Workshop: Introduction to Statistical Learning, Part III- Classification

2021-01-24

Contents

1 Introduction:

In Machine Learning we deal with different types of learning. Two major important classes of learning are called Unsupervised and Supervised learning.

The main goals in unsupervised learning are

1. Extract structure and postulate hypotheses about data generating process from observations $\mathbf{X}_1, \ldots, \mathbf{X}_n$, with $\mathbf{X}_i \in \mathbb{R}^p$.

2. Visualize, summarize and compress data.

These are called unsupervised learning because unlike supervised learning (below) there is no correct answers and there is no teacher. Learning methods are left to their own devices to discover and present the interesting structure in the data.

In supervised learning, in addition to the n observations of $\mathbf{X}$, we also have a response variable $Y \in \mathcal{Y}$. In this approach different techniques are used for prediction Y given $\mathbf{X}$. Important tools for this task are

1. **Classification**: when $Y \in \mathcal{Y}$ is categorical encoded by a discrete response, e.g., $\mathcal{Y} = \{0, 1\}$, or $\mathcal{Y} = \{1, \ldots, K\}$.

2. **Regression**: when Y is a numerical value that is observed and $\mathcal{Y} = \mathbb{R}$.

In supervised learning, given training data $(\mathbf{X}_i, Y_i)$, $i = 1, \ldots, n$, the goal is to accurately predict the class or response Y on a new observation of $\mathbf{X} = \mathbf{x}$. It is called supervised learning because the process of learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers for some of our observations (training data), the estimation (learning) method iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the proposed method achieves an acceptable level of performance.

In this chapter, that are mostly based on *An Introduction to Statistical Learning with Applications in R (ISLR)*, *ESL II* and the Machine learning course notes of Ryan Tibshirani, we learn about classification methods. Generally speaking, classification is the action of assigning an object to a category according to the characteristics of the object. In other word, classification refers to the task of analyzing a set of pre-classified data objects to learn a model (or a function) that can be used to classify an unseen data object into one of several predefined classes. Classification has various applications, such as learning from a patient database to diagnose a disease based on the symptoms of a patient, analyzing credit card transactions to identify fraudulent transactions, automatic recognition of letters or digits based on handwriting samples, and distinguishing highly active compounds from inactive ones based on the structures of compounds for drug discovery.

Before we start, let us again revisit some important differences between terminologies in Statistics and Machine learning communities:

Statistics	Machine Learning	Meaning
Classification	Supervised Learning	Predicting a discrete response from covariates
Data	Training sample	A simple random sample
Covariates	Features	X's
Classifier	Hypothesis	A map from feature space to the response space
Estimation	Learning	Finding a good classifier

2 Classification Problems

Classification, also known as *discrimination*, *pattern classification*, or *pattern recognition*, is a predictive task in which the response takes values across discrete categories, and in the most fundamental case, two categories. Classification is a *supervised learning* approach for which the true class labels for the data points are given in the training data. Our task is to analyze a set of pre-classified data objects to learn a model (or a function) that can be used to classify an unseen data object into one of several predefined classes.

1. In automated handwriting digit recognition, Y is one of the ten digits from 0 to 9. There are also 256 covariates $X_1, \ldots, X_{256}$ corresponding to the intensity value of the pixels in a say 16×16 image. Training data consists of samples of handwritten digits with their true lables.

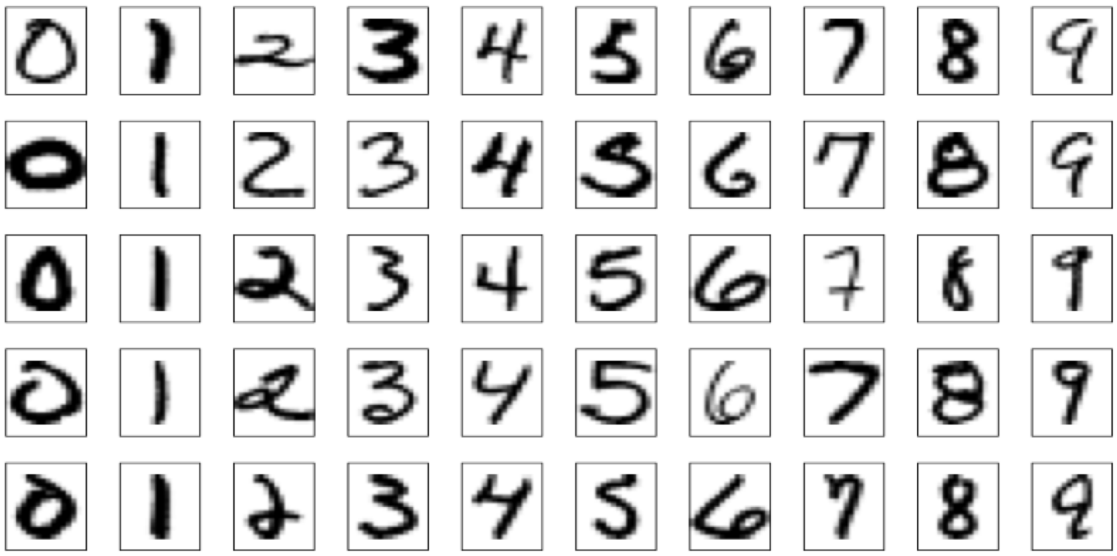


Figure 1: Training samples used in a handwriting digit recognition problem

2. Iris Flower classification: This is a famous data set which consists of 50 samples from each of three species of Iris flowers, *Iris Setosa*, *Iris virginica*, *Iris Versicolor*, see Figure 2. The length and width of the sepal and petal are measured for each specimen, and the task is to predict the species of a new Iris flower based on these features.

Figure 2: Three different species of the Iris data. Iris setosa (Left), Iris versicolor (Middle), and Iris virginica (Right)

3. Image Classification: In this application, the task is assigning an input image one label from a fixed set of categories. This is one of the core problems in Computer Vision that, despite its simplicity, has a large variety of practical applications. Here, the input consists of a training data set consisting of n images, each labeled with one of K different classes. Our task is to use the training set to learn what every one of the classes looks like. We refer to this step as training a classifier, or learning a model. In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before. We will then compare the true labels of these images to the ones predicted by the classifier. Intuitively, we're hoping that a lot of the predictions match up with the true answers (which we call the *ground truth*).

Figure 3: An example training set for four visual categories. In practice, we may have thousands of categories and hundreds of thousands of images for each category.

4. Predicting whether a patient will develop breast cancer or remain healthy, given genetic information.
5. Predicting whether or not a user will like a new product, based on his/her previous ratings.
6. Predicting the next elected president, based on various social, political, and historical measurements.

Similar to our usual setup, we observe a training data set consisting of pairs $(\mathbf{X}_i, Y_i), i = 1, \ldots, n$, where Y_i gives the class of the ith observation, and $\mathbf{X}_i \in \mathcal{X} \subseteq \mathbb{R}^p$ are the measurements of p predictor variables. A *classification rule* or *classifier* is a function $h : \mathcal{X} \to \{0, 1, \ldots, K-1\}$ where $\mathcal{X}$ is the domain of $\mathbf{X}$. When we observe $\mathbf{X} = \mathbf{x}$, we predict Y to be $h(\mathbf{x}) \in \{0, 1, \ldots, K-1\}$. For $K = 2$ we have a *binary classification* and for $K > 2$ we have a *multiclass classification* problem.

Note that although the class labels may actually be $Y_i \in \{healthy, sick\}$ or $Y_i \in \{cat, dog, mug, hat\}$, but we can always encode them as $Y_i \in \{0, 1, \ldots, K-1\}$ or $Y_i = \{1, 2, \ldots, K\}$ where K is the total number of classes.

Remark 1. *There is a big difference between classification and clustering. In clustering there is not a pre-defined notion of class membership (and sometimes, not even K), and we are not given labeled examples $(\mathbf{X}_i, Y_i)$, but only $\mathbf{X}_i, i = 1, \ldots, n$.*

Intiutively, the classification rule $h(\cdot)$ creates a partition on the input space $\mathcal{X}$. In other words, $h(\cdot)$ divides the input space (feature space) into a collection of regions, each labeled by one class. Boundaries of these regions can be rough or smooth, depending on the prediction function. For an important class of procedures these decision boundaries are linear and this is what we mean with *linear methods for classifications.*

For a binary classifictaion, the classification rule is such that $h : \mathcal{X} \to \{0, 1\}$. This is a linear classification if there exists a function $H(\mathbf{x}) = \beta_0 + \mathbf{x}^\top \boldsymbol{\beta}$ such that $h(\mathbf{x}) = I(H(\mathbf{x}) > 0)$. Here $H(\mathbf{x})$ is called a *linear discriminant function.* The decision boundary is the set

$$\{\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^p : H(\mathbf{x}) = 0\},$$

which corrsponds to a $(p-1)$-dimensional hyperplane within the p-dimensional input space $\mathcal{X}$.

Suppose we make a prediction $\hat{Y}_i(\mathbf{x}_i) = h(\mathbf{x}_i) \in \mathcal{Y}$ based on the observation $\mathbf{X}_i = \mathbf{x}_i$. We can use a loss function

$$L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+,$$

to formalize the quality of the prediction. A typical loss function is

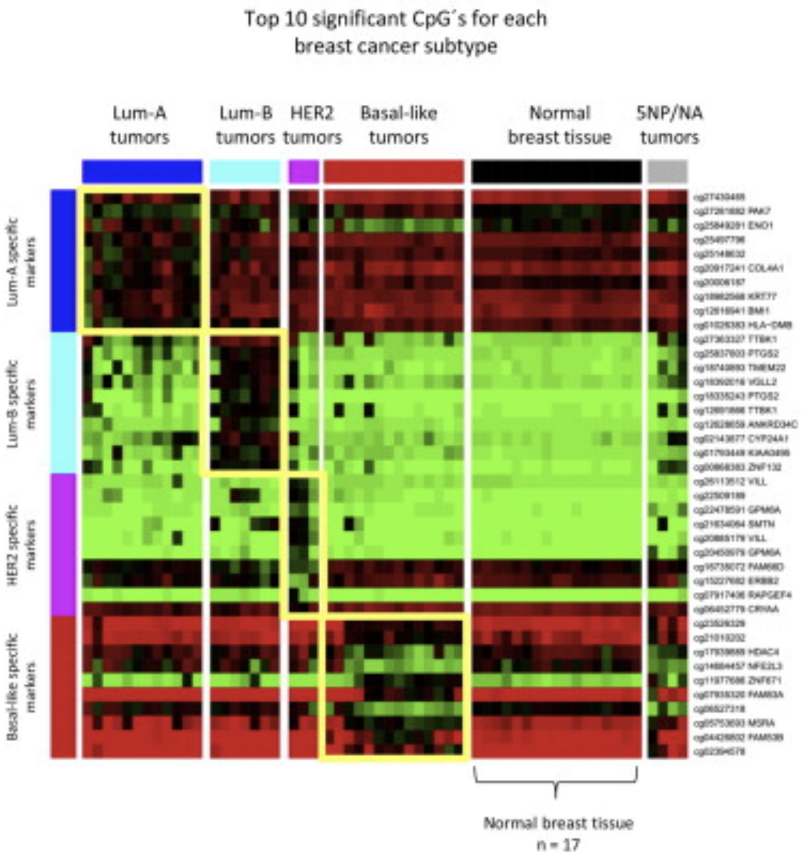


Figure 4: Top 10 significant CpG's for each breast cancer subtype

$$L(y_i, \hat{y}_i) = \begin{cases} 0 & y_i = \hat{y}_i, \\ 1 & y_i \neq \hat{y}_i. \end{cases}$$

The classification risk or the risk function R of a learner (classifier) $\hat{Y}(\mathbf{x}) = h(\mathbf{x})$ is given by the expected loss

$$R(h) = \mathbb{E}[L(Y, h(\mathbf{X}))] = \mathbb{P}(Y \neq h(\mathbf{X})),$$

where the expectation is with respect to the true (unknown) joint distribution of $\mathbf{X}$. The risk is unknown, but we can estimate it using the empirical classification error (*traning error*) rate as

$$\begin{aligned} R_n(h) &= \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(\mathbf{x}_i)) \\ &= \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq h(\mathbf{x}_i)). \end{aligned}$$

The training (classification) error rate is the proportioan oftraining observations that are misclassified by our proposed classifier. For a binary classification, let `Y.hat` be a 0-1 vector of the predicted class labels, and `y` be a 0-1 vector of the observed class labels. We can calculate the classification error rate by `mean(abs(Y.hat-y))` in R.

Test error is a much better gauge of how well classifier h generalizes to new data. The test error is in general larger than the training error and it is the error rate associated with a classifier on a new data set that has never been seen by the classifier before.

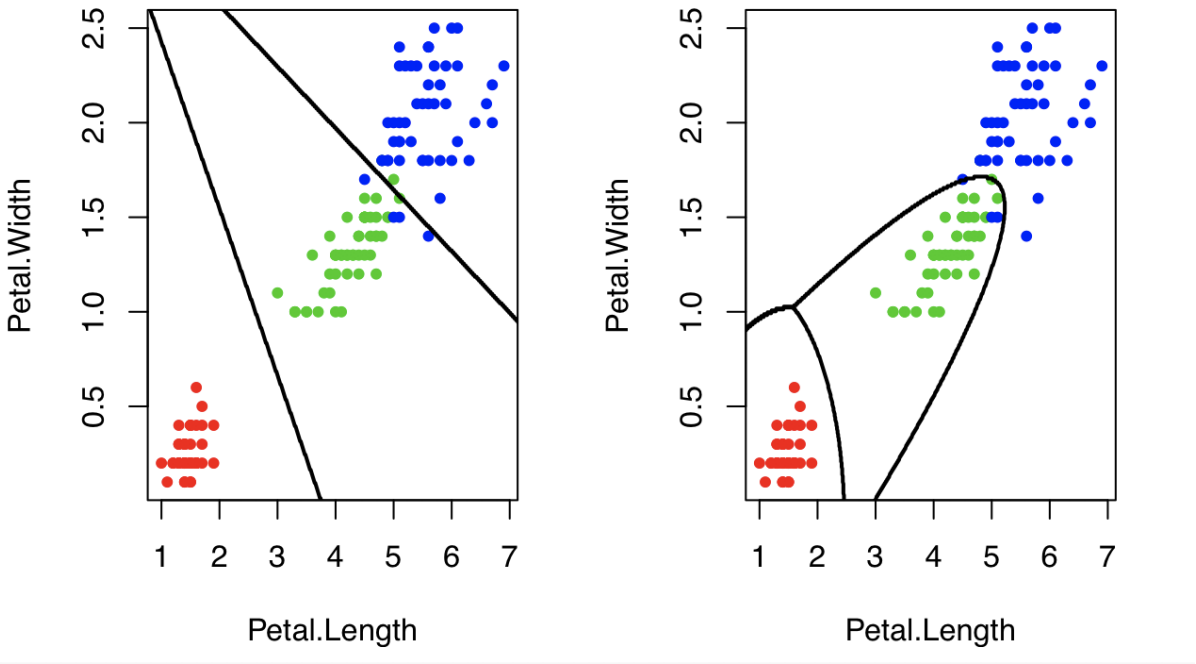


Figure 5: Linear versus non-linear boundaries in Iris flowers classification problem

Remark 2. *Re-coding the loss: If we code Y as $Y \in \{-1, 1\}$, then many classifiers can be written as*

$$h(\mathbf{x}) = sign\{H(\mathbf{x})\},$$

for some H. For linear classifiers $H(\mathbf{x}) = \beta_0 + \mathbf{x}^\top \boldsymbol{\beta}$ and the 0-1 loss function can be written as

$$I(Y \neq h(\mathbf{X})) = I(YH(\mathbf{X}) < 0),$$

and the risk is

$$R(h) = \mathbb{P}(Y \neq h(\mathbf{X})) = \mathbb{P}(YH(\mathbf{X}) < 0)).$$

This is an important remrak which will be used later on when we talk about AdaBoost and SVM approaches for classification.

2.1 Bayes Classifier for a Binary Classification Problem

If we knew the joint distribution of $(\mathbf{X}, Y)$, then we could easily find the optimal classifier.

Theorem 1. *For a binary classification, the rule $h : S_x \longrightarrow \{0, 1\}$ that minimizes $R(h)$ is given by*

$$h^*(\mathbf{x}) = \begin{cases} 1 & if\ m(\mathbf{x}) > \frac{1}{2}, \\ 0 & otherwise, \end{cases}$$

where $m(\mathbf{x}) = \mathbb{E}[Y|\mathbf{X} = \mathbf{x}] = \mathbb{P}(Y = 1|\mathbf{X} = \mathbf{x})$. The rule $h^*(\cdot)$ is called the Bayes rule. The risk $R(h^*)$ of the Bayes rule is called the Bayes risk. The set $\{\mathbf{x} \in \mathcal{X} : m(\mathbf{x}) = \frac{1}{2}\}$ is called the Bayes decision boundary.

To show the result in Theorem 1, one needs to verify that $R(h^*) \leq R(h)$ for any classifier $h \neq h^*$. To this end, observe that

$$
\begin{aligned}
R(h) &= \mathbb{P}(\{Y \neq h(\mathbf{X})\}) \\
&= \mathbb{E}_{Y,\mathbf{X}}[I\{Y \neq h(\mathbf{X})\}] \\
&= \mathbb{E}_{\mathbf{X}}\Big[\mathbb{E}_{Y|\mathbf{X}}[I\{Y \neq h(\mathbf{X})\}|\mathbf{X}]\Big] \\
&= \int \mathbb{P}(Y \neq h(\mathbf{x})|\mathbf{X} = \mathbf{x}) dP_{\mathbf{X}}(\mathbf{x}).
\end{aligned}
$$

So, it is enough to show that

$$
\mathbb{P}(Y \neq h(\mathbf{x})|\mathbf{X} = \mathbf{x}) - \mathbb{P}(Y \neq h^*(\mathbf{x})|\mathbf{X} = \mathbf{x}) \geq 0, \quad \text{for all } \mathbf{x} \in \mathcal{X}.
$$

First note that

$$
\begin{aligned}
\mathbb{P}(Y \neq h(\mathbf{x})|\mathbf{X} = \mathbf{x}) &= 1 - \mathbb{P}(Y = h(\mathbf{x})|\mathbf{X} = \mathbf{x}) \\
&= 1 - \Big(\mathbb{P}(Y = 1, h(\mathbf{X}) = 1|\mathbf{X} = \mathbf{x}) + \mathbb{P}(Y = 0, h(\mathbf{X}) = 0|\mathbf{X} = \mathbf{x})\Big) \\
&= 1 - \Big(h(\mathbf{x})\mathbb{P}(Y = 1|\mathbf{X} = \mathbf{x}) + (1 - h(\mathbf{x}))\mathbb{P}(Y = 0|\mathbf{X} = \mathbf{x})\Big) \\
&= 1 - \Big(h(\mathbf{x})m(\mathbf{x}) + (1 - h(\mathbf{x}))(1 - m(\mathbf{x}))\Big).
\end{aligned}
$$

Hence

$$
\begin{aligned}
\mathbb{P}(Y \neq h(\mathbf{x})|\mathbf{X} = \mathbf{x}) - \mathbb{P}(Y \neq h^*(\mathbf{x})|\mathbf{X} = \mathbf{x}) &= (2m(\mathbf{x}) - 1)(h^*(\mathbf{x}) - h(\mathbf{x})) \\
&= 2(m(\mathbf{x}) - \frac{1}{2})(h^*(\mathbf{x}) - h(\mathbf{x})).
\end{aligned}
$$

When $m(\mathbf{x}) \geq \frac{1}{2}$ we have $h^*(\mathbf{x}) = 1$ and since $h(\mathbf{x}) \leq 1$ the above expression is non-negative. When $m(\mathbf{x}) < \frac{1}{2}$, we have $h^*(\mathbf{x}) = 0$, then the above expression is again non-negative as $h(\mathbf{x}) \geq 0$, and this completes the proof.

