Notes of PRML and Scikit-learn

Subway Chan

2018 年 6 月 21 日

目录

1 Introduction

```python
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

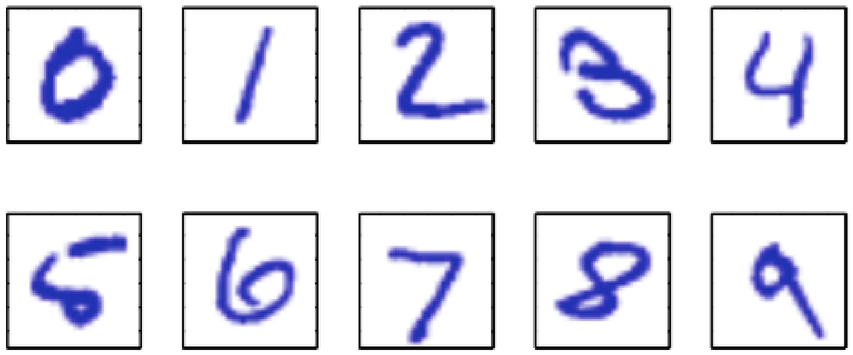


图 1.1: Examples of hand-written digits taken from US zip codes.

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.

```python
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

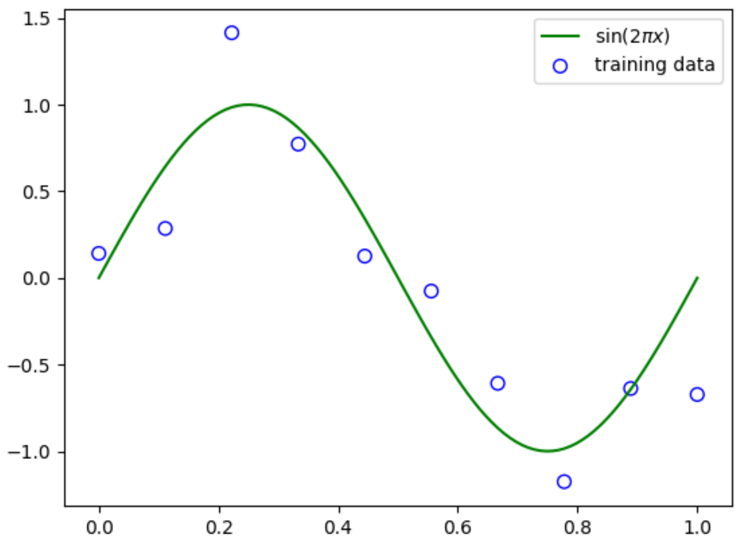


图 1.2: Plot of a training data set of $N = 10$ points, shown as blue circles, each comprising an observation of the input variable x along with the corresponding target variable t. The green curve shows the function $sin(2x)$ used to generate the data. Our goal is to predict the value of t for some new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

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

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

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

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

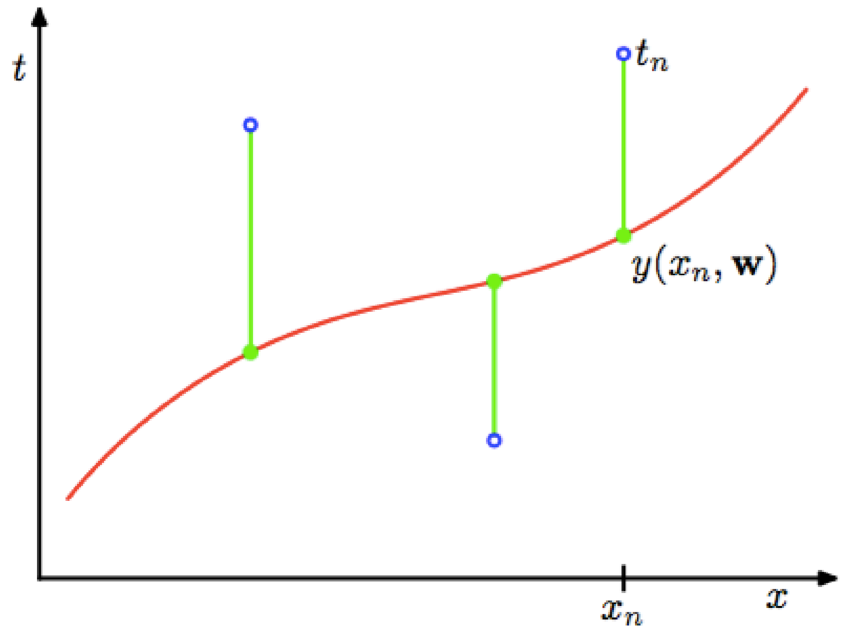


图 1.3: The error function (**??**) corresponds to (Mone half of) the sum of the squares of the displacements (shown by the vertical green bars) of each data point from the function $y(x, \mathbf{w})$.

