
predict(wine_glm,
        newdata = newx,
        type = "response")
```

```
##         1
## 0.05116964
```

We can also view the estimated posterior probabilities of the training set using `predict()` or using the `$fitted.values` component of the fit. The probabilities for the *non-reference group* is computed by default. We have also included the prediction of the whole dataset.[27] Figure 16 shows the posterior probabilities along with the true class labels.

[27] Just as before, we relevel the factors to set "2" as reference to make the predictions comparable to the original classes.

```
# Training set estimation of P(Y = 1)
post.prob.1 = wine_glm$fitted
# Training set estimation of P(Y = 2)
post.prob.2 = 1 - post.prob.1
# Predicted groups
Y.hat = as.factor(ifelse(post.prob.1>post.prob.2, 1, 2))
```

```
Y.hat <- relevel(Y.hat, ref = 2)
df <- data.frame("Class_1" = post.prob.1,
            "Class_2" = post.prob.2,
            "Predicted" = Y.hat,
            "Observed" = wine_new$Class)
head(df, 4)
```

```
##     Class_1      Class_2 Predicted Observed
## 1 0.9998671 1.329346e-04         1        1
## 2 0.9842244 1.577560e-02         1        1
## 3 0.9969940 3.006042e-03         1        1
## 4 0.9999998 2.424491e-07         1        1
```

As an example, for the first wine sample (1st row), $\hat{P}(Y = 1|\boldsymbol{x}) = 99.987\%$ and $\hat{P}(Y = 1|\boldsymbol{x}) = 0.013\%$. Thus this particular sample will be classified to group 1. Figure 17 shows the decision boundary of the classifier, that is, all values of (`Alcohol`, `Proline`) that has the same posterior probability of being in class 1 or 2, or equivalently, odds of 1 and log-odds of 0. Formally, we have the boundary is given by all the solutions of the linear equation (the estimated formula of log-odds):

$$-69.0276383 + 0.0136952 * Proline + 4.4531092 * Alcohol = 0.$$

We can use the function `errormatrix()` in the `klaR` package to obtain the training confusion matrix.[28]

```
# Confusion matrix
err <- klaR::errormatrix(true = wine_new$Class,
                         predicted = Y.hat,
                         relative = TRUE)
round(err, 3)
```

```
##         predicted
## true        1     2 -SUM-
##    1    0.949 0.051 0.051
##    2    0.042 0.958 0.042
##    -SUM- 0.500 0.500 0.046
```

Ideally, we should use cross-validation, training-testing sets to estimate the accuracy, as we have learned before.

Hypothesis testing and confidence intervals

We can get a summary of the fit as follows.

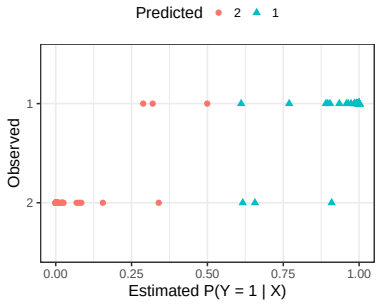

Figure 16: Esrimated posterior probabilities of class 1 along with the true class labels.

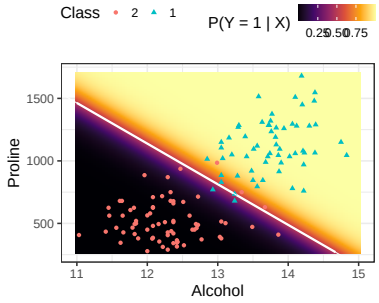

Figure 17: Decision boundary of a 2-class logistic regression based classifier.

[28] Alternatively, we can also use the function `confusionMatrix()` in the `caret` package for a detailed output.

