第一部分
装袋和提升

使用 pandas 预处理数据并构建标准回归和分类模型，并介绍使用 scikit-learn 默认参数的 XGBoost 模型。通过决策树（XGBoost 基学习器）、随机森林（装袋）和梯度提升评分比较，介绍微调数据集合和基于树的超参数的方法，探索 XGBoost 背后的实用理论。

本部分包括以下章节：

第 1 章“机器学习概览”

第 2 章“深入浅出决策树”

第 3 章“随机森林与装袋法”

第 4 章“从梯度提升到 XGBoost”

第 1 章　机器学习概览

本书主要介绍关于 XGBoost 的基础知识与技巧，XGBoost 是用于从表格数据中进行预测的最佳机器学习算法。

XGBoost 也称为极限梯度提升。本书将逐步地详细介绍 XGBoost 的结构、功能和强大的潜在能力，围绕 XGBoost 的产生、发展和应用展开论述。通过学习，读者将成为利用 XGBoost 根据真实数据进行预测的专家。

本章将在机器学习回归和分类的大背景下快速介绍 XGBoost。

本章重点关注机器学习的数据准备过程，该过程也被称为数据整理。除了构建机器学习模型外，还将介绍如何使用高效的 Python 代码来加载数据、描述数据、处理空值、将数据转换为数值列、将数据分割为训练和测试集、构建机器学习模型、实现交叉验证，以及比较使用 XGBoost 的线性回归和逻辑回归模型。

本章介绍的概念和库在全书中都有应用，本章内容主要包括以下 4 点：

（1）XGBoost 概览；

（2）数据整理；

（3）回归预测；

（4）分类预测。

1.1　XGBoost 概览

在 20 世纪 40 年代第一个神经网络出现后，机器学习获得认可。到了 20 世纪 50 年代，第一个基于机器学习的西洋跳棋程序获得冠军后，该领域得到了更多关注。又经过几十年的沉寂，在 20 世纪 90 年代，一款名为“深蓝”（Deep Blue）的超级计算机成功战胜了国际象棋冠军卡斯帕罗夫，机器学习领域再次引起轰动。随着计算能力的大幅提升，从 20 世纪 90 年代至 21 世纪初期，论文产出井喷式爆发，一些新的机器学习算法被开发出来，如随机森林和 AdaBoost。

提升（boosting）背后的常用思路是通过迭代改正错误，将弱学习器转换为强学习器。梯度提升的关键思路是使用梯度下降法来使残差误差最小化。本书前 4 章将重点

介绍如何从标准机器学习算法进化到梯度提升。

XGBoost 是极限梯度提升（Extreme Gradient Boosting）的缩写。“极限”是指挑战算力的极限，实现准确性和速度的极大提高。XGBoost 的流行很大程度上得益于其在 Kaggle 竞赛中的成功。在该竞赛中，参赛者们通过构建机器学习模型，尝试做出最好的预测并赢得丰厚的奖金。与其他模型相比，XGBoost 的表现已远超其竞争对手。

要想理解 XGBoost 的细节，需要在梯度提升算法的背景下理解机器学习的发展态势。本书将从机器学习的基础知识开始，构建起一个整体性的内容框架。

什么是机器学习

机器学习是指计算机从数据中学习的能力。2020 年，机器学习已经可以预测人类行为、推荐产品、识别面孔、战胜职业扑克选手、发现系外行星、辨识疾病、自动驾驶、实现个性化互联网，乃至能够直接与人类交流。机器学习正在引领人工智能革命，并影响着几乎所有的大型公司的盈利能力。

在实践运作中，机器学习需要实现特定的算法，并根据新进数据来调整算法的权重。机器学习的算法基于数据集，经过学习，从而可以预测股市趋势、公司利润、人类决策、亚原子粒子、最优交通路线，等等。

机器学习是将大数据转换为准确可预测的最佳工具。然而，机器学习并非凭空而生，它需要大量结构化的数据。

1.2　数据整理

数据整理是一个综合术语，它涵盖了机器学习开始之前的各个数据预处理阶段。数据加载、数据清理、数据分析和数据操作都包含在数据整理的范围内。

本章详细介绍了数据整理，所用示例旨在涵盖标准的数据处理问题，它们可以通过 pandas 快速处理。pandas 是作为专门用于处理数据分析的 Python 库。虽然本章范例并不需要读者具有 pandas 的使用经验，但具备基本的 pandas 知识将会有助于理解，所以本章会对所有代码进行解释，以便于不熟悉 pandas 的读者能够理解。