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.

```python
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

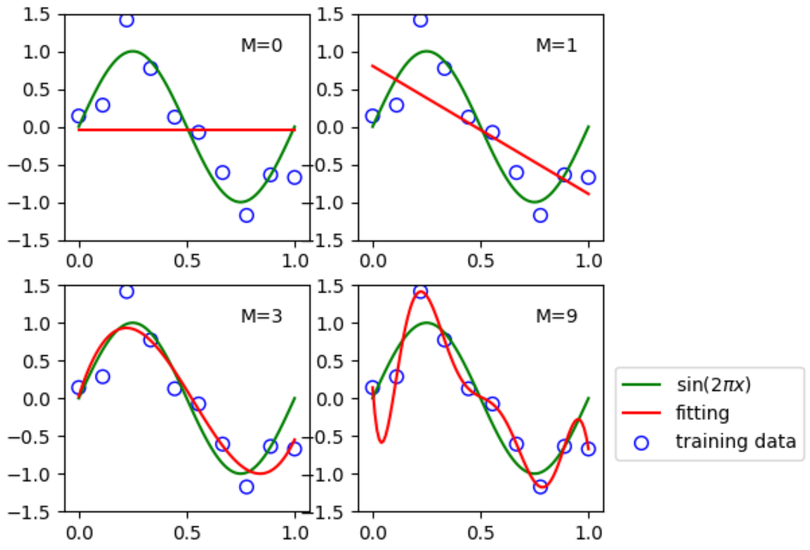


图 1.4: Plots of polynomials having various orders M, shown as red curves, fitted to the data set shown in Figure 1.2.

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

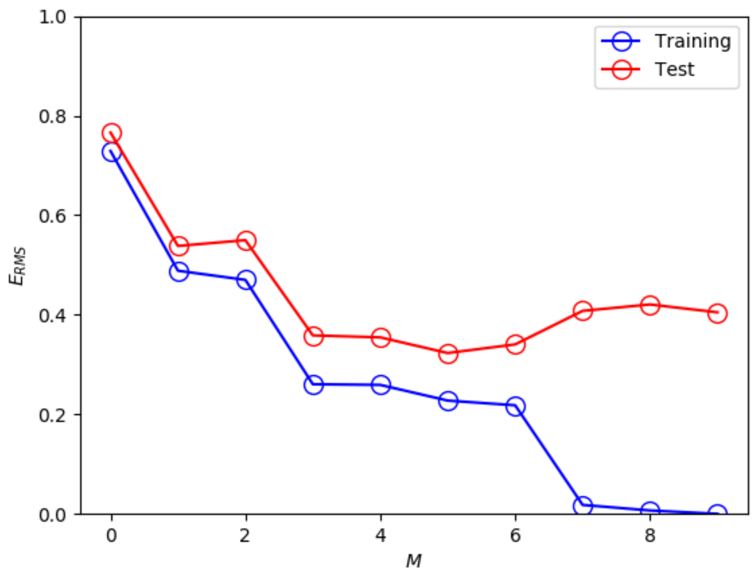


图 1.5: Graphs of the root-mean-square error, defined by (1.3), evaluated on the training set and on an independent test set for various values of M.

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

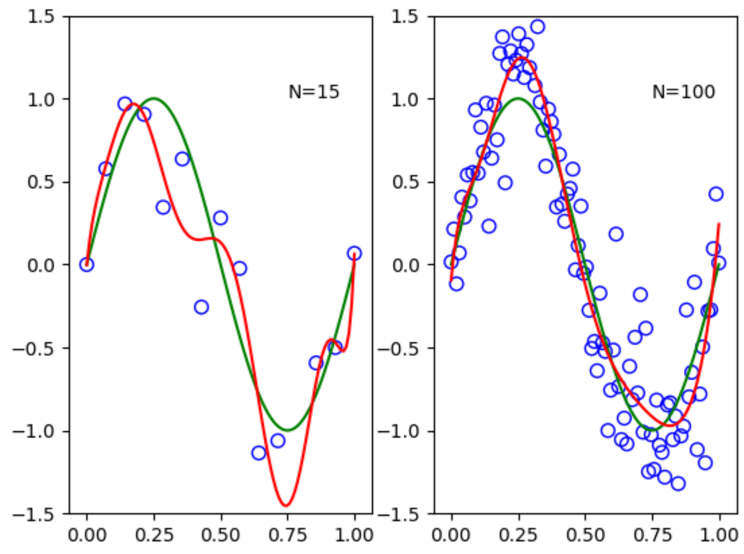


图 1.6: Plots of the solutions obtained by minimizing the sum-of-squares error function using the $M = 9$ polynomial for $N = 15$ data points (left plot) and $N = 100$ data points (right plot). We see that increasing the size of the data set reduces the over-fitting problem.

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

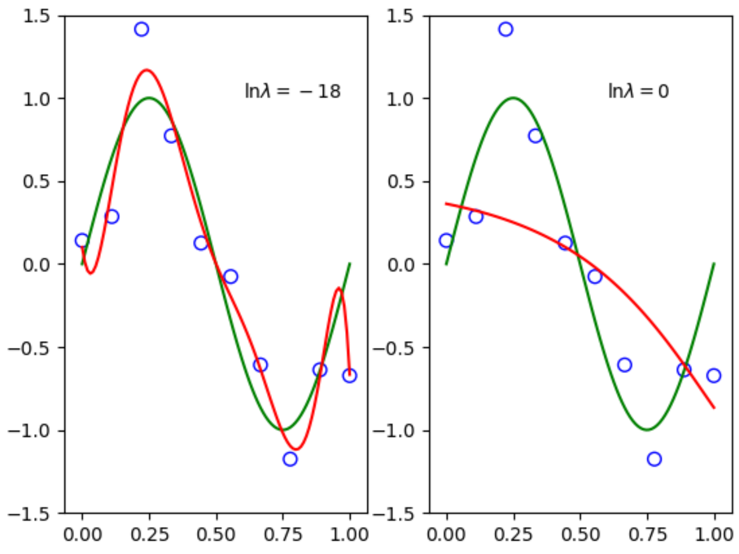


图 1.7: Plots of $M = 9$ polynomials fitted to the data set shown in Figure 1.2 using the regularized error function (**??**) for two values of the regularization parameter λ corresponding to $\ln\lambda = -18$ and $\ln\lambda = 0$. The case of no regularizer, i.e., $\lambda = 0$, corresponding to $\ln\lambda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