```
# testing each beta coefficient
summary(wine_glm)
```

```
##
## Call:
## glm(formula = Class ~ Proline + Alcohol, family = binomial(),
##     data = wine_new)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.19220  -0.03196  -0.00558   0.02786   1.57716
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -69.027638  22.757675  -3.033  0.00242 **
## Proline       0.013695   0.004362   3.140  0.00169 **
## Alcohol       4.453109   1.592916   2.796  0.00518 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 179.11  on 129  degrees of freedom
## Residual deviance:  19.72  on 127  degrees of freedom
## AIC: 25.72
##
## Number of Fisher Scoring iterations: 9
```

The summary of the fit produces z-tests for coefficient of each covariate[29]; it seems both `alcohol` and `proline` are associated with the group labels. Formally, the test statistic is essentially same as in linear regression: suppose we want to test whether `Alcohol` (the second predictor with coefficient β_2) has any association with Y. Thus we test for $H_0 : \beta_2 = 0$ vs. $H_1 : \beta_2 \neq 0$. The test statistic is

$$z = \frac{\widehat{\beta}_2 - 0}{\widehat{SE}(\widehat{\beta}_2)}.$$

Using large sample theory it can be shown that z approximately follows a $N(0,1)$ distribution. Thus we reject H_0 for large positive or large negative values of z. Equivalently, the p-value can be computed as

$$p - value = 2 * P(Z > |z|),$$

where Z denotes $N(0,1)$ random variable. Figure 18 shows the p-value of a two-sided z-test.

[29] This is actually an approximate (large sample) test.

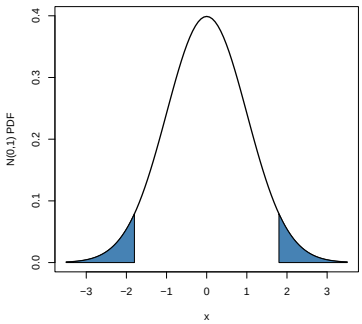

Figure 18: Two-tailed p-value for a z-test.

We can produce large sample $100(1-\alpha)\%$ confidence intervals for β_j as

$$[\widehat{\beta}_j \pm z_{1-\alpha/2} SE(\widehat{\beta}_j)],$$

where $z_{1-\alpha/2}$ is the $(1-\alpha/2)$ quantile of the $N(0,1)$ distribution. For example, a 95% confidence intervals for β_1 is

$$[0.0136952 \pm 1.96 * 0.004362] = [0.0051456, 0.0222448].$$

We can interpret this intervals as follows: with every unit increase in level of X_1 (`Proline` in our example), we can expect an *increase in log-odds* by an amount of 0.0051456 to 0.0222448. Equivalently, every unit increase in level of X_1, *odds* will be changed by a factor of 1.0051589 to 1.0224941 (in other words, increase in odds will be between 0.516 percent and 2.249 percent.)

Logistic regression with multiple classes

We can extend logistic regression presented for two classes to the case of multiple classes; the regression method is called *Multinomial Logistic Regression*. Suppose we have K classes, and we take the K-th class as the reference. The log-odds of classes vs the reference class K are modeled as follows:

$$log\left[\frac{P(Y=1|\mathbf{X}_i)}{P(Y=K|\mathbf{X}_i)}\right] = \beta_{10} + X_{i1}\beta_{11} + \ldots + X_{ip}\beta_{1p}.$$

$$log\left[\frac{P(Y=2|\mathbf{X}_i)}{P(Y=K|\mathbf{X}_i)}\right] = \beta_{20} + X_{i1}\beta_{21} + \ldots + X_{ip}\beta_{2p}.$$

$$\vdots$$

$$log\left[\frac{P(Y=K-1|\mathbf{X}_i)}{P(Y=K|\mathbf{X}_i)}\right] = \beta_{K-1,0} + X_{i1}\beta_{K-1,1} + \ldots + X_{ip}\beta_{K-1,p}.$$

Some algebra shows that the corresponding posterior probabilities are as follows:

$$P(Y=k|\mathbf{X}_i) = \frac{exp(\beta_{k0} + X_{i1}\beta_{k1} + \ldots + X_{ip}\beta_{kp})}{1 + \sum_{\ell=1}^{K-1} exp(\beta_{\ell 0} + X_{i1}\beta_{\ell 1} + \ldots + X_{ip}\beta_{\ell p})}, \quad k = 1, 2, \ldots, K-1,$$

$$P(Y=K|\mathbf{X}_i) = \frac{1}{1 + \sum_{\ell=1}^{K-1} exp(\beta_{\ell 0} + X_{i1}\beta_{\ell 1} + \ldots + X_{ip}\beta_{\ell p})}.$$

We can similarly build a classification rule as follows.

> An item with covariate $\boldsymbol{x}$ is predicted to be in class k if the estimated probability $\hat{P}(Y=k|\boldsymbol{x})$ is larger than the other posterior probabilities.

We can use the `multinom()` function in the `nnet` library to perform multinomial logistic regression. Let us consider the wine data with all the three classes.

