Perusal of PRML and Scikit-learn

Subway Chan

2018 年 10 月 22 日

目录

1 Introduction

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import norm
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge, BayesianRidge
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import mean_squared_error
import sys
sys.path.append("my-packages")
from bayesian_regressor import BayesianRegressor
from polynomial import PolynomialFeatures
pd.options.display.max_rows = 10
from tabulate import tabulate
tbl = lambda x: tabulate(x,headers="keys",tablefmt="orgtbl")
```

Listing 1.1: ch01-init

The problem of searching for patterns in data is a fundamental one and has a long and successful history. For instance, the extensive astronomical observations of Tycho Brahe in the 16th century allowed Johannes Kepler to discover the empirical laws of planetary motion, which in turn provided a springboard for the development of classical mechanics. Similarly, the discovery of regularities in atomic spectra played a key role in the development and verification of quantum physics in the early twentieth century. The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories.

Consider the example of recognizing handwritten digits, illustrated in Figure 1.1. Each digit corresponds to a 28×28 pixel image and so can be represented by a vector x comprising 784 real numbers. The goal is to build a machine that will take such a vector x as input and that will produce the

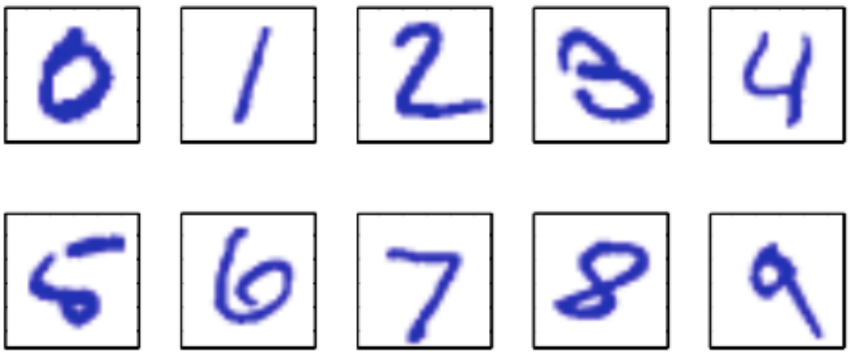


图 1.1: Examples of hand-written digits taken from US zip codes.

identity of the digit $0, \cdots, 9$ as the output. This is a nontrivial problem due to the wide variability of handwriting. It could be tackled(解决) using handcrafted rules or heuristics(启发法) for distinguishing the digits based on the shapes of the strokes, but in practice such an approach leads to a proliferation(增值) of rules and of exceptions(异常) to the rules and so on, and invariably(不约而同地) gives poor results.

Far better results can be obtained by adopting a machine learning approach in which a large set of N digits $x_1, \cdots, x_N$ called a *training set* is used to tune(调节) the parameters of an adaptive model. The categories of the digits in the *training set(训练集)* are known in advance, typically by inspecting them individually and hand-labeling them. We can express the category of a digit using *target vector(目标向量)* t, which represents the identity of the corresponding digit. Suitable techniques for representing categories in terms of vectors will be discussed later. Note that there is one such target vector t for each digit image x.

The result of running the machine learning algorithm can be expressed as a function $y(x)$ which takes a new digit image x as input and that generates an output vector y, encoded in the same way as the target vectors. The precise form of the function $y(x)$ is determined during the *training phase(训练阶段)*, also known as the *learning phase(学习阶段)*, on the basis of the

training data. Once the model is trained it can then determine the identity of new digit images, which are said to comprise(包含) a *test set(测试阶段)*. The ability to categorize correctly new examples that differ from those used for training is known as *generalization(泛化)*. In practical applications, the variability(变化性) of the input vectors will be such that the training data can comprise only a tiny fraction of all possible input vectors, and so **generalization is a central goal in pattern recognition**.

For most practical applications, the original input variables are typically *pre-processed(预处理)* to transform them into some new space of variables where, it is hoped, the pattern recognition problem will be easier to solve. For instance, in the digit recognition problem, the images of the digits are typically translated and scaled so that each digit is contained within a box of a fixed size. This greatly reduces the variability within each digit class, because the location and scale of all the digits are now the same, which makes it much easier for a subsequent(随后的) pattern recognition algorithm to distinguish between the different classes. This pre-processing stage is sometimes also called *feature extraction(特征提取)*. Note that new test data must be pre-processed using the same steps as the training data.

Pre-processing might also be performed in order to speed up computation. For example, if the goal is real-time face detection(检测) in a high-resolution(高分辨率) video stream, the computer must handle huge numbers of pixels(像素) per second, and presenting these directly to a complex pattern recognition algorithm may be computationally infeasible(不可行的). Instead, the aim is to find useful features that are fast to compute, and yet that also preserve useful discriminatory(有辨识力的) information enabling faces to be distinguished from non-faces. These features are then used as the inputs to the pattern recognition algorithm. For instance, the average value of the image intensity(图像灰度) over a rectangular subregion can be evaluated extremely efficiently (Viola and Jones, 2004), and a set of such features can prove very effective in fast face detection. Because the number of such

features is smaller than the number of pixels, this kind of pre-processing represents a form of dimensionality reduction(维数降低). Care must be taken during pre-processing because often information is discarded, and if this information is important to the solution of the problem then the overall accuracy of the system can suffer.

Applications in which the training data comprises examples of the input vectors along with their corresponding target vectors are known as *supervised learning(监督学习)* problems. Cases such as the digit recognition example, in which the aim is to assign each input vector to one of a finite number of discrete categories, are called *classification(分类)* problems. If the desired output consists of one or more continuous variables, then the task is called *regression(回归)*. An example of a regression problem would be the prediction of the yield in a chemical manufacturing process in which the inputs consist of the concentrations(浓度) of reactants(反应物), the temperature, and the pressure.

In other pattern recognition problems, the training data consists of a set of input vectors x without any corresponding target values. The goal in such *unsupervised learning(无监督学习)* problems may be to discover groups of similar examples within the data, where it is called *clustering(聚类)*, or to determine the distribution of data within the input space, known as *density estimation(密度估计)*, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of *visualization(数据可视化)*.

Finally, the technique of *reinforcement learning(反馈学习)* (Sutton and Barto, 1998) is concerned with the problem of finding suitable actions to take in a given situation in order to maximize a reward. Here the learning algorithm is not given examples of optimal outputs, in contrast to supervised learning, but must instead discover them by a process of trial and error. Typically there is a sequence of states and actions in which the learning algorithm is interacting(交互) with its environment. In many cases, the

current action not only affects the immediate reward but also has an impact on the reward at all subsequent time steps. For example, by using appropriate reinforcement learning techniques a neural network can learn to play the game of backgammon(西洋双陆棋) to a high standard (Tesauro, 1994). Here the network must learn to take a board position as input, along with the result of a dice throw, and produce a strong move as the output. This is done by having the network play against a copy of itself for perhaps a million games. A major challenge is that a game of backgammon can involve dozens of moves, and yet it is only at the end of the game that the reward, in the form of victory, is achieved. The reward must then be attributed appropriately to all of the moves that led to it, even though some moves will have been good ones and others less so. This is an example of a *credit assignment(信用分配)* problem. A general feature of reinforcement learning is the trade-off(权衡) between *exploration(探索)*, in which the system tries out new kinds of actions to see how effective they are, and *exploitation(利用)*, in which the system makes use of actions that are known to yield a high reward. Too strong a focus on either exploration or exploitation will yield poor results. Reinforcement learning continues to be an active area of machine learning research. However, a detailed treatment lies beyond the scope of this book.

Although each of these tasks needs its own tools and techniques, many of the key ideas that underpin(从下面支撑) them are common to all such problems. One of the main goals of this chapter is to introduce, in a relatively informal way, several of the most important of these concepts and to illustrate them using simple examples. Later in the book we shall see these same ideas re-emerge in the context of more sophisticated models that are applicable to real-world pattern recognition applications. This chapter also provides a self-contained introduction to three important tools that will be used throughout the book, namely **probability theory**, **decision theory**, and **information theory**. Although these might sound like daunting(令人

生畏的) topics, they are in fact straightforward, and a clear understanding of them is essential if machine learning techniques are to be used to best effect in practical applications.

1.1 Example: Polynomial Curve Fitting

We begin by introducing a simple regression problem, which we shall use as a running example throughout this chapter to motivate a number of key concepts. Suppose we observe a real-valued input variable x and we wish to use this observation to predict the value of a real-valued target variable t. For the present purposes, it is instructive(有启发性的) to consider an artificial example using synthetically(合成地, 人造地) generated data because we then know the precise process that generated the data for comparison against any learned model. The data for this example is generated from the function $sin(2x)$ with random noise included in the target values, as described in detail in Appendix A(?).

Now suppose that we are given a training set comprising N observations of x, written $x \equiv (x_1, \cdots, x_N)^T$, together with corresponding observations of the values of t, denoted $t \equiv (t_1, \cdots, t_N)^T$. Figure 1.2 shows a plot of a training set comprising $N = 10$ data points. The input data set x in Figure 1.2 was generated by choosing values of x_n, for $n = 1, \cdots, N$, spaced uniformly in range $[0, 1]$, and the target data set t was obtained by first computing the corresponding values of the function $sin(2x)$ and then adding a small level of random noise having a Gaussian distribution (the Gaussian distribution is discussed in Section 1.2.4(?)) to each such point in order to obtain the corresponding value t_n. By generating data in this way, we are capturing a property of many real data sets, namely that they possess an underlying regularity, which we wish to learn, but that individual observations are corrupted by random noise. This noise might arise from intrinsically stochastic (i.e. random) processes such as radioactive decay but more typically is due to there being sources of variability that are themselves

unobserved.

Our goal is to exploit this training set in order to make predictions of the value $\hat{t}$ of the target variable for some new value $\hat{x}$ of the input variable. As we shall see later, this involves implicitly trying to discover the underlying function $sin(2x)$. This is intrinsically(本质地) a difficult problem as we have to generalize from a finite data set. Furthermore the observed data are corrupted with noise, and so for a given x there is uncertainty as to the appropriate value for t. Probability theory, discussed in Section 1.2(?), provides a framework for expressing such uncertainty in a precise and quantitative manner, and decision theory, discussed in Section 1.5(?), allows us to exploit this probabilistic representation in order to make predictions that are optimal according to appropriate criteria.

```
def create_toy_data(func, sample_size=10, std=1):
    x = np.linspace(0, 1, sample_size)
    t = func(x) + np.random.normal(scale=std, size=x.shape)
    return x, t


def func(x):
    return np.sin(2 * np.pi * x)


std = 0.3
np.random.seed(1234)
data_train = pd.DataFrame(
    dict(zip(["x", "t"], create_toy_data(func, std=std, sample_size=10))))
data_test = pd.DataFrame(
    dict(zip(["x", "t"], create_toy_data(func, std=std, sample_size=100))))
data_plot = pd.DataFrame({"x": np.linspace(0, 1, 100)})
```

Listing 1.2: generate data

For the moment, however, we shall proceed rather informally and consider a simple approach based on curve fitting. In particular , we shall fit the data using a polynomial function of the form

```
plt.scatter(
    data_train["x"],
    data_train["t"],
    facecolor="none",
    edgecolor="b",
    s=50,
    label="training data")
plt.plot(data_plot["x"], data_plot.apply(func), c="g", label="$\sin(2\pi x)$")
plt.legend()
plt.savefig("img/fig:1.2.png")
plt.close("all")
```

Listing 1.3: fig:1.2

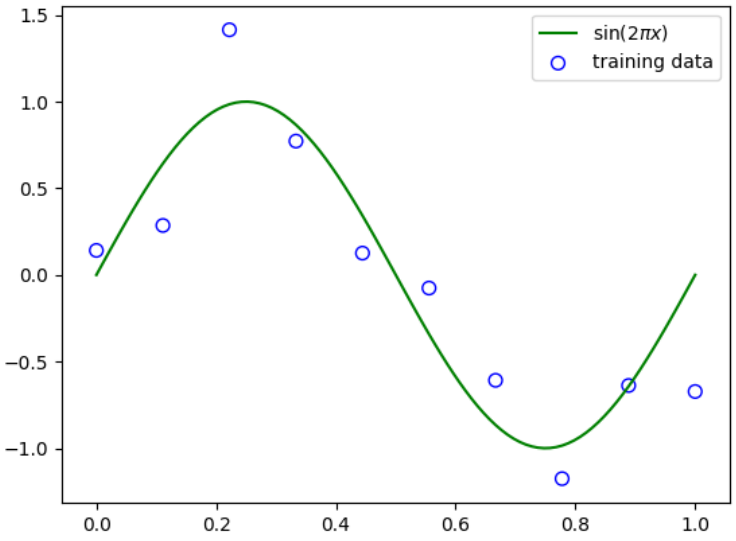


図 1.2: Plot of a training data set of $N = 10$ points, shown as blue circles, each comprising an observation of the input variable x along with the corresponding target variable t. The green curve shows the function $sin(2x)$ used to generate the data. Our goal is to predict the value of t for some new value of x, without knowledge of the green curve.

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j \tag{1.1}$$

where M is the *order(阶数)* of the polynomial, and x_j denotes x raised to the power of j. The polynomial coefficients $w_0, \cdots, w_M$ are collectively denoted by the vector $\mathbf{w}$. Note that, although the polynomial function $y(x, \mathbf{w})$ is a nonlinear function of x, it is a linear function of the coefficients $\mathbf{w}$. Functions, such as the polynomial, which are linear in the unknown parameters have important properties and are called linear models and will be discussed extensively in Chapters 3(?) and 4(?).

The values of the coefficients will be determined by fitting the polynomial to the training data. This can be done by minimizing an *error function(误差函数)* that measures the misfit between the function $y(x, \mathbf{w})$, for any given value of $\mathbf{w}$, and the training set data points. One simple choice of error function, which is widely used, is given by the sum of the squares of the errors between the predictions $y(x_n, \mathbf{w})$ for each data point x_n and the corresponding target values t_n, so that we minimize

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2 \tag{1.2}$$

where the factor of 1/2 is included for later convenience. We shall discuss the motivation for this choice of error function later in this chapter. For the moment we simply note that it is a nonnegative quantity that would be zero if, and only if, the function $y(x, \mathbf{w})$ were to pass exactly through each training data point. The geometrical interpretation(解释) of the sum-of-squares error function is illustrated in Figure 1.3.

We can solve the curve fitting problem by choosing the value of w for which $E(w)$ is as small as possible. Because the error function is a quadratic function of the coefficients w, its derivatives with respect to the coefficients will be linear in the elements of w, and so the minimization of the error

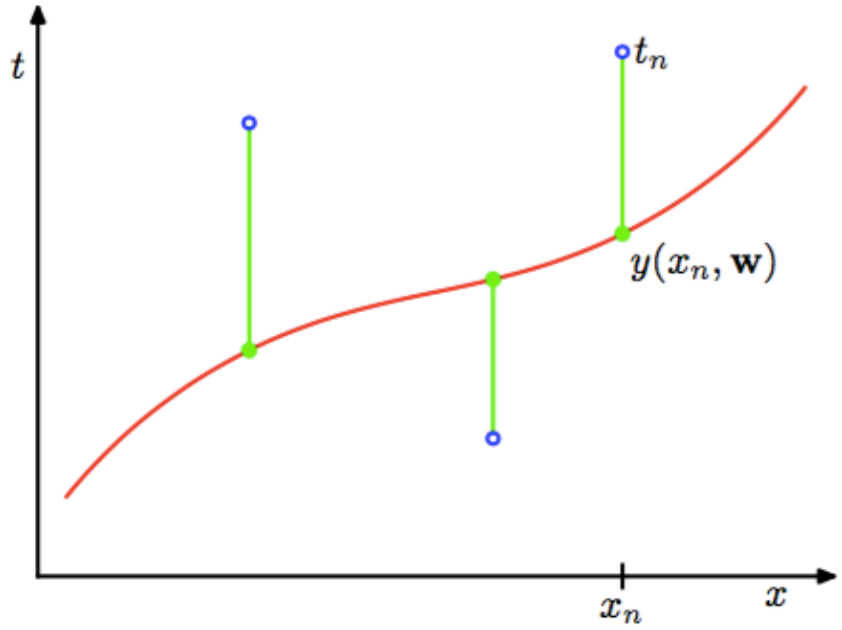


图 1.3: The error function (**??**) corresponds to (Mone half of) the sum of the squares of the displacements (shown by the vertical green bars) of each data point from the function $y(x, \mathbf{w})$.

function has a unique solution, denoted by $\mathbf{w}^*$, which can be found in closed form. The resulting polynomial is given by the function $y(x, \mathbf{w}^*)$.

There remains the problem of choosing the order M of the polynomial, and as we shall see this will turn out to be an example of an important concept called *model comparison(模型对比)* or *model selection(选择)*. In Figure 1.4, we show four examples of the results of fitting polynomials having orders $M = 0, 1, 3, 9$ to the data set shown in Figure 1.2.

```
for i, degree in enumerate([0, 1, 3, 9]):
    plt.subplot(2, 2, i + 1)
    feature = PolynomialFeatures(degree)
    X_train = feature.fit_transform(data_train["x"][:, None])
    X_plot = feature.fit_transform(data_plot["x"][:, None])
    model_train = LinearRegression(fit_intercept=False)
    model_train.fit(X_train, data_train["t"])
    data_plot["t"] = model_train.predict(X_plot)
    plt.scatter(
        data_train["x"],
        data_train["t"],
        facecolor="none",
        edgecolor="b",
        s=50,
        label="training data")
    plt.plot(
        data_plot["x"],
        data_plot["x"].apply(func),
        c="g",
        label="$\sin(2\pi x)$")
    plt.plot(data_plot["x"], data_plot["t"], c="r", label="fitting")
    plt.ylim(-1.5, 1.5)
    plt.annotate("M={}".format(degree), xy=(0.75, 1))
plt.subplots_adjust(right=0.75)
plt.legend(bbox_to_anchor=(1.05, 0.64), loc=2, borderaxespad=0.)
plt.savefig("img/fig:1.4.png")
plt.close("all")
```

Listing 1.4: fig:1.4

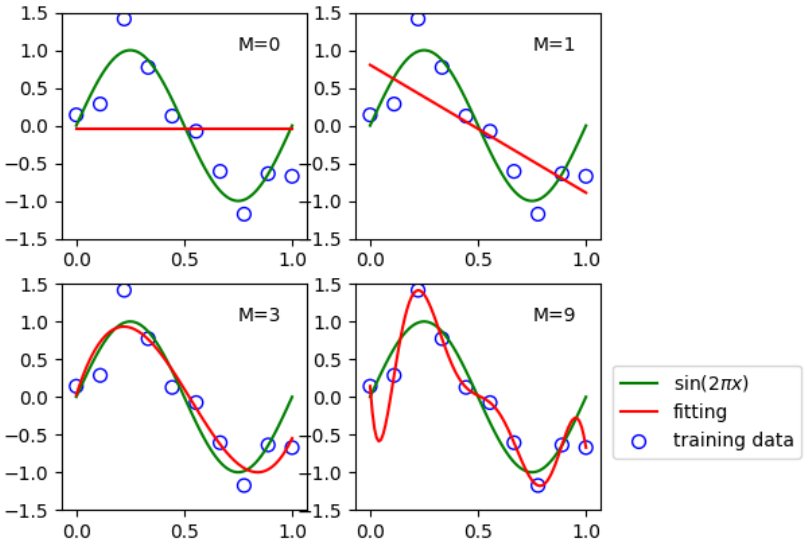


图 1.4: Plots of polynomials having various orders M, shown as red curves, fitted to the data set shown in Figure 1.2.

We notice that the constant ($M = 0$) and first order ($M = 1$) polynomials give rather poor fits to the data and consequently rather poor representations of the function $sin(2x)$. The third order ($M = 3$) polynomial seems to give the best fit to the function $sin(2x)$ of the examples shown in Figure 1.4. When we go to a much higher order polynomial ($M = 9$), we obtain an excellent fit to the training data. In fact, the polynomial passes exactly through each data point and $E(\mathbf{w}^*) = 0$. However, the fitted curve oscillates wildly and gives a very poor representation of the function $sin(2x)$. This latter behavior is known as *over-fitting(过拟合)*.

As we have noted earlier, the goal is to achieve good generalization by making accurate predictions for new data. We can obtain some quantitative insight into the dependence of the generalization performance on M by considering a separate test set comprising 100 data points generated using

exactly the same procedure used to generate the training set points but with new choices for the random noise values included in the target values. For each choice of M , we can then evaluate the residual value of $E(\mathbf{w}^*)$ given by (**??**) for the training data, and we can also evaluate $E(\mathbf{w}^*)$ for the test data set. It is sometimes more convenient to use the root-mean-square (RMS) error defined by

$$E_{RMS} = \sqrt{2E(\mathbf{w}^*)/N} \tag{1.3}$$

in which the division by N allows us to compare different sizes of data sets on an equal footing(基础), and the square root ensures that ERMS is measured on the same scale (and in the same units) as the target variable t. Graphs of the training and test set RMS errors are shown, for various values of M, in Figure 1.5. The test set error is a measure of how well we are doing in predicting the values of t for new data observations of x. We note from Figure 1.5 that small values of M give relatively large values of the test set error, and this can be attributed(归结于) to the fact that the corresponding **polynomials are rather inflexible and are incapable of capturing the oscillations(震荡) in the function** $sin(2x)$. Values of M in the range $3 \leqslant M \leqslant 8$ give small values for the test set error, and these also give reasonable representations of the generating function $sin(2x)$, as can be seen, for the case of $M = 3$, from Figure 1.4.

For $M = 9$, the training set error goes to zero, as we might expect because this polynomial contains 10 degrees of freedom corresponding to the 10 coefficients $w_0, \cdots, w_9$, and so can be tuned exactly to the 10 data points in the training set. However, the test set error has become very large and, as we saw in Figure 1.4, the corresponding function \$y(x, w)$exhibitswildoscillations$.

This may seem paradoxical because a polynomial of given order contains all lower order polynomials as special cases. The $M = 9$ polynomial is therefore capable of generating results at least as good as the $M = 3$

```
training_errors = []
test_errors = []
for degree in range(10):
    feature = PolynomialFeatures(degree)
    X_train = feature.fit_transform(data_train["x"][:, None])
    X_test = feature.transform(data_test["x"][:, None])
    model_train = LinearRegression(fit_intercept=False)
    model_train.fit(X_train, data_train["t"].values)
    data_train["y"] = model_train.predict(X_train)
    data_test["y"] = model_train.predict(X_test)
    training_errors.append(
        np.sqrt(mean_squared_error(data_train["y"], data_train["t"])))
    test_errors.append((np.sqrt(
        mean_squared_error(data_test["y"], data_test["t"]))))
plt.plot(
    training_errors, 'o-', mfc="none", mec="b", ms=10, c="b", label="Training")
plt.plot(test_errors, 'o-', mfc="none", mec="r", ms=10, c="r", label="Test")
plt.legend()
plt.xlabel("$M$")
plt.ylabel("$E_{RMS}$")
plt.ylim(0, 1)
plt.savefig("img/fig:1.5.png")
plt.close("all")
```

Listing 1.5: fig:1.5

```
mapping = {}
for degree in [0, 1, 3, 9]:
    feature = PolynomialFeatures(degree)
    X_train = feature.fit_transform(data_train["x"][:, None])
    model_train = LinearRegression(fit_intercept=False)
    model_train.fit(X_train, data_train["t"].values)
    mapping["$M=%d$" % degree] = pd.Series(model_train.coef_)
df = pd.DataFrame(mapping)
df.index = ["$w_%d^*$" % degree for degree in range(10)]
print(tbl(df.round(2).fillna("")))
```

Listing 1.6: tbl:1.1

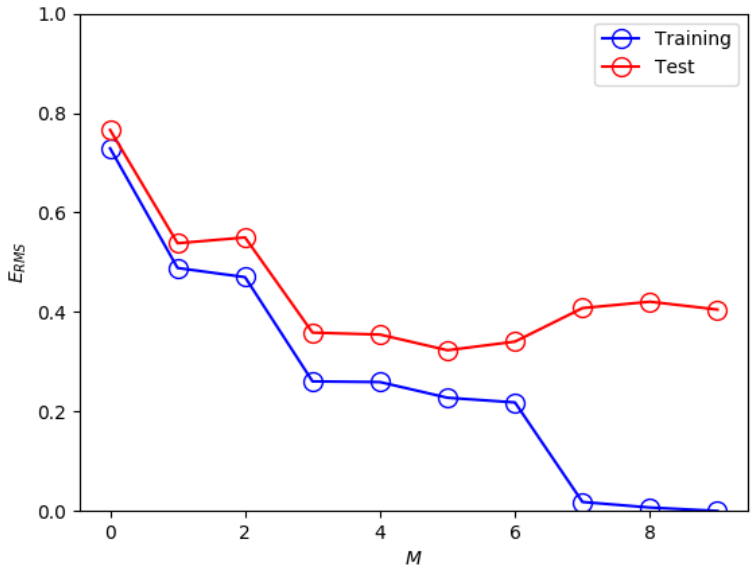


图 1.5: Graphs of the root-mean-square error, defined by (1.3), evaluated on the training set and on an independent test set for various values of M.

表 1.1: Table of the coefficients $\mathbf{w}^*$ for polynomials of various order. Observe how the typical magnitude of the coefficients increases dramatically as the order of the polynomial increases.

	$M=0$	$M=1$	$M=3$	$M=9$
w_0^*	-0.04	0.8	0.02	0.14
w_1^*		-1.7	9.02	-39.93
w_2^*			-25.82	663.06
w_3^*			16.23	-3265.66
w_4^*				5713.14
w_5^*				2429.63
w_6^*				-23252
w_7^*				34106.9
w_8^*				-21458.9
w_9^*				5103.07

polynomial. ($M = 9$ 的多项式因此能够产生至少与 $M = 3$ 一样好的结果。) Furthermore, we might suppose that the best predictor of new data would be the function sin(2x) from which the data was generated (and we shall see later that this is indeed the case). We know that a power series expansion of the function $sin(2x)$ contains terms of all orders, so we might expect that results should improve monotonically as we increase M.

We can gain some insight into the problem by examining the values of the coefficients $\mathbf{w}^*$ obtained from polynomials of various order, as shown in Table 1.1. We see that, as M increases, the magnitude of the coefficients typically gets larger. In particular for the $M = 9$ polynomial, the coefficients have become finely tuned to the data by developing large positive and negative values so that the corresponding polynomial function matches each of the data points exactly, but between data points (particularly near the ends of the range) the function exhibits the large oscillations observed in Figure 1.4. Intuitively(直觉地), what is happening is that the more flexible

polynomials with larger values of M are becoming increasingly tuned to the random noise on the target values. It is also interesting to examine the behavior of a given model as the size of the data set is varied, as shown in Figure 1.6. We see that, for a given model complexity, the over-fitting problem become less severe(严厉的) as the size of the data set increases. Another way to say this is that the larger the data set, the more complex (in other words more flexible) the model that we can afford to fit to the data. One rough(粗略的) heuristic that is sometimes advocated is that the number of data points should be no less than some multiple (say 5 or 10) of the number of adaptive parameters in the model. However, as we shall see in Chapter 3, the number of parameters is not necessarily the most appropriate measure of model complexity.

Also, there is something rather unsatisfying about having to limit the number of parameters in a model according to the size of the available training set. It would seem more reasonable to choose the complexity of the model according to the complexity of the problem being solved. We shall see that the least squares approach to finding the model parameters represents a specific case of *maximum likelihood(最大似然)* (discussed in Section 1.2.5(?)), and that the over-fitting problem can be understood as a general property of maximum likelihood. By adopting a *Bayesian* approach, the over-fitting problem can be avoided. We shall see that there is no difficulty from a Bayesian perspective in employing models for which the number of parameters greatly exceeds the number of data points. Indeed, in a Bayesian model the *effective(有效)* number of parameters adapts automatically to the size of the data set.

For the moment, however, it is instructive to continue with the current approach and to consider how in practice we can apply it to data sets of limited size where wemay wish to use relatively complex and flexible models. One technique that is often used to control the over-fitting phenomenon in such cases is that of *regularization(正则化)*, which involves(包含) adding a

```
for i, sample_size in enumerate([15, 100]):
    plt.subplot(1, 2, i + 1)
    feature = PolynomialFeatures(9)
    x_train_tmp, t_train_tmp = create_toy_data(func, sample_size, std=std)
    X_train_tmp = feature.fit_transform(x_train_tmp[:, None])
    model = LinearRegression(fit_intercept=False)
    model.fit(X_train_tmp, t_train_tmp)
    X_plot = feature.fit_transform(data_plot["x"][:, None])
    y_plot = model.predict(X_plot)
    plt.scatter(
        x_train_tmp,
        t_train_tmp,
        facecolor="none",
        edgecolor="b",
        s=50,
        label="training data")
    plt.plot(
        data_plot["x"],
        data_plot["x"].apply(func),
        c="g",
        label="$\sin(2\pi x)$")
    plt.plot(data_plot["x"], y_plot, c="r", label="fitting")
    plt.ylim(-1.5, 1.5)
    plt.annotate("N={}".format(sample_size), xy=(0.75, 1))
plt.savefig("img/fig:1.6.png")
plt.close("all")
```

Listing 1.7: fig:1.6

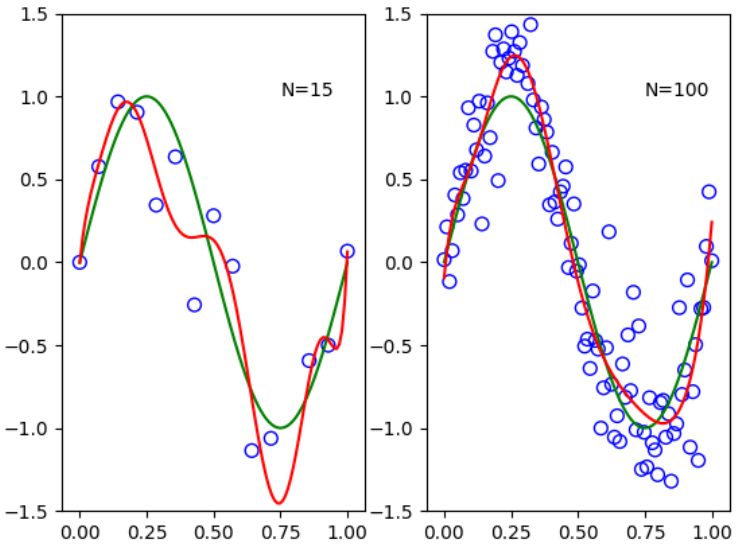


图 1.6: Plots of the solutions obtained by minimizing the sum-of-squares error function using the $M = 9$ polynomial for $N = 15$ data points (left plot) and $N = 100$ data points (right plot). We see that increasing the size of the data set reduces the over-fitting problem.

penalty term to the error function (**??**) in order to discourage the coefficients from reaching large values. The simplest such penalty term takes the form of a sum of squares of all of the coefficients, leading to a modified error function of the form

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w} - t_n)\}^2 + \frac{\lambda}{2} \parallel \mathbf{w} \parallel^2 \tag{1.4}$$

the coefficient λ governs the relative importance of the regularization term compared with the sum-of-squares error term. Note that often the coefficient w_0 is omitted(省略) from the regularizer because its inclusion causes the results to depend on the choice of origin for the target variable (Hastie et al., 2001), or it may be included but with its own regularization coefficient (we shall discuss this topic in more detail in Section 5.5.1(?)). Again, the error function in (**??**) can be minimized exactly in closed form. Techniques such as this are known in the statistics literature as *shrinkage(收缩)* methods because they reduce the value of the coefficients. The particular case of a quadratic regularizer is called *ridge regression(山脊回归)* (Hoerl and Kennard, 1970). In the context of neural networks, this approach is known as *weight decay(权值衰减)*.

Figure 1.7 shows the results of fitting the polynomial of order $M = 9$ to the same data set as before but now using the regularized error function given by (**??**). We see that, for a value of $\ln \lambda = -18$, the over-fitting has been suppressed(镇压) and we now obtain a much closer representation of the underlying function $sin(2x)$. If, however, we use too large a value for then we again obtain a poor fit, as shown in Figure 1.7 for $\ln \lambda = 0$. The corresponding coefficients from the fitted polynomials are given in Table 1.2, showing that regularization has the desired effect of reducing the magnitude of the coefficients.

The impact of the regularization term on the generalization error can be seen by plotting the value of the RMS error (1.3) for both training and test sets against $\ln \lambda$, as shown in Figure 1.8. We see that in effect λ now

```
for i, lamb in enumerate([-18, 0]):
    plt.subplot(1, 2, i + 1)
    feature = PolynomialFeatures(9)
    X_train = feature.fit_transform(data_train["x"][:, None])
    X_plot = feature.transform(data_plot["x"][:, None])
    model = Ridge(alpha=np.exp(lamb), fit_intercept=False)
    model.fit(X_train, data_train["t"])
    y_plot = model.predict(X_plot)
    plt.scatter(
        data_train["x"],
        data_train["t"],
        facecolor="none",
        edgecolor="b",
        s=50,
        label="training data")
    plt.plot(
        data_plot["x"],
        data_plot["x"].apply(func),
        c="g",
        label="$\sin(2\pi x)$")
    plt.plot(data_plot["x"], y_plot, c="r", label="fitting")
    plt.ylim(-1.5, 1.5)
    plt.annotate("$\ln\lambda = %d$" % lamb, xy=(0.6, 1))
    plt.annotate("M=9", xy=(-0.15, 1))
plt.savefig("img/fig:1.7.png")
plt.close("all")
```

Listing 1.8: fig:1.7

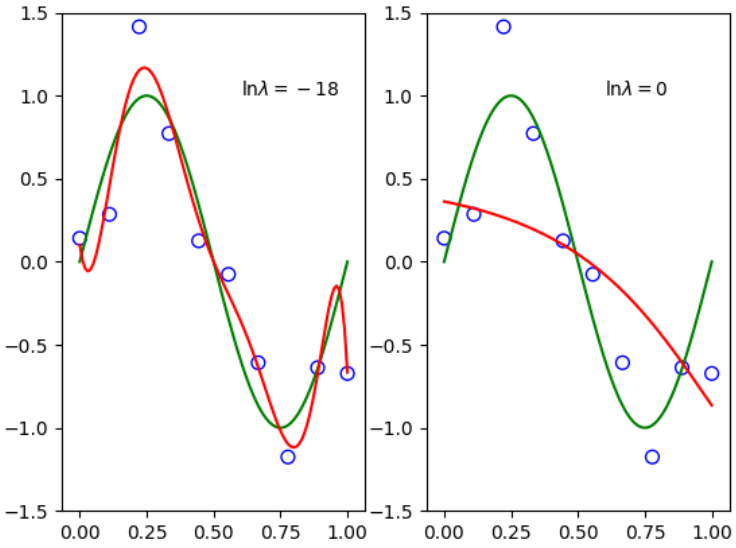


图 1.7: Plots of $M = 9$ polynomials fitted to the data set shown in Figure 1.2 using the regularized error function (**??**) for two values of the regularization parameter λ corresponding to $\ln \lambda = -18$ and $\ln \lambda = 0$. The case of no regularizer, i.e., $\lambda = 0$, corresponding to $\ln \lambda = -\infty$, is shown at the bottom right of Figure 1.4.

```
import pandas as pd
mapping = {}
infty = float("inf")
for index, _ in enumerate([-infty, -18, 0]):
    feature = PolynomialFeatures(9)
    X_train = feature.fit_transform(data_train["x"][:, None])
    alpha = np.exp(_)
    model_train = Ridge(alpha=alpha, fit_intercept=False)
    model_train.fit(X_train, data_train["t"].values)
    mapping["$\ln\lambda=%f$" % _] = model_train.coef_
df = pd.DataFrame(mapping)
df.index = ["$w_%d^*$" % _ for _ in range(10)]
print(tbl(df.round(2).fillna("")))
```

Listing 1.9: tbl:1.2

表 1.2: Table of the coefficients $\mathbf{w}^*$ for $M = 9$ polynomials with various values for the regularization parameter λ. Note that $\ln \lambda = -\infty$ corresponds to a model with no regularization, i.e., to the graph at the bottom right in Figure 1.4. We see that, as the value of λ increases, the typical magnitude of the coefficients gets smaller.

	$\ln \lambda = -18$	$\ln \lambda = -\infty$	$\ln \lambda = 0$
w_0^*	0.1	0.14	0.36
w_1^*	-11.09	-39.95	-0.32
w_2^*	221.99	663.39	-0.4
w_3^*	-1022.8	-3268.66	-0.31
w_4^*	1718.98	5727.18	-0.2
w_5^*	-513.64	2391.47	-0.11
w_6^*	-1250.07	-23189.7	-0.04
w_7^*	282.86	34046.6	0.01
w_8^*	1339.24	-21427.1	0.06
w_9^*	-766.24	5096.01	0.09

controls the effective complexity of the model and hence determines the degree of over-fitting.

```
training_errors = []
test_errors = []
for alpha in np.logspace(-40,20,100):
    feature = PolynomialFeatures(9)
    X_train = feature.fit_transform(data_train["x"][:,None])
    X_test = feature.fit_transform(data_test["x"][:,None])
    model = Ridge(alpha=alpha, fit_intercept=False)
    model.fit(X_train, data_train["t"])
    y_train = model.predict(X_train)
    y_test = model.predict(X_test)
    training_errors.append(np.sqrt(mean_squared_error(y_train, data_train["t"])))
    test_errors.append(np.sqrt(mean_squared_error(y_test, data_test["t"])))