```python
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,

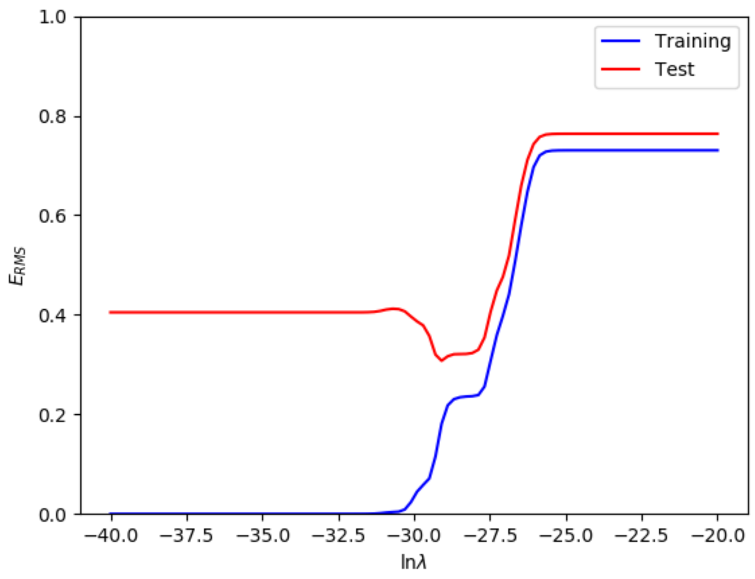


图 1.8: Graph of the root-mean-square error (1.3) versus $\ln\lambda$ for the $M = 9$ polynomial.

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

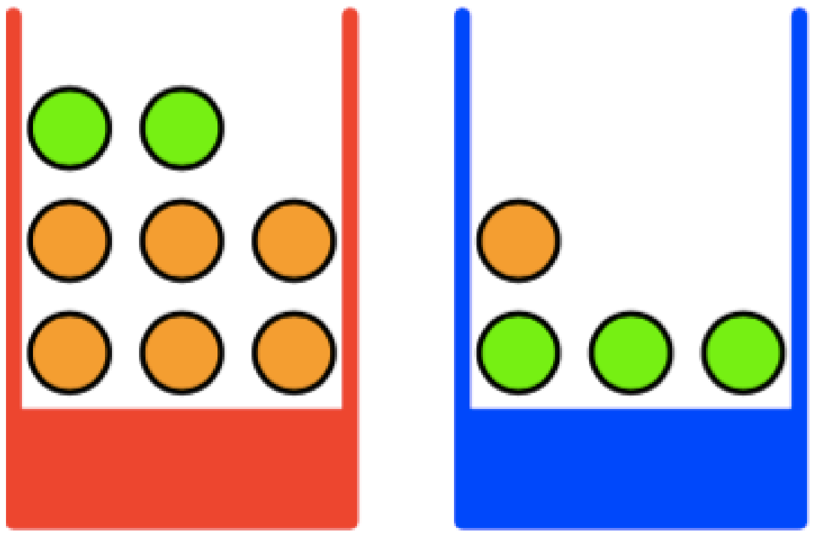


图 1.9: We use a simple example of two coloured boxes each containing fruit (apples shown in green and oranges shown in orange) to introduce the basic ideas of probability.

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

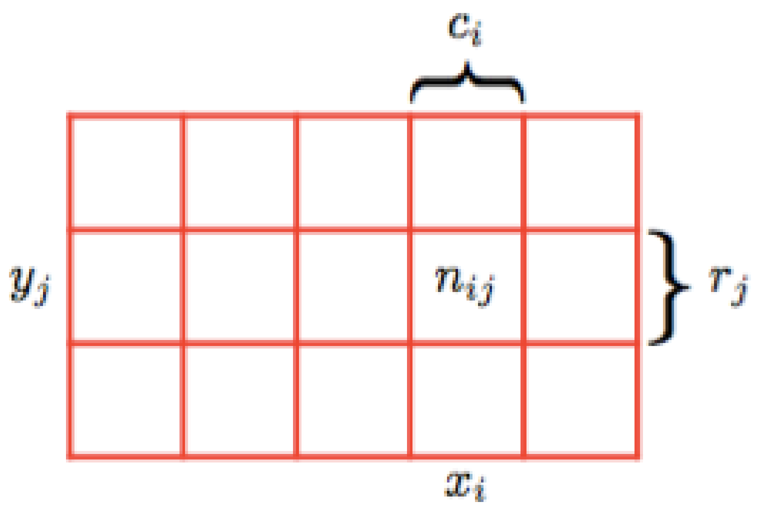


图 1.10: We can derive the sum and product rules of probability by considering two random variables, X, which takes the values xi where $i = 1, \cdots, M$, and Y, which takes the values yj where $j = 1, \cdots, L$. In this illustration we have M = 5 and L = 3. If we consider a total number N of instances 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 $p(X|Y = 1)$.