One can write the Bayes classifier $h^*(\mathbf{X})$ in different ways. From the Bayes' theorem, we have

$$
\begin{aligned}
m(\mathbf{x}) &= \mathbb{P}(Y = 1|\mathbf{X} = \mathbf{x}) \\
&= \frac{p(\mathbf{x}|Y = 1)\mathbb{P}(Y = 1)}{p(\mathbf{x}|Y = 1)\mathbb{P}(Y = 1) + p(\mathbf{x}|Y = 0)\mathbb{P}(Y = 0)} \\
&= \frac{\pi_1 f_1(\mathbf{x})}{\pi_1 f_1(\mathbf{x}) + \pi_0 f_0(\mathbf{x})}
\end{aligned}
$$

where $\pi_1 = \mathbb{P}(Y = 1)$, $\pi_0 = 1 - \pi_1$ and $f_j(\mathbf{x}) = p(\mathbf{x}|Y = j)$ is the conditional distribution of $\mathbf{X}$ given $Y = j$, $j = 0, 1$. It can be verified that

$$
m(\mathbf{x}) > \frac{1}{2} \quad \text{is equivalent to} \quad \frac{f_1(\mathbf{x})}{f_0(\mathbf{x})} > \frac{\pi_0}{\pi_1},
$$

and so

$$
h^*(\mathbf{x}) = \begin{cases} 1 & if\ \frac{f_1(\mathbf{x})}{f_0(\mathbf{x})} > \frac{\pi_0}{\pi_1}, \\ 0, & otherwise. \end{cases}
$$

Equivalently,

$$m(\mathbf{x}) > \frac{1}{2} \quad \text{is equivalent to} \quad \pi_1 f_1(\mathbf{x}) > \pi_0 f_0(\mathbf{x}).$$

In other words,

$$h_j^*(\mathbf{x}) = \underset{j\in\{0,1\}}{\text{argmax}} \{\pi_j f_j(\mathbf{x})\} = \begin{cases} 1 & if\ \pi_1 f_1(\mathbf{x}) > \pi_0 f_0(\mathbf{x}), \\ 0 & otherwise. \end{cases}$$

Also, the decision boundary of the Bayes rule is

$$\mathcal{D} = \{\mathbf{x} \in S_{\mathbf{X}} : \pi_1 f_1(\mathbf{x}) = \pi_0 f_0(\mathbf{x})\}.$$

Example 1: Suppose $X_0 \sim N(1,1)$ and $X_1 \sim N(-1,1)$ represent the distributions of a feature X in two sub-populations encoded by $Y \in \mathcal{Y} = \{0,1\}$ and assume that $\pi_1 = \pi_0 = \frac{1}{2}$. Using the Bayes rule for classification, $h(x) = 1$ if $\pi_1 f_1(x) > \pi_0 f_0(x)$. After pluging in the densities and since $\pi_0 = \pi_1 = 1/2$, one can easily observe that $h(x) = 1$ if

$$(x+1)^2 < (x-1)^2 \Longrightarrow x < 0.$$

In other words, the Bayes classifier is given by

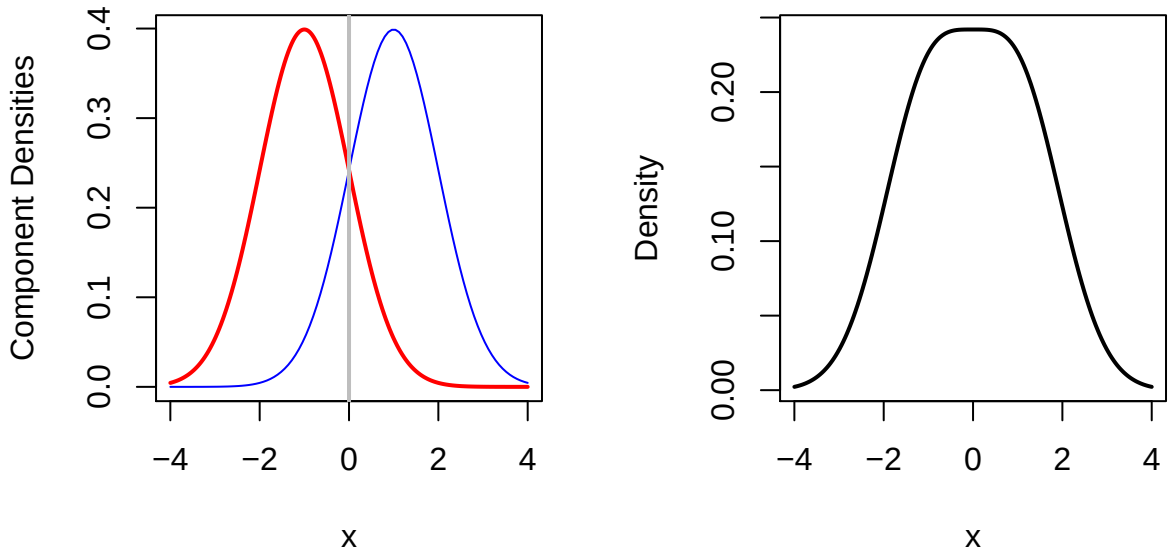
$$h^*(x) = \begin{cases} 1 & if\ x < 0, \\ 0 & if\ x \geq 0. \end{cases}$$

It is also interesting to calculate the true error rate associated with the Bayes classifier. In other word, the minimum error rate possible in this classification problem. To this end, we have

$$\begin{aligned} R(h^*) &= \mathbb{P}(h^*(X) \neq Y) \\ &= \mathbb{P}(X < 0 | Y = 0)\mathbb{P}(Y = 0) + \mathbb{P}(X \geq 0 | Y = 1)\mathbb{P}(Y = 1) \\ &= \frac{1}{2}(\mathbb{P}(Z < -1) + \mathbb{P}(Z > 1)) \\ &= \Phi(-1) = 0.159. \end{aligned}$$

This is the classification error associated with the optimum classifier under the 0-1 loss function.

```
par(mfrow=c(1, 2), pty="s")
x<-seq(-4, 4, by=0.01)
plot(x, dnorm(x, -1, 1),  type="l", lwd=2, ylab="Component Densities", col="red")
curve(dnorm(x, 1, 1), add=T, col="blue")
abline(v=0, col="grey", lwd=2)
plot(x,  0.5*(dnorm(x, -1, 1)+ dnorm(x, 1, 1)), lwd=2, ylab="Density", type="l")
```



Example 2: Suppose $X_0 \sim N(1, 1/3)$ and $X_1 \sim N(-1, 1)$ represent the distributions of a feature X in two sub-populations encoded by $Y \in \mathcal{Y} = \{0, 1\}$ and let $\pi_1 = \pi_0 = \frac{1}{2}$. Using the Bayes classifier, we have $h^*(x) = 1$ if $\pi_1 f_1(x) > \pi_0 f_0(x)$. After pluging in the densities and since $\pi_1 = \pi_0 = 1/2$, one can easily observe that $h^*(x) = 1$ if

$$\begin{aligned} e^{-\frac{(x+1)^2}{2}} > \sqrt{3} e^{-\frac{3(x-1)^2}{2}} &\Longrightarrow e^{\frac{3(x-1)^2}{2} - \frac{(x+1)^2}{2}} > \sqrt{3} \\ &\Longrightarrow e^{x^2 - 4x + 1} > \sqrt{3} \\ &\Longrightarrow x^2 - 4x + 1 - \frac{1}{2}\log 3 > 0 \\ &\Longrightarrow x < 0.116 \quad or \quad x > 3.884, \end{aligned}$$

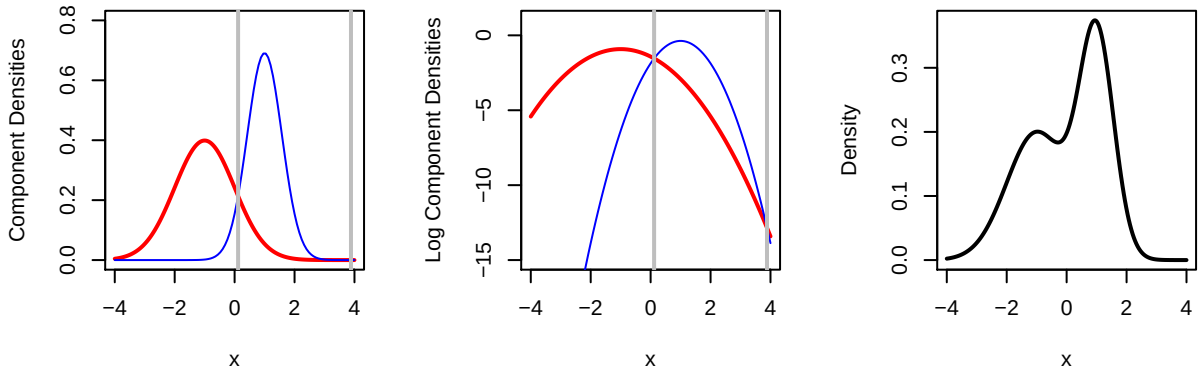
we get

$$h^*(x) = \begin{cases} 1 & if\ x < 0.116 \text{ or } x > 3.884, \\ 0 & if\ x \in [0.116, 3.884]. \end{cases}$$

Again, one can simply calculate the error rate associated with the Bayes classifier, which is given by

$$\begin{aligned} R(h^*) &= \mathbb{P}(h^*(X) \neq Y) \\ &= \frac{1}{2}\{\mathbb{P}(X < 0.116 | \mu = 1) + \mathbb{P}(X \geq 3.884 | \mu = 1)\} + \frac{1}{2}\mathbb{P}(0.116 < X < 3.884 | \mu = -1) \\ &= 0.1613. \end{aligned}$$

```
par(mfrow=c(1, 3), pty="s")
x<-seq(-4, 4, by=0.01)
plot(x, dnorm(x, -1, 1),  type="l", lwd=2, ylab="Component Densities",
     col="red", ylim=c(0, 0.8))
curve(dnorm(x, 1, sqrt(1/3)), add=T, col="blue")
abline(v=0.116, col="grey", lwd=2)
abline(v=3.884, col="grey", lwd=2)

x<-seq(-4, 4, by=0.01)
plot(x, log(dnorm(x, -1, 1)),  type="l", lwd=2, ylab="Log Component Densities",
     col="red", ylim=c(-15, 1))
curve(log(dnorm(x, 1, sqrt(1/3))), add=T, col="blue")
abline(v=3.884, col="grey", lwd=2)
abline(v=0.116, col="grey", lwd=2)


plot(x,  0.5*(dnorm(x, -1, 1)+ dnorm(x, 1, sqrt(1/3))), lwd=2, ylab="Density", type="l")
```



2.2 Bayes Classifier For Multiclass Classification Problems

We now consider the case where Y takes on more than two values, that is, $Y \in \{0, 1, \ldots, K-1\}$ for $K > 2$.

Theorem 2. *Let $R(h) = \mathbb{P}(h(\mathbf{X}) \neq Y)$ be the classification error of a classification rule $h(\mathbf{x})$. The Bayes rule $h^*(\mathbf{X})$ minimizing $R(h)$ can be written as*

$$h^*(\mathbf{x}) = \underset{j \in \{0,1,\ldots,K-1\}}{argmax} \mathbb{P}(Y = j | \mathbf{X} = \mathbf{x}).$$

To show this result, one can easily see that

$$\begin{aligned} R(h) &= 1 - \mathbb{P}(h(\mathbf{X}) = Y) \\ &= 1 - \sum_{j=0}^{K-1} \mathbb{P}(h(\mathbf{X}) = j, Y = j) \\ &= 1 - \sum_{j=0}^{K-1} \mathbb{E}_{Y,\mathbf{X}}\Big[I(h(\mathbf{X}) = j, Y = j)\Big] \\ &= 1 - \sum_{j=0}^{K-1} \mathbb{E}_{Y|\mathbf{X}}\Big[I(h(\mathbf{X}) = j)\mathbb{P}(Y = j|\mathbf{X})\Big]. \end{aligned}$$

Now, it is straighforward to see that

$$h^*(\mathbf{x}) = \underset{j \in \{0,1,\ldots,K-1\}}{\text{argmax}} \mathbb{P}(Y = j | \mathbf{X} = \mathbf{x})$$

achieves the minimum $R(h)$ and the associated minimum is

$$1 - \mathbb{E}[\max_j \mathbb{P}(Y = j|\mathbf{X})].$$

2.3 Why can't we use the best (Bayes) classification rule?

Firts, note that similar to the Binary classification, the multiclass best (Bayes) classifier $h^*(\mathbf{x}) = \underset{j=0,1,\ldots,K-1}{\text{argmax}} \mathbb{P}(Y = j|\mathbf{X} = \mathbf{x})$ can be written as

$$\begin{aligned} h^*(\mathbf{x}) &= \underset{j \in \{0,1,\ldots,K-1\}}{\text{argmax}} \mathbb{P}(Y = j|\mathbf{X} = \mathbf{x}) \\ &= \underset{j \in \{0,1,\ldots,K-1\}}{\text{argmax}} \left\{\frac{\pi_j f_j(\mathbf{x})}{f(\mathbf{x})}\right\} \\ &= \underset{j \in \{0,1,\ldots,K-1\}}{\text{argmax}} \{\pi_j f_j(\mathbf{x})\}, \end{aligned}$$

where $f(\mathbf{x}) = \sum_{k=0}^{K-1} \pi_k f_k(\mathbf{x})$ is the marginal distribution of $\mathbf{x}$. Unfortunately, the best classifier depends on unknon quantities. If we can estimate π_j and f_j then we can calculate the Bayes classification rule. Estimating π_j is easy but the more difficult problem is estimating f_j. So, in terms of ability to classify, having $f_j(\mathbf{x})$ is almost equivalent to having $\mathbb{P}(Y = j|\mathbf{X} = \mathbf{x})$. In general we need to use data to find some approximation to the Bayes rule. This can be done using different parametric and non-parametric approaches, each leading to different classification methods. Many classification techniques are based on using either models for the class densities or directly estimating $\mathbb{P}(Y = j|\mathbf{X} = \mathbf{x})$. For example,

- Linear and Quadratic discriminant analysis (LDA and QDA) use Gaussian densities for the conditional class densities $f_j(\mathbf{x})$, $j = 1, \ldots, K-1$.
- Mixture of normal densities are used for $f_j(\mathbf{x})$ to allow for nonlinear decision boundaries.
- General nonparametric density estimation approach can be used to estimate the class conditional densities.
- Naive Bayes methods are a variant of the previous case, and assume that each of the class densities are products of marginal densities; that is, they assume that inputs are conditionally independent in each class.
- Logistic regression tries to estimate $\mathbb{P}(Y = j|\mathbf{X} = \mathbf{x})$ directly and using training data.

At the risk of oversimplifying, there are three main stradegies:

1. *Empirical Risk Minimization (ERM)*: In this approach one chooses a set of classifers $\mathcal{H}$ and find $\hat{h} \in \mathcal{H}$ that minimizes some estimators of $R(h)$. Thus the learning algorithm defined by the ERM principle consists in solving some kind of optimization problem. If we do not restrict ourselves into $\mathcal{H}$ one can always overfit the data by finidng a classifier with ERM=0. This approach lays down its theorey on PAC (Probably Approximately Correct) learnability and under 0-1 loss function is known as an NP-hard problem.

2. *Density estimation*: In this approach we estimate $\mathbb{P}(Y = j|\mathbf{X} = \mathbf{x})$ by first nothing that

$$m_j(\mathbf{x}) = \mathbb{P}(Y = j|\mathbf{X} = \mathbf{x}) = \frac{\pi_j f_j(\mathbf{x})}{f(\mathbf{x})}.$$

   Let $\hat{f}_j(\mathbf{x})$ be an estimat of f_j from the $\mathbf{X}_i'$ for which $Y = j$ and $\hat{\pi}_j = \frac{\#\{Y_i = j\}}{n}$. Then, we estimate $m_j(\mathbf{x})$ by

$$\hat{m}_j(\mathbf{x}) = \hat{\mathbb{P}}(Y = j|\mathbf{X} = \mathbf{x}) = \frac{\hat{\pi}_j \hat{f}_j(\mathbf{x})}{\hat{f}(\mathbf{x})},$$

   and use this for classification. One can of course proceed differently as follow:

   - *Parametric density estimation:* Where we assume a parametric model for data in each class (e.g., LDA, QDA, ...).
   - *Nonparametric density estimation:* Where we use kernel desnity estimation or other nonparametric methods to estimate f_j's (e.g., Tree based methods, Naive Bayes, ...).

3. *Regression*: In this approach one can estimate $\mathbb{P}(Y = j|\mathbf{X} = \mathbf{x})$ using regression models such as logistic regression, or more advanced methods.

3 Classification with Regression Models

For a binary classifier problem, given $\mathbf{X}$ we only need to predict its class label $Y = 0$ or $Y = 1$. This is in contrast to a regression problem where we need to predict a real-valued response $Y \in \mathbb{R}$. Intutively, classification is a much easier task than regression. Let $m^*(\mathbf{x}) = \mathbb{E}(Y|\mathbf{X} = \mathbf{x})$ be the true regression function and let $h^*(\mathbf{x})$ be the corresponding Bayes rule. Suppose $\hat{m}(\mathbf{x})$ is an estimate of $m^*(\mathbf{x})$ and define the plug-in classification rule

$$\hat{h}(\mathbf{x}) = \begin{cases} 1, & if\ \hat{m}(\mathbf{x}) > \frac{1}{2}, \\ 0, & Otherwise. \end{cases}$$

We have the following theorem:

Theorem 3. *The risk of the plug-in classifier $\hat{h}$ satisfies*

$$R(\hat{h}) - R(h^*) \leq 2\sqrt{\int (\hat{m}(\mathbf{x}) - m^*(\mathbf{x}))^2 d\mathbb{P}_{\mathbf{X}}(\mathbf{x})}. \tag{1}$$

In other words, if the regression estimate $\hat{m}(\mathbf{x})$ is close to $m^*(\mathbf{x})$ then the plug-in classification risk will be close to the Bayes risk. The converse in *not* necessarily true. It is possible for $\hat{m}$ to be far from m^* and still lead to a good classifier. As long as $\hat{m}(\mathbf{x})$ and $m^*(\mathbf{x})$ are on the same side of $\frac{1}{2}$ they yeild the same classifier.