1.2.1　数据集 1：自行车租赁数据集

自行车租赁数据集是本书的第一个数据集。数据来源于 UCI 机器学习资料库，这是一家世界知名的免费对公众开放的数据仓库。本书的自行车租赁数据集已经根据原始数据集进行了调整，添加了一些空值，以便于读者练习纠正它们。

访问数据

数据整理的第一步是访问数据。具体步骤如下。

（1）下载数据。本书的所有文件都已存储在 GitHub 上。通过单击 Clone 按钮可将所有文件下载到本地计算机，如图 1.1 所示。

图 1.1　访问数据

下载数据后，将其移动到方便的位置，如桌面上的 Data 文件夹中。

（2）打开 Jupyter Notebook。单击 Anaconda，然后单击 Jupyter Notebook。或者，在终端中输入 Jupyter Notebook。打开网页浏览器后，应该能够看到一个文件夹和文件列表。进入与自行车租赁数据集相同的文件夹，然后选择 New → Notebook → Python 3 菜单项，如图 1.2 所示。

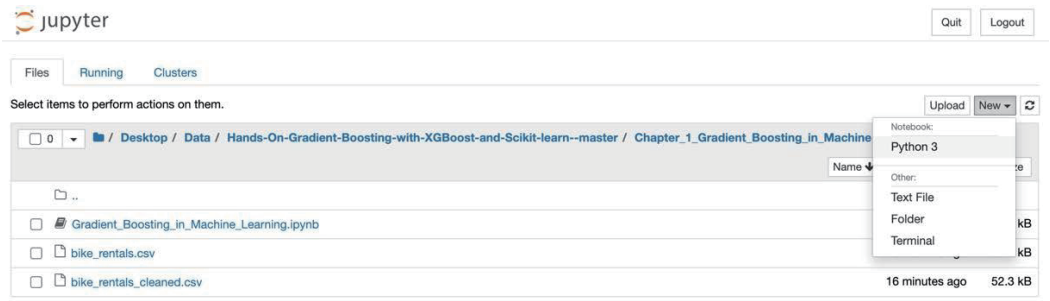

图 1.2　访问 Jupyter Notebook

> **提示**
>
> 如果在打开 Jupyter Notebook 时遇到困难，请参阅 Jupyter 的官方故障排除指南。

（3）在 Jupyter Notebook 的第一个单元格中输入以下代码：

```
import pandas as pd
```

按 Shift + Enter 键来运行单元。现在已经可以通过 pd 访问 pandas 库了。

（4）使用 pd.read_csv 加载数据。加载数据需要一个 read 函数。read 函数将数据存储为 DataFrame，它是一种用于查看、分析和操作数据的 pandas 对象。在载入数据时将文件名用单引号引起来，然后运行单元格，代码如下：

```
df_bikes = pd.read_csv('bike_rentals.csv')
```

如果数据文件与 Jupyter Notebook 位置不同，必须提供一个数据文件所在位置的目录，如 Download/bike_rental.csv。

现在，数据已经正确地存储在一个名为 df_bikes 的 DataFrame 中。

> **提示**
>
> Tab 键自动补全：在 Jupyter Notebook 上编码时，输入几个字符后，按 Tab 键。对于 CSV 文件，会看到文件名出现。用光标高亮显示该名称，然后按 Enter 键。如果文件名是唯一可用的选项，可以按 Enter 键。Tab 键补全将使编码过程更便捷可靠。

（5）使用 .head() 函数显示数据。最后一步是查看数据，以确保它已正确加载。作为 DataFrame 的一种方法，.head() 能够显示 DataFrame 前 5 行。还可以将任何正整数放在括号里，以查看对应数量的行。输入以下代码，然后按 Shift + Enter 键查看结果：

```
df_bikes.head()
```