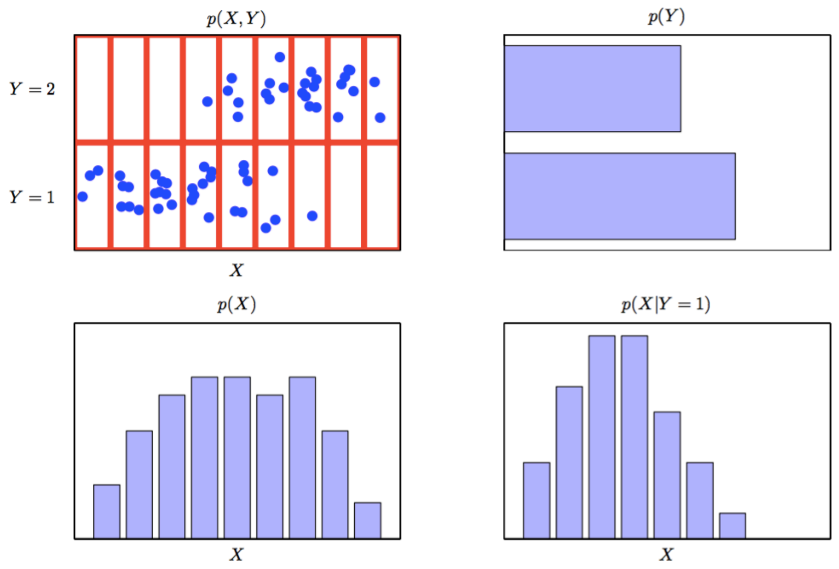


图 1.11: An illustration of a distribution over two variables, X, which takes 9 possible values, and Y , which takes two possible values. The top left figure shows a sample of 60 points drawn from a joint probability distribution over these variables. The remaining figures show histogram estimates of 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.

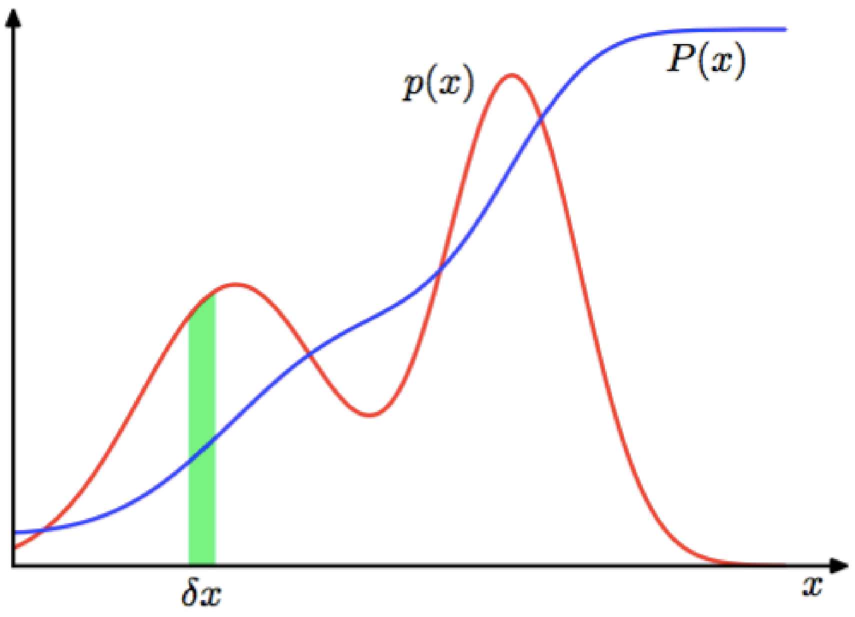


图 1.12: The concept of probability for discrete variables can be extended to that of a probability density $p(x)$ over a continuous variable x and is such that the probability of x lying in the interval $(x, x + \delta x)$ is given by $p(x)\delta x$ for $\delta x \rightarrow 0$. The probability density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

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

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

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

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

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

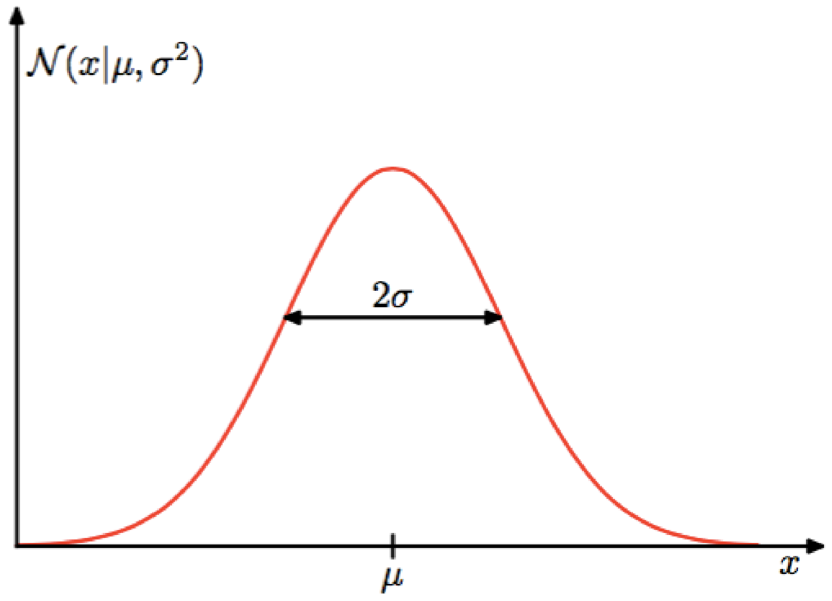


图 1.13: Plot of the univariate Gaussian showing the mean μ and the standard deviation σ.