```
library(nnet)
# Convert Class in wines data to a fcator
# and relevel to make 3 as reference
wines$Class <- as.factor(wines$Class)
wines$Class <- relevel(wines$Class, ref=3)
# multinomial logistic regression
multilogit <- multinom(Class ~ Proline + Alcohol,
                       data = wines,
                       maxit = 200, trace=FALSE)
# summary
summary(multilogit)
```

```
## Call:
## multinom(formula = Class ~ Proline + Alcohol, data = wines, maxit = 200,
##     trace = FALSE)
##
## Coefficients:
##   (Intercept)      Proline   Alcohol
## 1   -45.59355  0.017412051  2.348672
## 2    39.00287 -0.004735415 -2.824180
##
## Std. Errors:
##   (Intercept)     Proline    Alcohol
## 1 0.012260974 0.003256728 0.19427257
## 2 0.006280407 0.001748203 0.08287512
##
## Residual Deviance: 135.2578
## AIC: 147.2578
```

The option "trace = FALSE" in `multinom()` function suppresses the printing of convergence steps. The option "maxit = 200" sets the upper bound of the number of iterations to be performed to find the solutions.

The estimated posterior probabilities are as follows.

```
# estimated posterior probabilities
probs <- multilogit$fitted.values
head(probs)
```

```
##              3         1           2
## 1 1.705991e-03 0.9982906 3.365146e-06
## 2 2.430299e-02 0.9747533 9.437503e-04
## 3 2.603414e-03 0.9973369 5.972627e-05
## 4 8.947561e-07 0.9999991 1.665497e-10
## 5 7.479590e-01 0.1367403 1.153007e-01
```

```
## 6 2.248876e-06 0.9999978 7.798500e-10
```

Each row shows the estimated probability of group membership for the corresponding wine sample. For example, the first wine sample (first row), has a 99.82% probability of being in group 1. We can also visualize the posterior probabilities as shown in Figure 19.

We can also hypothesis testing for each β_{kj} using the standard errors reported in the summary output.

```
# Extract the coefficients and se
mlogit_sum <- summary(multilogit)
coef <- mlogit_sum$coefficients
se <- mlogit_sum$standard.errors
# z-statistics
z <- coef/se
z
```

```
##   (Intercept)   Proline   Alcohol
## 1   -3718.591  5.346487  12.08957
## 2    6210.246 -2.708733 -34.07754
```

