Chapter 10

Machine Learning and Data Science

10.1 Introduction

In the previous chapters, when discussing problems of detection or estimation, we have assumed that we know the joint distribution (PMF or PDF) that describes the relationships between the random variables. In detection, we used the conditional distributions of the observed data given various hypotheses, as well as the prior distributions on the hypotheses, to design algorithms that selected the best among the hypotheses based on the observed data. In estimation, we used the joint distribution of the observed and unobserved data to predict the value of the unobserved variables, based on the observed variables.

We have also covered discussed foundational concepts in statistics, based on estimating parameters from observed data. We focused on the sample mean of a random variable, based on a set of independent, identically distributed samples of that random variable. We showed that the sample mean is a good estimator of the true mean in this case, and developed interesting bounds on how many samples we need to get a "good enough" estimate of the true mean with high probability.

In this chapter, we provide a short introduction to the field of machine learning. Roughly speaking, machine learning is about making inferences such as detection or estimation, without knowing the joint distribution of random variables involved. Instead of knowledge of the distributions, we have training data that we claim represents independent, identically distributed samples generated by the same experiment. Using this data, machine learning algorithms design detection and estimation algorithms. The "learning" occurs by using the provided data samples to design detection or estimation algorithms.

In machine learning, we speak of two types of learning: Supervised learning and unsupervised learning. In supervised learning, the data provided includes samples of both the observations and the hypothesis labels. The machine learning algorithm can use the labels to learn an appropriate decision or estimation rule. In unsupervised learning, the data consists only of samples of the observations, without labels. The machine learning algorithm will have to uncover important features of the data and appropriate classes on its own. In this chapter, we will focus primarily on supervised learning, although we will discuss techniques such as clustering and principal component analysis that are used effectively in unsupervised learning.

The problem of classification in machine learning is equivalent to the problem of hypothesis testing in chapter 6. In these problems, the observations X can be discrete, continuous or a mixture of discrete and continuous variables, and the labels Y are a discrete hypothesis label. Similarly, the problem of regression is estimation 7 in a machine learning context. Here the observations are again random vectors of different types, and the label variable Y is a real number or vector that we are trying to estimate.

10.2 Learning probabilities from data

Suppose we were interested in "learning" a classifier for binary hypothesis testing problem. The design theory of Chapter 6 suggests that what we need to know are the likelihoods for the observation data $\underline{X}$ under both hypotheses. Assuming the observations are continuous random vectors, we need knowledge of $f_{\underline{X}|H_0}(\underline{x})$ and $f_{\underline{X}|H_1}(\underline{x})$. Knowing these, the decision rule consists of forming the likelihood ratio $\mathcal{L}(\underline{x})$ and comparing this ratio to a threshold, a design parameter that we choose.

In machine learning, the likelihood functions are not known. Instead, we are given training data for each hypotheses, of the form

$$(\underline{x}_1, H_0), \ldots, (\underline{x}_{n_0}, H_0); \quad (\underline{x}_{n_0+1}, H_1), \ldots, (\underline{x}_n, H_1),$$

where each data observation $\underline{x}_i$ is associated with a label that indicates whether this data was sampled from the likelihood $f_{\underline{X}|H_0}(\underline{x})$ or $f_{\underline{X}|H_1}(\underline{x})$. The data $\underline{X}$ are the observed features, and are assumed to be d-dimensional vectors in $\Re^d$. Our goal is to learn approximations to the likelihoods $f_{\underline{X}|H_0}(\underline{x})$ and $f_{\underline{X}|H_1}(\underline{x})$.

There are two types of approaches for estimating densities from data: parametric and non-parametric, which we describe below.

10.2.1 Parametric models

The parametric approach for machine learning assumes that we know the family that the densities $f_{\underline{X}|H_0}(\underline{x})$ or $f_{\underline{X}|H_1}(\underline{x})$ belong to. For instance, we assume that the densities are jointly Gaussian, with unknown means $\underline{\mu}_0, \underline{\mu}_1$ and unknown covariances Σ_0, Σ_1 respectively. The idea behind parametric density estimation is to use the training data to estimate the unknown parameters of the likelihood functions. The parameter estimation problems are straightforward, and use the maximum likelihood estimation algorithms described in Chapter 7.

Specifically, assume we are given n independent observations of a random vector $\underline{X} \in \Re^d$, denoted as $\underline{x}_1, \ldots, \underline{x}_n$. We assume that the joint PDF of $\underline{X}$ is one of a family of PDFs, parametrized by unknown parameters $\underline{\theta}$, so that $f_{\underline{X}}(\underline{x}) = g(\underline{x}, \underline{\theta})$. For instance, the function $g(\cdot)$ can be the joint density of a d-dimensional Gaussian random vector, with unknown mean and covariance matrix, which form the vector of unknown parameters $\underline{\theta}$. Based on the data, we estimate these parameters using an estimation algorithm such as maximum-likelihood, so that

$$\widehat{\underline{\theta}} = \underset{\underline{\theta}}{\operatorname{argmax}} \prod_{k=1}^{n} g(\underline{x}_k, \underline{\theta}).$$

The resulting approximate PDF of $\underline{X}$ is then $f_{\underline{X}}(\underline{x}) = g(\underline{x}, \widehat{\underline{\theta}})$. We illustrate this with examples below.

Example 10.1

We have two hypotheses we are trying to detect, based on a scalar Gaussian observation. Specifically, under hypothesis H_0, the observation X is a Gaussian random variable with mean 2, variance 4. Under H_1, it is a Gaussian random variable with mean 5, variance 4. Using the theory of Chapter 6, the maximum likelihood decision rule for this problem is

$$D_{ML}(x) = \begin{cases} H_1 & x \geq 2.5 \\ H_0 & x < 2.5 \end{cases}$$

and the probability of error in this detector is $Q(1.25) = 0.106$.

Now, assume we did not know the parameters of the Gaussian distribution, but instead are given 1000 samples (x_i, y_i), where $y_i \in \{H_0, H_1\}$, from each of the two distributions, $f_{X|H_0}(x), f_{X|H_1}(x)$. We assume the first 1000 samples come from H_0, and the second from H_1. We refer to the values x_i as the data, and the values y_i as the labels.

The vector of unknown parameters in the densities is $\underline{\theta} = [m_0, m_1, \sigma^2]$, corresponding to the unknown means and the common variance of the conditional densities $f_{X|H_0}(x), f_{X|H_1}(x)$. The parametric form of the likelihood function for each hypothesis is:

$$f_{X|H_k}(x) \equiv g_k(x, \underline{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-m_k)^2}{2\sigma^2}}, \quad k = 0, 1$$

Using this and the fact that each of the data samples is independent, we can write the estimation problem as

$$\widehat{\underline{\theta}} = \underset{\underline{\theta}}{\operatorname{argmax}} \prod_{j=1}^{1000} g_0(x_j, \underline{\theta}) \prod_{j=1001}^{2000} g_1(x_j, \underline{\theta})$$

We can maximize the logarithm of the right-hand side as a simplification, to obtain

$$\widehat{\underline{\theta}} = \underset{\underline{\theta}}{\operatorname{argmax}} \sum_{j=1}^{1000} \left(-\frac{\ln(2\pi\sigma^2)}{2} - \frac{(x_j - m_0)^2}{2\sigma^2} \right) + \sum_{j=1001}^{2000} \left(-\frac{\ln(2\pi\sigma^2)}{2} - \frac{(x_j - m_1)^2}{2\sigma^2} \right)$$

Differentiating with respect to m_0, m_1 and σ^2 and setting the results to zero yields the following values for the maximum likelihood estimates:

$$\widehat{m}_0 = \frac{1}{1000}\sum_{i=1}^{000} x_i; \quad \widehat{m}_1 = \frac{1}{1000}\sum_{i=1001}^{2000} x_i$$

Since we assume the variances of the two densities are the same, we estimate the joint variance as

$$\widehat{\sigma}^2 = \frac{1}{2000}\big(\sum_{i=j}^{1000}(x_j - \widehat{m}_0)^2 + \sum_{j=1001}^{2000}(x_i - \widehat{m}_1)^2\big).$$

With these estimates, we can approximate the conditional densities $f_{X|H_0}(x), f_{X|H_1}(x)$, and implement an optimal maximum likelihood decision rule, which results in:

$$D_{ML}(x) = \begin{cases} H_1 & x \geq \frac{\widehat{m}_1 - \widehat{m}_0}{2} \\ H_0 & \text{otherwise.} \end{cases}$$

and we predict our probability of error to be $Q(\frac{\widehat{m}_1 + \widehat{m}_0}{2\sqrt{\widehat{\sigma^2}}})$.

We implemented the above problem in a Python script described below:

```
import numpy as np
X0 = 2*np.random.randn(1000,)  #generate H0 train data
X1 = 2*np.random.randn(1000,1).ravel()+5 # H1 train data
test0 = 2*np.random.randn(1000,1).ravel()  #H0 test data
test1 = 2*np.random.randn(1000,1).ravel()+5  #H1 test data
m0 = sum(X0)/1000.
m1 = sum(X1)/1000.0
var = (sum((X0 - m0)**2)+sum((X1 - m1)**2)) /2000.0
T = 0.5*(m1 - m0)
Pe = (sum(test0 > T) + sum(test1 < T))/2000
print(m0,m1,var)
print(T,Pe,T/np.sqrt(var) )
```

and obtained the following estimates:

$$\widehat{m}_0 = -0.018; \quad \widehat{m}_1 = 4.906; \quad \widehat{\sigma}^2 = 4.135.$$

Based on these estimates, the threshold in the decision rule is 2.462, and the predicted performance is $Q(1.21) \approx 0.101$.

We generated 2000 additional samples as test data, and evaluated the empirical performance of our detector on the test data, as the script indicates. The empirical probability of error was $P_e = 0.105$, which is a bit higher than predicted, because the estimated models from the training data are different than the actual models used to generate the data.

In this example, we generated lots of data to estimate 3 unknown parameters: 2000 independent samples. We will see that this complicates the problem when we deal with larger numbers of unknown parameters.