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

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

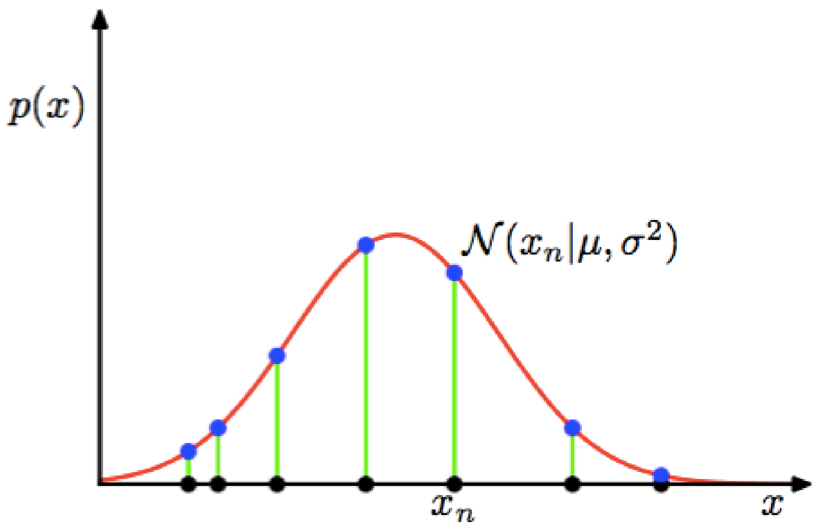


图 1.14: Illustration of the likelihood function for a Gaussian distribution, shown by the red curve. Here the black points denote a data set of values $\{x_n\}$, and the likelihood function given by (1.43) corresponds to the product of the blue values. Maximizing the likelihood involves adjusting the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

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

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

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N} x_n \qquad (1.45)$$

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

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

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

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

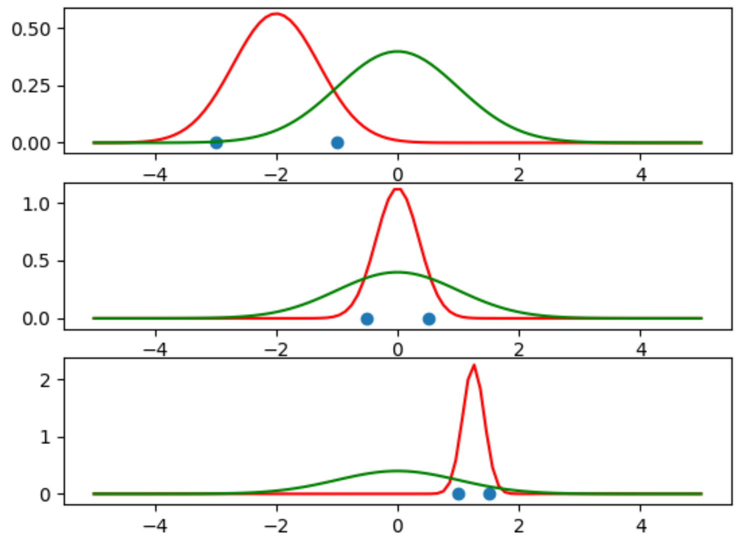
$$\mathbb{E}[\sigma^2_{ML}] = \left(\frac{N-1}{N}\right)\sigma^2 \tag{1.48}$$

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 **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

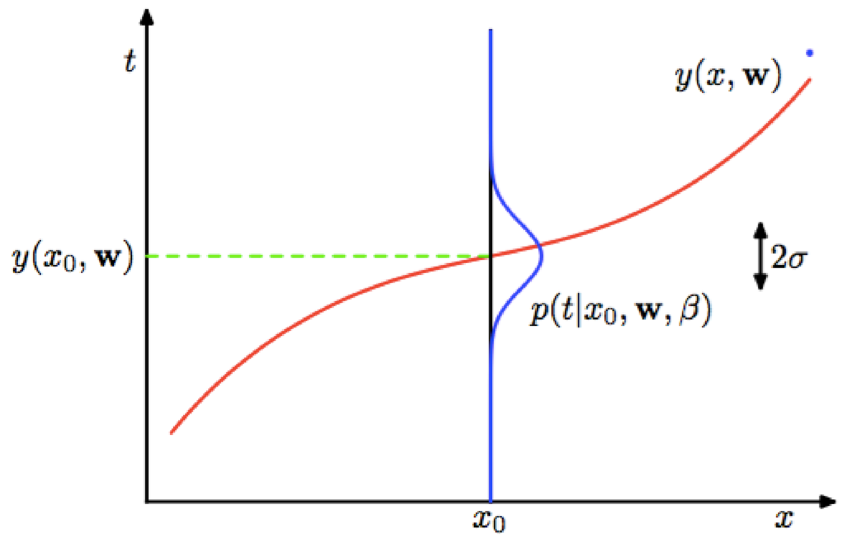


图 1.15: Schematic illustration of a Gaussian conditional distribution for t given x given by (1.50), in which the mean is given by the polynomial function $y(x, \mathbf{w})$, and the precision is given by the parameter β, which is related to the variance by $\beta^{-1} = \sigma^2$.