plt.plot(np.linspace(-40,-20,100), training_errors, '-',
         mfc="none", mec="b", ms=10,
         c="b", label="Training")
plt.plot(np.linspace(-40,-20,100), test_errors, '-',
         mfc="none", mec="r", ms=10,
         c="r", label="Test")
plt.legend()
plt.xlabel("$\ln\lambda$")
plt.ylabel("$E_{RMS}$")
plt.ylim(0,1)
plt.savefig("img/fig:1.8.png")
plt.close("all")
```

Listing 1.10: fig:1,8

The issue of model complexity is an important one and will be discussed at length in Section 1.3(?). Here we simply note that, if we were trying to solve a practical application using this approach of minimizing an error function, we would have to find a way to determine a suitable value for the model complexity. The results above suggest a simple way of achieving this, namely by taking the available data and partitioning it into a training set,

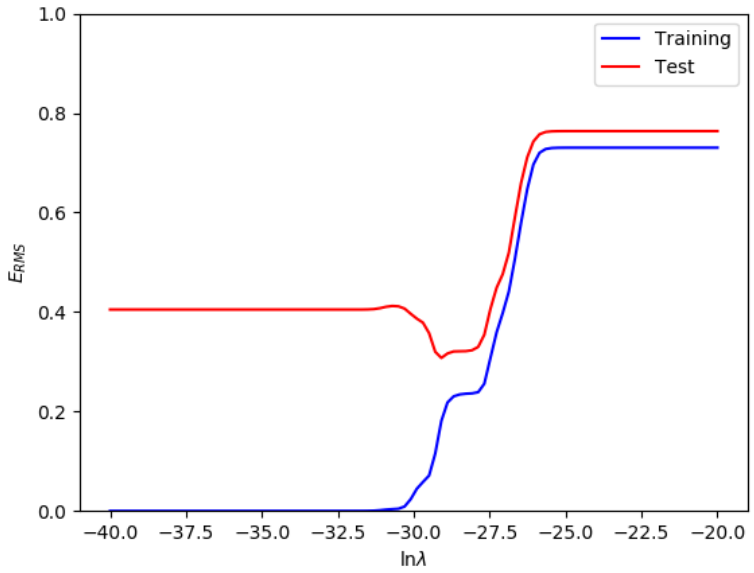


图 1.8: Graph of the root-mean-square error (1.3) versus $\ln \lambda$ for the $M = 9$ polynomial.

used to determine the coefficients w, and a separate *validation set(验证集)*, also called a *hold-out set(拿出集)*, used to optimize the model complexity (either M or λ). In many cases, however, this will prove to be too wasteful of valuable training data, and we have to seek more sophisticated approaches.

So far our discussion of polynomial curve fitting has appealed largely to intuition. We now seek a more principled approach to solving problems in pattern recognition by turning to a discussion of probability theory. As well as providing the foundation for nearly all of the subsequent developments in this book, it will also give us some important insights into the concepts we have introduced in the context of polynomial curve fitting and will allow us to extend these to more complex situations.

1.2 Probability Theory

A key concept in the field of pattern recognition is that of uncertainty. It arises both through noise on measurements, as well as through the finite size of data sets. Probability theory provides a consistent framework for the quantification and manipulation of uncertainty and forms one of the central foundations for pattern recognition. When combined with decision theory, discussed in Section 1.5, it allows us to make optimal predictions given all the information available to us, even though that information may be incomplete or ambiguous.

We will introduce the basic concepts of probability theory by considering a simple example. Imagine we have two boxes, one red and one blue, and in the red box we have 2 apples and 6 oranges, and in the blue box we have 3 apples and 1 orange. This is illustrated in Figure fig 1.9. Now suppose we randomly pick one of the boxes and from that box we randomly select an item of fruit, and having observed which sort of fruit it is we replace it in the box from which it came. We could imagine repeating this process many times. Let us suppose that in so doing we pick the red box 40% of the time and we pick the blue box 60% of the time, and that when we remove an

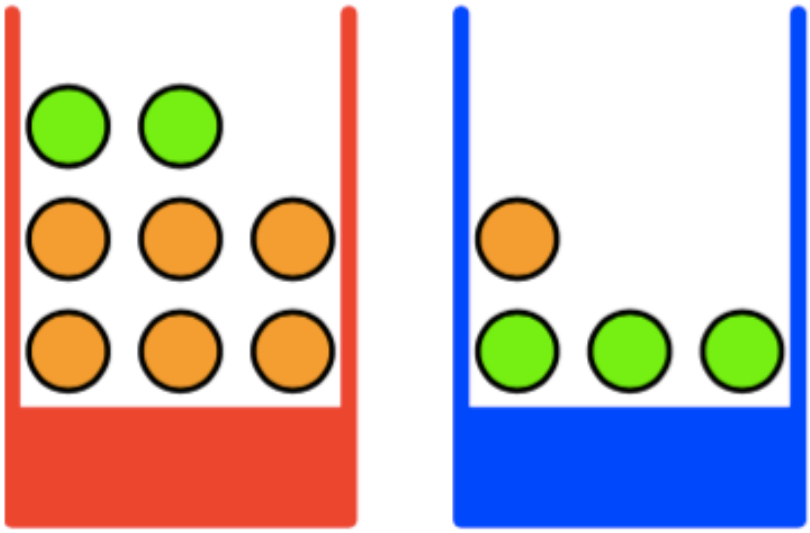


图 1.9: We use a simple example of two coloured boxes each containing fruit (apples shown in green and oranges shown in orange) to introduce the basic ideas of probability.

item of fruit from a box we are equally likely to select any of the pieces of fruit in the box.

In this example, the identity of the box that will be chosen is a random variable, which we shall denote by B. This random variable can take one of two possible values, namely r (corresponding to the red box) or b (corresponding to the blue box). Similarly, the identity of the fruit is also a random variable and will be denoted by F . It can take either of the values a (for apple) or o (for orange). To begin with, we shall define the probability of an event to be the fraction of times that event occurs out of the total number of trials, in the limit that the total number of trials goes to infinity. Thus the probability of selecting the red box is 4/10 and the probability of selecting the blue box is 6/10. We write these probabilities as $p(B = r) = 4/10$ and $p(B = b) = 6/10$. Note that, by definition, probabilities must lie in the interval $[0, 1]$. Also, if the events are mutually exclusive and if they include all possible outcomes (for instance, in this example the box must be either red or blue), then we see that the probabilities for those events must sum to one.

We can now ask questions such as: "what is the overall probability that the selection procedure will pick an apple?", or "given that we have chosen an orange, what is the probability that the box we chose was the blue one?" . We can answer questions such as these, and indeed much more complex questions associated with problems in pattern recognition, once we have equipped ourselves with the two elementary rules of probability, known as the *sum rule(加和规则)* and the *product rule(乘积规则)*. Having obtained these rules, we shall then return to our boxes of fruit example.

In order to derive the rules of probability, consider the slightly more general ex- ample shown in Figure 1.10 involving two random variables X and Y (which could for instance be the Box and Fruit variables considered above). We shall suppose that X can take any of the values x_i where $i = 1, \cdots, M$, and Y can take the values y_j where $j = 1, \cdots, L$. Consider a

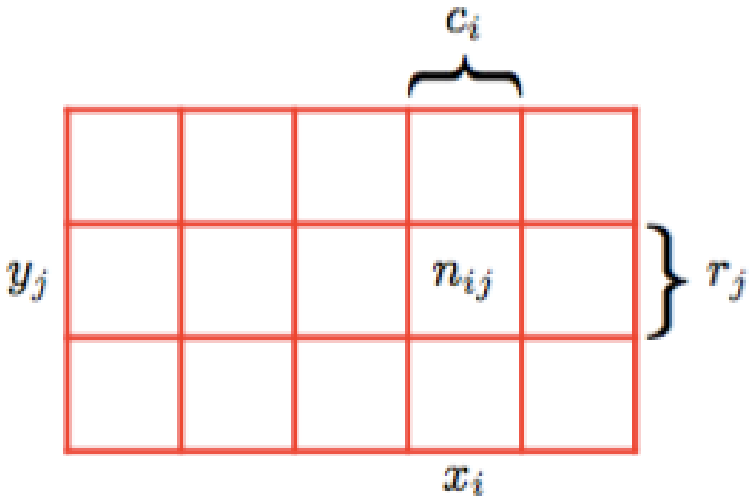


图 1.10: We can derive the sum and product rules of probability by considering two random variables, X, which takes the values xi where $i = 1, \cdots, M$, and Y, which takes the values yj where $j = 1, \cdots, L$. In this illustration we have M = 5 and L = 3. If we consider a total number N of instances of these variables, then we denote the number of instances where $X = xi$ and $Y = yj$ by n_{ij}, which is the number of y_j points in the corresponding cell of the array. The number of points in column i, corresponding to $X = x_i$, is denoted by c_i, and the number of points in row j, corresponding to $Y = y_j$, is denoted by r_j.

total of N trials in which we sample both of the variables X and Y, and let the number of such trials in which $X = x_i$ and $Y = y_j$ be n_{ij}. Also, let the number of trials in which X takes the value x_i (irrespective of the value that Y takes) be denoted by c_i, and similarly let the number of trials in which Y takes the value y_j be denoted by r_j . The probability that X will take the value x_i and Y will take the value y_j is written $p(X = x_i, Y = y_j)$ and is called the *joint probability(联合概率)* of $X = x_i$ and $Y = y_j$. It is given by the number of points falling in the cell i, j as a fraction of the total number of points, and hence

$$p(X = x_i, Y = y_i) = \frac{n_{ij}}{N} \tag{1.5}$$

Here we are implicitly considering the limit $N \to \infty$. Similarly, the probability that X takes the value x_i irrespective of the value of Y is written as $p(X = x_i)$ and is given by the fraction of the total number of points that fall in column i, so that

$$p(X = x_i) = \frac{ci}{N}. \tag{1.6}$$

Because the number of instances in column i in Figure 1.10 is just the sum of the number of instances in each cell of that column, we have $c_i = \sum_j n_{ij}$ and therefore, from (1.5) and (1.6), we have

$$p(X = x_i) = \sum_{j=1}^{L} p(X = x_i, Y = y_j) \tag{1.7}$$

which is the *sum rule(加和规则)* of probability. Note that $p(X = xi)$ is sometimes called the *marginal probability(边缘概率)*, because it is obtained by marginalizing, or summing out, the other variables (in this case Y).

If we consider only those instances for which $X = x_i$, then the fraction of such instances for which $Y = y_j$ is written $p(Y = y_j|X = x_i)$ and is called the *conditional probability(条件概率)* of $Y = y_j$ given $X = x_i$. It is obtained

by finding the fraction of those points in column i that fall in cell i, j and hence is given by

$$P(Y = y_j|X = x_i) = \frac{n_i j}{c_i} \tag{1.8}$$

From (1.5), (1.6), and (1.8), we can then derive the following relationship

$$\begin{aligned} p(X = x_i, Y = y_j) &= \frac{nij}{N} = \frac{n_{ij}}{c_i} \cdot \frac{c_i}{N} \\ &= p(Y = y_j|X = x_i)p(X = x_i) \end{aligned} \tag{1.9}$$

which is the *product rule(乘积规则)* of probability.

So far we have been quite careful to make a distinction between a random variable, such as the box B in the fruit example, and the values that the random variable can take, for example r if the box were the red one. Thus the probability that B takes the value r is denoted $p(B = r)$. Although this helps to avoid ambiguity, it leads to a rather cumbersome notation, and in many cases there will be no need for such pedantry(迂腐的). Instead, we may simply write $p(B)$ to denote a distribution over the random variable B, or $p(r)$ to denote the distribution evaluated for the particular value r, provided that the interpretation is clear from the context.

With this more compact notation, we can write the two fundamental rules of probability theory in the following form:

$$\textbf{sum rule} \quad p(X) = \sum_Y p(X, Y) \tag{1.10}$$

$$\textbf{product rule} \quad p(X, Y) = p(Y|X)p(X) \tag{1.11}$$

Here $p(X, Y)$ is a joint probability and is verbalized(描述) as "the probability of X and Y ". Similarly, the quantity $p(Y|X)$ is a conditional probability and is verbalized as "the probability of Y given X", whereas the quantity

p(X) is a marginal probability and is simply "the probability of X". These two simple rules form the basis for all of the probabilistic machinery that we use throughout this book.

From the product rule, together with the symmetry property $p(X, Y) = p(Y, X), we immediately obtain the following relationship between conditional probabilities

$$p(Y|X) = \frac{P(X|Y)p(Y)}{P(X)} \tag{1.12}$$

which is called *Bayes'theorem(贝叶斯定理)* and which plays a central role in pattern recognition and machine learning. Using the sum rule, the denominator in Bayes'theorem can be expressed in terms of the quantities appearing in the numerator

$$p(X) = \sum_{Y} p(X|Y)p(Y). \tag{1.13}$$

We can view the denominator in Bayes'theorem as being the normalization constant required to ensure that the sum of the conditional probability on the left-hand side of (1.12) over all values of Y equals one.

In Figure 1.11, we show a simple example involving a joint distribution over two variables to illustrate the concept of marginal and conditional distributions. Here a finite sample of $N = 60$ data points has been drawn from the joint distribution and is shown in the top left. In the top right is a histogram of the fractions of data points having each of the two values of Y . From the definition of probability, these fractions would equal the corresponding probabilities $p(Y)$ in the limit $N \to \infty$. We can view the histogram as a simple way to model a probability distribution given only a finite number of points drawn from that distribution. **Modeling distributions from data lies at the heart of statistical pattern recognition** and will be explored in great detail in this book. The remaining two plots in Figure 1.11 show the corresponding histogram estimates of $p(X)$ and $p(X|Y = 1)$.

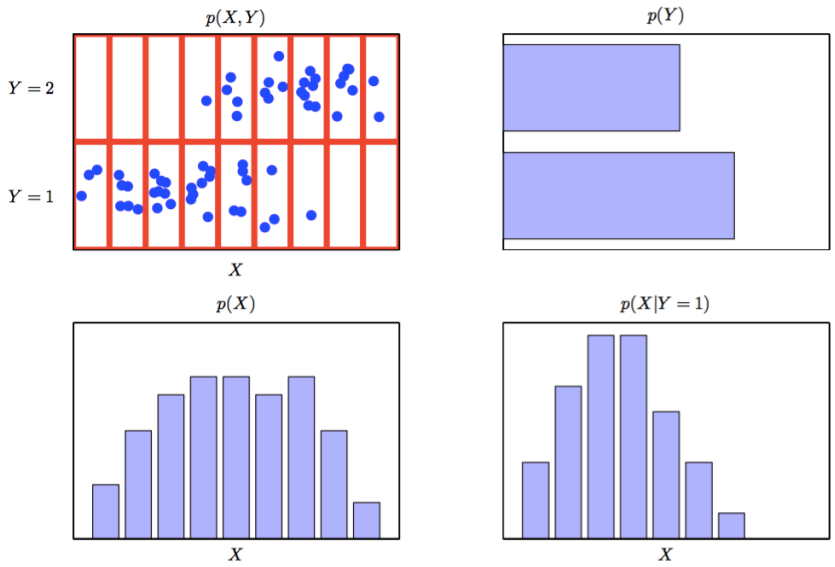


图 1.11: An illustration of a distribution over two variables, X, which takes 9 possible values, and Y , which takes two possible values. The top left figure shows a sample of 60 points drawn from a joint probability distribution over these variables. The remaining figures show histogram estimates of the marginal distributions $p(X)$ and $p(Y)$, as well as the conditional distribution $p(X|Y = 1)$ corresponding to the bottom row in the top left figure.

We can provide an important interpretation of Bayes'theorem as follows. If we had been asked which box had been chosen before being told the identity of the selected item of fruit, then the most complete information we have available is provided by the probability $p(B)$. We call this the *prior probability(先验概率)* because it is the probability available before we observe the identity of the fruit. Once we are told that the fruit is an orange, we can then use Bayes'theorem to compute the probability $p(B|F)$, which we shall call the *posterior probability(后验概率)* because it is the probability obtained after we have observed F . Note that in this example, the prior probability of selecting the red box was 4/10, so that we were more likely to select the blue box than the red one. However, once we have observed that the piece of selected fruit is an orange, we find that the posterior probability of the red box is now 2/3, so that it is now more likely that the box we selected was in fact the red one. This result accords with our intuition, as the proportion of oranges is much higher in the red box than it is in the blue box, and so the observation that the fruit was an orange provides significant evidence favoring the red box. In fact, the evidence is sufficiently strong that it outweighs the prior and makes it more likely that the red box was chosen rather than the blue one.

Finally, we note that if the joint distribution of two variables factorizes into the product of the marginals, so that $p(X, Y) = p(X)p(Y)$, then X and Y are said to be independent. From the product rule, we see that $p(Y|X) = p(Y)$, and so the conditional distribution of Y given X is indeed *independent(相互独立)* of the value of X. For instance, in our boxes of fruit example, if each box contained the same fraction of apples and oranges, then $p(F|B) = P(F)$, so that the probability of selecting, say, an apple is independent of which box is chosen.

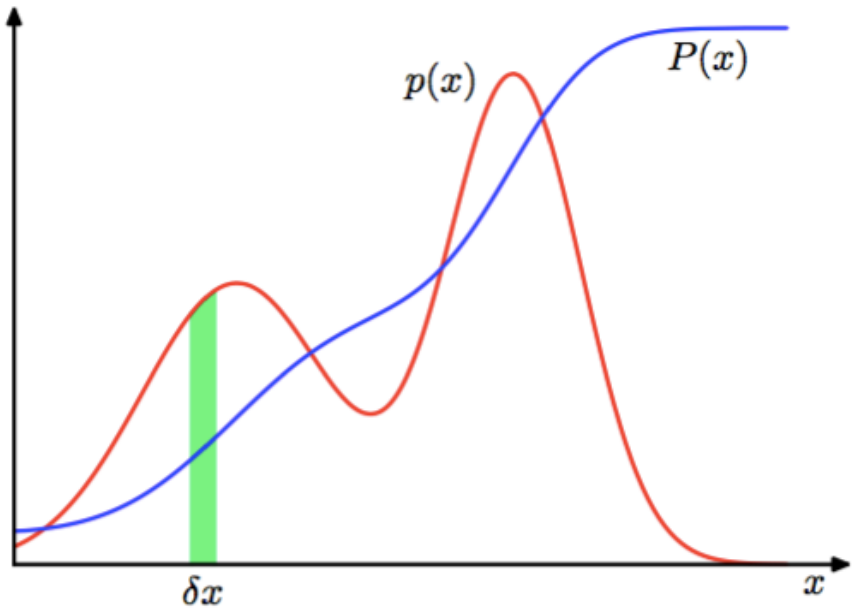


图 1.12: The concept of probability for discrete variables can be extended to that of a probability density $p(x)$ over a continuous variable x and is such that the probability of x lying in the interval $(x, x + \delta x)$ is given by $p(x)\delta x$ for $\delta x \to 0$. The probability density can be expressed as the derivative of a cumulative distribution function $P(x)$.

1.2.1 Probability densities

As well as considering probabilities defined over discrete sets of events, we also wish to consider probabilities with respect to continuous variables. We shall limit ourselves to a relatively informal discussion. If the probability of a real-valued variable x falling in the interval $(x, x+\delta x)$ is given by $p(x)\delta x$ for $\delta x \to 0$, then $p(x)$ is called the *probability density(*概率密度*)* over x. This is illustrated in Figure 1.12. The probability that x will lie in an interval (a, b) is then given by

$$p(x \in (a, b)) = \int_a^b p(x)dx \tag{1.14}$$

Because probabilities are nonnegative, and because the value of x must lie somewhere on the real axis, the probability density $p(x)$ must satisfy the two conditions

$$p(x) \geqslant 0 \tag{1.15}$$

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{1.16}$$

Under a nonlinear change of variable, a probability density transforms differently from a simple function, due to the Jacobian factor. For instance, if we consider a change of variables $x = g(y)$, then a function $f(x)$ becomes $\tilde{f}(y) = f(g(y))$. Now consider a probability density $p_x(x)$ that corresponds to a density $p_y(y)$ with respect to the new variable y, where the suffices denote the fact that $p_x(x)$ and $p_y(y)$ are different densities. Observations falling in the range $(x, x + \delta x)$ will, for small values of δx, be transformed into the range $(y, y + \delta y)$ where $p_x(x)\delta x \simeq py(y)\delta y$, and hence

$$\begin{aligned} p_y(y) &= p_x(x)\left|\frac{dx}{dy}\right| \\ &= p_x(g(y))|g'(y)|. \end{aligned} \tag{1.17}$$

One consequence of this property is that the concept of the **maximum of a probability density is dependent on the choice of variable**.

The probability that x lies in the interval $(-\infty, z)$ is given by the *cumulative distribution function(累积分布函数)* defined by

$$P(z) = \int_{-\infty}^{z} p(x)dx \tag{1.18}$$

which satisfies $P(x) = p(x)$, as shown in Figure 1.12.

If we have several continuous variables $x1, \cdots, x_D$, denoted collectively by the vector $\mathbf{x}$, then we can define a joint probability density $p(\mathbf{x}) = p(x_1, \cdots, x_D)$ such that the probability of $\mathbf{x}$ falling in an infinitesimal volume δx containing the point \$x\$is given by $p(\mathbf{x})\delta\mathbf{x}$. This multivariate probability density must satisfy

$$p(\mathbf{x}) \geqslant 0 \tag{1.19}$$

$$\int p(\mathbf{x})d\mathbf{x} = 1 \tag{1.20}$$

in which the integral is taken over the whole of $\mathbf{x}$ space. We can also consider joint probability distributions over a combination of discrete and continuous variables.

Note that if x is a discrete variable, then $p(x)$ is sometimes called a *probability mass function(概率质量函数)* because it can be regarded as a set of 'probability masses'concentrated at the allowed values of x.

The sum and product rules of probability, as well as Bayes'theorem, apply equally to the case of probability densities, or to combinations of discrete and continuous variables. For instance, if x and y are two real variables, then the sum and product rules take the form

$$p(x) = \int p(x, y)dy \tag{1.21}$$

$$p(x, y) = p(y|x)p(x) \tag{1.22}$$

A formal justification of the sum and product rules for continuous variables (Feller, 1966) requires a branch of mathematics called measure theory and lies outside the scope of this book. Its validity can be seen informally, however, by dividing each real variable into intervals of width Δ and considering the discrete probability distribution over these intervals. Taking the limit $\$\Delta \rightarrow 0$ \$ then turns sums into integrals and gives the desired result.

1.2.2 Expectations and covariances

One of the most important operations involving probabilities is that of finding weighted averages of functions. The average value of some function $f(x)$ under a probability distribution $p(x)$ is called the *expectation(期望)* of $f(x)$ and will be denoted by $[f]$. For a discrete distribution, it is given by

$$\mathbb{E}[f] = \sum_x p(x)f(x) \tag{1.23}$$

so that the average is weighted by the relative probabilities of the different values of x. In the case of continuous variables, expectations are expressed in terms of an integration with respect to the corresponding probability density

$$\mathbb{E}[f] = \int p(x)f(x)dx \tag{1.24}$$

In either case, if we are given a finite number N of points drawn from the probability distribution or probability density, then the expectation can be approximated as a finite sum over these points

$$\mathbb{E}[f] \simeq \frac{1}{N}\sum_{n=1}^{N} f(x_n) \tag{1.25}$$

We shall make extensive use of this result when we discuss sampling methods in Chapter 11(?). The approximation in (1.25) becomes exact in the limit $\$ \mathrm{N} \rightarrow \infty \$$.

Sometimes we will be considering expectations of functions of several variables, in which case we can use a subscript to indicate which variable is being averaged over, so that for instance

$$\mathbb{E}_x[f(x,y)] \tag{1.26}$$

denotes the average of the function $f(x,y)$ with respect to the distribution of x. Note that $\mathbb{E}_x[f(x,y)]$ will be a function of y.

We can also consider a conditional expectation with respect to a conditional distribution, so that

$$\mathbb{E}_x[f|y] = \sum_x p(x|y)f(x) \tag{1.27}$$

with an analogous definition for continuous variables.

The variance of $f(x)$ is defined by

$$var[f] = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2] \tag{1.28}$$

and provides a measure of how much variability there is in $f(x)$ around its mean value $\mathbb{E}[f(x)]$. Expanding out the square, we see that the variance can also be written in terms of the expectations of $f(x)$ and $f(x)^2$

$$var[f] = \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2 \tag{1.29}$$

In particular, we can consider the variance of the variable x itself, which is given by

$$var[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 \tag{1.30}$$

For two random variables x and y, the *covariance(协方差)* is defined by

$$\begin{aligned} \mathrm{cov}[x,y] &= \mathbb{E}_{x,y}[\{x - \mathbb{E}[x]\}\{y - \mathbb{E}[y]\}] \\ &= \mathbb{E}_{x,y}[xy] - \mathbb{E}[x]\mathbb{E}[y] \end{aligned} \tag{1.31}$$

which expresses the extent to which x and y vary together. If x and y are independent, then their covariance vanishes.

In the case of two vectors of random variables $\mathbf{x}$ and $\mathbf{y}$, the covariance is a matrix

$$\begin{aligned} \mathrm{cov}[\mathbf{x}, \mathbf{y}] &= \mathbb{E}_{\mathbf{x},\mathbf{y}}[\{\mathbf{x} - \mathbb{E}[\mathbf{x}]\}\{\mathbf{y} - \mathbb{E}[\mathbf{y}]\}] \\ &= \mathbb{E}_{\mathbf{x},\mathbf{y}}[\mathbf{x}\mathbf{y}^T] - \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{y}^T] \end{aligned} \tag{1.32}$$

If we consider the covariance of the components of a vector $\mathbf{x}$ with each other, then we use a slightly simpler notation $\mathrm{cov}[\mathbf{x}] \equiv \mathrm{cov}[\mathbf{x}, \mathbf{x}]$.

1.2.3 Bayesian probabilities

So far in this chapter, we have viewed probabilities in terms of the frequencies of random, repeatable events. We shall refer to this as the classical or frequentist interpretation of probability. Now we turn to the more general Bayesian view, in which probabilities provide a quantification of uncertainty.

Consider an uncertain event, for example whether the moon was once in its own orbit around the sun, or whether the Arctic ice cap will have disappeared by the end of the century. These are not events that can be repeated numerous times in order to define a notion of probability as we did earlier in the context of boxes of fruit. Nevertheless, we will generally have some idea, for example, of how quickly we think the polar ice is melting. If we now obtain fresh evidence, for instance from a new Earth observation satellite gathering novel forms of diagnostic information, we may revise(修正) our opinion on the rate of ice loss. Our assessment of such matters will affect the actions we take, for instance the extent to which we endeavor(努力) to reduce the emission(发射) of greenhouse gases. In such circumstances(情况), we would like to be able to quantify our expression of uncertainty and make precise revisions of uncertainty in the light of new evidence, as well as subsequently to be able to take optimal actions or decisions as a consequence.

This can all be achieved through the elegant, and very general, Bayesian interpretation of probability.

The use of probability to represent uncertainty, however, is not an ad-hoc(专门) choice, but is inevitable(不可避免的) if we are to respect common sense while making rational coherent inferences. For instance, Cox (1946) showed that if numerical values are used to represent degrees of belief(置信), then a simple set of axioms encoding common sense properties of such beliefs leads uniquely to a set of rules for manipulating degrees of belief that are equivalent to the sum and product rules of probability. This provided the first rigorous proof that probability theory could be regarded as an extension of Boolean logic to situations involving uncertainty (Jaynes, 2003). Numerous other authors have proposed different sets of properties or axioms that such measures of uncertainty should satisfy (Ramsey, 1931; Good, 1950; Savage, 1961; deFinetti, 1970; Lindley, 1982). In each case, the resulting numerical quantities behave precisely according to the rules of probability. It is therefore natural to refer to these quantities as (Bayesian) probabilities.

In the field of pattern recognition, too, it is helpful to have a more general notion of probability. Consider the example of polynomial curve fitting discussed in Section 1.1. It seems reasonable to apply the frequentist notion of probability to the random values of the observed variables t_n. However, we would like to address and quantify the uncertainty that surrounds the appropriate choice for the model parameters w. We shall see that, from a Bayesian perspective, we can use the machinery of probability theory to describe the uncertainty in model parameters such as w, or indeed in the choice of model itself.

Bayes'theorem now acquires a new significance. Recall that in the boxes of fruit example, the observation of the identity of the fruit provided relevant information that altered the probability that the chosen box was the red one. In that example, Bayes'theorem was used to convert a prior probability into a posterior probability by incorporating the evidence provided by the observed

data. As we shall see in detail later, we can adopt a similar approach when making inferences about quantities such as the parameters w in the polynomial curve fitting example. We capture our assumptions about w, before observing the data, in the form of a prior probability distribution $p(w)$. The effect of the observed data $\mathcal{D} = \{t_1, \cdots, t_N\}$ is expressed through the conditional probability $p(\mathcal{D}|\mathbf{w})$, and we shall see later, in Section 1.2.6, how this can be represented explicitly. Bayes'theorem, which takes the form

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})} \tag{1.33}$$

then allows us to evaluate the uncertainty in $\mathbf{w}$ after we have observed $\mathcal{D}$ in the form of the posterior probability $p(\mathbf{w}|\mathcal{D})$.

The quantity $p(\mathcal{D}|\mathbf{w})$ on the right-hand side of Bayes'theorem is evaluated for the observed data set $\mathcal{D}$ and can be viewed as a function of the parameter vector $\mathbf{w}$, in which case it is called the *likelihood function(似然函数)*. It expresses how probable the observed data set is for different settings of the parameter vector $\mathbf{w}$. Note that the likelihood is not a probability distribution over $\mathbf{w}$, and its integral with respect to w does not (necessarily) equal one.

Given this definition of likelihood, we can state Bayes'theorem in words

$$\text{posterior} \propto \text{likelihood} \times \text{prior} \tag{1.34}$$

where **all of these quantities are viewed as functions of $\mathbf{w}$**. The denominator in (1.34) is the normalization constant, which ensures that the posterior distribution on the left-hand side is a valid probability density and integrates to one. Indeed, integrating both sides of (1.34) with respect to $\mathbf{w}$, we can express the denominator in Bayes'theorem in terms of the prior distribution and the likelihood function

$$p(\mathcal{D}) = \int p(\mathcal{D}|\mathbf{w})p(\mathbf{w})d\mathbf{w} \tag{1.35}$$

In both the Bayesian and frequentist paradigms, the likelihood function $p(\mathcal{D}|\mathbf{w})$ plays a central role. However, the manner in which it is used is fundamentally different in the two approaches. In a frequentist setting, w is considered to be a fixed parameter, whose value is determined by some form of 'estimator', and error bars on this estimate are obtained by considering the distribution of possible data sets D. By contrast, from the Bayesian viewpoint there is only a single data set D (namely the one that is actually observed), and the uncertainty in the parameters is expressed through a probability distribution over w.

A widely used frequentist estimator is *maximum likelihood*, in which w is set to the value that maximizes the likelihood function $p(D|\mathbf{w})$. This corresponds to choosing the value of w for which the probability of the observed data set is maximized. In the machine learning literature, the negative log of the likelihood function is called an *error function(误差函数)*. Because the negative logarithm is a monotonically decreasing function, maximizing the likelihood is equivalent to minimizing the error.

One approach to determining frequentist error bars is the *bootstrap(自助法)* (Efron, 1979; Hastie et al., 2001), in which multiple data sets are created as follows. Suppose our original data set consists of N data points $X = \{x_1, \cdots, x_N\}$. We can create a new data set X_B by drawing N points at random from X, with replacement, so that some points in X may be replicated in X_B, whereas other points in X may be absent from X_B. This process can be repeated L times to generate L data sets each of size N and each obtained by sampling from the original data set X. The statistical accuracy of parameter estimates can then be evaluated by looking at the variability of predictions between the different bootstrap data sets.

One advantage of the Bayesian viewpoint is that the inclusion of prior knowledge arises naturally. Suppose, for instance, that a fair-looking coin is tossed three times and lands heads each time. A classical maximum likelihood estimate of the probability of landing heads would give 1, implying

that all future tosses will land heads! By contrast, a Bayesian approach with any reasonable prior will lead to a much less extreme conclusion.

There has been much controversy and debate associated with the relative merits of the frequentist and Bayesian paradigms, which have not been helped by the fact that there is no unique frequentist, or even Bayesian, viewpoint. For instance, one common criticism of the Bayesian approach is that the prior distribution is often selected on the basis of mathematical convenience rather than as a reflection of any prior beliefs. Even the subjective nature of the conclusions through their dependence on the choice of prior is seen by some as a source of difficulty. Reducing the dependence on the prior is one motivation for so-called *noninformative(无信息化)* priors. However, these lead to difficulties when comparing different models, and indeed Bayesian methods based on poor choices of prior can give poor results with high confidence. Frequentist evaluation methods offer some protection from such problems, and techniques such as cross-validation remain useful in areas such as model comparison.

This book places a strong emphasis on the Bayesian viewpoint, reflecting the huge growth in the practical importance of Bayesian methods in the past few years, while also discussing useful frequentist concepts as required.

Although the Bayesian framework has its origins in the 18th century, the practical application of Bayesian methods was for a long time severely limited by the difficulties in carrying through the full Bayesian procedure, particularly the need to marginalize (sum or integrate) over the whole of parameter space, which, as we shall see, is required in order to make predictions or to compare different models. The development of sampling methods, such as Markov chain Monte Carlo (discussed in Chapter 11(?)) along with dramatic improvements in the speed and memory capacity of computers, opened the door to the practical use of Bayesian techniques in an impressive range of problem domains. Monte Carlo methods are very flexible and can be applied to a wide range of models. However, they are computationally

intensive and have mainly been used for small-scale problems.

More recently, highly efficient deterministic approximation schemes such as *variational Bayes(变分贝叶斯)* and *expectation propagation(期望传播)* (discussed in Chapter 10(?)) have been developed. These offer a complementary alternative to sampling methods and have allowed Bayesian techniques to be used in large-scale applications (Blei et al., 2003).

1.2.4 The Gaussian distribution

We shall devote the whole of Chapter 2 to a study of various probability distributions and their key properties. It is convenient, however, to introduce here one of the most important probability distributions for continuous variables, called the *normal(正态)* or *Gaussian(高斯)* distribution. We shall make extensive use of this distribution in the remainder of this chapter and indeed throughout much of the book.

For the case of a single real-valued variable x, the Gaussian distribution is defined by

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} \tag{1.36}$$

which is governed by two parameters: μ, called the *mean(均值)*, and σ^2, called the *variance(方差)*. The square root of the variance, given by σ, is called the *standard deviation(标准差)*, and the reciprocal of the variance, written as $\beta = 1/\sigma^2$, is called the *precision(精度)*. We shall see the motivation for these terms shortly. Figure 1.13 shows a plot of the Gaussian distribution.

From the form of (1.36) we see that the Gaussian distribution satisfies

$$\mathcal{N}(x|\mu,\sigma^2) > 0 \tag{1.37}$$

Also it is straightforward to show that the Gaussian is normalized, so that

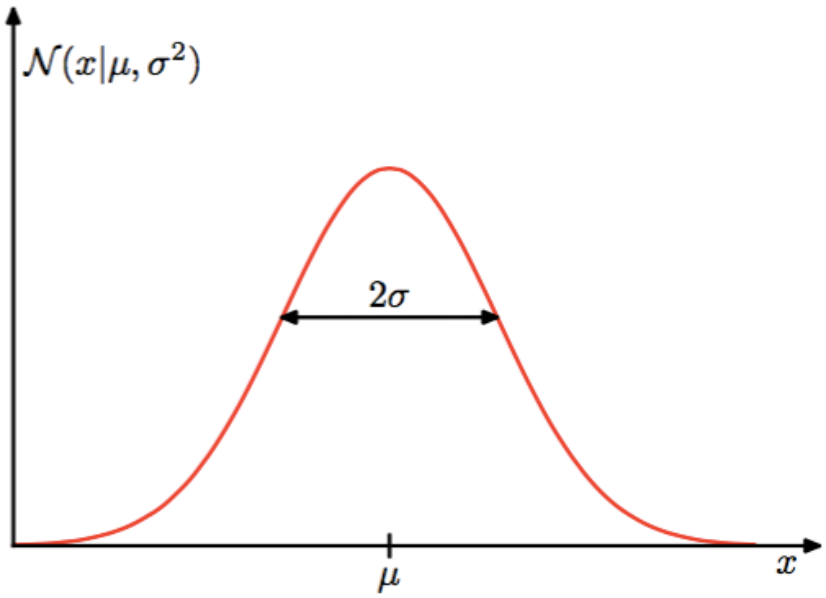


图 1.13: Plot of the univariate Gaussian showing the mean μ and the standard deviation σ.

$$\int_{-\infty}^{\infty} \mathcal{N}(x|\mu, \sigma^2) = 1 \tag{1.38}$$

Thus (1.36) satisfies the two requirements for a valid probability density.

We can readily find expectations of functions of x under the Gaussian distribution. In particular, the average value of x is given by

$$\mathbb{E}[x] = \int_{-\infty}^{\infty} \mathcal{N}(x|\mu, \sigma^2) x dx = \mu \tag{1.39}$$

Because the parameterμ represents the average value of x under the distribution, it is referred to as the mean. Similarly, for the second order moment(矩)

$$\mathbb{E}[x^2] = \int_{-\infty}^{\infty} \mathcal{N}(x|\mu, \sigma^2) x^2 dx = \mu^2 + \sigma^2 \tag{1.40}$$

From (1.39) and (1.40), it follows that the variance of x is given by

$$var[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 = \sigma^2 \tag{1.41}$$

and hence σ^2 is referred to as the variance parameter. The maximum of a distribution is known as its mode(众数). For a Gaussian, the mode coincides with the mean.

We are also interested in the Gaussian distribution defined over a D-dimensional vector x of continuous variables, which is given by

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{D}{2}}} \frac{1}{|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right\} \tag{1.42}$$

where the D-dimensional vector $\boldsymbol{\mu}$ is called the mean, the $D \times D$ matrix $\boldsymbol{\Sigma}$ is called the covariance, and $|\boldsymbol{\Sigma}|$ denotes the determinant of $\boldsymbol{\Sigma}$. We shall make use of the multivariate Gaussian distribution briefly in this chapter, although its properties will be studied in detail in Section 2.3(?).

Now suppose that we have a data set of observations $\boldsymbol{x} = (x_1, \cdots, x_N)^T$, representing N observations of the scalar variable $\mathbf{x}$. Note that we are using

the typeface **x** to distinguish this from a single observation of the vector-valued variable $(x_1, \cdots, x_D)^T$, which we denote by x. We shall suppose that the observations are drawn independently from a Gaussian distribution whose mean μ and variance σ^2 are unknown, and we would like to determine these parameters from the data set. Data points that are drawn independently from the same distribution are said to be *independent and identically distributed(独立同分布)*, which is often abbreviated to i.i.d. We have seen that the joint probability of two independent events is given by the product of the marginal probabilities for each event separately. Because our data set **x** is i.i.d., we can therefore write the probability of the data set, given μ and σ^2, in the form

$$p(\boldsymbol{x}|\mu, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}(x_n|\mu, \sigma^2) \tag{1.43}$$

When viewed as a function of μ and σ^2, this is the likelihood function for the Gaussian and is interpreted diagrammatically in Figure 1.14.

One common criterion for determining the parameters in a probability distribution using an observed data set is to find the parameter values that maximize the likelihood function. This might seem like a strange criterion because, from our foregoing discussion of probability theory, it would seem more natural to maximize the probability of the parameters given the data, not the probability of the data given the parameters. In fact, these two criteria are related, as we shall discuss in the context of curve fitting.

For the moment, however, we shall determine values for the unknown parameters μ and σ^2 in the Gaussian by maximizing the likelihood function (1.43). In practice, it is more convenient to maximize the log of the likelihood function. Because the logarithm is a monotonically increasing function of its argument, maximization of the log of a function is equivalent to maximization of the function itself. Taking the log not only simplifies the subsequent mathematical analysis, but it also helps numerically because the product of a large number of small probabilities can easily underflow

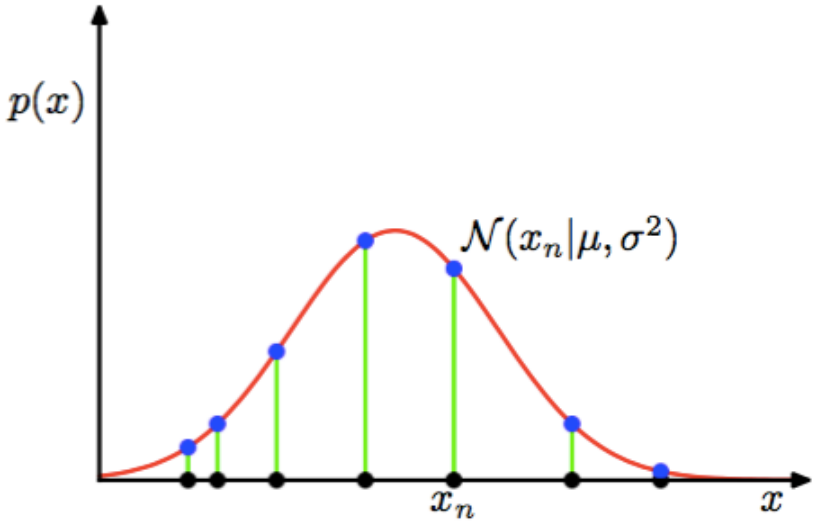


图 1.14: Illustration of the likelihood function for a Gaussian distribution, shown by the red curve. Here the black points denote a data set of values $\{x_n\}$, and the likelihood function given by (1.43) corresponds to the product of the blue values. Maximizing the likelihood involves adjusting the mean and variance of the Gaussian so as to maximize this product.

the numerical precision of the computer, and this is resolved by computing instead the sum of the log probabilities. From (1.36) and (1.43) the log likelihood function can be written in the form

$$\ln p(\boldsymbol{x}|\mu, \sigma^2) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n - \mu)^2 - \frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln(2\pi) \tag{1.44}$$

Maximizing (1.44) with respect to μ, we obtain the maximum likelihood solution given by

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{1.45}$$

which is the *sample mean(*样本均值*)*, i.e., the mean of the observed values $\{x_n\}$. Similarly, maximizing (1.44) with respect to σ^2, we obtain the maximum likelihood solution for the variance in the form

$$\sigma^2_{ML} = \frac{1}{N}\sum_{n=1}^{N}(x_n - \mu_{ML})^2 \tag{1.46}$$

which is the sample variance measured with respect to the sample mean μ_{ML}. Note that we are performing a joint maximization of (1.44) with respect to μ and σ^2, but in the case of the Gaussian distribution the solution for μ decouples from that for σ^2 so that we can first evaluate (1.45) and then subsequently use this result to evaluate (1.46).

Later in this chapter, and also in subsequent chapters, we shall highlight the significant limitations of the maximum likelihood approach. Here we give an indication of the problem in the context of our solutions for the maximum likelihood parameter settings for the univariate Gaussian distribution. In particular, we shall show that **the maximum likelihood approach systematically underestimates the variance of the distribution**. This is an example of a phenomenon called *bias(*偏移*)* and is related to the problem of over-fitting encountered in the context of polynomial curve fitting. We first note that the maximum likelihood solutions

μ_{ML} and σ^2_{ML} are functions of the data set values $x_1, \cdots, x_N$. Consider the expectations of these quantities with respect to the data set values, which themselves come from a Gaussian distribution with parameters μ and σ^2. It is straightforward to show that

$$\mathbb{E}[\mu_{ML}] = \mu \tag{1.47}$$

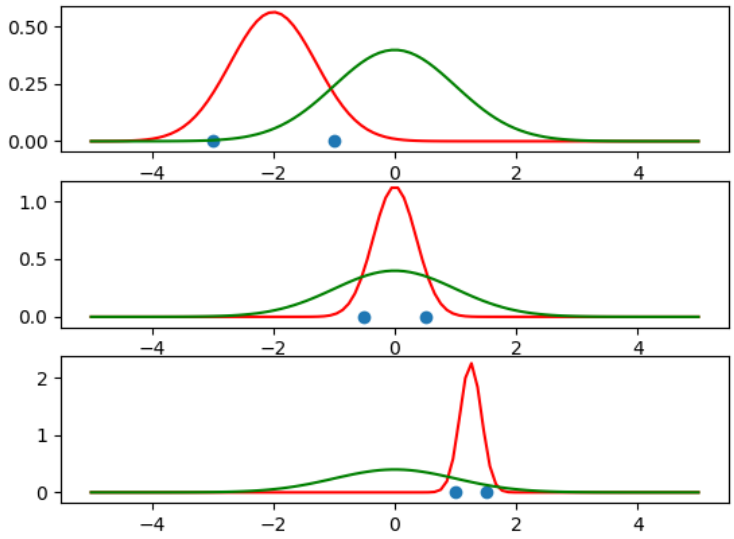
$$\mathbb{E}[\sigma^2_{ML}] = \left(\frac{N-1}{N}\right)\sigma^2 \tag{1.48}$$

so that on average the maximum likelihood estimate will obtain the correct mean but will underestimate the true variance by a factor $(N-1)/N$. The intuition behind this result is given by Figure 1.2.4.

```
x_plot = np.linspace(-5, 5, 100)
y_plot = norm.pdf(x_plot)
for index, x_obs in enumerate([[-1, -3], [0.5, -0.5], [1, 1.5]]):
    x_obs = np.array(x_obs)
    plt.subplot(3, 1, index + 1)
    mean = x_obs.mean()
    std = np.sqrt((len(x_obs) - 1) / len(x_obs)) * np.std(x_obs)
    plt.plot(x_plot, norm.pdf(x_plot, loc=mean, scale=std), "r")
    plt.plot(x_plot, y_plot, "g")
    plt.scatter(x_obs, [0, 0])
plt.savefig("img/fig:1.15.png")
plt.close("all")
```

Listing 1.11: fig:1.15

The green curve shows the true Gaussian distribution from which data is generated, and the three red curves show the Gaussian distributions obtained (a) by fitting to three data sets, each consisting of two data points shown in blue, using the maximum likelihood results (1.45) and (1.46). Averaged across the three data sets, the mean is correct, but the variance is systematically under-estimated because it is measured relative to the sample mean and not relative to the true mean.



From (1.48) it follows that the following estimate for the variance parameter is unbiased

$$\tilde{\sigma}^2 = \frac{N}{N-1}\sigma^2_{ML} = \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \mu_{ML})^2 \tag{1.49}$$

In Section 10.1.3(?), we shall see how this result arises automatically when we adopt a Bayesian approach.

Note that the bias of the maximum likelihood solution becomes less significant as the number N of data points increases, and in the limit $ N→$ the maximum likelihood solution for the variance equals the true variance of the distribution that generated the data. In practice, for anything other than small N, this bias will not prove to be a serious problem. However, throughout this book we shall be interested in more complex models with many parameters, for which the bias problems associated with maximum likelihood will be much more severe. In fact, as we shall see, the issue of bias in maximum likelihood lies at the root of the over-fitting problem that

we encountered earlier in the context of polynomial curve fitting.

1.2.5 Curve fitting re-visited

$$p(\mathbf{w}|\mathcal{D}) \propto p(\mathcal{D}|\mathbf{w})p(\mathbf{w})$$
$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

We have seen how the problem of polynomial curve fitting can be expressed in terms of error minimization. Here we return to the curve fitting example and view it from a probabilistic perspective, thereby gaining some insights into error functions and regularization, as well as taking us towards a full Bayesian treatment.

The goal in the curve fitting problem is to be able to make predictions for the target variable t given some new value of the input variable $\mathbf{x}$ on the basis of a set of training data comprising N input values $\boldsymbol{x} = (x_1, \cdots, x_N)^T$ and their corresponding target values $\boldsymbol{t} = (t_1, \cdots, t_N)^T$. We can express our uncertainty over the value of the target variable using a probability distribution. For this purpose, we shall assume that, given the value of x, the corresponding value of t has a Gaussian distribution with a mean equal to the value $y(x, \mathbf{w})$ of the polynomial curve given by (1.1). Thus we have

$$p(t|x, \mathbf{w}, \beta) = \mathcal{N}(t|y(x, \mathbf{w}), \beta^{-1}) \tag{1.50}$$

where, for consistency with the notation in later chapters, we have defined a precision parameter corresponding to the inverse variance of the distribution. This is illustrated schematically in Figure 1.15.

We now use the training data $\{\boldsymbol{x}, \boldsymbol{t}\}$ to determine the values of the unknown parameters w and β by maximum likelihood. If the data are assumed to be drawn independently from the distribution (1.50), then the likelihood function is given by

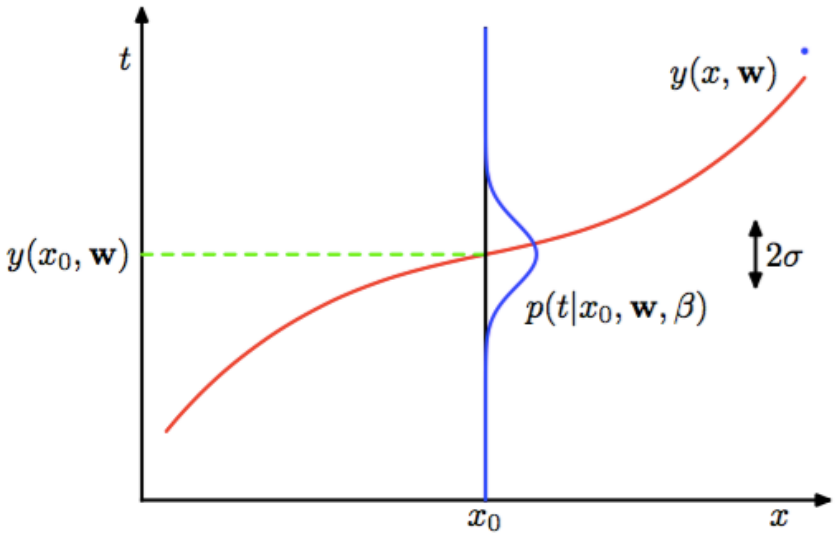


图 1.15: Schematic illustration of a Gaussian conditional distribution for t given x given by (1.50), in which the mean is given by the polynomial function $y(x, \mathbf{w})$, and the precision is given by the parameter β, which is related to the variance by $\beta^{-1} = \sigma^2$.

$$p(\boldsymbol{t}|\boldsymbol{x}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}((t_n)|y(x_n, \mathbf{w}), \beta^{-1}) \tag{1.51}$$

As we did in the case of the simple Gaussian distribution earlier, it is convenient to maximize the logarithm of the likelihood function. Substituting for the form of the Gaussian distribution, given by (1.36), we obtain the log likelihood function in the form

$$\ln p(\boldsymbol{t}|\boldsymbol{x}, \mathbf{w}, \beta) = -\frac{\beta}{2} \sum_{n=1} N\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) \tag{1.52}$$

Consider first the determination of the maximum likelihood solution for the polynomial coefficients, which will be denoted by $\mathbf{w}_{ML}$. These are determined by maximizing (1.52) with respect to w. For this purpose, we can omit the last two terms on the right-hand side of (1.52) because they do not depend on w. Also, we note that scaling the log likelihood by a positive constant coefficient does not alter the location of the maximum with respect to w, and so we can replace the coefficient /2 with 1/2. Finally, instead of maximizing the log likelihood, we can equivalently minimize the negative log likelihood. We therefore see that maximizing likelihood is equivalent, so far as determining mathbf{w} is concerned, to minimizing the sum-of-squares error function defined by (1.2). Thus the *sum-of-squares error function(平方和误差函数)* has arisen as a consequence of maximizing likelihood under the assumption of a Gaussian noise distribution.

We can also use maximum likelihood to determine the precision parameter β of the Gaussian conditional distribution. Maximizing (1.52) with respect to β gives

$$\frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}_{ML}) - t_n\}^2 \tag{1.53}$$

Again we can first determine the parameter vector $\mathrm{w_{ML}}$ governing the mean and subsequently use this to find the precision β_{ML} as was the case

for the simple Gaussian distribution.

Having determined the parameters w and β, we can now make predictions for new values of x. Because we now have a probabilistic model, these are expressed in terms of the *predictive distribution(预测分布)* that gives the probability distribution over t, rather than simply a point estimate, and is obtained by substituting the maximum likelihood parameters into (1.50) to give

$$p(t|x, \mathbf{w}_{ML}, \beta_{ML}) = \mathcal{N}(t|y(x, \mathbf{w}_{ML}), \beta_{ML}^{-1}) \tag{1.54}$$

Now let us take a step towards a more Bayesian approach and introduce a prior distribution over the polynomial coefficients w. For simplicity, let us consider a Gaussian distribution of the form

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left\{-\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}\right\} \tag{1.55}$$

where α is the precision of the distribution, and $M + 1$ is the total number of elements in the vector w for an $\mathrm{M^{th}}$ order polynomial. Variables such as α, which control the distribution of model parameters, are called *hyperparameters(超参数)*. Using Bayes'theorem, the posterior distribution for w is proportional to the product of the prior distribution and the likelihood function

$$p(\mathbf{w}|\boldsymbol{x}, \boldsymbol{t}, \alpha, \beta) \propto p(\boldsymbol{t}|\boldsymbol{x}, \mathbf{w}, \beta)p(\mathbf{w}|\alpha) \tag{1.56}$$

We can now determine w by finding the most probable value of w given the data, in other words by maximizing the posterior distribution. This technique is called *maximum posterior(最大后验)*, or simply MAP. Taking the negative logarithm of (1.56) and combining with (1.52) and (1.55), we find that the maximum of the posterior is given by the minimum of

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w} \tag{1.57}$$

Thus we see that maximizing the posterior distribution is equivalent to minimizing the regularized sum-of-squares error function encountered earlier in the form (1.4), with a regularization parameter given by $\lambda = \alpha/\beta$.

1.2.6 TODO Bayesian curve fitting

Although we have included a prior distribution $p(\mathbf{w}|\alpha)$, we are so far still making a point estimate of w and so this does not yet amount to a Bayesian treatment. In a fully Bayesian approach, we should consistently apply the sum and product rules of probability, which requires, as we shall see shortly, that we integrate over all values of w. **Such marginalizations lie at the heart of Bayesian methods for pattern recognition**.

In the curve fitting problem, we are given the training data $\mathbf{x}$ and $\mathbf{t}$, along with a new test point x, and our goal is to predict the value of t. We therefore wish to evaluate the predictive distribution $p(t|x, \boldsymbol{x}, \boldsymbol{t})$. Here we shall assume that the parameters α and β are fixed and known in advance (in later chapters we shall discuss how such parameters can be inferred from data in a Bayesian setting).

A Bayesian treatment simply corresponds to a consistent application of the sum and product rules of probability, which allow the predictive distribution to be written in the form

$$p(t|x, \boldsymbol{x}, \boldsymbol{t}) = \int p(t|x, \mathbf{w}) p(\mathbf{w}|\boldsymbol{x}, \boldsymbol{t}) d\mathbf{w} \tag{1.58}$$

Here $p(t|x, \mathbf{w})$ is given by (1.50), and we have omitted the dependence on α and β to simplify the notation. Here $p(\mathbf{w}|\boldsymbol{x}, \boldsymbol{t})$ is the posterior distribution over parameters, and can be found by normalizing the right-hand side of (1.56).

We shall see in Section 3.3(?) that, for problems such as the curve-fitting example, this posterior distribution is a Gaussian and can be evaluated analytically. Similarly, the integration in (1.58) can also be performed analytically with the result that the predictive distribution is given by a

Gaussian of the form

$$p(t|x, \boldsymbol{x}, \boldsymbol{t}) = \mathcal{N}(t|m(x), s^2(x)) \tag{1.59}$$

where the mean and variance are given by

$$m(x) = \beta \boldsymbol{\phi}(x)^T \mathbf{S} \sum_{n=1}^{N} \boldsymbol{\phi}(x_n) t_n \tag{1.60}$$

$$s^2(x) = \beta^{-1} + \boldsymbol{\phi}(x)^T \mathbf{S} \boldsymbol{\phi}(x) \tag{1.61}$$

Here the matrix S is given by

$$\mathbf{S} = \alpha \mathbf{I} + \beta \sum_{n=1}^{N} \boldsymbol{\phi}(x_n) \boldsymbol{\phi}^T(x) \tag{1.62}$$

where I is the unit matrix, and we have defined the vector $\boldsymbol{\phi}(x)$ with elements $\boldsymbol{\phi}_i(x) = x^i$ for $i = 0, \cdots, M$.

We see that the variance, as well as the mean, of the predictive distribution in (1.59) is dependent on x. The first term in (1.60) represents the uncertainty in the predicted value of t due to the noise on the target variables and was expressed already in the maximum likelihood predictive distribution (1.54) through β_{ML}^{-1} . However, the ML second term arises from the uncertainty in the parameters w and is a consequence of the Bayesian treatment. The predictive distribution for the synthetic(人造的) sinusoidal regression problem is illustrated in Figure 1.16.