图 1.3 展示的是 bike_rental.csv 数据集前 5 行数据。

```
# Import pandas
import pandas as pd
```

```
# Upload 'bike_rentals.csv' to dataFrame
df_bikes = pd.read_csv('bike_rentals.csv')
```

```
# Display first 5 rows
df_bikes.head()
```

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1.0	0.0	1.0	0.0	6.0	0.0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	2011-01-02	1.0	0.0	1.0	0.0	0.0	0.0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2	3	2011-01-03	1.0	0.0	1.0	0.0	1.0	1.0	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
3	4	2011-01-04	1.0	0.0	1.0	0.0	2.0	1.0	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
4	5	2011-01-05	1.0	0.0	1.0	0.0	3.0	1.0	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600

图 1.3　bike_rental.csv 数据集前 5 行数据

现在已经可以访问数据，接下来介绍三种理解数据的方法。

1.2.2　理解数据

以下是理解数据的三种重要方法。

1. .head() 函数

.head() 是一种广泛使用的方法，用来解释列名和列编号。如图 1.3 所示，dteday 字段表示日期，而 instant 字段则是一个有序索引。

2. .describe() 函数

可以使用 .describe() 函数来查看数字统计数据，代码如下：

```
df_bikes.describe()
```

上述代码的输出如图 1.4 所示。

	instant	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed
count	731.000000	731.000000	730.000000	730.000000	731.000000	731.000000	731.000000	731.000000	730.000000	730.000000	728.000000	726.000000
mean	366.000000	2.496580	0.500000	6.512329	0.028728	2.997264	0.682627	1.395349	0.495587	0.474512	0.627987	0.190476
std	211.165812	1.110807	0.500343	3.448303	0.167155	2.004787	0.465773	0.544894	0.183094	0.163017	0.142331	0.077725
min	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.059130	0.079070	0.000000	0.022392
25%	183.500000	2.000000	0.000000	4.000000	0.000000	1.000000	0.000000	1.000000	0.336875	0.337794	0.521562	0.134494
50%	366.000000	3.000000	0.500000	7.000000	0.000000	3.000000	1.000000	1.000000	0.499167	0.487364	0.627083	0.180971
75%	548.500000	3.000000	1.000000	9.750000	0.000000	5.000000	1.000000	2.000000	0.655625	0.608916	0.730104	0.233218
max	731.000000	4.000000	1.000000	12.000000	1.000000	6.000000	1.000000	3.000000	0.861667	0.840896	0.972500	0.507463

图 1.4　.describe() 函数的输出

可能需要向右滚动窗口才能看到所有列。比较均值（mean）和中位数（50%） 可以看出偏度。由图 1.4 可知，均值和中位数相差不大，因此数据大致对称。该图还显示了每列的最大值和最小值，以及四分位数和标准差（std）。

3. .info() 函数

.info() 函数会显示关于列和行的基本信息，代码如下：

```
df_bikes.info()
```

以下是上述代码的输出：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
 #          Column        Non-Null Count       Dtype
---         ------        --------------       -----
 0          instant       731 non-null         int64
 1          dteday        731 non-null         object
 2          season        731 non-null         float64
 3          yr            730 non-null         float64
 4          mnth          730 non-null         float64
 5          holiday       731 non-null         float64
```

```
 6          weekday      731 non-null        float64
 7          workingday   731 non-null        float64
 8          weathersit   731 non-null        int64
 9          temp         730 non-null        float64
10          atemp        730 non-null        float64
11          hum          728 non-null        float64
12          windspeed    726 non-null        float64
13          casual       731 non-null        int64
14          registered   731 non-null        int64
15          cnt          731 non-null        int64
dtypes:  float64(10),int64(5),object(1)
memory usage: 91.5+ KB
```

.info() 函数给出了行数、列数、列类型和非空值的数量。既然非空值在不同列之间不同，那么空值必定存在。

1.2.3　纠正空值

如果不纠正空值，则可能会在后续过程中出现意料之外的错误。本节将介绍多种方法用于纠正空值。这些范例不仅展示了如何纠正空值，还旨在突出 pandas 的广度和深度。以下方法可用于纠正空值。

1. 查找空值的数量

显示空值总数的代码如下所示：