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

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 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 M$^{\text{th}}$ 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

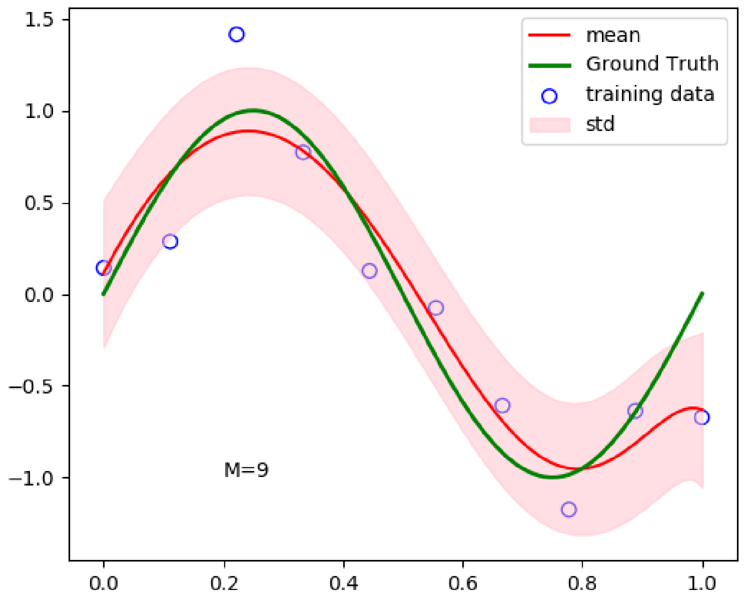


图 1.16: The predictive distribution resulting from a Bayesian treatment of polynomial curve fitting using an $M = 9$ polynomial, with the fixed parameters $\alpha = 5\times10^{-3}$ and $\beta = 11.1$ (corresponding to the known noise variance), in which the red curve denotes the mean of the predictive 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,

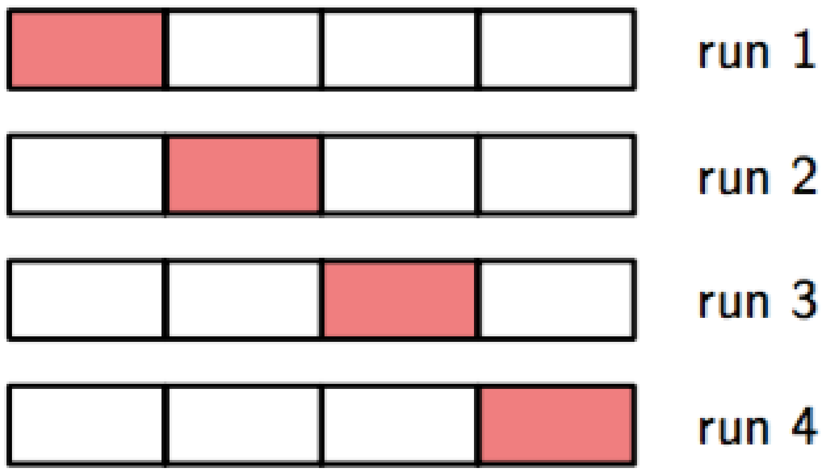


图 1.17: The technique of S-fold cross-validation, illustrated here for the case of S = 4, involves taking the available data and partitioning it into S groups (in the simplest case these are of equal size). Then S 1 of the groups are used to train a set of models that are then evaluated on the remaining 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(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

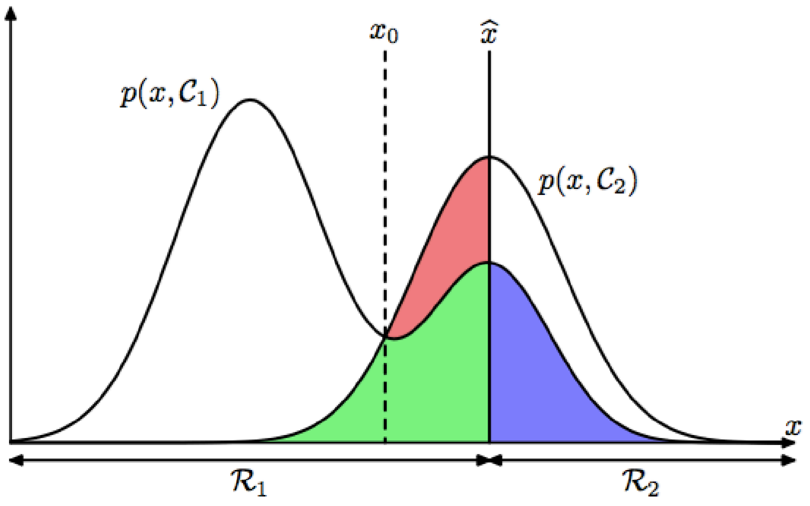


图 1.18: Schematic illustration of the joint probabilities $p(x, C_k)$ for each of two classes plotted against x, together with the decision boundary $x = \bar{x}$. Values of $x \geqslant \bar{x}$ are classified as class C_2 and hence belong to decision region R_2, whereas points $x < \bar{x}$ are classified 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

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

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 \$$\mathrm{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. 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.

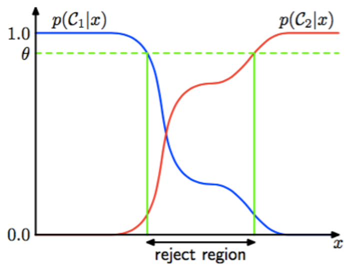


图 1.20: Illustration of the reject option. Inputs x such that the larger of the two posteior probabilities is less than or equal to θ some threshold θ will be rejected.

1.5.4 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 decisions. 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(热力学) 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(乘数)*. 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

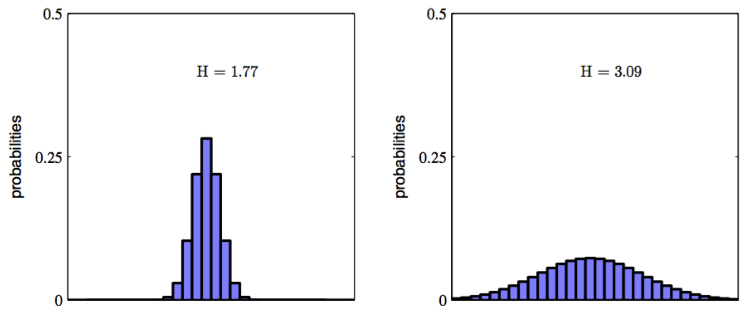


图 1.21: Histograms of two probability distributions over 30 bins illustrating the higher value of the entropy H for the broader distribution. The largest entropy would arise from a uniform distribution that would give $H = -ln(1/30) = 3.40$.