10.2.2 Nonparametric Density Estimation

Can we estimate the likelihoods $f_{\underline{X}|H_0}(\underline{x}), f_{\underline{X}|H_1}(\underline{x})$ without assuming a parametric form? There are numerous techniques that attempt to do this. The problem of non-parametric density estimation can be summarized as follows: given N independent samples $\underline{x}_k, k = 1, \ldots, N$ of an n-dimensional random vector $\underline{X}$ with PDF $f_{\underline{X}}(\underline{x})$, generate an estimate of the probability density function for all values $\underline{x} \in \Re^n$.

One approach is to construct an empirical probability mass function based on the observed samples, as follows:

$$\widehat{P}_{\underline{X}}(\underline{x}) = \begin{cases} 0 & \underline{x} \neq \underline{x}_k \text{ for some } k \in \{1, \ldots, N\} \\ \frac{1}{N} & \underline{x} = \underline{x}_k \text{ for some } k \in \{1, \ldots, N\}. \end{cases}$$

Unfortunately, this is a discrete probability mass function, and not a density. Furthermore, it assigns zero probability to obtaining any data values that are not in the training set $\underline{x}_k, k = 1, \ldots, N$.

The approach we propose for generating a better density estimate is **kernel density estimation** (KDE), which is a version of the Parzen's window estimator. KDE is a nonparametric density estimator, requiring no assumption that the underlying density function is from a parametric family. KDE will learn the shape of the density from the data automatically. The idea behind KDE is to blur the empirical probability mass function using a smooth kernel so that the probability mass function is extrapolated to a density that has range on values that were not included in a training set. Kernel density estimators smooth out the contribution of each observed data point over a local neighborhood of that data point.

Define a kernel $K(\underline{x})$ to be a smooth non-negative function with a peak at $\underline{x} = \underline{0}$, such that $\int_{\underline{x}} K(\underline{x})\, d\underline{x} = 1$. We can interpret the kernel as a probability density function, as it satisfies the non-negativity and normalization properties of probability densities. For most applications, we use standard Gaussian kernels, of the form

$$K_h(\underline{x}) = \frac{1}{(\sqrt{2\pi}h)^n} e^{-\frac{\underline{x}^T \underline{x}}{2h^2}},$$

which is the product of independent Gaussian densities with zero-mean and standard deviation h in each of the n dimensions of $\underline{X}$. The parameter h is known as the kernel width.

With this kernel, the KDE approximation is

$$\widehat{f}_{\underline{X}}(\underline{x}) = \frac{1}{N} \sum_{k=1}^{N} K_h(\underline{x} - \underline{x}_k)$$

Note that, for an arbitrary point $\underline{x} \in \Re^n$, the density will be determined primarily by the data points $\underline{x}_k$ that are within a distance of $3h$ of $\underline{x}$. The quality of a kernel estimate depends strongly on the value of its bandwidth h. It's important to choose the most appropriate bandwidth as a value that is too small or too large is not useful. Small values of h lead to very spiky estimates (not much smoothing) while larger h values lead to oversmoothing.

Example 10.2
Let's approximate the density of a Gaussian random variable X with mean 0, variance 1 using KDE. We select 100 independent, randomly generated samples of X, and we show the resulting KDE densities using different values for the width h in the figure below. We see that for values $h = 0.1$, the approximate density is too rough. For $h = 0.4, 0.5$, the density approximation is accurate. For $h = 0.6, 0.7$ we see the density is oversmoothed.

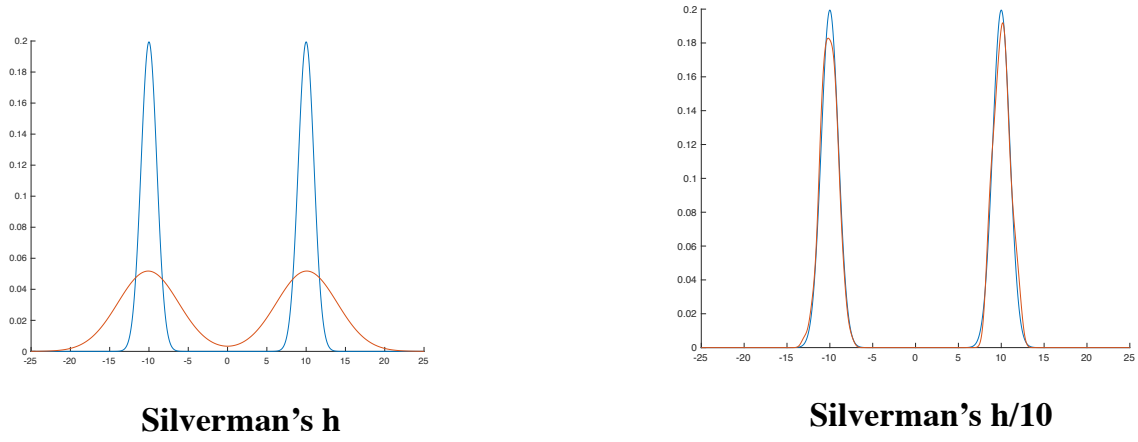
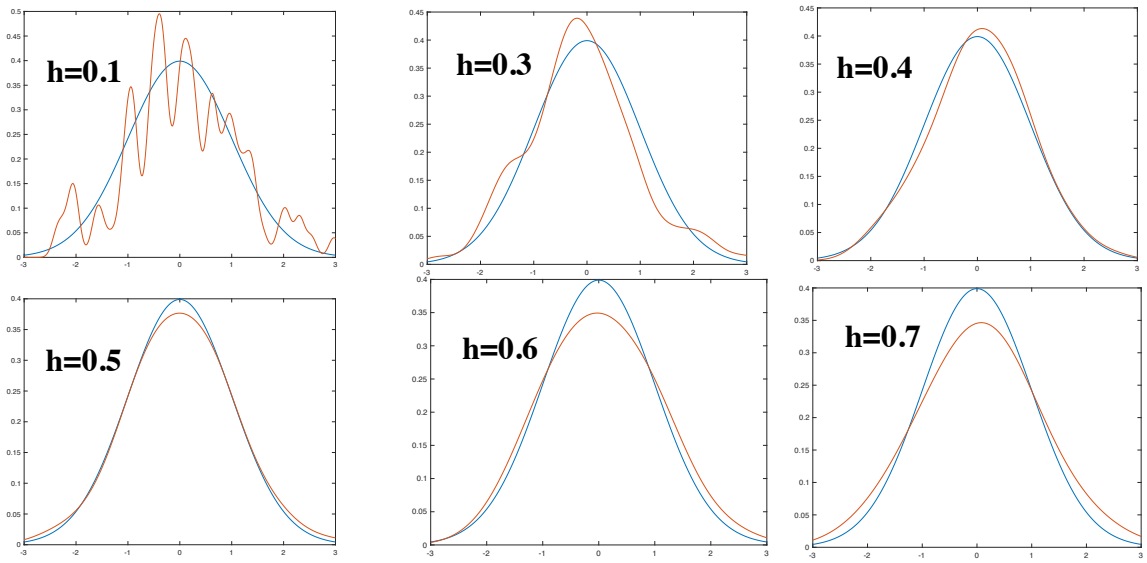


Figure 10.1: A multi-modal density where the standard deviation is not representative of the curvature. Red indicates the KDE approximation to the true density in blue.



There are many approaches to selecting the right value of h. A rule of thumb, known as Silverman's rule, is based on approximating Gaussian densities, and is given by $h = \left(\frac{4\widehat{\sigma}^5}{3N}\right)^{\frac{1}{5}} \approx 1.06\widehat{\sigma}N^{-1/5}$. If we evaluate this for the example above with $N = 100$ and $\sigma = 1$, we get $h \approx 0.42$. However, this rule can often be wrong if the density has multiple peaks. For instance, consider the density in Figure 10.1 below. Silverman's approximation yields an h that oversmooths the density, as the estimated standard deviation is more reflective of the spread in the two peaks than the curvature of the density. A different rule that reduces h by an order of magnitude yields an excellent approximation. In general, one selects h using a search that involves some form of cross-validation, where part of the data is used to estimate the density and another part is used to validate that the estimated density is accurate.

KDE can be integrated into classification problems with supervised learning. Given samples of $\underline{X}$ generated independently by $f_{\underline{X}|H_0}(\underline{x})$, we can construct a KDE estimate $\widehat{f}_{\underline{X}|H_0}(\underline{x})$ of this likelihood function. We can similarly construct a KDE estimate $\widehat{f}_{\underline{X}|H_1}(\underline{x})$. With these two estimates, the maximum likelihood decision rule is

$$\widehat{D}_{ML}(\underline{x}) = \begin{cases} H_1 & \widehat{f}_{\underline{X}|H_1}(\underline{x}) \geq \widehat{f}_{\underline{X}|H_0}(\underline{x}) \\ 0 & \text{otherwise.} \end{cases}$$

The main limitations of KDE estimators are two-fold. First, for high-dimensional data, it is hard to interpolate accurately. The number of data points required grows exponentially with the number of dimensions. Second, the complexity of the classifier is large: one has to compute the kernel distance to all the training points, which can be slow as the number of training points increases. We will show the performance we can achieve with KDE estimation on a real classification problem, described in the next section.

10.3 The IRIS data set

As a motivating application, we use a particular data set that was used by the father of modern statistics, Ronald Fisher. In addition to his work on statistics, Fisher was a mathematical biologist and applied statistics to maximum likelihood classification problems. This particular data set was reported in a paper in 1936, "The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis." The data was collected to quantify the morphologic variation of Iris flowers of three related species. Two of the three species were collected from the same pasture, picked on the same day and measured with the same apparatus. The IRIS data set is widely studied, and copies of it are easily downloaded on the internet.

The data set of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other. This discriminant is still referred to as Fisher's linear discriminant. The three types of flowers are shown in Figure 10.2.

Figure 10.2: The three types of iris flowers in the IRIS data set.