```
lw = 2
feature = PolynomialFeatures(degree=9)
X_train = feature.transform(data_train["x"])
X_plot = feature.transform(data_plot["x"])
model = BayesianRegressor(alpha=5e-3, beta=11.1)
model.fit(X_train, data_train["t"].values)
y_mean, y_std = model.predict(X_plot, return_std=True)
plt.figure(figsize=(6, 5))
plt.scatter(
    data_train["x"],
    data_train["t"],
    facecolor="none",
    edgecolor="b",
    s=50,
    label="training data")
plt.plot(data_plot["x"], y_mean, c="r", label="mean")
plt.fill_between(
    data_plot["x"],
    y_mean + y_std,
    y_mean - y_std,
    color='pink',
    label="std",
    alpha=0.5)
plt.plot(
    data_plot["x"],
    data_plot["x"].apply(func),
    color='g',
    linewidth=lw,
    label="Ground Truth")
plt.annotate("M=9", xy=(0.2, -1))
plt.legend()
plt.savefig("img/fig:1.17.png")
```

1.3 Model Selection

In our example of polynomial curve fitting using least squares, we saw that there was an optimal order of polynomial that gave the best generalization. The order of the polynomial controls the number of free parameters

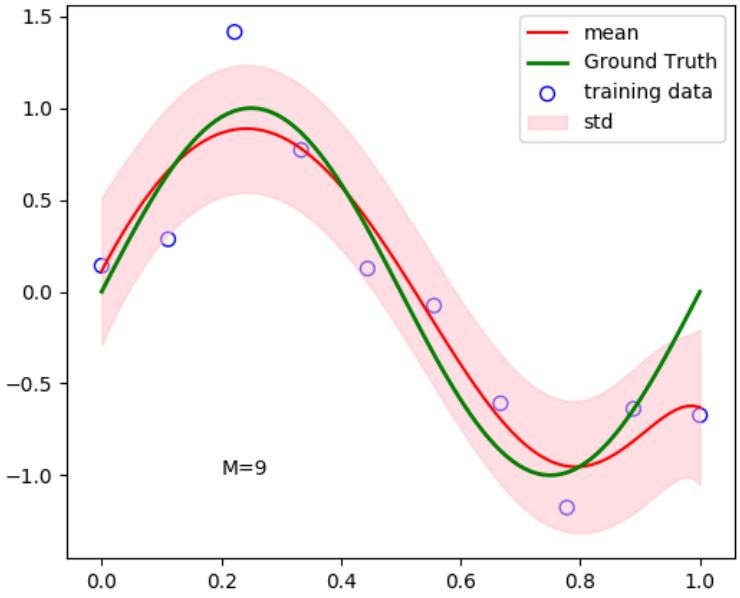


图 1.16: The predictive distribution resulting from a Bayesian treatment of polynomial curve fitting using an $M = 9$ polynomial, with the fixed parameters $\alpha = 5 \times 10^{-3}$ and $\beta = 11.1$ (corresponding to the known noise variance), in which the red curve denotes the mean of the predictive distribution and the red region corresponds to ± 1 standard deviation around the mean.

in the model and thereby governs the model complexity. With regularized least squares, the regularization coefficient λ also controls the effective complexity of the model, whereas for more complex models, such as mixture distributions or neural networks there may be multiple parameters governing complexity. In a practical application, we need to determine the values of such parameters, and the principal objective in doing so is usually to achieve the best predictive performance on new data. Furthermore, as well as finding the appropriate values for complexity parameters within a given model, we may wish to consider a range of different types of model in order to find the best one for our particular application.

We have already seen that, in the maximum likelihood approach, the performance on the training set is not a good indicator of predictive performance on unseen data due to the problem of over-fitting. If data is plentiful(丰富的), then one approach is simply to use some of the available data to train a range of models, or a given model with a range of values for its complexity parameters, and then to compare them on independent data, sometimes called a validation set(验证集), and select the one having the best predictive performance. If the model design is iterated many times using a limited size data set, then some over-fitting to the validation data can occur and so it may be necessary to keep aside a third test set on which the performance of the selected model is finally evaluated.

In many applications, however, the supply of data for training and testing will be limited, and in order to build good models, we wish to use as much of the available data as possible for training. However, if the validation set is small, it will give a relatively noisy estimate of predictive performance. One solution to this dilemma is to use *cross-validation(交叉验证)*, which is illustrated in Figure 1.17. This allows a proportion $(S-1)/S$ of the available data to be used for training while making use of all of the data to assess performance. When data is particularly scarce(稀疏), it may be appropriate to consider the case $S = N$, where N is the total number of data points,

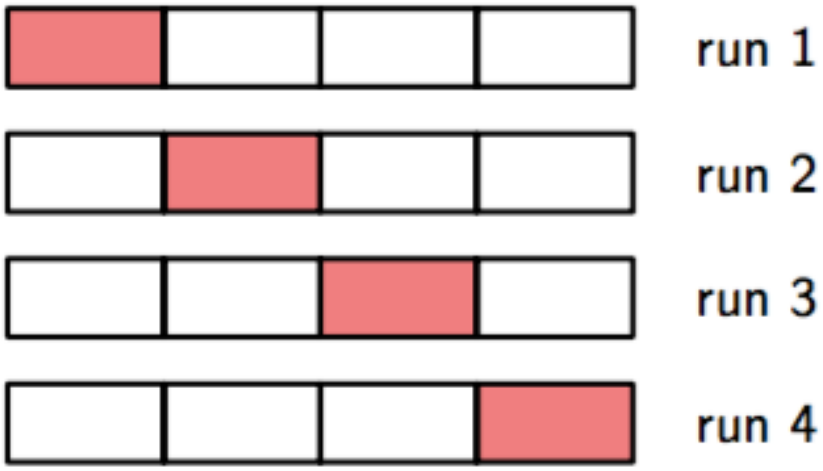


图 1.17: The technique of S-fold cross-validation, illustrated here for the case of S = 4, involves taking the available data and partitioning it into S groups (in the simplest case these are of equal size). Then S 1 of the groups are used to train a set of models that are then evaluated on the remaining group. This procedure is then repeated for all S possible choices for the held-out group, indicated here by the red blocks, and the performance scores from the S runs are then averaged.

which gives the *leave-one-out(留一法)* technique.

One major drawback of cross-validation is that the number of training runs that must be performed is increased by a factor of S, and this can prove problematic for models in which the training is itself computationally expensive. A further problem with techniques such as cross-validation that use separate data to assess performance is that we might have multiple complexity parameters for a single model (for instance, there might be several regularization parameters). Exploring combinations of settings for such parameters could, in the worst case, require a number of training runs that is exponential in the number of parameters. Clearly, we need a better approach. Ideally, **this should rely only on the training data and should allow multiple hyperparameters and model types to be compared in a single training run**. We therefore need to find a measure of performance which depends only on the training data and which does not suffer from bias due to over-fitting.

Historically various 'information criteria'have been proposed that attempt to correct for the bias of maximum likelihood by the addition of a penalty term to compensate(补偿) for the over-fitting of more complex models. For example, the *Akaike information criterion(赤池信息量准则)*, or AIC (Akaike, 1974), chooses the model for which the quantity

$$\ln p(\mathcal{D}|\mathbf{w}_{ML}) - M \tag{1.63}$$

is largest. Here $p(\mathcal{D}|\mathbf{w}_{ML})$ is the best-fit log likelihood, and M is the number of adjustable parameters in the model. A variant of this quantity, called the *Bayesian information criterion(贝叶斯信息准则)*, or BIC, will be discussed in Section 4.4.1(?). Such criteria do not take account of the uncertainty in the model parameters, however, and in practice they tend to favor overly simple models. We therefore turn in Section 3.4(?) to a fully Bayesian approach where we shall see how complexity penalties arise in a natural and principled way.

1.4 TODO The Curse of Dimensionality

1.5 Decision Theory

We have seen in Section 1.2 how probability theory provides us with a consistent mathematical framework for quantifying and manipulating uncertainty. Here we turn to a discussion of decision theory that, when combined with probability theory, allows us to make optimal decisions in situations involving uncertainty such as those encountered in pattern recognition.

Suppose we have an input vector x together with a corresponding vector t of target variables, and our goal is to predict t given a new value for x. For regression problems, t will comprise continuous variables, whereas for classification problems t will represent class labels. The joint probability distribution $p(\mathbf{x}, \mathbf{t})$ provides a complete summary of the uncertainty associated with these variables. Determination of p(x, t) from a set of training data is an example of *inference(推断)* and is typically a very difficult problem whose solution forms the subject of much of this book. In a practical application, however, we must often make a specific prediction for the value of t, or more generally take a specific action based on our understanding of the values t is likely to take, and this aspect(方向) is the subject(主题) of decision theory.

Consider, for example, a medical diagnosis(诊断) problem in which we have taken an X-ray image of a patient, and we wish to determine whether the patient has cancer or not. In this case, the input vector x is the set of pixel intensities(灰度值) in the image, and output variable t will represent the presence of cancer, which we denote by the class C_1, or the absence of cancer, which we denote by the class C_2. We might, for instance, choose t to be a binary variable such that $t = 0$ corresponds to class C_1 and $t = 1$ corresponds to class C_2. We shall see later that this choice of label values is particularly convenient for probabilistic models. The general inference problem then involves determining the joint distribution $p(\mathbf{x}, C_k)$, or equivalently $p(\mathbf{x}, t)$, which gives us the most complete probabilistic description of

the situation. Although this can be a very useful and informative quantity, in the end we must decide either to give treatment to the patient or not, and we would like this choice to be optimal in some appropriate sense (Duda and Hart, 1973). This is the decision step, and it is the subject of decision theory to tell us how to make optimal decisions given the appropriate probabilities. We shall see that the decision stage is generally very simple, even trivial, once we have solved the inference problem.

Here we give an introduction to the key ideas of decision theory as required forthe rest of the book. Further background, as well as more detailed accounts, can be found in Berger (1985) and Bather (2000).

Before giving a more detailed analysis, let us first consider informally how we might expect probabilities to play a role in making decisions. When we obtain the X-ray image x for a new patient, our goal is to decide which of the two classes to assign to the image. We are interested in the probabilities of the two classes given the image, which are given by $p(C_k|x)$. Using Bayes' theorem, these probabilities can be expressed in the form

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})} \tag{1.64}$$

Note that any of the quantities appearing in Bayes'theorem can be obtained from the joint distribution $p(\mathbf{x}, C_k)$ by either marginalizing or conditioning with respect to the appropriate variables. We can now interpret $p(C_k)$ as the prior probability for the class C_k, and $p(C_k|\mathbf{x})$ as the corresponding posterior probability. Thus $p(C_1)$ represents the probability that a person has cancer, before we take the X-ray measurement. Similarly, $p(C_1|\mathbf{x})$ is the corresponding probability, revised using Bayes'theorem in light of the information contained in the X-ray. If our aim is to minimize the chance of assigning x to the wrong class, **then intuitively we would choose the class having the higher posterior probability**. We now show that this intuition is correct, and we also discuss more general criteria for making decisions.

1.5.1 Minimizing the misclassification rate

Suppose that our goal is simply to make as few misclassifications as possible. We need a rule that assigns each value of x to one of the available classes. Such a rule will divide the input space into regions R_k called *decision regions(决策区域)*, one for each class, such that all points in R_k are assigned to class C_k. The boundaries between decision regions are called *decision boundaries(决策边界)* or *decision surfaces(决策面)*. Note that each decision region need not be contiguous but could comprise some number of disjoint regions. We shall encounter examples of decision boundaries and decision regions in later chapters. In order to find the optimal decision rule, consider first of all the case of two classes, as in the cancer problem for instance. A mistake occurs when an input vector belonging to class C_1 is assigned to class C_2 or vice versa. The probability of this occurring is given by

$$\begin{aligned} p(\text{mistake}) &= p(\mathbf{x} \in R_1, C_2) + p(\mathbf{x} \in R_2, C_1) \\ &= \int_{R_1} p(\mathbf{x}, C_2) d\mathbf{x} + \int_{R_2} p(\mathbf{x}, C_1) d\mathbf{x} \end{aligned} \tag{1.65}$$

We are free to choose the decision rule that assigns each point x to one of the two classes. Clearly to minimize $p(\text{mistake})$ we should arrange that each x is assigned to whichever class has the smaller value of the integrand(被积分函数) in (1.65). Thus, if $p(x, C_1) > p(x, C_2)$ for a given value of x, then we should assign that x to class C_1. From the product rule of probability we have $p(\mathbf{x}, C_k) = p(C_k|\mathbf{x})p(\mathbf{x})$. Because the factor $p(\mathbf{x})$ is common to both terms, we can restate this result as saying that **the minimum probability of making a mistake is obtained if each value of x is assigned to the class for which the posterior probability $p(C_k|\mathbf{x})$ is largest**. This result is illustrated for two classes, and a single input variable x, in Figure 1.18.

For the more general case of K classes, it is slightly easier to maximize the probability of being correct, which is given by

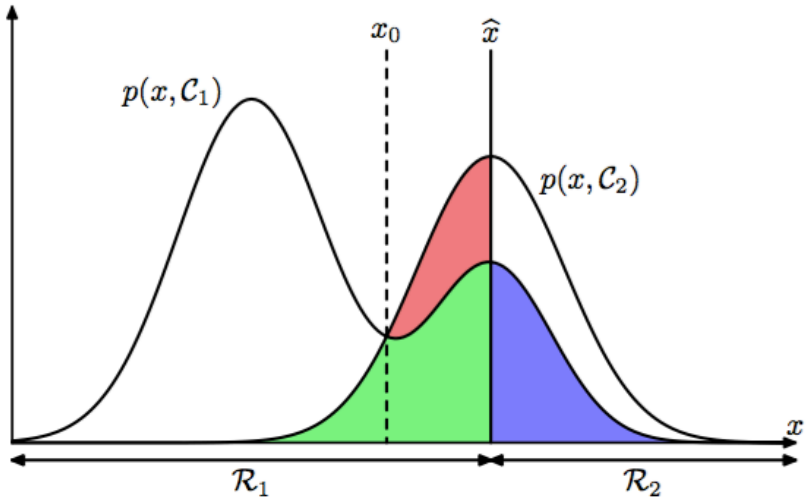


图 1.18: Schematic illustration of the joint probabilities $p(x, C_k)$ for each of two classes plotted against x, together with the decision boundary $x = \bar{x}$. Values of $x \geqslant \bar{x}$ are classified as class C_2 and hence belong to decision region R_2, whereas points $x < \bar{x}$ are classified as C_1 and belong to R_1. Errors arise from the blue, green, and red regions, so that for $x < \bar{x}$ the errors are due to points from class C_2 being misclassified as C_1 (represented by the sum of the red and green regions), and conversely for points in the region $x \geqslant \bar{x}$ the errors are due to points from class C_1 being misclassified as C_2 (represented by the blue region). As we vary the location of the decision boundary, the combined areas of the blue and green regions remains constant, whereas the size of the red region varies. The optimal choice for is where the curves for $p(x, C_1)$ and $p(x, C_2)$ cross, corresponding to $\bar{x} = x_0$, because in this case the red region disappears. This is equivalent to the minimum misclassification rate decision rule, which assigns each value of x to the class having the higher posterior probability $p(C_k|x)$.

$$
\begin{aligned}
p(\text{correct}) &= \sum_{k=1}^{K} p(\mathbf{x} \in R_k, C_k) \\
&= \sum_{k=1}^{K} \int_{R_k} p(\mathbf{x} \in R_k, C_k) d\mathbf{x}
\end{aligned}
\tag{1.66}
$$

which is maximized when the regions R_k are chosen such that each x is assigned to the class for which $p(\mathbf{x}, C_k)$ is largest. Again, using the product rule \$p(x,C_k) = p(C_k|x)p(x), and noting that the factor of $p(\mathbf{x})$ is common to all terms, we see that each x should be assigned to the class having the largest posterior probability $p(C_k|\mathbf{x})$.

1.5.2 Minimizing the expected loss

For many applications, our objective will be more complex than simply minimizing the number of misclassifications. Let us consider again the medical diagnosis problem. We note that, if a patient who does not have cancer is incorrectly diagnosed(被诊断) as having cancer, the consequences may be some patient distress plus the need for further investigations. Conversely, if a patient with cancer is diagnosed as healthy, the result may be premature(比预期早的) death due to lack of treatment. Thus the consequences of these two types of mistake can be dramatically different. It would clearly be better to make fewer mistakes of the second kind, even if this was at the expense of making more mistakes of the first kind.

We can formalize such issues through the introduction of a *loss function(损失函数)*, also called a *cost function(代价函数)*, which is a single, overall measure of loss incurred in taking any of the available decisions or actions. Our goal is then to minimize the total loss incurred. Note that some authors consider instead a *utility function(效用函数)*, whose value they aim to maximize. These are equivalent concepts if we take the utility to be simply the negative of the loss, and throughout this text we shall use the loss function convention. Suppose that, for a new value of x, the true class is C_k

$$\begin{array}{c} \\ \text{cancer} \\ \text{normal} \end{array} \begin{array}{c} \begin{array}{cc} \text{cancer} & \text{normal} \end{array} \\ \begin{pmatrix} 0 & 1000 \\ 1 & 0 \end{pmatrix} \end{array}$$

图 1.19: An example of a loss matrix with elements L_{kj} for the cancer treatment problem. The rows correspond to the true class, whereas the columns correspond to the assignment of class made by our decision criterion.

and that we assign x to class C_j (where j may or may not be equal to k). In so doing, we incur(招致) some level of loss that we denote by L_{kj} , which we can view as the k, j element of a *loss matrix(损失矩阵)*. For instance, in our cancer example, we might have a loss matrix of the form shown in Figure 1.19. This particular loss matrix says that there is no loss incurred if the correct decision is made, there is a loss of 1 if a healthy patient is diagnosed as having cancer, whereas there is a loss of 1000 if a patient having cancer is diagnosed as healthy.

The optimal solution is the one which minimizes the loss function. However, the loss function depends on the true class, which is unknown. For a given input vector x, our uncertainty in the true class is expressed through the joint probability distribution $p(\mathbf{x}, C_k)$ and so we seek instead to minimize the average loss, where the average is computed with respect to this distribution, which is given by

$$\mathbb{E}[L] = \sum_k \sum_j \int_{R_j} L_{kj} p(\mathbf{x}, C_k) d\mathbf{x} \tag{1.67}$$

Each x can be assigned independently to on e of the decision regions R_j . Our goal is to choose the regions R_j in order to minimize the expected loss (1.67), which implies that for each x we should minimize $\sum_k L_{kj} p(\mathbf{x}, C_k)$. As before, we can use the product rule \$p(x,C$_k$) = p(C$_k$|x)p(x) to eliminate the common factor of $p(\mathbf{x})$. Thus the decision rule that minimizes the expected

loss is the one that assigns each new x to the class j for which the quantity

$$\sum_k L_{kj} p(C_K|\mathbf{x}) \tag{1.68}$$

is a minimum. This is clearly trivial to do, once we know the posterior class probabilities $p(C_k|\mathbf{x})$.

1.5.3 The reject option

We have seen that classification errors arise from the regions of input space where the largest of the posterior probabilities $p(C_k|\mathbf{x})$ is significantly less than unity, or equivalently where the joint distributions $p(\mathbf{x}, C_k)$ have comparable values. These are the regions where we are relatively uncertain about class membership. In some applications, it will be appropriate to avoid making decisions on the difficult cases in anticipation of a lower error rate on those examples for which a classification decision is made. This is known as the *reject option(拒绝选项)*. For example, in our hypothetical medical illustration, it may be appropriate to use an automatic system to classify those X-ray images for which there is little doubt as to the correct class, while leaving a human expert to classify the more ambiguous cases. We can achieve this by introducing a threshold(阈值) θ and rejecting those inputs x for which the largest of the posterior probabilities $p(C_k|\mathbf{x})$ is less than or equal to θ. This is illustrated for the case of two classes, and a single continuous input variable x, in Figure 1.20. Note that setting $\theta = 1$ will ensure that all examples are rejected, whereas if there are K classes then setting $< 1/K$ will ensure that no examples are rejected.(抽屉原理) Thus the fraction of examples that get rejected is controlled by the value of θ.

We can easily extend the reject criterion to minimize the expected loss, when a loss matrix is given, taking account of the loss incurred when a reject decision is made.

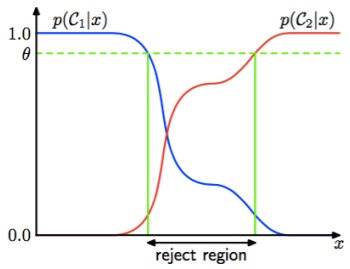


图 1.20: Illustration of the reject option. Inputs x such that the larger of the two posteior probabilities is less than or equal to θ some threshold θ will be rejected.

1.5.4 TODO Inference and decision

We have broken the classification problem down into two separate stages, **the *inference stage(推断阶段)* in which we use training data to learn a model for** $p(C_k|\mathbf{x})$, and the subsequent ***decision stage(决策阶段)* in which we use these posterior probabilities to make optimal class assignments**. An alternative possibility would be to solve both problems together and simply learn a function that maps inputs x directly into decisions. Such a function is called a *discriminant function(判别函数)*.

In fact, we can identify three distinct approaches to solving decision problems, all of which have been used in practical applications. These are given, in decreasing order of complexity, by:

1. (a)

   First solve the inference problem of determining the class-conditional densities $p(\mathbf{x}|Ck)$ for each class C_k individually. Also separately infer the prior class probabilities $p(C_k)$. Then use Bayes'theorem in the form

$$p(Ck|x) = \frac{p(x|Ck)p(Ck)}{p(x)} \tag{1.69}$$

to find the posterior class probabilities $p(C_k|\mathbf{x})$. As usual, the denominator in Bayes'theorem can be found in terms of the quantities appearing in the numerator, because

$$p(\mathbf{x}) = \sum_k p(\mathbf{x}|C_k)p(C_k) \tag{1.70}$$

Equivalently, we can model the joint distribution $p(\mathbf{x}, C_k)$ directly and then normalize to obtain the posterior probabilities. Having found the posterior probabilities, we use decision theory to determine class membership for each new input x. Approaches that explicitly or implicitly model the distribution of inputs as well as outputs are known as *generative models(生成式模式)*, because by sampling from them it is possible to generate synthetic data points in the input space.

2. (b)

   First solve the inference problem of determining the posterior class probabilities $p(Ck|\mathbf{x})$, and then subsequently use decision theory to assign each new x to one of the classes. Approaches that model the posterior probabilities directly are called *discriminative models(判别式模式)*.

3. (c)

   Find a function $f(x)$, called a discriminant function(判别函数), which maps each input x directly onto a class label. For instance, in the case of two-class problems, $f(\cdot)$ might be binary valued and such that $f = 0$ represents class C_1 and $f = 1$ represents class C_2. In this case, probabilities play no role.

4. **TODO** conclusion

   Let us consider the relative merits of these three alternatives. Approach (a) is the most demanding because it involves finding the joint

distribution over both x and C_k. For many applications, x will have high dimensionality, and consequently we may need a large training set in order to be able to determine the class-conditional densities to reasonable accuracy. Note that the class priors $p(C_k)$ can often be estimated simply from the fractions of the training set data points in each of the classes. One advantage of approach (a), however, is that it also allows the marginal density of data p(x) to be determined from (1.70). This can be useful for detecting new data points that have low probability under the model and for which the predictions maybe of low accuracy, which is known as *outlier detection(离群点检测)* or *novelty detection(异常检测)* (Bishop, 1994; Tarassenko, 1995).

However, if we only wish to make classification decisions, then it can be wasteful of computational resources, and excessively demanding of data, to find the joint distribution $p(\mathbf{x}, C_k)$ when in fact we only really need the posterior probabilities $p(Ck|\mathbf{x})$, which can be obtained directly through approach (b). Indeed, the class- conditional densities may contain a lot of structure that has little effect on the pos- terior probabilities, as illustrated in Figure 1.27. There has been much interest in exploring the relative merits of generative and discriminative approaches to machine learning, and in finding ways to combine them (Jebara, 2004; Lasserre et al., 2006). An even simpler approach is (c) in which we use the training data to find a discriminant function f(x) that maps each x directly onto a class label, thereby combining the inference and decision stages into a single learning problem. In the example of Figure 1.27, this would correspond to finding the value of x shown by the vertical green line, because this is the decision boundary giving the minimum probability of misclassification. With option (c), however, we no longer have access to the posterior probabilities p(Ck|x). There are many powerful reasons for wanting to compute the posterior probabilities, even if we subsequently use them to make de-

cisions. These include: Minimizing risk. Consider a problem in which the elements of the loss matrix are subjected to revision from time to time (such as might occur in a financialapplication). If we know the posterior probabilities, we can trivially revise the minimum risk decision criterion by modifying (1.81) appropriately. If we have only a discriminant function, then any change to the loss matrix would require that we return to the training data and solve the classification problem afresh. Rejectoption. Posteriorprobabilitiesallowustodeterminearejectioncriterionthat will minimize the misclassification rate, or more generally the expected loss, for a given fraction of rejected data points. Compensating for class priors. Consider our medical X-ray problem again, and suppose that we have collected a large number of X-ray images from the gen- eral population for use as training data in order to build an automated screening system. Because cancer is rare amongst the general population, we might find that, say, only 1 in every 1,000 examples corresponds to the presence of can- cer. If we used such a data set to train an adaptive model, we could run into severe difficulties due to the small proportion of the cancer class. For instance, a classifier that assigned every point to the normal class would already achieve 99.9% accuracy and it would be difficult to avoid this trivial solution. Also, even a large data set will contain very few examples of X-ray images corre- sponding to cancer, and so the learning algorithm will not be exposed to a broad range of examples of such images and hence is not likely to generalize well. A balanced data set in which we have selected equal numbers of exam- ples from each of the classes would allow us to find a more accurate model. However, we then have to compensate for the effects of our modifications to the training data. Suppose we have used such a modified data set and found models for the posterior probabilities. From Bayes'theorem (1.82), we see that the posterior probabilities are proportional to the prior probabilities,

which we can interpret as the fractions of points in each class. We can therefore simply take the posterior probabilities obtained from our artificially balanced data set and first divide by the class fractions in that data set and then multiply by the class fractions in the population to which we wish to apply the model. Finally, we need to normalize to ensure that the new posterior probabilities sum to one. Note that this procedure cannot be applied if we have learned a discriminant function directly instead of determining posterior probabilities. Combining models. For complex applications, we may wish to break the problem into a number of smaller subproblems each of which can be tackled by a sep- arate module. For example, in our hypothetical medical diagnosis problem, we may have information available from, say, blood tests as well as X-ray im- ages. Rather than combine all of this heterogeneous information into one huge input space, it may be more effective to build one system to interpret the X- ray images and a different one to interpret the blood data. As long as each of the two models gives posterior probabilities for the classes, we can combine the outputs systematically using the rules of probability. One simple way to do this is to assume that, for each class separately, the distributions of inputs for the X-ray images, denoted by xI, and the blood data, denoted by xB, areindependent, so that Section 8.2 p(xI,xB|Ck) = p(xI|Ck)p(xB|Ck). (1.84) This is an example of conditional independence property, because the indepen- dence holds when the distribution is conditioned on the class Ck. The posterior probability, given both the X-ray and blood data, is then given by p(Ck|xI, xB) p(xI, xB|Ck)p(Ck) p(xI |Ck)p(xB |Ck)p(Ck) p(Ck |xI)p(Ck |xB) (1.85) p(Ck) Thus we need the class prior probabilities p(Ck), which we can easily estimate from the fractions of data points in each class, and then we need to normalize the resulting posterior probabilities so they sum to one. The particular condi- tional independence assumption (1.84) is an example

of the naive Bayes model. Note that the joint marginal distribution p(xI , xB) will typically not factorize under this model. We shall see in later chapters how to construct models for combining data that do not require the conditional independence assumption (1.84).

1.5.5 TODO Loss functions for regression

1.6 TODO Information Theory

In this chapter, we have discussed a variety of concepts from probability theory and decision theory that will form the foundations for much of the subsequent discussion in this book. We close this chapter by introducing some additional concepts from the field of information theory, which will also prove useful in our development of pattern recognition and machine learning techniques. Again, we shall focus only on the key concepts, and we refer the reader elsewhere for more detailed discussions (Viterbi and Omura, 1979; Cover and Thomas, 1991; MacKay, 2003) .

We begin by considering a discrete random variable x and we ask how much information is received when we observe a specific value for this variable. The amount of information can be viewed as the 'degree of surprise' on learning the value of x. **If we are told that a highly improbable event has just occurred, we will have received more information than if we were told that some very likely event has just occurred**, and if we knew that the event was certain to happen we would receive no information. Our measure of information content will therefore depend on the probability distribution $p(x)$, and we therefore look for a quantity $h(x)$ that is a monotonic function of the probability $p(x)$ and that expresses the information content. The form of $h(\cdot)$ can be found by noting that if we have two events x and y that are unrelated, then the information gain from observing both of them should be the sum of the information gained from each of them separately, so that $h(x, y) = h(x) + h(y)$. Two unrelated events will be statistically independent and so $p(x, y) = p(x)p(y)$. From these two

relationships, it is easily shown that $h(x)$ must be given by the logarithm of $p(x)$ and so we have

$$h(x) = -\log_2 p(x) \tag{1.71}$$

where the negative sign ensures that information is positive or zero. Note that low probability events x correspond to high information content. The choice of basis for the logarithm is arbitrary, and for the moment we shall adopt the convention prevalent in information theory of using logarithms to the base of 2. In this case, as we shall see shortly, the units of $h(x)$ are bits ('binary digits').

Now suppose that a sender wishes to transmit the value of a random variable to a receiver. The average amount of information that they transmit in the process is obtained by taking the expectation of (1.71) with respect to the distribution $p(x)$ and is given by

$$H[x] = -\sum_x p(x) \log_2 p(x). \tag{1.72}$$

This important quantity is called the *entropy(熵)* of the random variable x. Note that $\lim_{p\to 0} p \ln p = 0$ and so we shall take $p(x) \ln p(x) = 0$ whenever we encounter a value for x such that $p(x) = 0$.

So far we have given a rather heuristic motivation for the definition of information (1.71) and the corresponding entropy (1.72). We now show that these definitions indeed possess useful properties. Consider a random variable x having 8 possible states, each of which is equally likely. In order to communicate the value of x to a receiver, we would need to transmit a message of length 3 bits. Notice that the entropy of this variable is given by

$$H[x] = -8 \times \frac{1}{8} \log_2 \frac{1}{8} = 3\text{bits}.$$

Now consider an example (Cover and Thomas, 1991) of a variable having 8 possible states$\{a, b, c, d, e, f, g, h\}$ for which the respective probabilities are given by $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64})$. The entropy in this case is given by

$$H[x] = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{8}\log_2\frac{1}{8} - \frac{4}{64}\log_2\frac{1}{64} = 2\text{bits}.$$

We see that the nonuniform distribution has a smaller entropy than the uniform one, and we shall gain some insight into this shortly when we discuss the interpretation of entropy in terms of disorder. For the moment, let us consider how we would transmit the identity of the variable's state to a receiver. We could do this, as before, using a 3-bit number. However, we can take advantage of the nonuniform distribution by using shorter codes for the more probable events, at the expense of longer codes for the less probable events, in the hope of getting a shorter average code length. This can be done by representing the states $\{a, b, c, d, e, f, g, h\}$ using, for instance, the following set of code strings: $0, 10, 110, 1110, 111100, 111101, 111110, 111111$. The average length of the code that has to be transmitted is then

$$\text{average code length} = \frac{1}{2}\times 1 + \frac{1}{4}\times 2 + \frac{1}{8}\times 4 + \frac{4}{64}\times 6 = 2\text{bits}$$

which again is the same as the entropy of the random variable. Note that shorter code strings cannot be used because it must be possible to disambiguate a concatenation of such strings into its component parts. For instance, 11001110 decodes uniquely into the state sequence c, a, d.

This relation between entropy and shortest coding length is a general one. The *noiseless coding theorem(无噪声编码定理)* (Shannon, 1948) states that **the entropy is a lower bound on the number of bits needed to transmit the state of a random variable**.

From now on, we shall switch to the use of natural logarithms in defining entropy, as this will provide a more convenient link with ideas elsewhere in this book. In this case, the entropy is measured in units of 'nats'instead of bits, which differ simply by a factor of $\ln_2$.

We have introduced the concept of entropy in terms of the average amount of information needed to specify the state of a random variable.

In fact, the concept of entropy has much earlier origins in physics where it was introduced in the context of equilibrium(平衡) thermodynamics(熱力学) and later given a deeper interpretation as a measure of disorder through developments in statistical mechanics(力学). We can understand this alternative view of entropy by considering a set of N identical(完全相同) objects that are to be divided amongst a set of bins, such that there are n_i objects in the i-th bin. Consider the number of different ways of allocating the objects to the bins. There are N ways to choose the first object, $(N-1)$ ways to choose the second object, and so on, leading to a total of $N!$ ways to allocate all N objects to the bins, where $N!$ (pronounced 'factorial N ') denotes the product $N \times (N-1) \times \times \cdots \times 2 \times 1$. However, we don't wish to distinguish between rearrangements of objects within each bin. In the i-th bin there are $n_i!$ ways of reordering the objects, and so the total number of ways of allocating the N objects to the bins is given by

$$W = \frac{N!}{\prod_i n_i!} \tag{1.73}$$

which is called the *multiplicity(乗数)*. The entropy is then defined as the logarithm of the multiplicity scaled by an appropriate constant

$$H = \frac{1}{N} \ln W = \frac{1}{N} \ln N! - \frac{1}{N} \ln \sum_i \ln n_i!. \tag{1.74}$$

We now consider the limit $N \to \infty$, in which the fractions n_i/N are held fixed, and apply Stirling's approximation

$$\ln N! \simeq N \ln N - N \tag{1.75}$$

which gives

$$H = -\lim_{N\to\infty} \sum_i \left(\frac{n_i}{N}\right) \ln \left(\frac{n_i}{N}\right) = -\sum_i p_i \in p_i \tag{1.76}$$

where we have used $\sum_i ni = N$. Here $p_i = \lim_{N\to N}(n_i/N)$ is the probability of an object being assigned to the i^th bin. In physics terminology,

the specific arrangements of objects in the bins is called a *microstate(*围观状态*)*, and the overall distribution of occupation numbers, expressed through the ratios n_i/N, is called a *macrostate(*宏观状态*)*. The multiplicity W is also known as the weight of the macrostate.

We can interpret the bins as the states x_i of a discrete random variable X, where $p(X = xi) = pi$. The entropy of the random variable X is then

$$H[p] = -\sum_i p(x_i) \ln p(x_i) \tag{1.77}$$

Distributions $p(x_i)$ that are sharply peaked around a few values will have a relatively low entropy, whereas those that are spread more evenly across many values will have higher entropy, as illustrated in Figure 1.21. Because $0 \leqslant p_i \leqslant 1$, the entropy is nonnegative, and it will equal its minimum value of 0 when one of the $p_i = 1$ and all other $p_{j\neq i} = 0$. The maximum entropy configuration can be found by maximizing H using a Lagrange multiplier to enforce the normalization constraint on the probabilities. Thus we maximize

$$\tilde{H} = -\sum_i p(x_i) \ln p(x_i) + \lambda \left(\sum_i p(x_i) - 1 \right) \tag{1.78}$$

from which we find that all of the $p(x_i)$ are equal and are given by $p(x_i) = 1/M$ where M is the total number of states x_i. The corresponding value of the entropy is then $H = \ln M$. This result can also be derived from Jensen's inequality (to be discussed shortly). To verify that the stationary point is indeed a maximum, we can evaluate the second derivative of the entropy, which gives

$$\frac{\partial^2 \tilde{H}}{\partial p(x_i) \partial p(x_j)} = -I_{ij} \frac{1}{p_i} \tag{1.79}$$

where I_{ij} are the elements of the identity matrix.

We can extend the definition of entropy to include distributions $p(x)$ over continuous variables x as follows. First divide x into bins of width

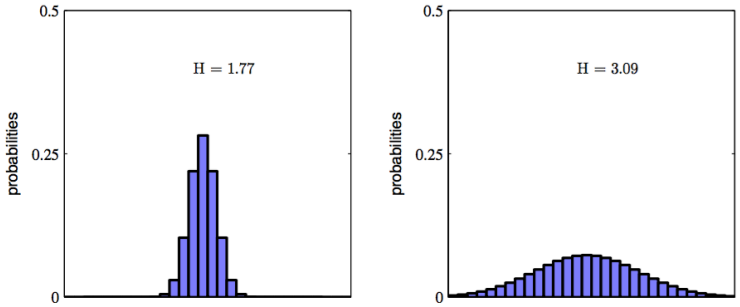


图 1.21: Histograms of two probability distributions over 30 bins illustrating the higher value of the entropy H for the broader distribution. The largest entropy would arise from a uniform distribution that would give $H = -ln(1/30) = 3.40$.

Δ. Then, assuming $p(x)$ is continuous, the *mean value theorem(均值定理)* (Weisstein, 1999) tells us that, for each such bin, there must exist a value x_i such that

$$\int_{i\Delta}^{(i+1)\Delta} p(x)dx = p(x_i)\Delta \tag{1.80}$$

We can now quantize(量化) the continuous variable x by assigning any value x to the value x_i whenever x falls in the i^{th} bin. The probability of observing the value x_i is then $p(x_i)\Delta$. This gives a discrete distribution for which the entropy takes the form

$$H_\Delta = -\sum_i p(x_i)\Delta \ln(p(x_i)\Delta) = -\sum_i p(x_i)\Delta \ln p(x_i) - \ln \Delta \tag{1.81}$$

where we have used $\sum_i p(x_i)\Delta = 1$, which follows from (1.80). We now omit the second term $-\ln\Delta$ on the right-hand side of (1.81) and then consider the limit $\Delta \to 0$. The first term on the right-hand side of (1.81) will approach the integral of $p(x)\ln p(x)$ in this limit so that

$$\lim_{\Delta \to 0} -\sum_i p(x_i)\Delta \ln p(x_i) = -\int p(x) \ln p(x) dx \tag{1.82}$$

where the quantity on the right-hand side is called the *differential entropy(微分熵)*. We see that the discrete and continuous forms of the entropy differ by a quantity $\ln \Delta$, which diverges(发散) in the limit $\Delta \to 0$. This reflects the fact that to **specify a continuous variable very precisely requires a large number of bits**. For a density defined over multiple continuous variables, denoted collectively by the vector $\mathbf{x}$, the differential entropy is given by

$$H[\mathbf{x}] = -\int p(\mathbf{x}) \ln p(\mathbf{x}) d\mathbf{x}. \tag{1.83}$$

In the case of discrete distributions, we saw that the maximum entropy configuration corresponded to an equal distribution of probabilities across the possible states of the variable. Let us now consider the maximum entropy configuration for a continuous variable. In order for this maximum to be well defined, it will be necessary to constrain the first and second moments of $p(x)$ as well as preserving the normalization constraint. We therefore maximize the differential entropy with the

$$\int_{-\infty}^{\infty} p(x) dx = 1 \tag{1.84}$$

$$\int_{-\infty}^{\infty} x p(x) dx = \mu \tag{1.85}$$

$$\int_{-\infty}^{\infty} x^2 p(x) dx = \sigma^2 \tag{1.86}$$

The constrained maximization can be performed using Lagrange multipliers so that we maximize the following functional with respect to $p(x)$

$$\begin{aligned} &-\int_{-\infty}^{\infty} p(x) \ln p(x) dx + \lambda_1 \left(\int_{-\infty}^{\infty} p(x) dx - 1 \right) \\ &+ \lambda_2 \left(\int_{-\infty}^{\infty} x p(x) dx - \mu \right) + \lambda_3 \left(\int_{-\infty}^{\infty} (x-\mu)^2 p(x) dx - \sigma^2 \right) \end{aligned} \tag{1.87}$$

Using the calculus of variations, we set the derivative of this functional to zero giving

$$p(x) = \exp\left\{-1 + \lambda_1 + \lambda_2 x + \lambda_3 (x-\mu)^2\right\}. \tag{1.88}$$

The Lagrange multipliers can be found by back substitution of this result into the three constraint equations, leading finally to the result

$$p(x) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\} \tag{1.89}$$

and so the distribution that maximizes the differential entropy is the Gaussian. Note that we did not constrain the distribution to be nonnegative when we maximized the entropy. However, because the resulting distribution is indeed nonnegative, we see with hindsight that such a constraint is not necessary.

1.6.1 Relative entropy and mutual information

1.7 Guide: Ordinary Least Squares

http://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares

`LinearRegression` fits a linear model with coefficients $w = (w_1, ..., w_p)$ to minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_w ||Xw - y||_2{}^2$$

`LinearRegression` will take in its `fit` method arrays `X`, `y` and will store the coefficients w of the linear model in its `coef_` member.

However, coefficient estimates for Ordinary Least Squares rely on the independence of the model terms. When terms are correlated and the columns of the design matrix X have an approximate linear dependence, the design matrix becomes close to singular and as a result, the least-squares estimate

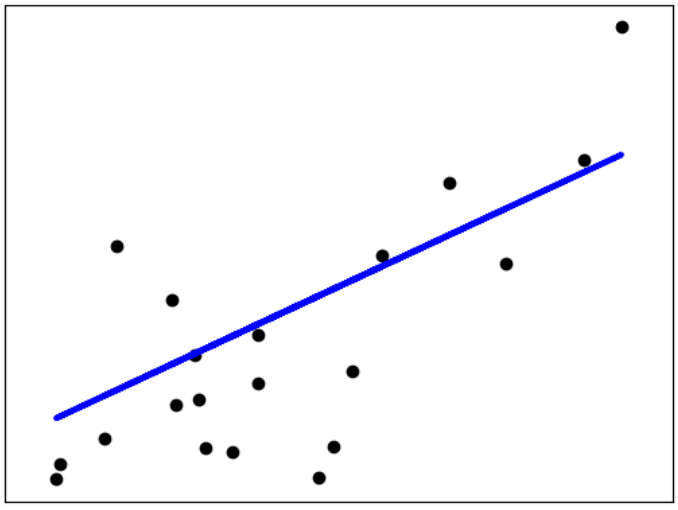
becomes highly sensitive to random errors in the observed response, producing a large variance. This situation of multicollinearity can arise, for example, when data are collected without an experimental design.

1.7.1 Example: Linear Regression Example

http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-glr-auto-examples-linear-model-plot-ols-py

This example uses the only the first feature of the diabetes dataset, in order to illustrate a two-dimensional plot of this regression technique. The straight line can be seen in the plot, showing how linear regression attempts to draw a straight line that will best minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation.

The coefficients, the residual sum of squares and the variance score are also calculated.



```
Coefficients:
```

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
# Load the diabetes dataset
diabetes = datasets.load_diabetes()
# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
# Create linear regression object
regr = linear_model.LinearRegression()
# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(diabetes_y_test, diabetes_y_pred))
# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.savefig("img/1.s.Linear-Regression-Example.png")
plt.close("all")
```

Listing 1.12: Linear Regression Example

```
 [938.23786125]
Mean squared error: 2548.07
Variance score: 0.47
```

1.7.2 Ordinary Least Squares Complexity

This method computes the least squares solution using a singular value decomposition of X. If X is a matrix of size (n, p) this method has a cost of $O(np^2)$, assuming that $n \geq p$.

1.8 Guide: Regression metrics

The `sklearn.metrics` module implements several loss, score, and utility functions to measure regression performance. Some of those have been enhanced to handle the `multioutput` case: $\text{mean}_{\text{squarederror}}$, $\text{mean}_{\text{absoluteerror}}$, $\text{explained}_{\text{variancescore}}$ and r2_{score}.

These functions have an multioutput keyword argument which specifies the way the scores or losses for each individual target should be averaged. The default is '`uniform_average`', which specifies a uniformly weighted mean over outputs. If an `ndarray` of shape `(n_outputs,)` is passed, then its entries are interpreted as weights and an according weighted average is returned. If `multioutput` is '`raw_values`' is specified, then all unaltered individual scores or losses will be returned in an array of shape `(n_outputs,)`.

The `r2_score` and `explained_variance_score` accept an additional value `'variance_weighted'` for the multioutput parameter. This option leads to a weighting of each individual score by the variance of the corresponding target variable. This setting quantifies the globally captured unscaled variance. If the target variables are of different scale, then this score puts more importance on well explaining the higher variance variables. `multioutput='variance_weighted'` is the default value for `r2_score` for backward compatibility. This will be changed to `uniform_average` in the future.

1.8.1 Explained variance score(解释方差分数)

The `explained_variance_score` computes the explained variance regression score.

If $\hat{y}$ is the estimated target output, y the corresponding (correct) target output, and Var is Variance, the square of the standard deviation, then the explained variance is estimated as follow:

$$\texttt{explained\_variance}(y, \hat{y}) = 1 - \frac{Var\{y - \hat{y}\}}{Var\{y\}}$$

The best possible score is 1.0, lower values are worse.

1.8.2 Mean absolute error

The `mean_absolute_error` function computes mean absolute error, a risk metric corresponding to the expected value of the absolute error loss or l_1 norm loss.

If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean absolute error (MAE) estimated over $\mathrm{n}_{\mathrm{samples}}$ is defined as

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{n_{\mathrm{samples}}} \sum_{i=0}^{n_{\mathrm{samples}}-1} \left| y_i - \hat{y}_i \right| .$$

1.8.3 Mean squared error

The `mean_squared_error` function computes mean square error, a risk metric corresponding to the expected value of the squared (quadratic) error or loss.

If $_{\mathrm{i}}$ is the predicted value of the i^{th} sample, and y_{i} is the corresponding true value, then the mean squared error (MSE) estimated over $\mathrm{n}_{\mathrm{samples}}$ is defined as

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

Examples:

See Gradient Boosting regression for an example of mean squared error usage to evaluate gradient boosting regression.

1.8.4 Median absolute error(绝对中位差)

The `median_absolute_error` is particularly interesting because it is robust to outliers. The loss is calculated by taking the median of all absolute differences between the target and the prediction.

If $\hat{y}_i$ is the predicted value of the i^{th} sample and y_i is the corresponding true value, then the median absolute error (MedAE) estimated over n_{samples} is defined as

$$\text{MedAE}(y, \hat{y}) = \text{median}(\mid y_1 - \hat{y}_1 \mid, \ldots, \mid y_n - \hat{y}_n \mid).$$

The $\text{median}_{\text{absoluteerror}}$ does not support multioutput.

1.8.5 R^2 score, the coefficient of determination

The $r2_score$ function computes R^2, the coefficient of determination. It provides a measure of how well future samples are likely to be predicted by the model. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

If $\hat{y}_i$ is the predicted value of the i^{th} sample and y_i is the corresponding true value, then the score $R estimated over n_{samples} is defined as

$R^2(y,) = 1 - \{\frac{}{\sum_{i=0}^{n_{\text{samples}} - 1} (y_i - _i)^2}\}\{\sum_{i=0}^{n}$
$\{\text{samples}\} - 1 (y_i -)^2\}$
where $= 1\frac{}{\{n_{\text{samples}}}\} \sum_{i=0}^{n_{\text{samples}} - 1} y_i$.

1.9 Guide: Ridge Regression

`Ridge` regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of coefficients. The ridge coefficients minimize a penalized residual sum of squares,

$$\min_w ||Xw - y||_2{}^2 + \alpha||w||_2{}^2$$

Here, $\alpha \geq 0$ is a complexity parameter that controls the amount of shrinkage: the larger the value of α, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.

As with other linear models, `Ridge` will take in its fit method arrays X, y and will store the coefficients w of the linear model in its `coef_` member:

1.9.1 Ridge Complexity

This method has the same order of complexity than an Ordinary Least Squares.

1. Setting the regularization parameter: generalized Cross-Validation

   `RidgeCV` implements ridge regression with built-in cross-validation of the alpha parameter. The object works in the same way as `GridSearchCV` except that it defaults to Generalized Cross-Validation (GCV), an efficient form of leave-one-out cross-validation:

```
from sklearn import linear_model
reg = linear_model.RidgeCV(alphas=[0.1, 1.0, 10.0])
print(reg.fit([[0, 0], [0, 0], [1, 1]], [0, .1, 1]))
print(reg.alpha_)
```

```
RidgeCV(alphas=[0.1, 1.0, 10.0], cv=None, fit_intercept=True, gcv_mode=None,
    normalize=False, scoring=None, store_cv_values=False)
0.1
```

1.9.2 References

"Notes on Regularized Least Squares", Rifkin & Lippert (technical report, course slides).

1.9.3 Example: Plot Ridge coefficients as a function of the regularization

Shows the effect of collinearity in the coefficients of an estimator.

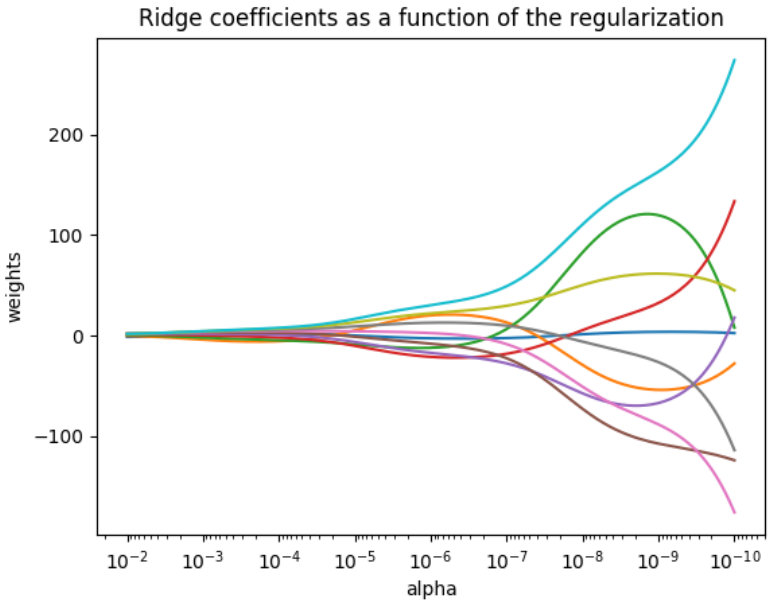
`Ridge` Regression is the estimator used in this example. Each color represents a different feature of the coefficient vector, and this is displayed as a function of the regularization parameter.