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

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

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

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

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

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

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:

$$\underset{w}{min}\, ||Xw - y||_2{}^2$$

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

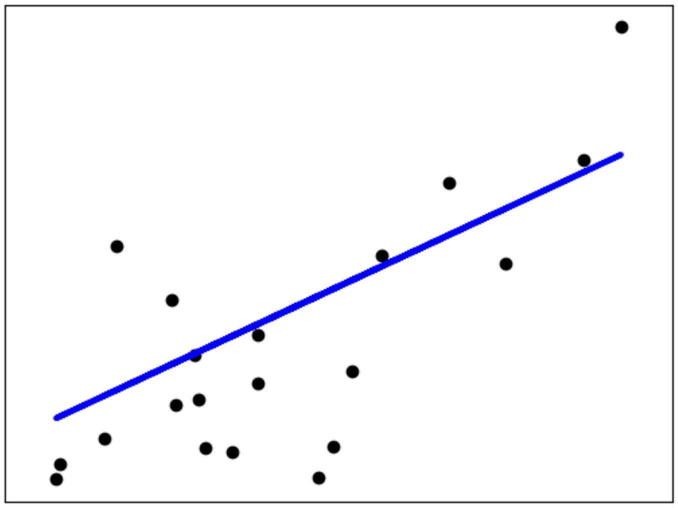
However, coefficient estimates for Ordinary Least Squares rely on the independence of the model terms. When terms are correlated and the columns of the design matrix X have an approximate linear dependence, the design matrix becomes close to singular and as a result, the least-squares estimate becomes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

```python
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 $\text{n}_{\text{samples}}$ is defined as

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

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 $_\text{i}$ 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 $\text{n}_{\text{samples}}$ 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

$\text{R}^2(\text{y},) = 1 - \{\frac{}{\sum_{\text{i=0}}^{\text{n}_\text{samples} - 1}} (\text{y}_\text{i} - {}_\text{i})^2\}\{\sum_\text{i=0}^\text{n}$

$\{\text{samples}\} - 1 \ (\text{y}_\text{i} -)^2\}$

where $= 1\frac{}{\{n_\text{samples}}\} \sum_{\text{i=0}}^{\text{n}_\text{samples} - 1} \text{y}_\text{i}.$

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,

$$\underset{w}{min}\, ||Xw - y||_2{}^2 + \alpha||w||_2{}^2$$

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.

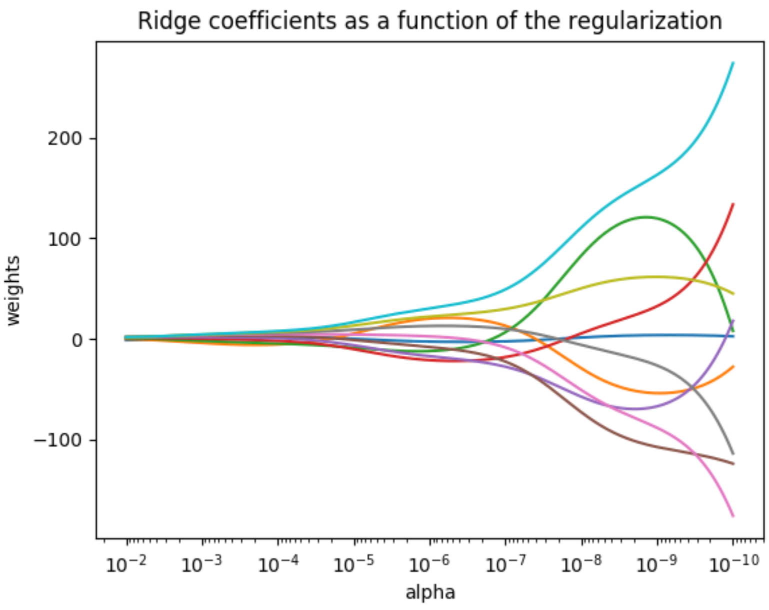
`Ridge` Regression is the estimator used in this example. Each color represents a different feature of the coefficient vector, and this is displayed as a function of the regularization parameter.

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.

```python
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:

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

Using the naive independence assumption that

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

for all i, this relationship is simplified to

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

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

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

$$\Downarrow$$

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

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

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

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.

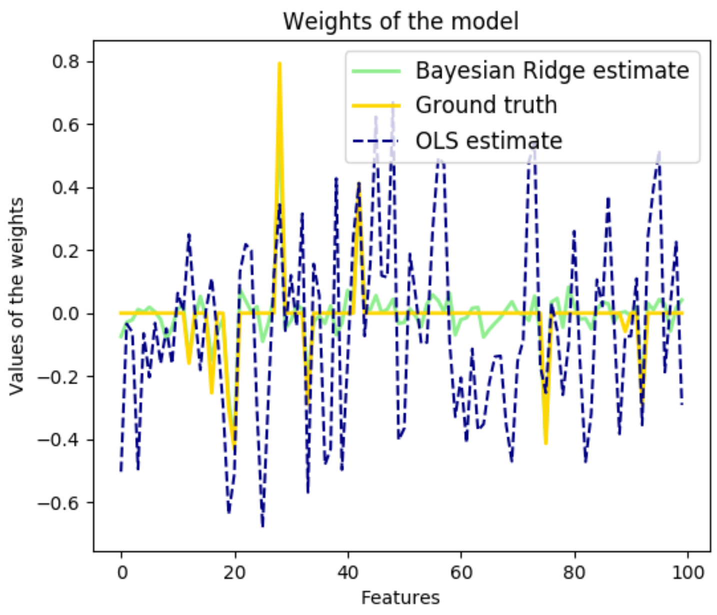
   The estimation of the model is done by iteratively maximizing the marginal log-likelihood of the observations.