The four features in this data correspond to the sepal length in centimeters, the sepal width in centimeters, the petal length in centimeters and the petal width in centimeters. The petal and sepal leaves of an iris flower are shown in Figure 10.3.

To illustrate the properties of this data set, we performed a statistical analysis using the Python Seaborn package, that shows a kernel density estimate for the marginal densities for each of the four features as well as plots showing pairs of features, color coded to each of the iris varieties. This output is shown in Figure 10.4. This exploratory analysis shows that it will be relatively easy to separate the Setosa variety from the other two varieties, because of its smaller petals, but it will be harder to separate the other two varieties.

Example 10.3

We use the IRIS data set to evaluate the performance of parametric max-likelihood classifiers and explore some of their limitations. The observations in the IRIS data set are 4-dimensional vectors. There are three hypotheses, which we denote as H_0 (Setosa), H_1 (Versicolor) and H_2 (Virginica). We only have 50 observation samples per hypotheses. We assume that the densities $f_{\underline{X}|H_0}(\underline{x}), f_{\underline{X}|H_1}(\underline{x}), f_{\underline{X}|H_2}(\underline{x})$ are joint Gaussian densities, with means $\underline{\mu}_0, \underline{\mu}_1, \underline{\mu}_2$ respectively and a common covariance matrix $\mathbf{\Sigma}$. Note that the number of parameters required to estimate the densities is three four-dimensional means and a four-by-four symmetric covariance matrix, leading to $4 \times 3 + 10 = 22$ unknown parameters to be estimated,

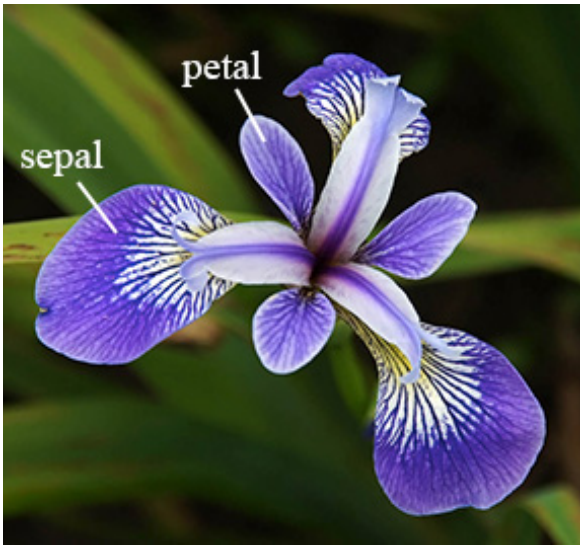


Figure 10.3: Names of leaves of iris flowers in the IRIS data set.

with 150 four-dimensional measurements. This is limited data for a problem with that many unknown parameters, and we can expect difficulty in estimating the densities accurately.

We assume the data provided is in the form $\{(\underline{x}_k, y_k)\}, k = 1, \ldots, 150$, where $y_k \in \{H_0, H_1, H_2\}$. The estimate of each of the class means is obtained using maximum likelihood estimation as

$$\widehat{\underline{\mu}}_j = \frac{1}{50} \sum_{k=1}^{150} \underline{x}_k I_{y_k = H_j}, \ j = 0, 1, 2,$$

where I_z is the indicator function which is 1 when z is true, and 0 otherwise. The covariance estimate is generated by

$$\widehat{\Sigma} = \frac{1}{150} \sum_{k=1}^{150} \left(\sum_{j=0}^{2} I_{y_k = H_j} (\underline{x}_k - \widehat{\underline{\mu}}_j)(\underline{x}_k - \widehat{\underline{\mu}}_j)^T \right).$$

We implemented the above equations and obtained the following estimates:

$$\widehat{\underline{\mu}}_0 = \begin{bmatrix} 5.006 \\ 3.428 \\ 1.462 \\ 0.246 \end{bmatrix} ; \ \widehat{\underline{\mu}}_1 = \begin{bmatrix} 5.936 \\ 2.77 \\ 4.26 \\ 1.326 \end{bmatrix} ; \ \widehat{\underline{\mu}}_2 = \begin{bmatrix} 6.588 \\ 2.974 \\ 5.552 \\ 2.026 \end{bmatrix} ; \ \widehat{\Sigma} = \begin{bmatrix} 0.261 & 0.091 & 0.165 & 0.038 \\ 0.091 & 0.114 & 0.055 & 0.032 \\ 0.165 & 0.05 & 0.183 & 0.042 \\ 0.038 & 0.032 & 0.042 & 0.041 \end{bmatrix}.$$

With these density estimates, we implemented a maximum likelihood classifier. On the training data, the performance of the classifier had a probability of error of 0.02. We subsequently broke the training data so that 70% of the data was used for training, and 30% was used for testing. The probability of error went up to 0.044, which is still very good given the limited amount of training data.

Example 10.4
We return to the IRIS data set, but using the KDE nonparametric estimators discussed previously to approximate the likelihood functions for each of the three species of iris flowers. For this problem, we can always get zero probability of error on the training data by selecting a small value of h, since this will put all the weight on the actual data point being tested, which is part of the training set. Thus, we divided the data into 70% training, 30% testing and evaluated the performance of the KDE maximum likelihood classifier described above. On the test data, our probability of error was 0.044, which is the same as the parametric estimator discussed in Example 10.3.

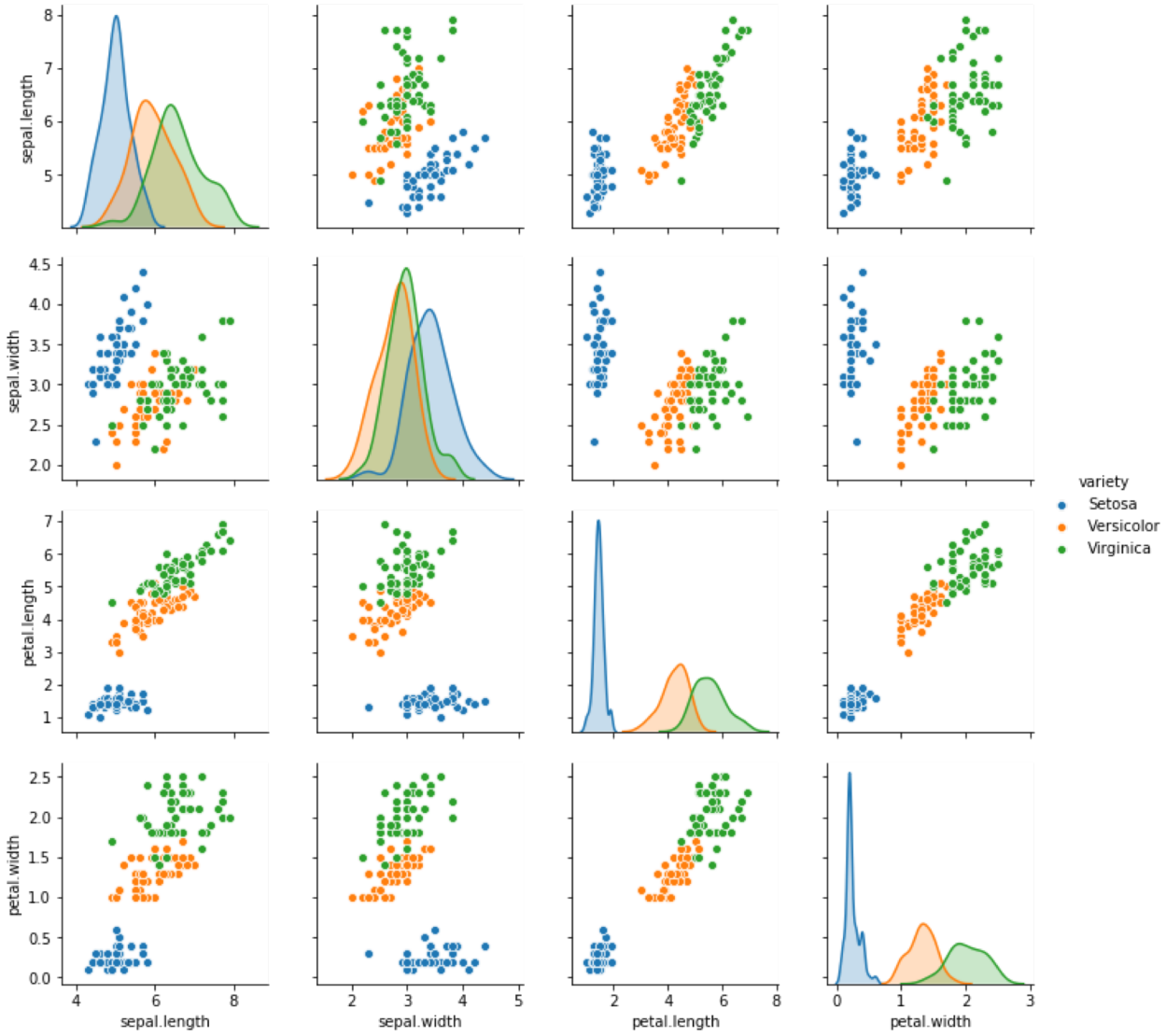


Figure 10.4: Seaborn pairs analysis of IRIS data.

10.4 Binary Classification

In this section, we focus on describing various machine learning approaches to the problem of selecting one of two hypotheses. This problem is a generalization of the binary hypothesis testing problem, to the case where we don't have probability models for the observations conditioned on the hypotheses. Instead, we are provided sample observation values that are obtained under each of the two hypotheses. Our goal is to design a decision rule that maps new observations into a selection of which hypothesis is best.

The binary classification problem has two hypotheses, H_0 and H_1. To simplify notation, we associate the decision value -1 with H_0 and $+1$ with H_1. We observe a vector of features $\underline{X}$ with values in $\Re^d$, and we want to design a decision rule $D(\underline{x})$ that maps the observed vector $\underline{X} = \underline{x}$ into one of the two decisions $\{-1, +1\}$.

