Python机器学习基础教程

Stephen CUI

February 17, 2023

Contents

List of Figures

Chapter 1

监督学习

1.1 分类与回归

监督机器学习问题主要有两种，分别叫作分类（classification）与回归（regression）。

分类问题的目标是预测类别标签（class label），这些标签来自预定义的可选列表。分类问题有时可分为二分类（binary classification，在两个类别之间进行区分的一种特殊情况）和多分类（multiclass classification，在两个以上的类别之间进行区分）。在二分类问题中，我们通常将其中一个类别称为正类（positive class），另一个类别称为反类（negative class）。这里的“正”并不代表好的方面或正数，而是代表研究对象。

回归任务的目标是预测一个连续值，编程术语叫作浮点数（floating-point number），数学术语叫作实数（real number）。

区分分类任务和回归任务有一个简单方法，就是问一个问题：输出是否具有某种连续性。如果在可能的结果之间具有连续性，那么它就是一个回归问题。

1.2 泛化、过拟合与欠拟合

在监督学习中，我们想要在训练数据上构建模型，然后能够对没见过的新数据（这些新数据与训练集具有相同的特性）做出准确预测。如果一个模型能够对没见过的数据做出准确预测，我们就说它能够从训练集**泛化**（generalize）到测试集。我们想要构建一个泛化精度尽可能高的模型。

判断一个算法在新数据上表现好坏的唯一度量，就是在测试集上的评估。然而从直觉上看[1]，我们认为简单的模型对新数据的泛化能力更好。因此，我们总想找到最简单的模型。构建一个对现有信息量来说过于复杂的模型，这被称为**过拟合**（overfitting）。如果你在拟合模型时过分关注训练集的细节，得到了一个在训练集上表现很好、但不能泛化到新数据上的模型，那么就存在过拟合。与之相反，如果你的模型过于简单，那么你可能无法抓住数据的全部内容以及数据中的变化，你的模型甚至在训练集上的表现就很差。选择过于简单的模型被称为**欠拟合**（underfitting）。

模型复杂度与数据集大小的关系

需要注意，模型复杂度与训练数据集中输入的变化密切相关：数据集中包含的数据点的变化范围越大，在不发生过拟合的前提下你可以使用的模型就越复杂。通常来说，收集更多的数据点可以有更

[1] 在数学上也可以证明这一点。

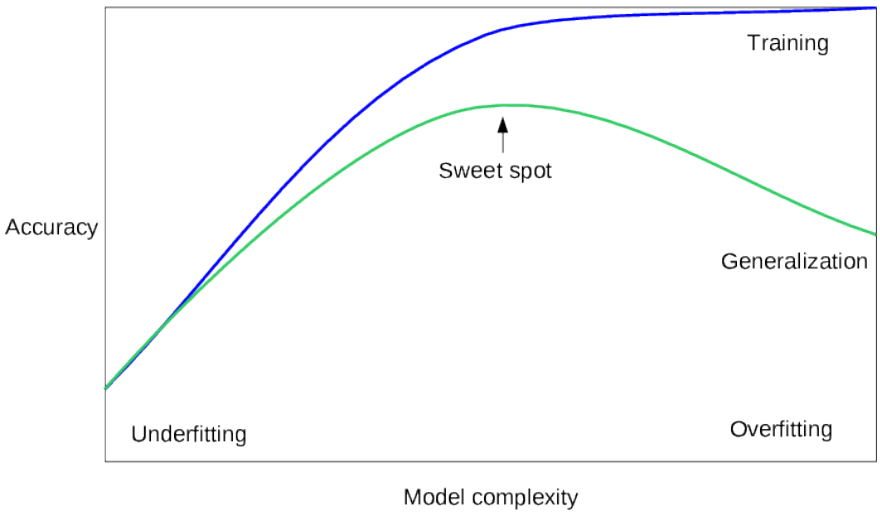

Figure 1.1: Trade-off of model complexity against training and test accuracy

Table 1.1: 一些样本数据集

名称	来源	特征量	用途
forge	模拟	26×2	二分类数据集
wave	模拟	40×1	回归算法数据集
cancer	真实	569×30	二分类数据集
boston	真实	506×13	回归数据集

大的变化范围，所以更大的数据集可以用来构建更复杂的模型。但是，仅复制相同的数据点或收集非常相似的数据是无济于事的。

收集更多数据，适当构建更复杂的模型，对监督学习任务往往特别有用。本书主要关注固定大小的数据集。在现实世界中，你往往能够决定收集多少数据，这可能比模型调参更为有效。永远不要低估更多数据的力量!

1.3 监督学习算法

现在开始介绍最常用的机器学习算法，并解释这些算法如何从数据中学习以及如何预测。我们还会讨论每个模型的复杂度如何变化，并概述每个算法如何构建模型。我们将说明每个算法的优点和缺点，以及它们最适应用于哪类数据。此外还会解释最重要的参数和选项的含义，许多算法都有分类和回归两种形式。

1.3.1 一些样本数据集

我们将使用一些数据集来说明不同的算法。其中一些数据集很小，而且是模拟的，其目的是强调算法的某个特定方面。其他数据集都是现实世界的大型数据集。

从特征较少的数据集（也叫低维数据集）中得出的结论可能并不适用于特征较多的数据集（也叫高维数据集）。只要你记住这一点，那么在低维数据集上研究算法也是很有启发的。

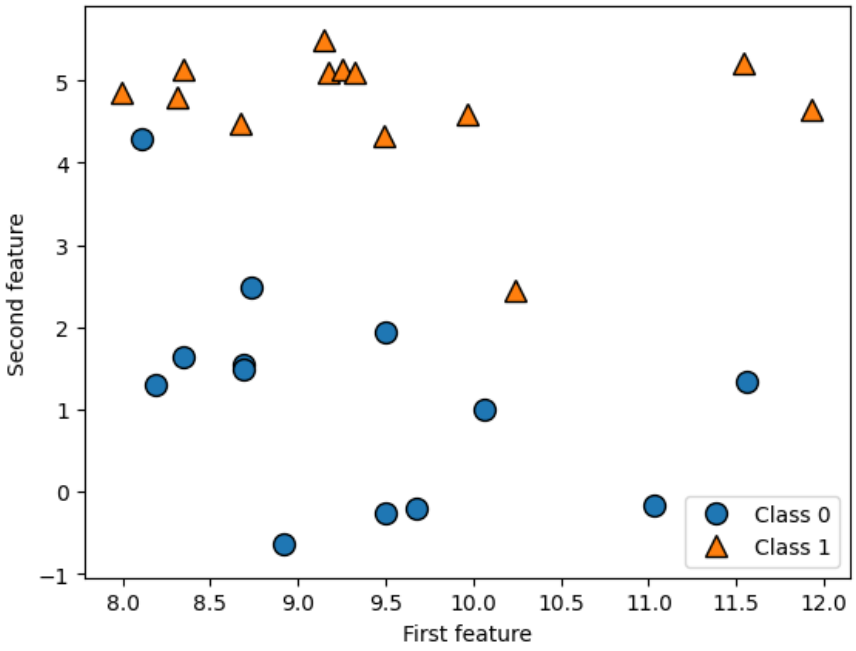

Figure 1.2: Scatter plot of the forge dataset

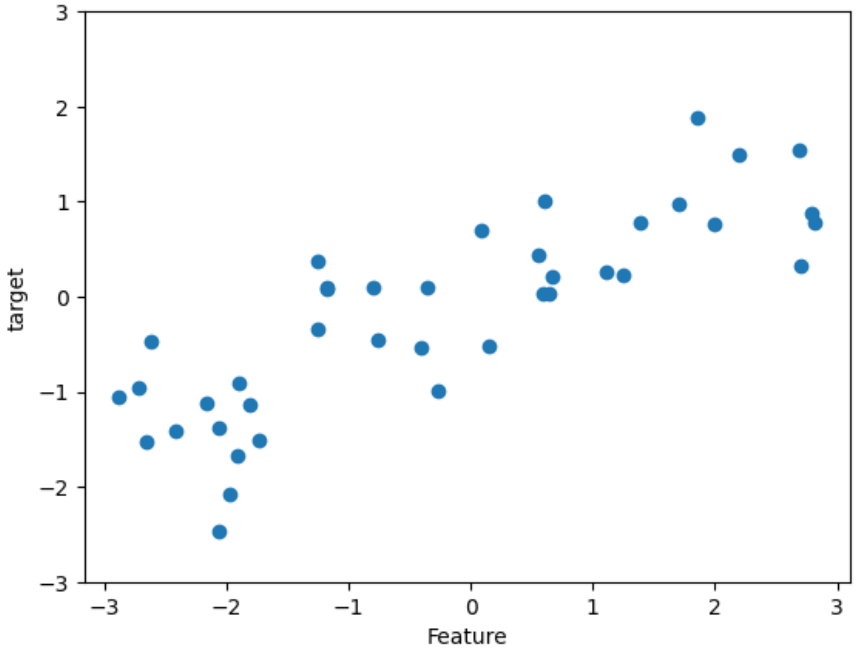

Figure 1.3: Plot of the wave dataset

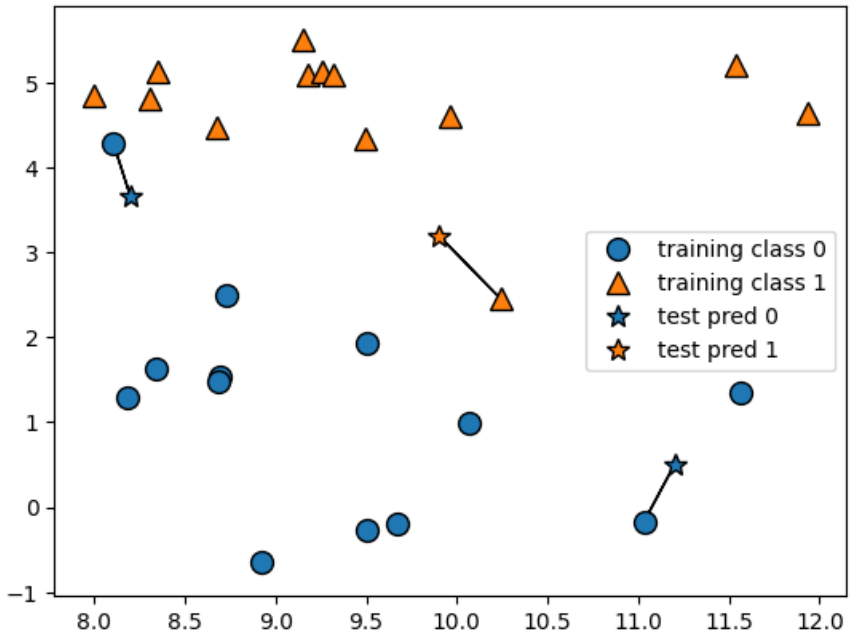

Figure 1.4: Predictions made by the one-nearest-neighbor model on the forge dataset

> 包含在 scikit-learn 中的数据集通常被保存为 Bunch 对象，里面包含真实数据以及一些数据集信息。关于 Bunch 对象，你只需要知道它与字典很相似，而且还有一个额外的好处，就是你可以用点操作符来访问对象的值（比如用 `bunch.key` 来代替 `bunch['key']`）。

对于我们的目的而言，我们需要扩展boston数据集，输入特征不仅包括这 13 个测量结果，还包括这些特征之间的乘积（也叫**交互项**）。换句话说，我们不仅将犯罪率和公路可达性作为特征，还将犯罪率和公路可达性的乘积作为特征。像这样包含导出特征的方法叫作**特征工程**（feature engineering）。

1.3.2 k近邻

k-NN 算法可以说是最简单的机器学习算法。构建模型只需要保存训练数据集即可。想要对新数据点做出预测，算法会在训练数据集中找到最近的数据点，也就是它的“最近邻”。

k近邻分类 k-NN 算法最简单的版本只考虑一个最近邻，也就是与我们想要预测的数据点最近的训练数据点。预测结果就是这个训练数据点的已知输出。Figure 1.4给出了这种分类方法在 forge 数据集上的应用。

除了仅考虑最近邻，我还可以考虑任意个（k 个）邻居。这也是 k 近邻算法名字的来历。在考虑多于一个邻居的情况时，我们用“投票法”（voting）来指定标签。将出现次数更多的类别（也就是 k 个近邻中占多数的类别）作为预测结果。

分析KNeighborsClassifier 分别将 1 个、3 个和 9 个邻居三种情况的决策边界可视化，见Figure 1.6。

从左图可以看出，使用单一邻居绘制的决策边界紧跟着训练数据。随着邻居个数越来越多，决策边界也越来越平滑。更平滑的边界对应更简单的模型。换句话说，使用更少的邻居对应更高的模型复杂度（如Figure 1.1右侧所示），而使用更多的邻居对应更低的模型复杂度（如Figure 1.1左侧所示）。

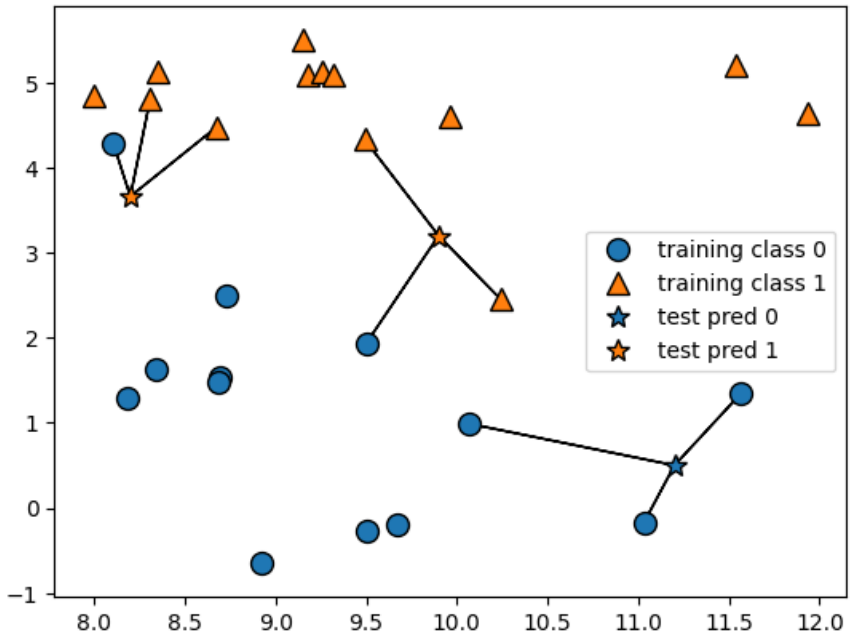

Figure 1.5: Predictions made by the three-nearest-neighbors model on the forge dataset

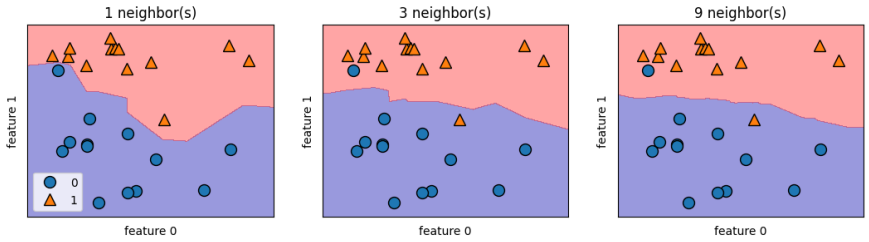

Figure 1.6: Decision boundaries created by the nearest neighbors model

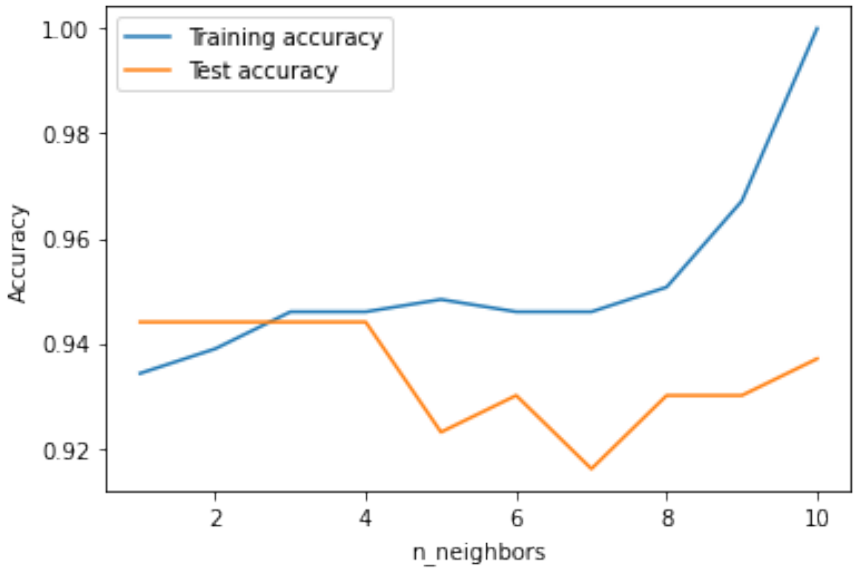

Figure 1.7: Comparison of training and test accuracy

假如考虑极端情况，即邻居个数等于训练集中所有数据点的个数，那么每个测试点的邻居都完全相同（即所有训练点），所有预测结果也完全相同（即训练集中出现次数最多的类别）。

我们来研究一下能否证实之前讨论过的模型复杂度和泛化能力之间的关系。先将数据集分成训练集和测试集，然后用不同的邻居个数对训练集和测试集的性能进行评估。输出结果见Figure 1.7：

注意，由于更少的邻居对应更复杂的模型，所以做了水平翻转。仅考虑单一近邻时，训练集上的预测结果十分完美。但随着邻居个数的增多，模型变得更简单，训练集精度也随之下降。单一邻居时的测试集精度比使用更多邻居时要低，这表示单一近邻的模型过于复杂。与之相反，当考虑 10 个邻居时，模型又过于简单，性能甚至变得更差。最佳性能在中间的某处，邻居个数大约为 6。

k近邻回归 k 近邻算法还可以用于回归。我们还是先从单一近邻开始，这次使用 wave 数据集。我们添加了 3 个测试数据点，在 x 轴上用绿色五角星表示。利用单一邻居的预测结果就是最近邻的目标值。在Figure 1.8中用蓝色五角星表示。

我们还可以用score方法来评估模型，对于回归问题，这一方法返回的是 R^2分数。R^2分数也叫作决定系数，是回归模型预测的优度度量，位于0到1之间。R^2等于1对应完美预测，R^2等于0对应常数模型，即总是预测训练集响应（`y_train`）的平均值：

分析KNeighborsRegressor 对于我们的一维数据集，可以查看所有特征取值对应的预测结果（图Figure 1.8）。

从图中可以看出，仅使用单一邻居，训练集中的每个点都对预测结果有显著影响，预测结果的图像经过所有数据点。这导致预测结果非常不稳定。考虑更多的邻居之后，预测结果变得更加平滑，但对训练数据的拟合也不好。

优点、缺点和参数 一般来说，KNeighbors 分类器有 2 个重要参数：邻居个数与数据点之间距离的度量方法。在实践中，使用较小的邻居个数（比如 3 个或 5 个）往往可以得到比较好的结果，但你应该

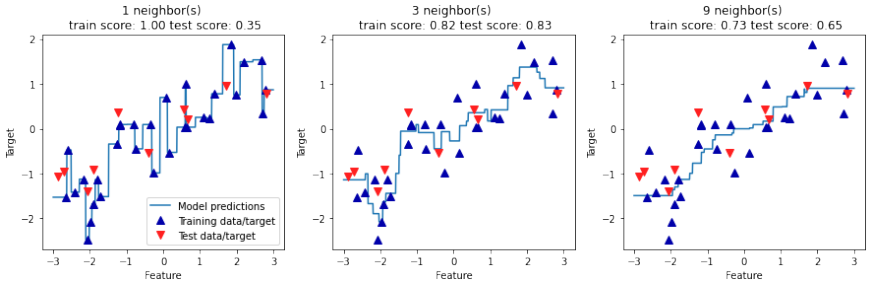

Figure 1.8: Comparing predictions made by nearest neighbors regression

调节这个参数。选择合适的距离度量方法超出了本书的范围。默认使用欧式距离，它在许多情况下的效果都很好。

k-NN 的优点之一就是模型很容易理解，通常不需要过多调节就可以得到不错的性能。在考虑使用更高级的技术之前，尝试此算法是一种很好的基准方法。构建最近邻模型的速度通常很快，但如果训练集很大（特征数很多或者样本数很大），预测速度可能会比较慢。使用 k-NN 算法时，对数据进行预处理是很重要的（见第 3 章）。这一算法对于有很多特征（几百或更多）的数据集往往效果不好，对于大多数特征的大多数取值都为 0 的数据集（所谓的稀疏数据集）来说，这一算法的效果尤其不好。

虽然 k 近邻算法很容易理解，但由于预测速度慢且不能处理具有很多特征的数据集，所以在实践中往往不会用到。

1.3.3 线性模型

线性模型利用输入特征的**线性函数**（linear function）进行预测

用于回归的线性模型 对于回归问题，线性模型预测的一般公式如下：

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \cdots + w[p] * x[p] + b$$

这里 $x[0]$ 到 $x[p]$ 表示单个数据点的特征（本例中特征个数为 $p+1$），w 和 b 是学习模型的参数，$\hat{y}$ 是模型的预测结果。对于单一特征的数据集，公式如下：

$$\hat{y} = w[0] * x[0] + b$$

对于有更多特征的数据集，w 包含沿每个特征坐标轴的斜率。或者，你也可以将预测的响应值看作输入特征的加权求和，权重由 w 的元素给出（可以取负值）。

用于回归的线性模型可以表示为这样的回归模型：对单一特征的预测结果是一条直线，两个特征时是一个平面，或者在更高维度（即更多特征）时是一个超平面。

如果将直线的预测结果与Figure 1.9 中 KNeighborsRegressor 的预测结果进行比较，你会发现直线的预测能力非常受限。似乎数据的所有细节都丢失了。从某种意义上来说，这种说法是正确的。假设目标 y 是特征的线性组合，这是一个非常强的（也有点不现实的）假设。但观察一维数据得出的观点有些片面。对于有多个特征的数据集而言，线性模型可以非常强大。特别地，如果特征数量大于训练数据点的数量，任何目标 y 都可以（在训练集上）用线性函数完美拟合。

线性回归（又名普通最小二乘法） 线性回归，或者**普通最小二乘法**（ordinary least squares，OLS），是回归问题最简单也最经典的线性方法。线性回归寻找参数 w 和 b，使得对训练集的预测值与真实的

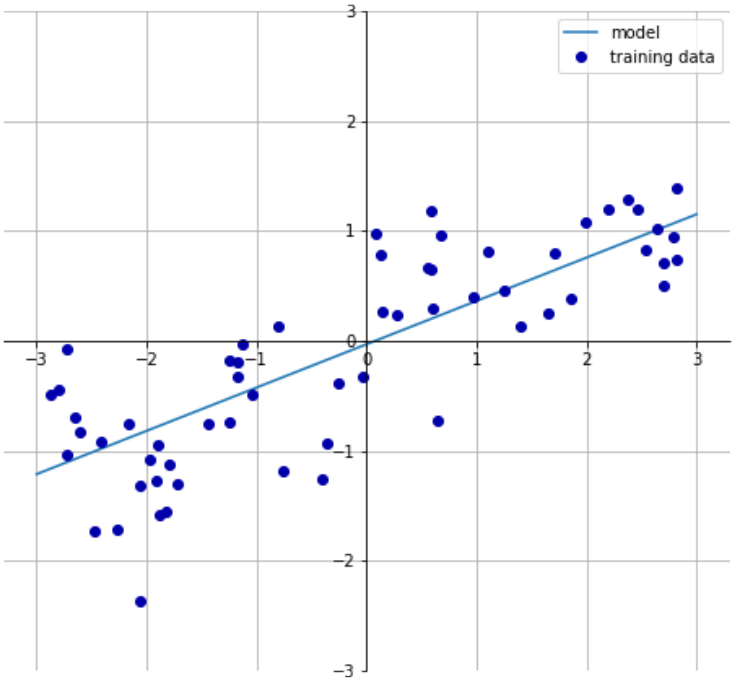

Figure 1.9: Predictions of a linear model on the wave dataset

回归目标值 y 之间的均方误差最小。**均方误差**（*mean squared error*）是预测值与真实值之差的平方和除以样本数。线性回归没有参数，这是一个优点，但也因此无法控制模型的复杂度。

斜率”参数（w，也叫作权重或系数）被保存在 `coef_` 属性中，而偏移或截距（b）被保存在 `intercept_` 属性中：

> 注意到了 `coef_` 和 `intercept_` 结尾处奇怪的下划线。scikit-learn总是将从训练数据中得出的值保存在以下划线结尾的属性中。这是为了将其与用户设置的参数区分开。

`intercept_` 属性是一个浮点数，而 `coef_` 属性是一个 NumPy 数组，每个元素对应一个输入特征。由于 wave 数据集中只有一个输入特征，所以 `lr.coef_` 中只有一个元素。

我们来看一下训练集和测试集的性能：

```
print('Training set score: {:.2f}'.format(lr.score(X_train, y_train)))
print('Training set score: {:.2f}'.format(lr.score(X_test, y_test)))
# 'Training set score: 0.67'
# 'Training set score: 0.66'
```

R^2 约为 0.66，这个结果不是很好，但我们可以看到，训练集和测试集上的分数非常接近。这说明可能存在欠拟合，而不是过拟合。对于这个一维数据集来说，过拟合的风险很小，因为模型非常简单（或受限）。然而，对于更高维的数据集（即有大量特征的数据集），线性模型将变得更加强大，过拟合的可能性也会变大。

```
print('Training set score: {:.2f}'.format(lr.score(X_train, y_train)))
print('Training set score: {:.2f}'.format(lr.score(X_test, y_test)))
# Training set score: 0.95
# Training set score: 0.61
```

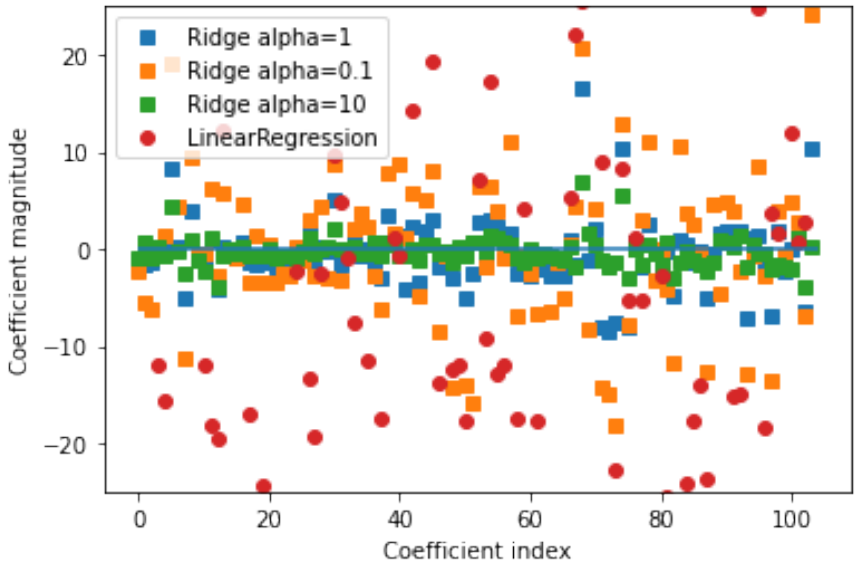

Figure 1.10: Comparing coefficient magnitudes for ridge regression

训练集和测试集之间的性能差异是过拟合的明显标志，因此我们应该试图找到一个可以控制复杂度的模型。标准线性回归最常用的替代方法之一就是岭回归（ridge regression）

岭回归 在岭回归中，对系数（w）的选择不仅要在训练数据上得到好的预测结果，而且还要拟合附加约束。我们还希望系数尽量小。换句话说，w 的所有元素都应接近于 0。直观上来看，这意味着每个特征对输出的影响应尽可能小（即斜率很小），同时仍给出很好的预测结果。这种约束是所谓正则化（regularization）的一个例子。正则化是指对模型做显式约束，以避免过拟合。岭回归用到的这种被称为 L2 正则化。

```
from sklearn.linear_model import Ridge
ridge = Ridge().fit(X_train, y_train)
print('Training set score: {:.2f}'.format(ridge.score(X_train, y_train)))
print('Training set score: {:.2f}'.format(ridge.score(X_test, y_test)))
# Training set score: 0.89
# Training set score: 0.75
```