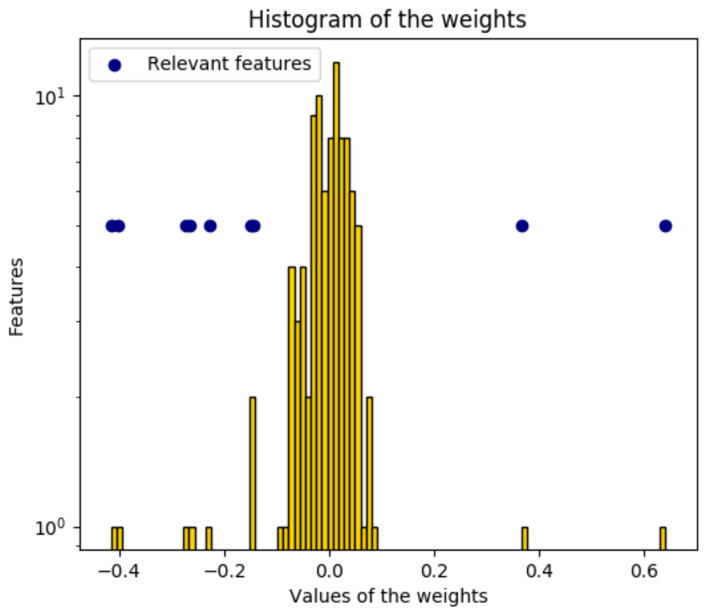
   We also plot predictions and uncertainties for Bayesian Ridge Regression for one dimensional regression using polynomial feature expansion. Note the uncertainty starts going up on the right side of the plot. This is because these test samples are outside of the range of the training samples.

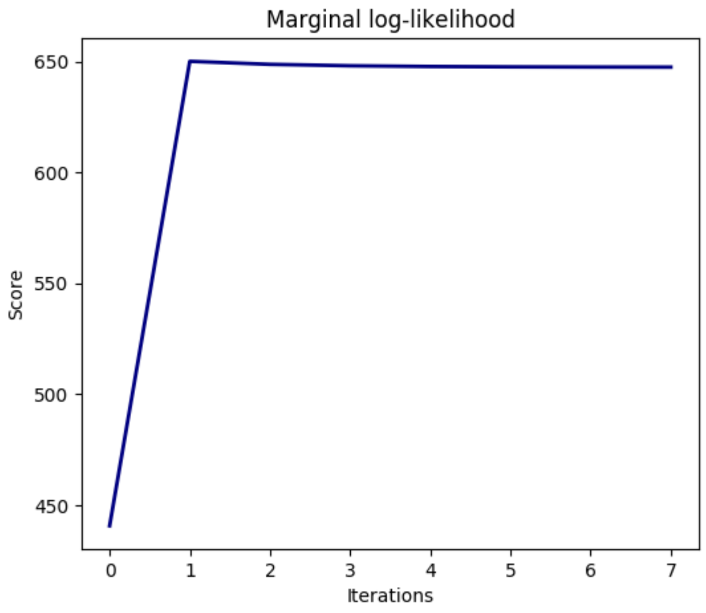
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.linear_model import BayesianRidge, LinearRegression
# ###############################################################################
# Generating simulated data with Gaussian weights
np.random.seed(0)
n_samples, 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)
```

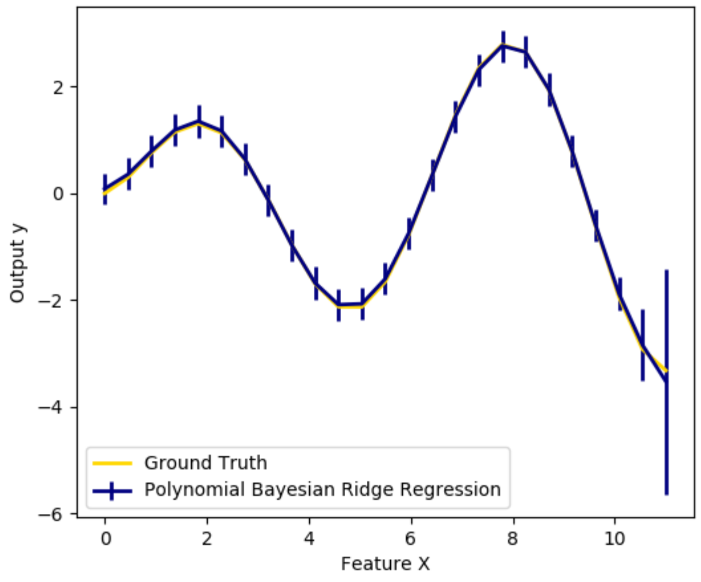
```
# #############################################################################
# Plot true weights, estimated weights, histogram of the weights, and
# predictions with standard deviations
lw = 2
plt.figure(figsize=(6, 5))
plt.title("Weights of the model")
plt.plot(
    clf.coef_,
    color='lightgreen',
    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")
```

```python
# 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