```
df_bikes.isna().sum().sum()
```

输出结果如下：

```
12
```

显示空值总数需要两个 .sum() 函数。第一个 .sum() 函数对每列的空值求和，第二个函数则对列计数求和。

2. 显示空值

以下代码显示包含空值的所有行：

```
df_bikes[df_bikes.isna().any(axis=1)]
```

上述代码可以如下分解：df_bikes[conditional] 是满足括号中条件的 df_bikes 的一个子集。df_bikes.isna().any() 函数收集所有空值，而 (axis=1) 参数指定的是列中的值，

因为在 pandas 中，行是 axis 0，列是 axis 1。

图 1.5 是自行车租赁数据集空值显示代码的输出结果。

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
56	57	2011-02-26	1.0	0.0	2.0	0.0	6.0	0.0	1	0.282500	0.282192	0.537917	NaN	424	1545	1969
81	82	2011-03-23	2.0	0.0	3.0	0.0	3.0	1.0	2	0.346957	0.337939	0.839565	NaN	203	1918	2121
128	129	2011-05-09	2.0	0.0	5.0	0.0	1.0	1.0	1	0.532500	0.525246	0.588750	NaN	664	3698	4362
129	130	2011-05-10	2.0	0.0	5.0	0.0	2.0	1.0	1	0.532500	0.522721	NaN	0.115671	694	4109	4803
213	214	2011-08-02	3.0	0.0	8.0	0.0	2.0	1.0	1	0.783333	0.707071	NaN	0.205850	801	4044	4845
298	299	2011-10-26	4.0	0.0	10.0	0.0	3.0	1.0	2	0.484167	0.472846	0.720417	NaN	404	3490	3894
388	389	2012-01-24	1.0	1.0	1.0	0.0	2.0	1.0	1	0.342500	0.349108	NaN	0.123767	439	3900	4339
528	529	2012-06-12	2.0	1.0	6.0	0.0	2.0	1.0	2	0.653333	0.597875	0.833333	NaN	477	4495	4972
701	702	2012-12-02	4.0	1.0	12.0	0.0	0.0	0.0	2	NaN	NaN	0.823333	0.124379	892	3757	4649
730	731	2012-12-31	1.0	NaN	NaN	0.0	1.0	0.0	2	0.215833	0.223487	0.577500	0.154846	439	2290	2729

图 1.5　自行车租赁数据集空值

从图 1.5 可以看出，windspeed、humidity 和 temperature 列以及最后一行中有空值。

> **提示**
>
> 如果读者第一次使用 pandas，可能需要一些时间来习惯它独特的表示法，可参考 Packt 出版的 *Hands-On Data Analysis with Pandas* 一书，其中有完备的讲解。

3. 纠正空值的其他策略

根据列和数据集的不同，采用不同的策略对空值进行纠正，具体如下所述：

1）使用中位数或平均数进行替换

用中位数或平均值代替空值，即用该列的平均值替换空值。例如，对于 'windspeed' 列，空值可以替换为中位数，代码如下：

```
df_bikes['windspeed'].fillna((df_bikes['windspeed'].edian()),lace=True)
```

df_bikes['windspeed'].fillna 表示将填充 'windspeed' 列的空值。df_bikes ['windspeed'].median() 是 'windspeed' 列的中位数。最后，inplace=True 代码确保了更改是永久性的。

> **提示**
>
> 中位数通常是比平均值更好的选择。中位数能保证一半数据大于给定值，一半数据小于给定值。相比之下，平均值容易受到异常值的影响。

如图 1-5 所示，df_bikes[df_bikes.isna().any(axis=1)] 显示第 56 行和 81 行中的风速为空值。这些行可以使用 .iloc（即索引位置，index location 的缩写）显示，代码如下：