可以看出，Ridge 在训练集上的分数要低于 LinearRegression，但在测试集上的分数更高。这和我们的预期一致。线性回归对数据存在过拟合。Ridge 是一种约束更强的模型，所以更不容易过拟合。复杂度更小的模型意味着在训练集上的性能更差，但泛化性能更好。由于我们只对泛化性能感兴趣，所以应该选择 Ridge 模型而不是 LinearRegression 模型。

Ridge 模型在模型的简单性（系数都接近于 0）与训练集性能之间做出权衡。简单性和训练集性能二者对于模型的重要程度可以由用户通过设置 alpha 参数来指定。alpha 的最佳设定值取决于用到的具体数据集。增大 alpha 会使得系数更加趋向于 0，从而降低训练集性能，但可能会提高泛化性能。

减小 alpha 可以让系数受到的限制更小，即在Figure 1.1中向右移动。对于非常小的 alpha 值，系数几乎没有受到限制，我们得到一个与 LinearRegression 类似的模型。

我们还可以查看 alpha 取不同值时模型的 `coef_` 属性，从而更加定性地理解 alpha 参数是如何改变模型的。更大的 alpha 表示约束更强的模型，所以我们预计大 alpha 对应的`coef_`元素比小 alpha 对应的

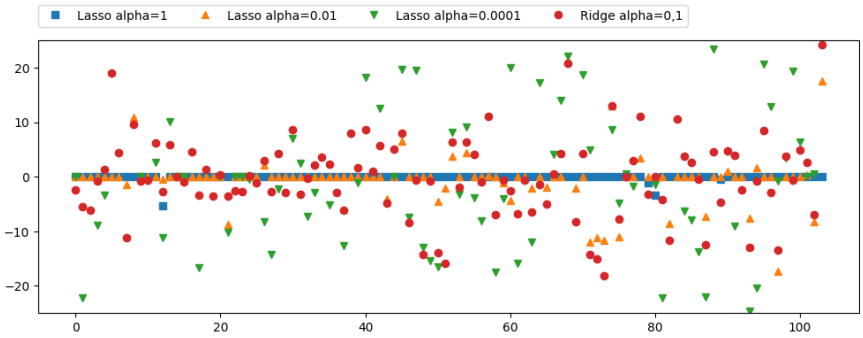

Figure 1.11: Comparing coefficient magnitudes for lasso regression

`coef_` 元素要小。这一点可以在Figure 1.10中得到证实。

还有一种方法可以用来理解正则化的影响，就是固定 alpha 值，但改变训练数据量。将模型性能作为数据集大小的函数进行绘图，这样的图像叫作**学习曲线**。

正如所预计的那样，无论是岭回归还是线性回归，所有数据集大小对应的训练分数都要高于测试分数。由于岭回归是正则化的，因此它的训练分数要整体低于线性回归的训练分数。但岭回归的测试分数要更高，特别是对较小的子数据集。如果少于 400 个数据点，线性回归学不到任何内容。随着模型可用的数据越来越多，两个模型的性能都在提升，最终线性回归的性能追上了岭回归。这里要记住的是，如果有足够多的训练数据，正则化变得不那么重要，并且岭回归和线性回归将具有相同的性能。**??**中还有一个有趣之处，就是线性回归的训练性能在下降。如果添加更多数据，模型将更加难以过拟合或记住所有的数据。

lasso 除了 Ridge，还有一种正则化的线性回归是 Lasso。与岭回归相同，使用 lasso 也是约束系数使其接近于 0，但用到的方法不同，叫作 L1 正则化。L1 正则化的结果是，使用 lasso 时某些系数刚好为 0。这说明某些特征被模型完全忽略。这可以看作是一种自动化的特征选择。某些系数刚好为 0，这样模型更容易解释，也可以呈现模型最重要的特征。

如你所见，Lasso 在训练集与测试集上的表现都很差。这表示存在欠拟合，我们发现模型只用到了 105 个特征中的 4 个。与 Ridge 类似，Lasso 也有一个正则化参数 alpha，可以控制系数趋向于 0 的强度。为了降低欠拟合，我们尝试减小 alpha。这么做的同时，我们还需要增加 `max_iter` 的值（运行迭代的最大次数）。

alpha 值变小，我们可以拟合一个更复杂的模型，在训练集和测试集上的表现也更好。模型性能比使用 Ridge 时略好一点，而且我们只用到了 105 个特征中的 33 个。这样模型可能更容易理解。

但如果把alpha设得太小，那么就会消除正则化的效果，并出现过拟合，得到LinearRegression 类似的结果：

在实践中，在两个模型中一般首选岭回归。但如果特征很多，你认为只有其中几个是重要的，那么选择 Lasso 可能更好。同样，如果你想要一个容易解释的模型，Lasso 可以给出更容易理解的模型，因为它只选择了一部分输入特征。scikit-learn 还提供了 ElasticNet类，结合了 Lasso 和 Ridge 的惩罚项。在实践中，这种结合的效果最好，不过代价是要调节两个参数：一个用于 L1 正则化，一个用于 L2 正则化。

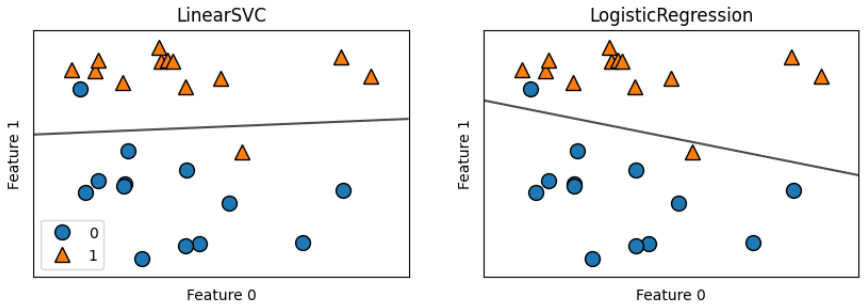

Figure 1.12: Decision boundaries of a linear SVM and logistic regression

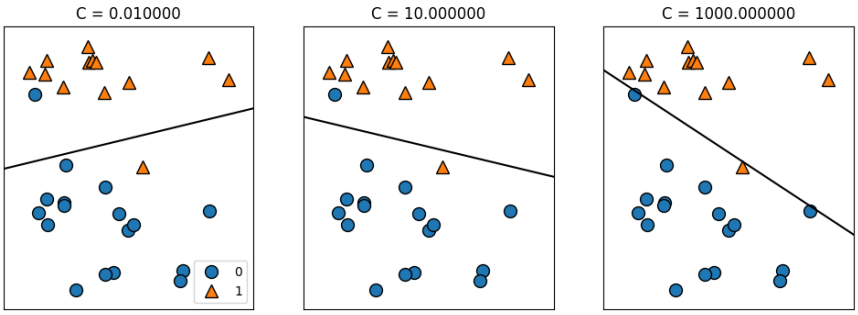

Figure 1.13: Decision boundaries of a linear SVM for different values of C

首先来看二分类。这时可以利用下面的公式进行预测：

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \cdots + w[p] * x[p] + b > 0$$

这个公式看起来与线性回归的公式非常相似，但我们没有返回特征的加权求和，而是为预测设置了阈值（0）。

对于用于回归的线性模型，输出 $\hat{y}$ 是特征的线性函数，是直线、平面或超平面（对于更高维的数据集）。对于用于分类的线性模型，**决策边界**是输入的线性函数。换句话说，（二元）线性分类器是利用直线、平面或超平面来分开两个类别的分类器。

学习线性模型有很多种算法。这些算法的区别在于以下两点：

- 系数和截距的特定组合对训练数据拟合好坏的度量方法；
- 是否使用正则化，以及使用哪种正则化方法。

由于数学上的技术原因，不可能调节 w 和 b 使得算法产生的误分类数量最少。对于我们的目的，以及对于许多应用而言，上面第一点（称为损失函数）的选择并不重要。

最常见的两种线性分类算法是 **Logistic回归**（logistic regression）和**线性支持向量机**（linear support vector machine，线性 SVM），

对于 LogisticRegression 和 LinearSVC，决定正则化强度的权衡参数叫作C。C值越大，对应的正则化越弱。换句话说，如果参数 C 值较大，那么 LogisticRegression和LinearSVC 将尽可能将训练集拟合到最好，而如果 C 值较小，那么模型更强调使系数量（*bmw*）接近于 0。

参数 C 的作用还有另一个有趣之处。较小的 C 值可以让算法尽量适应“大多数”数据点，而较大的 C 值更强调每个数据点都分类正确的重要性。

在Figure 1.13中，左图，C 值很小，对应强正则化。大部分属于类别 0 的点都位于底部，大部分属

于类别 1 的点都位于顶部。强正则化的模型会选择一条相对水平的线，有两个点分类错误。中间图，C 值稍大，模型更关注两个分类错误的样本，使决策边界的斜率变大。最后，右图，模型的 C 值非常大，使得决策边界的斜率也很大，现在模型对类别 0 中所有点的分类都是正确的。类别 1 中仍有一个点分类错误，这是因为对这个数据集来说，不可能用一条直线将所有点都分类正确。右侧图中的模型尽量使所有点的分类都正确，但可能无法掌握类别的整体分布。换句话说，这个模型很可能过拟合。

与回归的情况类似，用于分类的线性模型在低维空间中看起来可能非常受限，决策边界只能是直线或平面。同样，在高维空间中，用于分类的线性模型变得非常强大，当考虑更多特征时，避免过拟合变得越来越重要。

我们在乳腺癌数据集上详细分析 LogisticRegression：

```
# ConvergenceWarning
logreg = LogisticRegression().fit(X_train, y_train)
print('Training set score: {:.3f}'.format(logreg.score(X_train, y_train)))
print('Test set score: {:.3f}'.format(logreg.score(X_test, y_test)))
# Training set score: 0.946
# Test set score: 0.965
```

C=1 的默认值给出了相当好的性能，在训练集和测试集上都达到 95% 的精度。但由于训练集和测试集的性能非常接近，所以模型很可能是欠拟合的。欠拟合的模型可以考虑增加C值来更好的拟合模型。

```
# ConvergenceWarning
logreg100 = LogisticRegression(C=100).fit(X_train, y_train)
print('Training set score: {:.3f}'.format(logreg100.score(X_train, y_train)))
print('Test set score: {:.3f}'.format(logreg100.score(X_test, y_test)))
# Training set score: 0.944
# Test set score: 0.958
```

我们还可以研究使用正则化更强的模型时会发生什么。设置 $C = 0.01$：

```
# ConvergenceWarning
logreg001 = LogisticRegression(C=0.01).fit(X_train, y_train)
print('Training set score: {:.3f}'.format(logreg001.score(X_train, y_train)))
print('Test set score: {:.3f}'.format(logreg001.score(X_test, y_test)))
# Training set score: 0.934
# Test set score: 0.930
```

> 第 3 个系数那里发现有趣之处，这个系数是“平均周长”（mean perimeter），它的回归系数在不同的正则参数中符号发生了改变。这也说明，对线性模型系数的解释应该始终持保留态度。

优点、缺点和参数 线性模型的主要参数是正则化参数，在回归模型中叫作 alpha，在 LinearSVC 和 Logistic-Regression 中叫作 C。alpha 值较大或 C 值较小，说明模型比较简单。特别是对于回归模型而言，调节这些参数非常重要。通常在对数尺度上对 C 和 alpha 进行搜索。你还需要确定的是用 L1 正则化还是 L2 正则化。如果你假定只有几个特征是真正重要的，那么你应该用L1 正则化，否则应默认使用 L2 正则化。如果模型的可解释性很重要的话，使用 L1 也会有帮助。由于 L1 只用到几个特征，所以更容易解释哪些特征对模型是重要的，以及这些特征的作用。

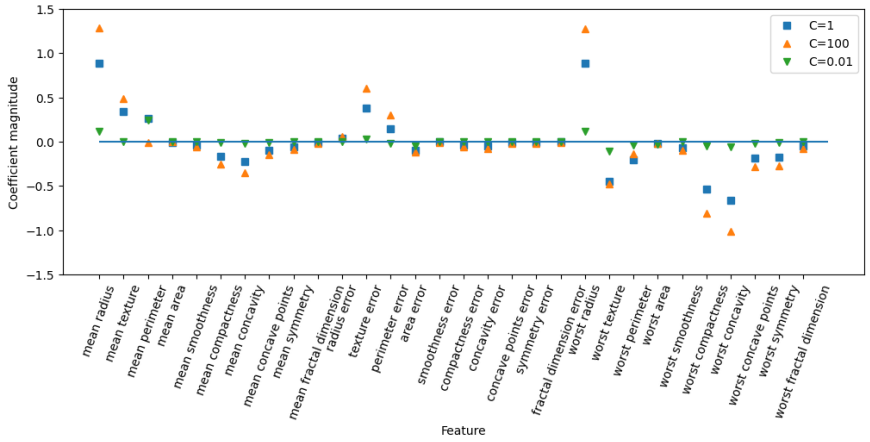

Figure 1.14: Coefficients learned by logistic regression for different C

线性模型的训练速度非常快，预测速度也很快。这种模型可以推广到非常大的数据集，对稀疏数据也很有效。如果你的数据包含数十万甚至上百万个样本，你可能需要研究如何使用 LogisticRegression 和 Ridge 模型的 solver='sag' 选项，在处理大型数据时，这一选项比默认值要更快。其他选项还有 SGDClassifier 类和 SGDRegressor 类，它们对本节介绍的线性模型实现了可扩展性更强的版本。

线性模型的另一个优点在于，利用我们之间见过的用于回归和分类的公式，理解如何进行预测是相对比较容易的。不幸的是，往往并不完全清楚系数为什么是这样的。如果你的数据集中包含高度相关的特征，这一问题尤为突出。在这种情况下，可能很难对系数做出解释。

如果特征数量大于样本数量，线性模型的表现通常都很好。它也常用于非常大的数据集，只是因为训练其他模型并不可行。但在更低维的空间中，其他模型的泛化性能可能更好。

方法链

scikit-learn 中所有模型的 fit 方法返回的都是 self。

```
logreg = LinearRegression().fit(X_train, y_train)
```

这里我们利用 fit 的返回值（即 self）将训练后的模型赋值给变量 logreg。这种方法调用的拼接（先调用 `__init__`，然后调用 fit）被称为方法链（method chaining）。scikit-learn 中方法链的另一个常见用法是在一行代码中同时 fit 和 predict。

```
logreg = LogisticRegression()
y_pred = logreg.fit(X_train, y_train).predict(X_test)
```

最后，你甚至可以在一行代码中完成模型初始化、拟合和预测：

```
y_pred = LogisticRegression().fit(X_train, y_train).predict(X_test)
```

不过这种非常简短的写法并不完美。一行代码中发生了很多事情，可能会使代码变得难以阅读。此外，拟合后的回归模型也没有保存在任何变量中，所以我们既不能查看它也不能用它来预测其他数据。

1.3.4 朴素贝叶斯分类器

朴素贝叶斯分类器是与线性模型非常相似的一种分类器，但它的训练速度往往更快。这种高效率所付出的代价是，朴素贝叶斯模型的泛化能力要比线性分类器（如LogisticRegression 和 LinearSVC）稍

差。朴素贝叶斯模型如此高效的原因在于，它通过单独查看每个特征来学习参数，并从每个特征中收集简单的类别统计数据。

scikit-learn 中实现了三种朴素贝叶斯分类器：GaussianNB、BernoulliNB 和 MultinomialNB。GaussianNB 可应用于任意连续数据，而BernoulliNB 假定输入数据为二分类数据，MultinomialNB 假定输入数据为计数数据。BernoulliNB 和MultinomialNB 主要用于文本数据分类。

1.3.5 决策树

决策树是广泛用于分类和回归任务的模型。本质上，它从一层层的 if/else 问题中进行学习，并得出结论。这一系列问题可以表示为一棵决策树，如图 2-22 所示。

make_moons

构造决策树

控制决策树的复杂度 通常来说，构造决策树直到所有叶结点都是纯的叶结点，这会导致模型非常复杂，并且对训练数据高度过拟合。纯叶结点的存在说明这棵树在训练集上的精度是 100%。训练集中的每个数据点都位于分类正确的叶结点中。

防止过拟合有两种常见的策略：一种是及早停止树的生长，也叫预剪枝（pre-pruning）；另一种是先构造树，但随后删除或折叠信息量很少的结点，也叫后剪枝（post-pruning）或剪枝（pruning）。预剪枝的限制条件可能包括限制树的最大深度、限制叶结点的最大数目，或者规定一个结点中数据点的最小数目来防止继续划分。

scikit-learn 的决策树在 DecisionTreeRegressor 类和 DecisionTreeClassifier 类中实现。scikit-learn 只实现了预剪枝，没有实现后剪枝。

分析决策树

树的特征重要性 回归树的用法和分析与分类树非常类似。但在将基于树的模型用于回归时，我们想要指出它的一个特殊性质。DecisionTreeRegressor（以及其他所有基于树的回归模型）不能外推（extrapolate），也不能在训练数据范围之外进行预测。

一旦输入超出了模型训练数据的范围，模型就只能持续预测最后一个已知数据点。树不能在训练数据的范围之外生成“新的”响应。所有基于树的模型都有这个缺点[2]。

优点、缺点和参数 控制决策树模型复杂度的参数是预剪枝参数，它在树完全展开之前停止树的构造。通常来说，选择一种预剪枝策略（设置 `max_depth`、`max_leaf_nodes` 或 `min_samples_leaf`）足以防止过拟合。

与前面讨论过的许多算法相比，决策树有两个优点：一是得到的模型很容易可视化，非专家也很容易理解（至少对于较小的树而言）；二是算法完全不受数据缩放的影响。由于每个特征被单独处理，而且数据的划分也不依赖于缩放，因此决策树算法不需要特征预处理，比如归一化或标准化。特别是特征的尺度完全不一样时或者二元特征和连续特征同时存在时，决策树的效果很好。

决策树的主要缺点在于，即使做了预剪枝，它也经常会过拟合，泛化性能很差。

[2]实际上，利用基于树的模型可以做出非常好的预测（比如试图预测价格会上涨还是下跌）。这个例子的目的并不是要说明对时间序列来说树是一个不好的模型，而是为了说明树在预测方式上的特殊性质。

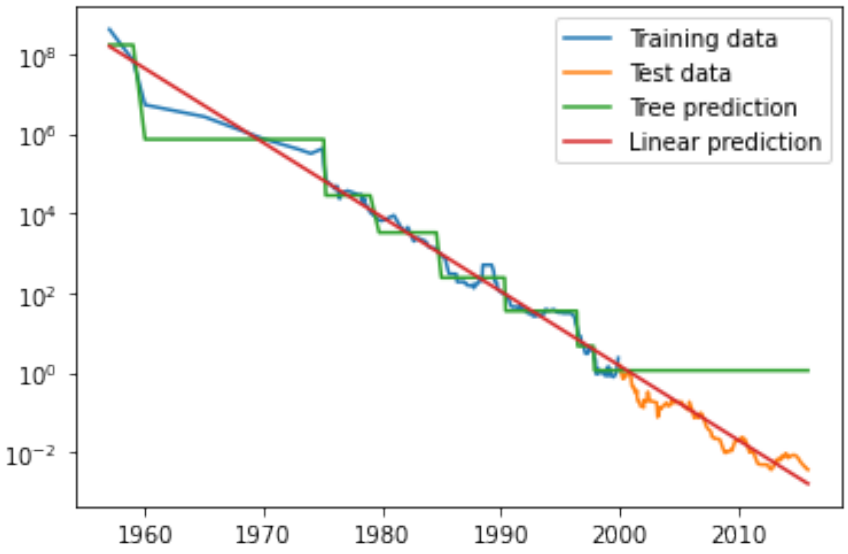

Figure 1.15: The tree has no ability to generate new responses

1.3.6 决策树集成

集成（ensemble）是合并多个机器学习模型来构建更强大模型的方法。

随机森林

构造随机森林

分析随机森林

优点、缺点和参数 用于回归和分类的随机森林是目前应用最广泛的机器学习方法之一。这种方法非常强大，通常不需要反复调节参数就可以给出很好的结果，也不需要对数据进行缩放。

从本质上看，随机森林拥有决策树的所有优点，同时弥补了决策树的一些缺陷。仍然使用决策树的一个原因是需要决策过程的紧凑表示。基本上不可能对几十棵甚至上百棵树做出详细解释，随机森林中树的深度往往比决策树还要大（因为用到了特征子集）。因此，如果你需要以可视化的方式向非专家总结预测过程，那么选择单棵决策树可能更好。

> 你应该记住，随机森林本质上是随机的，设置不同的随机状态（或者不设置 `random_state`参数）可以彻底改变构建的模型。森林中的树越多，它对随机状态选择的鲁棒性就越好。如果你希望结果可以重现，固定 `random_state` 是很重要的。

对于维度非常高的稀疏数据（比如文本数据），随机森林的表现往往不是很好。对于这种数据，使用线性模型可能更合适。即使是非常大的数据集，随机森林的表现通常也很好，训练过程很容易并行在功能强大的计算机的多个 CPU 内核上。不过，随机森林需要更大的内存，训练和预测的速度也比线性模型要慢。对一个应用来说，如果时间和内存很重要的话，那么换用线性模型可能更为明智。

需要调节的重要参数有 n_estimators 和 max_features，可能还包括预剪枝选项（如 max_depth）。n_estimators 总是越大越好。对更多的树取平均可以降低过拟合，从而得到鲁棒性更好的集成。不过收益是递减的，而且树越多需要的内存也越多，训练时间也越长。常用的经验法则就是“在你的时间 / 内存允许的情况下尽量多”。

max_features 决定每棵树的随机性大小，较小的 max_features 可以降低过拟合。一般来说，好的经验就是使用默认值：对于分类，默认值是 max_features=sqrt(n_features)；对于回归，默认值是 max_features=max_features。增大 max_features 或 max_leaf_nodes 有时也可以提高性能。它还可以大大降低用于训练和预测的时间和空间要求。

梯度提升回归树（梯度提升机） 虽然名字中含有“回归”，但这个模型既可以用于回归也可以用于分类。与随机森林方法不同，梯度提升采用连续的方式构造树，每棵树都试图纠正前一棵树的错误。默认情况下，梯度提升回归树中没有随机化，而是用到了强预剪枝。梯度提升树通常使用深度很小（1到5 之间）的树，这样模型占用的内存更少，预测速度也更快。

梯度提升背后的主要思想是合并许多简单的模型（在这个语境中叫作**弱学习器**），比如深度较小的树。每棵树只能对部分数据做出好的预测，因此，添加的树越来越多，可以不断迭代提高性能。

与随机森林相比，它通常对参数设置更为敏感，但如果参数设置正确的话，模型精度更高。

除了预剪枝与集成中树的数量之外，梯度提升的另一个重要参数是 learning_rate（学习率），用于控制每棵树纠正前一棵树的错误的强度。较高的学习率意味着每棵树都可以做出较强的修正，这样模型更为复杂。通过增大 n_estimators 来向集成中添加更多树，也可以增加模型复杂度，因为模型有更多机会纠正训练集上的错误。

> 由于梯度提升和随机森林两种方法在类似的数据上表现得都很好，因此一种常用的方法就是先尝试随机森林，它的鲁棒性很好。如果随机森林效果很好，但预测时间太长，或者机器学习模型精度小数点后第二位的提高也很重要，那么切换成梯度提升通常会有用。

想要将梯度提升应用在大规模问题上，可以研究一下 xgboost 包及其 Python 接口

优点、缺点和参数 梯度提升决策树是监督学习中最强大也最常用的模型之一。其主要缺点是需要仔细调参，而且训练时间可能会比较长。与其他基于树的模型类似，这一算法不需要对数据进行缩放就可以表现得很好，而且也适用于二元特征与连续特征同时存在的数据集。与其他基于树的模型相同，它也通常不适用于高维稀疏数据。

梯度提升树模型的主要参数包括树的数量 n_estimators 和学习率 learning_rate，后者用于控制每棵树对前一棵树的错误的纠正强度。这两个参数高度相关，因为 learning_rate 越低，就需要更多的树来构建具有相似复杂度的模型。随机森林的 n_estimators 值总是越大越好，但梯度提升不同，增大 n_estimators 会导致模型更加复杂，进而可能导致过拟合。通常的做法是根据时间和内存的预算选择合适的 n_estimators，然后对不同的learning_rate 进行遍历。

另一个重要参数是 max_depth（或 max_leaf_nodes），用于降低每棵树的复杂度。梯度提升模型的 max_depth 通常都设置得很小，一般不超过 5。

1.3.7 核支持向量机

核支持向量机（kernelized support vector machine, SVM）是可以推广到更复杂模型的扩展，这些模型无法被输入空间的超平面定义。虽然支持向量机可以同时用于分类和回归，但我们只会介绍用于分类的情况，它在 SVC 中实现。类似的概念也适用于支持向量回归，后者在 SVR 中实现。

核支持向量机背后的数学有点复杂，已经超出了本书的范围。你可以阅读 Hastie、Tibshirani 和 Friedman 合著的《统计学习基础》一书的第 12 章了解更多细节。

线性模型与非线性特征

核技巧

这里需要记住的是，向数据表示中添加非线性特征，可以让线性模型变得更强大。但是，通常来说我们并不知道要添加哪些特征，而且添加许多特征的计算开销可能会很大。幸运的是，有一种巧妙的数学技巧，让我们可以在更高维空间中学习分类器，而不用实际计算可能非常大的新的数据表示。这种技巧叫作核技巧（kernel trick），它的原理是直接计算扩展特征表示中数据点之间的距离（更准确地说是内积），而不用实际对扩展进行计算。

对于支持向量机，将数据映射到更高维空间中有两种常用的方法：一种是多项式核，在一定阶数内计算原始特征所有可能的多项式（比如 feature1 ** 2 * feature2 ** 5）；另一种是径向基函数（radial basis function，RBF）核，也叫高斯核。高斯核有点难以解释，因为它对应无限维的特征空间。一种对高斯核的解释是它考虑所有阶数的所有可能的多项式，但阶数越高，特征的重要性越小。

理解SVM

在训练过程中，SVM 学习每个训练数据点对于表示两个类别之间的决策边界的重要性。通常只有一部分训练数据点对于定义决策边界来说很重要：位于类别之间边界上的那些点。这些点叫作**支持向量**（support vector），支持向量机正是由此得名。

想要对新样本点进行预测，需要测量它与每个支持向量之间的距离。分类决策是基于它与支持向量之间的距离以及在训练过程中学到的支持向量重要性（保存在 SVC 的 `dual_coef_` 属性中）来做出的。数据点之间的距离由高斯核给出：

$$k_{rbf}(x_1, x_2) = exp(-\gamma||x_1 - x_2||^2)$$

这里 x_1 和 x_2 是数据点，$x_1 - x_2$ 表示欧氏距离，γ是控制高斯核宽度的参数。

SVM调参

Figure 1.16中，从左到右，我们将参数 gamma 的值从 0.1 增加到 10。gamma 较小，说明高斯核的半径较大，许多点都被看作比较靠近。这一点可以在图中看出：左侧的图决策边界非常平滑，越向右的图决策边界更关注单个点。小的 gamma 值表示决策边界变化很慢，生成的是复杂度较低的模型，而大的 gamma 值则会生成更为复杂的模型。

从上到下，我们将参数 C 的值从 0.1 增加到 1000。与线性模型相同，C 值很小，说明模型非常受限，每个数据点的影响范围都有限。你可以看到，左上角的图中，决策边界看起来几乎是线性的，误分类的点对边界几乎没有任何影响。再看左下角的图，增大 C 之后这些点对模型的影响变大，使得决策边界发生弯曲来将这些点正确分类。