Assume we are given training data of the form $(\underline{X}_1, Y_1), \ldots, (\underline{X}_n, Y_n)\}$, where $\underline{X}_k \in \Re^d$ is a sample observation value, and $Y_k \in \{-1, +1\}$ is the label that indicates which of the two hypotheses generated

that observation. The decision rule $D(\underline{x})$ assigns a label of -1 or 1 to each possible observed vector $\underline{x}$. We typically measure the performance of a decision rule by the error rate, which is the probability that a point is misclassified. Since we don't have probability models readily available, we compute this as the fraction of sample values that are misclassified, as we will show later.

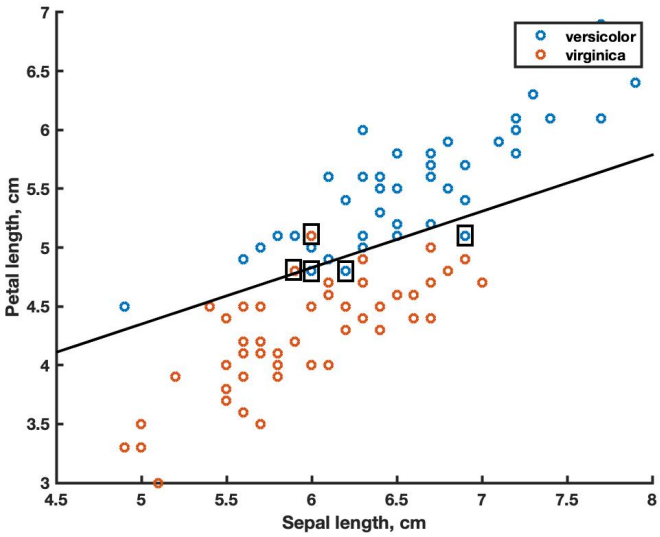


Figure 10.5: Sepal length versus petal length for two types of Iris flowers

Figure 10.5 plots a pair of features, sepal length and petal length, for two types of iris flowers, Versicolor and Virginica. The figure also shows a potential decision rule, indicated by a straight line. For data points above the line, the decision rule declares that flower type is Versicolor, whereas for data points below the line, the decision rule declares that the flower type is Virginica. The figure highlights the five data points where the decision rule makes errors, by including a black rectangle among the points.

In this section, we describe various approaches for designing binary classifiers using labeled training data and supervised training.

10.4.1 Clustering Classifiers

Assume we are given training data of the form $(\underline{X}_1, Y_1), \ldots, (\underline{X}_Y, Y_n)\}$, where $Y_k \in \{H_0, H_1\}$ is a categorical label. Given this labeled data, it is straightforward to compute the average observation under each hypothesis as:

$$\widehat{\underline{\mu}}_0 = \frac{\sum_{k=1}^{n} \underline{X}_k I_{Y_k=H_0}}{\sum_{k=1}^{n} I_{Y_k=H_0}}; \quad \widehat{\underline{\mu}}_1 = \frac{\sum_{k=1}^{n} \underline{X}_k I_{Y_k=H_1}}{\sum_{k=1}^{n} I_{Y_k=H_1}}$$

The clustering classifier assigns to an input value $\underline{x}$ the decision that has its average closest to the input value. That is,

$$D(\underline{x}) = \begin{cases} H_0, & \| \underline{X} - \widehat{\underline{\mu}}_0 \| < \| \underline{X} - \widehat{\underline{\mu}}_1 \| \\ H_1, & \| \underline{X} - \widehat{\underline{\mu}}_0 \| \geq \| \underline{X} - \widehat{\underline{\mu}}_1 \| \end{cases}$$

Note that the clustering classifier is easy to extend to K hypotheses, as long as training data is provided for each. Furthermore, the classifier can be extended to unsupervised classification, by analyzing the data using a clustering algorithm and discovering the clusters in the data.

The clustering classifier is computed in Figure 10.6 for the two length features for the IRIS data set, for the classes Versicolor and Virginica. The two class centers are shown as filled diamonds in each of the

classes. Looking at the results, we see that 15 out of the 100 data samples are classified incorrectly, for an error rate of 15% when evaluated using the training data.

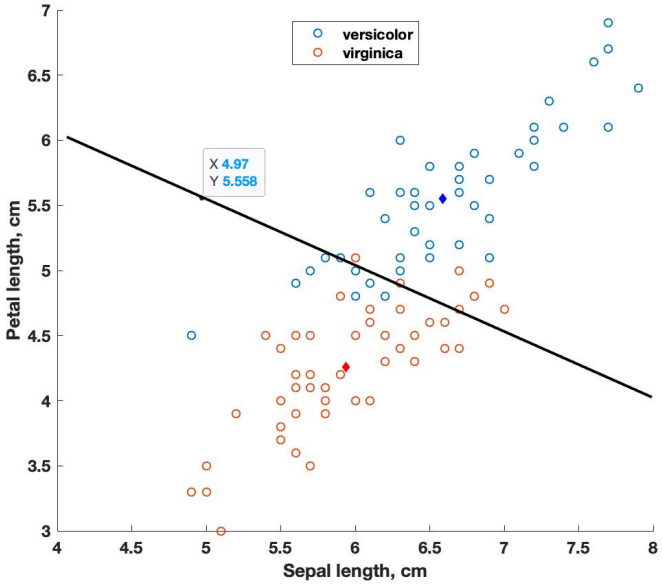


Figure 10.6: Illustration of clustering classifier

10.4.2 Nearest Neighbor and K-Nearest Neighbor Classifiers

Nearest neighbor classifiers are similar to clustering classifiers, but involve more computation. In clustering classifiers, the trained classifier only needs to remember the average values estimated for each class. In contrast, the classifier for nearest neighbor classifiers needs to remember all the training data provided.

To classify an input $\underline{x}$, the nearest neighbor classifier finds the training data point $\underline{X}_k$ that is closest to $\underline{x}$, and then assigns the label of that data point to the data $\underline{X}$. That is, one finds j^* such that

$$\| \underline{x} - \underline{X}_{j^*} \| \leq \| \underline{x} - X_j \| \text{ for all } j = 1, \ldots, n$$

The nearest neighbor classifier is then $D(\underline{x}) = Y_{j^*}$.

A simple extension of a nearest neighbor classifiers is to find the K nearest neighbors of the input data $\underline{x}$, denoted as $\underline{X}_{j_1}, \ldots, \underline{X}_{j_K}$. The label assigned to $\underline{x}$ would be determined by the labels $Y_{j_1}, \ldots, Y_{j_K}$, usually in majority voting. Finding the K nearest neighbors efficiently usually requires advanced data structures, particularly when the dimension d of the data is large.

10.4.3 Discriminant Analysis

Discriminant analysis classifiers are based on Gaussian parametric approximations of the likelihood functions $f_{\underline{X}|H_0}(\underline{x}), f_{\underline{X}|H_1}(\underline{x})$. The most common type of discriminant analysis is **Linear Discriminant Analysis (LDA)**, which assumes that the likelihood functions $f_{\underline{X}|H_0}(\underline{x}), f_{\underline{X}|H_1}(\underline{x})$ are jointly Gaussian with different means $\underline{\mu}_0, \underline{\mu}_1$ but with the same covariance matrix Σ. The decision rule is the maximum likelihood decision rule with the approximate densities.

Given training data of the form $(\underline{X}_1, Y_1), \ldots, (\underline{X}_Y, Y_n)\}$, where $Y_k \in \{H_0, H_1\}$ is a categorical label, compute the number of samples of type H_0, H_1 as $n_0 = sum_{k=1}^n \underline{X}_k I_{Y_k=H_0}, n_1 = sum_{k=1}^n \underline{X}_k I_{Y_k=H_1}$. Given this labeled data, it is straightforward to estimate the means of each of the two densities, in the same manner as we estimated the centers of the clustering classifiers earlier:

$$\widehat{\underline{\mu}}_0 = \frac{\sum_{k=1}^n \underline{X}_k I_{Y_k=H_0}}{n_0}; \quad \widehat{\underline{\mu}}_1 = \frac{\sum_{k=1}^n \underline{X}_k I_{Y_k=H_1}}{n_1}.$$

Estimating the common covariance matrix Σ is more involved. A common estimator is

$$\widehat{\Sigma} = \frac{1}{n-2} \sum_{k=1}^n \Big((\underline{X}_k - \widehat{\underline{\mu}}_0)^2 I_{Y_k=H_0} + (\underline{X}_k - \widehat{\underline{\mu}}_1)^2 I_{Y_k=H_1} \Big)$$

A different way of computing the same estimate is to first estimate $\widehat{\Sigma}_0, \widehat{\Sigma}_1$ as the covariances based on data labeled as H_0 or H_1 respectively, as:

$$\widehat{\Sigma}_0 = \frac{1}{n_0 - 1} \sum_{k=1}^n (\underline{X}_k - \widehat{\underline{\mu}}_0)^2 I_{Y_k=H_0}$$

$$\widehat{\Sigma}_1 = \frac{1}{n_1 - 1} \sum_{k=1}^n (\underline{X}_k - \widehat{\underline{\mu}}_1)^2 I_{Y_k=H_1}$$

We can subsequently combine these into a single estimate, as

$$\widehat{\Sigma} = \frac{1}{n-2} \big((n_0 - 1)\widehat{\Sigma}_0 + (n_1 - 1)\widehat{\Sigma}_1 \big).$$

With these estimates, we approximate the likelihoods of $\underline{X}$ given H_0, H_1 as

$$\widehat{f}_{\underline{X}|H_0}(\underline{x}) = \frac{1}{\sqrt{\det(2\pi\widehat{\Sigma})}} e^{-\frac{1}{2}(\underline{x} - \widehat{\underline{\mu}}_0)^T \widehat{\Sigma}^{-1} (\underline{x} - \widehat{\underline{\mu}}_0)}$$

$$\widehat{f}_{\underline{X}|H_1}(\underline{x}) = \frac{1}{\sqrt{\det(2\pi\widehat{\Sigma})}} e^{-\frac{1}{2}(\underline{x} - \widehat{\underline{\mu}}_1)^T \widehat{\Sigma}^{-1} (\underline{x} - \widehat{\underline{\mu}}_1)}$$