To show this result, we use the following observation from Theorem 3 that

$$\begin{aligned}\mathbb{P}(Y \neq \hat{h}(X)|\mathbf{X}=\mathbf{x}) - \mathbb{P}(Y \neq h^*(X)|\mathbf{X}=\mathbf{x}) &= (2\hat{m}(\mathbf{x})-1)(h^*(\mathbf{x}) - \hat{h}(\mathbf{x})) \\ &= 2|\hat{m}(x) - \frac{1}{2}|I(h^*(\mathbf{x}) \neq \hat{h}(\mathbf{x})).\end{aligned}$$

When $h^*(\mathbf{x}) \neq \hat{h}(\mathbf{x})$, there are two possible cases: (i) $\hat{h}(\mathbf{x}) = 1$ and $h^*(\mathbf{x}) = 0$; (ii) $\hat{h}(\mathbf{x}) = 0$ and $h^*(\mathbf{x}) = 1$. In both cases we have

$$|\hat{m}(\mathbf{x}) - m^*(\mathbf{x})| > |\hat{m}(\mathbf{x}) - \frac{1}{2}|.$$

Therefore,

$$\begin{aligned}\mathbb{P}(\hat{h}(\mathbf{X}) \neq Y) - \mathbb{P}(h^*(\mathbf{X}) \neq Y) &= 2\int |\hat{m}(x) - \frac{1}{2}|I(h^*(\mathbf{x}) \neq \hat{h}(\mathbf{x}))d\mathbb{P}_{\mathbf{X}}(\mathbf{x}) \\ &\leq 2\int |\hat{m}(\mathbf{x}) - m^*(\mathbf{x})|I(h^*(\mathbf{x}) \neq \hat{h}(\mathbf{x}))d\mathbb{P}_{\mathbf{X}}(\mathbf{x}) \\ &\leq 2\int |\hat{m}(\mathbf{x}) - m^*(\mathbf{x})|d\mathbb{P}_{\mathbf{X}}(\mathbf{x}) \\ &\leq 2\sqrt{\int (\hat{m}(\mathbf{x}) - m^*(\mathbf{x}))^2 d\mathbb{P}_{\mathbf{X}}(\mathbf{x})}.\end{aligned}$$

4 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis assumes multivariate normal distributions for data within each class. In this approach it is assumed that each class have its own mean $\mu_j \in \mathbb{R}^p$, but they all have a common covariance matrix $\Sigma \in \mathbb{R}^{p\times p}$. As we show below, when we calculate the classifier rule, equal covariance matrices cause the normalization factors, as well as the quadratic part in the exponents, to be independent of j. This also implies that the decision boundaries between class k and j become linear in $\mathbf{x}$; in p-dimension a hyperplane. In LDA one assumes that

$$f_j(\mathbf{x}) = \mathbb{P}(\mathbf{X}=\mathbf{x}|Y=j) = N_p(\mu_j, \Sigma),$$

with the following pdf

$$f_j(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu_j)^\top \Sigma^{-1}(\mathbf{x}-\mu_j)\right\}.$$

Now, the goal is to find j so that $\mathbb{P}(Y=j|\mathbf{X}=\mathbf{x})\cdot\pi_j \propto f_j(\mathbf{x})\cdot\pi_j$ is the largest. Since $\log(\cdot)$ is a monotone function, we can consider maximizing $\log(\pi_j \cdot f_j(\mathbf{x}))$ over $j = 1, \ldots, K$. We can define LDA rule as

$$
\begin{aligned}
f^{LDA}(\mathbf{x}) &= \underset{j=1,\ldots,K}{\operatorname{argmax}} \{\log(\pi_j \cdot f_j(\mathbf{x}))\} \\
&= \underset{j=1,\ldots,K}{\operatorname{argmax}} \left\{ \log \left[\frac{\pi_j}{(2\pi)^{p/2}|\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2}(\mathbf{x}-\mu_j)^\top \Sigma^{-1}(\mathbf{x}-\mu_j) \right\} \right] \right\} \\
&= \underset{j=1,\ldots,K}{\operatorname{argmax}} \left\{ -\frac{1}{2}(\mathbf{x}-\mu_j)^\top \Sigma^{-1}(\mathbf{x}-\mu_j) + \log(\pi_j) \right\} \\
&= \underset{j=1,\ldots,K}{\operatorname{argmax}} \left\{ \mathbf{x}^\top \Sigma^{-1}\mu_j - \frac{1}{2}\mathbf{x}^\top \Sigma^{-1}\mathbf{x} - \frac{1}{2}\mu_j^\top \Sigma^{-1}\mu_j + \log(\pi_j) \right\} \\
&= \underset{j=1,\ldots,K}{\operatorname{argmax}} \left\{ \mu_j^\top \Sigma^{-1}\mathbf{x} - \frac{1}{2}\mu_j^\top \Sigma^{-1}\mu_j + \log(\pi_j) \right\} \\
&= \underset{j=1,\ldots,K}{\operatorname{argmax}} \ \delta_j(\mathbf{x}).
\end{aligned}
$$

We call $\delta_j(\mathbf{x})$, $j = 1, \ldots, K$ the discriminant functions. Note that

$$\delta_j(\mathbf{x}) = \mu_j^\top \Sigma^{-1}\mathbf{x} - \frac{1}{2}\mu_j^\top \Sigma^{-1}\mu_j + \log(\pi_j) = a_j + b_j^\top \mathbf{x},$$

is just an affine function of $\mathbf{x}$, where $a_j = -\frac{1}{2}\mu_j^\top \Sigma^{-1}\mu_j + \log(\pi_j)$ and $b_j^\top = \mu_j^\top \Sigma^{-1}$.

In practice we do not know the parameters of the Gaussian distributions, and we will estimate them using a training sample $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and $y_i \in \{1, \ldots, K\}$, $i = 1, \ldots, n$. To this end we use

1. $\hat{\pi}_j = \frac{n_j}{n}$, the sample proportion of observations in class j.
2. $\hat{\mu_j} = \frac{1}{n_j} \sum_{y_i = j} \mathbf{x}_i$, as the mean of feature values for observations in class j.
3. $\hat{\Sigma} = \frac{1}{n-K} \sum_{j=1}^{K} \sum_{y_i = j} (\mathbf{x}_i - \hat{\mu}_j)(\mathbf{x}_i - \hat{\mu}_j)^\top$, as the pooled sample covariance matrix.

This gives estimated discriminant functions:

$$\hat{\delta}_j(\mathbf{x}) = \hat{\mu}_j^\top \hat{\Sigma}^{-1}\mathbf{x} - \frac{1}{2}\hat{\mu}_j^\top \hat{\Sigma}^{-1}\hat{\mu}_j + \log(\hat{\pi}_j) = \hat{a}_j + \hat{b}_j^\top \mathbf{x},$$

and the LDA rule reduces to

$$\hat{h}^{LDA}(\mathbf{x}) = \underset{j=1,\ldots,K}{\operatorname{argmax}} \ \hat{\delta}_j(\mathbf{x}).$$

5 Quadratic discriminant analysis

If Σ_j, $j = 1, \ldots, K$ are not assumed to be equal, then the convenient cancellations in our derivations of LDA do not occur. This results in the quadratic pieces in $\mathbf{x}$ end up remaining leading to a quadratic discriminant functions (QDA). QDA is similar to LDA except a covariance matrix must be estimated for each class j. We get the quadratic discriminant functions as

$$\delta_j(\mathbf{x}) = -\frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(\mathbf{x}-\mu_j)^\top \Sigma^{-1}(\mathbf{x}-\mu_j) + \log \pi_j.$$

The decision boundary between each pairs of classes k and l is described by a quadratic equation

$$\{\mathbf{x} \in \mathbb{R}^p : \delta_j(\mathbf{x}) = \delta_k(\mathbf{x})\}.$$

It is important to note that LDA & QDA have assumptions that are often restrictive:

1. Both LDA and QDA assume the the predictor variables $\mathbf{X}$ are drawn from a multivariate Gaussian distribution.

2. LDA assumes equality of covariances among the predictor variables $\mathbf{X}$ across all levels of Y. This assumption is relaxed with the QDA model.

3. LDA and QDA require the number of predictor variables (p) to be less then the sample size (n). Furthermore, its important to keep in mind that performance will severely decline as p approaches n. A simple rule of thumb is to use LDA & QDA on data sets where $n \geq 5 \times p$.

Also, when considering between LDA & QDA it is important to know that LDA is a much less flexible classifier than QDA, and so has substantially lower variance. This can potentially lead to improved prediction performance. But there is a trade-off: if LDA's assumption that the the predictor variable share a common variance across each Y response class is badly off, then LDA can suffer from high bias. Roughly speaking, LDA tends to be a better bet than QDA if there are relatively few training observations 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 is clearly untenable.

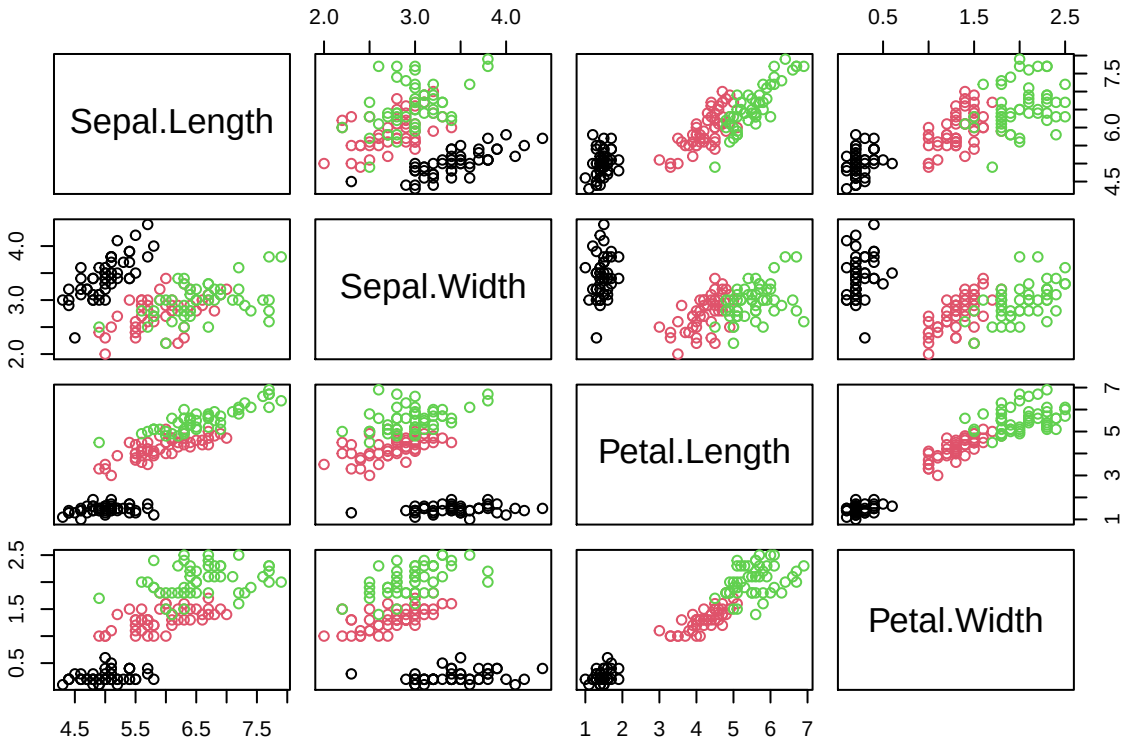
Example (IRIS Data): Here we consider Iris data set which perhaps is the best known database in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other. Predicted attribute is the class of iris plant. Data set includes the following variables:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class: – Iris Setosa – Iris Versicolour – Iris Virginica

```
data(iris)
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
ct<-rep(1:3, each=50)
pairs(iris[,1:4], col=ct)
```



We we only use the width and length as feature values to predict class type of iris plants. We use both LDA and QDA approaches. We also report prediction errors based on LDA and QDA methods:

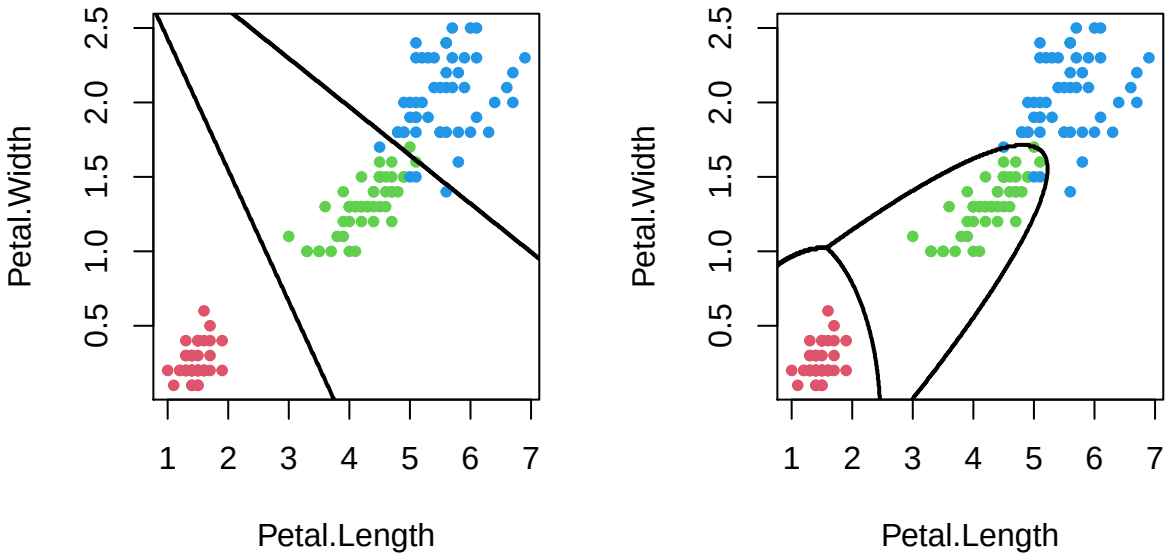
```
## save class labels
library(MASS)
iris.feature <- iris[,3:4]

iris.lda <- lda(iris.feature,grouping=ct)
##create a grid for our plotting surface
x <- seq(-1,10,0.01)
y <- seq(-1,7,0.01)
z <- as.matrix(expand.grid(x,y),0)
m <- length(x)
n <- length(y)
##classes are 1,2 and 3, so set contours at 1.5 and 2.5

par(mfrow=c(1, 2), pty="s")

plot(iris.feature,col=ct+1,pch=20,cex=1,cex.lab=1)
iris.ldp <- predict(iris.lda,z)$class
```

```
## Warning in predict.lda(iris.lda, z): variable names in 'newdata' do not match
## those in 'object'
```

```
contour(x,y,matrix(iris.ldp,m,n),
        levels=c(1.5,2.5), add=TRUE, d=FALSE, lty=1, lwd=2)


iris.qda <- qda(iris.feature,grouping=ct)
plot(iris.feature,col=ct+1,pch=20,cex=1,cex.lab=1)
iris.qdp <- predict(iris.qda,z)$class
```

```
## Warning in predict.qda(iris.qda, z): variable names in 'newdata' do not match
## those in 'object'
```

```
contour(x,y,matrix(iris.qdp,m,n),
        levels=c(1.5,2.5), add=TRUE, d=FALSE, lty=1, lwd=2)
```




```
ct.l<- table(predict(iris.lda, grouping=ct)$class, ct)
ct.l
```

```
##    ct
##      1  2  3
##   1 50  0  0
##   2  0 48  4
##   3  0  2 46
```

```
error.rate.1<-sum(sum(ct.l[row(ct.l)!=col(ct.l)]))/length(ct)
error.rate.1
```

```
## [1] 0.04
```

```
ct.q<- table(predict(iris.qda, grouping=ct)$class, ct)
ct.q
```

```
##    ct
##      1  2  3
##   1 50  0  0
##   2  0 49  2
##   3  0  1 48
```

```
error.rate.q<-sum(sum(ct.q[row(ct.q)!=col(ct.q)]))/length(ct)
error.rate.q
```

```
## [1] 0.02
```

6 On LDA computations

The decision boundaries for LDA are useful for graphical purposes, but to classify a new point $\mathbf{x}_0 \in \mathbb{R}^p$ we do not use them. To do this, we simply compute $\hat{\delta}_j(\mathbf{x}_0)$ for each $j = 1, \ldots, K$. LDA works very well in many practical applications even when compared with fancy alternative classification schemes.

As we showed earlier, LDA equivalently minimizes over $j = 1, \ldots, K$,

$$\frac{1}{2}(\mathbf{x} - \hat{\mu}_j)^\top \hat{\Sigma}^{-1}(\mathbf{x} - \hat{\mu}_i) - \log \hat{\pi}_j.$$

In practice, it often helps to factorize $\hat{\Sigma}$ using eigen-decomposition technique and rewrite it as

$$\hat{\Sigma} = UDU^\top,$$

where $U \in \mathbb{R}^{p \times p}$ has orthogonal columns (and rows) and $D = \text{diag}(d_1, \ldots, d_p)$ with $d_j \geq 0$ for each j. Note that under this decomposition,

$$D^{-1} = \text{diag}(1/d_1, \ldots, 1/d_p) \quad \text{and} \quad D^{-\frac{1}{2}} = \text{diag}(1/\sqrt{d_1}, \ldots, 1/\sqrt{d_p}).$$

We also have $\hat{\Sigma}^{-1} = UD^{-1}U^\top$, $D^{-\frac{1}{2}}D^{-\frac{1}{2}} = D^{-1}$, and

$$UD^{-1}U^\top UDU^\top = UD^{-1}DU^\top = UU^\top = \mathbb{I}.$$

So,

$$\begin{aligned}
\frac{1}{2}(\mathbf{x} - \hat{\mu}_j)^\top \hat{\Sigma}^{-1}(\mathbf{x} - \hat{\mu}_i) - \log \hat{\pi}_j &= \frac{1}{2}[U^\top(\mathbf{x} - \hat{\mu}_j)]^\top D^{-1}[U^\top(\mathbf{x} - \hat{\mu}_j)] - \log \hat{\pi}_j \\
&= \frac{1}{2}[D^{-\frac{1}{2}}U^\top(\mathbf{x} - \hat{\mu}_j)]^\top [D^{-\frac{1}{2}}U^\top(\mathbf{x} - \hat{\mu}_j)] - \log \hat{\pi}_j \\
&= \frac{1}{2}||D^{-\frac{1}{2}}U^\top \mathbf{x} - D^{-\frac{1}{2}}U^\top \hat{\mu}_j||_2^2 - \log \hat{\pi}_j \\
&= \frac{1}{2}||\tilde{\mathbf{x}} - \tilde{\mu}_j||^2 - \log \hat{\pi}_j,
\end{aligned}$$

which is essentially squared distance between $\tilde{\mathbf{x}}$ (transformed $\mathbf{x}$) and $\tilde{\mu}_j$ (transformed $\hat{\mu}_j$) adjusted by $\log \hat{\pi}_j$. This being said, LDA procedure can be described as

1. Compute the sample estimates $\hat{\pi}_j$, $\hat{\mu}_j$ and $\hat{\Sigma}$.
2. Factor $\hat{\Sigma}$ as $\hat{\Sigma} = UDU^\top$.
3. Transform the class centroids to $\tilde{\mu}_j = D^{-\frac{1}{2}}U^\top \hat{\mu}_j$.
4. Given any point $\mathbf{x} \in \mathbb{R}^p$, transform the point to $\tilde{\mathbf{x}} = D^{-\frac{1}{2}}U^\top \mathbf{x} \in \mathbb{R}^p$.
5. Classify according the nearest centroid in the transformed space after adjusting for class proportion, that is classify to class j for which
$$\frac{1}{2}||\tilde{\mathbf{x}} - \tilde{\mu}_j||^2 - \log \hat{\pi}_j$$
is the smallest.