This example also shows the usefulness of applying Ridge regression to highly ill-conditioned matrices. For such matrices, a slight change in the target variable can cause huge variances in the calculated weights. In such cases, it is useful to set a certain regularization (alpha) to reduce this variation (noise).

When alpha is very large, the regularization effect dominates the squared loss function and the coefficients tend to zero. At the end of the path, as alpha tends toward zero and the solution tends towards the ordinary least squares, coefficients exhibit big oscillations. In practise it is necessary to tune alpha in such a way that a balance is maintained between both.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
# X is the 10x10 Hilbert matrix
X = 1. / (np.arange(1, 11) + np.arange(0, 10)[:, np.newaxis])
y = np.ones(10)
# #############################################################################
# Compute paths
n_alphas = 200
alphas = np.logspace(-10, -2, n_alphas)
coefs = []
for a in alphas:
    ridge = linear_model.Ridge(alpha=a, fit_intercept=False)
    ridge.fit(X, y)
    coefs.append(ridge.coef_)
# #############################################################################
# Display results
ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Ridge coefficients as a function of the regularization')
plt.axis('tight')
plt.savefig("img/1.s.Plot-Ridge-coefficients-as-a-function-of-the-regularization.png")
plt.close("all")
```

Listing 1.13: Plot Ridge coefficients as a function of the regularization about Hilbert matrix



1.9.4 TODO Example: Classification of text documents using sparse features

1.10 TODO Guide: Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes'theorem with the "naive"assumption of independence between every pair of features. Given a class variable y and a dependent feature vector x_1 through x_n, Bayes'theorem states the following relationship:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Using the naive independence assumption that

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y),$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Since $P(x_1, \ldots, x_n)$ is constant given the input, we can use the following classification rule:

$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$.

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, **famously document classification and spam filtering**. They require a small amount of training data to estimate the necessary parameters. (For theoretical reasons why naive Bayes works well, and on which types of data it does, see the references below.)

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator, so the probability outputs from `predict_proba` are not to be taken too seriously.

1.10.1 References:

H. Zhang (2004). The optimality of Naive Bayes. Proc. FLAIRS.

1.10.2 TODO Gaussian Naive Bayes

`GaussianNB` implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

The parameters σ_y and μ_y are estimated using maximum likelihood.

The log likelihood function can be written in the form

$$\ln P(\boldsymbol{x}|\mu, \sigma^2) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n - \mu)^2 - \frac{N}{2}\ln^2 - \frac{N}{2}ln(2\pi)$$

Maximizing it with respect to μ and σ^2 respectively, we obtain the maximum likelihood solutions given by

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N} x_n$$
$$\sigma^2_{ML} = \frac{1}{N}(x_n - \mu_{ML})^2$$

Note that we are performing a joint maximization with respect to μ and σ^2, but **in the case of the Gaussian distribution the solution for μ decouples from that for σ^2.**

1.11 TODO Guide: Bayesian Regression

Bayesian regression techniques can be used to include regularization parameters in the estimation procedure: the regularization parameter is not set in a hard sense but tuned to the data at hand.

This can be done by introducing uninformative priors over the hyper parameters of the model. The ℓ_2 regularization used in Ridge Regression is equivalent to finding a maximum a posteriori estimation under a Gaussian prior over the parameters w with precision λ^{-1}. Instead of setting lambda manually, it is possible to treat it as a random variable to be estimated from the data.

To obtain a fully probabilistic model, the output y is assumed to be Gaussian distributed around Xw:

$$p(y|X, w, \alpha) = \mathcal{N}(y|Xw, \alpha)$$

Alpha is again treated as a random variable that is to be estimated from the data.

The advantages of Bayesian Regression are:

- It adapts to the data at hand.
- It can be used to include regularization parameters in the estimation procedure.

The disadvantages of Bayesian regression include:

- Inference of the model can be time consuming.

1.11.1 References

- A good introduction to Bayesian methods is given in C. Bishop: Pattern Recognition and Machine learning
- Original Algorithm is detailed in the book Bayesian learning for neural networks by Radford M. Neal

1.11.2 Bayesian Ridge Regression

BayesianRidge estimates a probabilistic model of the regression problem as described above. The prior for the parameter w is given by a spherical Gaussian:

$$p(w|\lambda) = \mathcal{N}(w|0, \lambda^{-1}\mathbf{I_p})$$

The priors over α and λ are chosen to be gamma distributions, the conjugate prior for the precision of the Gaussian.

The resulting model is called *Bayesian Ridge Regression*, and is similar to the classical Ridge. The parameters w, α and λ are estimated jointly during the fit of the model. The remaining hyperparameters are the parameters of the gamma priors over α and λ. These are usually chosen to be non-informative. The parameters are estimated by maximizing the marginal log likelihood.

By default $\alpha_1 = \alpha_2 = \lambda_1 = \lambda_2 = 10^{-6}$.

Bayesian Ridge Regression is used for regression:

```
from sklearn import linear_model
X = [[0., 0.], [1., 1.], [2., 2.], [3., 3.]]
Y = [0., 1., 2., 3.]
reg = linear_model.BayesianRidge()
print(reg.fit(X, Y))
```

```
BayesianRidge(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True,
       fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300,
       normalize=False, tol=0.001, verbose=False)
```

After being fitted, the model can then be used to predict new values:

```
reg.predict ([[1, 0.]])
```

```
array([0.50000013])
```

The weights w of the model can be access:

```
reg.coef_
```

```
array([0.49999993, 0.49999993])
```

Due to the Bayesian framework, the weights found are slightly different to the ones found by Ordinary Least Squares. However, Bayesian Ridge Regression is more robust to ill-posed problem.

1. **TODO** Examples: Bayesian Ridge Regression

   Computes a Bayesian Ridge Regression on a synthetic dataset.

   See Bayesian Ridge Regression for more information on the regressor.

   Compared to the OLS (ordinary least squares) estimator, the coefficient weights are slightly shifted toward zeros, which stabilises them.

   As the prior on the weights is a Gaussian prior, the histogram of the estimated weights is Gaussian.

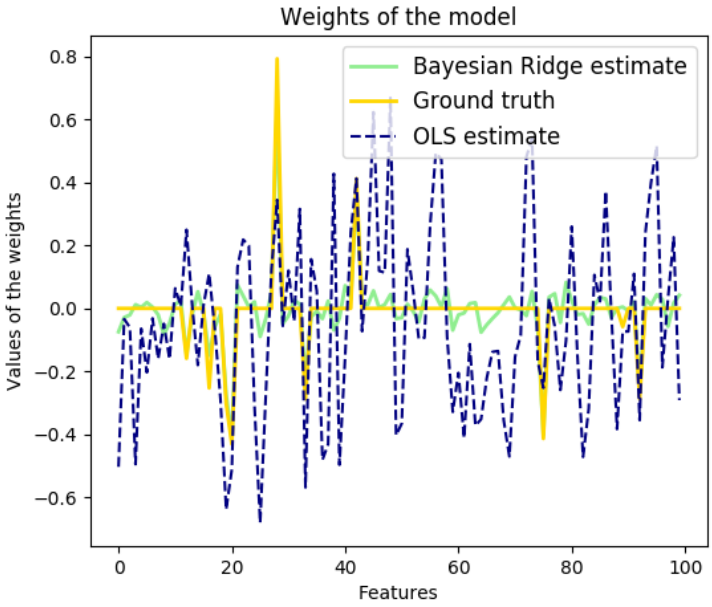
   The estimation of the model is done by iteratively maximizing the marginal log-likelihood of the observations.

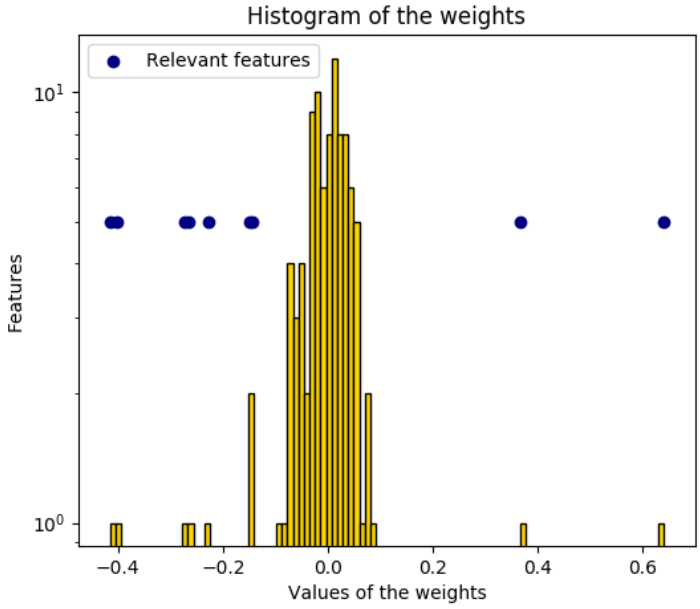
   We also plot predictions and uncertainties for Bayesian Ridge Regression for one dimensional regression using polynomial feature expansion. Note the uncertainty starts going up on the right side of the plot. This is because these test samples are outside of the range of the training samples.

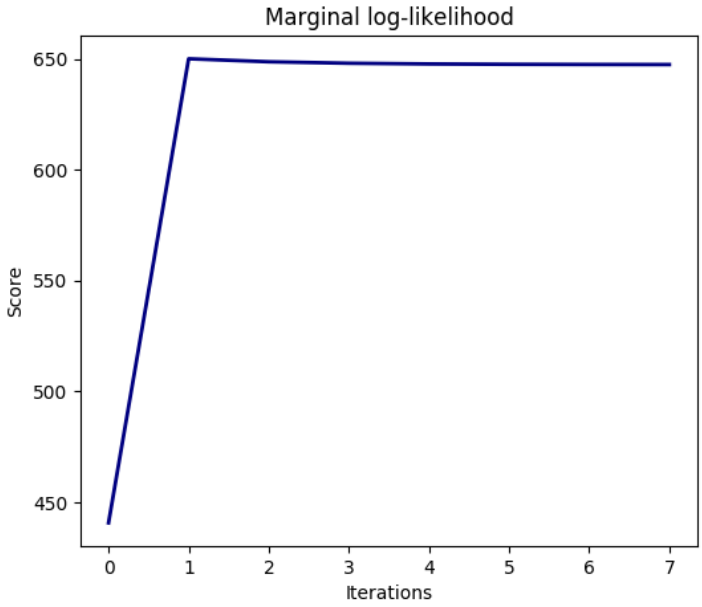
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.linear_model import BayesianRidge, LinearRegression
# #############################################################################
# Generating simulated data with Gaussian weights
np.random.seed(0)
n_samples, n_features = 100, 100
X = np.random.randn(n_samples, n_features)  # Create Gaussian data
# Create weights with a precision lambda_ of 4.
lambda_ = 4.
w = np.zeros(n_features)
# Only keep 10 weights of interest
relevant_features = np.random.randint(0, n_features, 10)
for i in relevant_features:
    w[i] = stats.norm.rvs(loc=0, scale=1. / np.sqrt(lambda_))
# Create noise with a precision alpha of 50.
alpha_ = 50.
noise = stats.norm.rvs(loc=0, scale=1. / np.sqrt(alpha_), size=n_samples)
# Create the target
y = np.dot(X, w) + noise
# #############################################################################
# Fit the Bayesian Ridge Regression and an OLS for comparison
clf = BayesianRidge(compute_score=True)
clf.fit(X, y)
ols = LinearRegression()
ols.fit(X, y)
```

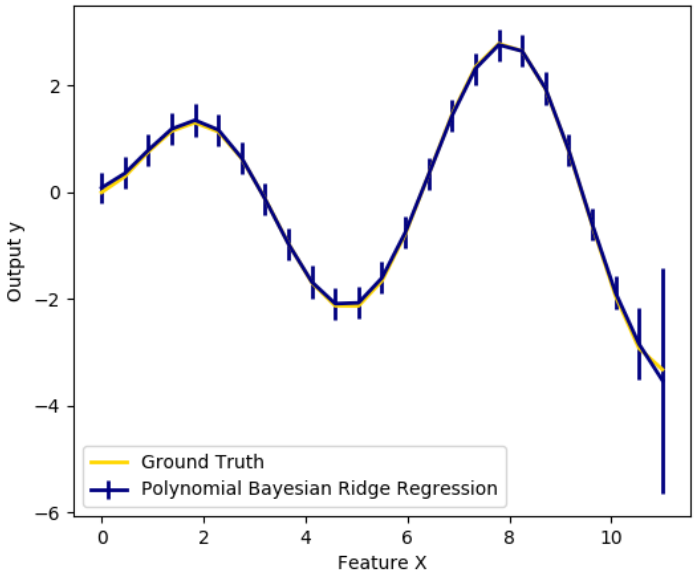
```
# ##############################################################################
# Plot true weights, estimated weights, histogram of the weights, and
# predictions with standard deviations
lw = 2
plt.figure(figsize=(6, 5))
plt.title("Weights of the model")
plt.plot(
    clf.coef_,
    color='lightgreen',
    linewidth=lw,
    label="Bayesian Ridge estimate")
plt.plot(w, color='gold', linewidth=lw, label="Ground truth")
plt.plot(ols.coef_, color='navy', linestyle='--', label="OLS estimate")
plt.xlabel("Features")
plt.ylabel("Values of the weights")
plt.legend(loc="best", prop=dict(size=12))
plt.savefig("img/1.s.BayesianRidgeRegression.1.png")
plt.figure(figsize=(6, 5))
plt.title("Histogram of the weights")
plt.hist(clf.coef_, bins=n_features, color='gold', log=True, edgecolor='black')
plt.scatter(
    clf.coef_[relevant_features],
    5 * np.ones(len(relevant_features)),
    color='navy',
    label="Relevant features")
plt.ylabel("Features")
plt.xlabel("Values of the weights")
plt.legend(loc="upper left")
plt.savefig("img/1.s.BayesianRidgeRegression.2.png")
plt.figure(figsize=(6, 5))
plt.title("Marginal log-likelihood")
plt.plot(clf.scores_, color='navy', linewidth=lw)
plt.ylabel("Score")
plt.xlabel("Iterations")
plt.savefig("img/1.s.BayesianRidgeRegression.3.png")
```

```
# Plotting some predictions for polynomial regression
def f(x, noise_amount):
    y = np.sqrt(x) * np.sin(x)
    noise = np.random.normal(0, 1, len(x))
    return y + noise_amount * noise


degree = 10
X = np.linspace(0, 10, 100)
y = f(X, noise_amount=0.1)
clf_poly = BayesianRidge()
clf_poly.fit(np.vander(X, degree), y)

X_plot = np.linspace(0, 11, 25)
y_plot = f(X_plot, noise_amount=0)
y_mean, y_std = clf_poly.predict(np.vander(X_plot, degree), return_std=True)
plt.figure(figsize=(6, 5))
plt.errorbar(
    X_plot,
    y_mean,
    y_std,
    color='navy',
    label="Polynomial Bayesian Ridge Regression",
    linewidth=lw)
plt.plot(X_plot, y_plot, color='gold', linewidth=lw, label="Ground Truth")
plt.ylabel("Output y")
plt.xlabel("Feature X")
plt.legend(loc="lower left")
plt.savefig("img/1.s.BayesianRidgeRegression.4.png")
plt.close("all")
```


Weights of the model
Bayesian Ridge estimate
Ground truth
OLS estimate
Values of the weights
Features
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
20
40
60
80
100


Histogram of the weights
Relevant features
Features
Values of the weights
10^1
10^0
−0.4
−0.2
0.0
0.2
0.4
0.6





2. References

- More details can be found in the article Bayesian Interpolation by MacKay, David J. C.

1.11.3 Automatic Relevance Determination - ARD

1.12 变分法初步

微分学的一个最早的应用就是求极值, 先是一元实值函数的极值, 然后是多元实值函数的极值. 建立了无穷维赋范线性空间上的微分学, 当然会利用它去探讨无穷维赋范线性空间上的实值函数的极值问题.

定理 1.1 假设 E 是个赋范线性空间, U 是 E 的一个开集, 映射 $f : U \to R$ 在 U 上有直到 $(k-1)$ 阶的导数, 而在点 $\mathbf{x} \in U$ 处有 k 阶导数 $f^{(k)}(\mathbf{x}) \in \mathbf{L}(E, \cdots, E; R)$, 其中 $k \geqslant 2$. 又设 $f'(\mathbf{x}) = 0, \cdots, f^{(k-1)}(\mathbf{x}) = 0$, 而 $f^{(k)}(\mathbf{x}) \neq 0$, 确切些,

$$\forall \mathbf{h} \in E \forall j \in \{1, \cdots, k-1\} \left(f^{(j)} \mathbf{h}^j = 0 \right) \tag{1.90}$$

而

$$\mathbf{h} \in E \left(f^{(k)}(\mathbf{x}) \mathbf{h}^k \neq 0 \right) \tag{1.91}$$

则

(1) $\mathbf{x}$ 是函数 f 的极值点的必要条件是: k 是偶数, 且 $f^{(k)}(x)\mathbf{h}^k$ 不取相异的符号.

(2) x 是函数 f 的极值点的充分条件是: $f^{(k)}(\mathbf{x})\mathbf{h}^k$ 在单位球面 $|h| = 1$ 上与零保持一个正的距离. 若在单位球面上有不等式:

$$|h| = 1 \Longrightarrow f^{(k)}(\mathbf{x})\mathbf{h}^k \geqslant \delta > 0 \tag{1.92}$$

其中 δ 是个不依赖于 $\mathbf{h}$ 的正数, 则 $\mathbf{x}$ 是函数 f 的局部极小值点; 若在单位球面上有不等式

$$|h| = 1 \Longrightarrow f^{(k)}(\mathbf{x})\mathbf{h}^k \leqslant \delta < 0 \tag{1.93}$$

其中 δ 是个不依赖于 $\mathbf{h}$ 的负数, 则 $\mathbf{x}$ 是函数 f 的局部极大值点.

Proof 略. □

注 1 在定理 *1.1* 的条件下, $\mathbf{x}$ 是函数 f 的极值点的必要条件是: $f'(\mathbf{x}) = \mathbf{0}$.

注 2 定理 *1.1* 可以推广到 U 是 E 的一个仿射子空间的开集的情形. 下面我们将遇到这个情形.

例 1.1 先介绍 *Banach* 空间 $C^1(K, \mathbf{R})$ 的概念, 其中 K 是 $\mathbf{R}^n$ 中满足条件 $K = \bar{K^o}$ 的紧子集. $C^1(K, \mathbf{R})$ 表示定义在 K^o 上一次连续可微, 且导数可连续延拓至 K 上的实值函数全体. $C^1(K, \mathbf{R})$ 上的范数定义如下:

$$|f|_{C^1(k)} = \max\left\{|f|_{C(K)}, |\partial_j f|_{C(K)}, j = 1, \cdots, n\right\}. \tag{1.94}$$

不难证明, 如上定义的范数与以下定义的范数等价:

$$|f|'_{C^1(K)} = |f|_{C(K)} + \sum_{j=1}^{n} |\partial_j f|_{C(K)}. \tag{1.95}$$

我们可以证明, $C^1(K, \mathbf{R})$ 相对于如上定义的范数构成一个 *Banach* 空间.

设 $L \in C^1(\mathbf{R}^3, \mathbf{R})$ 和 $f \in C^1([, b], R)$. 映射 $F : C^1([a, b], \mathbf{R}) \to \mathbf{R}$ 如下:

$$F(f) = \int_a^b L\left(x, f(x), f'(x)\right) dx. \tag{1.96}$$

为了研究 F, 引进以下两个映射:

$$F_1 : C^1([, b], \mathbf{R}) \to C([a, b], \mathbf{R}), \quad F_1(x) = L(x, f(x), f'(x)) \tag{1.97}$$

和

$$F_2 : C^1([, b], \mathbf{R}) \to \mathbf{R}, \quad F_2(g) = \int_a^b g(x)dx. \tag{1.98}$$

显然, $F = F_2 \circ F_1$, 而且是连续线性映射.

我们先证明: F_1 可微, 且

$$F_1'(f)h(x) = \partial_2 L(x, f(x), f'(x))h(x) + \partial_3 L(x, f(x), f'(x))h'(x) \tag{1.99}$$

其中 ∂_2 和 ∂_3 分别表示对 L 的第二和第三个自变量的求偏导数运算. 证明略.

我们有

$$F_1'(f)h = \int_a^b [\partial_2 L(x, f(x), f'(x))h(x) + \partial_3 L(x, f(x), f'(x))h'(x)]dx. \tag{1.100}$$

常常遇到这样的极值问题, 我们要求 f 限制在 C^1 的这样的仿射子空间上:

$$\{f \in C^1([a,b], \mathbf{R}) : f(a) = A, f(b = B)\} \tag{1.101}$$

其中 A 和 B 是两个给定的常数. 当我们考虑限制在以上仿射子空间上的极值问题时, *(1.100)* 中的 h 应满足条件:

$$f(a) = f(a) + h(a), \quad f(b) = f(b) + h(b). \tag{1.102}$$

换言之, h 应满足条件

$$h(a) = h(b) = 0 \tag{1.103}$$

这时, 假若 $L^2(\mathbf{R}^3, \mathbf{R})$, 通过一次分部积分, *(1.100)* 便可改写成

$$F'(f)h = \int_a^b \left[\partial_2 L\left(x, f(x), f'(x)\right) - \frac{d}{dx}\partial_3 L\left(x, f(x), f'(x)\right)\right] h(x)dx \tag{1.104}$$

由定理 *1.1* , 在 *(1.102)* 的条件下的 F 的极值问题的解应满足条件: 对于任何满足条件 *(1.103)* 的 C^1 中的函数 h, 有

$$F'(f)h = \int_a^b \left[\partial_2 L\left(x, f(x), f'(x)\right) - \frac{d}{dx}\partial_3 L\left(x, f(x), f'(x)\right)\right] h(x)dx = 0 \tag{1.105}$$

由此，根据下面的 *Du Bois Reymond* 引理，f 应满足以下的方程，它称为 ***Euler-Lagrange*** 方程*:*

$$\partial_2 L\left(x, f(x), f'(x)\right) - \frac{d}{dx}\partial_3 L\left(x, f(x), f'(x)\right) = 0 \tag{1.106}$$

引理 1.1 *(**Du Bois Reymond** 引理)* 设 $\phi \in C\left([a,b], \mathbf{R}\right)$, :

$$\forall h \in C^\infty \left(h(a) = h(b) = 0 \Longrightarrow \int_a^b \phi(x)h(x)dx = 0\right), \tag{1.107}$$

则 $\forall x \in [a,b](\phi(x) = 0)$.

Proof 略. □

1.13 Exercises

2 Probability Distributions

```
import numpy as np
from numpy.linalg import norm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import pandas as pd
from scipy import linalg
from scipy.special import i0, i1
from scipy.stats import dirichlet
from scipy.stats import beta
from scipy.stats import uniform
from scipy.stats import norm as normal
from scipy.stats import multivariate_normal
from scipy.stats import gamma
from scipy.stats import t
from scipy.stats import vonmises
from scipy.stats import vonmises_line
from sklearn.mixture import GaussianMixture
from matplotlib.colors import LogNorm


def Rot(x):
    theta_ = x*np.pi/180
    return np.array([[np.cos(theta_), -np.sin(theta_)], [np.sin(theta_), np.cos(theta_)]])
```

Listing 2.1: ch02-init

In Chapter 1, we emphasized the central role played by probability theory in the solution of pattern recognition problems. We turn now to an exploration of some particular examples of probability distributions and their properties. As well as being of great interest in their own right, these distributions can form building blocks for more complex models and will be used extensively throughout the book. The distributions introduced in this chapter will also serve another important purpose, namely to provide us with

the opportunity to discuss some key statistical concepts, such as Bayesian inference, in the context of simple models before we encounter them in more complex situations in later chapters.

One role for the distributions discussed in this chapter is to model the probability distribution $p(\mathbf{x})$ of a random variable x, given a finite set $\mathbf{x}_1, \cdots, \mathbf{x}_N$ of observations. This problem is known as *density estimation(密度估计)*. For the purposes of this chapter, we shall assume that the data points are independent and identically distributed. It should be emphasized that the problem of density estimation is fundamentally ill-posed, because there are infinitely many probability distributions that could have given rise to the observed finite data set. Indeed, any distribution $p(\mathbf{x})$ that is nonzero at each of the data points $\mathbf{x}_1, \cdots, \mathbf{x}_N$ is a potential candidate. The issue of choosing an appropriate distribution relates to the problem of model selection that has already been encountered in the context of polynomial curve fitting in Chapter 1 and that is a central issue in pattern recognition.

We begin by considering the binomial and multinomial distributions for discrete random variables and the Gaussian distribution for continuous random variables. These are specific examples of *parametric distributions(参数分布)*, so-called because they are governed by a small number of adaptive parameters, such as the mean and variance in the case of a Gaussian for example. To apply such models to the problem of density estimation, we need a procedure for determining suitable values for the parameters, given an observed data set. In a frequentist treatment, we choose specific values for the parameters by optimizing some criterion, such as the likelihood function. By contrast, in a Bayesian treatment we introduce prior distributions over the parameters and then use Bayes'theorem to compute the corresponding posterior distribution given the observed data.

> 在频率学家的观点中, 我们通过最优化某些准则 (例如似然函数) 来确定参数的具体值. 相反, 在贝叶斯观点中, 给定观察数据, 我们引入参数的先验分布, 然后使用贝叶斯定理来计算对应后验概

率分布。

We shall see that an important role is played by *conjugate priors(共轭先验)*, that lead to posterior distributions having the same functional form as the prior, and that therefore lead to a greatly simplified Bayesian analysis. For example, the conjugate prior for the parameters of the multinomial distribution is called the *Dirichlet distribution(狄利克雷分布)*, while the conjugate prior for the mean of a Gaussian is another Gaussian. All of these distributions are examples of the exponential family of distributions, which possess a number of important properties, and which will be discussed in some detail.

One limitation of the parametric approach is that it assumes a specific functional form for the distribution, which may turn out to be inappropriate for a particular application. An alternative approach is given by *non-parametric(非参数)* density estimation methods in which the form of the distribution typically depends on the size of the data set. Such models still contain parameters, but these control the model complexity rather than the form of the distribution. We end this chapter by considering three nonparametric methods based respectively on histograms, nearest-neighbors, and kernels.

2.1 Binary Variables

We begin by considering a single binary random variable $x \in \{0, 1\}$. For example, x might describe the outcome of flipping a coin, with $x = 1$ representing 'heads', and $x = 0$ representing 'tails'. We can imagine that this is a damaged coin so that the probability of landing heads is not necessarily the same as that of landing tails. The probability of $x = 1$ will be denoted by the parameter μ so that

$$p(x = 1|\mu) = \mu \tag{2.1}$$

where $0 \leqslant \mu \leqslant 1$, from which it follows that $p(x = 0|\mu) = 1-\mu$. The probability distribution over x can therefore be written in the form

$$\text{Bern}(x|\mu) = \mu^x(1-\mu)^{1-x} \tag{2.2}$$

which is known as the Bernoulli distribution. It is easily verified that this distribution is normalized and that it has mean and variance given by

$$\mathbb{E}[x] = \mu \tag{2.3}$$

$$var[x] = \mu(1-\mu) \tag{2.4}$$

Now suppose we have a data set $\mathcal{D} = \{x_1, \cdots, x_N\}$ of observed values of x. We can construct the likelihood function, which is a function of μ, on the assumption that the observations are drawn independently from $p(x|\mu)$, so that

$$p(\mathcal{D}|\mu) = \prod_{n=1}^{N} p(x_i|\mu) = \prod_{n=1}^{N} \mu^{x_n}(1-\mu)^{1-x_n} \tag{2.5}$$

In a frequentist setting, we can estimate a value for μ by maximizing the likelihood function, or equivalently by maximizing the logarithm of the likelihood. In the case of the Bernoulli distribution, the log likelihood function is given by

$$\ln p(\mathcal{D}|\mu) = \sum_{i=1}^{N} \ln p(x_n|\mu) = \sum_{i=1}^{N} \{x_n \ln \mu + (1-x_n)\ln(1-\mu)\} \tag{2.6}$$

At this point, it is worth noting that the log likelihood function depends on the N observations x_n only through their sum $\sum_n x_n$. This sum provides an example of a *sufficient statistic(充分统计量)* for the data under this distribution, and we shall study the important role of sufficient statistics in some detail. If we set the derivative of $\ln p(\mathcal{D}|\mu)$ with respect to μ equal to zero, we obtain the maximum likelihood estimator

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{2.7}$$

which is also known as the *sample mean(样本均值)*. If we denote the number of observations of $x = 1$ (heads) within this data set by m, then we can write (2.1.1) in the form

$$\mu_{ML} = \frac{m}{N} \tag{2.8}$$

so that the probability of landing heads is given, in this maximum likelihood framework, by the fraction of observations of heads in the data set.

Now suppose we flip a coin, say, 3 times and happen to observe 3 heads. Then $N = m = 3$ and $\mu_{ML} = 1$. In this case, the maximum likelihood result would predict that all future observations should give heads. Common sense tells us that this is unreasonable, and in fact this is an extreme example of the over-fitting associated with maximum likelihood. We shall see shortly how to arrive at more sensible conclusions through the introduction of a prior distribution over μ.

We can also work out the distribution of the number m of observations of $x = 1$, given that the data set has size N. This is called the *binomial distribution(二项分布)*, and from (2.6) we see that it is proportional to $\$\mu^{\mathrm{m}}(1\ \ \mu)^{\mathrm{Nm}}$. In order to obtain the normalization coefficient we note that out of N coin flips, we have to add up all of the possible ways of obtaining m heads, so that the binomial distribution can be written

$$\mathrm{Bin}(m|N,\mu) = \binom{N}{m}\mu^m(1-\mu)^{N-m} \tag{2.9}$$

where

$$\binom{N}{m} = \frac{N!}{(N-m)!m!} \tag{2.10}$$

is the number of ways of choosing m objects out of a total of N identical objects. Figure 2.1 shows a plot of the binomial distribution for $N = 10$ and $\mu = 0.25$.

```
from scipy.stats import binom
fig, ax = plt.subplots(1, 1)
n, p = 10, 0.25
x = np.arange(11)
y = binom.pmf(x, n, p)
ax.bar(x,y,color="b")
ax.set_xlabel("$m$")
plt.savefig("img/fig:2.1.png")
```

Listing 2.2: fig:2.1

The mean and variance of the binomial distribution can be found by using the result of Exercise 1.10(?), which shows that for independent events the mean of the sum is the sum of the means, and the variance of the sum is the sum of the variances. Because $m = x_1 + \cdots + x_N$, and for each observation the mean and variance are given by (2.3) and (2.4), respectively, we have

$$\mathbb{E}[m] \equiv \sum_{m=0}^{N} m\mathrm{Bin}(m|N,\mu) = N\mu \tag{2.11}$$

$$var[m] \equiv \sum_{m=0}^{N} (m - \mathbb{E}[m])^2 \mathrm{Bin}(m|N,\mu) = N\mu(1-\mu) \tag{2.12}$$

These results can also be proved directly using calculus.

2.1.1 TODO The beta distribution

We have seen in (**??**) that the maximum likelihood setting for the parameter μ in the Bernoulli distribution, and hence in the binomial distribution, is given by the fraction of the observations in the data set having x = 1. As we have already noted, this can give severely over-fitted results for small data sets. In order to develop a Bayesian treatment for this problem,

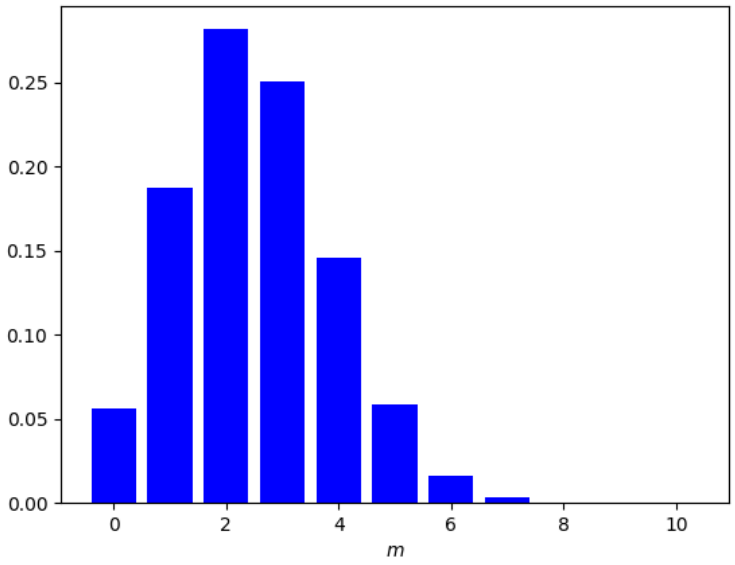


图 2.1: Histogram plot of the binomial distribution (2.9) as a function of m for $N = 10$ and $\mu = 0.25$.

we need to introduce a prior distribution $p(\mu)$ over the parameter μ. Here we consider a form of prior distribution that has a simple interpretation as well as some useful analytical properties. To motivate this prior, we note that the likelihood function takes the form of the product of factors of the form $\mu^x(1-\mu)^{1-x}$. If we choose a prior to be proportional to powers of μ and $(1-\mu)$, then the posterior distribution, which is proportional to the product of the prior and the likelihood function, will have the same functional form as the prior. This property is called *conjugacy(共轭性)* and we will see several examples of it later in this chapter. We therefore choose a prior, called the *beta* distribution, given by

$$\text{Beta}(\mu|a,b) = \frac{\Gamma(a+b)}{\Gamma a\Gamma(b)}\mu^{a-1}(1-\mu)^{b-1} \tag{2.13}$$

where $\gamma(x)$ is the gamma function defined by

$$\Gamma(x) \equiv \int_0^\infty u^{x-1}e^{-u}du \tag{2.14}$$

and the coefficient in (2.13) ensures that the beta distribution is normalized, so that

$$\int_0^1 \text{Beta}(\mu|a,b) = 1 \tag{2.15}$$

The mean and variance of the beta distribution are given by

$$\mathbb{E}[x] = \frac{a}{a+b} \tag{2.16}$$

$$var[\mu] = \frac{ab}{(a+b)^2(a+b+1)} \tag{2.17}$$

The parameters a and b are often called *hyperparameters(超参数)* because they control the distribution of the parameter μ. Figure 2.2 shows plots of the beta distribution for various values of the hyperparameters.

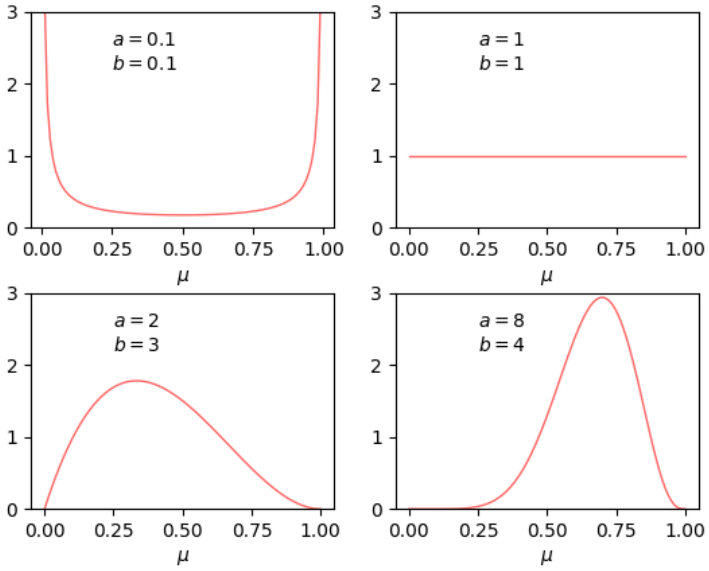


图 2.2: Plots of the beta distribution $\text{Beta}(\mu|a, b)$ given by (2.13) as a function of μ for various values of the hyperparameters a and b.

```
def create_beta_plot(a, b, ax, *args):
    x = np.linspace(0, 1, 100)
    ax[args].plot(x, beta.pdf(x, a, b), 'r-', lw=1, alpha=0.6)
    ax[args].annotate("$a={0}$\n$b={1}$".format(a, b), xy=(0.25, 2.2))
    ax[args].set_ylim(0, 3)
    ax[args].set_xlabel("$\mu$")


fig, ax = plt.subplots(2, 2)
create_beta_plot(0.1, 0.1, ax, 0, 0)
create_beta_plot(1, 1, ax, 0, 1)
create_beta_plot(2, 3, ax, 1, 0)
create_beta_plot(8, 4, ax, 1, 1)
fig.subplots_adjust(hspace=0.3)
fig.savefig("img/fig:2.2.png")
plt.close("all")
```

Listing 2.3: fig:2.2

The posterior distribution of μ is now obtained by multiplying the beta prior (2.13) by the binomial likelihood function (2.9) and normalizing. Keeping only the factors that depend on μ, we see that this posterior distribution has the form

$$p(\mu|m,l,a,b) \propto \mu^{m+a-1}(1-\mu)^{l+b-1} \tag{2.18}$$

where $l = N-m$, and therefore corresponds to the number of 'tails'in the coin example. We see that (2.18) has the same functional dependence on μ as the prior distribution, reflecting the conjugacy properties of the prior with respect to the likelihood function. Indeed, it is simply another beta distribution, and its normalization coefficient can therefore be obtained by comparison with (2.13) to give

$$p(\mu|m,l,a,b) = \frac{\Gamma(m+a+b+l)}{\Gamma(m+a)\Gamma(l+b)}\mu^{m+a-1}(1-\mu)^{l+b-1} \tag{2.19}$$

We see that the effect of observing a data set of m observations of $x = 1$ and l observations of $x = 0$ has been to increase the value of a by m, and the value of b by l, in going from the prior distribution to the posterior distribution. This allows us to provide a simple interpretation of the hyper-parameters a and b in the prior as an *effective number of observations(有效观测数)* of $x = 1$ and $x = 0$, respectively. Note that a and b need not be integers. Furthermore, the posterior distribution can act as the prior if we subsequently observe additional data. To see this, we can imagine taking observations one at a time and after each observation updating the current posterior distribution by multiplying by the likelihood function for the new observation and then normalizing to obtain the new, revised posterior distribution. At each stage, the posterior is a beta distribution with some total number of (prior and actual) observed values for $x = 1$ and $x = 0$ given by the parameters a and b. Incorporation of an additional observation of $x = 1$ simply corresponds to incrementing the value of a by 1, whereas for an observation of $x = 0$ we increment b by 1. Figure 2.3 illustrates one step in this process.

```
fig = plt.figure(figsize=(12, 3))
ax = fig.subplots(1, 3)
a, b = 2, 2
x = np.linspace(0, 1, 100)
y_plot = [beta.pdf(x, a, b), x, beta.pdf(x, a + 1, b)]
for index, (y, annotation) in enumerate(
        zip(y_plot, ["prior", "likelihood", "posterior"])):
    ax[index].set_ylim(0, 2)
    ax[index].set_xlim(0, 1)
    ax[index].set_ylim(0, 2)
    ax[index].plot(x, y, 'r-', lw=1, alpha=0.6)
    ax[index].annotate(annotation, xy=(0.1, 1.6))
fig.savefig("img/fig:2.3.png")
plt.close("all")
```

Listing 2.4: fig:2.3

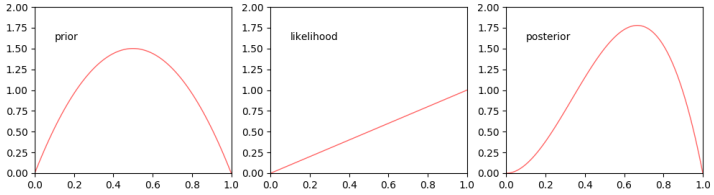


图 2.3: Illustration of one step of sequential Bayesian inference. The prior is given by a beta distribution with parameters $a = 2, b = 2$, and the likelihood function, given by (2.9) with $N = m = 1$, corresponds to a single observation of $x = 1$, so that the posterior is given by a beta distribution with parameters $a = 3, b = 2$.

We see that this *sequential(顺序)* approach to learning arises naturally when we adopt a Bayesian viewpoint. It is independent of the choice of prior and of the likelihood function and depends only on the assumption of i.i.d. data. Sequential methods make use of observations one at a time, or in small batches(批次), and then discard them before the next observations are used. They can be used, for example, in real-time learning scenarios where a steady stream of data is arriving, and predictions must be made before all of the data is seen. Because they do not require the whole data set to be stored or loaded into memory, sequential methods are also useful for large data sets. Maximum likelihood methods can also be cast into a sequential framework.

If our goal is to predict, as best we can, the outcome of the next trial, then we must evaluate the predictive distribution of x, given the observed data set D. From the sum and product rules of probability, this takes the form

$$p(x = 1|\mathcal{D}) = \int_0^1 p(x = 1|\mu)p(\mu|\mathcal{D})d\mu = \int_0^1 \mu p(\mu|\mathcal{D})d\mu = \mathbb{E}[\mu|\mathcal{D}] \quad (2.20)$$

Using the result (2.19) for the posterior distribution $p(\mu|\mathcal{D})$, together

with the result (2.16) for the mean of the beta distribution, we obtain

$$p(x = 1|\mathcal{D}) = \frac{m + a}{m + a + l + b} \tag{2.21}$$

which has a simple interpretation as the total fraction of observations (both real observations and fictitious prior observations) that correspond to $x = 1$. Note that in the limit of an infinitely large data set $m, l \to \infty$ the result (2.21) reduces to the maximum likelihood result (2.8). As we shall see, it is a very general property that the **Bayesian and maximum likelihood results will agree in the limit of an infinitely large data set**. For a finite data set, **the posterior mean for μ always lies between the prior mean and the maximum likelihood estimate** for μ corresponding to the relative frequencies of events given by ().

From Figure 2.2, we see that as the number of observations increases, so the posterior distribution becomes more sharply peaked. This can also be seen from the result (2.17) for the variance of the beta distribution, in which we see that the variance goes to zero for $a \to \infty$ or $b \to \infty$. In fact, we might wonder whether it is a general property of Bayesian learning that, as we observe more and more data, the uncertainty represented by the posterior distribution will steadily decrease.

> To address this, we can take a frequentist view of Bayesian learning and show that, on average, such a property does indeed hold. Consider a general Bayesian inference problem for a parameter θ for which we have observed a data set D, described by the joint distribution $p(, D). The follow result
>
> $$\mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\theta}] = \mathbb{E}_{\mathcal{D}}[\mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\theta}|\mathcal{D}]] \tag{2.22}$$
>
> where

says that the posterior mean of , averaged over the distribution generating the data,is equal to the prior mean of . Similarly, we can show that

(2.23)

2.2 Multinomial Variables

Binary variables can be used to describe quantities that can take one of two possible values. Often, however, we encounter discrete variables that can take on one of K possible mutually exclusive states. Although there are various alternative ways to express such variables, we shall see shortly that a particularly convenient representation is the 1-of-K scheme in which the variable is represented by a K-dimensional vector x in which one of the elements x_k equals 1, and all remaining elements equal 0. So, for instance if we have a variable that can take $K = 6$ states and a particular observation of the variable happens to correspond to the state where $x_3 = 1$, then x will be represented by

$$\mathbf{x} = (0, 0, 1, 0, 0, 0)^T. \tag{2.24}$$

Note that such vectors satisfy $\sum_{k=1}^{K} x_k = 1$. If we denote the probability of $x_k = 1$ by the parameter μ_{k}, then distribution of x is given

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{k=1}^{K} \mu^{x_k} \tag{2.25}$$

where $\boldsymbol{\mu} = (\mu_1, \cdots, \mu_K)^T$, and the parameters μ_{k} are constrained to satisfy $\mu_k \geqslant 0$ and $\sum_k \mu_k = 1$, because they represent probabilities. The distribution (2.25) can be regarded as a generalization of the Bernoulli distribution to more than two outcomes. It is easily seen that the distribution is normalized

$$\sum_{\mathbf{x}} p(\mathbf{x}|\boldsymbol{\mu}) = \sum_{k=1}^{K} \mu_k = 1 \tag{2.26}$$

and that

$$\mathbb{E}[\mathbf{x}|\boldsymbol{\mu}] = \sum_{\mathbf{x}} p(\mathbf{x}|\boldsymbol{\mu})\mathbf{x} = (\mu_1, \cdots, \mu_M)^T = \boldsymbol{\mu} \tag{2.27}$$

Now consider a data set D of N independent observations $x_1, \cdots, x_N$. The corresponding likelihood function takes the form

$$p(\mathcal{D}|\boldsymbol{\mu}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \mu_k^{x_{nk}} = \prod_{k=1}^{K} \mu_K^{(\sum_n x_{nk})} = \prod_{k=1}^{K} \mu_k^{m_k} \tag{2.28}$$

We see that the likelihood function depends on the N data points only through the K quantities

$$m_k = \sum_n x_{nk} \tag{2.29}$$

which represent the number of observations of $x_k = 1$. These are called the *sufficient statistics(充分统计量)* for this distribution.

In order to find the maximum likelihood solution for $\boldsymbol{\mu}$, we need to maximize $\ln p(\mathcal{D}|\boldsymbol{\mu})$ with respect to μ_k taking account of the constraint that the μ_k must sum to one. This can be achieved using a Lagrange multiplier λ and maximizing

$$\sum_{k=1}^{K} m_k \ln \mu_k + \lambda \left(\sum_{k=1}^{K} \mu_k - 1 \right) \tag{2.30}$$

Setting the derivative of (2.30) with respect to μ_k to zero, we obtain

$$\mu_k = -m_k/\lambda \tag{2.31}$$

We can solve for the Lagrange multiplier λ by substituting (2.31) into the constraint $\sum_k \mu_k = 1$ to give $\lambda = -N$. Thus we obtain the maximum likelihood solution in the form

$$\mu_k^{MK} = \frac{m_k}{N} \tag{2.32}$$

which is the fraction of the N observations for which $x_k = 1$.

We can consider the joint distribution of the quantities $m1, \cdots, m_K$, conditioned on the parameters $\boldsymbol{\mu}$ and on the total number N of observations. From (2.28) this takes the form

$$\text{Mult}(m_1, m_2, \cdots, m_k|\boldsymbol{\mu}, N) = \binom{N}{m_1 m_2 \cdots m_K} \prod_{k=1}^{K} \mu_k^{m_k} \tag{2.33}$$

which is known as the *multinomial distribution(多项式分布)*. The normalization coefficient is the number of ways of partitioning N objects into K groups of size $m_1, \cdots, m_K$ and is given by

$$\binom{N}{m_1 m_2 \cdots m_K} = \frac{N!}{m_1! m_2! \cdots_k!}. \tag{2.34}$$

Note that the variables m_k are subject to the constraint

$$\sum_{k=1}^{K} m_k = N \tag{2.35}$$

2.2.1 The Dirichlet distribution

We now introduce a family of prior distributions for the parameters $\{_k\}$ of the multinomial distribution (2.34). By inspection of the form of the multinomial distribution, we see that the conjugate prior is given by

$$p(\boldsymbol{\mu}|\boldsymbol{\alpha}) \propto \prod_{k=1}^{K} \mu_k^{(\alpha_k - 1)} \tag{2.36}$$

where $0 \leqslant \mu_k 1$ and $\sum_k \mu_k = 1$. Here $\alpha_1, \cdots, \alpha_K$ are the parameters of the distribution, and denotes $(\alpha_1, \cdots, \alpha_K)^T$. Note that, because of the summation constraint, the distribution over the space of the $\{\mu_k\}$ is confined to a simplex of dimensionality $K-1$, as illustrated for $K = 3$ in Figure 2.4.