To derive the max-likelihood classifier using these likelihood estimates, we compare the log-likelihood ratio to 0, as

$$\begin{aligned} \ln\big(\mathcal{L}(\underline{x})\big) &= \frac{1}{2}(\underline{x} - \widehat{\underline{\mu}}_0)^T \widehat{\Sigma}^{-1} (\underline{x} - \widehat{\underline{\mu}}_0) - \frac{1}{2}(\underline{x} - \widehat{\underline{\mu}}_1)^T \widehat{\Sigma}^{-1} (\underline{x} - \widehat{\underline{\mu}}_1) \\ &= (\widehat{\underline{\mu}}_1 - \widehat{\underline{\mu}}_0)^T \widehat{\Sigma}^{-1} \underline{x} - \frac{1}{2}(\widehat{\underline{\mu}}_1^T \widehat{\Sigma}^{-1} \widehat{\underline{\mu}}_1 - \widehat{\underline{\mu}}_0^T \widehat{\Sigma}^{-1} \widehat{\underline{\mu}}_0) \underset{H_0}{\overset{H_1}{\gtrless}} 0. \end{aligned}$$

This decision rule is a linear decision rule of the form

$$D(\underline{x}) = \begin{cases} H_1 & \text{if } \underline{a}^T \underline{x} \geq b, \\ H_0 & \text{elsewhere,} \end{cases}$$

where $\underline{a} = \widehat{\Sigma}^{-1}(\widehat{\underline{\mu}}_1 - \widehat{\underline{\mu}}_0)$, and $b = \frac{1}{2}(\widehat{\underline{\mu}}_1^T \widehat{\Sigma}^{-1} \widehat{\underline{\mu}}_1 - \widehat{\underline{\mu}}_0^T \widehat{\Sigma}^{-1} \widehat{\underline{\mu}}_0)$. The LDA decision rule is similar to the closest average decision rule, except that the distances between data and the centers are modified by the estimate of the inverse covariance. If $\widehat{\Sigma} = \mathbf{I}_d$, the d-dimensional identity matrix, then the LDA decision rule is equivalent to the closest average decision rule.

We show the LDA decision on the same two-dimensional data for the IRIS data set as before. Compared with the clustering classifier, the decision rule has shifted the orientation of the separation line based on the estimated covariance, and has reduced the number of errors to 5, leading to a 5% error on the training data.

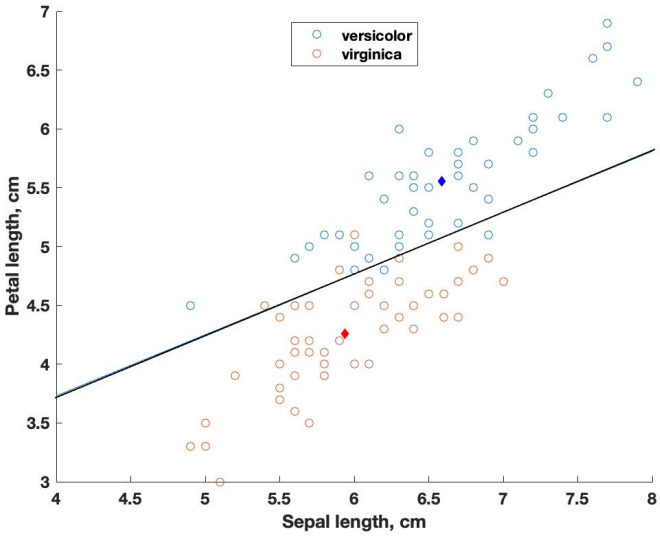


Figure 10.7: LDA decision rule for selecting between Versicolor and Virginica Iris

Instead of Linear Discriminant Analysis, we can do **Quadratic Discriminant Analysis (QDA)**, which is based on parametric modeling of the likelihood functions $f_{\underline{X}|H_0}(\underline{x}), f_{\underline{X}|H_1}(\underline{x})$ are jointly Gaussian with different means $\underline{\mu}_0, \underline{\mu}_1$ and different covariances Σ_1 and Σ_2. Given the same training data, the estimates can be obtained as

$$\widehat{\underline{\mu}}_0 = \frac{\sum_{k=1}^n \underline{X}_k I_{Y_k=H_0}}{n_0}; \quad \widehat{\underline{\mu}}_1 = \frac{\sum_{k=1}^n \underline{X}_k I_{Y_k=H_1}}{n_1}.$$

$$\widehat{\Sigma}_0 = \frac{1}{n_0-1}\sum_{k=1}^n (\underline{X}_k - \widehat{\underline{\mu}}_0)^2 I_{Y_k=H_0}; \quad \widehat{\Sigma}_1 = \frac{1}{n_1-1}\sum_{k=1}^n (\underline{X}_k - \widehat{\underline{\mu}}_1)^2 I_{Y_k=H_1}.$$

Computing the log-likelihood ratio results in

$$\begin{aligned}\ln\left(\mathcal{L}(\underline{x})\right) &= \frac{1}{2}(\underline{x}-\widehat{\underline{\mu}}_0)^T\widehat{\Sigma}_0^{-1}(\underline{x}-\widehat{\underline{\mu}}_0) - \frac{1}{2}(\underline{x}-\widehat{\underline{\mu}}_1)^T\widehat{\Sigma}_1^{-1}(\underline{x}-\widehat{\underline{\mu}}_1) + \frac{\ln(\det[\widehat{\Sigma}_0])}{2} - \frac{\ln(\det[\widehat{\Sigma}_1])}{2} \\ &= \frac{1}{2}\underline{x}^T(\widehat{\Sigma}_0^{-1} - \widehat{\Sigma}_1^{-1})\underline{x} + (\widehat{\underline{\mu}}_1^T\widehat{\Sigma}_1^{-1} - \widehat{\underline{\mu}}_0^T\widehat{\Sigma}_0^{-1})\underline{x} + \frac{\widehat{\underline{\mu}}_0^T\widehat{\Sigma}_0^{-1}\widehat{\underline{\mu}}_0 - \widehat{\underline{\mu}}_1^T\widehat{\Sigma}_1^{-1}\widehat{\underline{\mu}}_1}{2} + \frac{1}{2}\ln\left(\frac{\det[\widehat{\Sigma}_0]}{\det[\widehat{\Sigma}_1]}\right) \underset{H_0}{\overset{H_1}{\gtrless}} 0\end{aligned}$$

Figure 10.8 shows the boundary of the quadratic decision rule obtained by QDA for the two feature IRIS data set used in the previous examples. Although the empirical error rate on the training data is the same as the LDA empirical error rate 5%, the curvature of the region is likely to improve the error rate on test data.

10.4.4 Perceptron Classifier

Many modern approaches to binary classification use complex parametric decision functions, and use large-scale optimization techniques to select the parameters of such decision functions. Examples of such decision functions are neural networks, which use interconnected layers of linear and nonlinear elements with weights to map an observed input $\underline{x}$ to a decision $D(\underline{x})$. The training of such networks is beyond the scope of this course. However, we will describe a simple nonlinear neural network model for classification, proposed by Rosenblatt in 1958. It is known as Rosenblatt's perceptron network, and is illustrated in Figure **??**. Rosenblatt's perceptron was an attempt to model the processing of a neuron, based on earlier work (1943)

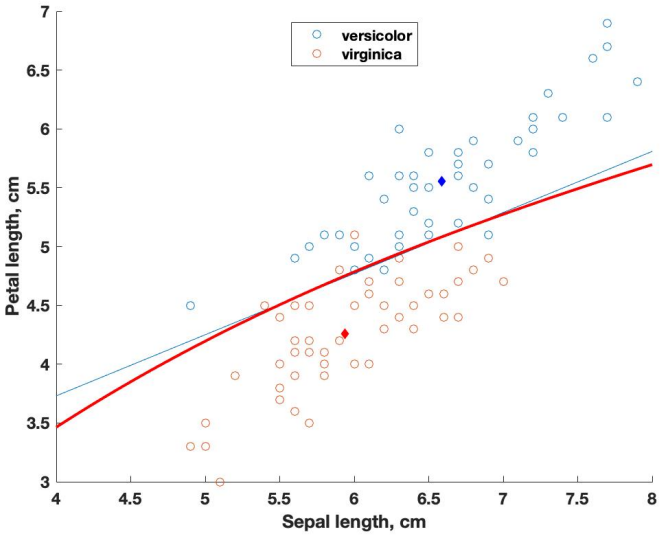


Figure 10.8: LDA decision rule for selecting between Versicolor and Virginica Iris

by McCulloch and Pitts. The figure shows how a vector of inputs, combined with weights is added and passed to a nonlinear function that examines the sign and selects the hypothesis.

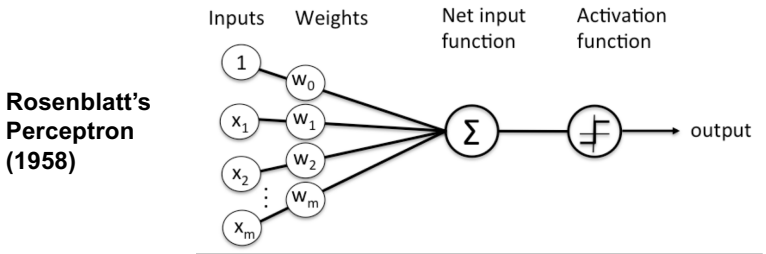


Figure 10.9: Rosenblatt's Perceptron Model.

Training a perceptron classifier can be accomplished without the use of iterative optimization techniques. The training is based on principles of regression: given data $(\underline{X}_k, Y_k), k = 1, \ldots, n$ with labels $Y_k \in \{H_0, H_1\}$, change the labels to numbers, where $Y_k = H_0$ is changed to $Y_k = -1$, and $Y_k = H_1$ is changed to $Y_k = 1$. We now have numerical data $\{(\underline{X}_1, Y_1), \ldots, (\underline{X}_n, Y_n)\}$.