```
df_bikes.iloc[[56,81]]
```

上述代码的输出结果如图 1.6 所示。

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
56	57	2011-02-26	1.0	0.0	2.0	0.0	6.0	0.0	1	0.282500	0.282192	0.537917	0.180971	424	1545	1969
81	82	2011-03-23	2.0	0.0	3.0	0.0	3.0	1.0	2	0.346957	0.337939	0.839565	0.180971	203	1918	2121

图 1.6　第 56 行和第 81 行

由图 1.6 可以看到，空值已用风速的中位数替换。

> **提示**
>
> 用户在使用 pandas 时经常会犯单括号或双括号错误。.iloc 对一个索引使用单括号，如 df_ bikes.iloc[56]。不过，df_bikes 也接受括号内的列表，以允许多个索引。多个索引需要双括号，如 df_bikes.iloc[[56，81]]。

2）按中位数 / 平均值分组

使用 .groupby() 函数可以更细致地纠正空值。.groupby() 函数按共享值组织行。因为每一行的数值实际上分布在四个共享的季节中，所以如按季节分组，总共有四行，每个季节一行。但是每个季节都有来自不同的行的不同的值，所以需要一种方法来组合（或者说聚合）这些值。聚合可供选择的方法有：.sum()、.count()、.mean() 和 .median()。下面我们使用 .median()，按季节对 df_bikes 进行分组，代码如下：

```
df_bikes.groupby(['season']).median()
```

图 1.7 是按季节对 df_bikes 分组的输出结果。

season	instant	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
1.0	366.0	0.5	2.0	0.0	3.0	1.0	1.0	0.285833	0.282821	0.543750	0.202750	218.0	1867.0	2209.0
2.0	308.5	0.5	5.0	0.0	3.0	1.0	1.0	0.562083	0.538212	0.646667	0.191546	867.0	3844.0	4941.5
3.0	401.5	0.5	8.0	0.0	3.0	1.0	1.0	0.714583	0.656575	0.635833	0.165115	1050.5	4110.5	5353.5
4.0	493.0	0.5	11.0	0.0	3.0	1.0	1.0	0.410000	0.409708	0.661042	0.167918	544.5	3815.0	4634.5

图 1.7　按季节对 df_bikes 分组的输出结果

如图 1.7 所示，各列值是中位数。

为了纠正 hum 列表征（湿度）中的空值，可以按季节获取湿度的中位数。纠正 hum 列中空值的代码是 df_bikes['hum'] = df_ bikes['hum'].fillna()。fillna 中的代码是期望值。通过 groupby 得到的值需要使用 .transform() 函数，如下所示：

```
df_bikes.groupby('season')['hum'].transform('median')
```

接着，将上述代码合并为一行较长的代码：

```
df_bikes['hum'] = df_bikes['hum'].fillna(df_bikes.
groupby('season')['hum'].transform('median'))
```

可以通过检查 df_bikes.iloc[[129, 213, 388]] 来验证转换是否正确。

3）获取特定行的中位数 / 平均值

在某些情况下，用特定行中的数据替换空值可能更具优势。在校正温度时，除了查阅历史记录外，前后两天的平均温度应该能够提供一种较好的预估。要想查找 'temp' 列中的空值，输入以下代码：

```
df_bikes[df_bikes['temp'].isna()]
```

图 1.8 是包括 'temp' 列空值的输出结果。

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
701	702	2012-12-02	4.0	1.0	12.0	0.0	0.0	0.0	2	NaN	NaN	0.823333	0.124379	892	3757	4649

图 1.8　'temp' 列空值的输出结果

索引 701 行包含空值。

为查找索引 701 行前后两天的平均温度，可按以下步骤进行计算：

（1）将第 700 行和第 702 行的温度值相加，然后除以 2。针对 'temp' 和 'atemp' 列执行以下操作：

```
mean_temp=(df_bikes.iloc[700]['temp']+df_bikes.iloc[702]['temp'])/2
mean_atemp=(df_bikes.iloc[700]['atemp']+df_bikes.iloc[702]['atemp'])/2
```

（2）替换 'temp' 和 'atemp' 两字段的空值，代码如下：

```
df_bikes['temp'].fillna((mean_temp), inplace=True)
df_bikes['atemp'].fillna((mean_atemp), inplace=True)
```

然后，读者可以自行验证空值是否如预期一般填充。

4）推算日期

纠正空值的最后一个策略还涉及日期。当提供真实日期时，可以推算出日期值。df_bikes['dteday'] 是一个日期列，但是 df_bikes.info() 所显示的列的类型是一个对象，通常表示为一个字符串。诸如年和月之类的日期对象必须要从 datetime 类型对象中推断出来。可以使用 to_datetime 函数将 df_ bikes['dteday'] 转换为 'datetime' 类型，如下所示：