虽然 SVM 的表现通常都很好，但它对参数的设定和数据的缩放非常敏感。特别地，它要求所有特征有相似的变化范围。我们来看一下每个特征的最小值和最大值，它们绘制在对数坐标上(Figure 1.17)

从Figure 1.17中，我们可以确定乳腺癌数据集的特征具有完全不同的数量级。这对其他模型来说（比如线性模型）可能是小问题，但对核 SVM 却有极大影响。

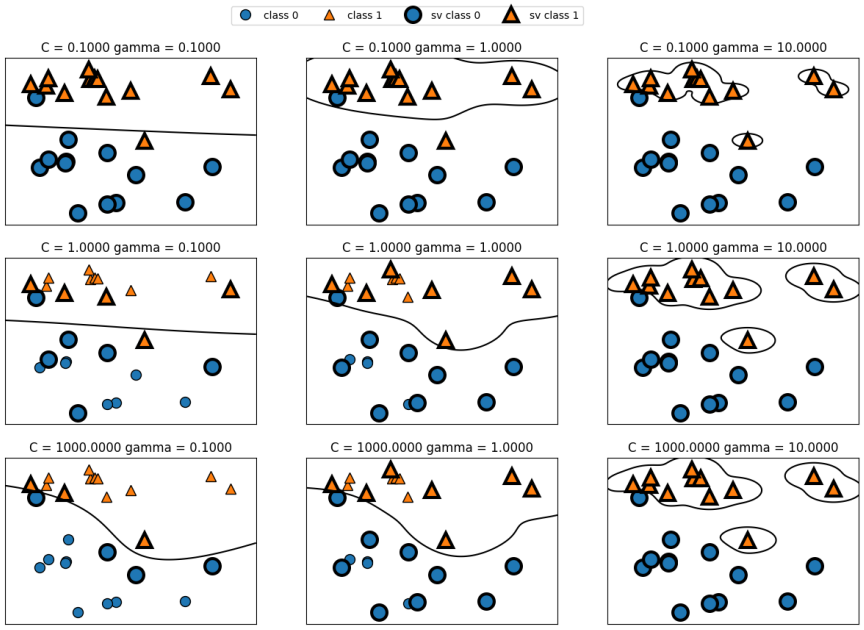

Figure 1.16: Decision boundaries and support vectors for different settings

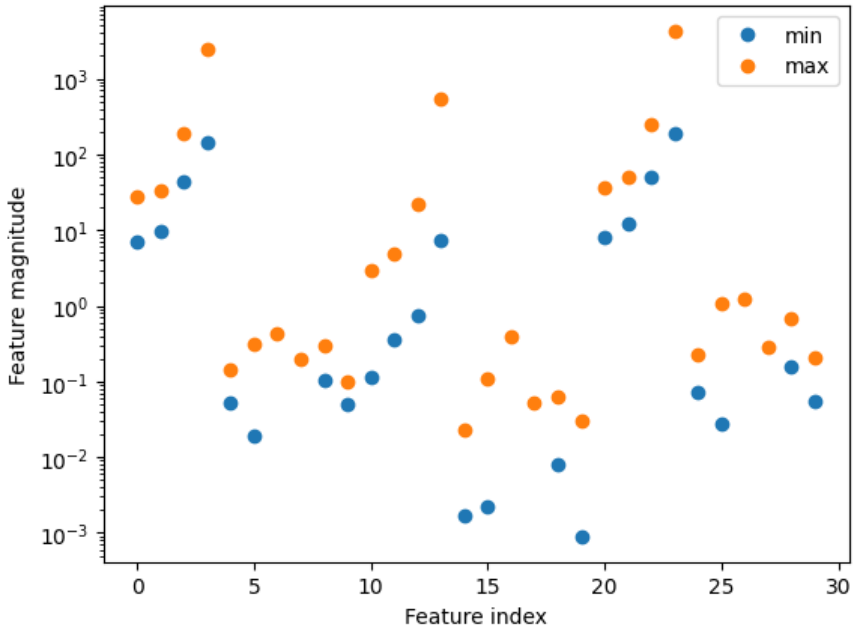

Figure 1.17: Feature ranges for the Breast Cancer dataset

为SVM预处理数据

解决这个问题的一种方法就是对每个特征进行缩放，使其大致都位于同一范围。核 SVM常用的缩放方法就是将所有特征缩放到 0 和 1 之间。我们将在Chapter 2学习如何使用 MinMaxScaler 预处理方法来做到这一点，到时会给出更多细节。

优点、缺点和参数

核支持向量机是非常强大的模型，在各种数据集上的表现都很好。SVM 允许决策边界很复杂，即使数据只有几个特征。它在低维数据和高维数据（即很少特征和很多特征）上的表现都很好，但对样本个数的缩放表现不好。在有多达 10 000 个样本的数据上运行 SVM可能表现良好，但如果数据量达到 100 000 甚至更大，在运行时间和内存使用方面可能会面临挑战。

SVM 的另一个缺点是，预处理数据和调参都需要非常小心。这也是为什么如今很多应用中用的都是基于树的模型，比如随机森林或梯度提升（需要很少的预处理，甚至不需要预处理）。此外，SVM 模型很难检查，可能很难理解为什么会这么预测，而且也难以将模型向非专家进行解释。

不过 SVM 仍然是值得尝试的，特别是所有特征的测量单位相似（比如都是像素密度）而且范围也差不多时。

核 SVM 的重要参数是正则化参数 C、核的选择以及与核相关的参数。虽然我们主要讲的是RBF 核，但 scikit-learn 中还有其他选择。RBF 核只有一个参数 gamma，它是高斯核宽度的倒数。gamma 和 C 控制的都是模型复杂度，较大的值都对应更为复杂的模型。因此，这两个参数的设定通常是强烈相关的，应该同时调节。

1.3.8 神经网络（深度学习）

虽然深度学习在许多机器学习应用中都有巨大的潜力，但深度学习算法往往经过精确调整，只适用于特定的使用场景。这里只讨论一些相对简单的方法，即用于分类和回归的**多层感知机**（multilayer perceptron，MLP），它可以作为研究更复杂的深度学习方法的起点。MLP 也被称为（普通）前馈神经网络，有时也简称为神经网络。

神经网络模型

MLP 可以被视为广义的线性模型（GLM），执行多层处理后得到结论。线性回归的预测公式简单来说，$\hat{y}$是输入特征 $x[0]$ 到 $x[p]$ 的加权求和，权重为学到的系数 $w[0]$ 到 $w[p]$。我们可以将这个公式可视化，

对于Figure 1.18所示的小型神经网络，计算回归问题的 $\hat{y}$ 的完整公式如下（使用 tanh 非线性）：

$$
\begin{aligned}
h[0] &= tanh(w[0,0]*x[0]+w[1,0]*x[1]+w[2,0]*x[2]+w[3,0]*x[3]+b[0]) \\
h[1] &= tanh(w[0,0]*x[0]+w[1,0]*x[1]+w[2,0]*x[2]+w[3,0]*x[3]+b[1]) \\
h[2] &= tanh(w[0,0]*x[0]+w[1,0]*x[1]+w[2,0]*x[2]+w[3,0]*x[3]+b[2]) \\
\hat{y} &= v[0]*h[0]+v[1]*h[1]+v[2]*h[2]+b
\end{aligned}
\tag{1.1}
$$

其中，w 是输入 x 与隐层 h 之间的权重，v 是隐层 h 与输出 $\hat{y}$ 之间的权重。权重 w 和 v 要从数据中学习得到，x 是输入特征，$\hat{y}$ 是计算得到的输出，h 是计算的中间结果。需要用户设置的一个重要参数是隐层中的结点个数。对于非常小或非常简单的数据集，这个值可以小到 10；对于非常复杂的数据，这个值可以大到 10 000。也可以添加多个隐层，如图Figure 1.19所示。

这些由许多计算层组成的大型神经网络，正是术语“深度学习”的灵感来源。

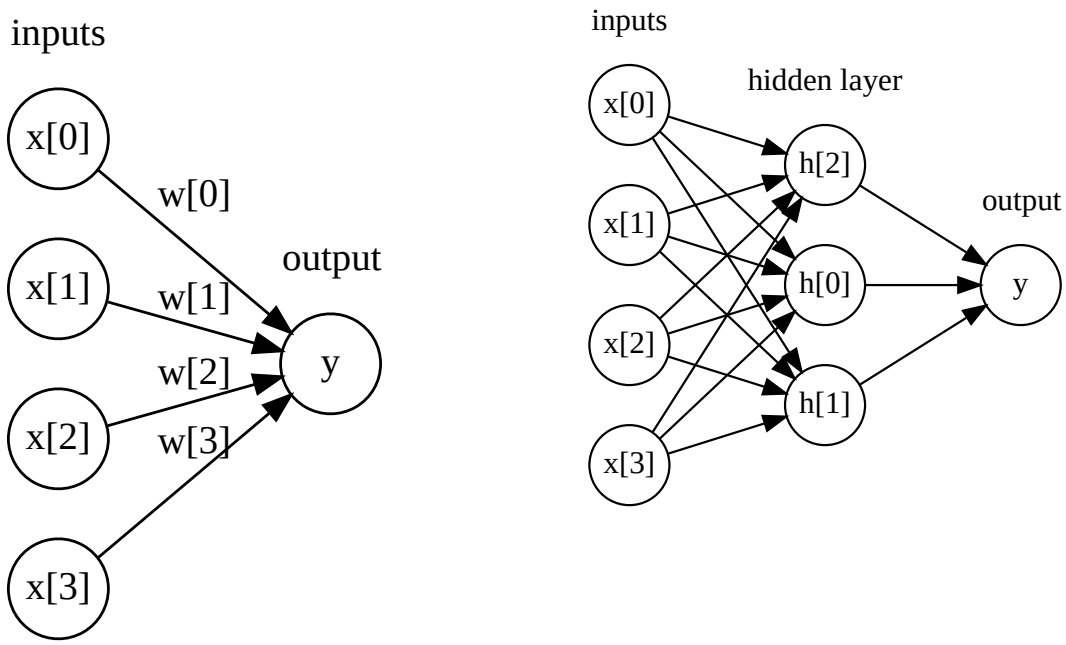

Figure 1.18: Illustration of a multilayer perceptron with a single hidden layer and linear model

神经网络调参

将 MLPClassifier 应用到之前用过的 `two_moons` 数据集上，以此研究 MLP 的工作原理。

默认情况下，MLP 使用 100 个隐结点，这对于这个小型数据集来说已经相当多了。我们可以减少其数量（从而降低了模型复杂度），但仍然得到很好的结果（图 2-49）：

Equation 1.1似乎有点问题，应该要写成矩阵的形式。

优点、缺点和参数

在机器学习的许多应用中，神经网络再次成为最先进的模型。它的主要优点之一是能够获取大量数据中包含的信息，并构建无比复杂的模型。给定足够的计算时间和数据，并且仔细调节参数，神经网络通常可以打败其他机器学习算法（无论是分类任务还是回归任务）。

这就引出了下面要说的缺点。神经网络——特别是功能强大的大型神经网络——通常需要很长的训练时间。它还需要仔细地预处理数据，正如我们这里所看到的。与 SVM 类似，神经网络在“均匀”数据上的性能最好，其中“均匀”是指所有特征都具有相似的含义。如果数据包含不同种类的特征，那么基于树的模型可能表现得更好。神经网络调参本身也是一门艺术。

估计神经网络的复杂度 最重要的参数是层数和每层的隐单元个数。你应该首先设置 1 个或 2 个隐层，然后可以逐步增加。每个隐层的结点个数通常与输入特征个数接近，但在几千个结点时很少会多于特征个数。

神经网络调参的常用方法是，首先创建一个大到足以过拟合的网络，确保这个网络可以对任务进行学习。知道训练数据可以被学习之后，要么缩小网络，要么增大 alpha 来增强正则化，这可以提高泛化性能。

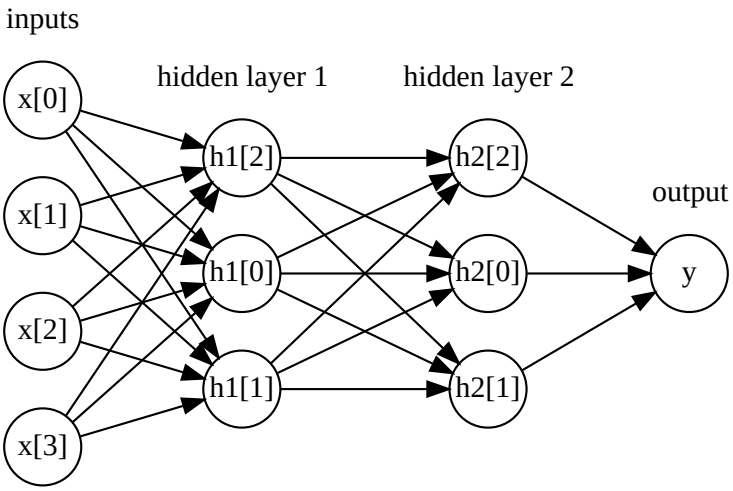

Figure 1.19: A multilayer perceptron with two hidden layers

主要关注模型的定义：层数、每层的结点个数、正则化和非线性。这些内容定义了我们想要学习的模型。还有一个问题是，如何学习模型或用来学习参数的算法，这一点由 solver 参数设定。solver 有两个好用的选项。默认选项是 'adam'，在大多数情况下效果都很好，但对数据的缩放相当敏感（因此，始终将数据缩放为均值为 0、方差为 1 是很重要的）。另一个选项是 'lbfgs'，其鲁棒性相当好，但在大型模型或大型数据集上的时间会比较长。还有更高级的 'sgd' 选项，许多深度学习研究人员都会用到。'sgd'选项还有许多其他参数需要调节，以便获得最佳结果。当你开始使用 MLP 时，我们建议使用 'adam' 和 'lbfgs'。

fit 会重置模型

scikit-learn 模型的一个重要性质就是，调用 fit 总会重置模型之前学到的所有内容。因此，如果你在一个数据集上构建模型，然后在另一个数据集上再次调用 fit，那么模型会“忘记”从第一个数据集中学到的所有内容。你可以对一个模型多次调用 fit，其结果与在“新”模型上调用 fit 是完全相同的。

1.4 分类器的不确定度估计

我们还没有谈到 scikit-learn 接口的另一个有用之处，就是分类器能够给出预测的不确定度估计。一般来说，你感兴趣的不仅是分类器会预测一个测试点属于哪个类别，还包括它对这个预测的置信程度。在实践中，不同类型的错误会在现实应用中导致非常不同的结果。想象一个用于测试癌症的医疗应用。假阳性预测可能只会让患者接受额外的测试，但假阴性预测却可能导致重病没有得到治疗。

scikit-learn 中有两个函数可用于获取分类器的不确定度估计：`decision_function` 和 `predict_proba`。大多数分类器（但不是全部）都至少有其中一个函数，很多分类器两个都有。

1.4.1 决策函数

1.4.2 预测概率

`predict_proba` 的输出是每个类别的概率，通常比 `decision_function` 的输出更容易理解。对于二分类问题，它的形状始终是 `(n_samples, 2)`。

在上一个输出中可以看到，分类器对大部分点的置信程度都是相对较高的。不确定度大小实际上反映了数据依赖于模型和参数的不确定度。过拟合更强的模型可能会做出置信程度更高的预测，即使可能是错的。复杂度越低的模型通常对预测的不确定度越大。如果模型给出的不确定度符合实际情况，那么这个模型被称为**校正**（calibrated）模型。在校正模型中，如果预测有 70% 的确定度，那么它在 70% 的情况下正确。

1.4.3 多分类问题的不确定度

1.5 小结与展望

- 最近邻：适用于小型数据集，是很好的基准模型，很容易解释。
- 线性模型：非常可靠的首选算法，适用于非常大的数据集，也适用于高维数据。
- 朴素贝叶斯：只适用于分类问题。比线性模型速度还快，适用于非常大的数据集和高维数据。精度通常要低于线性模型。
- 决策树：速度很快，不需要数据缩放，可以可视化，很容易解释。
- 随机森林：几乎总是比单棵决策树的表现要好，鲁棒性很好，非常强大。不需要数据缩放。不适用于高维稀疏数据。

- 梯度提升决策树：精度通常比随机森林略高。与随机森林相比，训练速度更慢，但预测速度更快，需要的内存也更少。比随机森林需要更多的参数调节。
- 支持向量机：对于特征含义相似的中等大小的数据集很强大。需要数据缩放，对参数敏感。
- 神经网络：可以构建非常复杂的模型，特别是对于大型数据集而言。对数据缩放敏感，对参数选取敏感。大型网络需要很长的训练时间。

面对新数据集，通常最好先从简单模型开始，比如线性模型、朴素贝叶斯或最近邻分类器，看能得到什么样的结果。对数据有了进一步了解之后，你可以考虑用于构建更复杂模型的算法，比如随机森林、梯度提升决策树、SVM 或神经网络。

Chapter 2

无监督学习与预处理

无监督学习包括没有已知输出、没有老师指导学习算法的各种机器学习。在无监督学习中，学习算法只有输入数据，并需要从这些数据中提取知识。

2.1 无监督学习的类型

这里将研究两种类型的无监督学习：数据集变换与聚类。

数据集的**无监督变换**（unsupervised transformation）是创建数据新的表示的算法，与数据的原始表示相比，新的表示可能更容易被人或其他机器学习算法所理解。无监督变换的一个常见应用是降维（dimensionality reduction），它接受包含许多特征的数据的高维表示，并找到表示该数据的一种新方法，用较少的特征就可以概括其重要特性。降维的一个常见应用是为了可视化将数据降为二维。

无监督变换的另一个应用是找到“构成”数据的各个组成部分。这方面的一个例子就是对文本文档集合进行主题提取。

聚类算法（clustering algorithm）将数据划分成不同的组，每组包含相似的物项。

2.2 无监督学习的挑战

无监督学习的一个主要挑战就是评估算法是否学到了有用的东西。无监督学习算法一般用于不包含任何标签信息的数据，所以我们不知道正确的输出应该是什么。因此很难判断一个模型是否“表现很好”。通常来说，评估无监督算法结果的唯一方法就是人工检查。

因此，如果数据科学家想要更好地理解数据，那么无监督算法通常可用于探索性的目的，而不是作为大型自动化系统的一部分。无监督算法的另一个常见应用是作为监督算法的预处理步骤。学习数据的一种新表示，有时可以提高监督算法的精度，或者可以减少内存占用和时间开销。

虽然预处理和缩放通常与监督学习算法一起使用，但缩放方法并没有用到与“监督”有关的信息，所以它是无监督的。

2.3 预处理与缩放

一些算法（如神经网络和 SVM）对数据缩放非常敏感。因此，通常的做法是对特征进行调节，使数据表示更适合于这些算法。通常来说，这是对数据的一种简单的按特征的缩放和移动。Figure 2.1 给出了一个简单的例子：

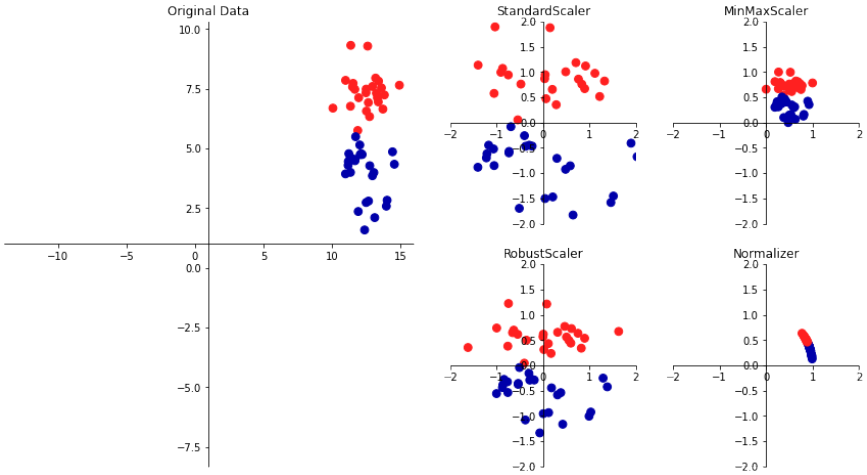

Figure 2.1: Different ways to rescale and preprocess a dataset

2.3.1 不同类型的预处理

Table 2.1: 一些缩放方式

缩放方式	作用位置	描述
StandardScaler	特征	确保每个特征的平均值为 0、方差为 1，使所有特征都位于同一量级。但这种缩放不能保证特征任何特定的最大值和最小值。
RobustScaler	特征	与StandardScaler 类似，确保每个特征的统计属性都位于同一范围。但 RobustScaler 使用的是中位数和四分位数 1，而不是平均值和方差。这样 RobustScaler 会忽略与其他点有很大不同的数据点（比如测量误差）。这些与众不同的数据点也叫异常值（outlier），可能会给其他缩放方法造成麻烦。
MinMaxScaler	特征	移动数据，使所有特征都刚好位于 0 到 1 之间。
Normalizer	样本	它对每个数据点进行缩放，使得特征向量的欧式长度等于 1。换句话说，它将一个数据点投射到半径为 1 的圆上（对于更高维度的情况，是球面）。这意味着每个数据点的缩放比例都不相同（乘以其长度的倒数）。如果只有数据的方向（或角度）是重要的，而特征向量的长度无关紧要，那么通常会使用这种归一化。

2.3.2 应用数据变换

常在应用监督学习算法之前使用预处理方法（比如缩放）。首先加载数据集并将其分为训练集和测试集（我们需要分开的训练集和数据集来对预处理后构建的监督模型进行评估）。

与之前构建的监督模型一样，我们首先导入实现预处理的类，然后将其实例化，然后，使用 fit 方法拟合缩放器（scaler），并将其应用于训练数据。对于 MinMaxScaler 来说，fit 方法计算训练集中每个特征的最大值和最小值。在对缩放器调用 fit 时只提供了 `X_train`，而不用 `y_train`。为了应用刚刚学习的变换（即对训练数据进行实际缩放），我们使用缩放器的 transform方法。在 scikit-learn 中，每当模型返回数据的一种新表示时，都可以使用 transform 方法。变换后的数据形状与原始数据相同，特征只

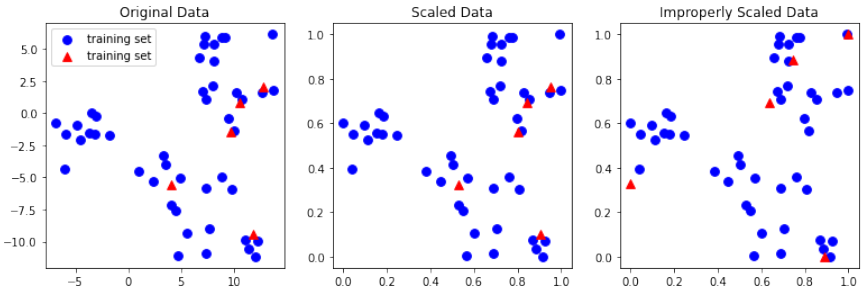

Figure 2.2: Effect of scaling training and test data

是发生了移动和缩放。

除了对训练数据做变换，还需要对测试集进行变换。

你可以发现，对测试集缩放后的最大值和最小值不是 1 和 0，这或许有些出乎意料。有些特征甚至在 0 1 的范围之外！对此的解释是，MinMaxScaler（以及其他所有缩放器）总是对训练集和测试集应用完全相同的变换。也就是说，transform 方法总是减去训练集的最小值，然后除以训练集的范围，而这两个值可能与测试集的最小值和范围并不相同。这个是显然的，因为 scaler 在 fit 的时候使用的训练数据

2.3.3 对训练数据和测试数据进行相同的缩放

在Figure 2.2中，第一张图是未缩放的二维数据集，其中训练集用圆形表示，测试集用三角形表示。第二张图中是同样的数据，但使用 MinMaxScaler 缩放。这里我们调用 fit 作用在训练集上，然后调用 transform 作用在训练集和测试集上。你可以发现，第二张图中的数据集看起来与第一张图中的完全相同，只是坐标轴刻度发生了变化。现在所有特征都位于 0 到 1 之间。你还可以发现，测试数据（三角形）的特征最大值和最小值并不是 1 和 0。

第三张图展示了如果我们对训练集和测试集分别进行缩放会发生什么。在这种情况下，对训练集和测试集而言，特征的最大值和最小值都是 1 和 0。但现在数据集看起来不一样。测试集相对训练集的移动不一致，因为它们分别做了不同的缩放。我们随意改变了数据的排列。这显然不是我们想要做的事情。

再换一种思考方式，想象你的测试集只有一个点。对于一个点而言，无法将其正确地缩放以满足 MinMaxScaler 的最大值和最小值的要求。但是，测试集的大小不应该对你的处理方式有影响。

快捷方式与高效的替代方法

通常来说，你想要在某个数据集上 fit 一个模型，然后再将其 transform。这是一个非常常见的任务，通常可以用比先调用 fit 再调用 transform 更高效的方法来计算。对于这种使用场景，所有具有 transform 方法的模型也都具有一个 `fit_transform` 方法。