Let the vector $\underline{a}$ correspond to the weights $w_1, \ldots, w_m$ that multiply the data $\underline{x}$, and the scalar b corresponds to the weight w_0 that is added as a bias to the nonlinear function in the perceptron classifier. We want to find the best linear predictor of Y from $\underline{X}$ to minimize the least-squares error:

$$\min_{\underline{a},b} \frac{1}{n} \sum_{k=1}^{n} (Y_k - \underline{a}^T \underline{X}_k - b)^2.$$

This optimization is known as linear regression, and is very similar to the problem of linear least-squares estimation, except that instead of PDFs to compute means and variances, we have sample data. Linear regression with minimum mean square error objectives was developed by Gauss in the early 19th century to estimate planetary orbits.

We can solve the linear regression problem using the LLSE solution of Chapter 7. Specifically, we compute

approximately

$$\mathbb{E}[\underline{X}] = \frac{1}{n}\sum_{k=1}^{n} \underline{X}_k; \quad \mathbb{E}[Y] = \frac{1}{n}\sum_{k=1}^{n} Y_k.$$

$$\mathsf{Var}[\underline{X}] = \Sigma_{\underline{X}} = \frac{1}{n-1}\sum_{k=1}^{n}(\underline{X}_k - \mathbb{E}[\underline{X}])(\underline{X}_k - \mathbb{E}[\underline{X}])^T; \mathsf{Cov}[Y, \underline{X}] = \Sigma_{Y,\underline{X}} = \frac{1}{n-1}\sum_{k=1}^{n}(\underline{X}_k - \mathbb{E}[\underline{X}])^T(Y_k - \mathbb{E}[Y]).$$

The LLSE estimator is

$$\widehat{y}_{LLSE}(\underline{x}) = \mathbb{E}[Y] + \Sigma_{Y,\underline{X}}\Sigma_{\underline{X}}^{-1}(\underline{x} - \mathbb{E}[X]),$$

which makes the optimal regression vector $\underline{a}^T = \Sigma_{Y,\underline{X}}\Sigma_{\underline{X}}^{-1}$ and the constat $b = \mathbb{E}[Y] - \underline{a}^T\mathbb{E}[x]$. This solution yields the optimal weights to use in the perceptron classifier.

As an alternative, we can convert the weight optimization problem by optimizing for the weights $\underline{w}$ directly, where $\underline{w}$ is a $(d+1)$-dimensional vector. We do this by forming the data matrix

We often convert this to a homogeneous formulation by grouping the constant d into the estimation vector $\underline{c}$, defining the vector $\underline{b}^T = \begin{bmatrix}\underline{c}^T & d\end{bmatrix}$, so that the estimate is of the form

$$\widehat{y}(\underline{x}) = \begin{bmatrix}\underline{c}^T & d\end{bmatrix}\begin{bmatrix}\underline{X} \\ 1\end{bmatrix} = \underline{b}^T\begin{bmatrix}\underline{X} \\ 1\end{bmatrix}.$$

This gets rid of the bias term d by merging it into the unknown vector $\underline{b}$. Note that this requires adding an extra dimension to the observations $\underline{X}$.

$$\mathbf{X} = \begin{bmatrix}1 & \underline{X}_1^T \\ 1 & \underline{X}_2^T \\ \vdots & \\ 1 & \underline{X}_n^T\end{bmatrix} \qquad \underline{y} = \begin{bmatrix}y_1 \\ y_2 \\ \vdots \\ y_n\end{bmatrix}.$$

Note that we have added an extra column of all 1 to the data, to account for the coefficient w_0 in the perceptron network. With this notation, the mean square error becomes

$$MSE = \frac{1}{n}\big(\underline{y} - \mathbf{X}\underline{w}\big)^T(\underline{y} - \mathbf{X}\underline{w}).$$

This is now a quadratic vector optimization problem. We solve this by taking the gradient with respect to $\underline{b}$ and setting this equal to zero:

$$\begin{aligned}\nabla_{\underline{b}} MSE = \nabla_{\underline{b}}\left(\frac{1}{n}\big(\underline{y} - \mathbf{X}\underline{w}\big)^T(\underline{y} - \mathbf{X}\underline{w})\right) = \frac{2}{n}\mathbf{X}^T\left(\mathbf{X}\underline{w} - \underline{y}\right) = 0 \\ \iff \mathbf{X}^T\mathbf{X}\underline{w}^* = \mathbf{X}^T\underline{y}\end{aligned}$$

The last set of equations are known as the **Normal Equations** for least squares estimation. As long as the dimension d is not very large, we can invert the matrix $\mathbf{X}^T\mathbf{X}$ and obtain the regression solution $\underline{w}^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\underline{y}$. For large d, different approaches are used, as matrix inversion can be expensive. When d is larger than the number of data points N, this matrix is not invertible, and one must use a different approach, such as computing its pseudo-inverse.

Example 10.5

Consider a 1-dimensional example of regression. The points $X_1, \ldots, X_{100}$ are uniformly spaced in the interval [-1,1]. We generate the measurements y_k as follows:

$$y_k = 2X_k + w_k$$

where w_k are independent samples of a Gaussian random variable with mean 0 and variance 1. Note that we have intentionally selected X to have average 0, and y to also have average 0, so that we can assume $w_0 = 0$. In this case, the data matrix is a vector:

$$\mathbf{X} = \begin{bmatrix}X_1 \\ X_2 \\ \vdots \\ X_n\end{bmatrix}.$$

Based on the 100 values of (X_k, y_k), we generate an estimator as follows:

$$\widehat{y}(x) = w^* x, \qquad w^* = \frac{\sum_{k=1}^N x_k y_k}{\sum_{k=1}^N x_k^2}$$

The results are show in Figure 10.10.

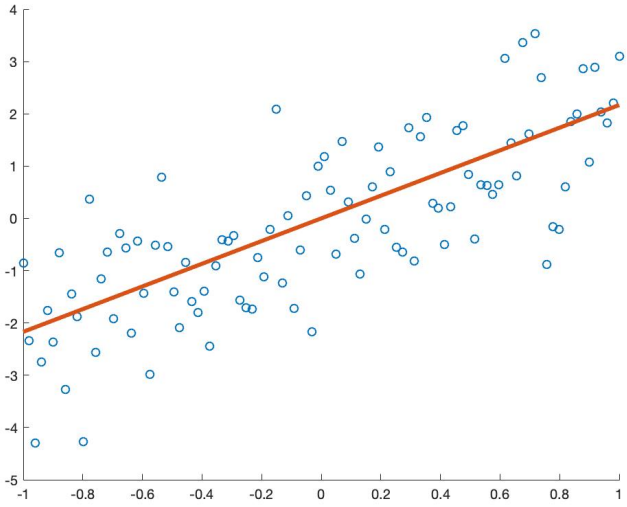


Figure 10.10: One-dimensional regression.

Example 10.6
Consider a 2-dimensional regression. We generate 400 sample points X_k uniformly spaced in the unit square $[-1, 1]^2$. Each of the points X_k is a 2-dimensional vector (X_{k1}, X_{k2}). At each of these points, we observe the function value

$$y_k = 2X_{k,1} + 3X_{k,2} + w_k$$

where w_k are independent samples of a Gaussian random variable with mean 0, variance 1.

The results of our 2-dimensional regression for this case are shown in Figure 10.11. The linear regression estimate is $(\underline{w}^*)^T = \begin{bmatrix} 2.028 & 3.033 \end{bmatrix}$, which are close to the true values used to simulate the data.

10.5 Dimensionality Reduction

In machine learning classification problems, the observations often involve large numbers of variables. For high-dimensional observations, it is hard to visualize the data and design the classifiers using either parametric techniques or optimization techniques. Dimensionality reduction algorithms reduce the number of random variables under consideration, by transforming the data to a smaller set of important features.

For making inferences and other decisions, not all dimensions of the observed data are critical. Consider images, expressed as long vectors. Certain "features" of the images are informative, but not all pixels contain relevant information; most images have similar borders, and the variability in certain regions is similar. Essentially, we wish to discover "features" where images differ the most and discard "features" with little variability.

There are many methods for dimensionality reduction. In this section, we focus on one of the simplest and most popular methods: **Principal Component Analysis (PCA)**. PCA finds linear combinations of the data that are uncorrelated and represent the best approximations to the variability in the data. It is

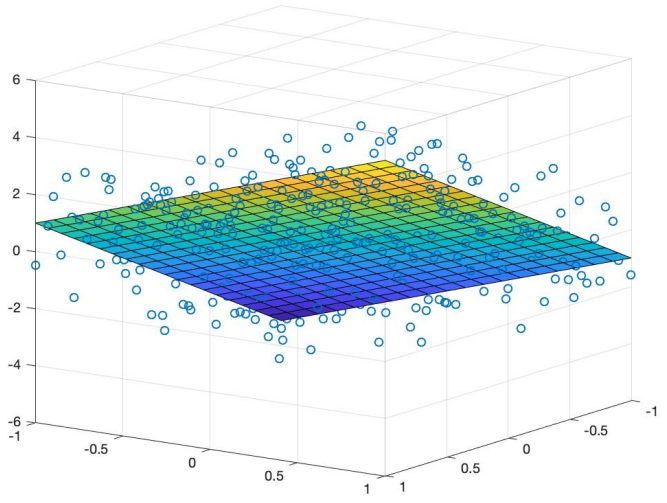


Figure 10.11: Two-dimensional regression.

based on second order statistics (means, variances, covariances), estimated from the training data. PCA is an unsupervised learning algorithm, as it ignores the labels in the data. It is simply trying to find low-dimensional approximations to the total data collected. PCA was invented in 1901 by Karl Pearson. It is also known as the Karhunen–Loève transform (KLT) in signal processing, and as factor analysis in statistics. PCA can be thought of as fitting a low-dimensional ellipsoid in a subspace to data. Each axis of the ellipsoid is a principal component. .

PCA has three main steps:

- Compute the sample covariance matrix of the full-dimensional data.
- Compute eigenvalues and eigenvectors of this covariance matrix.
- Select the eigenvectors associated with the largest k eigenvalues and transform your data into k-dimensional projections onto those eigenvectors.