```
df_bikes['dteday'] = pd.to_datetime(df_bikes['dteday'],infer_datetime_
format=True)
```

infer_datetime_format=True 允许 pandas 决定要存储的 datetime 对象，在大多数情况下这都是一种安全的选择。

要推断单个列，首先导入 datetime 库，代码如下：

```
import datetime as dt
```

现在可以用一些不同的方法来推断出空值的日期。标准的方法之一是将 'mnth' 列转换为从 'dteday' 列推断出的正确月份。这样做可以修复转换中可能出现的任何其他错误，当然前提要求 'dteday' 列必须是正确的。代码如下：

```
df_bikes['mnth']=df_bikes['dteday'].dt.month
```

验证修改结果非常重要。由于空日期值在最后一行，所以可以使用 .tail() 函数，它类似于 .head() 函数的 DataFrame 函数，不过显示的是最后 5 行，代码如下：

```
df_bikes.tail()
```

如图 1.9 所示，推断 'mnth' 字段日期后，显示最后 5 行内容。

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
726	727	2012-12-27	1.0	1.0	12	0.0	4.0	1.0	2	0.254167	0.226642	0.652917	0.350133	247	1867	2114
727	728	2012-12-28	1.0	1.0	12	0.0	5.0	1.0	2	0.253333	0.255046	0.590000	0.155471	644	2451	3095
728	729	2012-12-29	1.0	1.0	12	0.0	6.0	0.0	2	0.253333	0.242400	0.752917	0.124383	159	1182	1341
729	730	2012-12-30	1.0	1.0	12	0.0	0.0	0.0	1	0.255833	0.231700	0.483333	0.350754	364	1432	1796
730	731	2012-12-31	1.0	NaN	12	0.0	1.0	0.0	2	0.215833	0.223487	0.577500	0.154846	439	2290	2729

图 1.9　推断日期值的输出结果

由结果可知，月份值都是正确，但年份值还需要更改。

'dteday' 列中的年份均为 2012，但 'yr' 列提供的对应年份为 1.0。这是因为数据被归一化，并被转换为 0 ～ 1 的值。

归一化数据通常更有效，因为机器学习权重不必针对不同范围进行调整。

回到纠正空值的主题上来，可以使用 .loc() 函数来填写正确的数值。.loc() 函数可按行和列定位条目，代码如下所示：

```
df_bikes.loc[730,'yr']=1.0
```

现在，练习了如何纠正空值，并且了解到使用 pandas 的一些相关技巧，接下来解决非数字列的问题。

4. 删除非数字列

对于机器学习而言，所有的数据列都应该是数值类型的。由 df.info() 显示结果可知，

df_bikes['dteday'] 是唯一一个非数值列。此外，由于所有日期信息都存在于其他列中，所以 'dteday' 列是多余的。可按如下方式删除该列：

```
df_bikes = df_bikes.drop('dteday', axis=1)
```

现在，我们已经拥有了所有数值列，并且没有空值，完成了进行机器学习的数据准备。

1.3　回归预测

机器学习算法旨在使用来自一个或多个输入列的数据来预测一个输出列的值。这些预测所依赖的数学方程，也正是人们在解决常见的机器学习问题的过程中所用到的。大多数监督学习问题可以分为回归或分类两类。本节将在回归的背景下介绍机器学习。

1.3.1　预测自行车租赁数量

在自行车租赁数据集中，df_bikes['cnt'] 表示某一天的自行车租赁数量。预测这一列对于一家自行车租赁公司将非常有用。本节目标是根据已知数据预测特定一天中租赁的自行车数量，这些数据包括当日是否为假期或工作日，以及温度、湿度、风力等级等信息。

由数据集可知，df_bikes['cnt'] 是 df_bikes['casual'] 与 df_bikes['registered'] 的和。如果把 df_bikes['registered'] 和 df_bikes['casual'] 作为输入列，则预测总是 100% 准确，因为把这些列加起来，总是能够得到正确的结果。虽然完美预测在理论上是理想的，但把现实中未知的输入列包含进来，却是毫无意义的。

除了之前解释过的 'casual' 和 'registered' 外，所有当前列均可以用来预测 df_bikes['cnt']。使用 .drop 函数删除 'casual' 和 'registered' 列，代码如下：