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit(X).transform(X)
X_scaled_d = scaler.fit_transform(X_train)
y_scaled_d = scaler.transform(X_test)
```

虽然 `fit_transform` 不一定对所有模型都更加高效，但在尝试变换训练集时，使用这一方法仍然是很好的做法。

2.3.4 预处理对监督学习的作用

我们回到 cancer 数据集，观察使用 MinMaxScaler 对学习 SVC 的作用。首先，为了对比，我们再次在原始数据上拟合 SVC。

正如我们上面所见，数据缩放的作用非常显著。虽然数据缩放不涉及任何复杂的数学，但良好的做法仍然是使用 scikit-learn 提供的缩放机制，而不是自己重新实现它们，因为即使在这些简单的计算中也容易犯错。

你也可以通过改变使用的类将一种预处理算法轻松替换成另一种，因为所有的预处理类都具有相同的接口，都包含 fit 和 transform 方法

2.4 降维、特征提取与流形学习

利用无监督学习进行数据变换可能有很多种目的。最常见的目的就是可视化、压缩数据，以及寻找信息量更大的数据表示以用于进一步的处理。

为了实现这些目的，最简单也最常用的一种算法就是主成分分析。也将介绍另外两种算法：非负矩阵分解（NMF）和 t-SNE，前者通常用于特征提取，后者通常用于二维散点图的可视化。

2.4.1 主成分分析

主成分分析（principal component analysis，PCA）是一种旋转数据集的方法，旋转后的特征在统计上不相关。在做完这种旋转之后，通常是根据新特征对解释数据的重要性来选择它的一个子集。下面的例子Figure 2.3展示了 PCA 对一个模拟二维数据集的作用。

在Figure 2.3中，第一张图（左上）显示的是原始数据点，用不同颜色加以区分。算法首先找到方差最大的方向，将其标记为“成分 1”（Component 1）。这是数据中包含最多信息的方向（或向量），换句话说，沿着这个方向的特征之间最为相关。然后，算法找到与第一个方向正交（成直角）且包含最多信息的方向。在二维空间中，只有一个成直角的方向，但在更高维的空间中会有（无穷）多的正交方向。虽然这两个成分都画成箭头，但其头尾的位置并不重要。我们也可以将第一个成分画成从中心指向左上，而不是指向右下。利用这一过程找到的方向被称为主成分（principal component），因为它们是数据方差的主要方向。一般来说，主成分的个数与原始特征相同。

第二张图（右上）显示的是同样的数据，但现在将其旋转，使得第一主成分与 x 轴平行且第二主成分与 y 轴平行。在旋转之前，从数据中减去平均值，使得变换后的数据以零为中心。在 PCA 找到的旋转表示中，两个坐标轴是不相关的，也就是说，对于这种数据表示，除了对角线，相关矩阵全部为零。

我们可以通过仅保留一部分主成分来使用 PCA 进行降维。在这个例子中，我们可以仅保留第一个主成分，正如图 3-3 中第三张图所示（左下）。这将数据从二维数据集降为一维数据集。但要注意，我们没有保留原始特征之一，而是找到了最有趣的方向（第一张图中从左上到右下）并保留这一方向，即第一主成分。

最后，我们可以反向旋转并将平均值重新加到数据中。这样会得到图 3-3 最后一张图中的数据。这些数据点位于原始特征空间中，但我们仅保留了第一主成分中包含的信息。这种变换有时用于去除数据中的噪声影响，或者将主成分中保留的那部分信息可视化。

将 PCA 应用于 cancer 数据集并可视化

我们为每个特征创建一个直方图，计算具有某一特征的数据点在特定范围内（叫作bin）的出现频率。这样我们可以了解每个特征在两个类别中的分布情况，也可以猜测哪些特征能够更好地区分良性

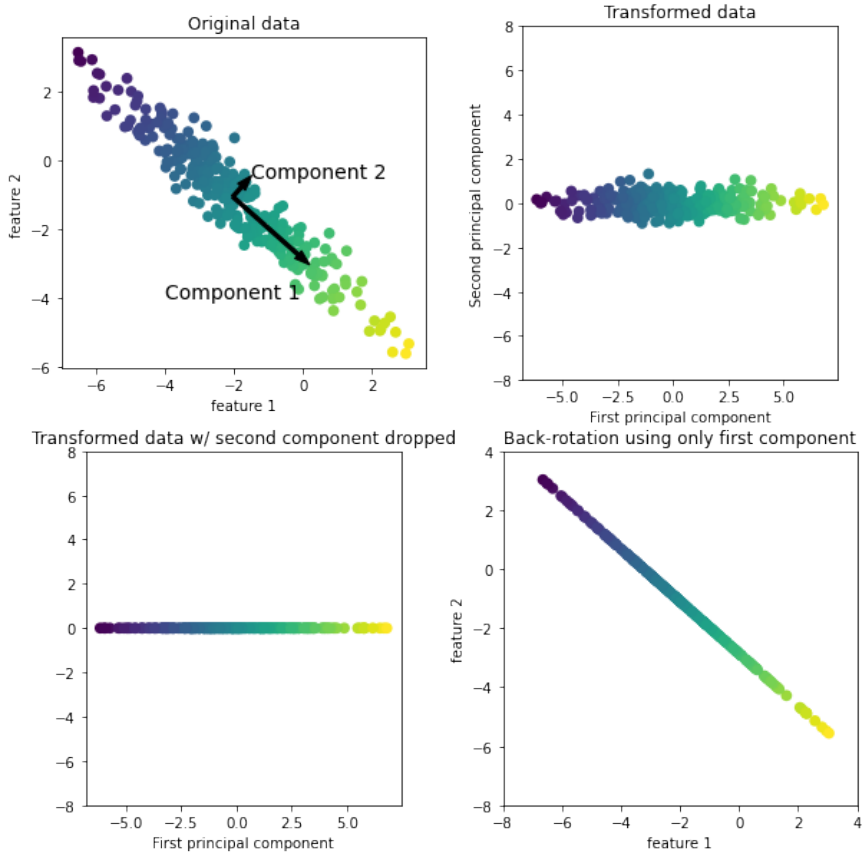

Figure 2.3: Transformation of data with PCA

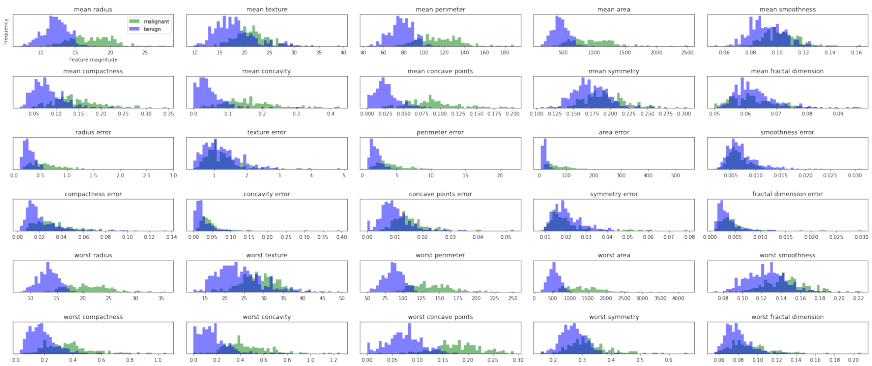

Figure 2.4: Per-class feature histograms on the Breast Cancer dataset

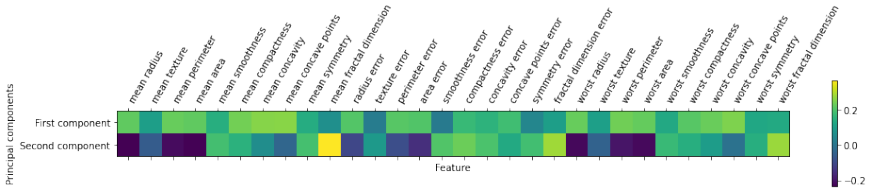

Figure 2.5: Heat map of the first two principal components on the Breast Cancer dataset

样本和恶性样本。例如，“smoothness error”特征似乎没有什么信息量，因为两个直方图大部分都重叠在一起，而“worst concave points”特征看起来信息量相当大，因为两个直方图的交集很小。

但是，这种图无法向我们展示变量之间的相互作用以及这种相互作用与类别之间的关系。利用 PCA，我们可以获取到主要的相互作用，并得到稍为完整的图像。我们可以找到前两个主成分，并在这个新的二维空间中用散点图将数据可视化。

学习并应用 PCA 变换与应用预处理变换一样简单。我们将 PCA 对象实例化，调用 fit 方法找到主成分，然后调用 transform 来旋转并降维。默认情况下，PCA 仅旋转（并移动）数据，但保留所有的主成分。为了降低数据的维度，我们需要在创建 PCA 对象时指定想要保留的主成分个数。

重要的是要注意，PCA 是一种无监督方法，在寻找旋转方向时没有用到任何类别信息。它只是观察数据中的相关性。对于这里所示的散点图，我们绘制了第一主成分与第二主成分的关系，然后利用类别信息对数据点进行着色。你可以看到，在这个二维空间中两个类别被很好地分离。这让我们相信，即使是线性分类器（在这个空间中学习一条直线）也可以在区分这个两个类别时表现得相当不错。

PCA 的一个缺点在于，通常不容易对图中的两个轴做出解释。主成分对应于原始数据中的方向，所以它们是原始特征的组合。但这些组合往往非常复杂。在拟合过程中，主成分被保存在 PCA 对象的 `components_` 属性中，`components_` 中的每一行对应于一个主成分，它们按重要性排序（第一主成分排在首位，以此类推）。列对应于 PCA 的原始特征属性。

在第一个主成分中，所有特征的符号相同（均为正，但前面我们提到过，箭头指向哪个方向无关紧要）。这意味着在所有特征之间存在普遍的相关性。。第二个主成分的符号有正有负，而且两个主成分都包含所有 30 个特征。这种所有特征的混合使得解释Figure 2.5中的坐标轴变得十分困难。

特征提取的特征脸

前面提到过，PCA 的另一个应用是特征提取。特征提取背后的思想是，可以找到一种数据表示，比给定的原始表示更适合于分析。特征提取很有用，它的一个很好的应用实例就是图像。图像由像素组成，通常存储为红绿蓝（RGB）强度。图像中的对象通常由上千个像素组成，它们只有放在一起才有意义。

我们得到的精度为 14.0%。对于包含 62 个类别的分类问题来说，这实际上不算太差（随机猜测的精度约为 1/62=1.5%），但也不算好。

这里就可以用到 PCA。想要度量人脸的相似度，计算原始像素空间中的距离是一种相当糟糕的方法。用像素表示来比较两张图像时，我们比较的是每个像素的灰度值与另一张图像对应位置的像素灰度值。这种表示与人们对人脸图像的解释方式有很大不同，使用这种原始表示很难获取到面部特征。例如，如果使用像素距离，那么将人脸向右移动一个像素将会发生巨大的变化，得到一个完全不同的表示。我们希望，使用沿着主成分方向的距离可以提高精度。这里我们启用 PCA 的白化（whitening）选项，它将主成分缩放到相同的尺度。变换后的结果与使用 StandardScaler 相同。

虽然我们肯定无法理解这些成分的所有内容，但可以猜测一些主成分捕捉到了人脸图像的哪些方

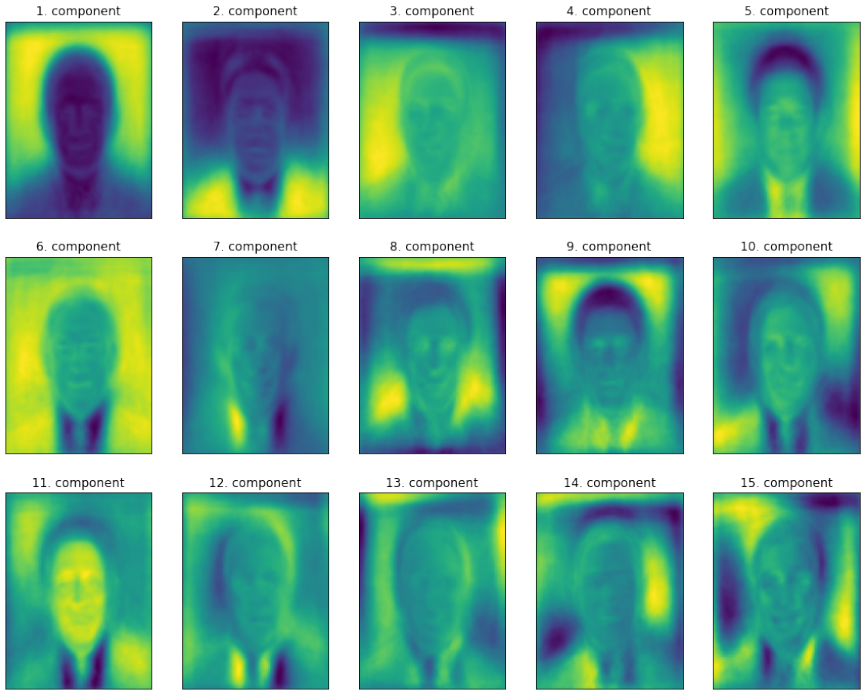

Figure 2.6: Component vectors of the first 15 principal components of the faces dataset

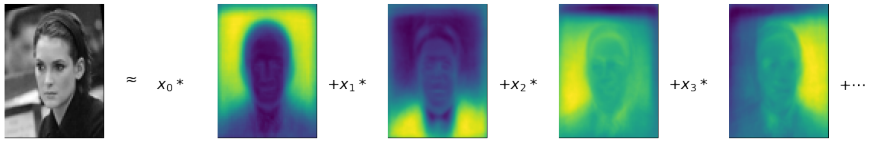

Figure 2.7: Schematic view of PCA as decomposing an image into a weighted sum of components

面。第一个主成分似乎主要编码的是人脸与背景的对比，第二个主成分编码的是人脸左半部分和右半部分的明暗程度差异，如此等等。虽然这种表示比原始像素值的语义稍强，但它仍与人们感知人脸的方式相去甚远。由于 PCA 模型是基于像素的，因此人脸的相对位置（眼睛、下巴和鼻子的位置）和明暗程度都对两张图像在像素表示中的相似程度有很大影响。但人脸的相对位置和明暗程度可能并不是人们首先感知的内容。在要求人们评价人脸的相似度时，他们更可能会使用年龄、性别、面部表情和发型等属性，而这些属性很难从像素强度中推断出来。重要的是要记住，算法对数据（特别是视觉数据，比如人们非常熟悉的图像）的解释通常与人类的解释方式大不相同。

我们对 PCA 变换的介绍是：先旋转数据，然后删除方差较小的成分。另一种有用的解释是尝试找到一些数字（PCA 旋转后的新特征值），使我们可以将测试点表示为主成分的加权求和（见Figure 2.7）。

这里 x_0、x_1 等是这个数据点的主成分的系数，换句话说，它们是图像在旋转后的空间中的表示。

我们还可以用另一种方法来理解 PCA 模型，就是仅使用一些成分对原始数据进行重建。在Figure 2.3 中，在去掉第二个成分并来到第三张图之后，我们反向旋转并重新加上平均值，这样就在原始空间中获得去掉第二个成分的新数据点，正如最后一张图所示。我们可以对人脸做类似的变换，将数据降维到只包含一些主成分，然后反向旋转回到原始空间。回到原始特征空间可以通过 `inverse_transform` 方法来实现。这里我们分别利用 10 个、50 个、100 个和 500 个成分对一些人脸进行重建并将其可视。

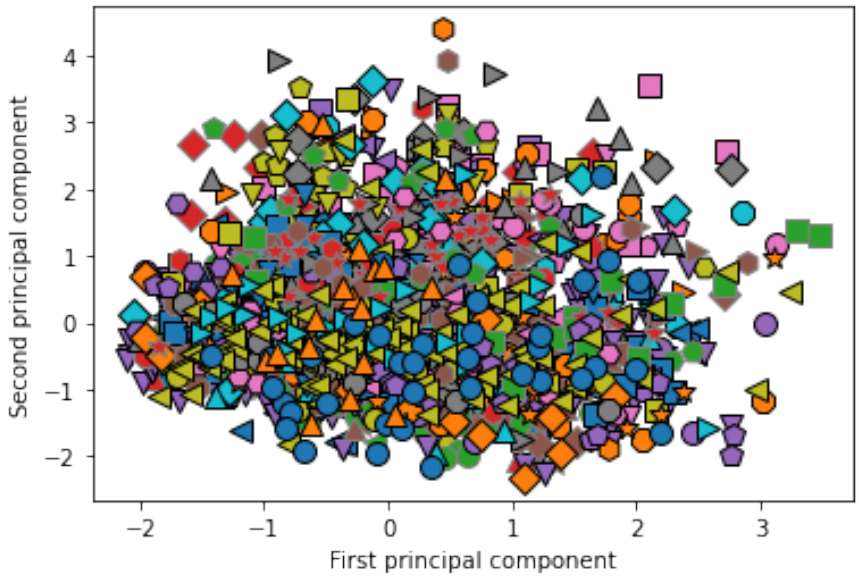

Figure 2.8: Scatter plot of the faces dataset using the first two principal components

在仅使用前 10 个主成分时，仅捕捉到了图片的基本特点，比如人脸方向和明暗程度。随着使用的主成分越来越多，图像中也保留了越来越多的细节。这对应于Figure 2.7的求和中包含越来越多的项。如果使用的成分个数与像素个数相等，意味着我们在旋转后不会丢弃任何信息，可以完美重建图像。

我们还可以尝试使用 PCA 的前两个主成分，将数据集中的所有人脸在散点图中可视化（Figure 2.8），其类别在图中给出。

2.4.2 非负矩阵分解

非负矩阵分解（non-negative matrix factorization，NMF）是另一种无监督学习算法，其目的在于提取有用的特征。它的工作原理类似于 PCA，也可以用于降维。与 PCA 相同，我们试图将每个数据点写成一些分量的加权求和。但在 PCA 中，我们想要的是正交分量，并且能够解释尽可能多的数据方差；而在 NMF 中，我们希望分量和系数均为非负，也就是说，我们希望分量和系数都大于或等于 0。因此，这种方法只能应用于每个特征都是非负的数据，因为非负分量的非负求和不可能变为负值。

将数据分解成非负加权求和的这个过程，对由多个独立源相加（或叠加）创建而成的数据特别有用，比如多人说话的音轨或包含多种乐器的音乐。在这种情况下，NMF 可以识别出组成合成数据的原始分量。总的来说，与 PCA 相比，NMF 得到的分量更容易解释，因为负的分量和系数可能会导致难以解释的抵消效应（cancellation effect）。举个例子，图3-9 中的特征脸同时包含正数和负数，我们在 PCA 的说明中也提到过，正负号实际上是任意的。

将 NMF 应用于模拟数据

与使用 PCA 不同，我们需要保证数据是正的，NMF 能够对数据进行操作。这说明数据相对于原点 (0,0) 的位置实际上对 NMF 很重要。因此，你可以将提取出来的非负分量看作是从 (0,0) 到数据的方向。

对于两个分量的 NMF（如左图所示），显然所有数据点都可以写成这两个分量的正数组合。如果有足够多的分量能够完美地重建数据（分量个数与特征个数相同），那么算法会选择指向数据极值的方向。

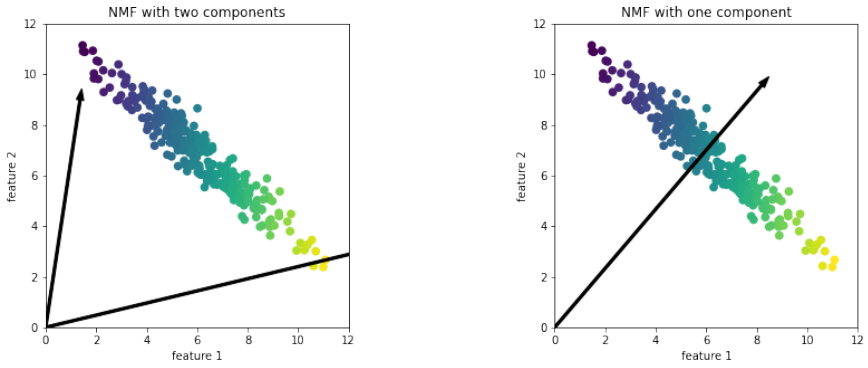

Figure 2.9: Components found by non-negative matrix factorization

如果我们仅使用一个分量，那么 NMF 会创建一个指向平均值的分量，因为指向这里可以对数据做出最好的解释。你可以看到，与 PCA 不同，减少分量个数不仅会删除一些方向，而且会创建一组完全不同的分量！ NMF 的分量也没有按任何特定方法排序，所以不存在“第一非负分量”：所有分量的地位平等。

NMF 使用了随机初始化，根据随机种子的不同可能会产生不同的结果。在相对简单的情况下（比如两个分量的模拟数据），所有数据都可以被完美地解释，那么随机性的影响很小（虽然可能会影响分量的顺序或尺度）。在更加复杂的情况下，影响可能会很大。

还有许多其他算法可用于将每个数据点分解为一系列固定分量的加权求和，正如 PCA 和NMF 所做的那样。讨论所有这些算法已超出了本书的范围，而且描述对分量和系数的约束通常要涉及概率论。如果你对这种类型的模式提取感兴趣，我们推荐你学习 scikit-learn 用户指南中关于独立成分分析（ICA）、因子分析（FA）和稀疏编码（字典学习）等的内容，所有这些内容都可以在关于分解方法的页面中找到。

2.4.3 用t-SNE进行流形学习

虽然 PCA 通常是用于变换数据的首选方法，使你能够用散点图将其可视化，但这一方法的性质（先旋转然后减少方向）限制了其有效性。有一类用于可视化的算法叫作**流形学习算法**（manifold learning algorithm），它允许进行更复杂的映射，通常也可以给出更好的可视化。其中特别有用的一个就是 t-SNE 算法。

流形学习算法主要用于可视化，因此很少用来生成两个以上的新特征。其中一些算法（包括 t-SNE）计算训练数据的一种新表示，但不允许变换新数据。这意味着这些算法不能用于测试集：更确切地说，它们只能变换用于训练的数据。流形学习对探索性数据分析是很有用的，但如果最终目标是监督学习的话，则很少使用。t-SNE 背后的思想是找到数据的一个二维表示，尽可能地保持数据点之间的距离。t-SNE 首先给出每个数据点的随机二维表示，然后尝试让在原始特征空间中距离较近的点更加靠近，原始特征空间中相距较远的点更加远离。t-SNE 重点关注距离较近的点，而不是保持距离较远的点之间的距离。换句话说，它试图保存那些表示哪些点比较靠近的信息。

t-SNE 的结果非常棒。所有类别都被明确分开。数字 1 和 9 被分成几块，但大多数类别都形成一个密集的组。要记住，这种方法并不知道类别标签：它完全是无监督的。但它能够找到数据的一种二维表示，仅根据原始空间中数据点之间的靠近程度就能够将各个类别明确分开。

t-SNE 算法有一些调节参数，虽然默认参数的效果通常就很好。你可以尝试修改 `perplexity` 和 `early_exaggeration`，但作用一般很小。

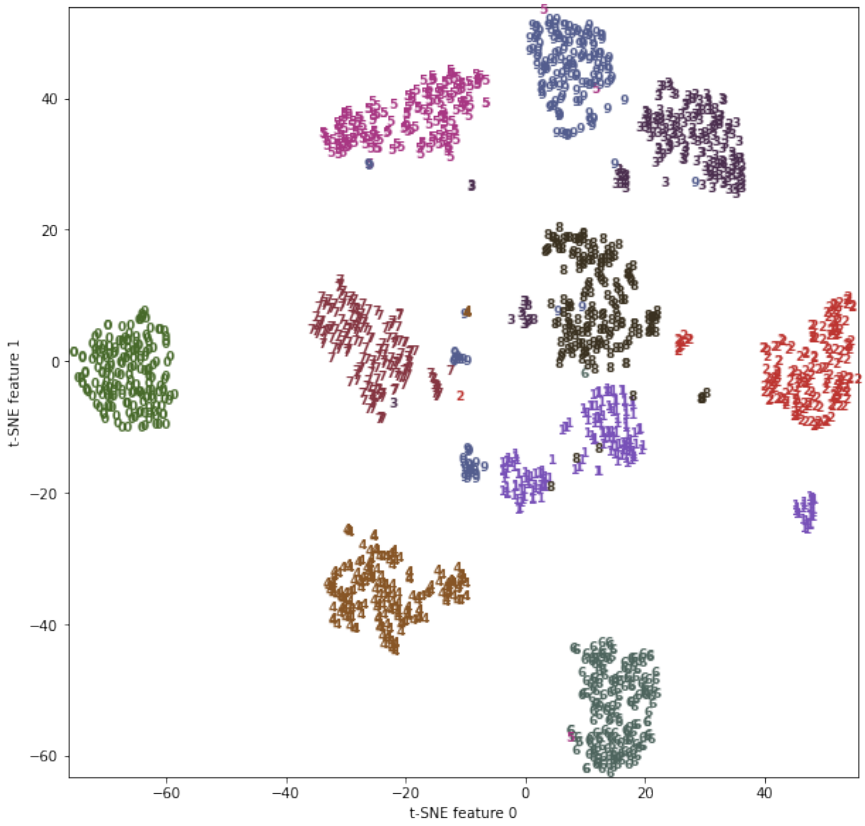

Figure 2.10: Scatter plot of the digits dataset using two components found by t-SNE

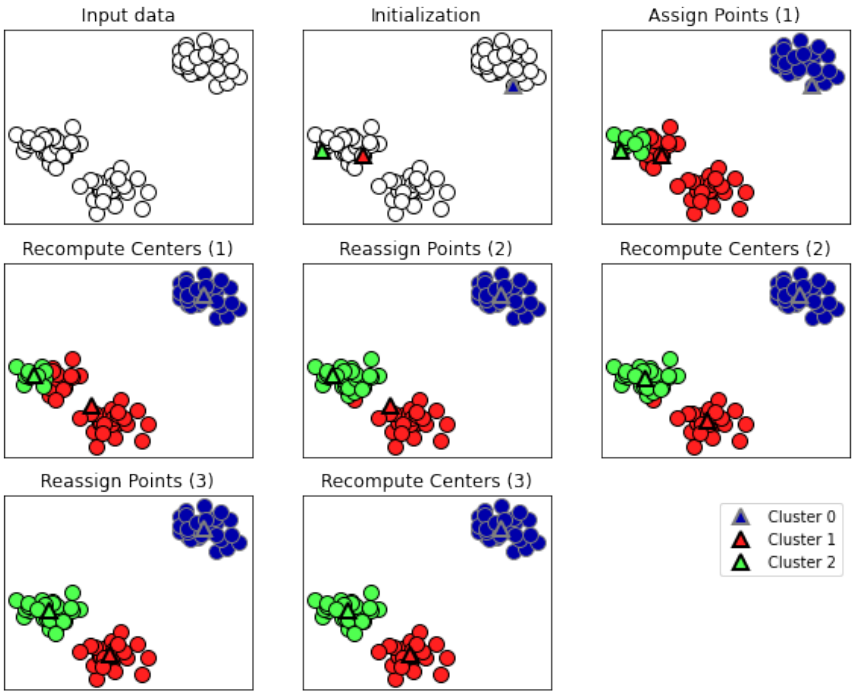

Figure 2.11: Input data and three steps of the k-means algorithm

2.5 聚类

聚类（clustering）是将数据集划分成组的任务，这些组叫作簇（cluster）。其目标是划分数据，使得一个簇内的数据点非常相似且不同簇内的数据点非常不同。与分类算法类似，聚类算法为每个数据点分配（或预测）一个数字，表示这个点属于哪个簇。

2.5.1 k均值聚类

k 均值聚类是最简单也最常用的聚类算法之一。它试图找到代表数据特定区域的簇中心（cluster center）。算法交替执行以下两个步骤：将每个数据点分配给最近的簇中心，然后将每个簇中心设置为所分配的所有数据点的平均值。如果簇的分配不再发生变化，那么算法结束。 scikit-learn 实现 k 均值相当简单。下面我们将其应用于上图中的模拟数据。我们将KMeans 类实例化，并设置我们要寻找的簇个数[1] 。然后对数据调用 fit 方法。

算法运行期间，为 X 中的每个训练数据点分配一个簇标签。你可以在 `kmeans.labels_` 属性中找到这些标签。

你也可以用 predict 方法为新数据点分配簇标签。预测时会将最近的簇中心分配给每个新数据点，但现有模型不会改变。聚类算法与分类算法有些相似，每个元素都有一个标签。但并不存在真实的标签，因此标签本身并没有先验意义。

对于刚刚在二维演示数据集上运行的聚类算法，这意味着我们不应该为其中一组的标签是 0、另一组的标签是 1 这一事实赋予任何意义。再次运行该算法可能会得到不同的簇编号，原因在于初始化的随机性质。

[1] 如果不指定，它的默认值是 8。使用这个值并没有什么特别的原因。

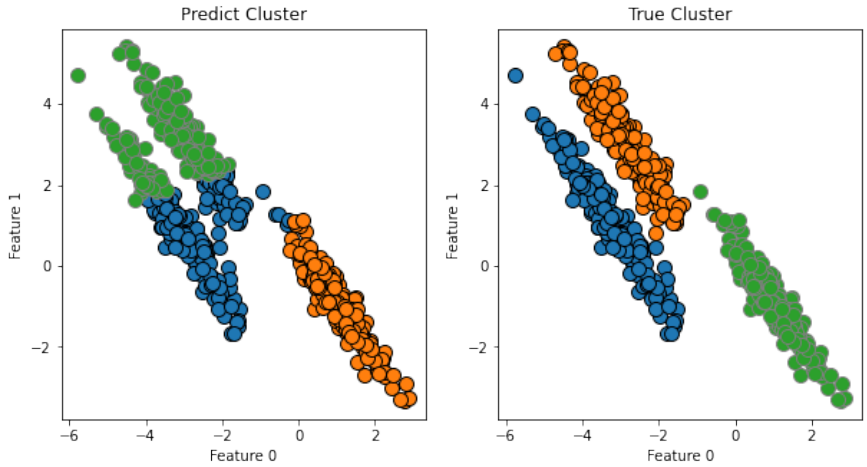

Figure 2.12: k-means fails to identify nonspherical clusters

k均值的失败案例

即使你知道给定数据集中簇的“正确”个数，k 均值可能也不是总能找到它们。每个簇仅由其中心定义，这意味着每个簇都是凸形（convex）。因此，k 均值只能找到相对简单的形状。k 均值还假设所有簇在某种程度上具有相同的“直径”，它总是将簇之间的边界刚好画在簇中心的中间位置。有时这会导致令人惊讶的结果。

k 均值还假设所有方向对每个簇都同等重要，见Figure 2.12。

如果簇的形状更加复杂，比如 `two_moons` 数据，那么 k 均值的表现也很差（见Figure 2.13）

这里我们希望聚类算法能够发现两个半月形。但利用 k 均值算法是不可能做到这一点的。

矢量量化，或者将k均值看作分解

虽然 k 均值是一种聚类算法，但在 k 均值和分解方法（比如之前讨论过的 PCA 和 NMF）之间存在一些有趣的相似之处。你可能还记得，PCA 试图找到数据中方差最大的方向，而NMF 试图找到累加的分量，这通常对应于数据的“极值”或“部分”。两种方法都试图将数据点表示为一些分量之和。与之相反，k 均值则尝试利用簇中心来表示每个数据点。你可以将其看作仅用一个分量来表示每个数据点，该分量由簇中心给出。这种观点将 k 均值看作是一种分解方法，其中每个点用单一分量来表示，这种观点被称为矢量量化（vector quantization）。

利用 k 均值做矢量量化的一个有趣之处在于，可以用比输入维度更多的簇来对数据进行编码。让我们回到 `two_moons` 数据。利用 PCA 或 NMF，我们对这个数据无能为力，因为它只有两个维度。使用 PCA 或 NMF 将其降到一维，将会完全破坏数据的结构。但通过使用更多的簇中心，我们可以用 k 均值找到一种更具表现力的表示（见Figure 2.14）。

我们使用了 10 个簇中心，也就是说，现在每个点都被分配了 0 到 9 之间的一个数字。我们可以将其看作 10 个分量表示的数据（我们有 10 个新特征），只有表示该点对应的簇中心的那个特征不为 0，其他特征均为 0。利用这个 10 维表示，现在可以用线性模型来划分两个半月形，而利用原始的两个特征是不可能做到这一点的。将到每个簇中心的距离作为特征，还可以得到一种表现力更强的数据表示。可以利用 kmeans 的 transform 方法来完成这一点。

k 均值是非常流行的聚类算法，因为它不仅相对容易理解和实现，而且运行速度也相对较快。k 均值可以轻松扩展到大型数据集，scikit-learn 甚至在 MiniBatchKMeans 类中包含了一种更具可扩展性的

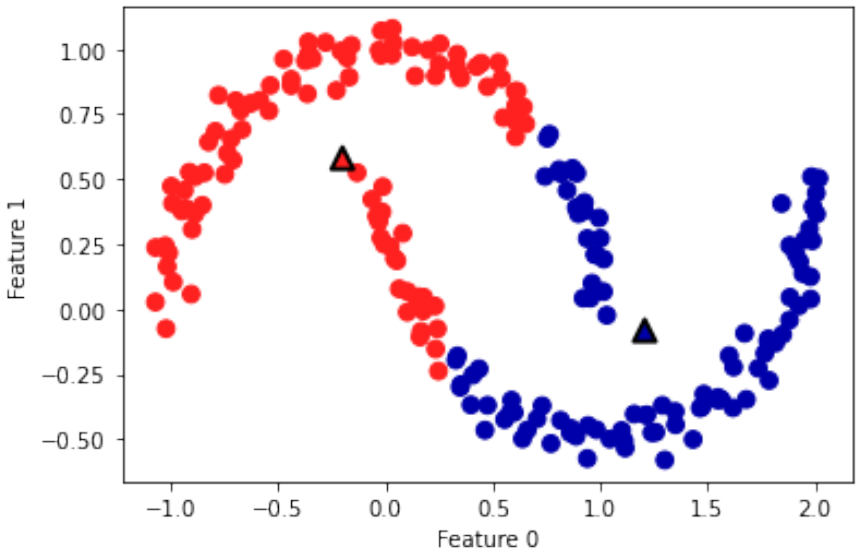

Figure 2.13: k-means fails to identify clusters with complex shapes

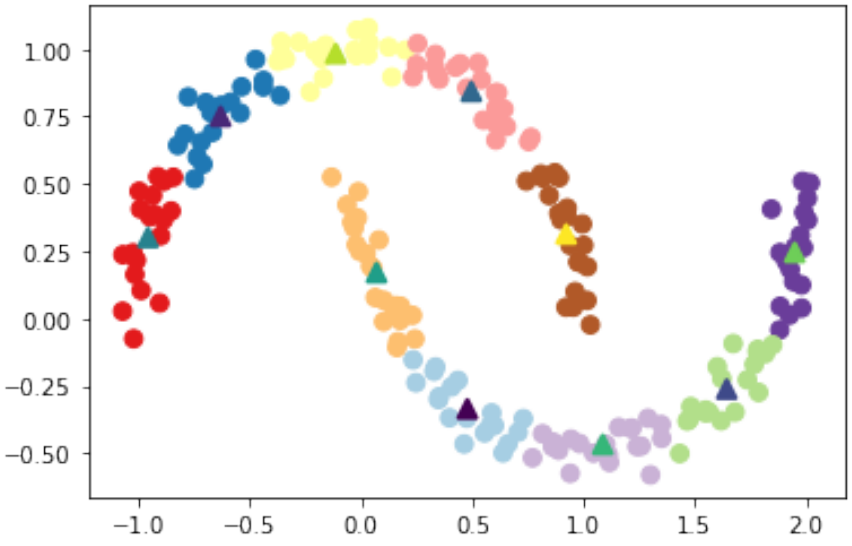

Figure 2.14: Using many k-means clusters to cover the variation in a complex dataset

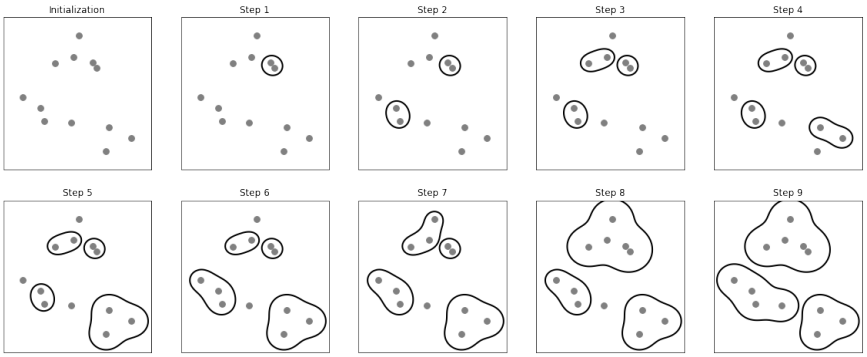

Figure 2.15: Agglomerative clustering iteratively joins the two closest clusters

变体，可以处理非常大的数据集。

k 均值的缺点之一在于，它依赖于随机初始化，也就是说，算法的输出依赖于随机种子。默认情况下，scikit-learn 用 10 种不同的随机初始化将算法运行 10 次，并返回最佳结果[2]。 k 均值还有一个缺点，就是对簇形状的假设的约束性较强，而且还要求指定所要寻找的簇的个数（在现实世界的应用中可能并不知道这个数字）。

2.5.2 凝聚聚类

凝聚聚类（agglomerative clustering）指的是许多基于相同原则构建的聚类算法，这一原则是：算法首先声明每个点是自己的簇，然后合并两个最相似的簇，直到满足某种停止准则为止。scikit-learn 中实现的停止准则是簇的个数，因此相似的簇被合并，直到仅剩下指定个数的簇。还有一些链接（linkage）准则，规定如何度量“最相似的簇”。这种度量总是定义在两个现有的簇之间。

scikit-learn 中实现了以下三种选项。

1. ward 默认选项。ward 挑选两个簇来合并，使得所有簇中的方差增加最小。这通常会得到大小差不多相等的簇。
2. average 链接将簇中所有点之间平均距离最小的两个簇合并。
3. complete 链接（也称为最大链接）将簇中点之间最大距离最小的两个簇合并。

ward 适用于大多数数据集，如果簇中的成员个数非常不同（比如其中一个比其他所有都大得多），那么 average 或 complete 可能效果更好。

由于算法的工作原理，凝聚算法不能对新数据点做出预测。因此 AgglomerativeClustering 没有 predict 方法。为了构造模型并得到训练集上簇的成员关系，可以改用 `fit_predict` 方法。结果如 Figure 2.16 所示。

虽然凝聚聚类的 scikit-learn 实现需要你指定希望算法找到的簇的个数，但凝聚聚类方法为选择正确的个数提供了一些帮助，我们将在下面讨论。

层次聚类与树状图

凝聚聚类生成了所谓的层次聚类（hierarchical clustering）。聚类过程迭代进行，每个点都从一个单点簇变为属于最终的某个簇。每个中间步骤都提供了数据的一种聚类（簇的个数也不相同）。有时候，同时查看所有可能的聚类是有帮助的。下一个例子（Figure 2.17）叠加显示了Figure 2.15 中所有可能的聚类，有助于深入了解每个簇如何分解为较小的簇。

[2]在这种情况下，“最佳”的意思是簇的方差之和最小。

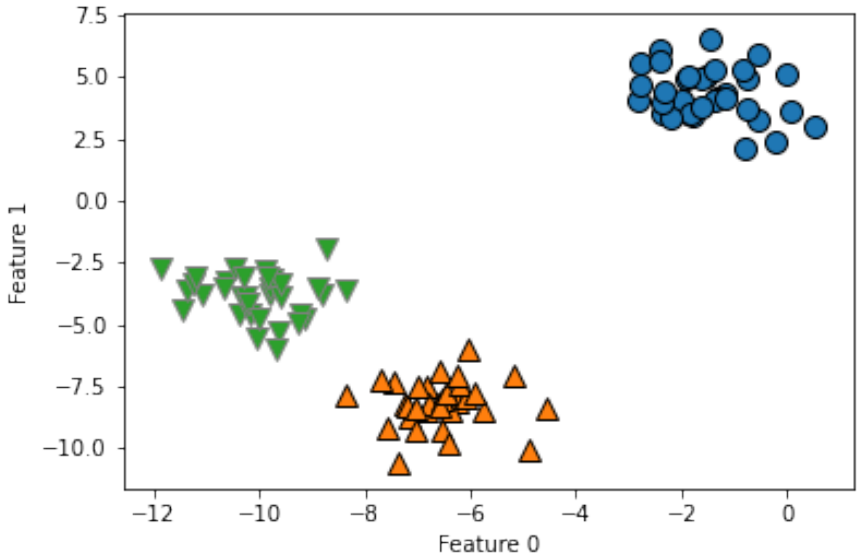

Figure 2.16: Cluster assignment using agglomerative clustering with three clusters

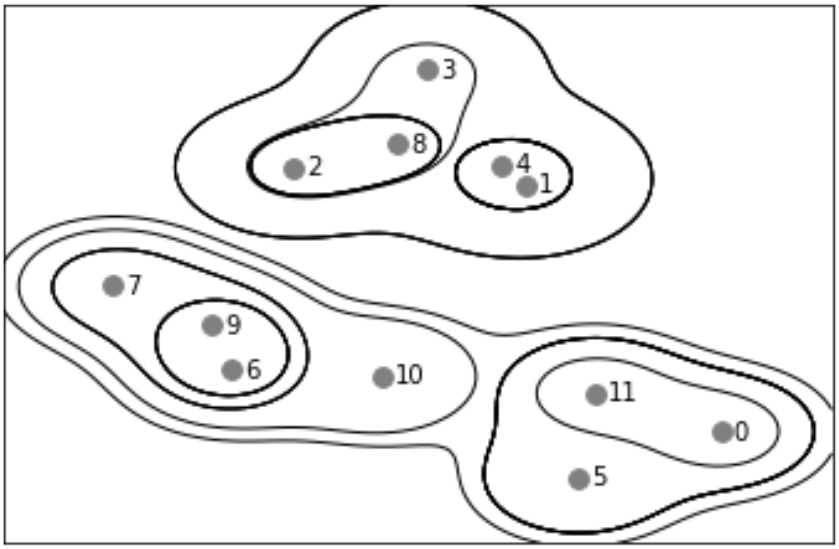

Figure 2.17: Hierarchical cluster assignment generated with agglomerative clustering

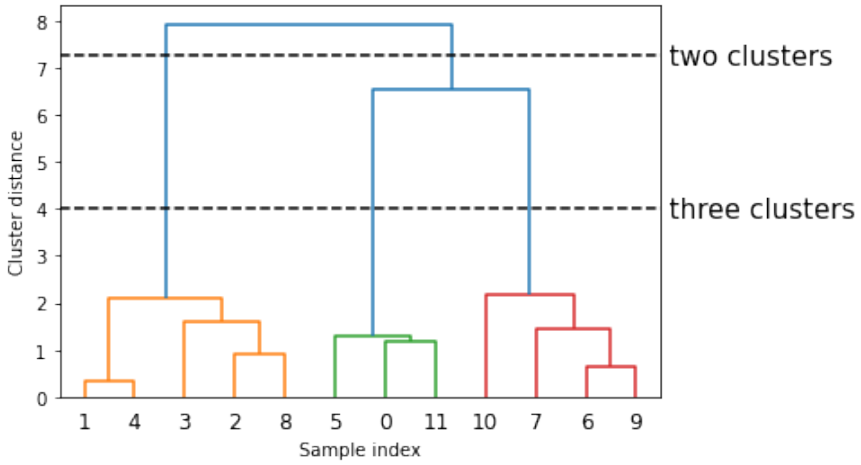

Figure 2.18: Dendrogram of the clustering

虽然这种可视化为层次聚类提供了非常详细的视图，但它依赖于数据的二维性质，因此不能用于具有两个以上特征的数据集。但还有另一个将层次聚类可视化的工具，叫作树状图（dendrogram），它可以处理多维数据集。

不幸的是，目前 scikit-learn 没有绘制树状图的功能。但你可以利用 SciPy 轻松生成树状图。SciPy 的聚类算法接口与 scikit-learn 的聚类算法稍有不同。SciPy 提供了一个函数，接受数据数组 X 并计算出一个链接数组（linkage array），它对层次聚类的相似度进行编码。然后我们可以将这个链接数组提供给 scipy 的 dendrogram 函数来绘制树状图（图 Figure 2.18）。

状图在底部显示数据点（编号从 0 到 11）。然后以这些点（表示单点簇）作为叶节点绘制一棵树，每合并两个簇就添加一个新的父节点。

从下往上看，数据点 1 和 4 首先被合并。接下来，点 6 和 9 被合并为一个簇，以此类推。在顶层有两个分支，一个由点 11、0、5、10、7、6 和 9 组成，另一个由点 1、4、3、2 和 8 组成。这对应于图中左侧两个最大的簇。

树状图的 y 轴不仅说明凝聚算法中两个簇何时合并，每个分支的长度还表示被合并的簇之间的距离。在这张树状图中，最长的分支是用标记为“three clusters”（三个簇）的虚线表示的三条线。它们是最长的分支，这表示从三个簇到两个簇的过程中合并了一些距离非常远的点。我们在图像上方再次看到这一点，将剩下的两个簇合并为一个簇也需要跨越相对较大的距离。

不幸的是，凝聚聚类仍然无法分离像 `two_moons` 数据集这样复杂的形状。但我们要学习的下一个算法 DBSCAN 可以解决这个问题。

2.5.3 DBSCAN

另一个非常有用的聚类算法是 DBSCAN（density-based spatial clustering of applications with noise，即“具有噪声的基于密度的空间聚类应用”）。DBSCAN 的主要优点是它不需要用户先验地设置簇的个数，可以划分具有复杂形状的簇，还可以找出不属于任何簇的点。DBSCAN 比凝聚聚类和 k 均值稍慢，但仍可以扩展到相对较大的数据集。

DBSCAN 的原理是识别特征空间的“拥挤”区域中的点，在这些区域中许多数据点靠近在一起。这些区域被称为特征空间中的密集（dense）区域。DBSCAN 背后的思想是，簇形成数据的密集区域，并由相对较空的区域分隔开。在密集区域内的点被称为核心样本（core sample，或核心点），它们的定

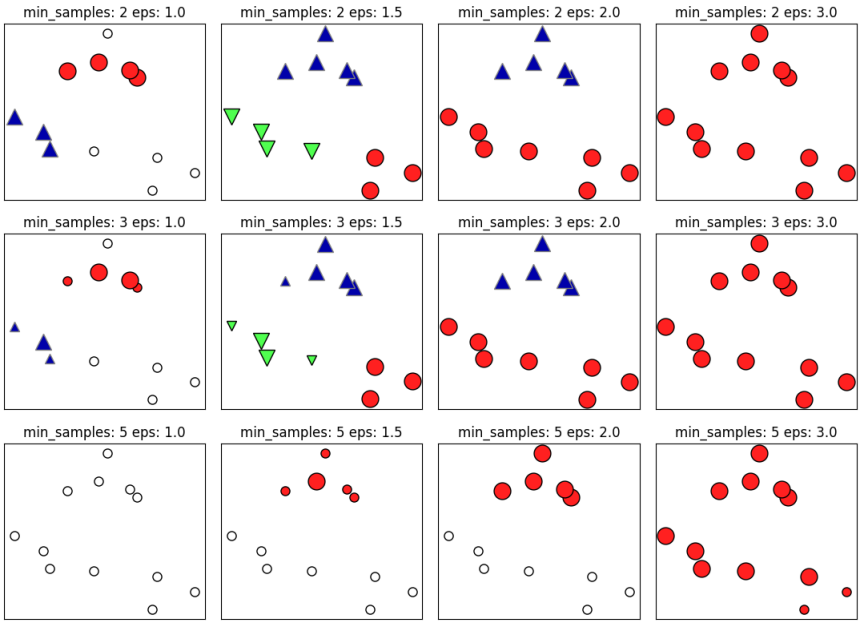

Figure 2.19: Cluster assignments found by DBSCAN

义如下。DBSCAN有两个参数：`min_samples` 和 eps。如果在距一个给定数据点 eps 的距离内至少有 `min_samples` 个数据点，那么这个数据点就是核心样本。DBSCAN 将彼此距离小于 eps 的核心样本放到同一个簇中。

> 算法首先任意选取一个点，然后找到到这个点的距离小于等于 eps 的所有的点。如果距起始点的距离在 eps 之内的数据点个数小于 `min_samples`，那么这个点被标记为噪声（noise），也就是说它不属于任何簇。如果距离在 eps 之内的数据点个数大于 `min_samples`，则这个点被标记为核心样本，并被分配一个新的簇标签。然后访问该点的所有邻居（在距离 eps 以内）。如果它们还没有被分配一个簇，那么就将刚刚创建的新的簇标签分配给它们。如果它们是核心样本，那么就依次访问其邻居，以此类推。簇逐渐增大，直到在簇的 eps 距离内没有更多的核心样本为止。然后选取另一个尚未被访问过的点，并重复相同的过程。

最后，一共有三种类型的点：核心点、与核心点的距离在 eps 之内的点（叫作边界点，boundary point）和噪声。如果 DBSCAN 算法在特定数据集上多次运行，那么核心点的聚类始终相同，同样的点也始终被标记为噪声。但边界点可能与不止一个簇的核心样本相邻。因此，边界点所属的簇依赖于数据点的访问顺序。一般来说只有很少的边界点，这种对访问顺序的轻度依赖并不重要。

我们将 DBSCAN 应用于演示凝聚聚类的模拟数据集。与凝聚聚类类似，DBSCAN 也不允许对新的测试数据进行预测，所以我们将使用 `fit_predict` 方法来执行聚类并返回簇标签。

Figure 2.19，属于簇的点是实心的，而噪声点则显示为空心的。核心样本显示为较大的标记，而边界点则显示为较小的标记。增大 eps（在图中从左到右），更多的点会被包含在一个簇中。这让簇变大，但可能也会导致多个簇合并成一个。增大 `min_samples`（在图中从上到下），核心点会变得更少，更多的点被标记为噪声。参数 eps 在某种程度上更加重要，因为它决定了点与点之间“接近”的含义。将 eps 设置得非常小，意味着没有点是核心样本，可能会导致所有点都被标记为噪声。将 eps 设置得非常

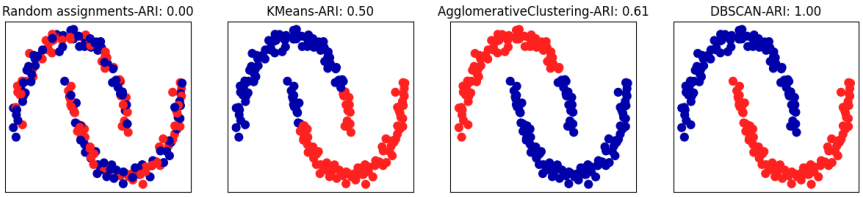

Figure 2.20: the supervised ARI score in four different cluster algorithm

大，可能会导致所有点形成单个簇。

设置 `min_samples` 主要是为了判断稀疏区域内的点被标记为异常值还是形成自己的簇。如果增大 `min_samples`，任何一个包含少于 `min_samples` 个样本的簇现在将被标记为噪声。因此，`min_samples` 决定簇的最小尺寸。在图 3-37 中 eps=1.5 时，从 `min_samples=3` 到 `min_samples=5`，你可以清楚地看到这一点。`min_samples=3` 时有三个簇：一个包含 4 个点，一个包含 5 个点，一个包含 3 个点。`min_samples=5` 时，两个较小的簇（分别包含 3 个点和 4个点）现在被标记为噪声，只保留包含 5 个样本的簇。虽然 DBSCAN 不需要显式地设置簇的个数，但设置 eps 可以隐式地控制找到的簇的个数。使用 StandardScaler 或 MinMaxScaler 对数据进行缩放之后，有时会更容易找到 eps 的较好取值，因为使用这些缩放技术将确保所有特征具有相似的范围。

2.5.4 聚类算法的对比与评估

在应用聚类算法时，其挑战之一就是很难评估一个算法的效果好坏，也很难比较不同算法的结果。

用真实值评估聚类

有一些指标可用于评估聚类算法相对于真实聚类的结果，其中最重要的是调整 rand 指数（*adjusted rand index, ARI*）和归一化互信息（*normalized mutual information, NMI*），二者都给出了定量的度量，其最佳值为 1，0 表示不相关的聚类（虽然 ARI 可以取负值）。

Figure 2.20 中，调整 rand 指数给出了符合直觉的结果，随机簇分配的分数为 0，而 DBSCAN（完美地找到了期望中的聚类）的分数为 1。

> 用这种方式评估聚类时，一个常见的错误是使用 `accuracy_score` 而不是 `adjusted_rand_score`、`normalized_mutual_info_score` 或其他聚类指标。使用精度的问题在于，它要求分配的簇标签与真实值完全匹配。但簇标签本身毫无意义——唯一重要的是哪些点位于同一个簇中。

在没有真实值的情况下评估聚类

我们刚刚展示了一种评估聚类算法的方法，但在实践中，使用诸如 ARI 之类的指标有一个很大的问题。在应用聚类算法时，通常没有真实值来比较结果。如果我们知道了数据的正确聚类，那么可以使用这一信息构建一个监督模型（比如分类器）。因此，使用类似 ARI和 NMI 的指标通常仅有助于开发算法，但对评估应用是否成功没有帮助。

有一些聚类的评分指标不需要真实值，比如**轮廓系数**（silhouette coeffcient）。但它们在实践中的效果并不好。轮廓分数计算一个簇的紧致度，其值越大越好，最高分数为 1。虽然紧致的簇很好，但紧致度不允许复杂的形状。

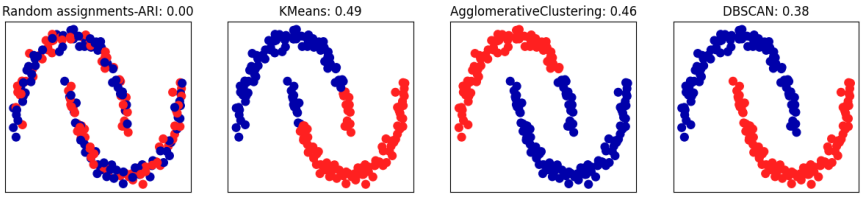

Figure 2.21: the unsupervised silhouette score

如你所见，k 均值的轮廓分数最高，尽管我们可能更喜欢 DBSCAN 的结果。对于评估聚类，稍好的策略是使用基于鲁棒性的（robustness-based）聚类指标。这种指标先向数据中添加一些噪声，或者使用不同的参数设定，然后运行算法，并对结果进行比较。其思想是，如果许多算法参数和许多数据扰动返回相同的结果，那么它很可能是可信的。

即使我们得到一个鲁棒性很好的聚类或者非常高的轮廓分数，但仍然不知道聚类中是否有任何语义含义，或者聚类是否反映了数据中我们感兴趣的某个方面。

回到人脸图像的例子。我们希望找到类似人脸的分组，比如男人和女人、老人和年轻人，或者有胡子的人和没胡子的人。假设我们将数据分为两个簇，关于哪些点应该被聚类在一起，所有算法的结果一致。我们仍不知道找到的簇是否以某种方式对应于我们感兴趣的概念。算法找到的可能是侧视图和正面视图、夜间拍摄的照片和白天拍摄的照片，或者 iPhone 拍摄的照片和安卓手机拍摄的照片。要想知道聚类是否对应于我们感兴趣的内容，唯一的办法就是对簇进行人工分析。

在人脸数据集上比较算法

这种类型的分析—— 尝试找出“奇怪的那一个” —— 被称为异常值检测（outlier detection）。

用 DBSCAN 分析人脸数据集 有趣的是，较大的簇从来没有超过一个。最多有一个较大的簇包含大多数点，还有一些较小的簇。这表示数据中没有两类或三类非常不同的人脸图像，而是所有图像或多或少地都与其他图像具有相同的相似度（或不相似度）。

用 k 均值分析人脸数据集 我们看到，利用 DBSCAN 无法创建多于一个较大的簇。凝聚聚类和 k 均值更可能创建均匀大小的簇，但我们需要设置簇的目标个数。我们可以将簇的数量设置为数据集中的已知人数(已知应该是62个不同的人的人脸)，虽然无监督聚类算法不太可能完全找到它们。相反，我们可以首先设置一个比较小的簇的数量

用凝聚聚类分析人脸数据集

2.5.5 聚类方法小结

聚类的应用与评估是一个非常定性的过程，通常在数据分析的探索阶段很有帮助。学习了三种聚类算法：k 均值、DBSCAN 和凝聚聚类。这三种算法都可以控制聚类的粒度（granularity）。k 均值和凝聚聚类允许你指定想要的簇的数量，而DBSCAN 允许你用 eps 参数定义接近程度，从而间接影响簇的大小。三种方法都可以用于大型的现实世界数据集，都相对容易理解，也都可以聚类成多个簇。

每种算法的优点稍有不同。k 均值可以用簇的平均值来表示簇。它还可以被看作一种分解方法，每个数据点都由其簇中心表示。DBSCAN 可以检测到没有分配任何簇的“噪声点”，还可以帮助自动判断簇的数量。与其他两种方法不同，它允许簇具有复杂的形状，正如我们在 `two_moons` 的例子中所看

到的那样。DBSCAN 有时会生成大小差别很大的簇，这可能是它的优点，也可能是缺点。凝聚聚类可以提供数据的可能划分的整个层次结构，可以通过树状图轻松查看。

2.6 小结与展望

找到数据的正确表示对于监督学习和无监督学习的成功通常都至关重要，预处理和分解方法在数据准备中具有重要作用。

分解、流形学习和聚类都是加深数据理解的重要工具，在没有监督信息的情况下，也是理解数据的仅有的方法。即使是在监督学习中，探索性工具对于更好地理解数据性质也很重要。通常来说，很难量化无监督算法的有用性，但这不应该妨碍你使用它们来深入理解数据。

估计器接口小结

scikit-learn 中的所有算法——无论是预处理、监督学习还是无监督学习算法——都被实现为类。这些类在 scikit-learn中叫作估计器（estimator）。估计器类包含算法，也保存了利用算法从数据中学到的模型。

为了应用算法，你首先需要将特定类的对象实例化，在构建模型对象时，你应该设置模型的所有参数。这些参数包括正则化、复杂度控制、要找到的簇的数量，等等。所有估计器都有 fit 方法，用于构建模型。fit 方法要求第一个参数总是数据 X，用一个 NumPy 数组或 SciPy 稀疏矩阵表示，其中每一行代表一个数据点。数据 X 总被假定为具有连续值（浮点数）的 NumPy 数组或 SciPy 稀疏矩阵。监督算法还需要有一个 y 参数，它是一维 NumPy 数组，包含回归或分类的目标值（即已知的输出标签或响应）。

在 scikit-learn 中，应用学到的模型主要有两种方法。要想创建一个新输出形式（比如 y）的预测，可以用 predict 方法。要想创建输入数据 X 的一种新表示，可以用 transform 方法。

Table 2.2: scikit-learn API summary

`estimator.fit(X_train, [y_train])`	
`estimator.predict(X_test)`	`estimator.transform(X_test)`
Classification	Preprocessing
Regression	Dimensionality Reduction
Clustering	Feature Extraction
	Feature Selection

此外，所有监督模型都有 `score(X_test, y_test)` 方法，可以评估模型。

Chapter 3

数据表示与特征工程

到目前为止，我们一直假设数据是由浮点数组成的二维数组，其中每一列是描述数据点的连续特征（continuous feature）。对于许多应用而言，数据的收集方式并不是这样。一种特别常见的特征类型就是分类特征（categorical feature），也叫离散特征（discrete feature）。这种特征通常并不是数值。分类特征与连续特征之间的区别类似于分类和回归之间的区别，只是前者在输入端而不是输出端。

无论你的数据包含哪种类型的特征，数据表示方式都会对机器学习模型的性能产生巨大影响。额外的特征扩充（augment）数据也很有帮助，比如添加特征的交互项（乘积）或更一般的多项式。

对于某个特定应用来说，如何找到最佳数据表示，这个问题被称为特征工程（feature engineering），它是数据科学家和机器学习从业者在尝试解决现实世界问题时的主要任务之一。用正确的方式表示数据，对监督模型性能的影响比所选择的精确参数还要大。

3.1 分类变量

作为例子，我们将使用美国成年人收入的数据集，该数据集是从 1994 年的普查数据库中导出的。

3.1.1 One-Hot编码（虚拟变量）

到目前为止，表示分类变量最常用的方法就是使用 one-hot 编码（one-hot-encoding）或 N 取一编码（one-out-of-N encoding），也叫虚拟变量（dummy variable）。虚拟变量背后的思想是将一个分类变量替换为一个或多个新特征，新特征取值为 0 和 1。对于线性二分类（以及 scikit-learn 中其他所有模型）的公式而言，0 和 1 这两个值是有意义的，我们可以像这样对每个类别引入一个新特征，从而表示任意数量的类别。

> 我们使用的 one-hot 编码与统计学中使用的虚拟编码（dummy encoding）非常相似，但并不完全相同。为简单起见，我们将每个类别编码为不同的二元特征。在统计学中，通常将具有 k 个可能取值的分类特征编码为 k-1 个特征（都等于零表示最后一个可能取值）。这么做是为了简化分析（更专业的说法是，这可以避免使数据矩阵秩亏）。

将数据转换为分类变量的 one-hot 编码有两种方法：一种是使用 pandas，一种是使用 scikit-learn。

检查字符串编码的分类数据

读取完这样的数据集之后，最好先检查每一列是否包含有意义的分类数据。在处理人工（比如网站用户）输入的数据时，可能没有固定的类别，拼写和大小写也存在差异，因此可能需要预处理。举

Table 3.1: The first few entries in the adult dataset

	age	workclass	education	gender	hours-per-week	occupation	income
0	39	State-gov	Bachelors	Male	40	Adm-clerical	<=50K
1	50	Self-emp-not-inc	Bachelors	Male	13	Exec-managerial	<=50K
2	38	Private	HS-grad	Male	40	Handlers-cleaners	<=50K
3	53	Private	11th	Male	40	Handlers-cleaners	<=50K
4	28	Private	Bachelors	Female	40	Prof-specialty	<=50K
5	37	Private	Masters	Female	40	Exec-managerial	<=50K
6	49	Private	9th	Female	16	Other-service	<=50K
7	52	Self-emp-not-inc	HS-grad	Male	45	Exec-managerial	>50K
8	31	Private	Masters	Female	50	Prof-specialty	>50K
9	42	Private	Bachelors	Male	40	Exec-managerial	>50K
10	37	Private	Some-college	Male	80	Exec-managerial	>50K

个例子，有人可能将性别填为“male”（男性），有人可能填为“man” （男人），而我们希望能用同一个类别来表示这两种输入。检查列的内容有一个好方法，就是使用 pandas Series（Series 是 DataFrame 中单列对应的数据类型）的 `value_counts` 函数，以显示唯一值及其出现次数。

在实际的应用中，你应该查看并检查所有列的值。

用 pandas 编码数据有一种非常简单的方法，就是使用 `get_dummies` 函数。`get_dummies` 函数自动变换所有具有对象类型（比如字符串）的列或所有分类的列（这是 pandas 中的一个特殊概念）。

将输出变量或输出变量的一些导出属性包含在特征表示中，这是构建监督机器学习模型时一个非常常见的错误。

注意：pandas 中的列索引包括范围的结尾，因此 `'age':'occupation_ Transport-moving'` 中包括 `occupation_ Transport-moving`。这与 NumPy 数组的切片不同，后者不包括范围的结尾，例如 `np.arange(11)[0:10]` 不包括索引编号为 10 的元素。

警告

在这个例子中，我们对同时包含训练数据和测试数据的数据框调用 `get_dummies`。这一点很重要，可以确保训练集和测试集中分类变量的表示方式相同。
如果训练集和测试集的数据字段不同，或者字段的未知排列不同，将导致严重的错误!!!

3.1.2 数字可以编码分类变量

在 adult 数据集的例子中，分类变量被编码为字符串。一方面，可能会有拼写错误；但另一方面，它明确地将一个变量标记为分类变量。无论是为了便于存储还是因为数据的收集方式，分类变量通常被编码为整数。例如，假设 adult 数据集中的人口普查数据是利用问卷收集的，workclass 的回答被记录为 0（在第一个框打勾）、1（在第二个框打勾）、2（在第三个框打勾），等等。现在该列包含数字 0 到 8，而不是像 ”Private” 这样的字符串。如果有人观察表示数据集的表格，很难一眼看出这个变量应该被视为连续变量还是分类变量。但是，如果知道这些数字表示的是就业状况，那么很明显它们是不