We describe the mathematics of these steps below, illustrating how they can be computed. The first step is straightforward, and based on estimation results. Assume we are given data consisting of N independent samples of a random vector $\underline{X}$, denoted as $\underline{X}_k, k = 1, \ldots, n$. We assume that $\underline{X}$ takes values in $\Re^d$, where d can be a large number. In many applications where the data is an image, d is often larger than n. The sample mean of this data, which is an estimate of the true mean $\underline{\mu}_{\underline{X}}$, is computed as:

$$\widehat{\underline{\mu}}_{\underline{X}} = \frac{1}{n} \sum_{k=1}^{n} \underline{X}_K$$

Similarly, the max-likelihood estimate of the covariance matrix, which is a biased estimate of the true covariance $\Sigma_{\underline{X}}$, is computed as

$$\widehat{\Sigma}_{\underline{X}} = \frac{1}{n} \sum_{k=1}^{n} (\underline{X}_K - \widehat{\mu}_{\underline{X}})(\underline{X}_K - \widehat{\mu}_{\underline{X}})^T$$

Now that we have the sample covariance, let's discuss the motivation for the second step: Computing the eigenvalues and eigenvectors of this covariance matrix. To do this, we will solve an approximation problem.

Assume we wanted to project the data $\underline{X}_k$ onto a one-dimensional subspace, defined by the unit vector $\underline{v}_1$. We do this by computing the inner product between $\underline{v}_1$ and each of the $\underline{X}_k$. Define these projection as $Z_k = \underline{v}_1^T \underline{X}_k, k = 1, \ldots, n$, which are n independent samples of a scalar random variable $Z = \underline{v}_1^T \underline{X}$.

The sample mean of these projections Z_k is

$$\widehat{\mu}_Z = \frac{1}{n} \sum_{k=1}^{n} \underline{v_1}^T \underline{X}_k = \underline{v}_1^T \widehat{\underline{\mu}}_{\underline{X}}.$$

Similarly, the sample covariance of Z is given by the biased estimate

$$\begin{aligned}\widehat{\sigma}_Z^2 &= \frac{1}{n} \sum_{k=1}^{(} Z_k - \widehat{\mu}_Z)(Z_k - \widehat{\mu}_Z) = \frac{1}{n} \sum_{k=1}^{n} (\underline{v}_1^T \underline{X}_k - \underline{v}_1^T \widehat{\underline{\mu}}_{\underline{X}})(\underline{X}_K^T \underline{v}_1 - \widehat{\underline{\mu}}_{\underline{X}}^T \underline{v}_1) \\ &= \underline{v}_1^T \left(\frac{1}{n} \sum_{k=1}^{n} (\underline{X}_k - \widehat{\underline{\mu}}_{\underline{X}})(\underline{X}_k^T - \widehat{\underline{\mu}}_{\underline{X}}^T) \right) \underline{v}_1 = \underline{v}_1^T \left(\widehat{\underline{\Sigma}}_{\underline{X}} \right) \underline{v}_1\end{aligned}$$

In the original d-dimensional space, the approximation to each $\underline{X}_k$ is given by $\underline{V}_k = \widehat{\underline{\mu}}_{\underline{X}} + Z_k \underline{v}_1$. Ideally, we want to select the unit vector $\underline{v}_1$ to best approximate the data on this one-dimensional affine space, by minimizing the mean-square error, defined as $MSE = \frac{1}{N} \sum_{k=1}^{N} (\underline{X}_k - \underline{V}_k)^T (\underline{X}_k - \underline{Y}_k)$. Define $\tilde{\underline{X}}_k = \underline{X}_k - \widehat{\underline{\mu}}_{\underline{X}}$. Then, the mean square error can be stated in terms the scalars Z_k as

$$\begin{aligned}MSE &= \frac{1}{n} \sum_{k=1}^{n} (\tilde{\underline{X}}_k - Z_k \underline{v}_1)^T (\tilde{\underline{X}}_k - Z_k \underline{v}_1) = \frac{1}{n} \sum_{k=1}^{n} \left(\tilde{\underline{X}}_k^T \tilde{\underline{X}}_k - 2 \tilde{\underline{X}}_k^T \underline{v}_1 Z_k + \underline{v}_1^T \underline{v}_1 Z_k^2 \right) \\ &= \frac{1}{n} \sum_{k=1}^{n} \left(\tilde{\underline{X}}_k^T \tilde{\underline{X}}_k - 2 \tilde{\underline{X}}_k^T \underline{v}_1 Z_k + Z_k^2 \right)\end{aligned}$$

because Z_k is a scalar, and $\underline{v}_1$ is a unit vector. However, recall that $Z_k = \tilde{\underline{X}}_k^T \underline{v}_1$. Hence,

$$\begin{aligned}MSE &= \frac{1}{n} \sum_{k=1}^{n} \tilde{\underline{X}}_k^T \tilde{\underline{X}}_k - \frac{1}{n} \sum_{k=1}^{n} Z_k^2 = \frac{1}{n} \sum_{k=1}^{n} \tilde{\underline{X}}_k^T \tilde{\underline{X}}_k - \frac{1}{n} \sum_{k=1}^{n} \underline{v}_1^T \tilde{\underline{X}}_k \tilde{\underline{X}}_k^T \underline{v}_1 \\ &= \frac{1}{n} \sum_{k=1}^{n} \tilde{\underline{X}}_k^T \tilde{\underline{X}}_k - \underline{v}_1^T \widehat{\underline{\Sigma}}_{\underline{X}} \underline{v}_1 = \frac{1}{n} \sum_{k=1}^{N} \tilde{\underline{X}}_k^T \tilde{\underline{X}}_k - \widehat{\sigma}_Z^2\end{aligned}$$

Thus, to minimize the mean square error in the approximation, we need to select the direction $\underline{v}_1$ to maximize the covariance $\widehat{\sigma}_Z^2$. This results in the following optimization problem:

$$\max_{\underline{v}_1 : \|\underline{v}_1\|^2 = 1} \underline{v}_1^T \left(\widehat{\underline{\Sigma}}_{\underline{X}} \right) \underline{v}_1$$

This is a well-studied optimization problem. The sample covariance matrix is a positive-semidefinite matrix with all eigenvalues real and non-negative, and a full set of orthonormal eigenvectors. Ordering the eigenvalues in decreasing order, so $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$, we select $\underline{v}_1$ to be the eigenvector corresponding to the largest eigenvalue λ_1. This eigenvector is called the first principal component.

Note that $\widehat{\underline{\Sigma}}_{\underline{X}} \underline{v}_1 = \lambda_1 \underline{v}_1$, so the mean square error is reduced by λ_1. To obtain the next principal component, we find the unit vector that is orthogonal to $\underline{v}_1$ and maximizes the sample projection covariance $\underline{v}_2^T \left(\widehat{\underline{\Sigma}}_{\underline{X}} \right) \underline{v}_2$. The solution of this problem is the normalized eigenvector corresponding to λ_2, which is the second principal component. When using the first two principal components as approximations, the mean square error is reduced by $\lambda_1 + \lambda_2$.

We can continue this process until we get all the d eigenvectors $\underline{v}_1, \underline{v}_2, \ldots, \underline{v}_d$ of $\widehat{\underline{\Sigma}}_{\underline{X}}$. Let $\mathbf{V} = \begin{bmatrix} \underline{v}_1 & \underline{v}_2 & \cdots & \underline{v}_p \end{bmatrix}$. Since the eigenvectors are chosen to be orthogonal and normalized, we have the following relationships:

$$\widehat{\underline{\Sigma}}_{\underline{X}} \mathbf{V} = \mathbf{V} \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_p); \quad \mathbf{V}^T \mathbf{V} = \mathbf{I}_d$$

where $\mathbf{I}_d$ is the d-dimensional identity matrix. For dimensionality reduction, we choose the k largest principal components, to create a projection matrix $\mathbf{V}_k = \begin{bmatrix} \underline{v}_1 & \underline{v}_2 & \dots & \underline{v}_k \end{bmatrix}$.

We can write all of the above operations in matrix notation. Let's define a couple of matrices: an $N \times p$ data matrix $\mathbf{X}$ that stacks all the data as columns in a matrix, with each row representing one sample $\underline{X}_k^T$. Let's also define the $n \times 1$ vector of all ones as $\underline{\mathbf{1}}_n$. Then,

$$\mathbf{X} = \begin{bmatrix} \underline{X}_1^T \\ \underline{X}_2^T \\ \vdots \\ \underline{X}_n^T \end{bmatrix}; \qquad \underline{\mathbf{1}}_n = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

With this notation, the sample mean and covariance of $\underline{X}$ are easily computed as:

$$\widehat{\underline{\mu}}_{\underline{X}} = \frac{1}{n}\mathbf{X}^T\underline{\mathbf{1}}_n; \quad \widetilde{\mathbf{X}} = \mathbf{X} - \underline{\mathbf{1}}_n\widehat{\underline{\mu}}_{\underline{X}}^T; \quad \widehat{\Sigma}_{\underline{X}} = \frac{1}{n}\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}}$$

PCA then computes the eigenvector-eigenvalue decomposition of the sample covariance $\widehat{\Sigma}_{\underline{X}}$ as

$$\widehat{\Sigma}_{\underline{X}} = \mathbf{V}\text{diag}(\lambda_1, \dots, \lambda_d)\mathbf{V}^T = \begin{bmatrix} \underline{v}_1 & \dots & \underline{v}_d \end{bmatrix} \text{diag}(\lambda_1, \dots, \lambda_p) \begin{bmatrix} \underline{v}_1^T \\ \dots \\ \underline{v}_d^T \end{bmatrix}$$

Usually, this decomposition is computed using a singular value decomposition algorithm, but there are many other ways of computing this. The final step is to pick a projection matrix $\mathbf{V}_k$ of dimension $d \times k$ with the eigenvectors corresponding to the k largest eigenvalues, as $\mathbf{V}_k = \begin{bmatrix} \underline{v}_1 & \underline{v}_2 & \dots & \underline{v}_k \end{bmatrix}$. We use this matrix to project the data vectors $\underline{X}_j, j = 1, \dots, N$ to k-dimensional "feature" vectors, as

$$\mathbf{X}_{reduce} = (\mathbf{X} - \underline{\mathbf{1}}_n\widehat{\underline{\mu}}_{\underline{X}}^T)]\mathbf{V}_k$$

The matrix $\mathbf{X}_{reduce}$ is of dimension $n \times k$, and each row j is a k- dimensional feature vector corresponding to the original data d-dimensional sample $\underline{X}_j$.