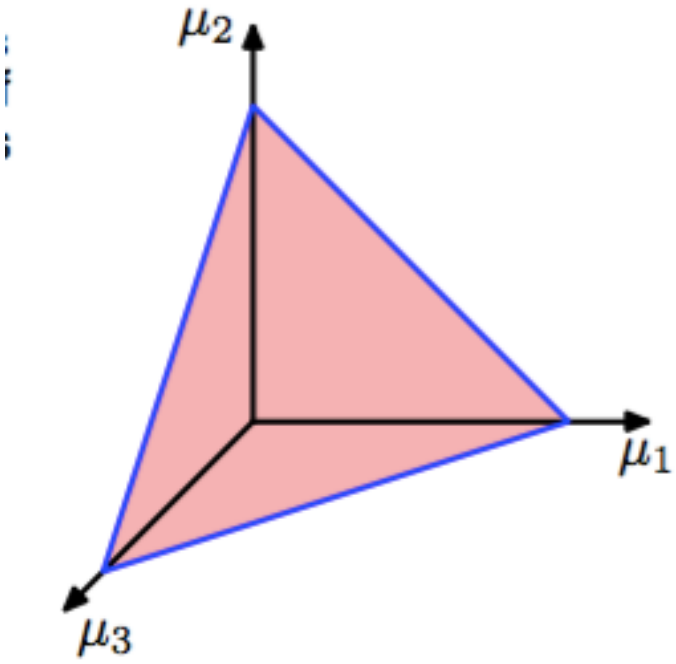


图 2.4: The Dirichlet distribution over three variables μ_1, μ_2, μ_3 is confined to a simplex (a bounded linear manifold) of the form shown, as a consequence of the constraints $0 \leqslant \mu_k \leqslant 1$ and $\sum_k \mu_k = 1$.

The normalized form for this distribution is by

$$\mathrm{Dir}(\boldsymbol{\mu}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_1), \cdots, \Gamma(\alpha_K)} \prod_{k=1}^{K} \mu_k^{\alpha_k - 1} \tag{2.37}$$

which is called the *Dirichlet distribution(狄利克雷分布)*. Here $\Gamma(x)$ is the gamma function defined by (2.14) while

$$\alpha_0 = \sum_{k=1}^{K} \alpha_k \tag{2.38}$$

Plots of the Dirichlet distribution over the simplex, for various settings of the parameters α_{k}, are shown in Figure 2.5.

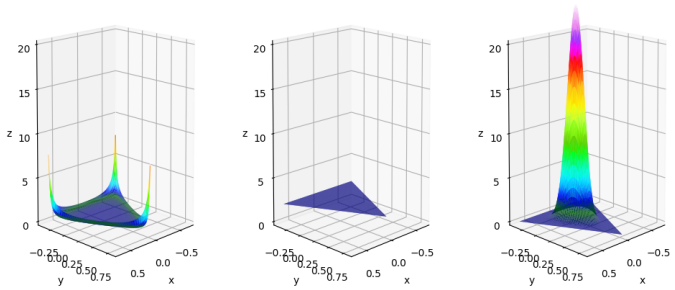


图 2.5: Plots of the Dirichlet distribution over three variables, where the two horizontal axes are coordinates in the plane of the simplex and the vertical axis corresponds to the value of the density. Here $\{\alpha_k\} = 0.1$ on the left plot, $\{\alpha_k\} = 1$ in the centre plot, and $\{\alpha_k\} = 10$ in the right plot.

Multiplying the prior (2.37) by the likelihood function (2.33), we obtain the posterior distribution for the parameters $\{\mu_k\}$ in the form

$$p(\boldsymbol{\mu}|\mathcal{D}, \boldsymbol{\alpha}) \propto p(\mathcal{D}|\boldsymbol{\mu}) p(\boldsymbol{\mu}|\boldsymbol{\alpha}) \propto \prod_{k=1}^{K} \mu_k^{\alpha_k + m_k - 1} \tag{2.39}$$

We see that the posterior distribution again takes the form of a Dirichlet distribution, confirming that the Dirichlet is indeed a conjugate prior for the multinomial. This allows us to determine the normalization coefficient by comparison with (2.37) so that

```
A = np.array([[-1, 1, 0], [-0.5, -0.5, 1]])
A[0] = A[0] / linalg.norm(A[0])
A[1] = A[1] / linalg.norm(A[1])
x_1 = np.arange(1, 100 - 1)
x_2 = np.zeros((98, 98))
for index, value in enumerate(x_1):
    x_2[index, 0:(98 - index)] = np.arange(1, 100 - value)
x = np.array([[0, 0, 0]])
for index1, value1 in enumerate(x_1):
    for index2, value2 in enumerate(x_2[index1, 0:(98 - index1)]):
        x = np.concatenate((x,
                            np.array([[value1, value2,
                                       100 - value1 - value2]])))
x = x[1::] / 100
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

fig = plt.figure(figsize=(12, 6))
Alpha_ = [[0.1] * 3, [1] * 3, [10] * 3]
for index, alpha_ in enumerate(Alpha_):
    value = [dirichlet.pdf(_, alpha_) for _ in x]
    y0, y1 = [(A @ _)[0] for _ in x], [(A @ _)[1] for _ in x]
    data = pd.DataFrame({"value": value, "y0": y0, "y1": y1})
    ax = fig.add_subplot(1, 3, index + 1, projection='3d')
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("z")
    ax.set_zlim(0, 20)
    ax.view_init(10, 45)
    im = ax.plot_trisurf(
        data["y0"], data["y1"], data["value"], cmap="gist_ncar")
plt.savefig("img/fig:2.5.png")
plt.close("all")
```

Listing 2.5: fig:2.5

$$p(\boldsymbol{\mu}|\mathcal{D},\boldsymbol{\alpha}) = \mathrm{Dir}(\boldsymbol{\mu}|\boldsymbol{\alpha}+\mathbf{m}) = \frac{\Gamma(\alpha_0+N)}{\Gamma(\alpha_0+m_1)\cdots\Gamma(\alpha_0+m_K)}\prod_{k=1}^{K}\mu_k^{\alpha_k+m_k-1} \tag{2.40}$$

where we have denoted $m = (m_1, \cdots, m_K)^T$. As for the case of the binomial distribution with its beta prior, we can interpret the parameters α_k of the Dirichlet prior as an effective number of observations of $x_k = 1$.

Note that two-state quantities can either be represented as binary variables and modelled using the binomial distribution (2.9) or as 1-of-2 variables and modelled using the multinomial distribution (2.33) with $K = 2$.

2.3 The Gaussian Distribution

The Gaussian, also known as the normal distribution, is a widely used model for the distribution of continuous variables. In the case of a single variable x, the Gaussian distribution can be written in the form

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}}\exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} \tag{2.41}$$

where μ is the mean and σ^2 is the variance. For a D-dimensional vector $\mathbf{x}$, the multivariate Gaussian distribution takes the form

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\exp\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\} \tag{2.42}$$

where $\boldsymbol{\mu}$ is a D-dimensional mean vector, $\boldsymbol{\Sigma}$ is a $D\times D$ covariance matrix, and $|\boldsymbol{\Sigma}|$ denotes the determinant of $\boldsymbol{\Sigma}$.

The Gaussian distribution arises in many different contexts and can be motivated from a variety of different perspectives. For example, we have already seen that for a single real variable, the distribution that maximizes the entropy is the Gaussian. This property applies also to the multivariate Gaussian.

Another situation in which the Gaussian distribution arises is when we consider the sum of multiple random variables. The *central limit theorem(中心极限定理)* (due to Laplace) tells us that, subject to certain mild(温和的) conditions, the sum of a set of random variables, which is of course itself a random variable, has a distribution that becomes increasingly Gaussian as the number of terms in the sum increases (Walker, 1969). We can illustrate this by considering N variables $x_1, \cdots, x_N$ each of which has a uniform distribution over the interval $[0, 1]$ and then considering the distribution of the mean $(x_1 + \cdots + x_N)/N$. For large N , this distribution tends to a Gaussian, as illustrated in Figure [fig:2.6]]. In practice, the convergence to a Gaussian as N increases can be very rapid. One consequence of this result is that the binomial distribution (2.9), which is a distribution over m defined by the sum of N observations of the random binary variable x, will tend to a Gaussian as $N \to \infty$ (see Figure 2.1 for the case of $N = 10$).

```
from pacal import *
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(12,4))
U = UniformDistr()
for index, N in enumerate([1,2,10]):
    ax = fig.add_subplot(1,3,index+1)
    ax.set_ylim(0,5)
    ax.set_xlim(0,1)
    S = iid_average(U,N)
    S.hist(bins=20,color="blue", edgecolor="b")
    ax.annotate("$N=%d$"%N, xy=(0.1,4))
plt.savefig("img/fig:2.6.png")
plt.close("all")
```

Listing 2.6: fig:2.6(python2)

The Gaussian distribution has many important analytical properties, and we shall consider several of these in detail. As a result, this section will be rather more technically involved than some of the earlier sections, and will require familiarity with various matrix identities. However, we strongly

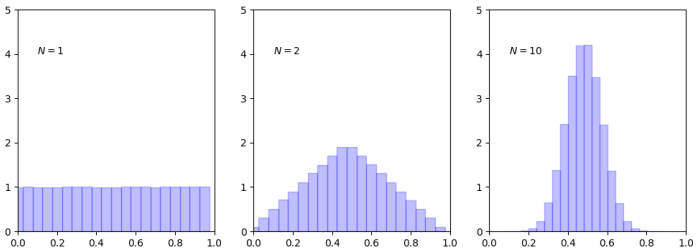


图 2.6: Histogram plots of the mean of N uniformly distributed numbers for various values of N. We observe that as N increases, the distribution tends towards a Gaussian.

encourage the reader to become proficient(精通的) in manipulating Gaussian distributions using the techniques presented here as this will prove invaluable in understanding the more complex models presented in later chapters.

We begin by considering the geometrical form of the Gaussian distribution. The functional dependence of the Gaussian on x is through the quadratic form

$$\Delta^2 = (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \tag{2.43}$$

which appears in the exponent. The quantity Δ is called the *Mahalanobis distance(马氏距离)* from $\boldsymbol{\mu}$ to $\mathbf{x}$ and reduces to the Euclidean distance when Σ is the identity matrix. The Gaussian distribution will be constant on surfaces in x-space for which this quadratic form is constant.

First of all, we note that the matrix Σ can be taken to be symmetric, without loss of generality, because any antisymmetric component would disappear from the exponent. Now consider the eigenvector equation for the covariance matrix

$$\Sigma \mathbf{u}_i = \lambda_i \mathbf{u}_i \tag{2.44}$$

where $i = 1, \cdots, D$. Because Σ is a real, symmetric matrix its eigenval-

ues will be real, and its eigenvectors can be chosen to form an orthonormal set, so that

$$\mathbf{u}_i^T \mathbf{u}_j = I_{ij} \tag{2.45}$$

where I_{ij} is the i, j element of the identity matrix and satisfies

$$I_i j = \begin{cases} 1, & \text{if} \quad i = j \\ 0, & \text{otherwise} \end{cases} \tag{2.46}$$

The covariance matrix Σ can be expressed as an expansion in terms of its eigenvectors in the form

$$\Sigma = \sum_{i=1}^{D} \lambda_i \mathbf{u}_i \mathbf{u}_i^T \tag{2.47}$$

and similarly the inverse covariance matrix Σ^{-1} can be expressed as

$$\Sigma^{-1} = \sum_{i=1}^{D} \frac{1}{\lambda_i} \mathbf{u}_i \mathbf{u}_i^T \tag{2.48}$$

Substituting (2.48) into (2.43), the quadratic form becomes

$$\Delta^2 = \sum_{i=1}^{D} \frac{y_i^2}{\lambda_i} \tag{2.49}$$

where we have defined

$$y_i = \mathbf{u}_i (\mathbf{x} - \boldsymbol{\mu}) \tag{2.50}$$

We can interpret $\{y_i\}$ as a new coordinate system defined by the orthonormal vectors u_i that are shifted and rotated with respect to the original x_i coordinates. Forming the vector $\mathbf{y} = (y_1, \cdots, y_D)^T$, we have

$$\mathbf{y} = \mathbf{U}(\mathbf{x} - \boldsymbol{\mu}) \tag{2.51}$$

where $\mathbf{U}$ is a matrix whose rows are given by u_i^T. From (2.45) it follows that $\mathbf{U}$ is an orthogonal matrix, i.e., it satisfies $\mathbf{U}\mathbf{U}^T = \mathbf{I}$, and hence also $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix.

The quadratic form, and hence the Gaussian density, will be constant on surfaces for which (2.50) is constant. If all of the eigenvalues λ_i are positive, then these surfaces represent ellipsoids, with their centers at μ and their axes oriented along u_i, and with scaling factors in the directions of the axes given by $\$\lambda_{\mathrm{i}}^{1/2}$, as illustrated in Figure 2.7.

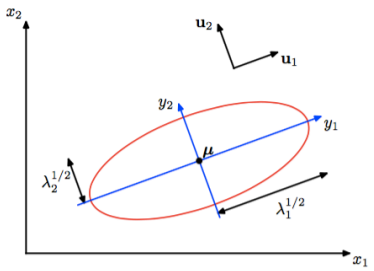


图 2.7: The red curve shows the elliptical surface of constant probability density for a Gaussian in a two-dimensional space $\mathbf{x} = (x_1, x_2)$ on which the density is $\exp(-1/2)$ of its value at $x = \mu$. The major axes of the ellipse are defined by the eigenvectors u_i of the covariance matrix, with corresponding eigenvalues λ_i.

For the Gaussian distribution to be well defined, it is necessary for all of the eigenvalues λ_i of the covariance matrix to be strictly positive, otherwise the distribution cannot be properly normalized. A matrix whose eigenvalues are strictly positive is said to be *positive definite(正定)*. In Chapter 12(?), we will encounter Gaussian distributions for which one or more of the eigenvalues are zero, in which case the distribution is singular and is confined to a subspace of lower dimensionality. If all of the eigenvalues are nonnegative, then the covariance matrix is said to be *positive semidefinite(半正定)*.

Now consider the form of the Gaussian distribution in the new coor-

dinate system defined by the y_i. In going from the $\mathbf{x}$ to the $\mathbf{y}$ coordinate system, we have a Jacobian matrix $\mathbf{J}$ with elements given by

$$J_{ij} = \frac{\partial x_i}{\partial y_j} = U_{ji} \tag{2.52}$$

where U_{ji} are the elements of the matrix $\mathbf{U}^T$. Using the orthonormality property of the matrix $\mathbf{U}$, we see that the square of the determinant of the Jacobian matrix is

$$|\mathbf{J}|^2 = |\mathbf{U}^T|^2 = |\mathbf{U}^T||\mathbf{U}| = |\mathbf{U}^T\mathbf{U}| = |\mathbf{I}| = 1 \tag{2.53}$$

and hence $|\mathbf{J}| = 1$. Also, the determinant $|\Sigma|$ of the covariance matrix can be written as the product of its eigenvalues, and hence

$$|\Sigma|^{1/2} = \prod_{j=1}^{D} \lambda_j^{1/2}. \tag{2.54}$$

Thus in the y_j coordinate system, the Gaussian distribution takes the form

$$p(\mathbf{y}) = p(\mathbf{x})|\mathbf{J}| = \prod_{j=1}^{D} \frac{1}{(2\pi\lambda_j)^{1/2}} \exp\left\{-\frac{y_i^2}{2\lambda_i}\right\} \tag{2.55}$$

which is the product of D independent univariate Gaussian distributions. The eigenvectors therefore define a new set of shifted and rotated coordinates with respect to which the joint probability distribution factorizes into a product of independent distributions. The integral of the distribution in the $\mathbf{y}$ coordinate system is then

$$\int p(\mathbf{y}) d\mathbf{y} = \prod_{j=1}^{D} \int_{-\infty}^{\infty} \frac{1}{(2\pi\lambda_j)^{1/2}} \exp\left\{-\frac{y_j^2}{2\lambda_j}\right\} dy_j = 1 \tag{2.56}$$

where we have used the result (1.38) for the normalization of the univariate Gaussian. This confirms that the multivariate Gaussian (2.42) is indeed normalized.

We now look at the moments of the Gaussian distribution and thereby provide an interpretation of the parameters $\boldsymbol{\mu}$ and Σ. The expectation of $\mathbf{x}$ under the Gaussian distribution is given by

$$\begin{aligned}\mathbb{E}[\mathbf{x}] &= \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}\int \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}\mathbf{x}d\mathbf{x} \\ &= \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}\int \exp\left\{-\frac{1}{2}\mathbf{z}^T\Sigma^{-1}\mathbf{z}\right\}(\mathbf{z}+\boldsymbol{\mu})d\mathbf{z}\end{aligned} \tag{2.57}$$

where we have changed variables using $\mathbf{z} = \mathbf{x}-\boldsymbol{\mu}$. We now note that the exponent is an even function of the components of $\mathbf{z}$ and, because the integrals over these are taken over the range $(-\infty, \infty)$, the term in $\mathbf{z}$ in the factor $(\mathbf{z}+\boldsymbol{\mu})$ will vanish by symmetry. Thus

$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu} \tag{2.58}$$

and so we refer to $\boldsymbol{\mu}$ as the mean of the Gaussian distribution.

We now consider second order moments of the Gaussian. In the univariate case, we considered the second order moment given by $\mathbb{E}[x^2]$. For the multivariate Gaussian, there are D^2 second order moments given by $\mathbb{E}[x_i x_j]$, which we can group together to form the matrix $\mathbb{E}[\mathbf{x}\mathbf{x}^T]$. This matrix can be written as

$$\begin{aligned}\mathbb{E}[\mathbf{x}\mathbf{x}^T] &= \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}\int \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}\mathbf{x}\mathbf{x}^T d\mathbf{x} \\ &= \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}\int \exp\left\{-\frac{1}{2}\mathbf{z}^T\Sigma^{-1}\mathbf{z}\right\}(\mathbf{z}+\boldsymbol{\mu})(\mathbf{z}+\boldsymbol{\mu})^T d\mathbf{z}\end{aligned} \tag{2.59}$$

where again we have changed variables using $\mathbf{z} = \mathbf{x}-\boldsymbol{\mu}$. Note that the cross-terms involving $\boldsymbol{\mu}\mathbf{z}^T$ and $\mathbf{z}\boldsymbol{\mu}^T$ will again vanish by symmetry. The term $\boldsymbol{\mu}\boldsymbol{\mu}^T$ is constant and can be taken outside the integral, which itself is unity because the Gaussian distribution is normalized. Consider the term involving $\mathbf{z}\mathbf{z}^T$. Again, we can make use of the eigenvector expansion of the

covariance matrix given by (2.44), together with the completeness of the set of eigenvectors, to write

$$\mathbf{z} = \sum_{j=1}^{D} y_j \mathbf{u}_j \tag{2.60}$$

where $y_j = \mathbf{u}_j^T \mathbf{z}$, which gives

$$\begin{aligned}
&\frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \int \exp\left\{-\frac{1}{2}\mathbf{z}^T\Sigma^{-1}\mathbf{z}\right\} \mathbf{z}\mathbf{z}^T d\mathbf{z} \\
&= \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \sum_{i=1}^{D}\sum_{j=1}^{D} \mathbf{u}_i\mathbf{u}_j^T \int \exp\left\{-\sum_{k=1}^{D}\frac{y_k^2}{2\lambda_k}\right\} y_i y_j d\mathbf{y} \\
&= \sum_{i=1}^{D} \mathbf{u}_i\mathbf{u}_i^T \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \int \exp\left\{-\sum_{k=1}^{D}\frac{y_k^2}{2\lambda_k}\right\} y_i^2 d\mathbf{y} \\
&= \sum_{i=1}^{D} \mathbf{u}_i\mathbf{u}_i^T \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} (2\pi)^{D/2}\lambda_i \prod_{k=1}^{D}\lambda_k^{1/2} \\
&= \sum_{i=1}^{D} \mathbf{u}_i\mathbf{u}_i^T \lambda_i = \Sigma
\end{aligned} \tag{2.61}$$

where we have made use of the eigenvector equation (2.44), together with the fact that the integral on the right-hand side of the middle line vanishes by symmetry unless $i = j$, and in the final line we have made use of the results (1.40) and (2.54), together with (2.47). Thus we have

$$\mathbb{E}[\mathbf{x}\mathbf{x}^T] = \boldsymbol{\mu}\boldsymbol{\mu}^T + \Sigma. \tag{2.62}$$

For single random variables, we subtracted the mean before taking second moments in order to define a variance. Similarly, in the multivariate case it is again convenient to subtract off the mean, giving rise to the *covariance(协方差)* of a random vector $\mathbf{x}$ defined by

$$var[\mathbf{x}] = \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^T] \tag{2.63}$$

For the specific case of a Gaussian distribution, we can make use of $E[\mathbf{x}] = \boldsymbol{\mu}$, together with the result (2.61), to give

$$\text{cov}[\mathbf{x}] = \Sigma \tag{2.64}$$

Because the parameter matrix Σ governs the covariance of $\mathbf{x}$ under the Gaussian distribution, it is called the covariance matrix.

Although the Gaussian distribution (2.42) is widely used as a density model, it suffers from some significant limitations. Consider the number of free parameters in the distribution. A general symmetric covariance matrix Σ will have $D(D+1)/2$ independent parameters, and there are another D independent parameters in $\boldsymbol{\mu}$, giving $D(D+3)/2$ parameters in total. For large D, the total number of parameters therefore grows quadratically with D, and the computational task of manipulating and inverting large matrices can become prohibitive. One way to address this problem is to use restricted forms of the covariance matrix. If we consider covariance matrices that are *diagonal(对角的)*, so that $\Sigma = diag(\sigma_i^2)$, we then have a total of $2D$ independent parameters in the density model. The corresponding contours of constant density are given by axis-aligned ellipsoids. We could further restrict the covariance matrix to be proportional to the identity matrix, $\Sigma = \sigma^2\mathbf{I}$, known as an *isotropic(各向同性的)* covariance, giving $D+1$ independent parameters in the model and spherical surfaces of constant density. The three possibilities of general, diagonal, and isotropic covariance matrices are illustrated in Figure 2.8. Unfortunately,where as such approaches limit the number of degrees of freedom in the distribution and make inversion of the covariance matrix a much faster operation, they also greatly restrict the form of the probability density and limit its ability to capture interesting correlations in the data.

```
plot_numert = 100
plot_x = np.linspace(-4, 4, plot_numert)
plot_y = np.linspace(-4, 4, plot_numert)
X, Y = np.meshgrid(plot_x, plot_y)
pos = np.dstack((X, Y))
Cov = np.empty((3, 2, 2))
Cov[0] = np.array([[2, 1], [1, 2]])
Cov[1] = np.array([[2, 0], [0, 1]])
Cov[2] = np.array([[2, 0], [0, 2]])
fig = plt.figure(figsize=(12, 4))
for index, cov in enumerate(Cov):
    Z = multivariate_normal.pdf(pos, cov=cov)
    ax = fig.add_subplot(1, 3, index + 1)
    ax.contourf(X, Y, Z, cmap="BuGn")
    cs = ax.contour(X, Y, Z, 2, colors='red', linewidth=0.5)
    ax.clabel(cs, inline=True, fontsize=10, colors="k")
plt.savefig("img/fig:2.8.png")
plt.close("all")
```

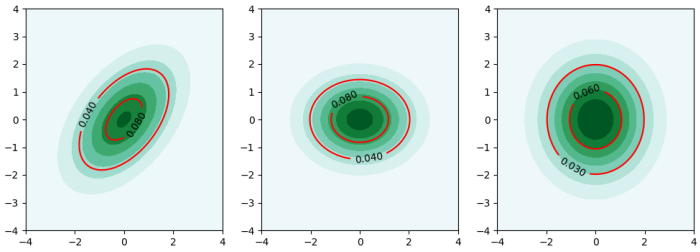


图 2.8: Contours of constant probability density for a Gaussian distribution in two dimensions in which the covariance matrix is (a) of general form, (b) diagonal, in which the elliptical contours are aligned with the coordinate axes, and (c) proportional to the identity matrix, in which the contours are concentric circles.

A further limitation of the Gaussian distribution is that it is intrinsically(内在的) unimodal(单峰的) (i.e., has a single maximum) and so is unable to provide a good approximation to multimodal distributions. Thus

the Gaussian distribution can be both too flexible, in the sense of having too many parameters, while also being too limited in the range of distributions that it can adequately represent. We will see later that the introduction of *latent variables(潜在变量)*, also called *hidden variables(隐藏变量)* or *unobserved variables(未观察变量)*, allows both of these problems to be addressed. In particular, a rich family of multimodal(多峰的) distributions is obtained by introducing discrete latent variables leading to mixtures of Gaussians, as discussed in Section 2.3.9. Similarly, the introduction of continuous latent variables, as described in Chapter 12(?), leads to models in which the number of free parameters can be controlled independently of the dimensionality D of the data space while still allowing the model to capture the dominant correlations in the data set. Indeed, these two approaches can be combined and further extended to derive a very rich set of hierarchical models that can be adapted to a broad range of practical applications. For instance, the Gaussian version of the *Markov random field(马尔科夫随机场)*, which is widely used as a probabilistic model of images, is a Gaussian distribution over the joint space of pixel intensities but rendered tractable through the imposition of considerable structure reflecting the spatial organization of the pixels. Similarly, the *linear dynamical system(线性动态系统)*, used to model time series data for applications such as tracking, is also a joint Gaussian distribution over a potentially large number of observed and latent variables and again is tractable due to the structure imposed on the distribution. A powerful framework for expressing the form and properties of such complex distributions is that of probabilistic graphical models, which will form the subject of Chapter 8(?).

2.3.1 Conditional Gaussian distributions

An important property of the multivariate Gaussian distribution is that if two sets of variables are jointly Gaussian, then the conditional distribution of one set conditioned on the other is again Gaussian. Similarly, the marginal

distribution of either set is also Gaussian.

Consider first the case of conditional distributions. Suppose $\mathbf{x}$ is a D-dimensional vector with Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$ and that we partition $\mathbf{x}$ into two disjoint subsets $\mathbf{x}_a$ and $\mathbf{x}_b$. Without loss of generality, we can take $\mathbf{x}_a$ to form the first M components of $\mathbf{x}$, with $\mathbf{x}_b$ comprising the remaining D M components, so that

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix} \tag{2.65}$$

We also define corresponding partitions of the mean vector $\boldsymbol{\mu}$ given by

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix} \tag{2.66}$$

and of the covariance matrix Σ given by

$$\Sigma = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix} \tag{2.67}$$

Note that the symmetry $\Sigma^T = \Sigma$ of the covariance matrix implies that Σ_{aa} and Σ_{bb} are symmetric, while $\Sigma_{ba} = \Sigma_{ab}^T$.

In many situations, it will be convenient to work with the inverse of the covariance matrix

$$\Lambda \equiv \Sigma^{-1} \tag{2.68}$$

which is known as the *precision matrix(精度矩阵)*. In fact, we shall see that some properties of Gaussian distributions are most naturally expressed in terms of the covariance, whereas others take a simpler form when viewed in terms of the precision. We therefore also introduce the partitioned form of the precision matrix

$$\Lambda = \begin{pmatrix} \Lambda_{aa} & \Lambda_{ab} \\ \Lambda_{ba} & \Lambda_{bb} \end{pmatrix} \tag{2.69}$$

corresponding to the partitioning (2.65) of the vector $\mathbf{x}$. Because the inverse of a symmetric matrix is also symmetric, we see that Λ_{aa} and Λ_{bb} are symmetric, while $\Lambda_{ab}^T = \Lambda_{ba}$. It should be stressed(强调) at this point that, for instance, Λ_{aa} is not simply given by the inverse of Σ_{aa}. In fact, we shall shortly examine the relation between the inverse of a partitioned matrix and the inverses of its partitions.

Let us begin by finding an expression for the conditional distribution $p(\mathbf{x}_a|\mathbf{x}_b)$. From the product rule of probability, we see that this conditional distribution can be evaluated from the joint distribution $p(\mathbf{x}) = p(\mathbf{x}_a, \mathbf{x}_b)$ simply by fixing $\mathbf{x}_b$ to the observed value and normalizing the resulting expression to obtain a valid probability distribution over $\mathbf{x}_a$. Instead of performing this normalization explicitly, we can obtain the solution more efficiently by considering the quadratic form in the exponent of the Gaussian distribution given by (2.43) and then reinstating the normalization coefficient at the end of the calculation. If we make use of the partitioning (2.65), (2.66), and (2.69), we obtain

$$\begin{aligned} -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}) = \\ -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Lambda_{aa}(\mathbf{x}-\boldsymbol{\mu}) - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Lambda_{ab}(\mathbf{x}-\boldsymbol{\mu}) \\ -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Lambda_{ba}(\mathbf{x}-\boldsymbol{\mu}) - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Lambda_{bb}(\mathbf{x}-\boldsymbol{\mu}) \end{aligned} \tag{2.70}$$

We see that as a function of $\mathbf{x}_a$, this is again a quadratic form, and hence the corresponding conditional distribution $p(\mathbf{x}_a|\mathbf{x}_b)$ will be Gaussian. Because this distribution is completely characterized by its mean and its covariance, our goal will be to identify expressions for the mean and covariance of $p(\mathbf{x}_a|\mathbf{x}_b)$ by inspection of (2.70).

This is an example of a rather common operation associated with Gaussian distributions, sometimes called 'completing the square', in which we are given a quadratic form defining the exponent terms in a Gaussian distribution, and we need to determine the corresponding mean and covariance.

Such problems can be solved straightforwardly by noting that the exponent in a general Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$ can be written

$$-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}) = \frac{1}{2}\mathbf{x}^T\Sigma^T\mathbf{x} + \mathbf{x}^T\Sigma^T\boldsymbol{\mu} + \text{const} \tag{2.71}$$

where 'const'denotes terms which are independent of $\mathbf{x}$, and we have made use of the symmetry of Σ. **Thus if we take our general quadratic form and express it in the form given by the right-hand side of (2.71), then we can immediately equate the matrix of coefficients entering the second order term in x to the inverse covariance matrix Σ^{-1} and the coefficient of the linear term in x to $\Sigma^{-1}\boldsymbol{\mu}$, from which we can obtain $\boldsymbol{\mu}$.**

Now let us apply this procedure to the conditional Gaussian distribution $p(\mathbf{x}_a|\mathbf{x}_b)$ for which the quadratic form in the exponent is given by (2.70). We will denote the mean and covariance of this distribution by $\boldsymbol{\mu}_{a|b}$ and $\Sigma_{a|b}$, respectively. Consider the functional dependence of (2.70) on $\mathbf{x}_a$ in which $\mathbf{x}_b$ is regarded as a constant. If we pick out all terms that are second order in $\mathbf{x}_a$, we have

$$-\frac{1}{2}\mathbf{x}_a^T\Lambda_{aa}\mathbf{x}_a \tag{2.72}$$

from which we can immediately conclude that the covariance (inverse precision) of $p(\mathbf{x}_a|\mathbf{x}_b)$ is given by

$$\Sigma_{a|b} = \Lambda_{aa}^{-1}. \tag{2.73}$$

Now consider all of the terms in (2.70) that are linear in $\mathbf{x}_a$

$$\mathbf{x}_a^T\{\Lambda_{aa}\boldsymbol{\mu}_a - \Lambda_{ab}(\mathbf{x}_b - \boldsymbol{\mu})_b)\} \tag{2.74}$$

where we have used $\Lambda_{ba}^T = \Lambda_{ab}$. From our discussion of the general form (**??**), the coefficient of $\mathbf{x}_a$ in this expression must equal $\Sigma_{a|b}^{-1}\boldsymbol{\mu}_{a|b}$ and hence

$$\begin{aligned}\boldsymbol{\mu}_{a|b} &= \Sigma_{a|b}\{\Lambda_{aa}\boldsymbol{\mu}_a - \Lambda_{ab}(\mathbf{x}_b - \boldsymbol{\mu}_b)\} \\ &= \boldsymbol{\mu}_a - \Lambda_{aa}^{-1}\Lambda_{ab}(\mathbf{x}_b - \boldsymbol{\mu}_b)\end{aligned} \tag{2.75}$$

where we have made use of (2.73).

The results (2.73) and (2.75) are expressed in terms of the partitioned precision matrix of the original joint distribution $p(\mathbf{x}_a, \mathbf{x}_b)$. We can also express these results in terms of the corresponding partitioned covariance matrix. To do this, we make use of the following identity for the inverse of a partitioned matrix

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} M & -MBD^{-1} \\ -D^{-1}CM & D^{-1} + D^{-1}CMBD^{-1} \end{pmatrix} \tag{2.76}$$

where we have defined

$$M = (A - BD^{-1}C)^{-1} \tag{2.77}$$

The quantity M^{-1} is known as the *Schur complement(舒尔补)* of the matrix on the left-hand side of (2.76) with respect to the submatrix D. Using the definition

$$\begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}^{-1} = \begin{pmatrix} \Lambda_{aa} & \Lambda_{ab} \\ \Lambda_{ba} & \Lambda_{bb} \end{pmatrix} \tag{2.78}$$

and making use of (2.76), we have

$$\Lambda_{aa} = (\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})^{-1} \tag{2.79}$$

$$\Lambda_{ab} = -(\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})^{-1}\Sigma_{ab}\Sigma_{bb}^{-1} \tag{2.80}$$

From these we obtain the following expressions for the mean and covariance of the conditional distribution $p(\mathbf{x}_a|\mathbf{x}_b)$

$$\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a + \Sigma_{ab}\Sigma_{bb}^{-1}(\mathbf{x} - \boldsymbol{\mu}_b) \tag{2.81}$$

$$\Sigma_{a|b} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba} \tag{2.82}$$

Comparing (2.73) and (2.82), we see that the conditional distribution $p(\mathbf{x}_a|\mathbf{x}_b)$ takes a simpler form when expressed in terms of the partitioned precision matrix than when it is expressed in terms of the partitioned covariance matrix. Note that the mean of the conditional distribution $p(\mathbf{x}_a|\mathbf{x}_b)$, given by (2.81), is a linear function of $\mathbf{x}_b$ and that the covariance, given by (2.82), is independent of x_a. This represents an example of a linear-Gaussian model.

2.3.2 Marginal Gaussian distributions

We have seen that if a joint distribution $p(\mathbf{x}_a, \mathbf{x}_b)$ is Gaussian, then the conditional distribution $p(\mathbf{x}_a|\mathbf{x}_b)$ will again be Gaussian. Now we turn to a discussion of the marginal distribution given by

$$p(\mathbf{x}_a) = \int p(\mathbf{x}_a, \mathbf{x}_b) d\mathbf{x}_b \tag{2.83}$$

which, as we shall see, is also Gaussian. Once again, our strategy for evaluating this distribution efficiently will be to focus on the quadratic form in the exponent of the joint distribution and thereby to identify the mean and covariance of the marginal distribution $p(\mathbf{x}_a)$.

The quadratic form for the joint distribution can be expressed, using the partitioned precision matrix, in the form (2.70). Because our goal is to integrate out $\mathbf{x}_b$, this is most easily achieved by first considering the terms involving $\mathbf{x}b$ and then completing the square in order to facilitate integration. Picking out just those terms that involve $\mathbf{x}_b$, we have

$$-\frac{1}{2}\mathbf{x}_b^T\Lambda_{bb}\mathbf{b} + \mathbf{x}_b^T\mathbf{m} = -\frac{1}{2}(\mathbf{x}_b - \Lambda_{bb}^{-1}\mathbf{m})^T(\mathbf{x}_b - \Lambda_{bb}^{-1}\mathbf{m}) + \frac{1}{2}\mathbf{m}^T\Lambda_{bb}^{-1}\mathbf{m} \tag{2.84}$$

where we have defined

$$\mathbf{m} = \Lambda_{bb}\boldsymbol{\mu}_b - \Lambda_{ba}(\mathbf{x}_a - \boldsymbol{\mu}_a). \tag{2.85}$$

We see that the dependence on xb has been cast into the standard quadratic form of a Gaussian distribution corresponding to the first term on the right-hand side of (2.84), plus a term that does not depend on xb (but that does depend on xa). Thus, when we take the exponential of this quadratic form, we see that the integration over xb required by (2.83) will take the form

$$\int \exp\left\{-\frac{1}{2}(\mathbf{x}_b - \Lambda_{bb}^{-1}\mathbf{m})^T\Lambda_{bb}(\mathbf{x}_b - \Lambda_{bb}^{-1}\mathbf{m})\right\} d\mathbf{x}_b \tag{2.86}$$

This integration is easily performed by noting that it is the integral over an unnormalized Gaussian, and so the result will be the reciprocal of the normalization coefficient. We know from the form of the normalized Gaussian given by (2.42), that this coefficient is independent of the mean and depends only on the determinant of the covariance matrix. Thus, by completing the square with respect to $\mathbf{x}_b$, we can integrate out $\mathbf{x}_b$ and the only term remaining from the contributions on the left-hand side of (2.84) that depends on xa is the last term on the right-hand side of (2.84) in which m is given by (2.85). Combining this term with the remaining terms from (2.70) that depend on $\mathbf{x}a$, we obtain

$$\begin{aligned}
&\frac{1}{2}[\Lambda_{bb}\boldsymbol{\mu}_b - \Lambda_{ba}(\mathbf{x}_a - \boldsymbol{\mu}_a)]^T\lambda_{bb}^{-1}[\Lambda_{bb}\boldsymbol{\mu}_b - \Lambda_{ba}(\mathbf{x}_a - \boldsymbol{\mu}_a)] \\
&\qquad - \frac{1}{2}\mathbf{x}_a^T\Lambda_{aa}\mathbf{x}_a + \mathbf{x}_a^T(\Lambda_{aa}\boldsymbol{\mu} + \Lambda_{ab}\boldsymbol{\mu}) + \text{const} \\
&\quad = -\frac{1}{2}\mathbf{x}_a^T(\Lambda_{aa} - \Lambda_{ab}\Lambda_{bb}^{-1}\Lambda_{ba})\mathbf{x}_a \\
&\qquad + \mathbf{x}_a^T(\Lambda_{aa} - \Lambda_{ab}\Lambda_{bb}^{-1}\Lambda_{ba})\boldsymbol{\mu}_a + \text{const}
\end{aligned} \tag{2.87}$$

where 'const'denotes quantities independent of $\mathbf{x}_a$. Again, by comparison with (2.71), we see that the covariance of the marginal distribution of $p(\mathbf{x}_a)$ is given by

$$\Sigma_a = (\Lambda_{aa} - \Lambda_{ab}\Lambda_{bb}^{-1}\Lambda_{ba})^{-1}. \tag{2.88}$$

Similarly, the mean is given by

$$\Sigma_a(\Lambda_{aa} - \Lambda_{ab}\Lambda_{bb}^{-1}\Lambda_{ba})\boldsymbol{\mu}_a = \boldsymbol{\mu}_a \tag{2.89}$$

where we have used (2.88). The covariance in (2.88) is expressed in terms of the partitioned precision matrix given by (2.69). We can rewrite this in terms of the corresponding partitioning of the covariance matrix given by (2.67), as we did for the conditional distribution. These partitioned matrices are related by

$$\begin{pmatrix} \Lambda_{aa} & \Lambda_{ab} \\ \Lambda_{ba} & \Lambda_{bb} \end{pmatrix}^{-1} = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix} \tag{2.90}$$

Making use of (2.76), we then have

$$(\Lambda_{aa} - \Lambda_{ab}\Lambda_{bb}^{-1}\Lambda_{ba})^{-1} = \Sigma_{aa} \tag{2.91}$$

Thus we obtain the intuitively satisfying result that the marginal distribution $p(\mathbf{x}_a)$ has mean and covariance given by

$$\mathbb{E}[\mathbf{x}_a] = \boldsymbol{\mu}_a \tag{2.92}$$

$$var[\mathbf{x}_a] = \Sigma_{aa} \tag{2.93}$$

We see that for a marginal distribution, the mean and covariance are most simply expressed in terms of the partitioned covariance matrix, in contrast to the conditional distribution for which the partitioned precision matrix gives rise to simpler expressions.

Our results for the marginal and conditional distributions of a partitioned Gaussian are summarized below.

Partitioned Gaussians

Given a joint Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\ )$ with $\Lambda \equiv \Sigma^{-1}$ and

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix}, \quad \boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix} \tag{2.94}$$

$$\Sigma = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}, \quad \Lambda = \begin{pmatrix} \Lambda_{aa} & \Lambda_{ab} \\ \Lambda_{ba} & \Lambda_{bb} \end{pmatrix}. \tag{2.95}$$

Conditional distribution:

$$p(\mathbf{x}_a|\mathbf{x}_b) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{a|b}, \Lambda_{aa}^{-1}) \tag{2.96}$$

$$\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a - \Lambda_{aa}^{-1}\Lambda_{ab}(\mathbf{x}_b - \boldsymbol{\mu}_b). \tag{2.97}$$

Marginal distribution:

$$p(\mathbf{x}_a) = \mathcal{N}(\mathbf{x}_a|\boldsymbol{\mu}_a, \Sigma_{aa}). \tag{2.98}$$

We illustrate the idea of conditional and marginal distributions associated with a multivariate Gaussian using an example involving two variables in Figure 2.9.

2.3.3 Bayes'theorem for Gaussian variables

In Sections 2.3.1 and 2.3.2, we considered a Gaussian $p(\mathbf{x})$ in which we partitioned the vector x into two subvectors $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$ and then found expressions for the conditional distribution $p(\mathbf{x}_a|\mathbf{x}_b)$ and the marginal distribution $p(\mathbf{x}_a)$. We noted that the mean of the conditional distribution $p(\mathbf{x}_a|\mathbf{x}_b)$ was a linear function of $\mathbf{x}_b$. Here we shall suppose that we are given a Gaussian marginal distribution $p(x)$ and a Gaussian conditional distribution $p(\mathbf{y}|\mathbf{x})$ in which $p(\mathbf{y}|\mathbf{x})$ has a mean that is a linear function of $\mathbf{x}$,

```
plot_numert = 100
plot_x = np.linspace(0, 1, plot_numert)
plot_y = np.linspace(0, 1, plot_numert)
X, Y = np.meshgrid(plot_x, plot_y)
pos = np.dstack((X, Y))
A = np.array([[0.2, 0.15], [0.05, 0.2]])
Cov = A.T @ A
Mean = np.array([0.5] * 2)
fig = plt.figure(figsize=(12, 6))
Z = multivariate_normal.pdf(pos, mean=Mean, cov=Cov)
z1 = norm.pdf(plot_x)
ax1 = fig.add_subplot(1, 2, 1)
ax1.contourf(X, Y, Z, cmap="BuGn")
cs = ax1.contour(X, Y, Z, 2, colors='red', linewidth=0.5)
ax1.clabel(cs, inline=True, fontsize=12, colors="k")
ax1.plot(plot_x, 0.7 * np.ones((100, 1)), "b")
ax1.annotate("$p(x_a,x_b)$", xy=(0.6, 0.2), fontsize=20, color="r")
ax1.annotate("$x_b = 0.7$", xy=(0.05, 0.8), fontsize=20, color="b")
ax1.set_xlabel("$x_a$")
ax1.set_ylabel("$x_b$")
cov_a = Cov[0, 0]
mean_a = Mean[0]
Pre = np.linalg.inv(Cov)
cov_a_b = 1 / Pre[0, 0]
mean_a_b = Mean[0] - cov_a_b * Pre[0, 1] * (0.7 - Mean[1])
marginal_a = multivariate_normal.pdf(
    plot_x,
    mean=mean_a,
    cov=cov_a,
)
conditional_a_b = multivariate_normal.pdf(plot_x, mean=mean_a_b, cov=cov_a_b)
ax2 = fig.add_subplot(1, 2, 2)
ax2.set_ylim(0, 4)
ax2.plot(plot_x, marginal_a, "b")
ax2.plot(plot_x, conditional_a_b, "r")
ax2.annotate("$p(x_a)$", xy=(0.1, 2), fontsize=15, color="b")
ax2.annotate("$p(x_a|x_b = 0.7)$", xy=(0.65, 3.5), fontsize=15, color="r")
ax2.set_xlabel("$x_a$")
plt.savefig("img/fig:2.9.png")
plt.close("all")
```

Listing 2.7: fig:2.9

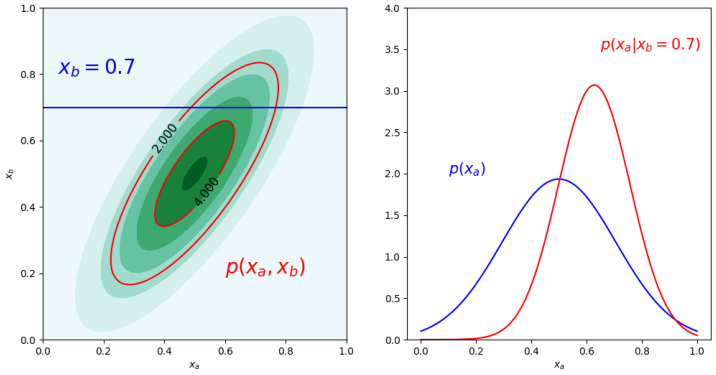


图 2.9: The plot on the left shows the contours of a Gaussian distribution $p(\mathbf{x}_a, \mathbf{x}_b)$ over two variables, and the plot on the right shows the marginal distribution $p(\mathbf{x}_a)$ (blue curve) and the conditional distribution $p(\mathbf{x}_a|\mathbf{x}_b)$ for $\mathbf{x}_b = 0.7$ (red curve).

and a covariance which is independent of $\mathbf{x}$. This is an example of a linear Gaussian model (Roweis and Ghahramani, 1999), which we shall study in greater generality in Section 8.1.4(?). We wish to find the marginal distribution $p(\mathbf{y})$ and the conditional distribution $p(\mathbf{x}|\mathbf{y})$. This is a problem that will arise frequently in subsequent chapters, and it will prove convenient to derive the general results here.

We shall take the marginal and conditional distributions to be

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Lambda^{-1}) \tag{2.99}$$

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|A\mathbf{x} + b, L^{-1}) \tag{2.100}$$

where $\boldsymbol{\mu}$, A, and b are parameters governing the means, and Λ and L are precision matrices. If $\mathbf{x}$ has dimensionality M and $\mathbf{y}$ has dimensionality D, then the matrix A has size $D \times M$.

First we find an expression for the joint distribution over $\mathbf{x}$ and $\mathbf{y}$. To do this, we define

$$\mathbf{z} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \tag{2.101}$$

and then consider the log of the joint distribution

$$\begin{aligned} \ln p(\mathbf{z}) &= \ln p(\mathbf{x}) + \ln p(\mathbf{y}|\mathbf{x}) \\ &= -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Lambda (\mathbf{x}-\boldsymbol{\mu}) \\ &\quad -\frac{1}{2}(\mathbf{y}-A\mathbf{x}-\mathbf{b})^T L(\mathbf{y}-A\mathbf{x}-\mathbf{b}) + \text{const} \end{aligned} \tag{2.102}$$

where 'const'denotes terms independent of $\mathbf{x}$ and $\mathbf{y}$. As before, we see that this is a quadratic function of the components of $\mathbf{z}$, and hence $p(\mathbf{z})$ is Gaussian distribution. To find the precision of this Gaussian, we consider the second order terms in (2.102), which can be written as

$$\begin{aligned} &-\frac{1}{2}\mathbf{x}^T(\Lambda + A^T L A)\mathbf{x} - \frac{1}{2}\mathbf{y}^T L \mathbf{y} + \frac{1}{2} y^T L A \mathbf{x} + \frac{1}{2}\mathbf{x}^T A^T \mathbf{y} \\ &= -\frac{1}{2}\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}^T \begin{pmatrix} \Lambda + A^T L A & -A^T L \\ -LA & L \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = -\frac{1}{2}\mathbf{z}^T R \mathbf{z} \end{aligned} \tag{2.103}$$

and so the Gaussian distribution over $z has precision (inverse covariance) matrix given by

$$R = \begin{pmatrix} \Lambda + A^T L A & -A^T L \\ -LA & L \end{pmatrix}. \tag{2.104}$$

The covariance matrix is found by taking the inverse of the precision, which can be done using the matrix inversion formula (2.76) to give

$$\text{cov}[\mathbf{z}] = R^{-1} = \begin{pmatrix} \Lambda^{-1} & \Lambda^{-1} A^T \\ A\Lambda^{-1} & L^{-1} + A\Lambda^{-1} A^T \end{pmatrix}. \tag{2.105}$$

Similarly, we can find the mean of the Gaussian distribution over $\mathbf{z}$ by identifying the linear terms in (2.102), which are given by

$$\mathbf{x}^T \Lambda \boldsymbol{\mu} - \mathbf{x}^T A^T L b + y^T L b = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}^T \begin{pmatrix} \Lambda \boldsymbol{\mu} - A^T L b \\ L b \end{pmatrix} \tag{2.106}$$

Using our earlier result (2.71) obtained by completing the square over the quadratic form of a multivariate Gaussian, we find that the mean of $\mathbf{z}$ is given by

$$\mathbb{E}[\mathbf{z}] = R^{-1} \begin{pmatrix} \Lambda \boldsymbol{\mu} - \mathbf{A}^T L b \\ L b \end{pmatrix} \tag{2.107}$$

Making use of (**??**), we then obtain

$$\mathbb{E}[\mathbf{z}] = \begin{pmatrix} \boldsymbol{\mu} \\ A\boldsymbol{\mu} + b \end{pmatrix} \tag{2.108}$$

Next we find an expression for the marginal distribution $p(\mathbf{y})$ in which we have marginalized over $\mathbf{x}$. Recall that the marginal distribution over a subset of the components of a Gaussian random vector takes a particularly simple form when expressed in terms of the partitioned covariance matrix. Specifically, its mean and covariance are given by (2.92) and (2.93), respectively. Making use of (2.105) and (2.108) we see that the mean and covariance of the marginal distribution $p(\mathbf{y})$ are given by

$$\mathbb{E}[\mathbf{y}] = A\boldsymbol{\mu} + b \tag{2.109}$$

$$\text{cov}[\mathbf{y}] = L^{-1} + A\Lambda^{-1}A^T. \tag{2.110}$$

A special case of this result is when $A = I$, in which case it reduces to the convolution of two Gaussians, for which we see that the mean of the convolution is the sum of the mean of the two Gaussians, and the covariance of the convolution is the sum of their covariances.