同的状态，不应该用单个连续变量来建模。

> 分类特征通常用整数进行编码。它们是数字并不意味着它们必须被视为连续特征。一个整数特征应该被视为连续的还是离散的（one-hot 编码的），有时并不明确。如果在被编码的语义之间没有顺序关系（比如 workclass 的例子），那么特征必须被视为离散特征。对于其他情况（比如五星评分），哪种编码更好取决于具体的任务和数据，以及使用哪种机器学习算法。

3.2 分箱、离散化、线性模型与树

数据表示的最佳方法不仅取决于数据的语义，还取决于所使用的模型种类。线性模型与基于树的模型（比如决策树、梯度提升树和随机森林）是两种成员很多同时又非常常用的模型，它们在处理不同的特征表示时就具有非常不同的性质。

正如你所知，线性模型只能对线性关系建模，对于单个特征的情况就是直线。决策树可以构建更为复杂的数据模型，但这强烈依赖于数据表示。有一种方法可以让线性模型在连续数据上变得更加强大，就是使用特征**分箱**（binning，也叫离散化，即 discretization）将其划分为多个特征，如下所述。

我们假设将特征的输入范围划分成固定个数的箱子（bin），比如 10 个，那么数据点就可以用它所在的箱子来表示。

为了确定这一点，我们首先需要定义箱子，可以用`np.linspace`函数来定义箱子。接下来，我们记录每个数据点所属的箱子。这可以用 np.digitize 函数轻松计算出来。要想在这个数据上使用 scikit-learn 模型，我们利用 preprocessing 模块的 OneHotEncoder 将这个离散特征变换为 one-hot 编码。OneHotEncoder 实现的编码与 `pandas.get_dummies` 相同，但目前它只适用于值为整数的分类变量。

虚线和实线完全重合，说明线性回归模型和决策树做出了完全相同的预测。对于每个箱子，二者都预测一个常数值。因为每个箱子内的特征是不变的，所以对于一个箱子内的所有点，任何模型都会预测相同的值。比较对特征进行分箱前后模型学到的内容，我们发现，线性模型变得更加灵活了，因为现在它对每个箱子具有不同的取值，而决策树模型的灵活性降低了。分箱特征对基于树的模型通常不会产生更好的效果，因为这种模型可以学习在任何位置划分数据。从某种意义上来看，决策树可以学习如何分箱对预测这些数据最为有用。此外，决策树可以同时查看多个特征，而分箱通常针对的是单个特征。不过，线性模型的表现力在数据变换后得到了极大的提高。

对于特定的数据集，如果有充分的理由使用线性模型——比如数据集很大、维度很高，但有些特征与输出的关系是非线性的——那么分箱是提高建模能力的好方法。

3.3 交互特征与多项式特征

想要丰富特征表示，特别是对于线性模型而言，另一种方法是添加原始数据的**交互特征**（interaction feature）和**多项式特征**（polynomial feature）。这种特征工程通常用于统计建模，但也常用于许多实际的机器学习应用中。们知道，线性模型不仅可以学习偏移，还可以学习斜率。想要向分箱数据上的线性模型添加斜率，一种方法是重新加入原始特征。这样的话模型在每个箱子中都学到一个偏移，还学到一个斜率。但是学到的斜率在所有箱子中都相同——只有一个 x 轴特征，也就只有一个斜率。因为斜率在所有箱子中是相同的，所以它似乎不是很有用。我们更希望每个箱子都有一个不同的斜率！为了实现这一点，我们可以添加交互特征或乘积特征，用来表示数据点所在的箱子以及数据点在 x 轴上的位置。这个特征是箱子指示符与原始特征的乘积。

你可以将乘积特征看作每个箱子 x 轴特征的单独副本。它在箱子内等于原始特征，在其他位置等于零。

使用分箱是扩展连续特征的一种方法。另一种方法是使用原始特征的多项式（polynomial）。对于给定特征 x，我们可以考虑 x^2、x^3、x^4，等等。这在 preprocessing 模块的 PolynomialFeatures 中实现。

将多项式特征与线性回归模型一起使用，可以得到经典的多项式回归（polynomial regression）模型。

如你所见，多项式特征在这个一维数据上得到了非常平滑的拟合。但高次多项式在边界上或数据很少的区域可能有极端的表现。

显然，在使用 Ridge 时，交互特征和多项式特征对性能有很大的提升。但如果使用更加复杂的模型（比如随机森林），情况会稍有不同。

3.4 单变量非线性变换

我们刚刚看到，添加特征的平方或立方可以改进线性回归模型。其他变换通常也对变换某些特征有用，特别是应用数学函数，比如 log、exp 或 sin。虽然基于树的模型只关注特征的顺序，但线性模型和神经网络依赖于每个特征的尺度和分布。如果在特征和目标之间存在非线性关系，那么建模就变得非常困难，特别是对于回归问题。log 和 exp 函数可以帮助调节数据的相对比例，从而改进线性模型或神经网络的学习效果。在处理具有周期性模式的数据时，sin 和 cos 函数非常有用。

大部分模型都在每个特征（在回归问题中还包括目标值）大致遵循高斯分布时表现最好。使用诸如 log和 exp 之类的变换并不稀奇，但却是实现这一点的简单又有效的方法。在一种特别常见的情况下，这样的变换非常有用，就是处理整数计数数据时。计数数据是指类似“用户 A 多长时间登录一次？”这样的特征。计数不可能取负值，并且通常遵循特定的统计模式。

这种类型的数值分布（许多较小的值和一些非常大的值）在实践中非常常见[1]。但大多数线性模型无法很好地处理这种数据。

为数据集和模型的所有组合寻找最佳变换，这在某种程度上是一门艺术。在这个例子中，所有特征都具有相同的性质，这在实践中是非常少见的情况。通常来说，只有一部分特征应该进行变换，有时每个特征的变换方式也各不相同。前面提到过，对基于树的模型而言，这种变换并不重要，但对线性模型来说可能至关重要。对回归的目标变量 y 进行变换有时也是一个好主意。尝试预测计数（比如订单数量）是一项相当常见的任务，而且使用 $log(y+1)$ 变换也往往有用。

从前面的例子中可以看出，分箱、多项式和交互项都对模型在给定数据集上的性能有很大影响，对于复杂度较低的模型更是这样，比如线性模型和朴素贝叶斯模型。与之相反，基于树的模型通常能够自己发现重要的交互项，大多数情况下不需要显式地变换数据。其他模型，比如 SVM、最近邻和神经网络，有时可能会从使用分箱、交互项或多项式中受益，但其效果通常不如线性模型那么明显。

3.5 自动化特征选择

有了这么多种创建新特征的方法，你可能会想要增大数据的维度，使其远大于原始特征的数量。但是，添加更多特征会使所有模型变得更加复杂，从而增大过拟合的可能性。在添加新特征或处理一般的高维数据集时，最好将特征的数量减少到只包含最有用的那些特征，并删除其余特征。这样会得到泛化能力更好、更简单的模型。但你如何判断每个特征的作用有多大呢？有三种基本的策略：**单变量统计**（univariate statistics）、**基于模型的选择**（model-based selection）和**迭代选择**（iterative selection）。

[1] 这是泊松分布，对计数数据相当重要。

3.5.1 单变量统计

在单变量统计中，我们计算每个特征和目标值之间的关系是否存在统计显著性，然后选择具有最高置信度的特征。对于分类问题，这也被称为**方差分析**（analysis of variance， ANOVA）。这些测试的一个关键性质就是它们是单变量的（univariate），即它们只单独考虑每个特征。因此，如果一个特征只有在与另一个特征合并时才具有信息量，那么这个特征将被舍弃。单变量测试的计算速度通常很快，并且不需要构建模型。另一方面，它们完全独立于你可能想要在特征选择之后应用的模型。

想要在 scikit-learn 中使用单变量特征选择，你需要选择一项测试——对分类问题通常是 `f_classif`（默认值），对回归问题通常是 `f_regression`——然后基于测试中确定的 p 值来选择一种舍弃特征的方法。所有舍弃参数的方法都使用阈值来舍弃所有 p 值过大的特征（意味着它们不可能与目标值相关，不能拒绝原假设）[2]。计算阈值的方法各有不同，最简单的是 SelectKBest 和 SelectPercentile，前者选择固定数量的 k 个特征，后者选择固定百分比的特征。

我们可以用 `get_support` 方法来查看哪些特征被选中，它会返回所选特征的布尔遮罩（mask）。

如果特征量太大以至于无法构建模型，或者你怀疑许多特征完全没有信息量，那么单变量特征选择还是非常有用的。

3.5.2 基于模型的特征选择

基于模型的特征选择使用一个监督机器学习模型来判断每个特征的重要性，并且仅保留最重要的特征。用于特征选择的监督模型不需要与用于最终监督建模的模型相同。特征选择模型需要为每个特征提供某种重要性度量，以便用这个度量对特征进行排序。决策树和基于决策树的模型提供了 `feature_importances_` 属性，可以直接编码每个特征的重要性。线性模型系数的绝对值也可以用于表示特征重要性。L1 惩罚的线性模型学到的是稀疏系数，它只用到了特征的一个很小的子集。这可以被视为模型本身的一种特征选择形式，但也可以用作另一个模型选择特征的预处理步骤。与单变量选择不同，基于模型的选择同时考虑所有特征，因此可以获取交互项（如果模型能够获取它们的话）。要想使用基于模型的特征选择，我们需要使用 SelectFromModel 变换器。SelectFromModel 类选出重要性度量（由监督模型提供）大于给定阈值的所有特征。

3.5.3 迭代特征选择

在单变量测试中，我们没有使用模型，而在基于模型的选择中，我们使用了单个模型来选择特征。在迭代特征选择中，将会构建一系列模型，每个模型都使用不同数量的特征。**有两种基本方法：开始时没有特征，然后逐个添加特征，直到满足某个终止条件；或者从所有特征开始，然后逐个删除特征，直到满足某个终止条件**。由于构建了一系列模型，所以这些方法的计算成本要比前面讨论过的方法更高。其中一种特殊方法是**递归特征消除** （recursive feature elimination，RFE），它从所有特征开始构建模型，并根据模型舍弃最不重要的特征，然后使用除被舍弃特征之外的所有特征来构建一个新模型，如此继续，直到仅剩下预设数量的特征。为了让这种方法能够运行，用于选择的模型需要提供某种确定特征重要性的方法，正如基于模型的选择所做的那样。

如果你不确定何时选择使用哪些特征作为机器学习算法的输入，那么自动化特征选择可能特别有用。它还有助于减少所需要的特征数量，加快预测速度，或允许可解释性更强的模型。在大多数现实情况下，使用特征选择不太可能大幅提升性能，但它仍是特征工程工具箱中一个非常有价值的工具。

[2] 原假设——无差异

3.6 利用专家知识

对于特定应用来说，在特征工程中通常可以利用专家知识（expert knowledge）。虽然在许多情况下，机器学习的目的是避免创建一组专家设计的规则，但这并不意味着应该舍弃该应用或该领域的先验知识。通常来说，领域专家可以帮助找出有用的特征，其信息量比数据原始表示要大得多。

在对这种时间序列上的预测任务进行评估时，我们通常希望从过去学习并预测未来。也就是说，在划分训练集和测试集的时候，我们希望使用某个特定日期之前的所有数据作为训练集，该日期之后的所有数据作为测试集。

LinearRegression 的效果差得多，而且周期性模式看起来很奇怪。其原因在于我们用整数编码一周的星期几和一天内的时间，它们被解释为连续变量。因此，线性模型只能学到关于每天时间的线性函数——它学到的是，时间越晚，租车数量越多。但实际模式比这要复杂得多。我们可以通过将整数解释为分类变量（用 OneHotEncoder 进行变换）来获取这种模式。

3.7 小结与展望

本章讨论了如何处理不同的数据类型（特别是分类变量）。我们强调了使用适合机器学习算法的数据表示方式的重要性，例如 one-hot 编码过的分类变量。还讨论了通过特征工程生成新特征的重要性，以及利用专家知识从数据中创建导出特征的可能性。特别是线性模型，可能会从分箱、添加多项式和交互项而生成的新特征中大大受益。对于更加复杂的非线性模型（比如随机森林和 SVM），在无需显式扩展特征空间的前提下就可以学习更加复杂的任务。在实践中，所使用的特征（以及特征与方法之间的匹配）通常是使机器学习方法表现良好的最重要的因素。

Chapter 4

模型评估与改进

为了评估我们的监督模型，我们使用 `train_test_split` 函数将数据集划分为训练集和测试集，在训练集上调用 fit 方法来构建模型，并且在测试集上用 score 方法来评估这个模型——对于分类问题而言，就是计算正确分类的样本所占的比例。

请记住，之所以将数据划分为训练集和测试集，是因为我们想要度量模型对前所未见的新数据的泛化性能。我们对模型在训练集上的拟合效果不感兴趣，而是想知道模型对于训练过程中没有见过的数据的预测能力。

本章我们将从两个方面进行模型评估。我们首先介绍交叉验证，然后讨论评估分类和回归性能的方法，其中前者是一种更可靠的评估泛化性能的方法，后者是在默认度量（score方法给出的精度和 R^2）之外的方法。

我们还将讨论网格搜索，这是一种调节监督模型参数以获得最佳泛化性能的有效方法。

4.1 交叉验证

交叉验证（cross-validation）是一种评估泛化性能的统计学方法，它比单次划分训练集和测试集的方法更加稳定、全面。在交叉验证中，数据被多次划分，并且需要训练多个模型。最常用的交叉验证是 k 折交叉验证（k-fold cross-validation），其中 k 是由用户指定的数字，通常取 5 或 10。在执行 5 折交叉验证时，首先将数据划分为（大致）相等的 5 部分，每一部分叫作折（fold）。接下来训练一系列模型。使用第 1 折作为测试集、其他折（2 5）作为训练集来训练第一个模型。利用 2 5 折中的数据来构建模型，然后在 1 折上评估精度。之后构建另一个模型，这次使用 2 折作为测试集，1、3、4、5 折中的数据作为训练集。利用 3、4、5 折作为测试集继续重复这一过程。对于将数据划分为训练集和测试集的这 5 次划分，每一次都要计算精度。最后我们得到了 5 个精度值。

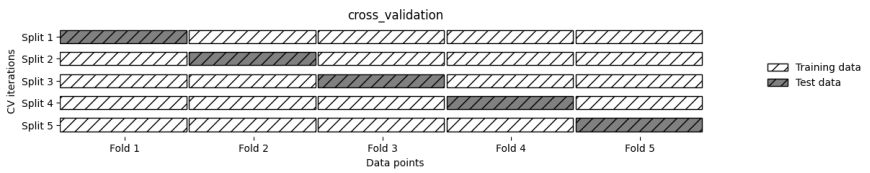

Figure 4.1: Data splitting in five-fold cross-validation

4.1.1 scikit-learn中的交叉验证

scikit-learn 是利用 `model_selection` 模块中的 `cross_val_score` 函数来实现交叉验证的。`cross_val_score` 函数的参数是我们想要评估的模型、训练数据与真实标签。默认情况下，`cross_val_score` 执行 5 折交叉验证，返回 5 个精度值。可以通过修改 cv 参数来改变折数。

总结交叉验证精度的一种常用方法是计算平均值。

4.1.2 交叉验证的优点

1. 减少实验的随机性：如果使用交叉验证，每个样例都会刚好在测试集中出现一次：每个样例位于一个折中，而每个折都在测试集中出现一次。因此，模型需要对数据集中所有样本的泛化能力都很好，才能让所有的交叉验证得分（及其平均值）都很高。
2. 提供模型对训练集选择的敏感性信息：假设对于 iris 数据集，我们观察到精度在 90% 到 100% 之间。这是一个不小的范围，它告诉我们将模型应用于新数据时在最坏情况和最好情况下的可能表现。
3. 对数据的使用更加高效：在使用 5 折交叉验证时，在每次迭代中我们可以使用 4/5（80%）的数据来拟合模型。在使用 10 折交叉验证时，我们可以使用 9/10（90%）的数据来拟合模型。更多的数据通常可以得到更为精确的模型。

交叉验证的主要缺点是增加了计算成本。现在我们要训练 k 个模型而不是单个模型，所以交叉验证的速度要比数据的单次划分大约慢 k 倍。

> 重要的是要记住，交叉验证不是一种构建可应用于新数据的模型的方法。交叉验证不会返回一个模型。在调用 `cross_val_score` 时，内部会构建多个模型，但交叉验证的目的只是评估给定算法在特定数据集上训练后的泛化性能好坏。

4.1.3 分层k折交叉验证和其他策略

将数据集划分为 k 折时，从数据的前 k 分之一开始划分，这可能并不总是一个好主意。对于iris数据集，数据的前三分之一是类别 0，中间三分之一是类别 1，最后三分之一是类别 2。想象一下在这个数据集上进行 3 折交叉验证。第 1 折将只包含类别 0，所以在数据的第一次划分中，测试集将只包含类别 0，而训练集只包含类别 1 和 2。由于在 3 次划分中训练集和测试集中的类别都不相同，因此这个数据集上的 3 折交叉验证精度为 0。这没什么帮助，因为我们在 iris 上可以得到比 0% 好得多的精度。

由于简单的 k 折策略在这里失效了，所以 scikit-learn 在分类问题中不使用这种策略，而是使用**分层 k 折交叉验证**（stratified k-fold cross-validation）。在分层交叉验证中，我们划分数据，使每个折中类别之间的比例与整个数据集中的比例相同，如 Figure 4.2 所示。

举个例子，如果 90% 的样本属于类别 A 而 10% 的样本属于类别 B，那么分层交叉验证可以确保，在每个折中 90% 的样本属于类别 A 而 10% 的样本属于类别 B。

对于回归问题，scikit-learn 默认使用标准 k 折交叉验证。也可以尝试让每个折表示回归目标的不同取值，但这并不是一种常用的策略，也会让大多数用户感到意外。

对交叉验证的更多控制

可以利用 cv 参数来调节 `cross_val_score` 所使用的折数。但 scikit-learn允许提供一个**交叉验证分离器**（cross-validation splitter）作为 cv 参数，来对数据划分过程进行更精细的控制。对于大多数使用场景而言，回归问题默认的 k 折交叉验证与分类问题的分层 k 折交叉验证的表现都很好，但有些情况

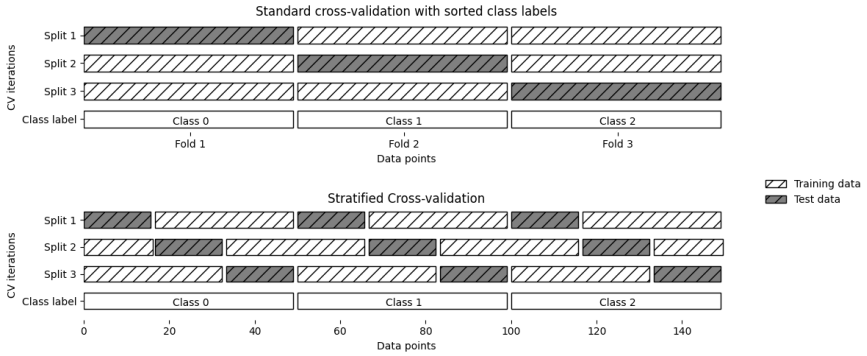

Figure 4.2: Comparison of standard cross-validation and stratified cross-validation

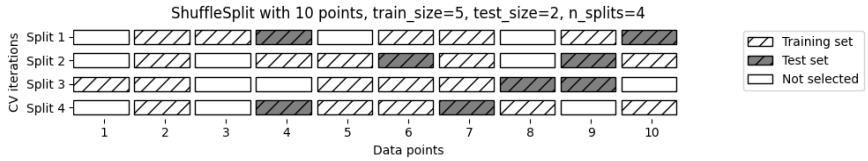

Figure 4.3: Shuffle-split cross-validation

下你可能希望使用不同的策略。比如说，我们想要在一个分类数据集上使用标准 k 折交叉验证来重现别人的结果。可以将 kfold 分离器对象作为 cv 参数传入 `cross_val_score`。

通过这种方法，我们可以验证，在 iris 数据集上使用 3 折交叉验证（不分层）确实是一个非常糟糕的主意。解决这个问题的另一种方法是将数据打乱来代替分层，以打乱样本按标签的排序。可以通过将 KFold 的 shuffle 参数设为 True 来实现这一点。如果我们将数据打乱，那么还需要固定 `random_state` 以获得可重复的打乱结果。

留一法交叉验证

另一种常用的交叉验证方法是**留一法**（leave-one-out）。你可以将留一法交叉验证看作是每折只包含单个样本的 k 折交叉验证。对于每次划分，你选择单个数据点作为测试集。这种方法可能非常耗时，特别是对于大型数据集来说，但在小型数据集上有时可以给出更好的估计结果。

打乱划分交叉验证

一种非常灵活的交叉验证策略是打乱划分交叉验证（shuffle-split cross-validation）。在打乱划分交叉验证中，每次划分为训练集取样 `train_size` 个点，为测试集取样 `test_size` 个（不相交的）点。将这一划分方法重复 `n_iter` 次。Figure 4.3 显示的是对包含 10 个点的数据集运行 4 次迭代划分，每次的训练集包含 5 个点，测试集包含 2 个点（你可以将 `train_size` 和 `test_size` 设为整数来表示这两个集合的绝对大小，也可以设为浮点数来表示占整个数据集的比例）。

打乱划分交叉验证可以在训练集和测试集大小之外独立控制迭代次数，这有时是很有帮助的。它还允许在每次迭代中仅使用部分数据，这可以通过设置 `train_size` 与 `test_size` 之和不等于 1 来实现。用这种方法对数据进行二次采样可能对大型数据上的试验很有用。

ShuffleSplit 还有一种分层的形式，其名称为 StratifiedShuffleSplit，它可以为分类任务提供更可靠的结果。

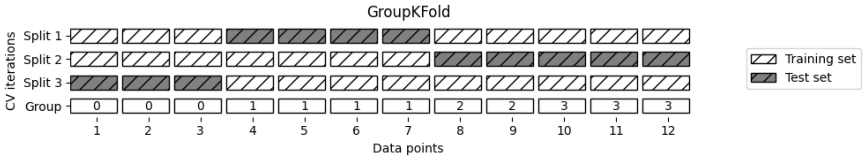

Figure 4.4: Label-dependent splitting with GroupKFold

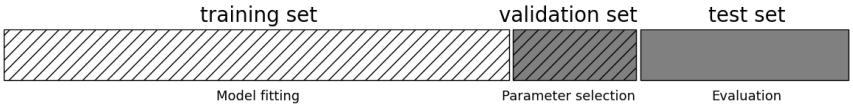

Figure 4.5: A threefold split of data into training set, validation set, and test set

分组交叉验证

GroupKFold is a variation of k-fold which ensures that the same group is not represented in both testing and training sets. 另一种非常常见的交叉验证适用于数据中的分组高度相关时。比如你想构建一个从人脸图片中识别情感的系统，并且收集了 100 个人的照片的数据集，其中每个人都进行了多次拍摄，分别展示了不同的情感。我们的目标是构建一个分类器，能够正确识别未包含在数据集中的人的情感。你可以使用默认的分层交叉验证来度量分类器的性能。但是这样的话，同一个人的照片可能会同时出现在训练集和测试集中。对于分类器而言，检测训练集中出现过的人脸情感比全新的人脸要容易得多。因此，为了准确评估模型对新的人脸的泛化能力，我们必须确保训练集和测试集中包含不同人的图像。

为了实现这一点，我们可以使用 GroupKFold，它以 groups 数组作为参数，可以用来说明照片中对应的是哪个人。这里的 groups 数组表示数据中的分组，在创建训练集和测试集的时候不应该将其分开，也不应该与类别标签弄混。

4.2 网格搜索

在尝试调参之前，重要的是要理解参数的含义。找到一个模型的重要参数（提供最佳泛化性能的参数）的取值是一项棘手的任务，但对于几乎所有模型和数据集来说都是必要的。

考虑一个具有 RBF（径向基函数）核的核 SVM 的例子，它在 SVC 类中实现，它有 2 个重要参数：核宽度 gamma 和正则化参数 C。

4.2.1 简单网格搜索

我们可以实现一个简单的网格搜索，在 2 个参数上使用 for 循环，对每种参数组合分别训练并评估一个分类器

4.2.2 参数过拟合的风险与验证集

我们尝试了许多不同的参数，并选择了在测试集上精度最高的那个，但这个精度不一定能推广到新数据上。由于我们使用测试数据进行调参，所以不能再用它来评估模型的好坏。我们需要一个独立的数据集来进行评估，一个在创建模型时没有用到的数据集。

为了解决这个问题，一种方法是再次划分数据，这样我们得到 3 个数据集：用于构建模型的训练集，用于选择模型参数的验证集（开发集），用于评估所选参数性能的测试集。Figure 4.5 给出了这3个集合的图示。

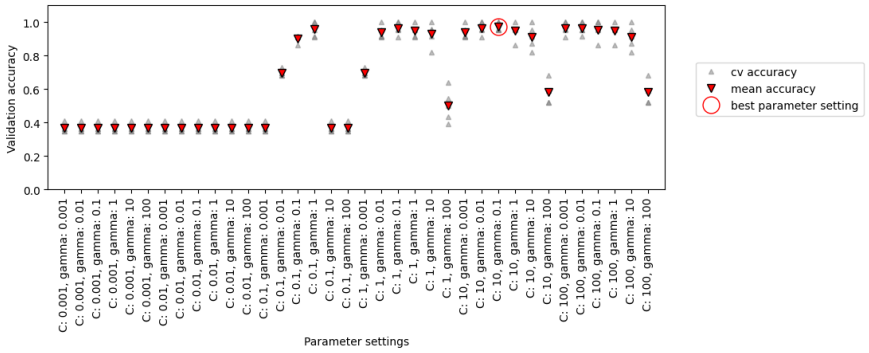

Figure 4.6: Results of grid search with cross-validation

训练集、验证集和测试集之间的区别对于在实践中应用机器学习方法至关重要。任何根据测试集精度所做的选择都会将测试集的信息“泄漏”（leak）到模型中。因此，保留一个单独的测试集是很重要的，它仅用于最终评估。好的做法是利用训练集和验证集的组合完成所有的探索性分析与模型选择，并保留测试集用于最终评估——即使对于探索性可视化也是如此。严格来说，在测试集上对不止一个模型进行评估并选择更好的那个，将会导致对模型精度过于乐观的估计。

4.2.3 带交叉验证的网格搜索

虽然将数据划分为训练集、验证集和测试集的方法（如上所述）是可行的，也相对常用，但这种方法对数据的划分方法相当敏感。为了得到对泛化性能的更好估计，我们可以使用交叉验证来评估每种参数组合的性能，而不是仅将数据单次划分为训练集与验证集。

> 交叉验证是在特定数据集上对给定算法进行评估的一种方法。但它通常与网格搜索等参数搜索方法结合使用。因此，许多人使用交叉验证（cross-validation）这一术语来通俗地指代带交叉验证的网格搜索。

划分数据、运行网格搜索并评估最终参数，这整个过程如 Figure 4.7 所示。

由于带交叉验证的网格搜索是一种常用的调参方法，因此 scikit-learn 提供了 GridSearchCV 类，它以估计器（estimator）的形式实现了这种方法。要使用 GridSearchCV 类，你首先需要用一个字典指定要搜索的参数。然后 GridSearchCV 会执行所有必要的模型拟合。字典的键是我们要调节的参数名称，字典的值是我们想要尝试的参数设置。

创建的 `grid_search` 对象的行为就像是一个分类器，我们可以对它调用标准的 fit、predict 和 score 方法[1]。

拟合 GridSearchCV 对象不仅会搜索最佳参数，还会利用得到最佳交叉验证性能的参数在整个训练数据集上自动拟合一个新模型。

利用交叉验证选择参数，我们实际上找到了一个在测试集上精度为 97% 的模型。重要的是，我们没有使用测试集来选择参数。我们找到的参数保存在 `best_params_` 属性中，而交叉验证最佳精度（对于这种参数设置，不同划分的平均精度）保存在 `best_score_` 中。

[1]用另一个估计器创建的 scikit-learn 估计器被称为元估计器（meta-estimator）。GridSearchCV 是最常用的元估计器

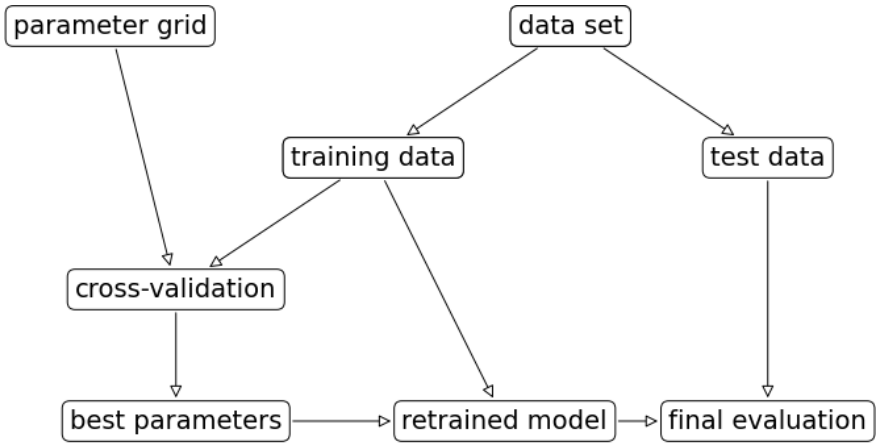

Figure 4.7: Overview of the process of parameter selection and model evaluation with gridsearch

> 同样，注意不要将 `best_score_` 与模型在测试集上调用 score 方法计算得到的泛化性能弄混。使用 score 方法（或者对 predict 方法的输出进行评估）采用的是在整个训练集上训练的模型。而 `best_score_` 属性保存的是交叉验证的平均精度，是在训练集上进行交叉验证得到的。

能够访问实际找到的模型，这有时是很有帮助的，比如查看系数或特征重要性。你可以用 `best_estimator_` 属性来访问最佳参数对应的模型，它是在整个训练集上训练得到的。由于 `grid_search` 本身具有 predict 和 score 方法，所以不需要使用 `best_estimator_` 来进行预测或评估模型。

分析交叉验证的结果

将交叉验证的结果可视化通常有助于理解模型泛化能力对所搜索参数的依赖关系。由于运行网格搜索的计算成本相当高，所以通常最好从相对比较稀疏且较小的网格开始搜索。然后我们可以检查交叉验证网格搜索的结果，可能也会扩展搜索范围。网格搜索的结果可以在 `cv_results_` 属性中找到，它是一个字典，其中保存了搜索的所有内容。

`cv_results_` 中每一行对应一种特定的参数设置。对于每种参数设置，交叉验证所有划分的结果都被记录下来，所有划分的平均值和标准差也被记录下来。由于我们搜索的是一个二维参数网格（C 和 gamma），所以最适合用热图可视化。

可以看到，SVC 对参数设置非常敏感。对于许多种参数设置，精度都在 40% 左右，这是非常糟糕的；对于其他参数设置，精度约为 96%。我们可以从这张图中看出以下几点。首先，我们调节的参数对于获得良好的性能非常重要。这两个参数（C 和 gamma）都很重要，因为调节它们可以将精度从 40% 提高到 96%。此外，在我们选择的参数范围中也可以看到输出发生了显著的变化。同样重要的是要注意，参数的范围要足够大：每个参数的最佳取值不能位于图像的边界上。

如果对于不同的参数设置都看不到精度的变化，也可能是因为这个参数根本不重要。通常最好在开始时尝试非常极端的值，以观察改变参数是否会导致精度发生变化。

基于交叉验证分数来调节参数网格是非常好的，也是探索不同参数的重要性的好方法。但是，你不应该在最终测试集上测试不同的参数范围——前面说过，只有确切知道了想要使用的模型，才能对

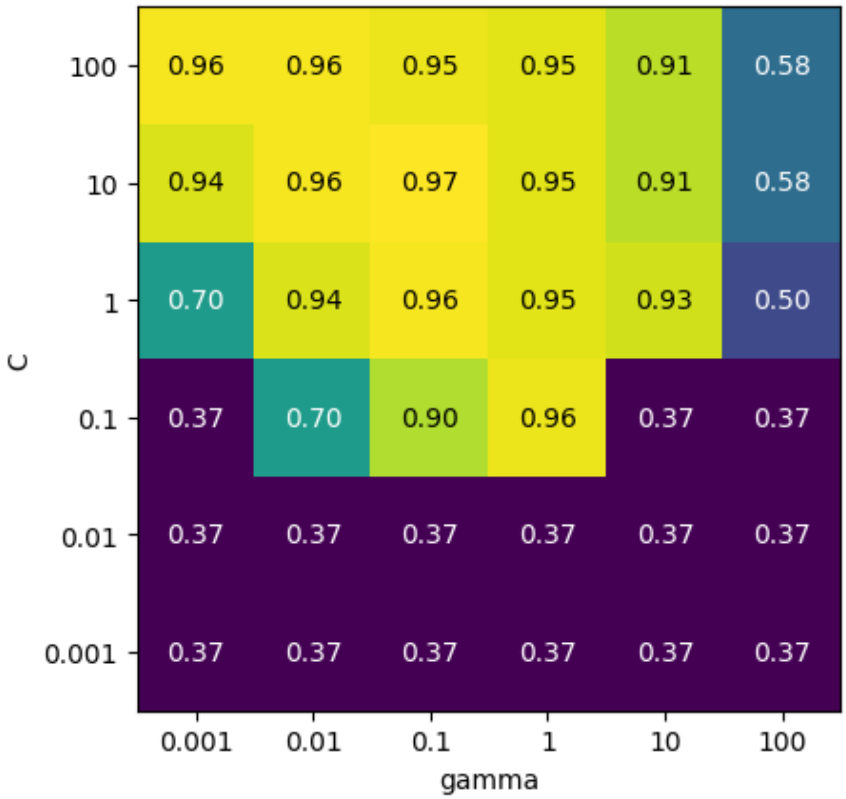

Figure 4.8: Heat map of mean cross-validation score as a function of C and gamma

测试集进行评估。

在非网格的空间中搜索

在某些情况下，尝试所有参数的所有可能组合（正如 GridSearchCV 所做的那样）并不是一个好主意。例如，SVC 有一个 kernel 参数，根据所选择的 kernel（内核），其他参数也是与之相关的。如果 kernel='linear'，那么模型是线性的，只会用到 C 参数。如果 kernel='rbf'，则需要使用 C 和 gamma 两个参数（但用不到类似 degree 的其他参数）。在这种情况下，搜索 C、gamma 和 kernel 所有可能的组合没有意义：如果 kernel='linear'，那么 gamma 是用不到的，尝试 gamma 的不同取值将会浪费时间。为了处理这种“条件”（conditional）参数，GridSearchCV 的 `param_grid` 可以是字典组成的列表（a list of dictionaries）。列表中的每个字典可扩展为一个独立的网格。

使用不同的交叉验证策略进行网格搜索

与 `cross_val_score` 类似，GridSearchCV 对分类问题默认使用分层 k 折交叉验证，对回归问题默认使用 k 折交叉验证。但是，你可以传入任何交叉验证分离器作为 GridSearchCV 的 cv 参数。

嵌套交叉验证 前面在使用 GridSearchCV 时，我们仍然将数据单次划分为训练集和测试集，这可能会导致结果不稳定，也让我们过于依赖数据的此次划分。我们可以再深入一点，不是只将原始数据一次划分为训练集和测试集，而是使用交叉验证进行多次划分，这就是所谓的**嵌套交叉验证**（nested cross-validation）。在嵌套交叉验证中，有一个外层循环，遍历将数据划分为训练集和测试集的所有划分。对于每种划分都运行一次网格搜索（对于外层循环的每种划分可能会得到不同的最佳参数）。然后，对于每种外层划分，利用最佳参数设置计算得到测试集分数。GridSearchCV虽然做了交叉验证，但是其只是找出最优的参数，使用这个最优参数的模型后仍然要交叉验证，否则只是一次划分数据的结果，随机波动性很大

交叉验证与网格搜索并行 虽然在许多参数上运行网格搜索和在大型数据集上运行网格搜索的计算量可能很大，但令人尴尬的是，这些计算都是并行的（parallel）。这也就是说，在一种交叉验证划分下使用特定参数设置来构建一个模型，与利用其他参数的模型是完全独立的。这使得网格搜索与交叉验证成为多个 CPU 内核或集群上并行化的理想选择。你可以将 `n_jobs` 参数设置为你想使用的 CPU 内核数量，从而在 GridSearchCV 和 `cross_val_score` 中使用多个内核。你可以设置 `n_jobs=-1` 来使用所有可用的内核。

你应该知道，scikit-learn 不允许并行操作的嵌套。因此，如果你在模型（比如随机森林）中使用了 `n_jobs` 选项，那么就不能在 GridSearchCV 使用它来搜索这个模型。如果你的数据集和模型都非常大，那么使用多个内核可能会占用大量内存，你应该在并行构建大型模型时监视内存的使用情况。

还可以在集群内的多台机器上并行运行网格搜索和交叉验证，不过目前scikit-learn 还不支持这一点。但是，如果你不介意编写 for 循环来遍历参数的话，可以使用 IPython 并行框架来进行并行网格搜索。

对于 Spark 用户，还可以使用最新开发的 spark-sklearn 包（应该已经被弃用了），它允许在已经建立好的 Spark 集群上运行网格搜索。

4.3 评估指标与评分

4.3.1 牢记最终目标

在选择指标时，你应该始终牢记机器学习应用的最终目标。在实践中，我们通常不仅对精确的预测感兴趣，还希望将这些预测结果用于更大的决策过程。在选择机器学习指标之前，你应该考虑应用的高级目标，这通常被称为商业指标（business metric）。对于一个机器学习应用，选择特定算法的结果被称为商业影响（business impact）。要想评估某个模型的商业影响，可能需要将它放在真实的生产环境中。在开发的初期阶段调参，仅为了测试就将模型投入生产环境往往是不可行的，因为可能涉及很高的商业风险或个人风险。

4.3.2 二分类指标

请记住，对于二分类问题，我们通常会说正类（positive class）和反类（negative class），而正类是我们要寻找的类。

错误类型

错误的阳性预测叫作假正例（false positive, FP），错误的阴性预测）叫作假反例（false negative, FN）。在统计学中，假正例也叫作第一类错误（type I error），假反例也叫作第二类错误（type II error）。

不平衡数据集

如果在两个类别中，一个类别的出现次数比另一个多很多，那么错误类型将发挥重要作用。这种一个类别比另一个类别出现次数多很多的数据集，通常叫作**不平衡数据集**（imbalanced dataset）或者**具有不平衡类别的数据集**（dataset with imbalanced classes）。要想对这种不平衡数据的预测性能进行量化，精度并不是一种合适的度量。

混淆矩阵

精确率（precision）度量的是被预测为正例的样本中有多少是真正的正例:

$$Precision = \frac{TP}{TP+FP}$$

如果目标是限制假正例的数量，那么可以使用准确率作为性能指标。准确率也被称为阳性预测值（positive predictive value，PPV）。

召回率（recall）度量的是正类样本中有多少被预测为正类：

$$Recall = \frac{TP}{TP+FN}$$

如果我们需要找出所有的正类样本，即避免假反例是很重要的情况下，那么可以使用召回率作为性能指标。召回率的其他名称有**灵敏度**（sensitivity）、**命中率**（hit rate）和真正例率（true positive rate，TPR）。

虽然准确率和召回率是非常重要的度量，但是仅查看二者之一无法为你提供完整的图景。将两种度量进行汇总的一种方法是 f-分数（f-score）或 f-度量（f-measure），它是准确率与召回率的调和平均：

$$F = 2\frac{precision * recall}{precision + recall}$$

这一特定变体也被称为 f1-分数（$f_1 - score$）。由于同时考虑了准确率和召回率，所以它对于不平衡的二分类数据集来说是一种比精度更好的度量。然而，f- 分数的一个缺点是比精度更加难以解释。

如果我们想要对准确率、召回率和 f1- 分数做一个更全面的总结，可以使用 `classification_report` 这个很方便的函数，它可以同时计算这三个值，并以美观的格式打印出来

考虑不确定性

混淆矩阵和分类报告为一组特定的预测提供了非常详细的分析。但是，预测本身已经丢弃了模型中包含的大量信息。大多数分类器都提供了一个 `decision_function` 或 `predict_proba` 方法来评估预测的不确定度。预测可以被看作是以某个固定点作为 `decision_function` 或 `predict_proba` 输出的阈值——在二分类问题中，我们使用 0 作为决策函数的阈值，0.5 作为 `predict_proba` 的阈值。

由于 `decision_function`的取值可能在任意范围，所以很难提供关于如何选取阈值的经验法则。

> 如果你设置了阈值，那么要小心不要在测试集上这么做。与其他任何参数一样，在测试集上设置决策阈值可能会得到过于乐观的结果。可以使用验证集或交叉验证来代替。

对于实现了 `predict_proba` 方法的模型来说，选择阈值可能更简单，因为 `predict_proba` 的输出固定在 0 到 1 的范围内，表示的是概率。默认情况下，0.5 的阈值表示，如果模型以超过 50% 的概率“确信”一个点属于正类，那么就将其划为正类。增大这个阈值意味着模型需要更加确信才能做出正类的判断（较低程度的确信就可以做出反类的判断）。虽然使用概率可能比使用任意阈值更加直观，但并非所有模型都提供了不确定性的实际模型（一棵生长到最大深度的 DecisionTree 总是 100% 确信其判断，即使很可能是错的）。这与校准（calibration）的概念相关：校准模型是指能够为其不确定性提供精确度量的模型。你可以在 Alexandru Niculescu-Mizil 和 Rich Caruana 的“Predicting Good Probabilities with Supervised Learning”这篇文章中找到更多内容。

精确率-召回率曲线

旦设定了一个具体目标（比如对某一类别的特定召回率或精确率），就可以适当地设定一个阈值。总是可以设置一个阈值来满足特定的目标，比如 90% 的召回率。难点在于开发一个模型，在满足这个阈值的同时仍具有合理的精确率——如果你将所有样本都划为正类，那么将会得到 100% 的召回率，但你的模型毫无用处。

对分类器设置要求（比如 90% 的召回率）通常被称为设置工作点（operating point）。在业务中固定工作点通常有助于为客户或组织内的其他小组提供性能保证。

在开发新模型时，通常并不完全清楚工作点在哪里。因此，为了更好地理解建模问题，很有启发性的做法是，同时查看所有可能的阈值或精确率和召回率的所有可能折中。利用一种叫作精确率 - 召回率曲线（precision-recall curve）的工具可以做到这一点。

曲线越靠近右上角，则分类器越好。右上角的点表示对于同一个阈值，准确率和召回率都很高。曲线从左上角开始，这里对应于非常低的阈值，将所有样本都划为正类。提高阈值可以让曲线向准确率更高的方向移动，但同时召回率降低。继续增大阈值，大多数被划为正类的点都是真正例，此时准确率很高，但召回率更低。随着准确率的升高，模型越能够保持较高的召回率，则模型越好。

Figure 4.9 可以看出，随机森林在极值处（要求很高的召回率或很高的准确率）的表现更好。在中间位置（精确率约为 0.7）SVM 的表现更好。如果我们只查看 f1-分数来比较二者的总体性能，那么可能会遗漏这些细节。f1-分数只反映了精确率-召回率曲线上的一个点，即默认阈值对应的那个点。

受试者工作特征（ROC）与AUC

还有一种常用的工具可以分析不同阈值的分类器行为：**受试者工作特征曲线**（receiver operating characteristics curve），简称为 ROC 曲线（ROC curve）。与准确率 - 召回率曲线类似，ROC 曲线考虑

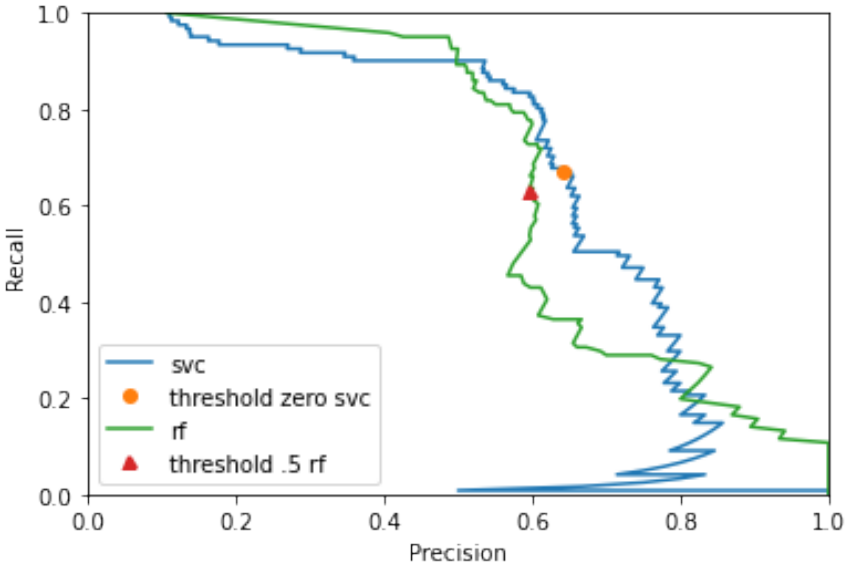

Figure 4.9: Comparing precision recall curves of SVM and random forest

了给定分类器的所有可能的阈值，但它显示的是假正例率（false positive rate，FPR）和真正例率（true positive rate，TPR），而不是报告准确率和召回率。回想一下，真正例率只是召回率的另一个名称，而假正例率（衰退率, fall out）则是假正例占所有反类样本的比例：

$$FPR = \frac{FP}{FP+TN}$$

最接近左上角的点可能是比默认选择更好的工作点。同样请注意，不应该在测试集上选择阈值，而是应该在单独的验证集上选择。

与准确率 - 召回率曲线一样，我们通常希望使用一个数字来总结 ROC 曲线，即曲线下的面积［通常被称为 AUC（area under the curve），这里的曲线指的就是 ROC 曲线］。

对于不平衡的分类问题来说，AUC 是一个比精度好得多的指标。AUC 可以被解释为评估正例样本的排名（ranking）。它等价于从正类样本中随机挑选一个点，由分类器给出的分数比从反类样本中随机挑选一个点的分数更高的概率。

强烈建议在不平衡数据上评估模型时使用 AUC。但请记住，AUC 没有使用默认阈值，因此，为了从高 AUC 的模型中得到有用的分类结果，可能还需要调节决策阈值。

4.3.3 多分类指标

多分类问题的所有指标基本上都来自于二分类指标，但是要对所有类别进行平均。多分类的精度被定义为正确分类的样本所占的比例。同样，如果类别是不平衡的，精度并不是很好的评估度量。

对于多分类问题中的不平衡数据集，最常用的指标就是多分类版本的 f- 分数。多分类 f- 分数背后的想法是，对每个类别计算一个二分类 f- 分数，其中该类别是正类，其他所有类别组成反类。然后，使用以下策略之一对这些按类别 f- 分数进行平均。

- “宏”（macro）平均：计算未加权的按类别 f- 分数。它对所有类别给出相同的权重，无论类别中的样本量大小。
- “加权”（weighted）平均：以每个类别的支持作为权重来计算按类别 f- 分数的平均值。分类报告中给出的就是这个值。

- “微”（micro）平均：计算所有类别中假正例、假反例和真正例的总数，然后利用这些计数来计算准确率、召回率和 f- 分数。

如果你对每个样本等同看待，那么推荐使用“微”平均 f_1-分数；如果你对每个类别等同看待，那么推荐使用“宏”平均 f_1-分数。

4.3.4 回归指标

对回归问题可以像分类问题一样进行详细评估，例如，对目标值估计过高与目标值估计过低进行对比分析。但是，对于我们见过的大多数应用来说，使用默认 R^2 就足够了，它由所有回归器的 score 方法给出。业务决策有时是根据均方误差或平均绝对误差做出的，这可能会鼓励人们使用这些指标来调节模型。但是一般来说，我们认为 R^2 是评估回归模型的更直观的指标。

4.3.5 在模型选择中使用评估指标

详细讨论了许多种评估方法，以及如何根据真实情况和具体模型来应用这些方法。但我们通常希望，在使用 GridSearchCV 或 `cross_val_score` 进行模型选择时能够使用 AUC 等指标。幸运的是，scikit-learn 提供了一种非常简单的实现方法，就是 scoring 参数，它可以同时用于 GridSearchCV 和 `cross_val_score`。你只需提供一个字符串，用于描述想要使用的评估指标。

对于分类问题，scoring 参数最重要的取值包括：accuracy（默认值）、`roc_auc`（ROC 曲线下方的面积）、`average_precision`（准确率 - 召回率曲线下方的面积）、f1、`f1_macro`、 `f1_micro` 和 `f1_weighted`（这四个是二分类的 f1- 分数以及各种加权变体）。对于回归问题，最常用的取值包括：r2（R2 分数）、`mean_squared_error`（均方误差）和 `mean_absolute_error`（平均绝对误差）。你可以在文档中找到所支持参数的完整列表，也可以查看 `metrics._scorer` 模块中定义的 SCORER 字典。