How do we select k? One way is to look at the reduction in mean square error. If $k = d$, the resulting mean square error is zero, so we have reduced the mean square error by a fraction of 1. For smaller k, the fraction reduction in means square error is $\frac{\lambda_1 + \dots + \lambda_k}{\lambda_1 + \dots + \lambda_d}$. Selecting this fraction so we reduce the approximation error to under 1% usually yields a good value for k.

To represent the approximation in the original space, we can convert k-dimensional feature vectors to d-dimensional estimates $\widehat{\underline{X}}_j$ of the original data points using the following expression:

$$\widehat{\mathbf{X}} = \mathbf{X}_{reduce}\mathbf{V}_k^T + \underline{\mathbf{1}}_n\widehat{\underline{\mu}}_{\underline{X}}^T$$

In coordinates, let $\underline{Z}_j$ be the transpose of the j-th row of $\mathbf{X}_{reduce}$, which is k-dimensional feature vector approximation of the observation $\underline{X}_j$. Then,

$$\widehat{\underline{X}}_j = \mathbf{V}_k\underline{Z}_j + \widehat{\underline{\mu}}_{\underline{X}}.$$

Example 10.7

To illustrate how PCA works, we show how to approximate a 3-dimensional Gaussian with a 2-dimensional projection. The random vector $\underline{X}$ is assumed to be three-dimensional, with mean $\underline{\mu}_{\underline{X}} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ and covariance matrix

$$\Sigma_{\underline{X}} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}^T.$$

The following MATLAB script implements a 2-dimensional PCA approximation of X.

```
mu=[1; 2; 3];      %mean % Generate Gaussian points
A = [1 2 3; 3 1 2; 2 3 1];
Cov = A * A';

 [V,D]=eig(Cov);   %eigen-decomposition
% generate an nx3 matrix of samples points and plot them
n=300;
X=mvnrnd(transpose(mu),Cov,n);
scatter3(X(:,1),X(:,2),X(:,3)); xlabel('x'); ylabel('y'); zlabel('z');
sample_mu=transpose(X)*ones(n,1)/n;
[V_s,Coeff,d_s]=pca(X); hold on;
%plot eigenvectors
quiver3(sample_mu(1),sample_mu(2),sample_mu(3),V_s(1,1),V_s(2,1),V_s(3,1),15,'r');
quiver3(sample_mu(1),sample_mu(2),sample_mu(3),V_s(1,2),V_s(2,2),V_s(3,2),8,'g');
quiver3(sample_mu(1),sample_mu(2),sample_mu(3),V_s(1,3),V_s(2,3),V_s(3,3),8,'b');
% compute reduced 2D representation and plot these points
X_red=Coeff(:,1:2)*transpose(V_s(:,1:2))+ones(n,1)*transpose(sample_mu);
scatter3(X_red(:,1),X_red(:,2),X_red(:,3),'red');
%compute approximation error per element
MSerror=(norm(X-X_red,'fro'))^2/n
```

The results are shown in the figure below, where we plotted both the sample points and their approximations. The figure includes two 3-D views. The view on the right shows that the approximations all lie in a 2-dimensional plane. The resulting mean square error is 2.84 in the approximation.

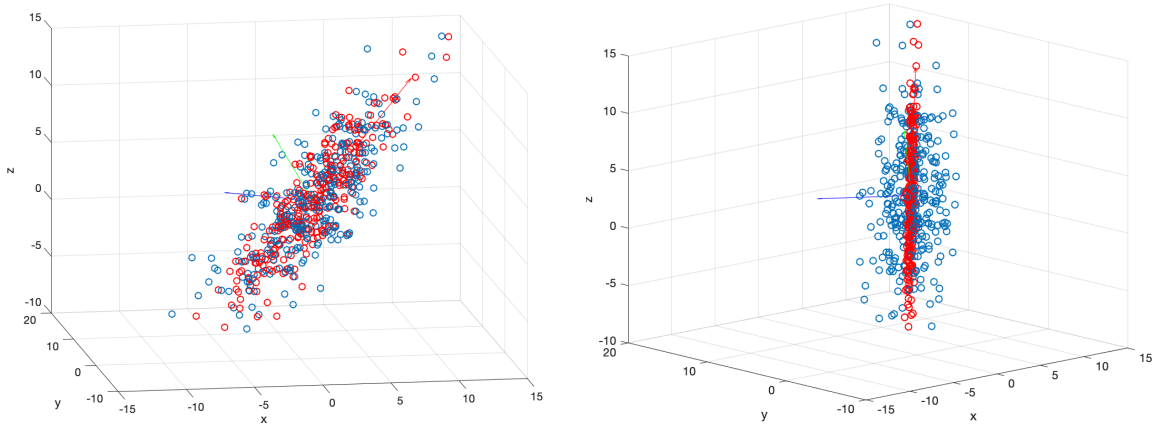


Figure 10.12: Two views of the approximation for Example 10.7.

Example 10.8
For our second example, we consider applying PCA to the IRIS data set, with two and three principal components. The results are shown in Figure 10.13 below. In this case, we cannot visualize the four-dimensional IRIS data and its approximation, so we show the two-dimensional features using the first two principal components, and the three-dimensional features using the first three principal components.

As the figures indicate, with two principal components, PCA captures 97.76% of the total variance in the data, and with three principal components, PCA captures 99.48% of the total variance in the data.

PCA is a powerful tool when combined with parametric classifiers such as LDA and QDA. By reducing the number of features in the data, one reduces the number of parameters which must be estimated in the likelihood functions, and can therefore generate better estimates with a smaller number of samples.

There are several important limitations of PCA. First, PCA ignores the labels in the data, and as such, may not find features that are best for classification. Instead, PCA find features that approximate the

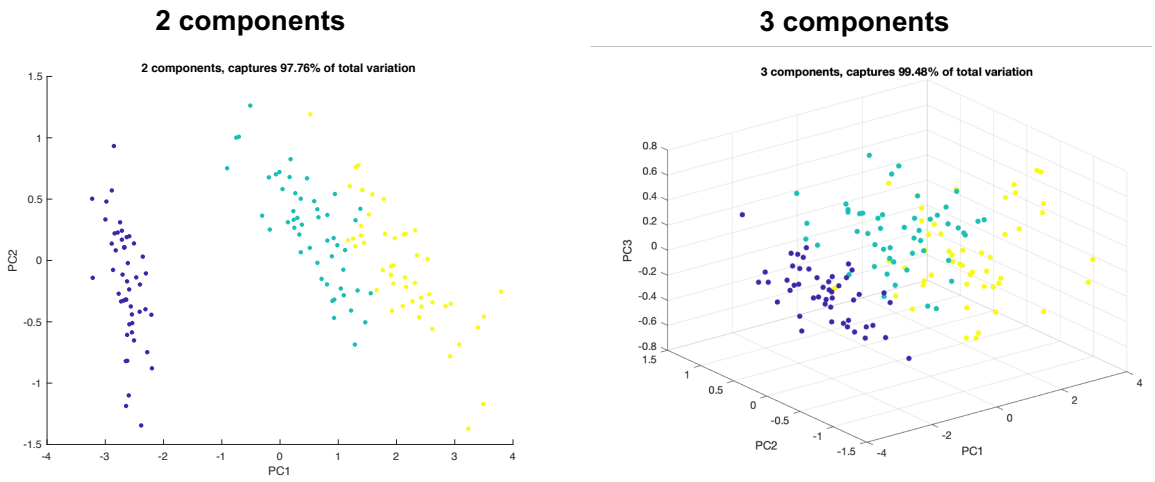


Figure 10.13: Two- and three-dimensional features for the IRIS data set.

data in lower dimensional spaces. Other dimensionality aggregation techniques such as Linear Discriminant Analysis are focused on supervised dimensionality reduction, and find features that separate the different hypotheses.

A second limitation of PCA is that the results depend on the scaling of the different variables. This is because the intrinsic distance used to approximate points is the Euclidean distance. Thus, in practice, one often normalizes each dimension of the data by an estimate of the standard deviation, so that all marginal distributions have variance 1.

The third limitation is that PCA is focused on finding features that are linear combinations of the data. In many feature sets, the best features may be nonlinear combinations. Extensions of PCA such as kernel PCA attempt to address this, by finding features using nonlinear kernels instead of inner product projections.

A common criticism of the features obtained by PCA is that they are linear combinations of all the dimensions d in the data, and hence do not provide insight into what the features mean. This is often expressed when PCA is used on medical data. Techniques such as sparse PCA and regularized PCA have been introduced to generate features that use only a fraction of the dimensions d.

In spite of the above limitations, PCA is broadly used as the first approach to dimensionality reduction because of its computational simplicity and its robust performance for high-dimensional data.

10.6 Summary

Machine learning addresses problems of classification and regression, in cases where we don't know the probabilistic relationship between observed values $\underline{X}$ and the labels or numeric values Y that we are trying to predict. Instead, we are provided with labeled training samples $(\underline{X}_1, Y_1), \ldots, (\underline{X}_N, Y_N)$ which form the basis for the design of classification and regression algorithms.

The algorithms discussed in this chapter comprise a small sample of the available techniques in the field. We have focused on algorithms where the learning is simple: there are no complex training procedures required to design the decision and estimation algorithms from training data. Thus, we avoided algorithms such as deep neural networks and support vector machines, where finding the parameters of the algorithm involve the solution of large-dimensional, complex optimization problems.