Finally, we seek an expression for the conditional $p(\mathbf{x}|\mathbf{y})$. Recall that the results for the conditional distribution are most easily expressed in terms of the partitioned precision matrix, using (2.73) and (2.75). Applying these

results to (2.105) and (2.108) we see that the conditional distribution $p(\mathbf{x}|\mathbf{y})$ has mean and covariance given by

$$\mathbb{E}[\mathbf{x}|\mathbf{y}] = (\Lambda + A^T L A)^{-1}\{A^T L(\mathbf{y} - b) + \Lambda\boldsymbol{\mu}\} \tag{2.111}$$

$$\text{cov}[\mathbf{x}|\mathbf{y}] = (\Lambda + A^T L A)^{-1}. \tag{2.112}$$

The evaluation of this conditional can be seen as an example of Bayes' theorem. We can interpret the distribution $p(\mathbf{x})$ as a prior distribution over $\mathbf{x}$. If the variable $\mathbf{y}$ is observed, then the conditional distribution $p(\mathbf{x}|\mathbf{y})$ represents the corresponding posterior distribution over $\mathbf{x}$. Having found the marginal and conditional distributions, we effectively expressed the joint distribution $p(z) = p(\mathbf{x})p(\mathbf{y}|\mathbf{x})$ in the form $p(\mathbf{x}|\mathbf{y})p(\mathbf{y})$. These results are summarized below.

Marginal and Conditional Gaussians

Given a marginal Gaussian distribution for $\mathbf{x}$ and a conditional Gaussian distribution for $\mathbf{y}$ given $\mathbf{x}$ in the form

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Lambda^{-1}) \tag{2.113}$$

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|A\mathbf{x} + b, L^{-1}) \tag{2.114}$$

the marginal distribution of $\mathbf{y}$ and the conditional distribution of $\mathbf{x}$ given $\mathbf{y}$ are given by

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|A\boldsymbol{\mu} + b, L^{-1} + A\Lambda^{-1}A^T) \tag{2.115}$$

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\Sigma\{A^T L(\mathbf{y} - b) + \Lambda\boldsymbol{\mu}\}, \Sigma) \tag{2.116}$$

where

$$\Sigma = (\Lambda + A^T L A)^{-1}. \tag{2.117}$$

2.3.4 Maximum likelihood for the Gaussian

Given a data set $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_N)^T$ in which the observations $\{\mathbf{x}_n\}$ are assumed to be drawn independently from a multivariate Gaussian distribution, we can estimate the parameters of the distribution by maximum likelihood. The log likelihood function is given by

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \Sigma) = -\frac{ND}{2}\ln(2\pi) - \frac{N}{2}\ln|\Sigma| - \frac{1}{2}\sum_{n=1}^{N}(\mathbf{x}_n - \boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}_n - \boldsymbol{\mu}). \tag{2.118}$$

By simple rearrangement, we see that the likelihood function depends on the data set only through the two quantities

$$\sum_{n=1}^{N}\mathbf{x}_n, \quad \sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^T. \tag{2.119}$$

These are known as the *sufficient statistics(充分统计量)* for the Gaussian distribution. Using (C.19)(?), the derivative of the log likelihood with respect to $\boldsymbol{\mu}$ is given by

$$\frac{\partial}{\partial\boldsymbol{\mu}}\ln p(\mathbf{X}|\boldsymbol{\mu}, \Sigma) = \sum_{n=1}^{N}\Sigma^{-1}(\mathbf{x}_n - \boldsymbol{\mu}) \tag{2.120}$$

and setting this derivative to zero, we obtain the solution for the maximum likelihood

$$\boldsymbol{\mu}_{ML} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \tag{2.121}$$

which is the mean of the observed set of data points. The maximization of (2.118) with respect to Σ is rather more involved(复杂的). The simplest

approach is to ignore the symmetry constraint and show that the resulting solution is symmetric as required. Alternative derivations of this result, which impose the symmetry and positive definiteness constraints explicitly, can be found in Magnus and Neudecker (1999). The result is as expected and takes the form

$$\Sigma_{ML} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \boldsymbol{\mu}_{ML})(\mathbf{x}_n - \boldsymbol{\mu}_{ML})^T. \tag{2.122}$$

which involves $\boldsymbol{\mu}_{ML}$ because this is the result of a joint maximization with respect to $\boldsymbol{\mu}$ and Σ. Note that the solution (2.121) for $\boldsymbol{\mu}_{ML}$ does not depend on Σ_{ML}, and so we can first evaluate $\boldsymbol{\mu}_{ML}$ and then use this to evaluate Σ_{ML}.

If we evaluate the expectations of the maximum likelihood solutions under the true distribution, we obtain the following results

$$\mathbb{E}[\boldsymbol{\mu}_{ML}] = \boldsymbol{\mu} \tag{2.123}$$

$$\mathbb{E}[\Sigma_{ML}] = \frac{N-1}{N}\Sigma \tag{2.124}$$

We see that the expectation of the maximum likelihood estimate for the mean is equal to the true mean. However, the maximum likelihood estimate for the covariance has an expectation that is less than the true value, and hence it is biased. We can correct this bias by defining a different estimator $\tilde{\Sigma}$ given by

$$\tilde{\Sigma} = \frac{1}{N-1}\sum_{n=1}^{N}(\mathbf{x}_n - \boldsymbol{\mu}_{ML})(\mathbf{x}_n - \boldsymbol{\mu}_{ML})^T. \tag{2.125}$$

Clearly from (2.124) and (2.125), the expectation of $\tilde{\Sigma}$ is equal to Σ.

2.3.5 TODO Sequential estimation

Our discussion of the maximum likelihood solution for the parameters of a Gaussian distribution provides a convenient opportunity to give a more

general discussion of the topic of sequential estimation for maximum likelihood. Sequential methods allow data points to be processed one at a time and then discarded and are important for on-line applications, and also where large data sets are involved so that batch processing of all data points at once is infeasible.

Consider the result (2.121) for the maximum likelihood estimator of the mean $\boldsymbol{\mu}_{ML}$, which we will denote by $\boldsymbol{\mu}^{(N)}$ when it is based on N observations. If we dissect(仔细分析) out the contribution from the final data point $\mathbf{x}_N$, we obtain

$$\begin{aligned}\boldsymbol{\mu}_{ML}^{(N)} &= \frac{1}{N}\sum_{n=1} N\mathbf{x}_n \\ &= \frac{1}{N}\mathbf{x}_N + \frac{1}{N}\sum_{n=1}^{N-1}\mathbf{x}_n \\ &= \frac{1}{N}\mathbf{x}_N + \frac{N-1}{N}\boldsymbol{\mu}_{ML}^{(N-1)} \\ &= \boldsymbol{\mu}_{ML}^{(N-1)} + \frac{1}{N}(\mathbf{x}_N - \boldsymbol{\mu}_{ML}^{(N-1)}).\end{aligned} \tag{2.126}$$

This result has a nice interpretation, as follows. After observing $N-1$ data points we have estimated $\boldsymbol{\mu}$ by $\boldsymbol{\mu}_{ML}^{(N-1)}$. We now observe data point $\mathbf{x}_N$, and we obtain our revised estimate $\boldsymbol{\mu}^{(N)}$ by moving the old estimate a small amount, proportional to $1/N$, in the direction of the 'error signal' $(\mathbf{x}_N - \boldsymbol{\mu}_{ML}^{(N-1)})$. Note that, as N increases, so the contribution from successive data points gets smaller.

The result (2.126) will clearly give the same answer as the batch result (2.121) because the two formulae are equivalent. However, we will not always be able to derive a sequential algorithm by this route, and so we seek a more general formulation of sequential learning, which leads us to the Robbins-Monro algorithm. Consider a pair of random variables θ and z governed by a joint distribution $p(z, \theta)$. The conditional expectation of z given defines a deterministic function $f(\theta)$ that is given by

$$f(\theta) \equiv \mathbf{E}[z|\theta] = \int zp(z|\theta)dz \tag{2.127}$$

and is illustrated schematically in Figure 2.10. Functions defined in this way are called *regression functions(回归函数)*.

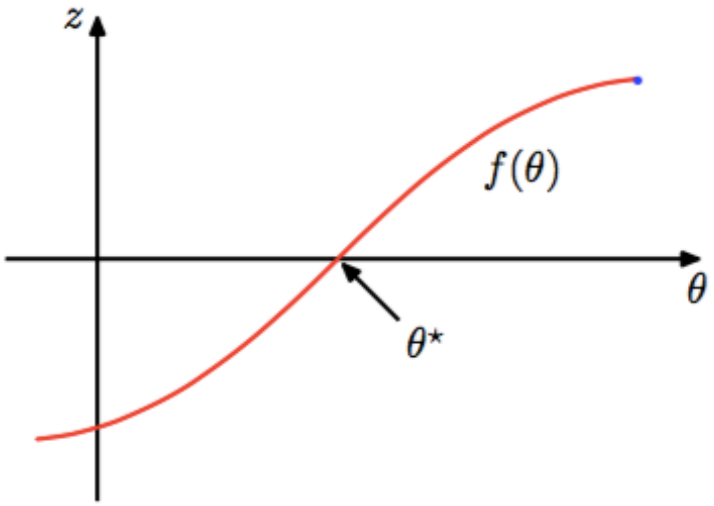


图 2.10: A schematic illustration of two correlated random variables z and θ, together with the regression function $f(\theta)$ given by the conditional expectation $\mathbb{E}[z|\theta]$. The Robbins-Monro algorithm provides a general sequential procedure for finding the root θ^* of such functions.

Our goal is to find the root θ^* at which $f(\theta^*) = 0$. If we had a large data set of observations of z and θ, then we could model the regression function directly and then obtain an estimate of its root. Suppose, however, that we observe values of z one at a time and we wish to find a corresponding sequential estimation scheme for $\theta*$. The following general procedure for solving such problems was given by Robbins and Monro (1951). We shall assume that the conditional variance of z is finite so that

$$\mathbb{E}[(z - f)^2|\theta] < \infty \tag{2.128}$$

and we shall also, without loss of generality, consider the case where $f(\theta) > 0$ for $\theta > \theta^*$ and $f(\theta) < 0$ for $\theta < \theta^*$, as is the case in Figure 2.10. The Robbins-Monro procedure then defines a sequence of successive estimates of the root θ^* given by

$$\theta^{(N)} = \theta^{(N-1)} + a_{N-1} z(\theta^{(N-1)}) \tag{2.129}$$

where $z(\theta^{(N)})$ is an observed value of z when θ takes the value $\theta^{(N)}$. The coefficients $\{a_N\}$ represent a sequence of positive numbers that satisfy the conditions

$$\lim_{N\to\infty} a_N = 0 \tag{2.130}$$

$$\sum_{N=1}^{\infty} a_N = \infty \tag{2.131}$$

$$\sum_{N=1}^{\infty} a_N^2 < \infty \tag{2.132}$$

It can then be shown (Robbins and Monro, 1951; Fukunaga, 1990) that the sequence of estimates given by (2.129) does indeed converge to the root with probability one. Note that the first condition (2.130) ensures that the successive corrections decrease in magnitude so that the process can converge to a limiting value. The second condition (2.131) is required to ensure that the algorithm does not converge short of the root, and the third condition (2.132) is needed to ensure that the accumulated noise has finite variance and hence does not spoil convergence.

Now let us consider how a general maximum likelihood problem can be solved sequentially using the Robbins-Monro algorithm. By definition, the maximum likelihood solution $_{ML}$ is a stationary point of the log likelihood function and hence satisfies

$$\frac{\partial}{\partial\theta}\left\{\frac{1}{N}\sum_{n=1}^{N}\ln p(\mathbf{x}_n|\theta)\right\}\Bigg|_{\theta_{ML}} = 0 \tag{2.133}$$

Exchanging the derivative and the summation, and taking the limit $N \to \infty$ we have

$$\lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \frac{\partial}{\partial\theta} \ln p(x_n|\theta) = \mathbb{E}_x \left[\frac{\partial}{\partial\theta} \ln p(x|\theta) \right] \tag{2.134}$$

and so we see that finding the maximum likelihood solution corresponds to find- ing the root of a regression function. We can therefore apply the Robbins-Monro procedure, which now takes the form

$$\theta^{(N)} = \theta^{(N-1)} + a_{n-1} \frac{\partial}{\partial\theta^{(N-1)}} \ln p(x_N|\theta^{(N-1)}). \tag{2.135}$$

As a specific example, we consider once again the sequential estimation of the mean of a Gaussian distribution, in which case the parameter $\theta(N)$ is the estimate $\mu_{ML}^{(N)}$ of the mean of the Gaussian, and the random variable z is given by

$$z = \frac{\partial}{\partial\mu_{ML}} \ln p(x|\mu_{ML}, \sigma^2) = \frac{1}{\sigma^2}(x - \mu_{ML}). \tag{2.136}$$

Thus the distribution of z is Gaussian with mean $\mu - \mu_{ML}$, as illustrated in Figure 2.11. Substituting (2.136) into (2.135), we obtain the univariate form of (2.126), provided we choose the coefficients aN to have the form $a_N = \sigma^2/N$. Note that although we have focussed on the case of a single variable, the same technique, together with the same restrictions (2.130)-(2.132) on the coefficients a_N , apply equally to the multivariate case (Blum, 1965).

2.3.6 Bayesian inference for the Gaussian

The maximum likelihood framework gave point estimates for the parameters $\boldsymbol{\mu}$ and Σ. Now we develop a Bayesian treatment by introducing prior distributions over these parameters. Let us begin with a simple example in which we consider a single Gaussian random variable x. We shall **suppose that the variance σ^2 is known, and we consider the task of**

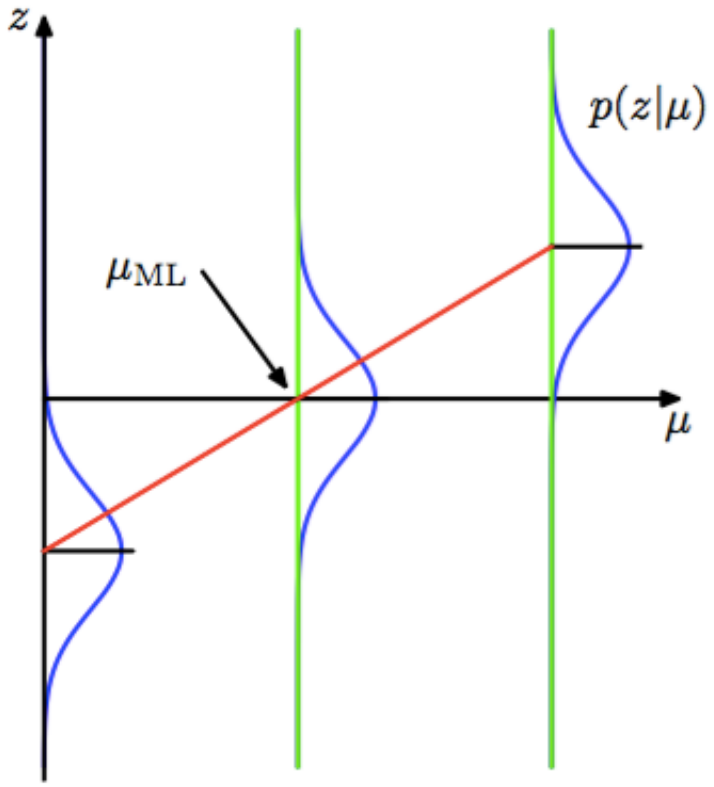


图 2.11: In the case of a Gaussian distribution, with θ corresponding to the mean μ, the regression function illustrated in Figure 2.10 takes the form of a straight line, as shown in red. In this case, the random variable z corresponds to the derivative of the log likelihood function and is given by $(x - \mu_{ML})/\sigma^2$, and its expectation that defines the regression function is a straight line given by $(\mu - \mu_{ML})/\sigma^2$. The root of the regression function corresponds to the maximum likelihood estimator μ_{ML}.

inferring the mean μ given a set of N observations $\mathbf{X} = \{x_1, \cdots, x_N\}$. The likelihood function, that is the probability of the observed data given μ, viewed as a function of μ, is given by

$$p(\mathbf{X}|\mu) = \prod_{n=1}^{N} p(x_n|\mu) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left\{-\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2\right\}. \quad (2.137)$$

Again we emphasize that the likelihood function $p(\mathbf{X}|\mu)$ is not a probability distribution over μ and is not normalized. We see that the likelihood function takes the form of the exponential of a quadratic form in μ. Thus if we choose a prior $p(\mu)$ given by a Gaussian, it will be a conjugate distribution for this likelihood function because the corresponding posterior will be a product of two exponentials of quadratic functions of and hence will also be Gaussian. We therefore take our prior distribution to be

$$p(\mu) = \mathcal{N}(\mu|\mu_0, \sigma_0^2) \quad (2.138)$$

and the posterior distribution is given by

$$p(\mu|\mathbf{X}) \propto p(\mathbf{X}|\mu)p(\mu) \quad (2.139)$$

Simple manipulation involving completing the square in the exponent shows that the posterior distribution is given by

$$p(\mu|\mathbf{x}) = \mathcal{N}(\mu|\mu_N, \sigma_N^2) \quad (2.140)$$

where

$$\mu_N = \frac{\sigma^2}{N_0^2+^2}\mu_0 + \frac{N\sigma_0^2}{N\sigma_0^2+\sigma^2}\mu_{ML} \quad (2.141)$$

$$\frac{1}{\sigma_N^2} = \frac{1}{\sigma_0^2} + \frac{N}{\sigma^2} \quad (2.142)$$

in which μ_{ML} is the maximum likelihood solution for μ given by the sample mean

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{2.143}$$

It is worth spending a moment studying the form of the posterior mean and variance. First of all, we note that the mean of the posterior distribution given by (2.141) is a compromise between the prior mean μ_0 and the maximum likelihood solution μ_{ML}. If the number of observed data points $N = 0$, then (2.141) reduces to the prior mean as expected. For $N \to \infty$, the posterior mean is given by the maximum likelihood solution. Similarly, consider the result (2.142) for the variance of the posterior distribution. We see that this is most naturally expressed in terms of the inverse variance, which is called the precision. Furthermore, the precisions are additive, so that the precision of the posterior is given by the precision of the prior plus one contribution of the data precision from each of the observed data points. As we increase the number of observed data points, the precision steadily increases, corresponding to a posterior distribution with steadily decreasing variance. With no observed data points, we have the prior variance, whereas if the number of data points $N \to \infty$, the variance σ_N^2 goes to zero and the posterior distribution becomes infinitely peaked around the maximum likelihood solution. We therefore see that the maximum likelihood result of a point estimate for μ given by (2.143) is recovered precisely from the Bayesian formalism in the limit of an infinite number of observations. Note also that for finite N , if we take the limit $\sigma_0^2 \to \infty$ in which the prior has infinite variance then the posterior mean (2.141) reduces to the maximum likelihood result, while from (2.142) the posterior variance is given by $\sigma_N^2 = \sigma^2/N$.

We illustrate our analysis of Bayesian inference for the mean of a Gaussian distribution in Figure 2.12. The generalization of this result to the case of a D-dimensional Gaussian random variable $\mathbf{x}$ with known covariance and unknown mean is straightforward.

We have already seen how the maximum likelihood expression for the mean of a Gaussian can be re-cast as a sequential update formula in which

```
mean = 0.8
std = 0.1
X = np.random.normal(loc=mean, scale=std, size=N)
mean_ml = X.mean()
std_prior = std
mean_prior = 0
fig = plt.figure(figsize=(10, 6.18))
ax = fig.add_subplot(111)
for N, color in zip([0, 1, 2, 10], ["k", "g", "b", "r"]):
    std_N = norm([std_prior, std / np.sqrt(N)], -2)
    mean_N = ((std**2) / (N*std_prior**2 + std**2))*mean_prior + \
        ((N*std_prior**2)/(N*std_prior**2+std**2))*mean_ml
    x_plot = np.linspace(-1, 1, 100)
    y_plot_0 = normal.pdf(x_plot, loc=mean_prior, scale=std_prior)
    y_plot_N = normal.pdf(x_plot, loc=mean_N, scale=std_N)
    ax.plot(x_plot, y_plot_N, color, label="$N={0}$".format(N))
    ax.legend(loc=2)
fig.savefig("img/fig:2.12.png")
plt.close("all")
```

Listing 2.8: fig:2.12

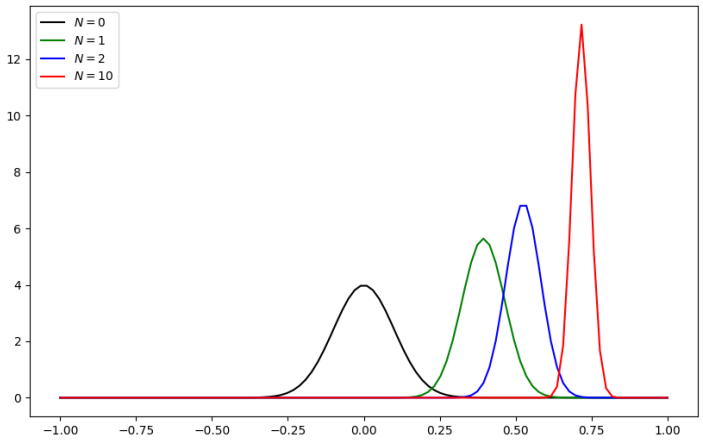


图 2.12: Illustration of Bayesian inference for the mean μ of a Gaussian distribution, in which the variance is assumed to be known. The curves show the prior distribution over μ (the curve labelled $N = 0$), which in this case is itself Gaussian, along with the posterior distribution given by (2.140) for increasing numbers N of data points. The data points are generated from a Gaussian of mean 0.8 and variance 0.1, and the prior is chosen to have mean 0. In both the prior and the likelihood function, the variance is set to the true value.

the mean after observing N data points was expressed in terms of the mean after observing $N-1$ data points together with the contribution from data point x_N. In fact, the Bayesian paradigm(范例) leads very naturally to a sequential view of the inference problem. To see this in the context of the inference of the mean of a Gaussian, we write the posterior distribution with the contribution from the final data point x_N separated out so that

$$p(\mu|\mathbf{X}) \propto \left[p(\mu) \prod_{n=1}^{N-1} p(x_n|\mu) \right] p(x_N|\mu) \tag{2.144}$$

The term in square brackets is (up to a normalization coefficient) just the posterior distribution after observing $N-1$ data points. We see that this can be viewed as a prior distribution, which is combined using Bayes' theorem with the likelihood function associated with data point xN to arrive at the posterior distribution after observing N data points. This sequential view of Bayesian inference is very general and applies to any problem in which the observed data are assumed to be independent and identically distributed.

So far, we have assumed that the variance of the Gaussian distribution over the data is known and our goal is to infer the mean. Now let us **suppose that the mean is known and we wish to infer the variance**. Again, our calculations will be greatly simplified if we choose a conjugate form for the prior distribution. It turns out to be most convenient to work with the precision $\lambda \equiv 1/\sigma^2$. The likelihood function for λ takes the form

$$p(\mathbf{X}|\lambda) = \prod_{n=1}^{N} \mathcal{N}(x_n|\mu, \lambda^{-1}) \propto \lambda^{N/2} \exp\left\{ -\frac{\lambda}{2} \sum_{n=1}^{N} (x_n - \mu)^2 \right\} \tag{2.145}$$

The corresponding conjugate prior should therefore be proportional to the product of a power of λ and the exponential of a linear function of λ. This corresponds to the gamma distribution which is defined by

$$\text{Gam}(\lambda|a,b) = \frac{1}{\Gamma(a)} b^a \lambda^{a-1} \exp(-b\lambda). \tag{2.146}$$

Here $\Gamma(a)$ is the gamma function that is defined by (2.14) and that ensures that (2.146) is correctly normalized. The gamma distribution has a finite integral if $a > 0$, and the distribution itself is finite if $a \geqslant 1$. It is plotted, for various values of a and b, in Figure 2.13. The mean and variance of the gamma distribution are given by

$$\mathbb{E}[\lambda] = \frac{a}{b} \tag{2.147}$$

$$var[x] = \frac{a}{b^2} \tag{2.148}$$

Consider a prior distribution $\text{Gam}(\lambda|a_0, b_0)$. If we multiply by the likelihood function (2.145), then we obtain a posterior distribution

$$p(\lambda|\mathbf{X}) \propto \lambda^{a_0-1}\lambda^{N/2} \exp\left\{-b_0\lambda - \frac{\lambda}{2}\sum_{n=1}^{N}(x_n - \mu)^2\right\} \tag{2.149}$$

which we recognize as a gamma distribution of the form $\text{Gam}(\lambda|a_N, b_N)$ where

$$a_N = a_0 + \frac{N}{2} \tag{2.150}$$

$$b_N = b_0 + \frac{1}{2}\sum_{n=1}^{N}(x_n - \mu)^2 = b_0 + \frac{N}{2}\sigma_{ML} \tag{2.151}$$

where σ^2_{ML} is the maximum likelihood estimator of the variance. Note that in (2.149) there is no need to keep track of the normalization constants in the prior and the likelihood function because, if required, the correct coefficient can be found at the end using the normalized form (2.146) for the gamma distribution.

```
x_plot = np.linspace(0, 2, 100)
fig = plt.figure(figsize=(12, 4))
for index, (a, b) in enumerate([(0.1, 0.1), (1, 1), (4, 6)]):
    y_plot = gamma.pdf(x_plot, a, scale=1 / b)
    ax = fig.add_subplot(1, 3, index + 1)
    ax.plot(x_plot, y_plot, "r")
    ax.set_ylim(0, 2)
    ax.set_xlim(0, 2)
    ax.annotate("$a={0}$\n$b={1}$".format(a, b), xy=(1, 1.5), fontsize=15)
fig.savefig("img/fig:2.13.png")
plt.close("all")
```

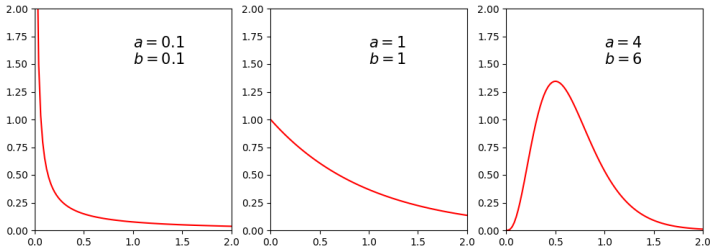


图 2.13: Plot of the gamma distribution $\text{Gam}(\lambda|a,b)$ defined by (2.146) for various values of the parameters a and b.

From (2.150), we see that the effect of observing N data points is to increase the value of the coefficient a by $N/2$. Thus we can interpret the parameter a_0 in the prior in terms of $2a_0$ 'effective'prior observations. Similarly, from (2.151) we see that the N data points contribute $N\sigma^2_{ML}/2$ to the parameter b, where σ^2_{ML} is the variance, and so we can interpret the parameter b_0 in the prior as arising from the $2a_0$ 'effective'prior observations having variance $2b_0/(2a_0) = b_0/a_0$. Recall that we made an analogous interpretation for the Dirichlet prior. These distributions are examples of the exponential family, and we shall see that the interpretation of a conjugate prior in terms of effective fictitious data points is a general one for the exponential family of distributions.

根据公式 (2.150), 我们看到观测 N 个数据点的效果是把系数 a 的值增加 N. 因此我们可以把先验分布中的参数 a_0 看成 $2a_0$ 个"有效"先验观测. 类似地, 根据公式 (2.151), 我们看到 N 个数据点对参数 b 贡献了 $N\sigma_{ML}^2/2$, 其中 σ^2 是方差, 因此我们可以把先验分布中的参数 b_0 看成方差为 $2b_0/2a_0 = b_0$ 的 $2a_0$ 个"有效"先验观测. 回忆一下, 我们对于狄利克雷分布做过类似的表述. 这些分布都是指数族分布的例子, 我们会看到, 对于指数族分布来说, 把共轭先验看成有效假想数据点是一个很通用的思想.

Instead of working with the precision, we can consider the variance itself. The conjugate prior in this case is called the inverse gamma distribution, although we shall not discuss this further because we will find it more convenient to work with the precision.

Now **suppose that both the mean and the precision are unknown**. To find a conjugate prior, we consider the dependence of the likelihood function on μ and λ

$$\begin{aligned} p(\mathbf{X}|\mu,\lambda) &= \prod_{n=1}^{N}\left(\frac{\lambda}{2\pi}\right)^{1/2}\exp\left\{-\frac{\lambda}{2}(x_n-\lambda)^2\right\} \\ &\propto \left[\lambda^{1/2}\exp\left(-\frac{\lambda\mu^2}{2}\right)\right]^N \exp\left\{\lambda\mu\sum_{n=1}^{N}x_n - \frac{\lambda}{2}\sum_{n=1}^{N}Nx_n^2\right\} \end{aligned} \tag{2.152}$$

We now wish to identify a prior distribution $p(\mu,\lambda)$ that has the same functional dependence on μ and λ as the likelihood function and that should therefore take the form

$$\begin{aligned} p(\mu,\lambda) &\propto \left[\lambda^{2\pi 1/2}\exp\left(-\frac{\lambda\mu^2}{2}\right)\right]^\beta \\ &= \exp\left\{-\frac{\beta\lambda}{2}(\mu-c/\beta)^2\right\}\lambda^{\beta/2}\exp\left\{-\left(d-\frac{c^2}{2\beta}\right)\lambda\right\} \end{aligned} \tag{2.153}$$

where c, d, and β are constants. Since we can always write $p(\mu,\lambda) = p(\mu|\lambda)p(\lambda)$, we can find $p(\mu|\lambda)$ and $p(\lambda)$ by inspection. In particular, we see

that $p(\mu|\lambda)$ is a Gaussian whose precision is a linear function of λ and that $p(\lambda)$ is a gamma distribution, so that the normalized prior takes the form

$$p(\mu, \lambda) = \mathcal{N}(\mu|\mu_0, (\beta\lambda)^{-1})\text{Gam}(\lambda|a, b) \tag{2.154}$$

where we have defined new constants given by $\mu_0 = c/\beta, a = 1+\beta/2, b = d-c^2/2\beta$. The distribution (2.154) is called the *normal-gamma* or *Gaussian-gamma* distribution and is plotted in Figure [[fig:2.14]. Note that this is not simply the product of an independent Gaussian prior over μ and a gamma prior over λ, because the precision of μ is a linear function of λ. Even if we chose a prior in which μ and λ were independent, the posterior distribution would exhibit a coupling between the precision of μ and the value of λ.

```
fig = plt.figure(figsize=(10, 6.18))
ax = fig.add_subplot(111)
plot_mu = np.linspace(-2, 2, 100)
plot_lambda = np.linspace(0, 2, 100)
X, Y = np.meshgrid(plot_mu, plot_lambda)
mu_0, beta_, a, b = 0, 2, 5, 6
Z = normal.pdf(plot_mu, loc=mu_0, scale=1 / (beta_ * Y)) * \
    gamma.pdf(Y, a, scale=1/b)
ax.contourf(X,Y,Z,cmap="BuGn")
ax.set_ylabel("$\lambda$", fontsize=20, rotation=0)
ax.set_xlabel("$\mu$", fontsize=20)
fig.savefig("img/fig:2.14.png")
plt.close("all")
```

Listing 2.9: fig:2.14

In the case of the multivariate Gaussian distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Lambda^{-1})$ for a D-dimensional variable $\mathbf{x}$, the conjugate prior distribution for the mean $\boldsymbol{\mu}$, assuming the precision is known, is again a Gaussian. For known mean and unknown precision matrix Λ, the conjugate prior is the *Wishart* distribution given by

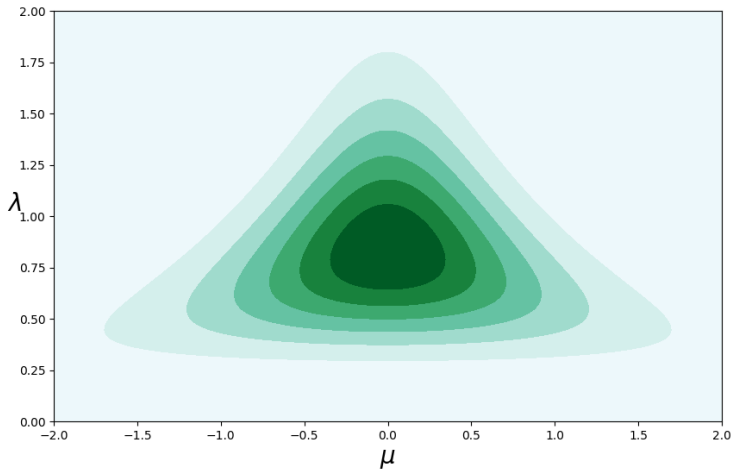


图 2.14: Contour plot of the normal-gamma distribution (2.154) for parameter values $\mu_0 = 0, \beta = 2, a = 5$ and $b = 6$.

$$\mathcal{W}(\Lambda|\mathbf{W}, \nu) = B|\Lambda|^{(-D-1)/2} \exp\left(-\frac{1}{2}\mathrm{Tr}(W^{-1}\Lambda)\right) \tag{2.155}$$

where is called the number of *degrees of freedom(自由度)* of the distribution, $\mathbf{W}$ is a $D \times D$ scale matrix, and $\mathrm{Tr}(\cdot)$ denotes the trace. The normalization constant B is given by

$$B(\mathbf{W}, \nu) = |\mathbf{W}|^{-\nu/2} \left(2^{\nu D/2} \pi^{D(D-1)/4} \prod_{i=1}^{D} \Gamma\left(\frac{\nu + 1 - i}{2}\right)\right)^{-1} \tag{2.156}$$

Again, it is also possible to define a conjugate prior over the covariance matrix itself, rather than over the precision matrix, which leads to the *inverse Wishart* distribution, although we shall not discuss this further. If both the mean and the precision are unknown, then, following a similar line of reasoning to the univariate case, the conjugate prior is given by

$$p(\boldsymbol{\mu}, \Lambda|\boldsymbol{\mu}_0, \beta, \mathbf{W}, \nu) = \mathcal{N}(\boldsymbol{\mu}|\mu_0, (\beta\Lambda)^{-1})\mathcal{W}(\Lambda|\mathcal{W}, \nu) \tag{2.157}$$

which is known as the *normal-Wishart* or *Gaussian-Wishart* distribution.

2.3.7 Student's t-distribution

We have seen that the conjugate prior for the precision of a Gaussian is given by a gamma distribution. If we have a univariate Gaussian $\mathcal{N}(x|\mu, \tau^{-1})$ together with a Gamma prior $\mathrm{Gam}(\tau|a, b)$ and we integrate out the precision, we obtain the marginal distribution of x in the form

$$\begin{aligned} p(x|\mu, a, b) &= \int_{-\infty}^{\infty} \mathcal{N}(x|\mu, \tau^{-1})\mathrm{Gam}(\tau|a, b)d\tau \\ &= \int_{-\infty}^{\infty} \frac{b^a e^{-b\tau}\tau^{a-1}}{\Gamma(a)} \left(\frac{\tau}{2\pi}\right)^{1/2} \exp\left\{-\frac{\tau}{2}(x-\mu)^2\right\} d\tau \\ &= \frac{b^a}{\Gamma(a)} \left(\frac{1}{2\pi}\right)^{1/2} \left[b + \frac{(x-\mu)^2}{2}\right]^{-a-1/2} \end{aligned} \tag{2.158}$$

where we have made the change of variable $z = \tau[b + (x-\mu)^2/2]$. By convention we define new parameters given by $\nu = 2a$ and $\lambda = a/b$, in terms of which the distribution $p(x|\mu, a, b)$ takes the form

$$\mathrm{St}(x|\mu, \lambda, \nu) = \frac{\Gamma(\nu/2 + 1/2)}{\Gamma(\nu/2)} \left(\frac{\lambda}{\pi\nu}\right)^{1/2} \left[1 + \frac{\lambda(x-\mu)^2}{\nu}\right]^{-\nu/2-1/2} \tag{2.159}$$

which is known as Student's t-distribution. The parameter λ is sometimes called the *precision(精度)* of the t-distribution, even though it is not in general equal to the inverse of the variance. The parameter ν is called the *degrees of freedom(自由度)*, and its effect is illustrated in Figure 2.15. For the particular case of $\nu = 1$, the t-distribution reduces to the Cauchy distribution, while in the limit $\nu \to \infty$ the t-distribution $\mathrm{St}(x|\mu, \lambda, \nu)$ becomes a Gaussian $\mathcal{N}(x|\mu, \lambda^{-1})$ with mean μ and precision λ.

From (2.158), we see that Student's t-distribution is obtained by adding up an infinite number of Gaussian distributions having the same mean but

```
x_plot = np.linspace(-5, 5, 1000)
fig = plt.figure(figsize=(10, 6.18))
ax = fig.add_subplot(111)
ax.set_xlim(-5, 5)
ax.set_ylim(0, 0.5)
for (df, color) in zip([0.1, 1, 1e10], ["r", "b", "g"]):
    y_plot = t.pdf(x_plot, df=df)
    ax.plot(x_plot, y_plot, color, label="$\\nu={0}$".format(df))
ax.legend(loc=2)
fig.savefig("img/fig:2.15.png")
plt.close("all")
```

Listing 2.10: fig:2.15

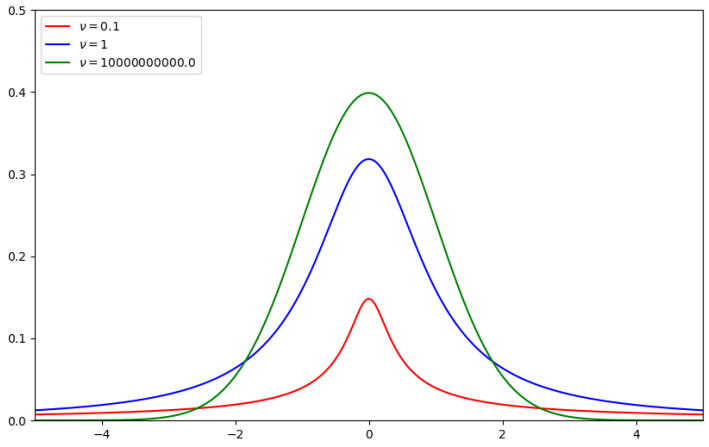


图 2.15: Plot of Student's t-distribution (2.159) for $\mu = 0$ and $\lambda = 1$ for various values of ν. The limit $\nu \to \infty$ corresponds to a Gaussian distribution with mean μ and precision λ.

different precisions. This can be interpreted as an **infinite mixture of Gaussians** (Gaussian mixtures will be discussed in detail in Section 2.3.9. The result is a distribution that in general has longer 'tails'than a Gaussian, as was seen in Figure 2.15. This gives the t-distribution an important property called *robustness(鲁棒性)*, which means that it is much less sensitive than the Gaussian to the presence of a few data points which are *outliers(离群值)*. The robustness of the t-distribution is illustrated in Figure [[fig:2.16], which compares the maximum likelihood solutions for a Gaussian and a t-distribution. Note that the maximum likelihood solution for the t-distribution can be found using the expectation-maximization (EM) algorithm. Here we see that the effect of a small number of outliers is much less significant for the t-distribution than for the Gaussian. Outliers can arise in practical applications either because the process that generates the data corresponds to a distribution having a heavy tail or simply through mislabelled data. Robustness is also an important property for regression problems. Unsurprisingly, the least squares approach to regression does not exhibit robustness, because it corresponds to maximum likelihood under a (conditional) Gaussian distribution. By basing a regression model on a heavy-tailed distribution such as a t-distribution, we obtain a more robust model.

If we go back to (2.158) and substitute the alternative parameters = 2a, = a/b, and = b/a, we see that the t-distribution can be written in the form

$$\mathrm{St}(x|\mu,\lambda,\nu) = \int_0^\infty \mathcal{N}(x|\mu,(\eta\lambda)^{-1})\mathrm{Gam}(\eta|\nu/2,\nu/2)d\eta \tag{2.160}$$

We can then generalize this to a multivariate Gaussian N (x|,) to obtain the cor- responding multivariate Student's t-distribution in the form

$$\mathrm{St}(\mathbf{x}|\boldsymbol{\mu},\Lambda,\nu) = \int_0^\infty \mathcal{N}(\mathbf{x}|\boldsymbol{\mu},(\eta\Lambda)^{-1})\mathrm{Gam}(\eta|\nu/2,\nu/2)d\eta \tag{2.161}$$

```
X1 = normal.rvs(size=20, random_state=1234)
X2 = np.concatenate([X1, normal.rvs(loc=20., size=3, random_state=1234)])
fig = plt.figure(figsize=(12, 6))
for index in range(2):
    X = eval("X{0}".format(index+1))
    tdof, tloc, tscale = t.fit(X)
    nloc, nscale = normal.fit(X)
    x = np.linspace(-5, 25, 1000)
    ax = fig.add_subplot(1, 2, index+1)
    ax.plot(x, t.pdf(x, df=tdof, loc=tloc, scale=tscale),
            "r", label="student's t", linewidth=2)
    ax.plot(x, normal.pdf(x, loc=nloc, scale=nscale),
            "g", label="gaussian", linewidth=1)
    ax.hist(X, range=(-5, 25), bins=50, normed=1,
            label="samples", rwidth=0.8, color="navy")
    ax.legend()
fig.savefig("img/fig:2.16.png")
plt.close("all")
```

Listing 2.11: fig:2.16

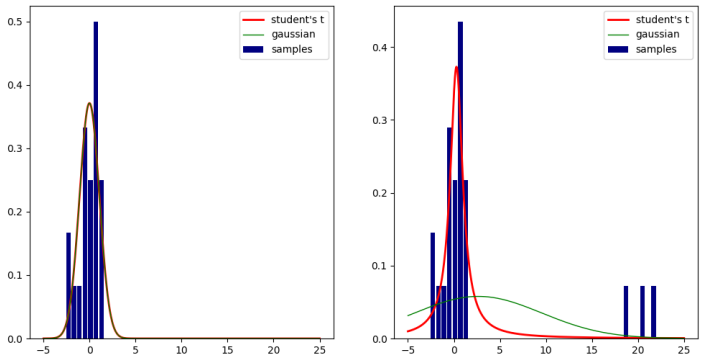


图 2.16: Illustration of the robustness of Student's t-distribution compared to a Gaussian. (a) Histogram distribution of 30 data points drawn from a Gaussian distribution, together with the maximum likelihood fit obtained from a t-distribution (red curve) and a Gaussian (green curve, largely hidden by the red curve). Because the t-distribution contains the Gaussian as a special case it gives almost the same solution as the Gaussian. (b) The same data set but with three additional outlying data points showing how the Gaussian (green curve) is strongly distorted by the outliers, whereas the t-distribution (red curve) is relatively unaffected.

Using the same technique as for the univariate case, we can evaluate this integral to give

$$\mathrm{St}(\mathbf{x}|\boldsymbol{\mu}, \Lambda, \nu) = \frac{\Gamma(D/2 + \nu/2)}{\Gamma(\nu/2)} \frac{|\Lambda|^{1/2}}{(\pi\nu)^{D/2}} \left[1 + \frac{\Delta^2}{\nu}\right]^{-D/2-\nu/2} \tag{2.162}$$

where D is the dimensionality of $\mathbf{x}$, and Δ^2 is the squared Mahalanobis distance defined by

$$\Delta^2 = (\mathbf{x} - \boldsymbol{\mu})^T \Lambda (\mathbf{x} - \boldsymbol{\mu}) \tag{2.163}$$

This is the multivariate form of Student's t-distribution and satisfies the following properties

$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu} \qquad \text{if} \quad \nu > 1 \tag{2.164}$$

$$\mathrm{cov}[\mathbf{x}] = \frac{\nu}{(\nu - 2)} \Lambda^{-1} \qquad \text{if} \quad \nu > 2 \tag{2.165}$$

$$\mathrm{mode}[\mathbf{x}] = \boldsymbol{\mu} \tag{2.166}$$

with corresponding results for the univariate case.

2.3.8 Periodic variables

Although Gaussian distributions are of great practical significance, both in their own right and as building blocks for more complex probabilistic models, there are situations in which they are inappropriate as density models for continuous variables. One important case, which arises in practical applications, is that of periodic variables.

An example of a periodic variable would be the wind direction at a particular geographical location. We might, for instance, measure values of wind direction on a number of days and wish to summarize this using a parametric distribution. Another example is calendar time, where we may be interested in modeling quantities that are believed to be periodic

over 24 hours or over an annual cycle. Such quantities can conveniently be represented using an angular (polar) coordinate $0 < \theta < 2\pi$.

We might be tempted to treat periodic variables by choosing some direction as the origin and then applying a conventional distribution such as the Gaussian. Such an approach, however, would give results that were strongly dependent on the arbitrary choice of origin. Suppose, for instance, that we have two observations at $\theta_1 = 1^\circ$ and $\theta_2 = 359^\circ$, and we model them using a standard univariate Gaussian distribution. If we choose the origin at 0°, then the sample mean of this data set will be 180° with standard deviation 179°, whereas if we choose the origin at 180°, then the mean will be 0° and the standard deviation will be 1°. We clearly need to develop a special approach for the treatment of periodic variables.

Let us consider the problem of evaluating the mean of a set of observations $\mathcal{D} = \{\theta_1, \cdots, \theta_N\}$ of a periodic variable. From now on, we shall assume that θ is measured in radians. We have already seen that the simple average $(\theta_1 + \cdots + \theta_N)/N$ will be strongly coordinate dependent. To find an invariant measure of the mean, we note that the observations can be viewed as points on the unit circle and can therefore be described instead by two-dimensional unit vectors $x_1, \cdots, x_N$ where $\| x_n \| = 1$ for $n = 1, \cdots, N$, as illustrated in Figure 2.17. We can average the vectors $\{x_n\}$ instead to give

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} x_n \tag{2.167}$$

and then find the corresponding angle θ of this average. Clearly, this definition will ensure that the location of the mean is independent of the origin of the angular coordinate. Note that $\mathbf{x}$ will typically lie inside the unit circle. The Cartesian coordinates of the observations are given by $x_n = (\cos\theta_n, \sin\theta_n)$, and we can write the Cartesian coordinates of the sample mean in the form $x = (r\cos\theta, r\sin\theta)$. Substituting into (**??**) and equating the x_1 and x_2 components then gives

$$\bar{r} = \cos\bar{\theta} = \frac{1}{N}\sum_{n=1} N\cos\theta_n \quad \bar{r} = \sin\bar{\theta} = \frac{1}{N}\sum_{n=1} N\sin\theta_n \tag{2.168}$$

Taking the ratio, and using the identity $\tan\theta = \sin\theta/\cos\theta$, we can solve for θ to give

$$\bar{\theta} = \tan^{-1}\left\{\frac{\sum_n \sin\theta_n}{\sum_n \cos\theta_n}\right\} \tag{2.169}$$

Shortly, we shall see how this result arises naturally as the maximum likelihood estimator for an appropriately defined distribution over a periodic variable.