```
from sklearn.metrics._scorer import SCORERS
print('Available scores: \n{}'.format(sorted(SCORERS.keys())))
```

4.4 小结

有两个特别的要点，这里需要重复一下，因为它们经常被新的从业人员所忽视。第一个要点与交叉验证有关。交叉验证或者使用测试集让我们可以评估一个机器学习模型未来的表现。但是，如果我们使用测试集或交叉验证来选择模型或选择模型参数，那么我们就“用完了”测试数据，而使用相同的数据来评估模型未来的表现将会得到过于乐观的估计。因此，我们需要将数据集划分为训练数据、验证数据与测试数据，其中训练数据用于模型构建，验证数据用于选择模型与参数，测试数据用于模型评估。我们可以用交叉验证来代替每一次简单的划分。最常用的形式是训练/测试划分用于评估，然后对训练集使用交叉验证来选择模型与参数。

第二个要点与用于模型选择与模型评估的评估指标或评分函数有关。如何利用机器学习模型的预测结果做出商业决策，可以参考 Data Science for Business。但是，机器学习任务的最终目标很少是构建一个高精度的模型。一定要确保你用于模型评估与选择的指标能够很好地替代模型的实际用途。在实际当中，分类问题很少会遇到平衡的类别，假正例和假反例也通常具有非常不同的后果。你一定要了解这些后果，并选择相应的评估指标。

Chapter 5

算法链与管道

大多数机器学习应用不仅需要应用单个算法，而且还需要将许多不同的处理步骤和机器学习模型链接在一起。本章将介绍如何使用 Pipeline 类来简化构建变换和模型链的过程。我们将重点介绍如何将 Pipeline 和 GridSearchCV 结合起来，从而同时搜索所有处理步骤中的参数。

5.1 用预处理进行参数选择