Transformation $\mathbf{x}_i \to \tilde{\mathbf{x}}_i = D^{-\frac{1}{2}}U^\top \mathbf{x}_i$ is called sphering the data points. This is because if we think of $\mathbf{x} \in \mathbb{R}^p$ as a random variable with covariance matrix $\hat{\Sigma}$, then

$$Cov(\tilde{\mathbf{x}}_i) = Cov(D^{-\frac{1}{2}}U^\top \mathbf{x}_i) = D^{-\frac{1}{2}}U^\top \hat{\Sigma} U D^{-\frac{1}{2}} = \mathbb{I}.$$

Now, what LDA is doing is comparing $\frac{1}{2}||\tilde{\mathbf{x}} - \tilde{\mu}_j||^2 - \log \hat{\pi}_j$ across the classes $j = 1, \ldots, K$. Consider the affine subspace $M \subset \mathbb{R}^p$ spanned by the transformed centroids $\tilde{\mu}_1, \ldots, \tilde{\mu}_K$, which has dimension $K-1$. For any $\mathbf{x} \in \mathbb{R}^p$, we can decompose $\tilde{\mathbf{x}}$ as

$$\tilde{\mathbf{x}} = Project_M^{\tilde{\mathbf{x}}} + Project_{M^\perp}^{\tilde{\mathbf{x}}},$$

so,

$$\begin{aligned} ||\tilde{\mathbf{x}} - \tilde{\mu}_j||^2 &= ||Project_M^{\tilde{\mathbf{x}}} - \tilde{\mu}_j + Project_{M^\perp}^{\tilde{\mathbf{x}}}||^2 \\ &= ||Project_M^{\tilde{\mathbf{x}}} - \tilde{\mu}_j||^2 + ||Project_{M^\perp}^{\tilde{\mathbf{x}}}||^2. \end{aligned}$$

Note that the second term does not depend on j. In other words, the LDA classification rule does not change if we project the points to be classified onto M, since the distances orthogonal to M do not matter. So, LDA procedure can be seen as follow:

1. Compute the sample estimates $\hat{\pi}_j$, $\hat{\mu}_j$ and $\hat{\Sigma}$.
2. Sphere the data points based on factoring $\hat{\Sigma}$ using eigen-decomposition.
3. Project down data points to the affine subspace spanned by the sphere centroids. This can all be summarized by a single linear transformation $A \in \mathbb{R}^{(K-1)\times p}$.
4. Given any point $\mathbf{x} \in \mathbb{R}^p$, transform to $\tilde{\mathbf{x}} = A\mathbf{x} \in \mathbb{R}^{K-1}$ and classify according to the class j for which

   $$\frac{1}{2}||\tilde{\mathbf{x}} - \tilde{\mu}_j||^2 - \log \hat{\pi}_j$$

   is smallest, where $\tilde{\mu}_j = A\hat{\mu}_j$ with $A = D^{-\frac{1}{2}}U^\top$.

This way of describing LDA might look complicated, but actually it is much simpler. After applying A we reduce the problem from p dimension to $K-1$ dimensions, this is basically nearest centroid classification, with a simple modification to adjust for class proportions:

$$\hat{h}^{LDA}(\mathbf{x}) = \underset{j=1,\ldots,K}{\operatorname{argmin}} \left\{ \frac{1}{2}||\tilde{\mathbf{x}} - \tilde{\mu}_j||^2 - \log \hat{\pi}_j \right\}.$$

In `R` the matrix $A^\top$ is exactly what is returned by the `scaling` component from the `lda` function in the `MASS` package.

Example (Olive Oil Data Set) The data set that we analyze here is the data on the percentage composition of eight fatty acids found by lipid fraction of 572 Italian olive oils. The data come from three regions: Southern Italy, Sardinia, and Northern Italy. Within each region there are a number of different areas. Southern Italy comprises North Apulia, Calabria, South Apulia, and Sicily. Sardinia is divided into Inland Sardinia and Costal Sardinia. Northern Italy comprises Umbria, East Liguria, and West Liguria. The olive oil data set is a data frame with 572 observations and 10 columns. The first column gives the region: (1) Southern Italy, (2) Sardinia, or (3) Northern Italy. The second column gives the area: (1) North Apulia, (2) Calabria, (3) South Apulia, (4) Sicily, (5) Inland Sardinia, (6) Costal Sardinia, (7) East Liguria, (8) West Liguria, and (9) Umbria. The other $p = 8$ columns are observations on variables measuring the percentage composition of 8 different fatty acids.

Using the sphering approach, these data can be transformed to data points $A\mathbf{x}_i \in R^2$, $i = 1, \ldots, 572$. Here M is a 2-dimensional subspace, since there are $K = 3$ classes and the transformation has dimension $A \in \mathbb{R}^{2\times 8}$. Let us first look at the data set, which can be found in the package `classifly` in `R`.

```
library(classifly)
data(olives)
str(olives)
```

```
## 'data.frame':    572 obs. of  10 variables:
##  $ Region     : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Area       : Factor w/ 9 levels "Calabria","Coast-Sardinia",..: 5 5 5 5 5 5 5 5 5 5 ...
##  $ palmitic   : int  1075 1088 911 966 1051 911 922 1100 1082 1037 ...
##  $ palmitoleic: int  75 73 54 57 67 49 66 61 60 55 ...
##  $ stearic    : int  226 224 246 240 259 268 264 235 239 213 ...
##  $ oleic      : int  7823 7709 8113 7952 7771 7924 7990 7728 7745 7944 ...
##  $ linoleic   : int  672 781 549 619 672 678 618 734 709 633 ...
##  $ linolenic  : int  36 31 31 50 50 51 49 39 46 26 ...
##  $ arachidic  : int  60 61 63 78 80 70 56 64 83 52 ...
##  $ eicosenoic : int  29 29 29 35 46 44 29 35 33 30 ...
```

```
head(olives)
```

```
##   Region         Area palmitic palmitoleic stearic oleic linoleic linolenic
## 1      1 North-Apulia     1075          75     226  7823      672        36
## 2      1 North-Apulia     1088          73     224  7709      781        31
## 3      1 North-Apulia      911          54     246  8113      549        31
## 4      1 North-Apulia      966          57     240  7952      619        50
## 5      1 North-Apulia     1051          67     259  7771      672        50
## 6      1 North-Apulia      911          49     268  7924      678        51
##   arachidic eicosenoic
## 1        60         29
## 2        61         29
## 3        63         29
## 4        78         35
## 5        80         46
## 6        70         44
```

As we showed earlier, working in the transformed space makes it easier to draw boundaries. Now the decision boundary between class j and k is the set of say $\mathbf{z} \in \mathbb{R}^{K-1}$ such that

$$\frac{1}{2}||\mathbf{z} - \tilde{\mu}_j||^2 - \log \hat{\pi}_j = \frac{1}{2}||\mathbf{z} - \tilde{\mu}_k||^2 - \log \hat{\pi}_k.$$

By simple calculations, this can be simplified to $A^\top \mathbf{z} = B$ as follow

$$(\tilde{\mu}_j - \tilde{\mu}_i)^\top \mathbf{z} = \log \frac{\hat{\pi}_k}{\hat{\pi}_j} + \frac{1}{2}(||\tilde{\mu}_j||^2 - ||\tilde{\mu}_k||^2).$$

As an example, when $K = 3$, so that $\mathbf{z} \in \mathbb{R}^2$, this is just the line given by $z_2 = a + bz_1$, where

$$a = \frac{\log \frac{\hat{\pi}_k}{\hat{\pi}_j} + \frac{1}{2}(||\tilde{\mu}_j||^2 - ||\tilde{\mu}_k||^2)}{\tilde{\mu}_{j2} - \tilde{\mu}_{k2}},$$

and

$$b = \frac{\tilde{\mu}_{k1} - \tilde{\mu}_{j1}}{\tilde{\mu}_{j2} - \tilde{\mu}_{k2}}.$$

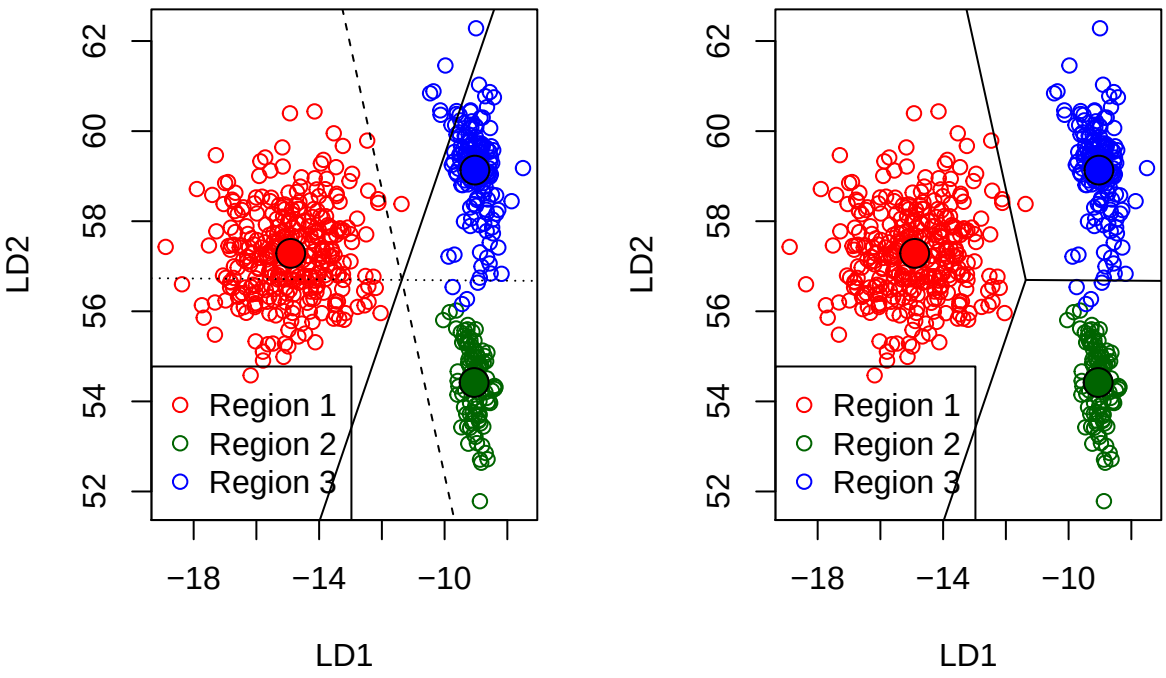
Using this approach in our Olive Oil data set we can reduce the problem to only looking at 2, rather than 8 dimensions. Plus, now the decision boundaries are pretty easy to draw, because it's essentially nearest centroid classification as described earlier.

```r
par(mfrow=c(1, 2))
y = as.numeric(olives[,1])
x = as.matrix(olives[,3:10])

n = nrow(x)
p = ncol(x)

pi = numeric(3)
mu = matrix(0,3,8)

for (j in 1:3) {
  pi[j] = sum(y==j)/n
  mu[j,] = colMeans(x[y==j,])
}

Sigma = matrix(0,p,p)
for (j in 1:3) {
  A = scale(x[y==j,],center=T,scale=F)
  Sigma = Sigma + t(A)%*%A
}
Sigma = Sigma/(n-3)
Sigmainv = solve(Sigma)

lda.classify = function(x0, pi, mu, Sigmainv) {
  K = length(pi)
  delta = numeric(K)
  for (j in 1:K) {
    delta[j] = t(x0) %*% Sigmainv %*% mu[j,] -
      0.5 * t(mu[j,]) %*% Sigmainv %*% mu[j,] + log(pi[j])
  }
  return(which.max(delta))
}

yhat = numeric(n)
for (i in 1:n) {
  yhat[i] = lda.classify(x[i,],pi,mu,Sigmainv)
}

# Training errors
sum(yhat!=y)
```

```
## [1] 5
```

```r
a = lda(x,y)

at = a$scaling # In terms of lecture notation, this is A^T
z = x %*% at

cols = c("red","darkgreen","blue")
plot(z,col=cols[y])
legend("bottomleft",pch=21,col=cols,
```

```
       legend=c("Region 1","Region 2","Region 3"))


mu = a$means %*% at
pi = a$prior

points(mu,col=cols,pch=19,cex=2)
points(mu,pch=21,cex=2)

getab = function(j,k,mu,pi) {
  b = (mu[k,1]-mu[j,1])/(mu[j,2]-mu[k,2])
  normj = sum(mu[j,]^2)
  normk = sum(mu[k,]^2)
  a = (log(pi[k]/pi[j]) + 0.5*(normj-normk))/(mu[j,2]-mu[k,2])
  return(list(a=a,b=b))
}

# 1 and 2
ab12 = getab(1,2,mu,pi)
abline(a=ab12$a,b=ab12$b,lty=1)

# 1 and 3
ab13 = getab(1,3,mu,pi)
abline(a=ab13$a,b=ab13$b,lty=2)

# 2 and 3
ab23 = getab(2,3,mu,pi)
abline(a=ab23$a,b=ab23$b,lty=3)

## Find points of intersection
zx = (ab12$a-ab13$a)/(ab13$b-ab12$b)
zy = ab12$a + ab12$b*zx

# Now redraw with the appropriate boundaries
plot(z,col=cols[y])
legend("bottomleft",pch=21,col=cols,
       legend=c("Region 1","Region 2","Region 3"))
points(mu,col=cols,pch=19,cex=2)
points(mu,pch=21,cex=2)

segments(zx,zy,-15,ab12$a+ab12$b*(-15))
segments(zx,zy,-7,ab23$a+ab23$b*(-7))
segments(zx,zy,-15,ab13$a+ab13$b*(-15))
```



7 Reduced-rank linear discriminant analysis

The dimension reduction from p to $K-1$ that we studied in the last section was exact, in that we did not change the LDA rule at all. In practice K might still be large and visualizing decision boundaries might not be possible. In such situations one might want to reduce further to a dimension say $L < K-1$, when K is large. Reduced-rank linear discriminant analysis (RRLDA) is a nice way to project down to lower than $K-1$ dimensions. It chooses a lower dimensional subspace so as to spread out the centroids as much as possible. The approach is similar to the principal component analysis. The reduced dimensions in RRLDA approach are computed by looking at the principal components directions of the matrix of transformed centroids.

Example (Crab data):
The crabs data frame has 200 rows and 8 columns, describing 5 morphological measurements on 50 crabs each of two color forms and both sexes, of the species Leptograpsus variegatus collected at Fremantle, W. Australia. We first create 4 classes of crabs using Speices (B=Blue, O=Orange) and Sex (F=Female, M=Male) to get classes BM, OM, BF, OF. Then we perform LDA and QDA on our data set for classifications.

```
library(MASS)
data(crabs)
head(crabs)
```

```
##   sp sex index   FL  RW   CL   CW  BD
## 1  B   M     1  8.1 6.7 16.1 19.0 7.0
## 2  B   M     2  8.8 7.7 18.1 20.8 7.4
## 3  B   M     3  9.2 7.8 19.0 22.4 7.7
## 4  B   M     4  9.6 7.9 20.1 23.1 8.2
## 5  B   M     5  9.8 8.0 20.3 23.0 8.2
## 6  B   M     6 10.8 9.0 23.0 26.5 9.8
```

```
par(mfrow=c(1, 2))

zz<-as.factor(paste(crabs[,1], crabs[,2], sep=""))
```

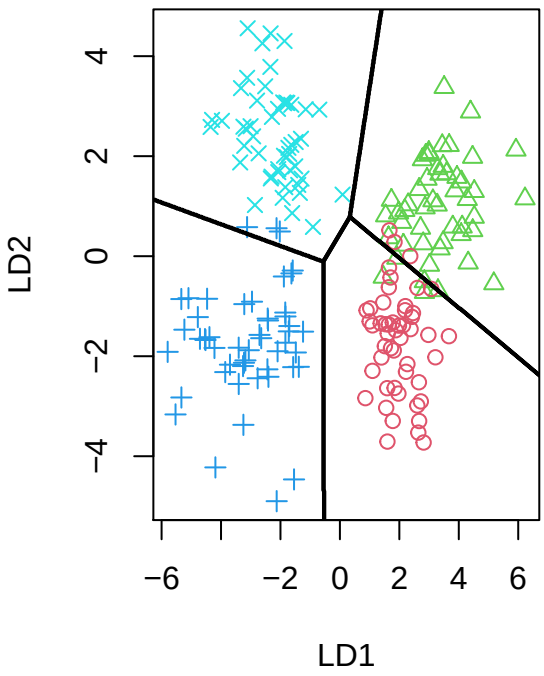
```
ct<-as.numeric(zz)

crab.lda<-lda(crabs[, 4:8], grouping=ct)
crab.ldp<-predict(crab.lda)
plot(crab.ldp$x, col=ct+1, pch=ct)

x <- seq(-10,10,0.02)
y <- seq(-10,10,0.02)
z <- as.matrix(expand.grid(x,y,0))
m <- length(x)
n <- length(y)

cb.ldap <- lda(crab.ldp$x,ct)
cb.ldpp <- predict(cb.ldap,z)$class
```

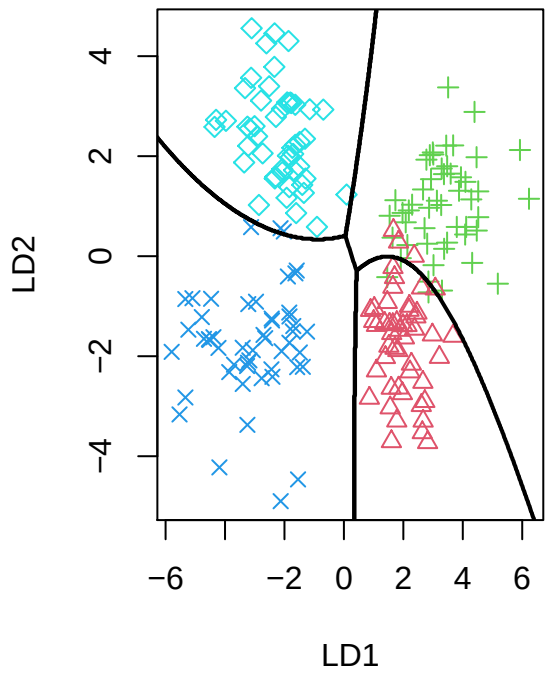
```
## Warning in predict.lda(cb.ldap, z): variable names in 'newdata' do not match
## those in 'object'
```

```
contour(x,y,matrix(cb.ldpp,m,n), levels=c(1.5,2.5, 3.5),
add=TRUE,d=FALSE,lty=1,lwd=2)



crab.qda<-qda(crabs[, 4:8], grouping=ct)
crab.qdp<-predict(crab.qda)
plot(crab.ldp$x, col=ct+1, pch=ct+1)

cb.qdap <- qda(crab.ldp$x,ct)
cb.qdpp <- predict(cb.qdap,z)$class
```

```
## Warning in predict.qda(cb.qdap, z): variable names in 'newdata' do not match
## those in 'object'
```

```
contour(x,y,matrix(cb.qdpp,m,n), levels=c(1.5, 2.5,3.5),
add=TRUE,d=FALSE,lty=1,lwd=2)
```






```
ct.l<- table(predict(crab.lda, grouping=ct)$class, ct)
ct.l
```

```
##    ct
##      1  2  3  4
##   1 50  5  0  0
##   2  0 45  0  0
##   3  0  0 47  0
##   4  0  0  3 50
```

```
error.rate.1<-sum(sum(ct.l[row(ct.l)!=col(ct.l)]))/length(ct)
error.rate.1
```

```
## [1] 0.04
```

```
ct.q<- table(predict(crab.qda, grouping=ct)$class, ct)
ct.q
```

```
##    ct
##      1  2  3  4
##   1 48  4  0  0
##   2  2 46  0  0
##   3  0  0 48  0
##   4  0  0  2 50
```

```
error.rate.q<-sum(sum(ct.q[row(ct.q)!=col(ct.q)]))/length(ct)
error.rate.q
```

```
## [1] 0.04
```

8 Fisher's Linear Discriminent Analysis (two class problem)

The idea of Fisher LDA is to first reduce the dimension of the feature space (covariates) to only one dimension by projecting the data onto a line such that along this new line classes are maximally separated from each other. In other words, suppose we have a binary classification problem where we have access to p features $\mathbf{X} = (X_1, \ldots, X_p)$ and the response is coded as $Y \in \{0, 1\}$. In PCA the idea was to search for directions in the data that have largest variance and subsequently project the data onto it. In Fisher's approach the class labels are also used to perform some sort of supervised dimension reduction.