```
df_bikes = df_bikes.drop(['casual', 'registered'], axis=1)
```

至此，数据集就已准备就绪了。

1.3.2　保存数据以备将来使用

本书将会多次用到自行车租赁数据集。读者可将整理后的数据集导出为 CSV 文件以供将来使用，不必每次运行该笔记本代码进行数据整理，执行如下代码：

```
df_bikes.to_csv('bike_rentals_cleaned.csv', index=False)
```

上述代码中的 index=False 参数能够防止索引创建额外的列。

1.3.3　声明预测列和目标列

机器学习的工作原理是，对每个预测列（输入列）进行数学运算以确定目标列（输出列）。

通常预测列用大写字母 X 表示，目标列用小写字母 y 表示。

由于在自行车租赁数据集中，目标列是最后一列，将数据分成预测列和目标列的过程可以通过使用索引符号进行切分来完成，代码如下：

```
X=df_bikes.iloc[:, :-1]
y=df_bikes.iloc[:, -1]
```

上述代码中，逗号将列与行分开。第一个冒号表示所有行都被包括在内。逗号后的 :-1 表示从第一列开始一直到最后一列，但不包括最后一列。第二个 -1 只取最后一列。

1.3.4　理解回归

在现实中，预测得出的自行车租赁数量可能会是任何非负整数。当目标列取值的值域无限时，机器学习问题其实就是**回归问题**。

最常见的回归算法是线性回归。线性回归将每个预测列作为一个多项式变量，并将其值乘以系数（也称为权重）来预测目标列。内部采用梯度下降法以最小化误差。线性回归的预测结果可以是任何实数。

在运行线性回归之前，必须将数据分成训练集和测试集。训练集将数据拟合到算法中，使用目标列来最小化误差。待模型建立后，会根据测试数据对其进行评分。

一定要保留一个测试数据集用于模型评分，这一点非常重要。在大数据研究中，由于存在大量数据点可供训练，过拟合训练集的情况很常见。过拟合通常是不好的，因为模型会过于紧密地调整自己以适应离群值、异常情况或临时趋势。强大的机器学习模型不仅能够很好地泛化到新数据，而且也能在一定程度上较准确地捕捉手头数据的细微差别，换句话说就是能够在两者之间达到良好的平衡。这个概念将在第 2 章中详细探讨。

1.3.5　访问 scikit-learn

本书所有的机器学习库都将通过 scikit-learn 来处理。scikit-learn 的可调整范围、易用性和计算能力使其成为全球最广泛使用的机器学习库之一。

从 scikit-learn 中导入 train_test_split 和 LinearRegression，代码如下：

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

接下来，将数据分为训练集和测试集，代码如下：

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=2)
```

注意，上述代码中的参数 random_state=2 意味着正在选择一个伪随机数生成器的种子，以确保可重现的结果。

1.3.6　关闭警告信息

在构建第一个机器学习模型之前，先将所有警告予以关闭。scikit-learn 的警告用于通知用户功能库未来的变化。一般不建议关闭警告，但由于本书代码已经通过测试，所以在 Jupyter Notebook 中关闭它以节省空间。可以按照以下方式关闭警告：

```
import warnings
warnings.filterwarnings('ignore')
```

接下来将构建第一个模型。

1.3.7　线性回归建模

线性回归模型可分以下几步建立完成：

（1）初始化一个机器学习模型，代码如下：

```
lin_reg = LinearRegression()
```

（2）在训练集上拟合模型，从而开始构建机器学习模型。X_train 是预测列，而 y_train 是目标列，代码如下：

```
lin_reg.fit(X_train, y_train)
```

（3）对测试集进行预测。通过使用 lin_reg 模型的 .predict 函数，将测试集中的预

测列 X_test 的预测值存储为 y_pred，代码如下：

```
y_pred = lin_reg.predict(X_test)
```

（4）借助测试集比较预测结果。对模型进行评分需要有一个比较基准。线性回归的标准是**均方根误差（RMSE）**。计算均方根误差需要两个步骤：先计算 mean_squared_error，即每个预测值与实际值的差值的平方和的平均数，然后再对该平均数求平方根。导入 mean_squared_error，并且使用 Numerical Python 库（通常称为 NumPy 库）求平方根，这是一个专为配合 pandas 而设计的速度极快的算法库。