```
# p-value
pv <- 2*pnorm(abs(z), lower.tail = FALSE)
pv
```

```
##   (Intercept)      Proline       Alcohol
## 1           0 8.967795e-08  1.199076e-33
## 2           0 6.754064e-03 1.587078e-254
```

We see for both classes 1 and 2, the two predictor variables are significant at any reasonable test level.

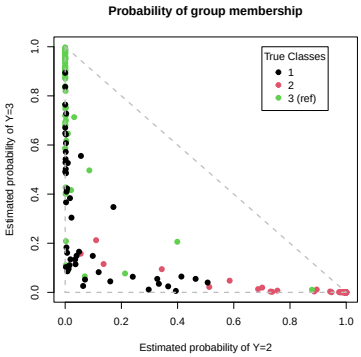

Figure 19: Estimated posterior probailities for the wine data using multinomial logistic regression.

Softmax coding

In various areas in machine learning (including deep learning) an alternative formulation of logistic regression is often used, called *softmax* coding. Instead of setting a class as reference class, and modeling the other classes against the reference class, in softmax, we treat each class symmetrically. Specifically, we posit the model

$$P(Y=k|\mathbf{X}_i) = \frac{exp(\beta_{k0} + X_{i1}\beta_{k1} + \ldots + X_{ip}\beta_{kp})}{\sum_{\ell=1}^{K} exp(\beta_{\ell 0} + X_{i1}\beta_{\ell 1} + \ldots + X_{ip}\beta_{\ell p})}, \quad k = 1, 2, \ldots, K.$$

Thus, rather than estimating coefficients for $K-1$ classes, we actually estimate coefficients for all K classes.

It can be shown that the softmax coding is equivalent to the reference class based coding described before, in the sense that the fitted

values, log odds between any pair of classes, and other key model outputs will remain the same, regardless of coding.

Issues to consider

There are some situations where logistic regression might not perform well. One such situation is *complete (or quasi-complete) separation* of the data.

This situation happens when the outcome variable separates a predictor completely. This leads to perfect prediction of the outcome by the predictor. Consider the following data set with binary response Y and two predictors X_1 and X_2. Figure 20 shows relationship between Y and the two predictors. In such a case, logistic regression may produce unreasonable over-inflated estimates of regression coefficients.

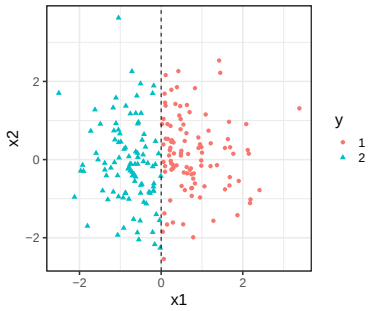

Figure 20: Simulated data set. The response separates X_1 completely, but not X_2.

```
glm(y ~ x1 + x2, family = binomial())
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##
## Call:  glm(formula = y ~ x1 + x2, family = binomial())
##
## Coefficients:
## (Intercept)           x1           x2
##       6.801     -851.898       -5.579
##
## Degrees of Freedom: 199 Total (i.e. Null);  197 Residual
## Null Deviance:       277.2
## Residual Deviance: 4.285e-07     AIC: 6
```

In general, if there is a linear combination $Z = aX_1 + bX_2$ that is completely separated by Y, logistic regression will fail to produce reasonable results. Figure 21 shows one such example where the data is completely separated by the line $X_1 + X_2 = 0$.

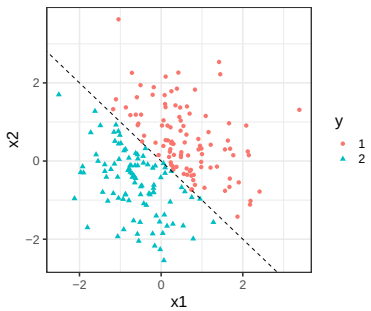

Figure 21: Simulated data set. The response separates the data completely – the boundary (dashed line) is $X_1 + X_2 = 0$. Negative values of $X_1 + X_2$ corresponds to $Y = 2$, and positive values corresponds to $Y = 1$.

```
glm(y ~ x1 + x2, family = binomial())
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##
## Call:  glm(formula = y ~ x1 + x2, family = binomial())
##
```

```
## Coefficients:
## (Intercept)           x1           x2
##       14.54     -1578.34     -1550.26
##
## Degrees of Freedom: 199 Total (i.e. Null);  197 Residual
## Null Deviance:       277.1
## Residual Deviance: 2.051e-06     AIC: 6
```

Although the examples above shows complete separation using continuous predictors, it is more like to happen when using categorical predictors coded by dummy variables. Small sample size might contribute to this problem as well. In such situations, applying other classification methods (e.g., LDA) is preferred.

Since logistic regression deals with binary outcome, often it requires more sample size that linear regression. Multinomial logit regression requires even more sample size than binary logistic regression due to the fact that it estimates parameters for multiple classes.

In presence of categorical predictor, it might happen that there are some combination of predictor and response values that are not present in the data. In such a case, logistic fit may become unstable, or might even fail to converge.

As a practical example of perfect separability, consider the wines data with two classes, "1" and "2", but with all 13 predictors. Note that glm did not converge, and produces extremely inflated standard errors.