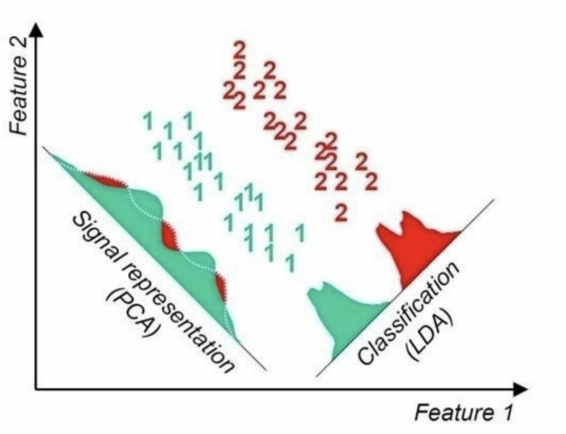


Figure 6: PCA versus Fisher's dimension reduction method

In Fisher's method, we are looking for a linear combination

$$U = \mathbf{w}^\top \mathbf{X} = \sum_{j=1}^{p} w_j X_j$$

with the best choice of $\mathbf{w}^\top = (w_1, \ldots, w_p)$ such that U best separate the data. There are many linear combinations of X that can be constructed. However, in classification, Fisher looked for a linear combination that maximizes the separability of projected data points. The following figure considers a binary classification problem and shows two different directions that one might want to project the training data points that are sitting in a two dimensional space, i.e.,

$$Data = \{(\mathbf{x}_i, y_i),\ i = 1, 2, \ldots, n\} \quad \text{with} \quad \mathbf{x}_i = (x_{1i}, x_{2i})$$

to form a new set of data points

$$Data^* = \{(u_i, y_i), i = 1, \ldots, n\},$$

such that the separation between data points

$$Data_1^* = \{(u_i, y_i = 1), i = 1, \ldots, n_1\}$$

and

$$Data_0^* = \{(u_i, y_i = 0), i = 1, \ldots, n_0\},$$

is maximized. See the following figure for a visual explanation of the idea of Fisher's LDA.

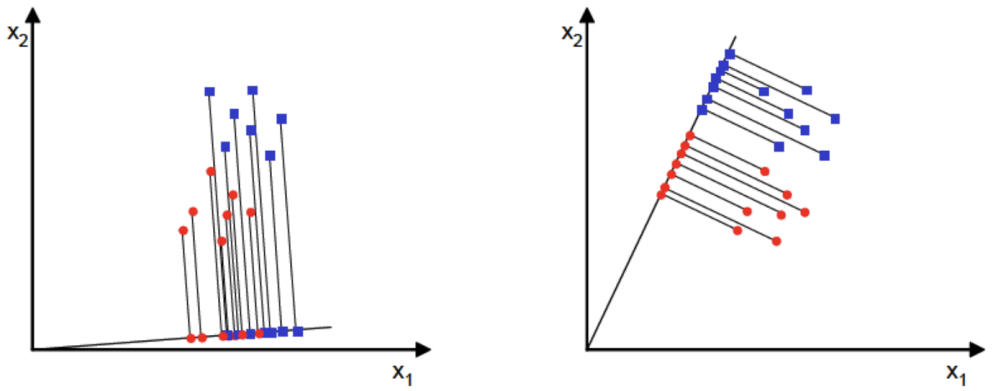


Figure 7: The Idea Behind Fisher's Linear Dimension Reduction Technique for Classification

After we obtain such a one dimensional data points, then we perform classification with the one dimensional covariate U instead of $\mathbf{X}$. For example, for a Binary classification, Fisher's classification rule is going to be

$$h_F(\mathbf{x}) = \begin{cases} 1 & \mathbf{w}^\top \mathbf{x} \geq \theta \\ 0 & \text{Otherwise} \end{cases}, \quad \text{where } \theta \text{ is a chosen threshold.}$$

In order to achieve the above described goal, one needs to define what we mean by separation of the groups and how to find a good projection vector to give a good separation. Fisher defined a good separation the one that makes groups (classes) to have means that are far apart relative to their spread. Let

$$\mathbb{E}[U|Y=j] = \mathbb{E}[\mathbf{w}^\top \mathbf{X}|Y=j] = \mathbf{w}^\top \mu_j,$$

where

$$\boldsymbol{\mu}_j = \mathbb{E}[\mathbf{X}|Y=j].$$

Also, suppose $\mathbb{V}ar(\mathbf{X}) = \Sigma$, and observe that

$$\mathbb{V}ar(U) = \mathbb{V}ar(\mathbf{w}^\top \mathbf{X}) = \mathbf{w}^\top \Sigma \mathbf{w}.$$

Consider a binary classification with $y \in \{0, 1\}$. Fisher defined his LDA as a linear function $\mathbf{w}^\top \mathbf{X}$ that maximizes the following separation measure

$$\begin{aligned} \mathcal{J}(\mathbf{w}) &= \frac{(\mathbb{E}[U|Y=0] - \mathbb{E}[U|Y=1])^2}{\mathbb{V}ar(Y)} \\ &= \frac{(\mathbf{w}^\top \mu_0 - \mathbf{w}^\top \mu_1)^2}{\mathbf{w}^\top \Sigma \mathbf{w}} \\ &= \frac{\mathbf{w}^\top (\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)^\top \mathbf{w}}{\mathbf{w}^\top \Sigma \mathbf{w}} \\ &= \frac{(\mathbf{w}\boldsymbol{\delta})^2}{\mathbf{w}^\top \Sigma \mathbf{w}}, \quad \text{with} \quad \boldsymbol{\delta} = (\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1). \end{aligned}$$

Therefore, he was looking for a projection direction where examples from the same class are projected very close to each other and at the same time the projected means are as farther apart as possible.

Note that the quantity $\mathcal{J}(\mathbf{w})$arises in physics and is often called *Rayleigh* coefficient. In $\mathcal{J}(\mathbf{w})$ we have two quantities that play important roles in constructing Fisher's LDA. The first quantity is

$$\mathbf{w}^\top (\boldsymbol{\mu}_o - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)^\top \mathbf{w},$$

which measures the between class scatter. We also have $\mathbf{w}^\top \Sigma \mathbf{w}$ which is nothing more than within-class scatter. In general these quantities are unknown and should be estimated using our training data. To this end, let

$$n_j = \sum_{i=1}^{n} \mathbb{I}(y_i = j) = \text{Number of data points from class } j \text{ in our trating data.}$$

Let $\bar{\mathbf{X}}_j$ be the sample mean vector of $\mathbf{X}$'s for group j, and S_j be the sample covariance matrix in group j. Define

$$\widehat{\mathcal{J}}(\mathbf{w}) = \frac{\mathbf{w}^\top S_B \mathbf{w}}{\mathbf{w}^\top S_W \mathbf{w}},$$

where

$$S_B = (\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1)(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1)^\top,$$

contains the squares and cross products of the component differences between the means of the observations in two different classes, and

$$S_W = \frac{n_0 - 1}{n_0 + n_1 - 2} S_0 + \frac{n_1 - 1}{n_0 + n_1 - 2} S_1,$$

where we need $(n_0 + n_1 - 2) > p$ or otherwise S_W is singular and the usual inverse does not exists. An important property to notice about the objective function $\widehat{\mathcal{J}}(\mathbf{w})$ is that it is invariant with respect to rescaling of $\mathbf{w}$ to $\alpha\mathbf{w}$. Hence, as we are only interested in the direction to project the data, length is not important and we can always choose $\mathbf{w}$ such that the denominator is simply $\mathbf{w}^\top S_W \mathbf{w} = 1$ (note that this is a scaler itself). So, we can transform the problem of maximizing $\widehat{\mathcal{J}}(\mathbf{w})$ into the following constrained optimization:

$$\text{Maximize } \widehat{\mathcal{J}}(\mathbf{w}) = \frac{\mathbf{w}^\top S_B \mathbf{w}}{\mathbf{w}^\top S_W \mathbf{w}} \quad \text{subject to} \quad \mathbf{w}^\top S_W \mathbf{w} = 1,$$

which is equivalent to

$$\text{Maximize } \mathbf{w}^\top S_B \mathbf{w} \quad \text{subject to} \quad \mathbf{w}^\top S_W \mathbf{w} = 1.$$

This can be easily solved using the Lagrangian multiplier method by solving

$$\mathcal{L}(\mathbf{w}, \lambda) = \mathbf{w}^\top S_B \mathbf{w} + \lambda(\mathbf{w}^\top S_W \mathbf{w} - 1).$$

Solving this results in

$$S_B \mathbf{w} = \lambda S_w \mathbf{w}.$$

Note that, give that S_W is positive definite and by multiplying both sides with S_W^{-1} we have

$$S_W^{-1} S_B \mathbf{w} = \lambda \mathbf{w}.$$

In other words, it looks like and eigenvalue equation if $S_W^{-1} S_B$ was symmetric. This is indeed a generalized eigenvalue problem that you can solve using any eigenvalue routine. As we showed in our notes for PCA, The vector

$$\mathbf{w}^* = S_W^{-1}(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1)$$

gives a solution to this problem and the vector

$$U = \mathbf{w}^\top \mathbf{x} = (\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1)^\top S_W^{-1} \mathbf{x},$$

is the resulting Fisher linear discriminant function.

Remark 3. *To better understand this, one can apply the following transformation using the fact that S_B is symmetric positive definite and can be written as $S_B^{\frac{1}{2}} S_B^{\frac{1}{2}}$. This can be done using the eigenvalue decomposition of $S_B = P\Lambda P^\top$ with $PP^\top = P^\top P = \mathbb{I}$. So, we have*

$$S_W^{-1} S_B \mathbf{w} = \lambda \mathbf{w} \quad \Longrightarrow \quad S_B^{\frac{1}{2}} S_W^{-1} S_B^{\frac{1}{2}} (S_B^{\frac{1}{2}} \mathbf{w}) = \lambda (S_B^{\frac{1}{2}} \mathbf{w}).$$

Let $\mathbf{v} = S_B^{\frac{1}{2}} \mathbf{w}$ then we get

$$S_B^{\frac{1}{2}} S_W^{-1} S_B^{\frac{1}{2}} \mathbf{v} = \lambda \mathbf{v}.$$

Now, this is a regular eigenvalue problem as $S_B^{\frac{1}{2}} S_W^{-1} S_B^{\frac{1}{2}}$ is a symmetric, positive definite matrix. As we showed earlier in our PCA note, one can find $\mathbf{v}^*$ as the solution to the above equation using the eigenvalues and corresponding eigenvectors of $S_B^{\frac{1}{2}} S_W^{-1} S_B^{\frac{1}{2}}$. Since we would like to maximize the objective function, we need to simply take $\mathbf{v}^*$ to be the eigenvector associated with the largest eigenvalue of $S_B^{\frac{1}{2}} S_W^{-1} S_B^{\frac{1}{2}}$. Using this, we get

$$\mathbf{w}^* = S_B^{-\frac{1}{2}} \mathbf{v}^*,$$

and accordingly

$$U = \mathbf{v}^{*\top} S_B^{-1} \boldsymbol{x}.$$

Given a training data set $\{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$ we can easily find the Fisher's sample linear discriminant function as

$$U^* = (\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1)^\top S_W^{-1} \mathbf{x}.$$

we can employ it as a classification device. Let

$$\bar{U}_0^* = \mathbf{w}^* \bar{\mathbf{x}}_0 \quad \text{and} \quad \bar{U}_1^* = \mathbf{w}^* \bar{\mathbf{x}}_1$$

and define m be the midpoint between the projected means of the features for two classes in our data set as

$$m = \frac{1}{2}(\bar{U}_0^* + \bar{U}_1^*) = \frac{1}{2}(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1)^\top S_W^{-1} (\bar{\mathbf{x}}_0 + \bar{\mathbf{x}}_1).$$

Now, the classification rule based on Fisher's method is defined as

$$h_F(\mathbf{x}_j) = \begin{cases} 1 & U_j^* > m \\ 0 & \text{otherwise} \end{cases}.$$

The maximum vale of $\widehat{\mathcal{J}}(\mathbf{w})$ is given by

$$\max_{\mathbf{w}} \frac{\mathbf{w}^\top S_B \mathbf{w}}{\mathbf{w}^\top S_W \mathbf{w}} = (\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1)^\top S_W^{-1} (\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1) = D^2,$$

where D^2 is the sample squared distance. For two populations the maximum relative separation that can be obtained by considering linear combinations of the features is equal to the distance D. Now, D^2 can be used to test whether the population class means $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_0$ differ significantly or not and so a test for equality of the population class means can be considered as a test for significance of the separation that can be achieved.

Suppose that the features $\mathbf{X}^\top = (X_1, \ldots, X_p)$ in the populations associated with $Y = 0$ and Y_1 are distributed according to multivariate normal distributions with common covariance matrix Σ. Then, a test for

$$H_0 : \boldsymbol{\mu}_0 = \boldsymbol{\mu}_1 \quad \text{versus} \quad H_1 = \boldsymbol{\mu}_0 \neq \boldsymbol{\mu}_1$$

can be conducted using the following test statistic

$$\left(\frac{n_0+n_1-p-1}{(n_0+n_1-2)p}\right)\left(\frac{n_0 n_1}{n_0+n_1}\right)D^2 \sim F_{p,n_0+n_1-p-1}.$$

If H_0 is rejected we conclude that the separation between the two classes is significant. Note that this does not necessarily mean that classification is good. The efficacy of a classification procedure can be evaluated independently of any test of separation. If separation is not significant, one should search for other useful classification methods.

9 Logistic Regression For Classificastion

Consider the case where we have two classes $\mathcal{Y} = \{1, 2\}$ and let C denote a variable indicating class identity. Suppose

$$\mathbb{P}(\mathbf{X} = \mathbf{x}|C = j) = N_p(\mu_j, \Sigma), \quad j = 1, 2.$$

As we saw, using the Bayes rule

$$\mathbb{P}(C = j|\mathbf{X} = \mathbf{x}) = \frac{\mathbb{P}(\mathbf{X} = \mathbf{x}|C = j)\mathbb{P}(C = j)}{\mathbb{P}(\mathbf{X} = \mathbf{x})}.$$

Plugging in the normal densitis we showed that we classify observations to class 1 if

$$\mathbb{P}(C = 1|\mathbf{X} = \mathbf{x}) > \mathbb{P}(C = 2|\mathbf{X} = \mathbf{x}),$$

or equivalenetly

$$\log\left(\frac{\mathbb{P}(C = 1|\mathbf{X} = \mathbf{x})}{\mathbb{P}(C = 2|\mathbf{X} = \mathbf{x})}\right) > 0.$$

After, simple claclualtions we can show that

$$\begin{aligned}
\log\left(\frac{\mathbb{P}(C = 1|\mathbf{X} = \mathbf{x})}{\mathbb{P}(C = 2|\mathbf{X} = \mathbf{x})}\right) &= \log\left(\frac{\pi_1}{\pi_2}\frac{\frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}}\exp\{-\frac{1}{2}(\mathbf{x}-\mu_1)^\top\Sigma^{-1}(\mathbf{x}-\mu_1)\}}{\frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}}\exp\{-\frac{1}{2}(\mathbf{x}-\mu_2)^\top\Sigma^{-1}(\mathbf{x}-\mu_2)\}}\right) \\
&= \log(\frac{\pi_1}{\pi_2}) - \frac{\mu_1^\top\Sigma^{-1}\mu_1}{2} + \frac{\mu_2^\top\Sigma^{-1}\mu_2}{2} + (\mu_1-\mu_2)^\top\Sigma^{-2}\mathbf{x} \\
&= \beta_0 + \beta_1^\top\mathbf{x}.
\end{aligned}$$

In other words, the log odds of class 1 versus 2 is a linear function of $\mathbf{x}$. Estimating μ_1, μ_2, π_1, π_1 and Σ amounts to estimating β_0 and β_1. One might ask, why don't we directly estimate these coefficients? Do we really need to assume normality for $\mathbf{X}$ associated with each class?

In general, there are two big branches of methods for classification. One is called generative modeling, the other is called discriminative modeling. A generative model is a model for generating all values for a phenomenon, both those that can be observed in the world and target variables that can only be computed from those observed. A generative algorithm models how the data was generated in order to categorize a signal. Examples of genarative models are LDA, QDA, Naive Bayes, and they often require strong modeling assumptions. These methods can be used for multi-purpose tasks.