We now consider a periodic generalization of the Gaussian called the *von Mises* distribution. Here we shall limit our attention to univariate distributions, although periodic distributions can also be found over hyperspheres of arbitrary dimension. For an extensive discussion of periodic distributions, see Mardia and Jupp (2000).

By convention, we will consider distributions $p(\theta)$ that have period 2π. Any probability density $p(\theta)$ defined over θ must not only be nonnegative and integrate to one, but it must also be periodic. Thus $p(\theta)$ must satisfy the three conditions

$$p(\theta) \geqslant 0 \tag{2.170}$$

$$\int_0^{2\pi} p(\theta)d\theta = 1 \tag{2.171}$$

$$p(\theta + 2\pi) = p(\theta) \tag{2.172}$$

From (2.172), it follows that $p(\theta + M2\pi) = p(\theta)$ for any integer M.

We can easily obtain a Gaussian-like distribution that satisfies these three properties as follows. Consider a Gaussian distribution over two variables $\mathbf{x} = (x_1, x_2)$ having mean $\boldsymbol{\mu} = (\mu_1, \mu_2)$ and a covariance matrix $\Sigma = \sigma^2 I$ where I is the 2 Œ 2 identity matrix, so that

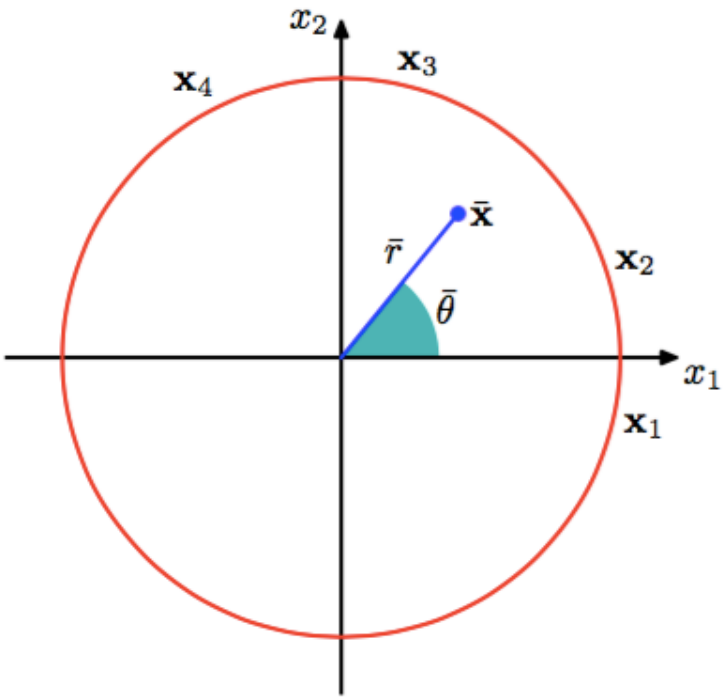


图 2.17: Illustration of the representation of values θ_n of a periodic variable as two-dimensional vectors $\mathbf{x}_n$ living on the unit circle. Also shown is the average $\mathbf{x}$ of those vectors.

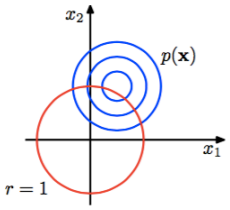


图 2.18: The von Mises distribution can be derived by considering a two-dimensional Gaussian of the form (2.173), whose density contours are shown in blue and conditioning on the unit circle shown in red.

$$p(x_1, x_2) = \frac{1}{2\pi\sigma^2} \exp\left\{-\frac{(x_1 - \mu_1)^2 + (x_2 - \mu_2)^2}{2\sigma^2}\right\} \tag{2.173}$$

The contours of constant $p(\mathbf{x})$ are circles, as illustrated in Figure 2.18 Now suppose we consider the value of this distribution along a circle of fixed radius. Then by construction this distribution will be periodic, although it will not be normalized. We can determine the form of this distribution by transforming from Cartesian coordinates (x_1, x_2) to polar coordinates (r, θ) so that

$$x_1 = r\cos\theta, \quad x_2 = r\sin\theta. \tag{2.174}$$

We also map the mean $\boldsymbol{\mu}$ into polar coordinates by writing

$$\mu_1 = r_0\cos\theta, \quad \mu_2 = r_0\cos\theta. \tag{2.175}$$

Next we substitute these transformations into the two-dimensional Gaussian distribution (2.173), and then condition on the unit circle $r = 1$, noting that we are interested only in the dependence on θ. Focusing on the exponent in the Gaussian distribution we have

$$\begin{aligned} &-\frac{1}{2\sigma^2}\{(r\cos\theta - r_0\cos\theta_0)^2 + (r\sin\theta - r_0\sin\theta_0)^2\} \\ &\quad = -\frac{1}{2\sigma^2}\{1 + r_0^2 - 2r_0\cos\theta\cos\theta_0 - 2r_0\sin\theta\sin\theta_0\} \\ &\quad = \frac{r_0}{\sigma^2}\cos(\theta - \theta_0) + \text{const} \end{aligned} \tag{2.176}$$

where 'const'denotes terms independent of θ, and we have made use of the following trigonometrical identities

$$\cos^2 A + \sin^2 A = 1 \tag{2.177}$$

$$\cos A\cos B + \sin A\sin B = \cos(A - B) \tag{2.178}$$

If we now define $m = r_0/\sigma^2$, we obtain our final expression for the distribution of $p(\theta)$ along the unit circle $r = 1$ in the form

$$p(\theta|\theta_0, m) = \frac{1}{2\pi I_0(m)} \exp\{m\cos(\theta - \theta_0)\} \tag{2.179}$$

which is called the von Mises distribution, or the *circular normal(环形正态分布)*. Here the parameter θ_0 corresponds to the mean of the distribution, while m, which is known as the *concentration* parameter, is analogous to the inverse variance (precision) for the Gaussian. The normalization coefficient in (2.179) is expressed in terms of $I_0(m)$, which is the zeroth-order Bessel function of the first kind (Abramowitz and Stegun, 1965) and is defined by

$$I_0(m) = \frac{1}{2\pi} \int_0^{2\pi} \exp\{m\cos\theta\} d\theta \tag{2.180}$$

For large m, the distribution becomes approximately Gaussian. The von Mises distribution is plotted in Figure 2.19, and the function $I_0(m)$ is plotted in Figure 2.20.

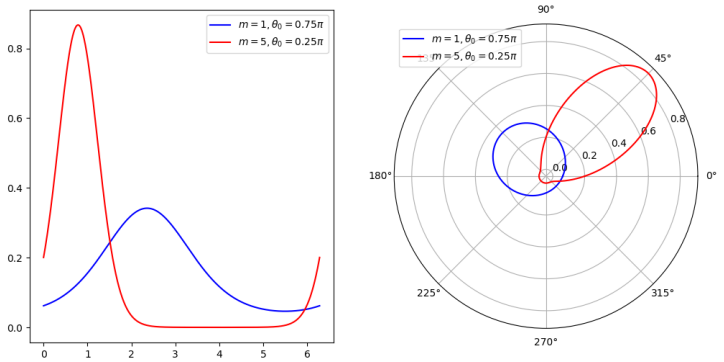


图 2.19: The von Mises distribution plotted for two different parameter values, shown as a Cartesian plot on the left and as the corresponding polar plot on the right.

```
fig = plt.figure(figsize=(12, 6))
x_plot = np.linspace(0, 2*np.pi, 10000)
M = [1, 5]
Theta_0 = np.pi * np.array([0.75, 0.25])
Color = ["b", "r"]
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2, projection="polar")
for index, (m, theta_0, color) in enumerate(zip(M, Theta_0, Color)):
    y_plot = vonmises.pdf(x_plot, m, loc=theta_0)
    ax1.plot(x_plot, y_plot, color, label="$m={0}, \\theta_0={1}\\pi$".format(
        m, round(theta_0/np.pi, 2)))
    ax2.plot(x_plot, y_plot, color, label="$m={0}, \\theta_0={1}\\pi$".format(
        m, round(theta_0/np.pi, 2)))
ax1.legend()
ax2.legend()
fig.savefig("img/fig:2.19.png")
plt.close("all")
```

Listing 2.12: fig:2.19

```
fig = plt.figure(figsize=(12, 6))
x_plot = np.linspace(0, 10, 100)
ax1 = fig.add_subplot(121)
y_plot = i0(x_plot)
ax1.plot(x_plot, y_plot, "r")
ax2 = fig.add_subplot(122)
y_plot = i1(x_plot) / i0(x_plot)
ax2.plot(x_plot, y_plot, "r")
plt.savefig("img/fig:2.20.png")
plt.close("all")
```

Listing 2.13: 2.20

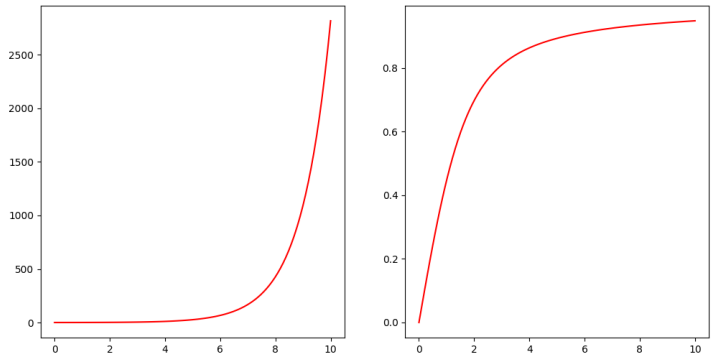


图 2.20: Plot of the Bessel function $I_0(m)$ defined by (2.180), together with the function A(m) defined by (2.186).

Now consider the maximum likelihood estimators for the parameters θ_0 and m for the von Mises distribution. The log likelihood function is given by

$$\ln p(\mathcal{D}|\theta_0, m) = -N \ln(2\pi) - N \ln I_0(m) + m \sum_{n=1}^{N} \cos(\theta_n - \theta_0) \tag{2.181}$$

Setting the derivative with respect to θ_0 equal to zero gives

$$\sum_{n=1}^{N} \sin(\theta_n - \theta_0) = 0. \tag{2.182}$$

To solve for θ_0, we make use of the trigonometric identity

$$\sin(A - B) = \cos B \sin A - \cos A \sin B \tag{2.183}$$

from which we obtain

$$\theta_0^{ML} = \tan^{-1} \left\{ \frac{\sum_n \sin \theta_0}{\sum_n \cos \theta_0} \right\} \tag{2.184}$$

which we recognize as the result (2.169) obtained earlier for the mean of the observations viewed in a two-dimensional Cartesian space.

Similarly, maximizing (2.181) with respect to m, and making use of $I_0'(m) = I_1(m)$ (Abramowitz and Stegun, 1965), we have

$$A(m) = \frac{1}{N}\sum_{n=1}^{N}\cos(\theta_n - \theta_0^{ML}) \tag{2.185}$$

where we have substituted for the maximum likelihood solution for θ_{ML} (recalling that we are performing a joint optimization over θ and m), and we have defined

$$A(m) = \frac{I_1(m)}{I_0(m)}. \tag{2.186}$$

The function $A(m)$ is plotted in Figure 2.20. Making use of the trigonometric (2.178), we can write (2.185) in the form

$$A(m_{ML}) = \left(\frac{1}{N}\sum_{n=1}^{N}\cos\theta_n\right)\cos\theta_0^{ML} - \left(\frac{1}{N}\sum_{n=1}^{N}\sin\theta_n\right)\sin\theta_0^{ML} \tag{2.187}$$

The right-hand side of (2.187) is easily evaluated, and the function $A(m)$ can be inverted numerically.

For completeness, we mention briefly some alternative techniques for the construction of periodic distributions. The simplest approach is to use a histogram of observations in which the angular coordinate is divided into fixed bins. This has the virtue of simplicity and flexibility but also suffers from significant limitations, as we shall see when we discuss histogram methods in more detail in Section 2.5. Another approach starts, like the von Mises distribution, from a Gaussian distribution over a Euclidean space but now marginalizes onto the unit circle rather than conditioning (Mardia and Jupp, 2000). However, this leads to more complex forms of distribution and will not be discussed further. Finally, any valid distribution over the real axis (such as a Gaussian) can be turned into a periodic distribution by

mapping successive intervals of width 2π onto the periodic variable $(0, 2\pi)$, which corresponds to 'wrapping'the real axis around unit circle. Again, the resulting distribution is more complex to handle than the von Mises distribution.

One limitation of the von Mises distribution is that it is unimodal. By forming mixtures of von Mises distributions, we obtain a flexible framework for modelling periodic variables that can handle multimodality. For an example of a machine learning application that makes use of von Mises distributions, see Lawrence et al. (2002), and for extensions to modelling conditional densities for regression problems, see Bishop and Nabney (1996).

2.3.9 Mixtures of Gaussians

While the Gaussian distribution has some important analytical properties, it suffers from significant limitations when it comes to modeling real data sets. Consider the example shown in Figure 2.21. This is known as the 'Old Faithful'data set, and comprises 272 measurements of the eruption of the Old Faithful geyser at Yellowstone National Park in the USA. Each measurement comprises the duration of the eruption in minutes (horizontal axis) and the time in minutes to the next eruption (vertical axis). We see that the data set forms two dominant clumps(块), and that a simple Gaussian distribution is unable to capture this structure, whereas a linear superposition of two Gaussians gives a better characterization of the data set.

Such superpositions(叠加), formed by taking linear combinations of more basic distributions such as Gaussians, can be formulated as probabilistic models known as *mixture distributions(混合模型)* (McLachlan and Basford, 1988; McLachlan and Peel, 2000). In Figure 2.22 we see that a linear combination of Gaussians can give rise to(产生) very complex densities. By using a sufficient number of Gaussians, and by adjusting their means and covariances as well as the coefficients in the linear combination, almost

```
x1 = multivariate_normal.rvs(
    mean=[2, 55],
    cov=np.diag([0.2, 40]) @ Rot_(5 * np.pi / 180),
    size=100,
    random_state=1234)
x2 = multivariate_normal.rvs(
    mean=[5, 80],
    cov=np.diag([0.4, 40]) @ Rot_(10 * np.pi / 180),
    size=100,
    random_state=4321)
X_train = np.vstack((x1, x2))
fig = plt.figure(figsize=(12, 6))
x = np.linspace(1, 6)
y = np.linspace(40, 100)
X, Y = np.meshgrid(x, y)
XX = np.array([X.ravel(), Y.ravel()]).T
model = GaussianMixture(n_components=1)
model.fit(X_train)
Z1 = -model.score_samples(XX)
Z1 = Z1.reshape(X.shape)
ax1 = fig.add_subplot(121)
ax1.contour(X, Y, Z1, colors="b", levels=np.logspace(0.61, 0.7, 3))
ax1.scatter(X_train[:, 0], X_train[:, 1], facecolor="none", edgecolor="g")
ax1.set_xlim(1, 6)
ax1.set_ylim(40, 100)
model = GaussianMixture(n_components=2)
model.fit(X_train)
Z2 = -model.score_samples(XX)
Z2 = Z2.reshape(X.shape)
ax2 = fig.add_subplot(122)
ax2.contour(X, Y, Z2, colors="b", levels=np.logspace(0.61, 0.8, 3))
ax2.scatter(X_train[:, 0], X_train[:, 1], facecolor="none", edgecolor="g")
ax2.set_xlim(1, 6)
ax2.set_ylim(40, 100)
fig.savefig("img/fig:2.21.png")
plt.close("all")
```

Listing 2.14: fig:2.21

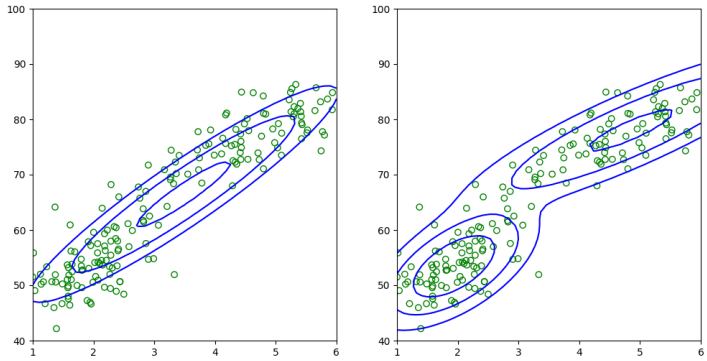


图 2.21: Plots of the 'old faithful'data(synthetic data actually) in which the blue curves show contours of constant probability density. On the left is a single Gaussian distribution which has been fitted to the data using maximum likelihood. Note that this distribution fails to capture the two clumps in the data and indeed places much of its probability mass in the central region between the clumps where the data are relatively sparse. On the right the distribution is given by a linear combination of two Gaussians which has been fitted to the data by maximum likelihood using techniques discussed Chapter 9, and which gives a better rep- resentation of the data.

any continuous density can be approximated to arbitrary accuracy.

```
x_plot = np.linspace(0, 10, 1000)
Loc = [2, 4, 6]
Scale = [0.5, 1, 0.2]
fig = plt.figure(figsize=(10, 6.18))
ax = fig.add_subplot(111)
Y = np.zeros(x_plot.shape)
for loc, scale in zip(Loc, Scale):
    y = normal.pdf(x_plot, loc=loc, scale=scale)
    ax.plot(x_plot, y, "b")
    Y += y
ax.plot(x_plot, Y, "r")
fig.savefig("img/fig:2.22.png")
plt.close("all")
```

Listing 2.15: fig:2.22

We therefore consider a superposition of K Gaussian densities of the form

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k) \tag{2.188}$$

which is called a *mixture of Gaussians(混合高斯)*. Each Gaussian density $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k)$ is called a *component(成分)* of the mixture and has its own mean $\boldsymbol{\mu}_k$ and covariance Σ_k. Contour and surface plots for a Gaussian mixture having 3 components are shown in Figure 2.23

In this section we shall consider Gaussian components to illustrate the framework of mixture models. More generally, mixture models can comprise linear combinations of other distributions. For instance, in Section 9.3.3(?) we shall consider mixtures of Bernoulli distributions as an example of a mixture model for discrete variables.

The parameters π_k in (2.188) are called *mixing coefficients(混合系数)*. If we integrate both sides of (2.188) with respect to $\mathbf{x}$, and note that both $p(\mathbf{x})$ and the individual Gaussian components are normalized, we obtain

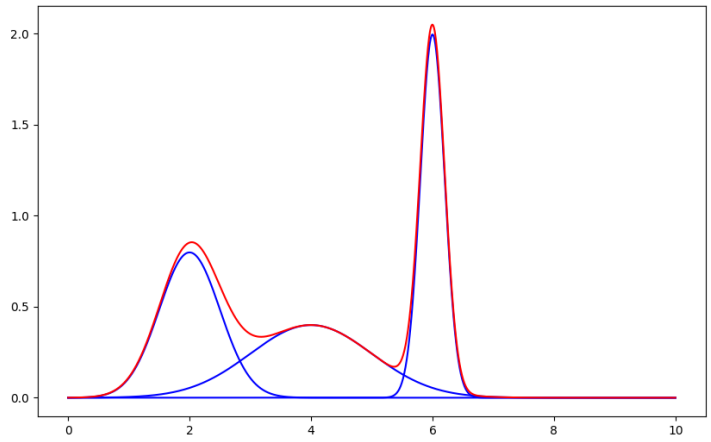


图 2.22: Example of a Gaussian mixture distribution $p(x)$ in one dimension showing three Gaussians (each scaled by a coefficient) in blue and their sum in red.

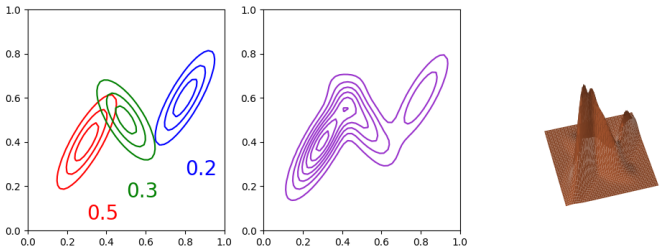


图 2.23: Illustration of a mixture of 3 Gaussians in a two-dimensional space. (a) Contours of constant density for each of the mixture components, in which the 3 components are denoted red, blue and green, and the values of the mixing coefficients are shown below each component. (b) Contours of the marginal probability density $p(\mathbf{x})$ of the mixture distribution. (c) A surface plot of the distribution $p(\mathbf{x})$.

```
fig = plt.figure(figsize=(12, 4))
x_plot = np.linspace(0, 1)
y_plot = np.linspace(0, 1)
X, Y = np.meshgrid(x_plot, y_plot)
Z = pd.DataFrame()
Mean = np.array([[0.3, 0.4], [0.5, 0.5], [0.8, 0.6]])
Color = ["r", "g", "b"]
Cov = np.stack((np.diag((.01, .02)) @ Rot(30), np.diag((.01, .015)) @ Rot(-30),
                np.diag((.01, .02)) @ Rot(30)))
Proportion = [0.5, 0.3, 0.2]
ax1 = fig.add_subplot(131)
Z = np.zeros(X.shape)
for index, (mean, color, cov, proportion) in enumerate(
        zip(Mean, Color, Cov, Proportion)):
    z = multivariate_normal.pdf(np.dstack((X, Y)), mean=mean, cov=cov)
    Z += proportion * z
    ax1.contour(X, Y, z, 3, colors=color)
    ax1.annotate(proportion, xy=mean - [0, 0.35], color=color, fontsize=20)
ax2 = fig.add_subplot(132)
ax2.contour(X, Y, Z, 7, colors="darkorchid")
ax3 = fig.add_subplot(133, projection="3d")
ax3.plot_surface(X, Y, Z, color="sienna")
ax3.view_init(45, -105)
ax3.axis("off")
ax3.grid(False)
fig.savefig("img/fig:2.23.png")
plt.close("all")
```

Listing 2.16: fig:2.23

$$\sum_{k=1}^{K} \pi_k = 1. \tag{2.189}$$

Also, the requirement that $p(\mathbf{x}) \geqslant 0$, together with $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k) \geqslant 0$, implies $\pi_k \geqslant 0$ for all k. Combining this with the condition (2.189) we obtain

$$0 \leqslant \pi_k \leqslant 1. \tag{2.190}$$

We therefore see that the mixing coefficients satisfy the requirements to be probabilities.

From the sum and product rules, the marginal density is given by

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(k)p(\mathbf{x}|k) \tag{2.191}$$

which is equivalent to (2.188) in which we can view $\pi_k = p(k)$ as the prior probability of picking the k^{th} component, and the density $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k) = p(\mathbf{x}|k)$ as the probability of $\mathbf{x}$ conditioned on k. As we shall see in later chapters, an important role is played by the posterior probabilities $p(k|\mathbf{x})$, which are also known as *responsibilities(责任)*. From Bayes' theorem these are given by

$$\begin{aligned} \gamma_k(\mathbf{x}) &\equiv p(k|\mathbf{x}) \\ &= \frac{p(k)p(\mathbf{x}|k)}{\sum_l p(l)p(\mathbf{x}|l)} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k)}{\sum_l \pi_l \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_l, \Sigma_l)}. \end{aligned} \tag{2.192}$$

We shall discuss the probabilistic interpretation of the mixture distribution in greater detail in Chapter 9(?).

The form of the Gaussian mixture distribution is governed by the parameters $\boldsymbol{\pi}, \boldsymbol{\mu}$ and Σ, where we have used the notation $\boldsymbol{\pi} \equiv \{\pi_1, \cdots, \pi_K\}, \boldsymbol{\mu} \equiv \{\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_K\}$ and $\Sigma \equiv \{\Sigma_1, \cdots, \Sigma_K\}$. One way to set the values of these

parameters is to use maximum likelihood. From (2.188) the log of the likelihood function is given by

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \Sigma) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \Sigma_k) \right\} \tag{2.193}$$

where $\mathbf{X} = \{\mathbf{x}1, \cdots, \mathbf{x}_N\}$. We immediately see that the situation is now much more complex than with a single Gaussian, due to the presence of the summation over k inside the logarithm. As a result, the maximum likelihood solution for the parameters no longer has a closed-form analytical solution. One approach to maximizing the likelihood function is to use iterative numerical optimization techniques (Fletcher, 1987; Nocedal and Wright, 1999; Bishop and Nabney, 2008). Alternatively we can employ a powerful framework called expectation maximization, which will be discussed at length in Chapter 9(?).

2.4 The Exponential Family

The probability distributions that we have studied so far in this chapter (with the exception of the Gaussian mixture) are specific examples of a broad class of distributions called the *exponential family(指数族)* (Duda and Hart, 1973; Bernardo and Smith, 1994). Members of the exponential family have many important properties in common, and it is illuminating to discuss these properties in some generality.

The exponential family of distributions over $\mathbf{x}$, given parameters $\boldsymbol{\eta}$, is defined to be the set of distributions of the form

$$p(\mathbf{x}|\boldsymbol{\eta}) = h(\mathbf{x})g(\boldsymbol{\eta}) \exp\{\boldsymbol{\eta}^T \mathbf{u}(\mathbf{x})\} \tag{2.194}$$

where $\mathbf{x}$ may be scalar or vector, and may be discrete or continuous. Here $\boldsymbol{\eta}$ are called the *natural parameters(自然参数)* of the distribution, and $\mathbf{u}(\mathbf{x})$ is some function of $\mathbf{x}$. The function $g(\boldsymbol{\eta})$ can be interpreted as the

coefficient that ensures that the distribution is normalized and therefore satisfies

$$g(\boldsymbol{\eta}) \int h(\mathbf{x}) \exp\{\boldsymbol{\eta}^T \mathbf{u}(\mathbf{x})\} d\mathbf{x} = 1 \tag{2.195}$$

where the integration is replaced by summation if $\mathbf{x}$ is a discrete variable.

We begin by taking some examples of the distributions introduced earlier in the chapter and showing that they are indeed members of the exponential family. Consider first the Bernoulli distribution

$$p(x|\mu) = \text{Bern}(x|\mu) = \mu^x (1-\mu)^{1-x}. \tag{2.196}$$

Expressing the right-hand side as the exponential of the logarithm, we have

$$\begin{aligned} p(x|\mu) &= \exp\{x \ln \mu + (1-x) \ln(1-\mu)\} \\ &= (1-\mu) \exp\left\{ \ln(\frac{\mu}{1-\mu}) x \right\}. \end{aligned} \tag{2.197}$$

Comparison with (2.194) allows us to identify

$$\eta = \ln(\frac{\mu}{1-\mu}) \tag{2.198}$$

which we can solve for μ to give $\mu = \sigma(\eta)$, where

$$\sigma(\eta) = \frac{1}{1 + \exp(-\eta)} \tag{2.199}$$

is called the *logistic sigmoid* function. Thus we can write the Bernoulli distribution using the standard representation (2.194) in the form

$$p(x|\eta) = \sigma(-\eta) \exp(\eta x) \tag{2.200}$$

where we have used $1 - \sigma(\eta) = \sigma(-\eta)$, which is easily proved from (2.199). Comparison with (2.194) shows that

$$u(x) = x \tag{2.201}$$

$$h(x) = 1 \tag{2.202}$$

$$g(\eta) = \sigma(-\eta) \tag{2.203}$$

Next consider the multinomial distribution that, for a single observation $\mathbf{x}$, takes the form

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{k=1}^{M} \mu_k^{x_k} = \exp\left\{\sum_{k=1}^{M} x_k \ln \mu_k\right\} \tag{2.204}$$

where $\mathbf{x} = (x_1, \cdots, x_N)^T$. Again, we can write this in the standard representation (2.194) so that

$$p(\mathbf{x}|\boldsymbol{\eta}) = \exp(\boldsymbol{\eta}^T \mathbf{x}) \tag{2.205}$$

where $\eta_k = \ln \mu_k$, and we have defined $\boldsymbol{\eta} = (\eta_1, \cdots, \eta_M)^T$. Again, comparing with (2.194) we have

$$\mathbf{u}(\mathbf{x}) = \mathbf{x} \tag{2.206}$$

$$h(\mathbf{x}) = 1 \tag{2.207}$$

$$g(\boldsymbol{\eta}) = 1. \tag{2.208}$$

Note that the parameters η_k are not independent because the parameters μ_k are subject to the constraint

$$\sum_{k=1}^{M} \mu_k = 1 \tag{2.209}$$

so that, given any $M-1$ of the parameters μ_k, the value of the remaining parameter is fixed. In some circumstances, it will be convenient to remove this constraint by expressing the distribution in terms of only $M-1$ parameters. This can be achieved by using the relationship (2.209) to eliminate

μ_M by expressing it in terms of the remaining $\{\mu_k\}$ where $k = 1, \cdots, M-1$, thereby leaving $M-1$ parameters. Note that these remaining parameters are still subject to the constraints

$$0 \leqslant \mu_k \leqslant 1, \quad \sum_{k=1}^{M-1} \mu_k \leqslant 1. \tag{2.210}$$

Making use of the constraint (2.209), the multinomial distribution in this representation then becomes

$$\begin{aligned} &\exp\left\{\sum_{k=1}^{M} x_k \ln \mu_k\right\} \\ &\quad = \exp\left\{\sum_{k=1}^{M-1} x_k \ln \mu_k \left(1 - \sum_{k=1}^{M-1} x_k\right) \ln\left(1 - \sum_{k=1}^{M-1} \mu_k\right)\right\} \\ &\quad = \exp\left\{\sum_{k=1}^{M-1} x_k \ln\left(\frac{\mu_k}{1 - \sum_{k=1}^{M-1} x_k}\right) + \ln\left(1 - \sum_{k=1}^{M-1} \mu_k\right)\right\} \end{aligned} \tag{2.211}$$

We now identify

$$\ln\left(\frac{\mu_k}{1 - \sum_j \mu_j}\right) = \eta_k \tag{2.212}$$

which we can solve for μ_k by first summing both sides over k and then rearranging and back-substituting to give

$$\mu_k = \frac{\exp(\eta_k)}{1 + \sum_j \exp(\eta_j)} \tag{2.213}$$

This is called the *softmax* function, or the *normalized exponential(*ソフトマックス関数*, or* 正規化指数*).* In this representation, the multinomial distribution therefore takes the form

$$p(\mathbf{x}|\boldsymbol{\eta}) = \left(1 + \sum_{k=1}^{M-1} \exp(\eta_k)\right)^{-1} \exp(\boldsymbol{\eta}^T \mathbf{x}) \tag{2.214}$$

This is standard form of exponential family, with parameter vector $\boldsymbol{\mu} = (\eta_1, \cdots, \eta_{M-1})^T$ in which

$$\mathbf{u}(\mathbf{x}) = \mathbf{x} \tag{2.215}$$

$$h(\mathbf{x}) = 1 \tag{2.216}$$

$$g(\boldsymbol{\eta}) = \left(1 + \sum_{k=1}^{M-1} \exp(\eta_k)\right)^{-1} \tag{2.217}$$

Finally, let us consider the Gaussian distribution. For the univariate Gaussian, we have

$$\begin{aligned} p(x|\mu, \sigma^2) &= \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} \\ &= \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}x^2 + \frac{\mu}{\sigma^2}x - \frac{1}{2\sigma^2}\mu^2\right\} \end{aligned} \tag{2.218}$$

which, after some simple rearrangement, can be cast in the standard exponential family form (2.194) with

$$\boldsymbol{\eta} = \begin{pmatrix} \mu/\sigma^2 \\ -1/2\sigma^2 \end{pmatrix} \tag{2.219}$$

$$\mathbf{u}(\mathbf{x}) = \begin{pmatrix} x \\ x^2 \end{pmatrix} \tag{2.220}$$

$$h(\mathbf{x}) = (2\pi)^{-1/2} \tag{2.221}$$

$$g(\boldsymbol{\eta}) = (-2\eta_2)^{1/2} \exp\left(\frac{\eta_1^2}{4\eta_2}\right). \tag{2.222}$$

2.4.1 Maximum likelihood and sufficient statistics

Let us now consider the problem of estimating the parameter vector $\boldsymbol{\eta}$ in the general exponential family distribution (2.194) using the technique

of maximum likelihood. Taking the gradient of both sides of (2.195) with respect to $\boldsymbol{\eta}$, we have

$$\begin{aligned}\nabla g(\boldsymbol{\eta}) \int h(\mathbf{x}) \exp\{\boldsymbol{\eta}^T \mathbf{u}(\mathbf{x})\} d\mathbf{x} \\ + g(\mu) \int h(\mathbf{x}) \exp\{\boldsymbol{\eta}^T \boldsymbol{\mu}(\mathbf{x})\} \mathbf{u}(\mathbf{x}) d\mathbf{x} = 0\end{aligned} \tag{2.223}$$

Rearranging, and making use again of (2.195) then gives

$$-\frac{1}{g(\boldsymbol{\eta})} \nabla g(\boldsymbol{\eta}) = g(\boldsymbol{\eta}) \int h(\mathbf{x}) \exp\{\boldsymbol{\eta}^T \mathbf{u}(\mathbf{x})\} \mathbf{u}(\mathbf{x}) d\mathbf{x} = \mathbb{E}[\mathbf{u}(\mathbf{x})] \tag{2.224}$$

where we have used (2.194). We therefore obtain the result

$$-\nabla \ln g(\boldsymbol{\eta}) = \mathbb{E}[\mathbf{u}(\mathbf{x})]. \tag{2.225}$$

Note that the covariance of $\mathbf{u}(\mathbf{x})$ can be expressed in terms of the second derivatives of $g(\boldsymbol{\eta})$, and similarly for higher order moments. Thus, provided we can normalize a distribution from the exponential family, we can always find its moments by simple differentiation.

Now consider a set of independent identically distributed data denoted by $\mathbf{X} = \{x_1, \cdots, x_n\}$, for which the likelihood function is given by

$$p(\mathbf{X}|\boldsymbol{\mu}) = \left(\prod_{n=1}^{N} h(\mathbf{x}_n) \right) g(\boldsymbol{\eta})^N \exp \left\{ \boldsymbol{\eta}^T \sum_{n=1}^{N} \mathbf{u}(\mathbf{x}_n) \right\}. \tag{2.226}$$

Setting the gradient of $\ln p(\mathbf{X}|\boldsymbol{\eta})$ with respect to $\boldsymbol{\eta}$ to zero, we get the following condition to be satisfied by the maximum likelihood estimator $\boldsymbol{\eta}_{ML}$

$$-\nabla \ln g(\boldsymbol{\eta}_{ML}) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{u}(\mathbf{x}_n) \tag{2.227}$$

which can in principle be solved to obtain $\boldsymbol{\eta}_{ML}$. We see that th e solution for the maximum likelihood estimator depends on the data only

through $\sum_n \mathbf{u}(\mathbf{x})$, which is therefore called the *sufficient statistic(充分统计量)* of the distribution (2.194). We do not need to store the entire data set itself but only the value of the sufficient statistic. For the Bernoulli distribution, for example, the function $\mathbf{u}(\mathbf{x})$ is given just by $\mathbf{x}$ and so we need only keep the sum of the data points $\{x_n\}$, whereas for the Gaussian $\mathbf{u}(x) = (x, x^2)^T$, and so we should keep both the sum of $\{x_n\}$ and the sum of $\{x_n^2\}$.

If we consider the limit $N \to \infty$, then the right-hand side of (2.227) becomes $\mathbb{E}[\mathbf{u}(\mathbf{x})]$, and so by comparing with (2.225) we see that in this limit $\boldsymbol{\eta}_{ML}$ will equal the true value $\boldsymbol{\eta}$.

In fact, this sufficiency property holds also for Bayesian inference, although we shall defer discussion of this until Chapter 8(?) when we have equipped ourselves with the tools of graphical models and can thereby gain a deeper insight into these important concepts.

2.4.2 Conjugate priors

We have already encountered the concept of a conjugate prior several times, for example in the context of the Bernoulli distribution (for which the conjugate prior is the beta distribution) or the Gaussian (where the conjugate prior for the mean is a Gaussian, and the conjugate prior for the precision is the Wishart distribution). In general, for a given probability distribution $p(\mathbf{x}|\boldsymbol{\eta})$, we can seek a prior $p(\boldsymbol{\eta})$ that is conjugate to the likelihood function, so that the posterior distribution has the same functional form as the prior. For any member of the exponential family (2.194), there exists a conjugate prior that can be written in the form

$$p(\boldsymbol{\eta}|\boldsymbol{\chi}, \nu) = f(\boldsymbol{\chi}, \nu) g(\boldsymbol{\eta})^\nu \exp\{\nu \boldsymbol{\eta}^T \boldsymbol{\chi}\} \tag{2.228}$$

where $f(\boldsymbol{\chi}, \nu)$ is a normalization coefficient, and $g(\boldsymbol{\eta})$ is the same function as appears in (2.194). To see that this is indeed conjugate, let us multiply the prior (2.228) by the likelihood function (2.226) to obtain the

posterior distribution, up to a normalization coefficient, in the form

$$p(\boldsymbol{\mu}|\mathbf{X}, \boldsymbol{\chi}, \nu) \propto g(\boldsymbol{\eta})^{\nu+N} \exp\left\{\boldsymbol{\eta}^T \left(\sum_{n=1}^{N} \mathbf{u}(\mathbf{x}_n) + \nu\boldsymbol{\chi}\right)\right\}. \tag{2.229}$$

This again takes the same functional form as the prior (2.228), confirming conjugacy(共轭性). Furthermore, we see that the parameter ν can be interpreted as a effective number of pseudo-observations(假想观测) in the prior, each of which has a value for the sufficient statistic $\mathbf{u}(\mathbf{x})$ given by $\boldsymbol{\chi}$.

2.4.3 Noninformative priors

2.5 Nonparametric Methods

2.5.1 Kernel density estimators

2.5.2 Nearest-neighbour methods

2.6 Exercises

2.7 Guide: Statistics (scipy.stats)

2.7.1 Random Variables

In the code samples below we assume that the `scipy.stats` package is imported as

```
from scipy import stats
```

and in some cases we assume that individual objects are imported as

```
from scipy.stats import norm
```

For consistency between Python 2 and Python 3, we'll also ensure that print is a function:

```
from __future__ import print_function
```

1. Getting Help

First of all, all distributions are accompanied with help functions. To obtain just some basic information we print the relevant docstring: `print(stats.norm.__doc__)`.

To find the support, i.e., upper and lower bound of the distribution, call:

```
print('bounds of distribution lower: %s, upper: %s' % (norm.a, norm.b))
```

```
bounds of distribution lower: -inf, upper: inf
```

We can list all methods and properties of the distribution with `dir(norm)`. As it turns out, some of the methods are private methods although they are not named as such (their name does not start with a leading underscore), for example `veccdf`, are only available for internal calculation (those methods will give warnings when one tries to use them, and will be removed at some point).

To obtain the real main methods, we list the methods of the frozen distribution. (We explain the meaning of a frozen distribution below).

```
rv = norm()
print(dir(rv))  # reformatted
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__form
```

Finally, we can obtain the list of available distribution through introspection:

```
dist_continu = [
    d for d in dir(stats)
    if isinstance(getattr(stats, d), stats.rv_continuous)
]
dist_discrete = [
    d for d in dir(stats) if isinstance(getattr(stats, d), stats.rv_discrete)
]
print('number of continuous distributions: %d' % len(dist_continu))
print('number of discrete distributions:   %d' % len(dist_discrete))
```

```
number of continuous distributions: 96
number of discrete distributions:   13
```

2. Common Methods

   The main public methods for continuous RVs are:

   rvs Random Variates

   pdf Probability Density Function

   cdf Cumulative Distribution Function

   sf Survival Function (1-CDF)

   ppf Percent Point Function (Inverse of CDF)

   isf Inverse Survival Function (Inverse of SF)

   stats Return mean, variance, (Fisher's) skew, or (Fisher's) kurtosis

   moment non-central moments of the distribution

   Let's take a normal RV as an example.

```
norm.cdf(0)
```

```
0.5
```

   To compute the `cdf` at a number of points, we can pass a list or a `numpy` array.

```
print(norm.cdf([-1., 0, 1]))
import numpy as np
print(norm.cdf(np.array([-1., 0, 1])))
```

```
[0.15865525 0.5        0.84134475]
[0.15865525 0.5        0.84134475]
```

   Thus, the basic methods such as `pdf`, `cdf`, and so on are vectorized.

   Other generally useful methods are supported too:

```
print(norm.mean(), norm.std(), norm.var())
print(norm.stats(moments="mv"))
```

```
0.0 1.0 1.0
(array(0.), array(1.))
```

To find the median of a distribution we can use the percent point function `ppf`, which is the inverse of the `cdf`:

```
norm.ppf(0.5)
```

```
0.0
```

To generate a sequence of random variates, use the size keyword argument:

```
norm.rvs(size=3)
```

```
array([ 1.71140892,  0.05670092, -0.50788142])
```

Note that drawing random numbers relies on generators from `numpy.random` package. In the example above, the specific stream of random numbers is not reproducible across runs. To achieve reproducibility, you can explicitly seed a global variable

```
np.random.seed(1234)
```

Relying on a global state is not recommended though. A better way is to use the $\text{random}_{\text{state}}$ parameter which accepts an instance of `numpy.random.RandomState` class, or an integer which is then used to seed an internal RandomState object:

```
norm.rvs(size=5, random_state=1234)
```

```
array([ 0.47143516, -1.19097569,  1.43270697, -0.3126519 , -0.72058873])
```

Don't think that `norm.rvs(5)` generates 5 variates:

```
norm.rvs(5)
```

```
5.471435163732493
```

Here, `5` with no keyword is being interpreted as the first possible keyword argument, `loc`, which is the first of a pair of keyword arguments taken by all continuous distributions. This brings us to the topic of the next subsection.

3. Shifting and Scaling

   All continuous distributions take `loc` and `scale` as keyword parameters to adjust the location and scale of the distribution, e.g. for the standard normal distribution the location is the mean and the scale is the standard deviation.

```
norm.stats(loc=3, scale=4, moments="mv")
```

```
(array(3.), array(16.))
```

   In many cases the standardized distribution for a random variable `X` is obtained through the transformation `(X - loc) / scale`. The default values are `loc = 0` and `scale = 1`.

   Smart use of loc and scale can help modify the standard distributions in many ways. To illustrate the scaling further, the `cdf` of an exponentially distributed `RV` with mean $1/\lambda$ is given by

$$F(x) = 1 - \exp(-\lambda x)$$

   By applying the scaling rule above, it can be seen that by taking `scale = 1./lambda` we get the proper scale.

```
from scipy.stats import expon
print(expon.mean(scale=3.))
```

```
3.0
```

> Note
>
> Distributions that take shape parameters may require more than simple application of `loc` and/or `scale` to achieve the desired form. For example, the distribution of 2-D vector lengths given a constant vector of length R perturbed(受扰动的) by independent $\mathcal{N}(0, \sigma^2)$ deviations(偏差) in each component is rice(R/σ, scale $= \sigma$). The first argument is a shape parameter that needs to be scaled along with x.

The uniform distribution is also interesting:

```
from scipy.stats import uniform
print(uniform.cdf([0, 1, 2, 3, 4, 5], loc=1, scale=4))
```

```
[0.   0.   0.25 0.5  0.75 1.  ]
```

Finally, recall from the previous paragraph that we are left with the problem of the meaning of `norm.rvs(5)`. As it turns out, calling a distribution like this, the first argument, i.e., the 5, gets passed to set the `loc` parameter. Let's see:

```
np.mean(norm.rvs(5, size=500))
```

```
5.009835510696999
```

Thus, to explain the output of the example of the last section: `norm.rvs(5)` generates a single normally distributed random variate with mean `loc=5`, because of the default `size=1`.

We recommend that you set `loc` and `scale` parameters explicitly, by passing the values as keywords rather than as arguments. Repetition can be minimized when calling more than one method of a given RV by using the technique of Freezing a Distribution, as explained below.

4. Shape Parameters

   While a general continuous random variable can be shifted and scaled with the `loc` and `scale` parameters, some distributions require additional shape parameters. For instance, the gamma distribution, with density

$$\gamma(x, a) = \frac{\lambda(\lambda x)^{a-1}}{\Gamma(a)} e^{-\lambda x},$$

   requires the shape parameter a. Observe that setting λ can be obtained by setting the scale keyword to $1/\lambda$.

   Let's check the number and name of the shape parameters of the gamma distribution. (We know from the above that this should be 1.)

```
from scipy.stats import gamma
print(gamma.numargs)
print(gamma.shapes)
```

```
1
a
```

   Now we set the value of the shape variable to 1 to obtain the exponential distribution, so that we compare easily whether we get the results we expect.

```
gamma(1, scale=2.).stats(moments="mv")
```

```
(array(2.), array(4.))
```

   Notice that we can also specify shape parameters as keywords:

```
gamma(a=1, scale=2.).stats(moments="mv")
```

```
(array(2.), array(4.))
```

5. Freezing a Distribution

   Passing the `loc` and `scale` keywords time and again can become quite bothersome(令人讨厌的). The concept of freezing a RV is used to solve such problems.

```
rv = gamma(1, scale=2.)
```

   By using `rv` we no longer have to include the scale or the shape parameters anymore. Thus, distributions can be used in one of two ways, either by passing all distribution parameters to each method call (such as we did earlier) or by freezing the parameters for the instance of the distribution. Let us check this:

```
rv.mean(), rv.std()
```

```
(2.0, 2.0)
```

   This is indeed what we should get.

6. Broadcasting

   The basic methods `pdf` and so on satisfy the usual `numpy` broadcasting rules. For example, we can calculate the critical values for the upper tail of the t distribution for different probabilities and degrees of freedom.

```
stats.t.isf([0.1, 0.05, 0.01], [[10], [11]])
```

```
array([[1.37218364, 1.81246112, 2.76376946],
       [1.36343032, 1.79588482, 2.71807918]])
```

   Here, the first row are the critical values for 10 degrees of freedom and the second row for 11 degrees of freedom (d.o.f.). Thus, the broadcasting rules give the same result of calling `isf` twice:

```
print(stats.t.isf([0.1, 0.05, 0.01], 10))
print(stats.t.isf([0.1, 0.05, 0.01], 11))
```

```
[1.37218364 1.81246112 2.76376946]
[1.36343032 1.79588482 2.71807918]
```

If the array with probabilities, i.e., `[0.1, 0.05, 0.01]` and the array of degrees of freedom i.e., `[10, 11, 12]`, have the same array shape, then element wise matching is used. As an example, we can obtain the 10% tail for 10 d.o.f., the 5% tail for 11 d.o.f. and the 1% tail for 12 d.o.f. by calling

```
stats.t.isf([0.1, 0.05, 0.01], [10, 11, 12])
```

```
array([1.37218364, 1.79588482, 2.68099799])
```

7. **TODO** Specific Points for Discrete Distributions

   Discrete distribution have mostly the same basic methods as the continuous distributions. However `pdf` is replaced the probability mass function `pmf`, no estimation methods, such as `fit`, are available, and `scale` is not a valid keyword parameter. The location parameter, keyword `loc` can still be used to shift the distribution.

   The computation of the `cdf` requires some extra attention. In the case of continuous distribution the cumulative distribution function is in most standard cases strictly monotonic increasing in the bounds (a, b) and has therefore a unique inverse. The `cdf` of a discrete distribution, however, is a step function, hence the inverse `cdf`, i.e., the percent point function, requires a different definition:

   ```
   ppf(q) = min{x : cdf(x) >= q, x integer}
   ```

   For further info, see the docs here.