```
wines <- read.table("data/Wines.txt", header = TRUE)
wines_all <- glm(as.factor(Class) ~ .,
                 data = wines[1:130,],
                 family = binomial())
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(wines_all)
```

```
##
## Call:
## glm(formula = as.factor(Class) ~ ., family = binomial(), data = wines[1:130,
##     ])
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -2.553e-05  -2.110e-08   2.110e-08   2.110e-08   2.919e-05
```

```
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  4.358e+02  1.067e+06   0.000        1
## Alcohol     -1.975e+01  9.891e+04   0.000        1
## Malic       -9.063e+00  5.437e+04   0.000        1
## Ash         -5.743e+01  1.170e+05   0.000        1
## Alcal        7.387e+00  1.396e+04   0.001        1
## Mg          -3.931e-01  2.462e+03   0.000        1
## Phenol       4.134e+00  2.037e+05   0.000        1
## Flav        -1.271e+01  8.464e+04   0.000        1
## Nonf         1.805e+01  7.368e+05   0.000        1
## Proan        1.448e+01  7.417e+04   0.000        1
## Color        3.624e+00  4.266e+04   0.000        1
## Hue          1.966e+01  5.242e+05   0.000        1
## Abs         -3.509e+01  9.601e+04   0.000        1
## Proline     -8.307e-02  2.426e+02   0.000        1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1.7911e+02  on 129  degrees of freedom
## Residual deviance: 4.4477e-09  on 116  degrees of freedom
## AIC: 28
##
## Number of Fisher Scoring iterations: 25
```

Comparison of a few classifiers

Chapter 4.5 of the textbook *Introduction to Statistical Learning* provides a nice comparison of the different classifiers we learned in this chapter. I recommend you read the chapter to get more insight. We will provide only a brief overview of the discussion presented in the book.

Analytically, we can compare the form of log-odds of LDA, QDA, naive Bayes and logistic regression. We can show that:

- LDA and logistic regression models the log-odds as a linear combination of the predictors.
- QDA models the log-odds as a quadratic function of the predictors.
- Naive Bayes classifier model the log-odds as a sum of non-linear functions of the predictors.

We can observe that LDA is a special case of QDA since LDA assumes the covariance matrix for each class is the same. On the other hand, any classifier with a linear decision boundary is a special case of naive Bayes. However, neither QDA nor naive Bayes is a special case of the other. Between LDA and logistic regression, we expect LDA to perform better if the assumptions for LDA are satisfied.