```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
grid = GridSearchCV(SVC(), param_grid=param_grid, cv=5)
grid.fit(X_train_scaled, y_train)
print('Best cross-validation accuracy: {:.2f}'.format(grid.best_score_))
print('Best set score: {:.2f}'.format(grid.score(X_test_scaled, y_test)))
print('Best parameter:', grid.best_params_)
```

上面的代码中有一个不易察觉的陷阱。在缩放数据时，我们使用了训练集中的所有数据来找到训练的方法。然后，我们使用缩放后的训练数据来运行带交叉验证的网格搜索。对于交叉验证中的每次划分，原始训练集的一部分被划分为训练部分，另一部分被划分为测试部分。测试部分用于度量在训练部分上所训练的模型在新数据上的表现。但是，我们在缩放数据时已经使用过测试部分中所包含的信息。请记住，交叉验证每次划分的测试部分都是训练集的一部分，我们使用整个训练集的信息来找到数据的正确缩放。

为了解决这个问题，在交叉验证的过程中，应该在进行任何预处理之前完成数据集的划分。任何从数据集中提取信息的处理过程都应该仅应用于数据集的训练部分，因此，任何交叉验证都应该位于处理过程的“最外层循环”。

在 scikit-learn 中，要想使用 `cross_val_score` 函数和 GridSearchCV 函数实现这一点，可以使用 Pipeline 类。Pipeline 类可以将多个处理步骤合并（glue）为单个 scikit-learn 估计器。Pipeline 类本身具有 fit、predict 和 score 方法，其行为与 scikit-learn 中的其他模型相同。

5.2 构建管道

构建一个由步骤列表组成的管道对象。每个步骤都是一个元组，其中包含一个名称（你选定的任意字符串，但不应该以双下划线开头）和一个估计器的实例。

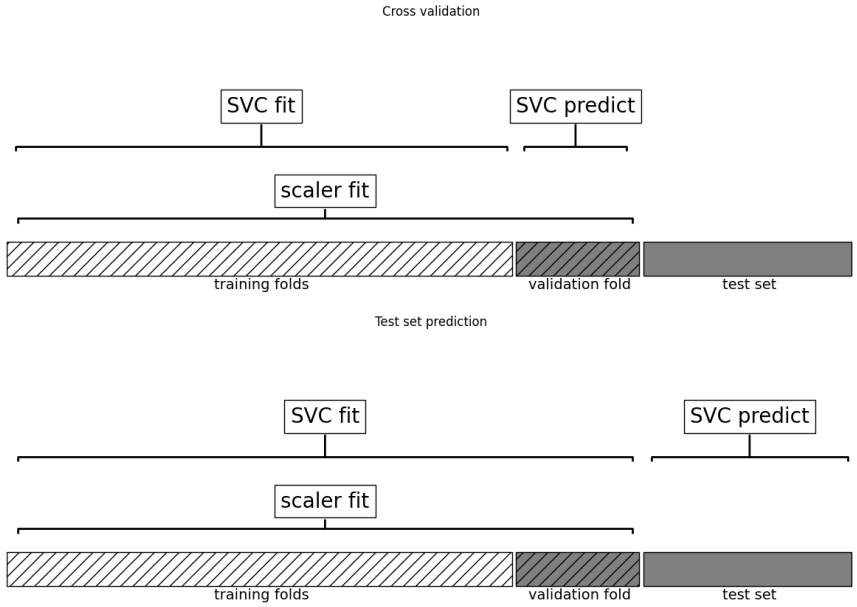

Figure 5.1: Data usage when preprocessing outside the cross-validation loop

5.3 在网格搜索中使用管道

在网格搜索中使用管道的工作原理与使用任何其他估计器都相同。我们定义一个需要搜索的参数网格，并利用管道和参数网格构建一个 GridSearchCV。不过在指定参数网格时存在一处细微的变化。我们需要为每个参数指定它在管道中所属的步骤。为管道定义参数网格的语法是为每个参数指定步骤名称，后面加上 __（双下划线），然后是参数名称。比如，要想搜索 SVC 的 C 参数，必须使用 "svm__C" 作为参数网格字典的键。

在交叉验证中，信息泄露的影响大小取决于预处理步骤的性质。使用测试部分来估计数据的范围，通常不会产生可怕的影响，但在特征提取和特征选择中使用测试部分，则会导致结果的显著差异。

举例说明信息泄露

在 Hastie、Tibshirani 与 Friedman 合著的An Introduction to Statistical Learning，中译版《统计学习导论》一书中给出了交叉验证中信息泄露的一个很好的例子，这里我们复制了一个修改版本。我们考虑一个假想的回归任务，包含从高斯分布中独立采样的 100 个样本与 10 000 个特征。我们还从高斯分布中对响应进行采样：