   We can look at the hypergeometric distribution as an example

   ```
   from scipy.stats import hypergeom
   [M, n, N] = [20, 7, 12]
   ```

If we use the `cdf` at some integer points and then evaluate the `ppf` at those `cdf` values, we get the initial integers back, for example

```
x = np.arange(4)*2
print(x)
prb = hypergeom.cdf(x, M, n, N)
print(prb)
print(hypergeom.ppf(prb, M, n, N))
```

```
[0 2 4 6]
[1.03199174e-04 5.21155831e-02 6.08359133e-01 9.89783282e-01]
[0. 2. 4. 6.]
```

If we use values that are not at the kinks of the `cdf` step function, we get the next higher integer back:

```
print(hypergeom.ppf(prb + 1e-8, M, n, N))
print(hypergeom.ppf(prb - 1e-8, M, n, N))
```

```
[1. 3. 5. 7.]
[0. 2. 4. 6.]
```

8. Fitting Distributions

The main additional methods of the not frozen distribution are related to the estimation of distribution parameters:

fit maximum likelihood estimation of distribution parameters, including location

and scale

fit$_{\text{locscale}}$ estimation of location and scale when shape parameters are given

nnlf negative log likelihood function

expect Calculate the expectation of a function against the `pdf` or `pmf`

9. Performance Issues and Cautionary Remarks

   The performance of the individual methods, in terms of speed, varies widely by distribution and method. The results of a method are obtained in one of two ways: either by explicit calculation, or by a generic algorithm that is independent of the specific distribution.

   Explicit calculation, on the one hand, requires that the method is directly specified for the given distribution, either through analytic formulas or through special functions in `scipy.special` or `numpy.random` for `rvs`. These are usually relatively fast calculations.

   The generic methods, on the other hand, are used if the distribution does not specify any explicit calculation. To define a distribution, only one of `pdf` or `cdf` is necessary; all other methods can be derived using numeric integration and root finding. However, these indirect methods can be very slow. As an example, `rgh = stats.gausshyper.rvs(0.5, 2, 2, 2, size=100)` creates random variables in a very indirect way and takes about 19 seconds for 100 random variables on my computer, while one million random variables from the standard normal or from the t distribution take just above one second.

10. Remaining Issues

    The distributions in `scipy.stats` have recently been corrected and improved and gained a considerable test suite, however a few issues remain:

    - the distributions have been tested over some range of parameters, however in some corner ranges, a few incorrect results may remain.
    - the maximum likelihood estimation in fit does not work with default starting parameters for all distributions and the user needs

to supply good starting parameters. Also, for some distribution using a maximum likelihood estimator might inherently not be the best choice.

2.7.2 Building Specific Distributions

The next examples shows how to build your own distributions. Further examples show the usage of the distributions and some statistical tests.

1. Making a Continuous Distribution, i.e., Subclassing $\text{rv}_{\text{continuous}}$

   Making continuous distributions is fairly simple.

```
from scipy import stats
class deterministic_gen(stats.rv_continuous):
    def _cdf(self, x):
        return np.where(x < 0, 0., 1.)
    def _stats(self):
        return 0., 0., 0., 0.
```

```
deterministic = deterministic_gen(name="deterministic")
print(deterministic.cdf(np.arange(-3, 3, 0.5)))
```

```
[0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1.]
```

   Interestingly, the `pdf` is now computed automatically:

```
deterministic.pdf(np.arange(-3, 3, 0.5))
```

```
array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 5.83333333e+04, 4.16333634e-12,
       4.16333634e-12, 4.16333634e-12, 4.16333634e-12, 4.16333634e-12])
```

   Be aware of the performance issues mentions in Performance Issues and Cautionary Remarks. The computation of unspecified common methods can become very slow, since only general methods are called which, by their very nature, cannot use any specific information about the distribution. Thus, as a cautionary example:

```
from scipy.integrate import quad
print(quad(deterministic.pdf, -1e-1, 1e-1))
```

```
(4.163336342344337e-13, 0.0)
```

But this is not correct: the integral over this `pdf` should be 1. Let's make the integration interval smaller:

```
quad(deterministic.pdf, -1e-3, 1e-3)  # warning removed
```

```
/Users/subway/.virtualenvs/py3env/lib/python3.6/site-packages/scipy/integrate/qu
  If increasing the limit yields no improvement it is advised to analyze
  the integrand in order to determine the difficulties.  If the position of a
  local difficulty can be determined (singularity, discontinuity) one will
  probably gain from splitting up the interval and calling the integrator
  on the subranges.  Perhaps a special-purpose integrator should be used.
  warnings.warn(msg, IntegrationWarning)
(1.000076872229173, 0.0010625571718182458)
```

This looks better. However, the problem originated from the fact that the `pdf` is not specified in the class definition of the deterministic distribution.

2. **TODO** Subclassing $\text{rv}_{\text{discrete}}$

   In the following we use `stats.rv_discrete` to generate a discrete distribution that has the probabilities of the truncated normal for the intervals centered around the integers.

   (a) General Info

   From the docstring of `rv_discrete`, `help(stats.rv_discrete)`, "You can construct an arbitrary discrete `rv` where `P{X=x_k} = pk` by passing to the `rv_discrete` initialization method (through the values= keyword) a tuple of sequences `(xk, pk)` which describes

only those values of `X (xk)` that occur with nonzero probability `(pk)`."

Next to this, there are some further requirements for this approach to work:

- The keyword name is required.
- The support points of the distribution xk have to be integers.
- The number of significant digits (decimals) needs to be specified.

In fact, if the last two requirements are not satisfied an exception may be raised or the resulting numbers may be incorrect.

(b) An Example

Let's do the work. First

```
npoints = 20   # number of integer support points of the distribution minus 1
npointsh = npoints // 2
npointsf = float(npoints)
nbound = 4   # bounds for the truncated normal
normbound = (1+1/npointsf) * nbound   # actual bounds of truncated normal
grid = np.arange(-npointsh, npointsh+2, 1)   # integer grid
gridlimitsnorm = (grid-0.5) / npointsh * nbound   # bin limits for the truncnorm
gridlimits = grid - 0.5   # used later in the analysis
grid = grid[:-1]
probs = np.diff(stats.truncnorm.cdf(gridlimitsnorm, -normbound, normbound))
gridint = grid
```

And finally we can subclass `rv_discrete`:

```
normdiscrete = stats.rv_discrete(values=(gridint,
            np.round(probs, decimals=7)), name='normdiscrete')
```

Now that we have defined the distribution, we have access to all common methods of discrete distributions.

```
print('mean = %6.4f, variance = %6.4f, skew = %6.4f, kurtosis = %6.4f' %
      normdiscrete.stats(moments='mvsk'))
```

```
mean = -0.0000, variance = 6.3302, skew = 0.0000, kurtosis = -0.0076
```

```
nd_std = np.sqrt(normdiscrete.stats(moments='v'))
```

(c) Testing the Implementation

Let's generate a random sample and compare observed frequencies with the probabilities.

```
n_sample = 500
np.random.seed(87655678)   # fix the seed for replicability
rvs = normdiscrete.rvs(size=n_sample)
f, l = np.histogram(rvs, bins=gridlimits)
sfreq = np.vstack([gridint, f, probs*n_sample]).T
print(sfreq)
```

```
[[-1.00000000e+01  0.00000000e+00  2.95019349e-02]
 [-9.00000000e+00  0.00000000e+00  1.32294142e-01]
 [-8.00000000e+00  0.00000000e+00  5.06497902e-01]
 [-7.00000000e+00  2.00000000e+00  1.65568919e+00]
 [-6.00000000e+00  1.00000000e+00  4.62125309e+00]
 [-5.00000000e+00  9.00000000e+00  1.10137298e+01]
 [-4.00000000e+00  2.60000000e+01  2.24137683e+01]
 [-3.00000000e+00  3.70000000e+01  3.89503370e+01]
 [-2.00000000e+00  5.10000000e+01  5.78004747e+01]
 [-1.00000000e+00  7.10000000e+01  7.32455414e+01]
 [ 0.00000000e+00  7.40000000e+01  7.92618251e+01]
 [ 1.00000000e+00  8.90000000e+01  7.32455414e+01]
 [ 2.00000000e+00  5.50000000e+01  5.78004747e+01]
 [ 3.00000000e+00  5.00000000e+01  3.89503370e+01]
 [ 4.00000000e+00  1.70000000e+01  2.24137683e+01]
 [ 5.00000000e+00  1.10000000e+01  1.10137298e+01]
 [ 6.00000000e+00  4.00000000e+00  4.62125309e+00]
 [ 7.00000000e+00  3.00000000e+00  1.65568919e+00]
 [ 8.00000000e+00  0.00000000e+00  5.06497902e-01]
 [ 9.00000000e+00  0.00000000e+00  1.32294142e-01]
 [ 1.00000000e+01  0.00000000e+00  2.95019349e-02]]
```

Next, we can test, whether our sample was generated by our

normdiscrete distribution. This also verifies whether the random numbers are generated correctly.

The chi-square test requires that there are a minimum number of observations in each bin. We combine the tail bins into larger bins so that they contain enough observations.

```
f2 = np.hstack([f[:5].sum(), f[5:-5], f[-5:].sum()])
p2 = np.hstack([probs[:5].sum(), probs[5:-5], probs[-5:].sum()])
ch2, pval = stats.chisquare(f2, p2*n_sample)
```

```
print('chisquare for normdiscrete: chi2 = %6.3f pvalue = %6.4f' % (ch2, pval))
```

```
chisquare for normdiscrete: chi2 = 12.466 pvalue = 0.4090
```

The pvalue in this case is high, so we can be quite confident that our random sample was actually generated by the distribution.

2.7.3 TODO Analysing One Sample

2.7.4 TODO Comparing two samplesű

2.7.5 TODO Kernel Density Estimation

2.8 TODO Guide: Gaussian mixture models

`sklearn.mixture` is a package which enables one to learn Gaussian Mixture Models (diagonal, spherical, tied and full covariance matrices supported), sample them, and estimate them from data. Facilities(设施) to help determine the appropriate number of components are also provided.

A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. **One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians**.

Scikit-learn implements different classes to estimate Gaussian mixture models, that correspond to different estimation strategies, detailed below.

2.8.1 TODO Gaussian Mixture

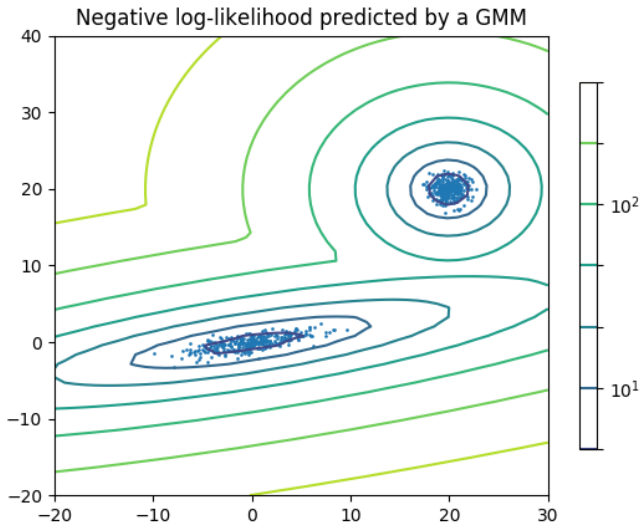
The `GaussianMixture` object implements the expectation-maximization (EM) algorithm for fitting mixture-of-Gaussian models. It can also draw confidence ellipsoids for multivariate models, and compute the Bayesian Information Criterion to assess the number of clusters in the data. A `GaussianMixture.fit` method is provided that learns a Gaussian Mixture Model from train data. Given test data, it can assign to each sample the Gaussian it mostly probably belong to using the `GaussianMixture.predict` method.

The `GaussianMixture` comes with different options to constrain the covariance of the difference classes estimated: spherical, diagonal, tied or full covariance.

1. Example: Density Estimation for a Gaussian mixture

   Plot the density estimation of a mixture of two Gaussians. Data is generated from two Gaussians with different centers and covariance matrices.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
from sklearn import mixture

n_samples = 300

# generate random sample, two components
np.random.seed(0)

# generate spherical data centered on (20, 20)
shifted_gaussian = np.random.randn(n_samples, 2) + np.array([20, 20])

# generate zero centered stretched Gaussian data
C = np.array([[0., -0.7], [3.5, .7]])
stretched_gaussian = np.dot(np.random.randn(n_samples, 2), C)

# concatenate the two datasets into the final training set
X_train = np.vstack([shifted_gaussian, stretched_gaussian])

# fit a Gaussian Mixture Model with two components
clf = mixture.GaussianMixture(n_components=2, covariance_type='full')
clf.fit(X_train)

# display predicted scores by the model as a contour plot
x = np.linspace(-20., 30.)
y = np.linspace(-20., 40.)
X, Y = np.meshgrid(x, y)
XX = np.array([X.ravel(), Y.ravel()]).T
Z = -clf.score_samples(XX)
Z = Z.reshape(X.shape)

CS = plt.contour(X, Y, Z, norm=LogNorm(vmin=1.0, vmax=1000.0),
                 levels=np.logspace(0, 3, 10))
CB = plt.colorbar(CS, shrink=0.8, extend='both')
plt.scatter(X_train[:, 0], X_train[:, 1], .8)

plt.title('Negative log-likelihood predicted by a GMM')
plt.axis('tight')
plt.savefig("img/2.sk.Density-Estimation-for-a-Gaussian-mixture.png")
plt.close("all")
```



2. **TODO** Example: GMM covariances

   Demonstration of several covariances types for Gaussian mixture models.

   See Gaussian mixture models for more information on the estimator.

   Although GMM are often used for clustering, we can compare the obtained clusters with the actual classes from the dataset. We initialize the means of the Gaussians with the means of the classes from the training set to make this comparison valid.

   We plot predicted labels on both training and held out test data using a variety of GMM covariance types on the iris dataset. We compare GMMs with spherical, diagonal, full, and tied covariance matrices in increasing order of performance. Although one would expect full covariance to perform best in general, it is prone to overfitting on small datasets and does not generalize well to held out test data.

   On the plots, train data is shown as dots, while test data is shown as crosses. The iris dataset is four-dimensional. Only the first two

dimensions are shown here, and thus some points are separated in other dimensions.

3. Pros and cons of class GaussianMixture

表 2.1: Pros

Speed	It is the fastest algorithm for learning mixture models
Agnostic	As this algorithm maximizes only the likelihood, it will not bias the means towards zero, or bias the cluster sizes to have specific structures that might or might not apply.

表 2.2: Cons

Singularities	When one has insufficiently many points per mixture, estimating the covariance matrices becomes difficult, and the algorithm is known to diverge and find solutions with infinite likelihood unless one regularizes the covariances artificially.
Number of components	This algorithm will always use all the components it has access to, needing held-out data or information theoretical criteria to decide how many components to use in the absence of external cues.

4. **TODO** Selecting the number of components in a classical Gaussian Mixture Model

2.8.2 TODO Variational Bayesian Gaussian Mixture

2.9 Example: Decorator

```
import time

def timer(parameter):

    def outer_wrapper(func):

        def wrapper(*args, **kwargs):
            if parameter == 'task1':
                start = time.time()
                func(*args, **kwargs)
                stop = time.time()
                print("the task1 run time is :", stop - start)
            elif parameter == 'task2':
                start = time.time()
                func(*args, **kwargs)
                stop = time.time()
                print("the task2 run time is :", stop - start)

        return wrapper

    return outer_wrapper

@timer(parameter='task1')
def task1():
    time.sleep(2)
    print("in the task1")

@timer(parameter='task2')
def task2():
    time.sleep(2)
    print("in the task2")

task1()
task2()
```

```
in the task1
the task1 run time is : 2.002086877822876
in the task2
the task2 run time is : 2.0035531520843506
```

3 Linear Models for Regression

The focus so far in this book has been on unsupervised learning, including topics such as density estimation and data clustering. We turn now to a discussion of supervised learning, starting with regression. The goal of regression is to predict the value of one or more continuous target variables t given the value of a D-dimensional vector $\mathbf{x}$ of input variables. We have already encountered an example of a regression problem when we considered polynomial curve fitting in Chapter 1. The polynomial is a specific example of a broad class of functions called linear regression models, which share the property of being linear functions of the adjustable parameters, and which will form the focus of this chapter. The simplest form of linear regression models are also linear functions of the input variables. However, we can obtain a much more useful class of functions by taking linear combinations of a fixed set of nonlinear functions of the input variables, known as *basis functions(基函数)*. Such models are linear functions of the parameters, which gives them simple analytical properties, and yet can be nonlinear with respect to the input variables.

Given a training data set comprising N observations $\{\mathbf{x}_n\}$, where $n = 1, \cdots, N$, together with corresponding target values $\{t_n\}$, the goal is to predict the value of t for a new value of $\mathbf{x}$. In the simplest approach, this can be done by directly constructing an appropriate function $y(\mathbf{x})$ whose values for new inputs $\mathbf{x}$ constitute the predictions for the corresponding values of t. More generally, from a probabilistic perspective, we aim to model the predictive distribution $p(t|\mathbf{x})$ because this expresses our uncertainty about the value of t for each value of $\mathbf{x}$. From this conditional distribution we can make predictions of t, for any new value of $\mathbf{x}$, in such a way as to minimize the expected value of a suitably chosen loss function. As discussed in Section 1.5.5, a common choice of loss function for real-valued variables is the squared loss, for which the optimal solution is given by the conditional expectation of t.

Although linear models have significant limitations as practical techniques for pattern recognition, particularly for problems involving input spaces of high dimensionality, they have nice analytical properties and form the foundation for more sophisticated models to be discussed in later chapters.

3.1 Linear Basis Function Models

The simplest linear model for regression is one that involves a linear combination of the input variables

$$y(x, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_D x_D \tag{3.1}$$

where $\mathbf{x} = (x_1, \cdots, x_D)^T$. This is often simply known as *linear regression(线性回归)*. The key property of this model is that it is a linear function of the parameters $w_0, \cdots, w_D$. It is also, however, a linear function of the input variables x_i, and this imposes significant limitations on the model. We therefore extend the class of models by considering linear combinations of fixed nonlinear functions of the input variables, of the form

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) \tag{3.2}$$

where $\phi_j(x)$ are known as basis functions. By denoting the maximum value of the index j by $M-1$, the total number of parameters in this model will be M .

The parameter w_0 allows for any fixed offset in the data and is sometimes called a *bias parameter(偏置参数)* (not to be confused with 'bias'in a statistical sense). It is often convenient to define an additional dummy 'basis function'$\phi_0(\mathbf{x}) = 1$ so that

$$y(\mathbf{x}, w) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) \tag{3.3}$$

where $\mathbf{w} = (w_0, \cdots, w_{M-1})^T$ and $\boldsymbol{\phi} = (\phi_0, \cdots, \phi_{M-1})^T$. In many practical applications of pattern recognition, we will apply some form of fixed pre-processing, or feature extraction, to the original data variables. If the original variables comprise the vector $\mathbf{x}$, then the features can be expressed in terms of the basis functions $\{\phi_j(\mathbf{x})\}$.

By using nonlinear basis functions, we allow the function $y(\mathbf{x}, \mathbf{w})$ to be a nonlinear function of the input vector $\mathbf{x}$. Functions of the form (3.2) are called linear models, however, because this function is linear in $\mathbf{w}$. It is this linearity in the parameters that will greatly simplify the analysis of this class of models. However, it also leads to some significant limitations, as we discuss in Section 3.6.

The example of polynomial regression considered in Chapter 1 is a particular example of this model in which there is a single input variable x, and the basis functions take the form of powers of x so that $\phi_j(x) = x^j$. One limitation of polynomial basis functions is that they are global functions of the input variable, so that changes in one region of input space affect all other regions. This can be resolved by dividing the input space up into regions and fit a different polynomial in each region, leading to *spline functions(样条函数)* (Hastie et al., 2001).

There are many other possible choices for the basis functions, for example

$$\phi_j(x) = \exp -\frac{(x - \mu_j)^2}{2s^2} \tag{3.4}$$

where the μ_j govern the locations of the basis functions in input space, and the parameter s governs their spatial scale. These are usually referred to as 'Gaussian'basis functions, although it should be noted that they are not required to have a probabilistic interpretation, and in particular the normalization coefficient is unimportant because these basis functions will be multiplied by adaptive parameters w_j.

Another possibility is the sigmoidal basis function of the form

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right) \tag{3.5}$$

where $\sigma(a)$ is the logistic sigmoid function defined by

$$\sigma(a) = \frac{1}{1 + \exp(-a)}. \tag{3.6}$$

Equivalently, we can use the "tanh" function because this is related to the logistic sigmoid by $tanh(a) = 2\sigma(a) - 1$, and so a general linear combination of logistic sigmoid functions is equivalent to a general linear combination of "tanh" functions. These various choices of basis function are illustrated in Figure 3.1.

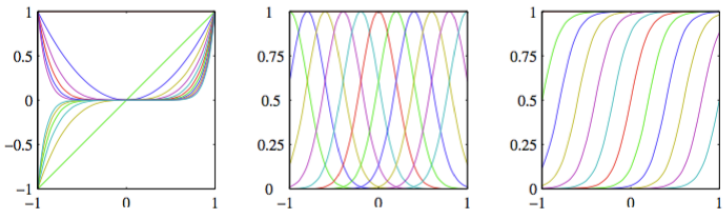


图 3.1: Examples of basis functions, showing polynomials on the left, Gaussians of the form (3.4) in the centre, and sigmoidal of the form (3.5) on the right.

Yet another possible choice of basis function is the Fourier basis, which leads to an expansion in sinusoidal functions. Each basis function represents a specific frequency and has infinite spatial extent. By contrast, basis functions that are localized to finite regions of input space necessarily comprise a spectrum of different spatial frequencies. In many signal processing applications, it is of interest to consider basis functions that are localized in both space and frequency, leading to a class of functions known as wavelets(小波). These are also defined to be mutually orthogonal, to simplify their application. Wavelets are most applicable when the input values live on a regular lattice, such as the successive time points in a temporal sequence,

or the pixels in an image. Useful texts on wavelets include Ogden (1997), Mallat (1999), and Vidakovic (1999).

Most of the discussion in this chapter, however, is independent of the particular choice of basis function set, and so for most of our discussion we shall not specify the particular form of the basis functions, except for the purposes of numerical illustration. Indeed, much of our discussion will be equally applicable to the situation in which the vector $\boldsymbol{\phi}(x)$ of basis functions is simply the identity $\boldsymbol{\phi}(x) = \mathbf{x}$. Furthermore, in order to keep the notation simple, we shall focus on the case of a single target variable t. However, in Section 3.1.5, we consider briefly the modifications needed to deal with multiple target variables.

3.1.1 Maximum likelihood and least squares

In Chapter 1, we fitted polynomial functions to data sets by minimizing a sum-of-squares error function. We also showed that this error function could be motivated as the maximum likelihood solution under an assumed Gaussian noise model. Let us return to this discussion and consider the least squares approach, and its relation to maximum likelihood, in more detail.

As before, we assume that the target variable t is given by a deterministic function $y(\mathbf{x}, \mathbf{w})$ with additive Gaussian noise so that

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \tag{3.7}$$

where ϵ is a zero mean Gaussian random variable with precision (inverse variance) β. Thus we can write

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}). \tag{3.8}$$

Recall that, if we assume a squared loss function, then the optimal prediction, for a new value of $\mathbf{x}$, will be given by the conditional mean of the target variable. In the case of a Gaussian conditional distribution of the form (3.8), the conditional mean will be simply

$$\mathbb{E}[t|\mathbf{x}] = \int tp(t|\mathbf{x})\,dx = y(\mathbf{x}, \mathbf{w}) \tag{3.9}$$

Note that the Gaussian noise assumption implies that the conditional distribution of t given $\mathbf{x}$ is unimodal, which may be inappropriate for some applications. An extension to mixtures of conditional Gaussian distributions, which permit multimodal conditional distributions, will be discussed in Section 14.5.1.

Now consider a data set of inputs $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ with corresponding target values $t_1, \cdots, t_N$. We group the target variables $\{t_n\}$ into a column vector that we denote by $\mathbf{t}$ where the typeface is chosen to distinguish it from a single observation of a multivariate target, which would be denoted $\mathbf{t}$. Making the assumption that these data points are drawn independently from the distribution (3.8), we obtain the following expression for the likelihood function, which is a function of the adjustable parameters $\mathbf{w}$ and β, in the form

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}) \tag{3.10}$$

where we have used (3.2). Note that in supervised learning problems such as regression (and classification), we are not seeking to model the distribution of the input variables. Thus $\mathbf{x}$ will always appear in the set of conditioning variables, and so from now on we will drop the explicit x from expressions such as $p(t|\mathbf{x}, \mathbf{w}, \beta)$ in order to keep the notation uncluttered. Taking the logarithm of the likelihood function, and making use of the standard form (1.46) for the univariate Gaussian, we have

$$\begin{aligned} \ln p(\mathbf{t}|\mathbf{w}, \beta) &= \sum_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}) \\ &= \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta E_D(\mathbf{w}) \end{aligned} \tag{3.11}$$

where the sum-of-squares error function is defined by

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\boldsymbol{\Phi}(\mathbf{x}_n)\}^2. \tag{3.12}$$

Having written down the likelihood function, we can use maximum likelihood to determine $\mathbf{w}$ and β. Consider first the maximization with respect to $\mathbf{w}$. As observed already in Section 1.2.5, we see that maximization of the likelihood function under a conditional Gaussian noise distribution for a linear model is equivalent to minimizing a sum-of-squares error function given by $E_D(\mathbf{w})$. The gradient of the log likelihood function (3.11) takes the form

$$\nabla \ln p(\boldsymbol{t}|\mathbf{w}, \beta) = \sum_{n=1}^{N}\{t_n - \mathbf{w}^T\boldsymbol{\phi}(x_n)\}\boldsymbol{\phi}(x_n)^T. \tag{3.13}$$

Setting this gradient to zero gives

$$0 = \sum_{n=1}^{N} t_n\boldsymbol{\phi}(\mathbf{x}_n)^T - \mathbf{w}^T\left(\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^T\right). \tag{3.14}$$

Solving for $\mathbf{w}$ we obtain

$$\mathbf{w}_{ML} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t} \tag{3.15}$$

which are known as the *normal equations(规范方程)* for the least squares problem. Here $\boldsymbol{\Phi}$ is an $N \times M$ matrix, called the *design matrix(设计矩阵)*, whose elements are given by $\boldsymbol{\Phi}_{nj} = \phi_j(\mathbf{x}_n)$, so that

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix} \tag{3.16}$$

The quantity

$$\boldsymbol{\Phi}^\dagger \equiv (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T \tag{3.17}$$

is known as the *Moore-Penrose pseudo-inverse(Moore-Penrose 伪逆矩阵)* of the matrix $\mathbf{\Phi}$ (Rao and Mitra, 1971; Golub and Van Loan, 1996). It can be regarded as a generalization of the notion of matrix inverse to nonsquare matrices. Indeed, if $\mathbf{\Phi}$ is square and invertible, then using the property $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$ we see that $\mathbf{\Phi}^{+} \equiv \mathbf{\Phi}^{-1}$.

At this point, we can gain some insight into the role of the bias parameter w_0. If we make the bias parameter explicit, then the error function (3.12) becomes

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - _0 - \sum_{j=1}^{M-1} w_j\phi_j(\mathbf{x}_n)\}^2. \tag{3.18}$$

Setting the derivative with respect to w_0 equal to zero, and solving for w_0, we obtain

$$w_0 = \bar{t} - \sum_{j=1}^{M-1} w_j\bar{\phi_j} \tag{3.19}$$

where we have defined

$$\bar{t} = \frac{1}{N}\sum_{n=1}^{N} t_n, \quad \bar{\phi_j} = \frac{1}{N}\sum_{n=1}^{N}\phi_j(\mathbf{x}_n). \tag{3.20}$$

Thus the bias w_0 compensates for the difference between the averages (over the training set) of the target values and the weighted sum of the averages of the basis function values.

We can also maximize the log likelihood function (3.11) with respect to the noise precision parameter β, giving

$$\frac{1}{\beta} = \frac{1}{N}\sum_{n=1}^{N}\{t_n - \mathbf{w}_{ML}^T\boldsymbol{\phi}(\mathbf{x}_n)\}^2 \tag{3.21}$$

and so we see that the inverse of the noise precision is given by the *residual variance(残留方差)* of the target values around the regression function.

3.1.2 Geometry of least squares

At this point, it is instructive to consider the geometrical interpretation of the least-squares solution. To do this we consider an N-dimensional space whose axes are given by the t_n, so that $\boldsymbol{t} = (t_1, \cdots, t_N)^T$ is a vector in this space. Each basis function $\phi_j(\mathbf{x}_n)$, evaluated at the N data points, can also be represented as a vector in the same space, denoted by φ_j, as illustrated in Figure 3.2. Note that φ_j corresponds to the j^{th} column of $\boldsymbol{\Phi}$, whereas $\boldsymbol{\phi}(\mathbf{x}_n)$ corresponds to the n^{th} row of $\boldsymbol{\Phi}$. If the number M of basis functions is smaller than the number N of data points, then the M vectors $\phi_j(\mathbf{x}_n)$ will span a linear subspace $\mathcal{S}$ of dimensionality M. We define $\boldsymbol{y}$ to be an N-dimensional vector whose nth element is given by $y(\mathbf{x}_n, \mathbf{w})$, where $n = 1, \cdots, N$. Because $\boldsymbol{y}$ is an arbitrary linear combination of the vectors φ_j , it can live anywhere in the M-dimensional subspace. The sum-of-squares error (3.12) is then equal (up to a factor of 1/2) to the squared Euclidean distance between $\boldsymbol{y}$ and $\boldsymbol{t}$. Thus the least-squares solution for $\mathbf{w}$ corresponds to that choice of $\boldsymbol{y}$ that lies in subspace $\mathcal{S}$ and that is closest to $\boldsymbol{t}$. Intuitively, from Figure 3.2, we anticipate(预期) that this solution corresponds to the orthogonal projection of $\boldsymbol{t}$ onto the subspace $\mathcal{S}$. This is indeed the case, as can easily be verified by noting that the solution for $\boldsymbol{y}$ is given by $\boldsymbol{\Phi}\mathbf{w}_{ML}$, and then confirming that this takes the form of an orthogonal projection.

In practice, a direct solution of the normal equations can lead to numerical difficulties when $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ is close to singular. In particular, when two or more of the basis vectors φ_j are co-linear, or nearly so, the resulting parameter values can have large magnitudes. Such near degeneracies will not be uncommon when dealing with real data sets. The resulting numerical difficulties can be addressed using the technique of *singular value decomposition(奇异值分解)*, or SVD (Press et al., 1992; Bishop and Nabney, 2008). Note that the addition of a regularization term ensures that the matrix is non- singular, even in the presence of degeneracies.

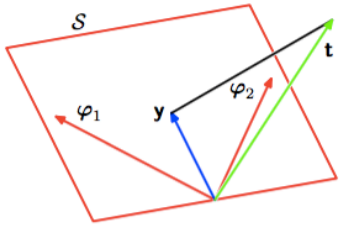


图 3.2: Geometrical interpretation of the least-squares solution, in an N-dimensional space whose axes are the values of $t_1, \cdots, t_N$. The least-squares regression function is obtained by finding the orthogonal projection of the data vector t onto the subspace spanned by the basis functions $\phi_j(x)$ in which each basis function is viewed as a vector ϕ_j of length N with elements $\phi_j(x_n)$.

3.1.3 Sequential learning

Batch techniques, such as the maximum likelihood solution (3.15), which in- volve processing the entire training set in one go, can be computationally costly for large data sets. As we have discussed in Chapter 1, if the data set is sufficiently large, it may be worthwhile to use sequential algorithms, also known as *on-line* algorithms, in which the data points are considered one at a time, and the model parameters up-dated after each such presentation. Sequential learning is also appropriate for real-time applications in which the data observations are arriving in a continuous stream, and predictions must be made before all of the data points are seen.

We can obtain a sequential learning algorithm by applying the technique of *stochastic gradient descent(随机梯度下降)*, also known as *sequential gradient descent(顺序梯度下降)*, as follows. If the error function comprises a sum over data points $E = \sum_n E_n$, then after presentation(展示) of pattern n, the stochastic gradient descent

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \nabla E_n \tag{3.22}$$

where τ denotes the iteration number, and η is a learning rate parameter. We shall discuss the choice of value for η shortly. The value of $\mathbf{w}$ is initialized to some starting vector $\mathbf{w}^{(0)}$. For the case of the sum-of-squares error function (3.12), this gives

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta(t_n - \mathbf{w}^{(\tau)T}\phi_n)\phi_n \tag{3.23}$$

where $\phi_n = \phi(\mathbf{x}_n)$. This is known as *least-mean-squares(最小均方)* or the *LMS algorithm*. The value of η needs to be chosen with care to ensure that the algorithm converges (Bishop and Nabney, 2008).

3.1.4 Regularized least squares

In Section 1.1, we introduced the idea of adding a regularization term to an error function in order to control over-fitting, so that the total error function to be minimized takes the form

$$E_D(\mathbf{w}) + \lambda E_W(\mathbf{w}) \tag{3.24}$$

where λ is the regularization coefficient that controls the relative importance of the data-dependent error $\$\mathrm{E_D(w)}$ and the regularization term $E_W(\mathbf{w})$. One of the simplest forms of regularizer is given by the sum-of-squares of the weight vector elements

$$E_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}. \tag{3.25}$$

If we also consider the sum-of-squares error function given by

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\}^2 \tag{3.26}$$

then the total error function becomes

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}. \tag{3.27}$$

This particular choice of regularizer is known in the machine learning literature as *weight decay* because in sequential learning algorithms, it encourages weight values to decay towards zero, unless supported by the data. In statistics, it provides an example of a *parameter shrinkage(参数收缩)* method because it shrinks parameter values towards zero. It has the advantage that the error function remains a quadratic function of $\mathbf{w}$, and so its exact minimizer can be found in closed form. Specifically, setting the gradient of (3.27) with respect to $\mathbf{w}$ to zero, and solving for $\mathbf{w}$ as before, we obtain

$$\mathbf{w} = (\lambda\mathbf{I} + \mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\boldsymbol{t} \tag{3.28}$$

This represents a simple extension of the least-squares solution (3.15).

A more general regularizer is sometimes used, for which the regularized error takes the form

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\phi(\mathbf{x})\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q \tag{3.29}$$

where $q = 2$ corresponds to the quadratic regularizer (3.27). Figure 3.3 shows contours of the regularization function for different values of q.

The case of $q = 1$ is know as the *lasso(套索)* in the statistics literature (Tibshirani, 1996). It has the property that if λ is sufficiently large, some of the coefficients $\mathbf{w}_j$ are driven to zero, leading to a sparse model in which the corresponding basis functions play no role. To see this, we first note that minimizing (**??**) is equivalent to minimizing the unregularized sum-of-squares error (3.12) subject to the constraint

$$\sum_{j=1}^{M}|w_j|^q \leqslant \eta \tag{3.30}$$

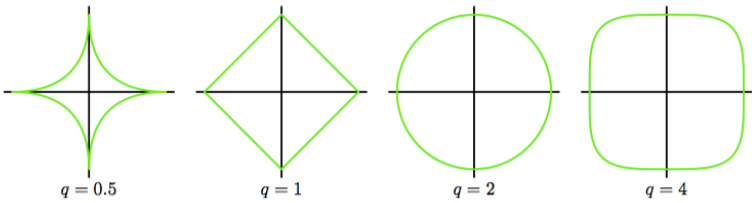


图 3.3: Contours of the regularization term in (3.29) for various values of the parameter q.

for an appropriate value of the parameter η, where the two approaches can be related using Lagrange multipliers. The origin of the sparsity can be seen from Figure 3.4, which shows that the minimum of the error function, subject to the constraint (3.30). As λ is increased, so an increasing number of parameters are driven to zero.

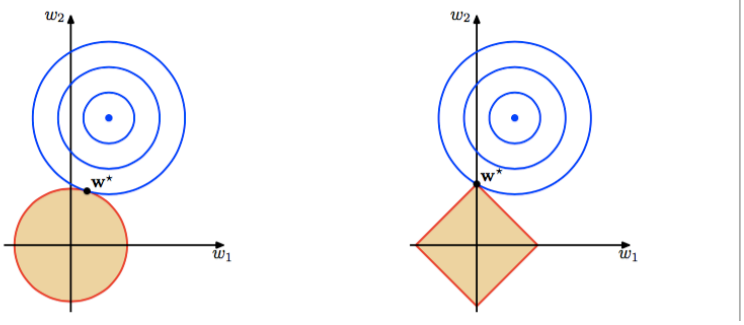


图 3.4: Plot of the contours of the unregularized error function (blue) along with the constraint region (3.30) for the quadratic regularizer $q = 2$ on the left and the lasso regularizer $q = 1$ on the right, in which the optimum value for the parameter vector $\mathbf{w}$ is denoted by $\mathbf{w}^*$. The lasso gives a sparse solution in which $w_1 = 0$.

Regularization allows complex models to be trained on data sets of limited size without severe over-fitting, essentially by limiting the effective

model complexity. However, the problem of determining the optimal model complexity is then shifted from one of finding the appropriate number of basis functions to one of determining a suitable value of the regularization coefficient λ. We shall return to the issue of model complexity later in this chapter.

For the remainder of this chapter we shall focus on the quadratic regularizer (3.27) both for its practical importance and its analytical tractability.

3.1.5 Multiple outputs

So far, we have considered the case of a single target variable t. In some applications, we may wish to predict $K > 1$ target variables, which we denote collectively by the target vector $\boldsymbol{t}$. This could be done by introducing a different set of basis functions for each component of $\boldsymbol{t}$, leading to multiple, independent regression problems. However, a more interesting, and more common, approach is to use the same set of basis functions to model all of the components of the target vector so that

$$\mathbf{y}(\mathbf{x}, \mathbf{w}) = \mathbf{W}^T \phi(\mathbf{x}) \tag{3.31}$$

where $\mathbf{y}$ is a K-dimensional column vector, $\mathbf{W}$ is an M Œ K matrix of parameters, and $\phi(\mathbf{x})$ is an M-dimensional column vector with elements $\phi_j(\mathbf{x})$, with $\phi_0(\mathbf{x}) = 1$ as before. Suppose we take the conditional distribution of the target vector to be an isotropic Gaussian of the form

$$p(\boldsymbol{t}|\mathbf{x}, \mathbf{W}, \beta) = \mathcal{N}(\boldsymbol{t}|\mathbf{W}^T \phi(\mathbf{x}), \beta^{-1}\mathbf{I}). \tag{3.32}$$

If we have a set of observations $\boldsymbol{t}_1, \cdots, \boldsymbol{t}_N$, we can combine these into a matrix $\boldsymbol{T}$ of size $N \times K$ such that the n^{th} row is given by t_n^T . Similarly, we can combine the input vectors $\mathbf{x}_1, \cdots, \mathbf{x}_N$ into a matrix $\mathbf{X}$. The log likelihood function is then given by

$$\begin{aligned}\ln p(\mathbf{T}|\boldsymbol{X}, \mathbf{W}, \beta) &= \sum_{n=1}^{N} \mathcal{N}(\mathbf{t}_n|\mathbf{W}^T\phi(\mathbf{x}_n), \beta^{-1}\mathbf{I}) \\ &= \frac{NK}{2}\ln(\frac{\beta}{2\pi}) - \frac{\beta}{2}\sum_{n=1}^{N}\|\mathbf{t}_n - \mathbf{W}^T\phi(\mathbf{x}_n)\|^2.\end{aligned} \tag{3.33}$$

As before, we can maximize this function with respect to $\mathbf{W}$, giving

$$\mathbf{W}_{ML} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{T}. \tag{3.34}$$

If we examine this result for each target variable t_k, we have

$$\mathbf{w}_k = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}_k = \boldsymbol{\Phi}^+\boldsymbol{t}_k \tag{3.35}$$

where $\boldsymbol{t}_k$ is an N-dimensional column vector with components t_{nk} for $n = 1, \cdots, N$. Thus the solution to the regression problem decouples between the different target variables, and we need only compute a single pseudo-inverse matrix $\boldsymbol{\Phi}^+$, which is shared by all of the vectors $\mathbf{w}_k$.

The extension to general Gaussian noise distributions having arbitrary covariance matrices is straightforward. Again, this leads to a decoupling into K independent regression problems. This result is unsurprising because the parameters $\mathbf{W}$ define only the mean of the Gaussian noise distribution, and we know from Section 2.3.4 that the maximum likelihood solution for the mean of a multivariate Gaussian is independent of the covariance. From now on, we shall therefore consider a single target variable t for simplicity(朴素).

3.2 The Bias-Variance Decomposition

So far in our discussion of linear models for regression, we have assumed that the form and number of basis functions are both fixed. As we have seen in Chapter 1, the use of maximum likelihood, or equivalently least squares, can lead to severe over-fitting if complex models are trained using data sets of limited size. However, limiting the number of basis functions in

order to avoid over-fitting has the side effect of limiting the flexibility of the model to capture interesting and important trends in the data. Although the introduction of regularization terms can control over-fitting for models with many parameters, this raises the question of how to determine a suitable value for the regularization coefficient λ. Seeking the solution that minimizes the regularized error function with respect to both the weight vector $\mathbf{w}$ and the regularization coefficient λ is clearly not the right approach since this leads to the unregularized solution with $\lambda = 0$.

As we have seen in earlier chapters, the phenomenon of over-fitting is really an unfortunate property of maximum likelihood and does not arise when we marginalize over parameters in a Bayesian setting. In this chapter, we shall consider the Bayesian view of model complexity in some depth. Before doing so, however, it is instructive to consider a frequentist viewpoint of the model complexity issue, known as the *bias-variance trade-off(偏置-方差折中)*. Although we shall introduce this concept in the context of linear basis function models, where it is easy to illustrate the ideas using simple examples, the discussion has more general applicability.

In Section 1.5.5, when we discussed decision theory for regression problems, we considered various loss functions each of which leads to a corresponding optimal prediction once we are given the conditional distribution $p(t|\mathbf{x})$. A popular choice is the squared loss function, for which the optimal prediction is given by the conditional expectation, which we denote by $h(\mathbf{x})$ and which is given by

$$h(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}]] = \int tp(t|\mathbf{x})\,\mathrm{dt} \tag{3.36}$$

At this point, it is worth distinguishing between the squared loss function arising from decision theory and the sum-of-squares error function that arose in the maximum likelihood estimation of model parameters. We might use more sophisticated techniques than least squares, for example regularization or a fully Bayesian approach, to determine the conditional distribution

$p(t|\mathbf{x})$. These can all be combined with the squared loss function for the purpose of making predictions.

We showed in Section 1.5.5 that the expected squared loss can be written in the form

$$\mathbb{E}[L] = \int \{y(\mathbf{x}) - h(\mathbf{x})\}^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int \{\mathrm{h}(\mathbf{x}) - \mathrm{t}\}^2 \mathrm{p}(\mathbf{x}, \mathrm{t})\,\mathrm{d}\mathbf{x}\,\mathrm{dt}. \tag{3.37}$$

Recall that the second term, which is independent of $y(\mathbf{x})$, arises from the intrinsic noise on the data and represents the minimum achievable value of the expected loss. The first term depends on our choice for the function $y(\mathbf{x})$, and we will seek a solution for $y(\mathbf{x})$ which makes this term a minimum. Because it is nonnegative, the smallest that we can hope to make this term is zero. If we had an unlimited supply of data (and unlimited computational resources), we could in principle find the regression function $h(\mathbf{x})$ to any desired degree of accuracy, and this would represent the optimal choice for $y(\mathbf{x})$. However, in practice we have a data set $\mathcal{D}$ containing only a finite number N of data points, and consequently we do not know the regression function $h(\mathbf{x})$ exactly.

If we model the $h(\mathbf{x})$ using a parametric function $y(\mathbf{x}, \mathbf{w})$ governed by a parameter vector $\mathbf{w}$, then from a Bayesian perspective the uncertainty in our model is expressed through a posterior distribution over $\mathbf{w}$. A frequentist treatment, however, involves making a point estimate of $\mathbf{w}$ based on the data set $\mathcal{D}$, and tries instead to interpret the uncertainty of this estimate through the following thought experiment. Suppose we had a large number of data sets each of size N and each drawn independently from the distribution $p(t, \mathbf{x})$. For any given data set $\mathcal{D}$, we can run our learning algorithm and obtain a prediction function $y(\mathbf{x}; \mathcal{D})$. Different data sets from the ensemble will give different functions and consequently different values of the squared loss. The performance of a particular learning algorithm is then assessed by taking the average over this ensemble of data sets.

Consider the integrand of the first term in (3.37), which for a particular data set $\mathcal{D}$ takes the form

$$\{y(\mathbf{x};\mathcal{D}) - h(\mathbf{x})\}^2 \tag{3.38}$$

Because this quantity will be dependent on the particular data set $\mathcal{D}$, we take its average over the ensemble of data sets. If we add and subtract the quantity $\mathbb{E}_D[y(\mathbf{x};\mathcal{D})]$ inside the braces, and then expand, we obtain

$$\begin{aligned}
&\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathbb{D}}[y(\mathbf{x};\mathcal{D})] + \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};)] - h(\mathbf{x})\}^2 \\
&\quad = \{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathbb{D}}[y(\mathbf{x};\mathcal{D})]\}^2 + \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};)] - h(\mathbf{x})\}^2 \\
&\qquad + 2\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathbb{D}}[y(\mathbf{x};\mathcal{D})]\}\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};)] - h(\mathbf{x})\}
\end{aligned} \tag{3.39}$$

We now take the expectation of this expression with respect to D and note that the final term will vanish, giving

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D}}[\{y(\mathbf{x};\mathcal{D}) - h(\mathbf{x})\}^2] \\
&\quad = \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]^2\}]}_{\text{variance}}.
\end{aligned} \tag{3.40}$$

We see that the expected squared difference between $y(\mathbf{x};\mathcal{D})$ and the regression function $h(\mathbf{x})$ can be expressed as the sum of two terms. The first term, called the squared *bias(偏置)*, represents the extent to which the average prediction over all data sets differs from the desired regression function. The second term, called the *variance(方差)*, measures the extent to which the solutions for individual data sets vary around their average, and hence this measures the extent to which the function $y(\mathbf{x};\mathcal{D})$ is sensitive to the particular choice of data set. We shall provide some intuition to support these definitions shortly when we consider a simple example.

So far, we have considered a single input value $\mathbf{x}$. If we substitute this expansion back into (3.37), we obtain the following decomposition of the expected squared loss

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise} \tag{3.41}$$

where

$$(\text{bias})^2 = \int \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x} \tag{3.42}$$

$$\text{variace} = \int \mathbb{E}_{\mathcal{D}}\left[y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2\right] p(\mathbf{x})\,\mathrm{d}\mathbf{x} \tag{3.43}$$

$$\text{noise} = \int \{h(\mathbf{x}-t)\}^2 p(\mathbf{x},t)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t \tag{3.44}$$

and the bias and variance terms now refer to integrated quantities.

Our goal is to minimize the expected loss, which we have decomposed into the sum of a (squared) bias, a variance, and a constant noise term. As we shall see, there is a trade-off between bias and variance, with very flexible models having low bias and high variance, and relatively rigid models having high bias and low variance. The model with the optimal predictive capability is the one that leads to the best balance between bias and variance. This is illustrated by considering the sinusoidal data set from Chapter 1. Here we generate 100 data sets, each containing N = 25 data points, independently from the sinusoidal curve h(x) = sin(2x). The data sets are indexed by l = 1,. . . ,L, where L = 100, and for each data set D(l) we fit a model with 24 Gaussian basis functions by minimizing the regularized error function (3.27) to give a prediction function y(l)(x) as shown in Figure 3.5. The top row corresponds to a large value of the regularization coefficient that gives low variance (because the red curves in the left plot look similar) but high bias (because the two curves in the right plot are very different). Conversely on the bottom row, for which is small, there is large variance (shown by the high variability between the red curves in the left plot) but low bias (shown by the good fit between the average model fit and the original sinusoidal function). Note that the result of averaging many solutions for the complex model with M = 25 is a very good fit to the regression function, which suggests that

averaging may be a beneficial procedure. Indeed, a weighted averaging of multiple solutions lies at the heart of a Bayesian approach, although the averaging is with respect to the posterior distribution of parameters, not with respect to multiple data sets.