The textbook *Introduction to Statistical Learning* also provides numerical comparison between various classifiers discussed so far. Figure 22 shows test error rates under different scenarios – see the textbook for details for each scenarios.

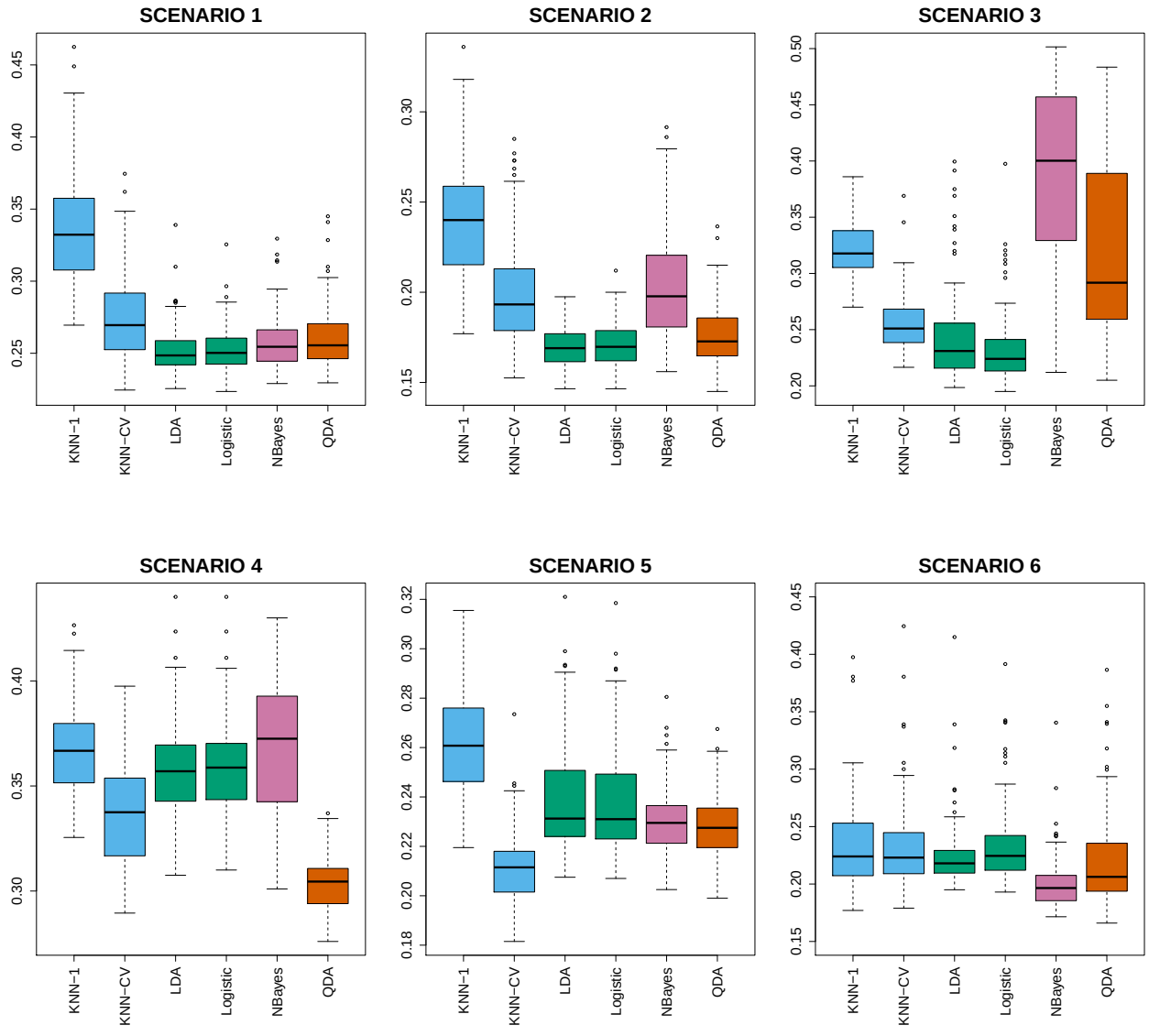

Figure 22: Test error rates for a few classifiers in linear (top row) and nonlinear (bottom row) scenarios described in *Introduction to Statistical Learning*, section 4.5.

In general, no single classifier performs best in every scenario.

Their test performance depends on the underlying structure (distribution, variance-covariance patterns) of the the data. If it often a good idea to build several classifiers, and evaluate them using their test error rate.

High-Dimensional Problems

When the number of predictors p is larger than (or close to) the sample size n, the methods described in this section suffer from numerical instability or simply can not be applied to the data.[30] We can apply similar strategies discussed for linear regression here as well: regularization/shrinkage and dimension reduction methods. Some classifier such as naive Bayes are more appropriate in hig-dimensional data than others.

For example, we can extend LDA for high-dimensional data by assuming a *diagonal* covariance matrix (i.e., assuming features are independent in each class). This method is called *Diagonal Linear Discriminant Analysis*.[31] Similar approach can be taken for QDA as well resulting in *Diagonal Quadratic Discriminant Analysis*.[32] Regularized versions of LDA such as *nearest shrunken centroids (NSC)*, *Regularized discriminant analysis (RDA)*, and many other methods are also available in literature. R packages such as `sparsediscrim`, `HDclassif`, `HiDimDA` among others have various classifiers for high-dimensional data.

Like linear regression, we can develop ridge, lasso and elastic net methods for logistic regression as well. All these methods are available in `glmnet()` package. As before, these methods shrink the regression coefficients towards zero, can be used in high-dimensional setting. We show the lasso based logistic regression fit of the two-class wines data (class 1 and 2) with the penalty parameter λ chosen by CV below.[33]

[30] Recall same issues in Linear Regression.

[31] A special case of Naive Bayes classifier, see section 18 of *Element of Statistical Learning* by Hastie et al. if you are interested in more details.

[32] Dudoit, S., Fridlyand, J., and Speed, T. P. (2002). "Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data," Journal of the American Statistical Association, 97, 457, 77-87.

[33] `glmnet()` automatically scales the predictors before estimating the regression coefficients, and then outputs the coefficients in the original scale.