Generative models are used in machine learning for either modeling data directly (i.e., modeling observations drawn from a probability density function), or as an intermediate step to forming a conditional probability density function. Generative models are typically probabilistic, specifying a joint probability distribution over observation and target (label) values. A conditional distribution can be formed from a generative model through Bayes' rule. One of the advantages of generative algorithms is that you can use $\mathbb{P}(x, y)$ to generate new data similar to existing data.

By contrast, discriminative models provide a model only for the target variable(s), generating them by analyzing the observed variables. In simple terms, discriminative models infer outputs based on inputs, while generative models generate both inputs and outputs, typically given some hidden parameters. Discriminative algorithms generally give better performance in classification tasks. These models learn to perform better on the given task and require weaker modeling assumptions. These models can easily overfit our data though!

Logistic regression for classification is a discriminative modeling approach, where we estimate the posterior probabilities of classes given $\mathbf{X}$ directly without assuming the marginal distribution on $\mathbf{X}$.

In logistic regression, we assume that

$$\log\left(\frac{\mathbb{P}(C=1|\mathbf{X}=\mathbf{x})}{\mathbb{P}(C=2|\mathbf{X}=\mathbf{x})}\right)=\beta_0+\boldsymbol{\beta}^\top\mathbf{x},$$

for some unknown $\beta_0\in\mathbb{R}$ and $\beta\in\mathbb{R}^p$ which we will estimate directly.

Under such assumption and since $\mathbb{P}(C=1|\mathbf{X}=\mathbf{x})=1-\mathbb{P}(C=2|\mathbf{X}=\mathbf{x})$ one can easily see that

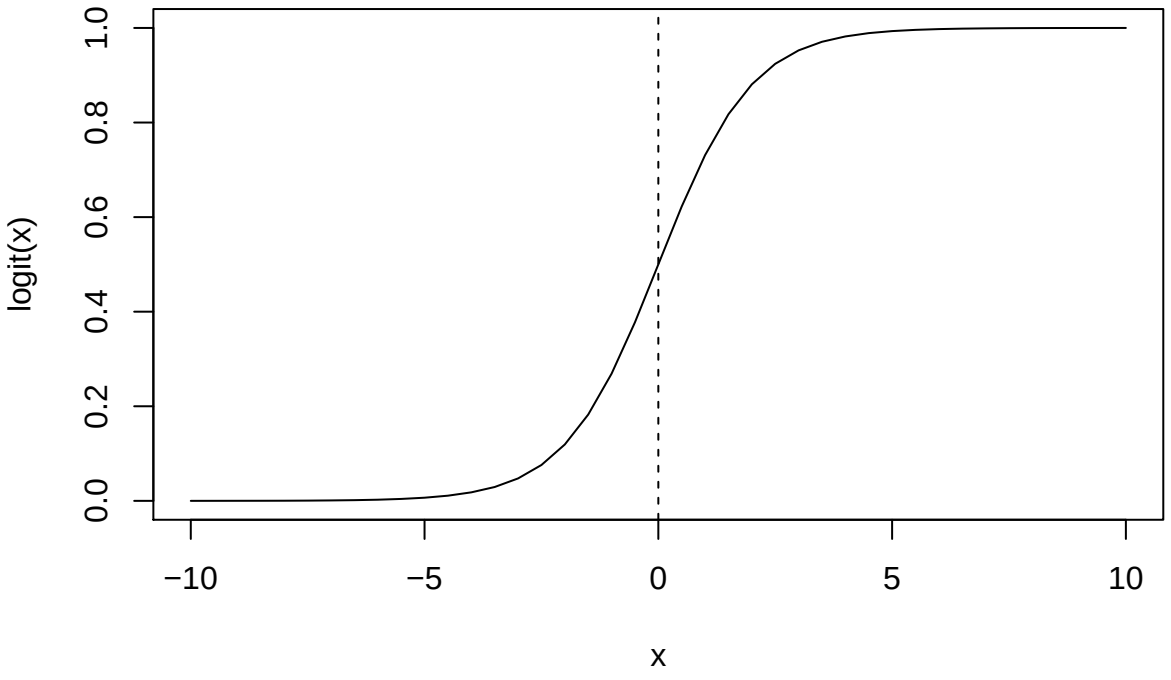
$$\log\left(\frac{\mathbb{P}(C=1|\mathbf{X}=\mathbf{x})}{1-\mathbb{P}(C=1|\mathbf{X}=\mathbf{x})}\right)=\beta_0+\beta^\top\mathbf{x}\iff\frac{\mathbb{P}(C=1|\mathbf{X}=\mathbf{x})}{1-\mathbb{P}(C=1|\mathbf{X}=\mathbf{x})}=\exp\{\beta_0+\beta^\top\mathbf{x}\}$$
$$\iff\mathbb{P}(C=1|\mathbf{X}=\mathbf{x})=\frac{\exp\{\beta_0+\beta^\top\mathbf{x}\}}{1+\exp\{\beta_0+\beta^\top\mathbf{x}\}}.$$

Therefor we only need to assume that

$$\mathbb{P}(C=1|\mathbf{X}=\mathbf{x})=s(\beta_0+\beta^\top\mathbf{x}),$$

with $s(u)=\frac{e^u}{1+e^u}$, $s:\mathbb{R}^p\longrightarrow[0,1$ being the sigmond or logistic function.

```
logit <- function(x) {
  return(1/(1+exp(-x)))
}
x <- seq(-10,10,0.5)
plot(x,logit(x), type="l")
abline(v=0,lty=2)
```



Remark One can replace the logit function with other monotone functions the are bounded in $[0, 1]$. Common example is using normal CDF or essentially the CDF of any continuous random variable for $s(\cdot)$.

The logistic function, or sigmoid function, has a real domain and a [0,1] range, which is appropriate to represent probability at some adequate contexts.

Binary Logistic Regression is a special type of regression where binary response variable is related to a set of explanatory variables, which can be discrete and/or continuous. The important point here to note is that in linear regression, the expected values of the response variable are modeled based on combination of values taken by the predictors. In logistic regression Probability or Odds of the response taking a particular value is modeled based on combination of values taken by the predictors.

Logistic regression is applicable, for example, if:

- we want to model the probabilities of a response variable as a function of some explanatory variables.
- we want to perform descriptive discriminate analyses such as describing the differences between individuals in separate groups as a function of explanatory variables.
- we want to predict probabilities that individuals fall into two categories of the binary response as a function of some explanatory variables.
- we want to classify individuals into two categories based on explanatory variables.

Suppose we are given a sample $(\mathbf{x}_i, C_i)$, $i = 1, \ldots, n$. Assume that class labels are conditionally independent given $\mathbf{x}_1, \ldots, \mathbf{x}_n$. For convenience, let

$$Y_i = \begin{cases} 1, & C_i = 1 \\ 0, & C_i = 2. \end{cases}$$

Let $\theta = (\beta_0, \beta_1)$ and suppose $p(\mathbf{x}; \theta) = \text{logit}(\mathbb{E}_\theta(Y|\mathbf{x})) = \text{logit}(\beta_0 + \beta^\top \mathbf{x}) = \frac{e^{\beta_0 + \beta^\top \mathbf{x}}}{1+e^{\beta_0 + \beta^\top \mathbf{x}}}$. Under this setting

$$Y_i|\mathbf{x}_i \sim \text{Bin}(1, p(\mathbf{x}_i; \theta)) = \text{Bin}(1, \frac{e^{\beta_0 + \beta^\top \mathbf{x}_i}}{1 + e^{\beta_0 + \beta^\top \mathbf{x}_i}}).$$

Now, the likelihood function of the sample is

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{n} p(\mathbf{x}_i; \theta)^{y_i} (1 - p(\mathbf{x}_i; \theta))^{1-y_i} \\ &= \prod_{i=1}^{n} \left(\frac{p(\mathbf{x}_i; \theta)}{1 - p(\mathbf{x}_i; \theta)} \right)^{y_i} (1 - p(\mathbf{x}_i; \theta)) \end{aligned}$$

As $\frac{p(\mathbf{x}_i;\theta)}{1-p(\mathbf{x}_i;\theta)} = e^{\beta_0 + \beta^\top \mathbf{x}_i}$, the log-likelihood function is given by

$$l(\beta_0, \beta_1) = \sum_{i=1}^{n} \left\{ y_i(\beta_0 + \beta^\top \mathbf{x}_i) - \log(1 + e^{\beta_0 + \beta^\top \mathbf{x}_i}) \right\}.$$

By taking the first derivatives with respect to β_0 and β_1 we get

$$\begin{cases} \frac{\partial l(\beta_0, \beta_1)}{\partial \beta_0} = \sum_{i=1}^{n} (y_i - p(x_i; \theta)) = 0 \\ \frac{\partial l(\beta_0, \beta_1)}{\partial \beta_1} = \sum_{i=1}^{n} x_i (y_i - p(x_i; \theta)) = 0 \end{cases}$$

The first equation specifies that $\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} p(\mathbf{x}_i; \theta)$, that is the expected number of class ones matches the observed number.

Note that one can simply write the score equations in the matrix notation as $\mathbf{X}^\top(\mathbf{Y} - \mathbf{P}) = 0$, where $\mathbf{X}_{n\times 2}$ is similar to the design matrix in the linear regression where first column in only ones, $\mathbf{Y}_{n\times 1}$ is the vector of

response variables and $\mathbf{P}_{n\times 1}$ is the vector of fitted probabilities with the ith element being $p(x_i;\theta)$. To solve these equations we use the Newton-Raphson algorithm, which requires the second derivative of Hessian matrix given by

$$\frac{\partial^2 l(\theta)}{\partial\theta\partial\theta^\top} = -\mathbf{X}^\top\mathbf{W}\mathbf{X},$$

where $\mathbf{W}_{N\times N}$ is a diagonal matrix with ith element $p(x_i;\theta)(1-p(x_i;\theta))$. Starting with θ^{old}, a single Newton update is

$$\begin{aligned}\theta^{new} &= \theta^{old} - \left(\frac{\partial^2 l(\theta)}{\partial\theta\partial\theta^\top}\right)^{-1}_{|\theta=\theta^{old}} \frac{\partial l(\theta)}{\partial\theta}|_{\theta=\theta^{old}} &(2)\\ &= \theta^{old} + (\mathbf{X}^\top\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top(\mathbf{Y}-\mathbf{P}) &(3)\\ &= (\mathbf{X}^\top\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{W}(\mathbf{X}^\top\theta^{old} + \mathbf{W}^{-1}(\mathbf{Y}-\mathbf{P})). &(4)\end{aligned}$$

Note that if we assume $\mathbf{Z}^{old} = \mathbf{X}^\top\theta^{old} + \mathbf{W^{old}}^{-1}(\mathbf{Y}-\mathbf{P})$, then one can easily write

$$\theta^{new} = (\mathbf{X}^\top\mathbf{W}^{old}\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{W}^{old}\mathbf{Z}^{old},$$

which is similar to the parameter estimates that one will get using the weighted least square methods. In the literature $\mathbf{Z}^{old}$ is called the adjusted response. These equations get solved repeatedly, since at each iteration $\mathbf{P}$ changes by updating θ. This algorithm is refereed to as *iteratively least sqaures** or IRLS since each iteration solves the weighted least squares problem

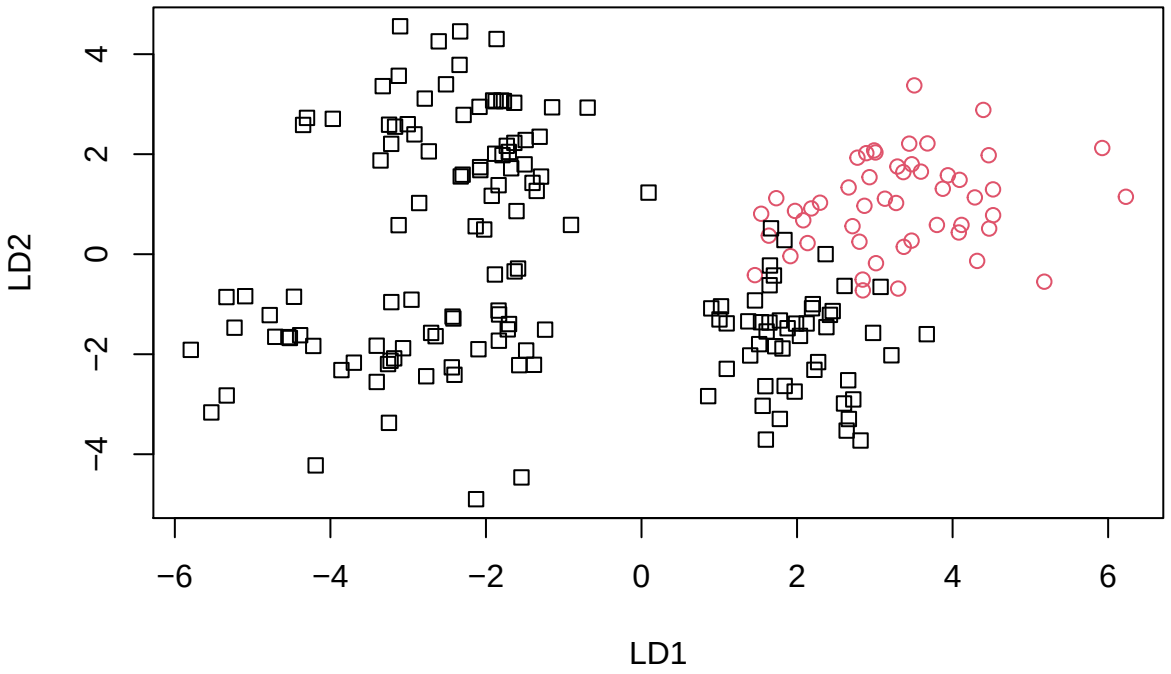
$$\theta^{new} \longleftarrow \text{argmin}_\theta(\mathbf{Z}^{old}-\mathbf{X}\theta)^\top\mathbf{W}^{old}(\mathbf{Z}^{old}-\mathbf{X}\theta).$$

One can easily use $\theta^{(0)} = 0$ as the initial value to star this numerical algorithm. In `R`, `glm()` function fits *generalized linear models*, a general class of models that includes logistic regression if we pass `family=binomial` as one of the parameters of the function. Here is how it works:

```
logistic.fit<- glm(Y~X, data=data.set, family=binomial)
summary(logistic.fit)
coef(logistic.fit)
logistic.probs<- predict(logistic.fit, type="response")
logistic.probs
```

Example (Crab data):
Consider the crab data set again. Suppose we are interested to predict if a crab belongs to class BM, or not, that is, crab is a member of the second class defined by $\{OM, BF, OF\}$. To be able to visualize the result, we use principal component (or reduced-rank LDA) to work with LD1 and LD2.

```
library(MASS)
data(crabs)

zz<-as.factor(paste(crabs[,1], crabs[,2], sep=""))

ct<-as.numeric(zz)

y<-as.numeric(ct==2)

crab.lda<-lda(crabs[, 4:8], grouping=ct)
crab.ldp<-predict(crab.lda)
plot(crab.ldp$x[,1:2], col=y+1, pch=y)
```



Now, we can implement the logistic regression. However, to visualize the decision boundaries we first create a grid of points and then use the logistic regression to predict class for each grid point and then use contour to visualize the boundaries. We use both logistic regression that is linear and quadratic

```
par(mfrow=c(1, 2))

crab.new<-data.frame(Y=y, LD1= crab.ldp$x[,1], LD2= crab.ldp$x[,2])

logistic.fit<- glm(Y~., data=crab.new,  family=binomial)
```

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

```
summary(logistic.fit)
```

```
##
## Call:
## glm(formula = Y ~ ., family = binomial, data = crab.new)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.71546  -0.04300  -0.00133   0.00002   2.71880
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -7.4706     1.8780  -3.978 6.95e-05 ***
## LD1           3.4470     0.8676   3.973 7.10e-05 ***
## LD2           2.9320     0.8332   3.519 0.000433 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 224.934  on 199  degrees of freedom
## Residual deviance:  31.588  on 197  degrees of freedom
## AIC: 37.588
##
## Number of Fisher Scoring iterations: 10
```

```
coef(logistic.fit)
```

```
## (Intercept)         LD1         LD2
##   -7.470611    3.447040    2.932024
```

```
logistic.probs<- predict(logistic.fit,  type="response")



x1 <- seq(-10,10,0.02)
x2 <- seq(-10,10,0.02)
z <- as.matrix(expand.grid(x1,x2))
grid.data<-data.frame(LD1=z[,1], LD2= z[,2])
m <- length(x1)
n <- length(x2)



cb.l.reg <- predict(logistic.fit, newdata=grid.data, type="response")

plot(crab.ldp$x[,1:2], col=y+1, pch=y)
contour(x1,x2,matrix(cb.l.reg,m,n), levels=c(0.5),
add=TRUE,d=FALSE,lty=1,lwd=1)

contour(x1,x2,matrix(cb.l.reg,m,n), levels=c(0.1, 0.25, 0.75, 0.1),
add=TRUE,d=FALSE,lty=2,lwd=2)



logistic.fit.2<- glm(Y~ (LD1+ LD2)^2, data=crab.new,  family=binomial)
summary(logistic.fit.2)
```

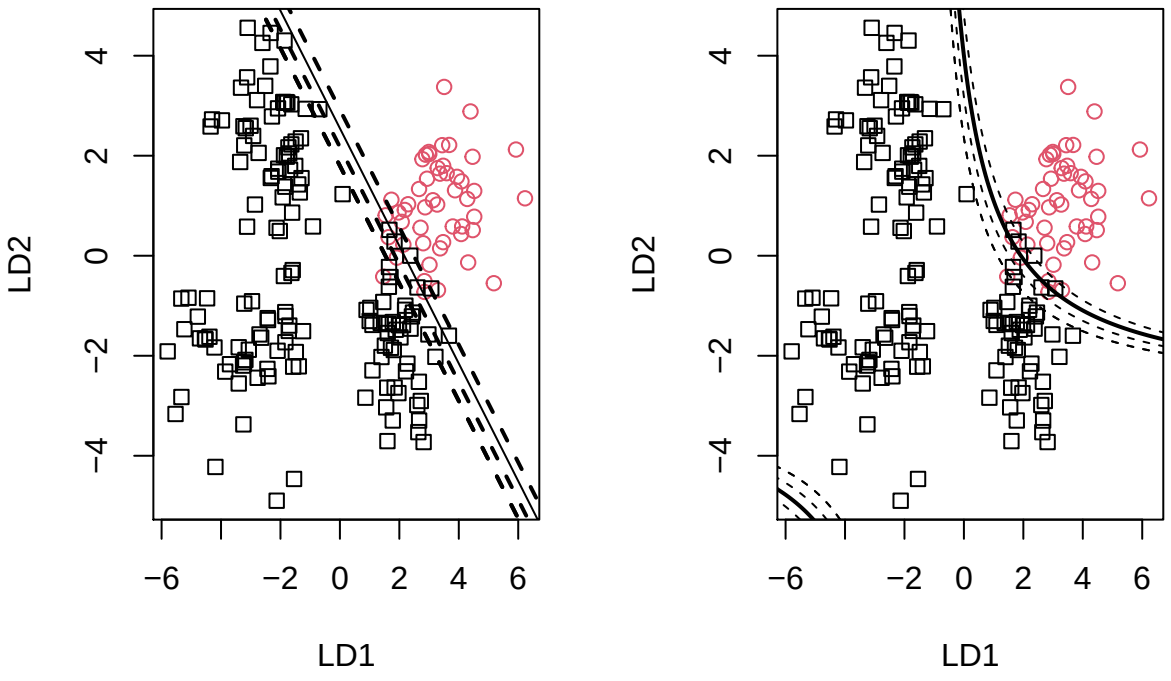
```
##
## Call:
## glm(formula = Y ~ (LD1 + LD2)^2, family = binomial, data = crab.new)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.73674  -0.01663  -0.00484   0.00000   2.24864
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -5.3476     2.1077  -2.537  0.01117 *
## LD1           2.7889     0.9993   2.791  0.00526 **
## LD2           1.3352     1.0289   1.298  0.19437
```

```
## LD1:LD2       0.9925     0.5241   1.894  0.05825 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 224.934  on 199  degrees of freedom
## Residual deviance:  25.059  on 196  degrees of freedom
## AIC: 33.059
##
## Number of Fisher Scoring iterations: 10
```

```
coef(logistic.fit.2)
```

```
## (Intercept)         LD1         LD2     LD1:LD2
##  -5.3476383   2.7888687   1.3352383   0.9925143
```

```
logistic.probs.2<- predict(logistic.fit.2,  type="response")


cb.l.reg.2 <- predict(logistic.fit.2, newdata=grid.data, type="response")

plot(crab.ldp$x[,1:2], col=y+1, pch=y)
contour(x1,x2,matrix(cb.l.reg.2,m,n), levels=c(0.5),
add=TRUE,d=FALSE,lty=1,lwd=2)

contour(x1,x2,matrix(cb.l.reg.2,m,n), levels=c(0.1, 0.25, 0.75, 0.1),
add=TRUE,d=FALSE,lty=2,lwd=1)
```



Example (Spam Emails):

This is a data set collected at Hewlett-Packard Labs, that classifies 4601 e-mails as spam or non-spam. Indeed 2788 e-mails classified as "nonspam" and 1813 classified as "spam". In addition to this class label there are

57 variables indicating the frequency of certain words and characters in the e-mail. A data frame with 4601 observations and 58 variables. This data set can be found in the package `kernlab`.