```
rng = np.random.RandomState(seed=0)
X = rng.normal(size=(100, 10_000))
y = rng.normal(size=(100,))
```

考虑到我们创建数据集的方式，数据 X 与目标 y 之间没有任何关系（它们是独立的），所以应该不可能从这个数据集中学到任何内容。现在我们将完成下列工作。首先利用 SelectPercentile 特征选择从 10 000 个特征中选择信息量最大的特征，然后使用交叉验证对 Ridge 回归进行评估：

```
select = SelectPercentile(score_func=f_regression, percentile=5).fit(X, y)
X_selected = select.transform(X)
print('X_selected.shape: {}'.format(X_selected.shape))
```

```
print('Cross-validation accuracy (cv only on ridge): {:.2f}'.format(
np.mean(cross_val_score(Ridge(), X_selected, y, cv=5))
))
```

交叉验证计算得到的平均 R^2 为 0.91，表示这是一个非常好的模型。这显然是不对的，因为我们的数据是完全随机的。这里的特征选择从 10 000 个随机特征中（碰巧）选出了与目标相关性非常好的一些特征。由于我们在交叉验证之外对特征选择进行拟合，所以它能够找到在训练部分和测试部分都相关的特征。从测试部分泄露出去的信息包含的信息量非常大，导致得到非常不切实际的结果。我们将这个结果与正确的交叉验证（使用管道）进行对比：

```
pipe = Pipeline([
        ('select', SelectPercentile(score_func=f_regression, percentile=5)),
        ('ridge', Ridge())
    ])
print('Cross-validation accuracy (pipeline): {:.2f}'.format(
np.mean(cross_val_score(pipe, X, y, cv=5))
))
```

这一次我们得到了负的 R^2 分数，表示模型很差。利用管道，特征选择现在位于交叉验证循环内部。也就是说，仅使用数据的训练部分来选择特征，而不使用测试部分。特征选择找到的特征在训练集中与目标相关，但由于数据是完全随机的，这些特征在测试集中并不与目标相关。在这个例子中，修正特征选择中的数据泄露问题，结论也由“模型表现很好”变为“模型根本没有效果”。

5.4 通用的管道接口

Pipeline 类不但可用于预处理和分类，实际上还可以将任意数量的估计器连接在一起。对于管道中估计器的唯一要求就是，除了最后一步之外的所有步骤都需要具有 transform 方法，这样它们可以生成新的数据表示，以供下一个步骤使用。

在调用 Pipeline.fit 的过程中，管道内部依次对每个步骤调用 fit 和 transform，其输入是前一个步骤中 transform 方法的输出。对于管道中的最后一步，则仅调用 fit。

请记住，pipeline.steps 是由元组组成的列表，所以 `pipeline.steps[0][1]` 是第一个估计器，`pipeline.steps[1]` 是第二个估计器，以此类推。

管道的最后一步不需要具有 predict 函数，比如说，我们可以创建一个只包含一个缩放器和一个 PCA 的管道。由于最后一步（PCA）具有 transform 方法，所以我们可以对管道调用 transform，以得到将 PCA.transform 应用于前一个步骤处理过的数据后得到的输出。管道的最后一步只需要具有 fit 方法。

5.4.1 用make_pipeline方便地创建管道

使用PipeLine创建管道有时有点麻烦，我们通常不需要为每一个步骤提供用户指定的名称。有一个很方便的函数 `make_pipeline`，可以为我们创建管道并根据每个步骤所属的类为其自动命名，可以通过查看 steps 属性来查看步骤的名称。一般来说，步骤名称只是类名称的小写版本。如果多个步骤属于同一个类，则会附加一个数字。

5.4.2 访问步骤属性

通常来说，你希望检查管道中某一步骤的属性——比如线性模型的系数或 PCA 提取的成分。要想访问管道中的步骤，最简单的方法是通过 `named_steps` 属性，它是一个字典，将步骤名称映射为估计器。

5.4.3 访问网格搜索管道中的属性

5.5 网格搜索预处理步骤与模型参数

5.6 网格搜索选择使用哪个模型

你甚至可以进一步将 GridSearchCV 和 Pipeline 结合起来：还可以搜索管道中正在执行的实际步骤（比如用 StandardScaler 还是用 MinMaxScaler）。这样会导致更大的搜索空间，应该予以仔细考虑。尝试所有可能的解决方案，通常并不是一种可行的机器学习策略。

5.7 小结

本章介绍了 Pipeline 类，这是一种通用工具，可以将机器学习工作流程中的多个处理步骤链接在一起。现实世界中的机器学习应用很少仅涉及模型的单独使用，而是需要一系列处理步骤。使用管道可以将多个步骤封装为单个 Python 对象，这个对象具有我们熟悉的 scikit-learn 接口 fit、predict 和 transform。特别是使用交叉验证进行模型评估与使用网格搜索进行参数选择时，使用 Pipeline 类来包括所有处理步骤对正确的评估至关重要。利用 Pipeline 类还可以让代码更加简洁，并减少不用 pipeline 类构建处理链时可能会犯的错误（比如忘记将所有变换器应用于测试集，或者应用顺序错误）的可能性。选择特征提取、预处理和模型的正确组合，这在某种程度上是一门艺术，通常需要一些试错。但是有了管道，这种“尝试”多个不同的处理步骤是非常简单的。在进行试验时，要小心不要将处理过程复杂化，并且一定要评估一下模型中的每个组件是否必要。

Chapter 6

处理文本数据

文本数据通常被表示为由字符组成的字符串。在上面给出的所有例子中，文本数据的长度都不相同。这个特征显然与前面讨论过的数值特征有很大不同，我们需要先处理数据，然后才能对其应用机器学习算法。

6.1 用字符串表示的数据类型

文本通常只是数据集中的字符串，但并非所有的字符串特征都应该被当作文本来处理。可能会遇到四种类型的字符串数据：

- 分类数据
- 可以在语义上映射为类别的自由字符串
- 结构化字符串数据
- 文本数据

分类数据（categorical data）是来自固定列表的数据。比如你通过调查人们最喜欢的颜色来收集数据，你向他们提供了一个下拉菜单，可以从“红色”“绿色”“蓝色”“黄色”“黑色”“白色”“紫色”和“粉色”中选择。这样会得到一个包含 8 个不同取值的数据集，这 8 个不同取值表示的显然是分类变量。你可以通过观察来判断你的数据是不是分类数据（如果你看到了许多不同的字符串，那么不太可能是分类变量），并通过计算数据集中的唯一值并绘制其出现次数的直方图来验证你的判断。

现在想象一下，你向用户提供的不是一个下拉菜单，而是一个文本框，让他们填写自己最喜欢的颜色。许多人的回答可能是像“黑色”或“蓝色”之类的颜色名称。其他人可能会出现笔误，使用不同的单词拼写（比如“gray”和“grey”），或使用更加形象的具体名称（比如“午夜蓝色”）。

从文本框中得到的回答属于上述列表中的第二类，可以在语义上映射为类别的自由字符串（free strings that can be semantically mapped to categories）。可能最好将这种数据编码为分类变量，你可以利用最常见的条目来选择类别，也可以自定义类别，使用户回答对应用有意义。这样你可能会有一些标准颜色的类别，可能还有一个“多色”类别（对于像“绿色与红色条纹”之类的回答）和“其他”类别（对于无法归类的回答）。这种字符串预处理过程可能需要大量的人力，并且不容易自动化。如果你能够改变数据的收集方式，那么我们强烈建议，对于分类变量能够更好表示的概念，不要使用手动输入值。

通常来说，手动输入值不与固定的类别对应，但仍有一些内在的结构（structure），比如地址、人名或地名、日期、电话号码或其他标识符。这种类型的字符串通常难以解析，其处理方法也强烈依赖

于上下文和具体领域。

最后一类字符串数据是自由格式的文本数据（text data），由短语或句子组成。例子包括推文、聊天记录和酒店评论，还包括莎士比亚文集、维基百科的内容或古腾堡计划收集的 50 000 本电子书。在文本分析的语境中，数据集通常被称为语料库（corpus），每个由单个文本表示的数据点被称为文档（document）。这些术语来自于信息检索（information retrieval，IR）和自然语言处理（natural language processing，NLP）的社区，它们主要针对文本数据。

语料库
文档

6.2 示例应用：电影评论的情感分析

作为本章的一个运行示例，我们将使用由斯坦福研究员 Andrew Maas 收集的 IMDb（Internet Movie Database，互联网电影数据库）网站的电影评论数据集。

6.3 将文本数据表示为词袋

用于机器学习的文本表示有一种最简单的方法，也是最有效且最常用的方法，就是使用词袋（bag-of-words）表示。使用这种表示方式时，我们舍弃了输入文本中的大部分结构，如章节、段落、句子和格式，只计算语料库中每个单词在每个文本中的出现频次。

对于文档语料库，计算词袋表示包括以下三个步骤：

1. 分词（tokenization）。将每个文档划分为出现在其中的单词 [称为词例（token）]，比如按空格和标点划分。
2. 构建词表（vocabulary building）。收集一个词表，里面包含出现在任意文档中的所有词，并对它们进行编号（比如按字母顺序排序）。
3. 编码（encoding）。对于每个文档，计算词表中每个单词在该文档中的出现频次。

6.3.1 将词袋应用于测试数据集

词袋表示是在 CountVectorizer 中实现的，它是一个变换器（transformer）。

6.3.2 将词袋应用于电影评论

> 如果一个文档中包含训练数据中没有包含的单词，并对其调用 CountVectorizer 的 transform 方法，那么这些单词将被忽略，因为它们没有包含在字典中。这对分类来说不是一个问题，因为从不在训练数据中的单词中学不到任何内容。但对于某些应用而言（比如垃圾邮件检测），添加一个特征来表示特定文档中有多少个所谓“词表外”单词可能会有所帮助。为了实现这一点，你需要设置 `min_df`，否则这个特征在训练期间永远不会被用到。

6.4 停用词

删除没有信息量的单词还有另一种方法，就是舍弃那些出现次数太多以至于没有信息量的单词。有两种主要方法：使用特定语言的停用词（stopword）列表，或者舍弃那些出现过于频繁的单词。scikit-learn 的 `feature_extraction.text` 模块中提供了英语停用词的内置列表

6.5 用tf-idf缩放数据

另一种方法是按照我们预计的特征信息量大小来缩放特征，而不是舍弃那些认为不重要的特征。最常见的一种做法就是使用**词频 - 逆向文档频率**（term frequency–inverse document frequency，tf-idf）方法。这一方法对在某个特定文档中经常出现的术语给予很高的权重，但对在语料库的许多文档中都经

常出现的术语给予的权重却不高。如果一个单词在某个特定文档中经常出现，但在许多文档中却不常出现，那么这个单词很可能是对文档内容的很好描述。

scikit-learn 在两个类中实现了 tf-idf 方法：TfidfTransformer 和 TfidfVectorizer，前者接受 CountVectorizer 生成的稀疏矩阵并将其变换，后者接受文本数据并完成词袋特征提取与 tf-idf 变换。单词 w 在文档 d 中的 tf-idf 分数在 TfidfTransformer 类和 TfidfVectorizer 类中都有实现，其计算公式如下所示：

$$tfidf(w,d) = tf \log \frac{N+1}{N_w+1} + 1$$

其中 N 是训练集中的文档数量，N_w 是训练集中出现单词 w 的文档数量，tf（词频）是单词 w 在查询文档 d（你想要变换或编码的文档）中出现的次数。两个类在计算 tf-idf 表示之后都还应用了 L2 范数。换句话说，它们将每个文档的表示缩放到欧几里得范数为 1。利用这种缩放方法，文档长度（单词数量）不会改变向量化表示。由于 tf-idf 实际上利用了训练数据的统计学属性，因此必须要使用Pipeline操作，避免超参调优出现高估的次优解。

请记住，tf-idf 缩放的目的是找到能够区分文档的单词，但它完全是一种无监督技术。因此，这里的“重要”不一定与我们研究的标签相关。

6.6 研究模型系数

6.7 多个单词的词袋（n元分词）

使用词袋表示的主要缺点之一是完全舍弃了单词顺序。因此，“it’s bad, not good at all”（电影很差，一点也不好）和“it’s good, not bad at all”（电影很好，还不错）这两个字符串的词袋表示完全相同，尽管它们的含义相反。将“not”（不）放在单词前面，这只是上下文很重要的一个例子（可能是一个极端的例子）。幸运的是，使用词袋表示时有一种获取上下文的方法，就是不仅考虑单一词例的计数，而且还考虑相邻的两个或三个词例的计数。两个词例被称为二元分词（bigram），三个词例被称为三元分词（trigram），更一般的词例序列被称为 n 元分词（n-gram）。我们可以通过改变 CountVectorizer 或 TfidfVectorizer 的 `ngram_range` 参数来改变作为特征的词例范围。`ngram_range` 参数是一个元组，包含要考虑的词例序列的最小长度和最大长度。

对于大多数应用而言，最小的词例数量应该是 1，因为单个单词通常包含丰富的含义。在大多数情况下，添加二元分词会有所帮助。添加更长的序列（一直到五元分词）也可能有所帮助，但这会导致特征数量的大大增加，也可能会导致过拟合，因为其中包含许多非常具体的特征。原则上来说，二元分词的数量是一元分词数量的平方，三元分词的数量是一元分词数量的三次方，从而导致非常大的特征空间。在实践中，更高的 n 元分词在数据中的出现次数实际上更少，原因在于（英语）语言的结构，不过这个数字仍然很大。

6.8 高级分词、词干提取与词形还原

CountVectorizer 和 TfidfVectorizer 中的特征提取相对简单，还有更为复杂的方法。在更加复杂的文本处理应用中，通常需要改进的步骤是词袋模型的第一步：分词（tokenization）。这一步骤为特征提取定义了一个单词是如何构成的。

词表中通常同时包含某些单词的单数形式和复数形式，与名词的单复数形式一样，将不同的动词形式及相关单词视为不同的词例，这不利于构建具有良好泛化性能的模型。

这个问题可以通过用**词干**（word stem）表示每个单词来解决，这一方法涉及找出或合并（conflate）所有具有相同词干的单词。如果使用基于规则的启发法来实现（比如删除常见的后缀），那么通常将其

称为**词干提取**（stemming）。如果使用的是由已知单词形式组成的字典（明确的且经过人工验证的系统），并且考虑了单词在句子中的作用，那么这个过程被称为**词形还原**（lemmatization），单词的标准化形式被称为**词元**（lemma）。词干提取和词形还原这两种处理方法都是标准化（normalization）的形式之一，标准化是指尝试提取一个单词的某种标准形式。标准化的另一个有趣的例子是拼写校正，这种方法在实践中很有用。

虽然 scikit-learn 没有实现这两种形式的标准化，但 CountVectorizer 允许使用 tokenizer 参数来指定使用你自己的分词器将每个文档转换为词例列表。我们可以使用 spacy 的词形还原了创建一个可调用对象，它接受一个字符串并生成一个词元列表。

6.9 主题建模与文档聚类

常用于文本数据的一种特殊技术是主题建模（topic modeling），这是描述将每个文档分配给一个或多个主题的任务（通常是无监督的）的概括性术语。这方面一个很好的例子是新闻数据，它们可以被分为“政治”“体育”“金融”等主题。如果为每个文档分配一个主题，那么这是一个文档聚类任务。如果每个文档可以有多个主题，那么这个任务与第 3 章中的分解方法有关。我们学到的每个成分对应于一个主题，文档表示中的成分系数告诉我们这个文档与该主题的相关性强弱。通常来说，人们在谈论主题建模时，他们指的是一种叫作隐含狄利克雷分布（Latent Dirichlet Allocation，LDA）的特定分解方法。

6.9.1 隐含狄利克雷分布

从直观上来看，LDA 模型试图找出频繁共同出现的单词群组（即主题）。LDA 还要求，每个文档可以被理解为主题子集的“混合”。重要的是要理解，机器学习模型所谓的“主题”可能不是我们通常在日常对话中所说的主题，而是更类似于 PCA 或 NMF 所提取的成分，它可能具有语义，也可能没有。即使 LDA“主题”具有语义，它可能也不是我们通常所说的主题。

对于无监督的文本文档模型，通常最好删除非常常见的单词，否则它们可能会支配分析过程。

6.10 小结

自然语言和文本处理是一个很大的研究领域。如果你想学习更多内容，我们推荐阅读 Steven Bird、Ewan Klein 和 Edward Loper 合著的 Natural Language Processing with Python 一书，其中给出了 NLP 的概述，并介绍了 nltk 这个用于 NLP 的 Python 库。另一本很好且概念性更强的书是 Christopher Manning、Prabhakar Raghavan 和 Hinrich Schuütze 合著的标准参考，Introduction to Information Retrieval，其中介绍了信息检索、NLP 和机器学习中的基本算法。这两本书都有可以免费访问的在线版本。如前所述，CountVectorizer 类和 TfidfVectorizer 类仅实现了相对简单的文本处理方法。对于更高级的文本处理方法，我们推荐使用 Python 包 spacy（一个相对较新的包，但非常高效，且设计良好）、nltk（一个非常完善且完整的库，但有些过时）和 gensim（着重于主题建模的 NLP 包）。

Chapter 7

总结

7.1 处理机器学习问题

7.2 从原型到生产

论你能否在生产系统中使用 scikit-learn，重要的是要记住，生产系统的要求与一次性的分析脚本不同。如果将一个算法部署到更大的系统中，那么会涉及软件工程方面的很多内容，比如可靠性、可预测性、运行时间和内存需求。对于在这些领域表现良好的机器学习系统来说，简单就是关键。请仔细检查数据处理和预测流程中的每一部分，并问你自己这些问题：每个步骤增加了多少复杂度？每个组件对数据或计算基础架构的变化的鲁棒性有多高？每个组件的优点能否使其复杂度变得合理？如果你正在构建复杂的机器学习系统，我们强烈推荐阅读 Google 机器学习团队的研究者发布的这篇论文：“Machine Learning: The High Interest Credit Card of Technical Debt”

7.3 测试生产系统

我们介绍了如何基于事先收集的测试集来评估算法的预测结果。这被称为离线评估（offline evaluation）。但如果你的机器学习系统是面向用户的，那么这只是评估算法的第一步。下一步通常是在线测试（online testing）或实时测试（live testing），对在整个系统中使用算法的结果进行评估。改变网站向用户呈现的推荐结果或搜索结果，可能会极大地改变用户行为，并导致意想不到的结果。为了防止出现这种意外，大部分面向用户的服务都会采用 A/B 测试（A/B testing），这是一种盲的（blind）用户研究形式。在 A/B 测试中，在用户不知情的情况下，为选中的一部分用户提供使用算法 A 的网站或服务，而为其余用户提供算法 B。对于两组用户，在一段时间内记录相关的成功指标。然后对算法 A 和算法 B 的指标进行对比，并根据这些指标在两种方法中做出选择。使用 A/B 测试让我们能够在实际情况下评估算法，这可能有助于我们发现用户与模型交互时的意外后果。通常情况下，A 是一个新模型，而 B 是已建立的系统。在线测试中还有比 A/B 测试更为复杂的机制，比如 bandit 算法。John Myles White 的 Bandit Algorithms for Website Optimization。

7.4 构建你自己的估计器

scikit-learn 中实现的大量工具和算法，可用于各种类型的任务。但是，你通常需要对数据做一些特殊处理，这些处理方法没有在 scikit-learn 中实现。在将数据传入 scikit-learn 模型或管道之前，只做数据预处理可能也足够了。但如果你的预处理依赖于数据，而且你还想使用网格搜索或交叉验证，那么事情就变得有点复杂了。我们在第 6 章中讨论过将所有依赖于数据的处理过程放在交叉验证循

环中的重要性。那么如何同时使用你自己的处理过程与 scikit-learn 工具？有一种简单的解决方案：构建你自己的估计器！实现一个与 scikit-learn 接口兼容的估计器是非常简单的，从而可以与 Pipeline、GridSearchCV 和 `cross_val_score` 一起使用。你可以在 scikit-learn 文档中找到详细说明，但下面是其要点。实现一个变换器类的最简单的方法，就是从 BaseEstimator 和 TransformerMixin 继承，然后实现 `__init__`、`fit` 和 `predict` 函数。

7.5 下一步怎么走

7.5.1 理论

- Hastie、Tibshirani 和 Friedman 合著的统计学习导论
- Stephen Marsland 的 Machine Learning: An Algorithmic Perspective
- Christopher Bishop 的 Pattern Recognition and Machine Learning
- Kevin Murphy 的 Machine Learning: A Probabilistic Perspective

7.5.2 排序、推荐系统与其他学习类型

重点介绍最常见的机器学习任务：监督学习中的分类与回归，无监督学习中的聚类和信号分解。还有许多类型的机器学习，都有很多重要的应用。有两个特别重要的主题没有包含在本书中。第一个是排序问题（ranking），对于特定查询，我们希望检索出按相关性排序的答案。你今天可能已经使用过排序系统，它是搜索引擎的运行原理。你输入搜索查询并获取答案的有序列表，它们按相关性进行排序。Manning、 Raghavan 和 Schuütze 合著的 Introduction to Information Retrieval 一书给出了对排序问题的很好介绍。第二个主题是推荐系统（recommender system），就是根据用户偏好向他们提供建议。你可能已经在“您可能认识的人”“购买此商品的顾客还购买了”或“您的最佳选择”等标题下遇到过推荐系统。另一种常见的应用是时间序列预测（比如股票价格），这方面也有大量的文献。还有许多类型的机器学习任务，比我们这里列出的要多得多，我们建议你从书籍、研究论文和在线社区中获取信息，以找到最适合你实际情况的范式。

7.5.3 推广到更大的数据集

如果你需要处理 TB 级别的数据，或者需要节省处理大量数据的费用，那么有两种基本策略：核外学习（out-of-core learning）与集群上的并行化（parallelization over a cluster）。

核外学习是指从无法保存到主存储器的数据中进行学习，但在单台计算机上（甚至是一台计算机的单个处理器）进行学习。数据从硬盘或网络等来源进行读取，一次读取一个样本或者多个样本组成的数据块，这样每个数据块都可以读入 RAM。然后处理这个数据子集并更新模型，以体现从数据中学到的内容。然后舍弃该数据块，并读取下一块数据。

另一种扩展策略是将数据分配给计算机集群中的多台计算机，让每台计算机处理部分数据。对于某些模型来说这种方法要快得多，而可以处理的数据大小仅受限于集群大小。但是，这种计算通常需要相对复杂的基础架构。目前最常用的分布式计算平台之一是在 Hadoop 之上构建的 spark 平台。spark 在 MLLib 包中包含一些机器学习功能。如果你的数据已经位于 Hadoop 文件系统中，或者你已经使用 spark 来预处理数据，那么这可能是最简单的选项。但如果你还没有这样的基础架构，建立并集成一个 spark 集群可能花费过大。前面提到的 vw 包提供了一些分布式功能，在这种情况下可能是更好的解决方案。