```
library(glmnet)
set.seed(1102)
# CV to choose lambda
logit_cv <- cv.glmnet(x = as.matrix(wine_new[,-1]),
             y = wine_new$Class,
             family = binomial(),
             alpha = 1)
```

```
# Final fit with lambda chosen by 1-SE rule
wine_lasso <- glmnet(x = as.matrix(wine_new[,-1]),
             y = wine_new$Class,
```

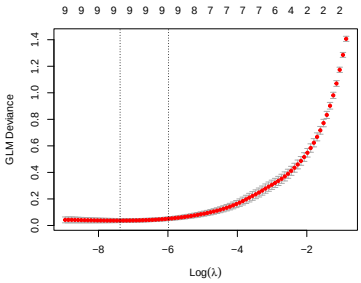

Figure 23: CV results for logistic regression of wines data with lasso penalty.

```
              family = binomial(),
              alpha = 1,
              lambda = logit_cv$lambda.1se)
```

```
# Estimated coefs
coef(wine_lasso)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) -55.12013884
## Alcohol       2.71260772
## Malic         0.90390641
## Ash           5.23709395
## Alcal        -0.55394362
## Mg            .
## Phenol        .
## Flav          0.27146489
## Nonf          .
## Proan         .
## Color         0.44396146
## Hue          -0.55507251
## Abs           2.12543584
## Proline       0.01019674
```

Dimension reduction method like PCA can still be applied to the predictors before building classifiers. PCA often is useful in providing better visualization of the data. For example, the full wines data has 13 predictors, which are hard to visualize in a plot. Let us perform PCA and plot the first few PCs along with class labels.

```
# wine data PCA
wine_pca <- prcomp(wines[, -1], scale. = TRUE)
wine_score <- wine_pca$x
```

Figure 24 shows results from LDA using only the first two PCs as predictors. We can see that PC1 and PC2 almost completely separate the three classes. Thus we have reduced dimension from 13 to two. Keep in mid that PCA is an unsupervised technique, and such a nice classification performance may not always happen. Also, even though PC1 and PC2 lead to excellent classification, together they only explain 55% variation of the data. The opposite can be true as well – the first two PCs of some data might explain a large amount of variation but fail to produce good classification results.

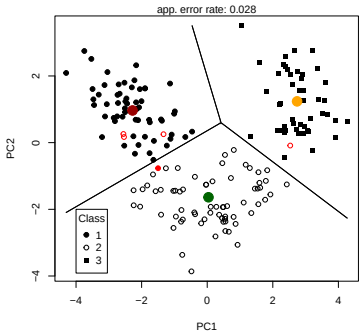

Figure 24: LDA based classification of wines data using first two PCs.

```
summary(wine_pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6     PC7
## Standard deviation     2.169 1.5802 1.2025 0.9587 0.92369 0.80103 0.74232
## Proportion of Variance 0.362 0.1921 0.1112 0.0707 0.06563 0.04936 0.04239
## Cumulative Proportion  0.362 0.5541 0.6653 0.7360 0.80163 0.85098 0.89337
##                           PC8     PC9   PC10    PC11    PC12    PC13
## Standard deviation     0.59039 0.53746 0.5008 0.47517 0.41079 0.32151
## Proportion of Variance 0.02681 0.02222 0.0193 0.01737 0.01298 0.00795
## Cumulative Proportion  0.92018 0.94240 0.9617 0.97907 0.99205 1.00000
```