The first 48 variables contain the frequency of the variable name (e.g., business) in the e-mail. If the variable name starts with num (e.g., num650) the it indicates the frequency of the corresponding number (e.g., 650). The variables 49-54 indicate the frequency of the characters ';', '(', '[', '!', '$', and '#'. The variables 55-57 contain the average, longest and total run-length of capital letters. Variable 58 indicates the type of the mail and is either "nonspam" or "spam", i.e. unsolicited commercial e-mail.

The "spam" concept is diverse: advertisements for products/web sites, make money fast schemes, chain letters, pornography... This collection of spam e-mails came from the collectors' postmaster and individuals who had filed spam. The collection of non-spam e-mails came from filed work and personal e-mails, and hence the word 'george' and the area code '650' are indicators of non-spam. These are useful when constructing a personalized spam filter. One would either have to blind such non-spam indicators or get a very wide collection of non-spam to generate a general purpose spam filter. These data have been taken from the UCI Repository Of Machine Learning Databases at http://www.ics.uci.edu/~mlearn/MLRepository.html

```
library(kernlab)
data(spam)
dim(spam)
```

```
## [1] 4601   58
```

```
spam[1:3, 1:8]
```

```
##   make address  all num3d  our over remove internet
## 1 0.00    0.64 0.64     0 0.32 0.00   0.00     0.00
## 2 0.21    0.28 0.50     0 0.14 0.28   0.21     0.07
## 3 0.06    0.00 0.71     0 1.23 0.19   0.19     0.12
```

To be able to use logistic regression to predict spam/non-spam, suppose

$$Y = \begin{cases} 1, & \text{Spam}, \\ 0, & \text{Non-Spam}. \end{cases}$$

```
#Remember Last column  indicates the type of the mail and is either "nonspam" or "spam"
 table(spam[, ncol(spam)])
```

```
##
## nonspam    spam
##    2788    1813
```

```
 table(as.numeric(spam[, ncol(spam)])-1)
```

```
##
##    0    1
## 2788 1813
```

```r
 Y<-as.numeric(spam[, ncol(spam)])-1
#All the other columns are features in our problem
 X <- spam[ ,-ncol(spam)]

 logistic.fit <- glm(Y ~ ., data=X,family=binomial)
```

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

```r
 summary(logistic.fit)
```

```
##
## Call:
## glm(formula = Y ~ ., family = binomial, data = X)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -4.127  -0.203   0.000   0.114   5.364
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -1.569e+00  1.420e-01 -11.044  < 2e-16 ***
## make              -3.895e-01  2.315e-01  -1.683 0.092388 .
## address           -1.458e-01  6.928e-02  -2.104 0.035362 *
## all                1.141e-01  1.103e-01   1.035 0.300759
## num3d              2.252e+00  1.507e+00   1.494 0.135168
## our                5.624e-01  1.018e-01   5.524 3.31e-08 ***
## over               8.830e-01  2.498e-01   3.534 0.000409 ***
## remove             2.279e+00  3.328e-01   6.846 7.57e-12 ***
## internet           5.696e-01  1.682e-01   3.387 0.000707 ***
## order              7.343e-01  2.849e-01   2.577 0.009958 **
## mail               1.275e-01  7.262e-02   1.755 0.079230 .
## receive           -2.557e-01  2.979e-01  -0.858 0.390655
## will              -1.383e-01  7.405e-02  -1.868 0.061773 .
## people            -7.961e-02  2.303e-01  -0.346 0.729557
## report             1.447e-01  1.364e-01   1.061 0.288855
## addresses          1.236e+00  7.254e-01   1.704 0.088370 .
## free               1.039e+00  1.457e-01   7.128 1.01e-12 ***
## business           9.599e-01  2.251e-01   4.264 2.01e-05 ***
## email              1.203e-01  1.172e-01   1.027 0.304533
## you                8.131e-02  3.505e-02   2.320 0.020334 *
## credit             1.047e+00  5.383e-01   1.946 0.051675 .
## your               2.419e-01  5.243e-02   4.615 3.94e-06 ***
## font               2.013e-01  1.627e-01   1.238 0.215838
## num000             2.245e+00  4.714e-01   4.762 1.91e-06 ***
## money              4.264e-01  1.621e-01   2.630 0.008535 **
## hp                -1.920e+00  3.128e-01  -6.139 8.31e-10 ***
## hpl               -1.040e+00  4.396e-01  -2.366 0.017966 *
## george            -1.177e+01  2.113e+00  -5.569 2.57e-08 ***
## num650             4.454e-01  1.991e-01   2.237 0.025255 *
## lab               -2.486e+00  1.502e+00  -1.656 0.097744 .
## labs              -3.299e-01  3.137e-01  -1.052 0.292972
```

```
## telnet            -1.702e-01  4.815e-01  -0.353 0.723742
## num857             2.549e+00  3.283e+00   0.776 0.437566
## data              -7.383e-01  3.117e-01  -2.369 0.017842 *
## num415             6.679e-01  1.601e+00   0.417 0.676490
## num85             -2.055e+00  7.883e-01  -2.607 0.009124 **
## technology         9.237e-01  3.091e-01   2.989 0.002803 **
## num1999            4.651e-02  1.754e-01   0.265 0.790819
## parts             -5.968e-01  4.232e-01  -1.410 0.158473
## pm                -8.650e-01  3.828e-01  -2.260 0.023844 *
## direct            -3.046e-01  3.636e-01  -0.838 0.402215
## cs                -4.505e+01  2.660e+01  -1.694 0.090333 .
## meeting           -2.689e+00  8.384e-01  -3.207 0.001342 **
## original          -1.247e+00  8.064e-01  -1.547 0.121978
## project           -1.573e+00  5.292e-01  -2.973 0.002953 **
## re                -7.923e-01  1.556e-01  -5.091 3.56e-07 ***
## edu               -1.459e+00  2.686e-01  -5.434 5.52e-08 ***
## table             -2.326e+00  1.659e+00  -1.402 0.160958
## conference        -4.016e+00  1.611e+00  -2.493 0.012672 *
## charSemicolon     -1.291e+00  4.422e-01  -2.920 0.003503 **
## charRoundbracket  -1.881e-01  2.494e-01  -0.754 0.450663
## charSquarebracket -6.574e-01  8.383e-01  -0.784 0.432914
## charExclamation    3.472e-01  8.926e-02   3.890 0.000100 ***
## charDollar         5.336e+00  7.064e-01   7.553 4.24e-14 ***
## charHash           2.403e+00  1.113e+00   2.159 0.030883 *
## capitalAve         1.199e-02  1.884e-02   0.636 0.524509
## capitalLong        9.118e-03  2.521e-03   3.618 0.000297 ***
## capitalTotal       8.437e-04  2.251e-04   3.747 0.000179 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6170.2  on 4600  degrees of freedom
## Residual deviance: 1815.8  on 4543  degrees of freedom
## AIC: 1931.8
##
## Number of Fisher Scoring iterations: 13
```

To see how good is this classification, we can proceed as follow:

```
prob.fit <- predict(logistic.fit,type="response")
predicted_spam <- as.numeric( prob.fit>0.5)
table(predicted_spam,Y)
```

```
##                Y
## predicted_spam    0    1
##              0 2666  194
##              1  122 1619
```

```
 predicted_spam <- as.numeric( prob.fit>0.99)
table(predicted_spam,Y)
```

```
##                Y
## predicted_spam    0    1
##              0 2776 1095
##              1   12  718
```

As we see out of 730 emails marked as spam, 12 were actually not spam. But this is training error. Let us use some parts of our data for test and divide our data to training and test sets. To this end, we set aside 500 emails as our test and remaining part for training our logistic regression model. We then calculate the test and training error by using corresponding confusion tables.

```
set.seed(150)
n <- length(Y)
i <- sample.int(n, size = 500, replace = FALSE)
train <- (1:n)[-i]
test  <- (1:n)[i]

logistic.fit.train <- glm(Y[train] ~ ., data=X[train,],family=binomial)
```

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

```
prob.fit.train <- predict(logistic.fit.train,newdata=X[train,],type="response")
prob.test  <- predict(logistic.fit.train,newdata=X[test,],type="response")
predicted.spam.train <- as.numeric(prob.fit.train > 0.95)
predicted.spam.test  <- as.numeric(prob.test > 0.95)
table(predicted.spam.train, Y[train])
```

```
##
## predicted.spam.train    0    1
##                    0 2461  737
##                    1   22  881
```

```
table(predicted.spam.test, Y[test])
```

```
##
## predicted.spam.test   0   1
##                   0 302  81
##                   1   3 114
```

10 LDA versus logistic regression (Generative versus Discriminative Learning)

As we explained earlier both LDA and logistc regression model the log odds as a linear function of $\mathbf{x} \in \mathbb{R}^p$. In LDA

$$\log\left\{\frac{\mathbb{P}(C=1|\mathbf{X}=\mathbf{x})}{\mathbb{P}(C=2|\mathbf{X}=\mathbf{x})}\right\} = \alpha_0 + \alpha^\top \mathbf{x}$$

where α_0 and α are based on π_j, μ_j and Σ following a strong normality assumption for the distribution of feature values associated with each class. In LDA, estimating α_0 and α is easy.

In logistic regression, however,

$$\log\left\{\frac{\mathbb{P}(C=1|\mathbf{X}=\mathbf{x})}{\mathbb{P}(C=2|\mathbf{X}=\mathbf{x})}\right\}=\beta_0+\boldsymbol{\beta}^\top\mathbf{x},$$

where we estimate β_0 and $\boldsymbol{\beta}$ directly based on the maximum likelihood approach. Generally speaking, logistic regression is more flexible than LDA becasue it does not assume anything about the distribution of $\mathbf{X}$. LDA assumes that $\mathbf{X}$ is normally distributed withinh each class, so thatthe marginal distribution of $\mathbf{X}$ is a mixture of homogenuous normal distribuions

$$\mathbf{X}\sim\sum_{j=1}^{K}\pi_j N(\mu_j,\Sigma).$$

One might easily argue that logistic regression is more robust to situations in which class conditional densities are not normal. On the other side, if the true class conditional densities are normal, or close to it, LDA will be more efficinet, meaning that for logistic regression to perform comparably it will need more data.

Comparing logistic regression with LDA is essentially comparing two important types of models is Statistical learning, that is *Discriminative* versus *Generative models*.

Algorithms that try to learn $\mathbb{P}(y|\mathbf{x})$ directly (such as logistic regression), or algorithms that try to learn mappings directly from the space of inputs $\mathcal{X}$ to the labels $\{0,1\}$, (such as the perceptron algorithm) are called *discriminative* learning algorithms. In a *generative* onde develops algorithms that instead try to model $\mathbb{P}(\mathbf{x}|y)$ (and $\mathbb{P}(y)$).

In LDA we estimate joint distribution of $(\mathbf{X},Y)$ by maximizing the full likelihood

$$\prod_{i=1}^{n}\mathbb{P}(\mathbf{x}_i,y_i)=\prod_{i=1}^{n}\mathbb{P}(\mathbf{x}_i|y_i)\times\prod_{i=1}^{n}\mathbb{P}(y_i),$$

where we assume given y_i the feature values are distributed as a Gaussian distribution. Here Y_i have Bernoulli distributions.

In a logistic regression, we maximize the conditional likelihood $\prod_{i=1}^{n}\mathbb{P}(y_i|\mathbf{x}_i)$ but we ignore the second them $\mathbb{P}(\mathbf{x}_i)$. In other words, we look at the problem through teh following decomposition:

$$\prod_{i=1}^{n}\mathbb{P}(\mathbf{x}_i,y_i)=\prod_{i=1}^{n}\mathbb{P}(y_i|\mathbf{x}_i)\times\prod_{i=1}^{n}\mathbb{P}(\mathbf{x}_i),$$

where we ignore the second part of the right hand side. Since classification only requires the knowledge of $\mathbb{P}(y|\mathbf{x})$ we do not need to estimate the whole joint distribution. Logistic regression leaves teh marginal distribution $\mathbb{P}(\mathbf{x})$ unspecified and relies on less parametric assumptions than LDA. This is an advantage of logistic regression over LDA.

Example (Spam Email): We might be wondering how does logistic regression model perform compared with LDA approach. Remember that LDA requires strong normality assumption!

```
library(MASS)
lda.result<- lda(x=X[train,],grouping=Y[train])
prob.lda <- predict(lda.result,newdata=X[test,])$posterior[,2]
predicted.spam.lda <- as.numeric(prob.lda > 0.95)
table(predicted.spam.test, Y[test])
```

```
##
## predicted.spam.test   0   1
##                   0 302  81
##                   1   3 114
```

```r
table(predicted.spam.lda, Y[test])
```

```
##
## predicted.spam.lda   0   1
##                  0 299 120
##                  1   6  75
```

11 Multiclass Logistic Regression

Let the numbner of classes $K \geq 3$. The logistic regression model arises from the desire to model the posterior probabilities for $K \geq 3$ classes via linear functions in $\mathbf{X}$, while at the same time ensuring that they sum to one and remain in $[0, 1]$. The model has the following form:

$$\begin{aligned}
\log\left\{\frac{\mathbb{P}(C=1|\mathbf{X}=\mathbf{x})}{\mathbb{P}(C=K|\mathbf{X}=\mathbf{x})}\right\} &= \beta_{10} + \boldsymbol{\beta}_1^\top \mathbf{x} \\
\log\left\{\frac{\mathbb{P}(C=2|\mathbf{X}=\mathbf{x})}{\mathbb{P}(C=K|\mathbf{X}=\mathbf{x})}\right\} &= \beta_{20} + \boldsymbol{\beta}_2^\top \mathbf{x} \\
\vdots \quad &= \quad \vdots \\
\log\left\{\frac{\mathbb{P}(C=K-1|\mathbf{X}=\mathbf{x})}{\mathbb{P}(C=K|\mathbf{X}=\mathbf{x})}\right\} &= \beta_{(K-1)0} + \boldsymbol{\beta}_{K-1}^\top \mathbf{x}.
\end{aligned}$$

The model is specified in terms of $(K-1)$ log-odds or logit transofmations (reflectcing that probabilities will sum to one). It is worth notinng that the choice of the denominator in the above formulation is arbitrary and one can replace that with theprobbaility of any other classes. One can easily show that

$$\begin{aligned}
\mathbb{P}(C=j|\mathbf{X}=\mathbf{x}) &= \frac{e^{\beta_{j0}+\beta_j^\top \mathbf{x}}}{1+\sum_{l=1}^{K-1} e^{\beta_{l0}+\boldsymbol{\beta}_l^\top \mathbf{x}}}, \quad j = 1, 2, \ldots, K-1, \\
\mathbb{P}(C=K|\mathbf{X}=\mathbf{x}) &= \frac{1}{1+\sum_{l=1}^{K-1} e^{\beta_{l0}+\boldsymbol{\beta}_l^\top \mathbf{x}}},
\end{aligned}$$

which sum to one. For this formulation, let

$$\boldsymbol{\beta} = (\beta_{10}, \boldsymbol{\beta}_1^\top, \beta_{20}, \boldsymbol{\beta}_2^\top, \ldots, \beta_{(K-1)0}, \boldsymbol{\beta}_{K-1}^\top)_{D\times 1}^\top, \quad D = (K-1)(p+1),$$

be the unknown parameters of our multiclass logist regression formulation. Also, let $\boldsymbol{\beta}_l = (\beta_{l0}, \boldsymbol{\beta}_l^\top)^\top$. Then

$$\mathbb{P}(C=l|\mathbf{X}=\mathbf{x}) = \mathbb{P}_l(\mathbf{x}; \boldsymbol{\beta}).$$

Now, based on a trainings sample $(\mathbf{x}_1, C_1), \ldots, (\mathbf{x}_N, C_N)$ where C_i is the class label of the i-th observation such that $C_i \in \{1, 2, \ldots, K\}$, we would like to estimate model parameters. Let $\mathcal{Y}$ be the concatenated indicator vectors of dimention $N \times (K-1)$, that is,

$$\mathcal{Y} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_{K-1} \end{pmatrix} \quad \text{with} \quad \mathbf{y}_i = \begin{pmatrix} \mathbb{I}(C_1 = i) \\ \mathbb{I}(C_2 = i) \\ \vdots \\ \mathbb{I}(C_{K-1} = i) \end{pmatrix}, \quad 1 \leq i \leq K-1.$$

Suppose also that $\mathcal{P}$ is the concatenated vector of fitted probabilities of dimension $N(K-1)$, that is

$$\mathcal{P} = \begin{pmatrix} \mathcal{P}_1 \\ \mathcal{P}_2 \\ \vdots \\ \mathcal{P}_{K-1} \end{pmatrix} \quad \text{with} \quad \mathcal{P}_i = \begin{pmatrix} \mathbb{P}_i(\mathbf{x}_1; \boldsymbol{\beta}) \\ \mathbb{P}_i(\mathbf{x}_2; \boldsymbol{\beta}) \\ \vdots \\ \mathbb{P}_i(\mathbf{x}_N; \boldsymbol{\beta}) \end{pmatrix}, \quad 1 \leq i \leq K-1.$$

Similarly, let

$$\mathcal{X} = \begin{pmatrix} \mathbf{X} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{X} & \cdots & \mathbf{0} \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{X} \end{pmatrix}_{N(K-1)\times(p+1)(K-1)}$$

and

$$\mathcal{W} = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1(K-1)} \\ W_{21} & W_{22} & \cdots & W_{2(K-1)} \\ \vdots & \vdots & \cdots & \vdots \\ W_{(K-1)1} & W_{(K-1)2} & \cdots & W_{(K-1)(K-1)} \end{pmatrix}_{N(K-1)\times N(K-1)},$$

where each W_{km}, $1 \leq k, m \leq K-1$ is an $N \times N$ diagonal matrix such that

1. When $k = m$, W_{mk} has $\mathbb{P}_k(\mathbf{x}_i, \boldsymbol{\beta}^{old})(1 - \mathbb{P}_k(\mathbf{x}_i, \boldsymbol{\beta}^{old}))$ as its i-th diagonal elemnt.
2. When $k \neq m$, the i-th diagonal element in W_{mk} is $-\mathbb{P}_k(\mathbf{x}_i, \boldsymbol{\beta}^{old})\mathbb{P}_m(\mathbf{x}_i, \boldsymbol{\beta}^{old})$.

Similar to the case with $K = 2$, we first calculate

$$\frac{\partial l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \mathcal{X}^\top(\mathcal{Y} - \mathcal{P})$$
$$\frac{\partial^2 l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top} = -\mathcal{X}^\top \mathcal{W} \mathcal{X}.$$

The formula for updating $\boldsymbol{\beta}^{new}$ for $K = 2$ holds for multiclass as well and we have

$$\boldsymbol{\beta}^{new} = (\mathcal{X}^\top \mathcal{W} \mathcal{X})^{-1} \mathcal{X}^\top \mathcal{W} \mathcal{Z},$$

where

$$\mathcal{Z} = \mathcal{X}\boldsymbol{\beta}^{old} + \mathcal{W}^{-1}(\mathcal{Y} - \mathcal{P}),$$

or

$$\boldsymbol{\beta}^{new} = \boldsymbol{\beta}^{old} + (\mathcal{X}^\top \mathcal{W} \mathcal{X})^{-1} \mathcal{X}^\top (\mathcal{Y} - \mathcal{P}).$$

For initialization one option is to take $\boldsymbol{\beta} = \mathbf{0}$. Convergence is not guranteed but it is often the case.

12 Naive Bayes Classifier

As we discussed earlier one approach to classification is through a generateive model where one need to specify the conditional distribution of the feature variable $\mathbf{x}$ given the class lables y, that is $\mathbb{P}(\mathbf{x}|y)$. In order to apply the generative model, we need to either estimate estimate of specify the multivariate conditional distribution of $\mathbf{X}$. There are many methods to do this. One strategy is based on nonparametric methods using kernel density estimation which makes very weak assumptions about the class-conditional distributions. Another approach is to make strong parametric assumptions about the form of the class-conditional distributions, such as what we did in LDA or QDA. In LDA for example, we assume the distribution of the features in each class is ellipsodial such as multivariate normal which is the most familiar member of such distributions. In ellipsodial distributions $\mathbb{P}(\mathbf{x}|y)$ factorizes into terms which involve only pairs (X_i, X_j), so the other higher order interactions are assumed to be zero and not to exist.

A Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem (from Bayesian statistics) with even stronger assumptions that $\mathbb{P}(\mathbf{x}|y)$ will factorize into a product of its univariate marginals, that the components of $\mathbf{X}$ are independent, that is $\mathbb{P}(\mathbf{x}|y) = \prod_{i=1}^{p} \mathbb{P}(x_i|y)$. Clearly this model is most likely to be unrealistic for most problems. This approach which sometimes is called an _idiot's$ Bayes methodology is simply a very simplistic model where assumes that the underlying probability model would be an "independent feature model".

Naive Bayes technique is especially appropriate when teh dimension p of the feature space is high and density estimation becomes very unattractive. By assuming the independency of the features give a class $Y = j$, feastures in $\mathbf{X}$ are independent, and we have

$$f_j(\mathbf{x}) = \prod_{l=1}^{P} f_{jl}(x_l).$$

Now, the individual class-conditional marginal densities f_{jl} can each be estimated separately using one-dimesnional kernel density estimates. Note that when a component X_j of $\mathbf{X}$ is discrete, then an appropriate histogran estimate can be used.

In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature. For example, a loan application might be approaved if the applicant has a job, with age in the range 30 to 40 years old, and is not married. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier considers all of these properties to independently contribute to the probability that a loan applicatin is approved. Depending on the precise nature of the probability model, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without believing in Bayesian probability or using any Bayesian methods.

In spite of their naive design and apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. As we explain below, there are sound reasons for this: its intrinsic simplicity means low variance in its probability estimates; although these will typically be biased, this may not matter in classification situations as long as the rank order is preserved; in many situations the variables undergone a selection process which tends to reduce their interdependencies. More over, since the model has teh form of a product, by taking the logs it can be converted into a sum with the usual consequent computational advantages.

Note that in work on supervised learning methods for classification it is standard ti refer to $\mathbb{P}(Y = j) = \pi_j$ as the class j prior probability as its gives the probability that an object belongs to class j prior to observing any information about the object. Combining the prior with $\mathbb{P}(\mathbf{x}|Y = j) = f_j(\mathbf{x})$, as shown below, gives the posterior probability, after observing $\mathbf{x}$. This combination is done via the Bayes theorem and this is why the method is called Bayes. Note however that after we formulated the problem using the Bayesian approach one does not really need to stick with the Bayesian approach and can estimate the unknown parameters of class-conditional distributions using standard approaches like maximum likelihood method, etc.

12.1 Explaining Naive Bayes

Given features $\mathbf{X} = (X_1, X_2, \dots X_n)$ and a class label $Y \in \{1, 2, \dots, K\}$, we wish to know

$$\mathbb{P}(Y = j|\mathbf{X} = \mathbf{x}) = \mathbb{P}(Y = j|X_1 = x_1, X_2 = x_2, \dots X_p = x_p)$$

By Bayes' Theorem:

$$\begin{aligned}\mathbb{P}(Y = j|\mathbf{X} = \mathbf{x}) &= \frac{\mathbb{P}(Y = u)\mathbb{P}(\mathbf{X} = \mathbf{x}|Y = j)}{\mathbb{P}(\mathbf{X} = \mathbf{x})} \\ &\propto \pi_j f_j(\mathbf{x}).\end{aligned}$$

where $\pi_j = \mathbb{P}(Y = j)$ and $f_j(\mathbf{x})$ is the class-conditional distribution of the features $\mathbf{X}$ in class j.

Naive Bayes 'naively' assumes that every feature X_r is conditionally independent of every other X_s $(r \neq s)$ given $Y = j$, i.e.:

$$p(Y = j|\mathbf{X} = \mathbf{x}) \propto \pi_j \prod_i^p f_{ji}(x_i)$$

Now, one can use the MAP (Maximum Aposteriori) method to perform classification to assign an object with observed features $\mathbf{x}$ to class j^* if

$$j^* = \text{argmax}_j \Big\{ \pi_j \prod_i^p f_{ji}(x_i) \Big\}.$$

Note that in a classifiction problem when $Y \in \{1, \dots, K\}$, by taking calss K as the base, one can take the logit transform of $\mathbb{P}(Y = j|\mathbf{X})$ to get

$$\begin{aligned}\log \frac{p(Y = j|\mathbf{X} = \mathbf{x})}{p(Y = K|\mathbf{X} = \mathbf{x})} &= \log \frac{\pi_j f_j(\mathbf{X})}{\pi_K f_K(\mathbf{X})} \\ &= \log \frac{\pi_j \prod_i^p f_i(X_i)}{\pi_K \prod_i^p f_{Ki}(X_i)} \\ &= \log \frac{\pi_j}{\pi_K} + \sum_{i=1}^p \log \frac{f_{ji}(X_i)}{f_{Ki}(X_i)} \\ &= \alpha_j + \sum_{i=1}^p g_{ji}(X_i),\end{aligned}$$

which has the form of a generalized additive model.

##Why the independence assumption is not so absurd?

In spite of the fact that the assumption of independence is almost always wrong and the covariance matrices are rarely diagonal, yet practical comparison of the Naive Bayes classification approach with more modern and sophisticated classification approaches have shown that the Naive Bayes perform surprisingly well. There seem to be very few studies which report poor performance for the independence Bayes model in comparison with other methods.

One important reason that the Naive Bayes approach works well in many applications is that it requires fewer parameters to be estimated than alternative methods which seek to model interactions in the individual class-conditional $\mathbf{x}$ distributions. This is because models with fewer parameters will tend to have a lower variance for the estimates of $\mathbb{P}(Y = j|\mathbf{x})$. Sometimes, the reduction in variance resulting from the relatively few parameters

involved in the Naive Bayes model will be more than compensate for any increase in the bias, and one might suspect that this often occurs in practice and is one explanation for the frequent good performance of the method.

In general, if simple classification is the aim, then bias may not matter that much. The best achievable classification results will be obtained provided that $\hat{\mathbb{P}}(Y = j|\mathbf{x}) > \hat{\mathbb{P}}(Y = i|\mathbf{x})$ whenever $\mathbb{P}(Y = j|\mathbf{x}) > \mathbb{P}(Y = i|\mathbf{x})$, and the bias, even severe bias will not matter provided it is in the right direction. For classification accurate probability estimates ate not required-a;; that need be preserved is the rank order.

In practice, the Naive Bayes method may beat flexible and highly parametric alternatives when small sample sizes are involved because of the relatively low variance of its estimates of $\mathbb{P}(Y = j|\mathbf{x})$, and may in fact also do well when we have large sample sizes because relatively large bias may not mater over most of the possible patterns.

In many practical situations the variables will go under a variable selection process before being combined to yield a classification rule. Typically, this process leads to the elimination of variables which are highly correlated with those already included as such variables are less likely to contribute much additional information to the problem. This means that, in real problems, there will be a tendency towards selecting weakly correlated variables,s o the Naive Bayes approach maybe realistic model in such situations.

It might also be possible that the independence model will yield the optimal separating surface even if the variables are correlated.

12.2 Naive Bayes Classification in R

Performing the Naive Bayes classification is simple in `R`. This can be done using:

- `naiveBayes()` that creates a classifier given observation data and the class for each observation. This is in the e1071 package.
- `predict()` receives the classifier, some observations, and returns a vector with their predicted classes

The `naiveBayes()` assumes normal distributions for the features and estimates the unknown model parameters using the MLE approach. You can also use `NaiveBayes()' in`klaR‘ package.

Example Naive Bayes classification for Iris Data.

```
data(iris) # load iris dataset
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

```r
library(e1071)
# create a classifier using naive bayes using the first 4 columns as the data,
# the last coloumn as the class for each observation
classifier<-naiveBayes(iris[,1:4], iris[,5], type="raw")
```

Use `predict()` that receives the classifier object plus some observations (iris[,-5] is the original data without its class) and returns a sugested class for each observation.

```r
predicted.classes <- predict(classifier, iris)
head(predicted.classes,n=12)
```

```
##  [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa
## [11] setosa setosa
## Levels: setosa versicolor virginica
```

Then the method `table()` presents a confusion matrix between the sugested classe vector with the real class vector. In this case the classification was very good, but this is a biased result since we are using the same data in the training and in the test sets

```r
table(predicted.classes, iris[,5], dnn=list('predicted','actual'))
```

```
##             actual
## predicted    setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         47         3
##   virginica       0          3        47
```

```r
classifier$apriori / sum(classifier$apriori) # the prior distribution for the classes
```

```
## iris[, 5]
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
```

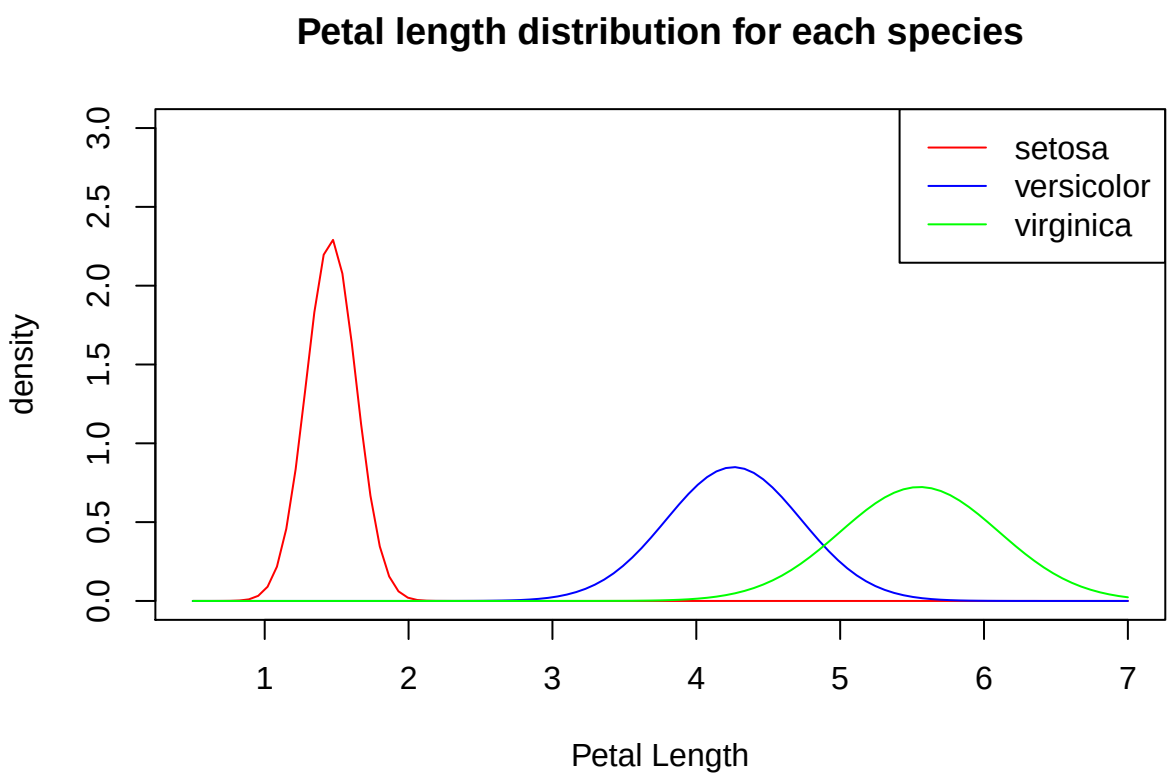
Since the predictor variables here are all continuous, the naive Bayes classifier generates three Gaussian (Normal) distributions for each predictor variable: one for each value of the class variable Species. The first column is the mean, the 2nd column is the standard deviation.

```r
classifier$tables$Petal.Length
```

```
##             Petal.Length
## iris[, 5]     [,1]      [,2]
##   setosa     1.462 0.1736640
##   versicolor 4.260 0.4699110
##   virginica  5.552 0.5518947
```

Let's plot these ones:

```
plot(0:3, xlim=c(0.5,7), col="red", ylab="density",
     type="n", xlab="Petal Length",
     main="Petal length distribution for each species")
curve(dnorm(x, classifier$tables$Petal.Length[1,1],
            classifier$tables$Petal.Length[1,2]),
            add=TRUE, col="red")
curve(dnorm(x, classifier$tables$Petal.Length[2,1],
            classifier$tables$Petal.Length[2,2]),
            add=TRUE, col="blue")
curve(dnorm(x, classifier$tables$Petal.Length[3,1],
            classifier$tables$Petal.Length[3,2]),
            add=TRUE, col ="green")
legend("topright", c("setosa", "versicolor", "virginica"),
            col = c("red","blue","green"), lwd=1)
```



These values could also be accessed directly in the dataset. Say for `Petal.Length`:

```
mean(iris[iris$Species=="setosa",]$Petal.Length)
```

```
## [1] 1.462
```

```
sd(iris[iris$Species=="setosa",]$Petal.Length)
```

```
## [1] 0.173664
```

So, let's say we want to find, without using the R function, all three possible values of $\mathbb{P}$(Class lable|Features):

```r
observation <- data.frame(Sepal.Length = 5.0,
                          Sepal.Width  = 3.2,
                          Petal.Length = 1.5,
                          Petal.Width  = 0.3)  # this observation lies within Setosa cluster
```

For setosa class:

$$\mathbb{P}(C = \text{setosa}|\text{observation}) \propto \mathbb{P}(C = \text{setosa})\mathbb{P}(\text{Sepal.Length} = 5.0|C = \text{setosa})\mathbb{P}(\text{Sepal.Width} = 3.2|C = \text{setosa})\mathbb{P}(\text{Petal.Le}$$

A similar computation is needed for the other two classes. Let's implement them:

```r
iris.classes <- c("setosa","versicolor","virginica")
iris.attributes <- names(iris)[-5]

means     <- rep(0,length(iris.attributes))
sds       <- rep(0,length(iris.attributes))
densities <- rep(0,length(iris.attributes))

p.Cs <- c(0,0,0)              # p(C=class)
p.Cs_observation <- c(0,0,0)  # p(C=class | observation)

for (c in 1:length(iris.classes)) {
  p.Cs[c] <- nrow(iris[iris$Species==iris.classes[c],]) / nrow(iris)

  for(i in 1:length(iris.attributes)) {
    means[i] <- sapply(iris[iris$Species==iris.classes[c],][i], mean)
    sds[i]   <- sapply(iris[iris$Species==iris.classes[c],][i], sd)
    densities[i] <- dnorm(as.numeric(observation[i]), means[i], sds[i])
  }

  p.Cs_observation[c] <- p.Cs[c] * prod(densities) # the final value for each class
}

names(p.Cs_observation) <- c("setosa","versicolor","virginica")
p.Cs_observation
```

```
##       setosa   versicolor    virginica
## 2.466810e+00 1.953732e-15 4.920209e-23
```

```r
p.Cs_observation / sum(p.Cs_observation) # normalize
```

```
##       setosa   versicolor    virginica
## 1.000000e+00 7.920075e-16 1.994563e-23
```

This means that our naive bayes would predict 'setosa' with **VERY** high likelihood.

With the use of naiveBayes() we get the same prediction:

```r
# type="raw" shows the probabilities
predict(classifier, observation, type="raw")
```

```
##      setosa   versicolor    virginica
## [1,]      1 7.920075e-16 1.994563e-23
```