（5）导入 mean_squared_error 和 NumPy 库，计算均方误差并对其开方，代码如下：

```
from sklearn.metrics import mean_squared_error
import numpy as np
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
```

（6）打印均方根误差值：

```
print("RMSE: %0.2f" % (rmse))
```

上述代码执行结果如下：

```
RMSE: 898.21
```

图 1.10 是构建的第一个机器学习模型的所有代码的屏幕截图。

```
# Initialize LinearRegression model
lin_reg = LinearRegression()

# Fit lin_reg on training data
lin_reg.fit(X_train, y_train)

# Predict X_test using lin_reg
y_pred = lin_reg.predict(X_test)

# Import mean_squared_error
from sklearn.metrics import mean_squared_error

# Import numpy
import numpy as np

# Compute mean_squared_error as mse
mse = mean_squared_error(y_test, y_pred)

# Compute root mean squared error as rmse
rmse = np.sqrt(mse)

# Display root mean squared error
print("RMSE: %0.2f" % (rmse))
```

```
RMSE: 898.21
```

图 1.10　构建机器学习模型的代码

在不知道预期的自行车出租数量范围的情况下，很难确定 898 这个出租量误差的优劣。

可以在 df_bikes['cnt'] 列上使用 .describe() 函数来获取范围等统计信息，代码如下：

```
df_bikes['cnt'].describe()
```

上述代码输出如下：

```
count        731.000000
mean        4504.348837
std         1937.211452
min           22.000000
25%         3152.000000
50%         4548.000000
75%         5956.000000
max         8714.000000
Name:cnt, dtype:float64
```

这个范围为 22 ～ 8714，平均值为 4504，标准差为 1937。所以上述预测均方根误差为 898，虽然不算差，但也不是太好。

1.3.8　XGBoost

线性回归是解决回归问题的众多可用算法之一。其他回归算法可能会产生更好的结果。一般的策略是尝试使用不同的回归器来比较评分。本书将尝试众多回归器，包括决策树、随机森林、梯度提升和 XGBoost。

本书稍后将全面介绍 XGBoost。目前， XGBoost 包含一个称为 XGBRegressor 的回归器，可用于任何回归数据集，包括刚刚演示的自行车租赁数据集。接下来在自行车租赁数据集上，将 XGBRegressor 与线性回归进行回归结果比较。

如果还未安装 XGBoost，则按照序中所述方法安装。

1.3.9　XGBRegressor

安装好 XGBoost 后，可以按如下方式导入 XGBRegressor：

```
from xgboost import XGBRegressor
```

建立 XGBRegressor 的一般步骤与 LinearRegression 相同，唯一的区别是初始化所要使用的是 XGBRegressor，而不是 LinearRegression，接下来详细介绍使用步骤。

（1）初始化机器学习模型，代码如下：

```
xg_reg = XGBRegressor()
```

（2）在训练集上拟合模型。如果在使用 XGBoost 时收到警告信息，忽略即可：

```
xg_reg.fit(X_train, y_train)
```

（3）对测试集进行预测，代码如下：

```
y_pred = xg_reg.predict(X_test)
```

（4）与测试集进行预测比对，代码如下：

```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
```

（5）打印预测结果，代码如下：

```
print("RMSE: %0.2f" % (rmse))
```

上述代码输出如下：

```
RMSE: 705.11
```

与线性回归相比，XGBRegressor 在自行车租赁数据集预测上表现更好，具体原因将在第 5 章详细介绍。

1.3.10　交叉验证

将数据分成不同的训练集和测试集会得出不同的结果，因此单个测试评分并不可靠。由于将数据拆分为训练集和测试集是任意的，所以不同的 random_state 会导致不同的均方根误差。

一种解决不同数据集分组评分差异的方法是 K 折交叉验证。该方法是将数据多次分割为不同的训练集和测试集，然后再取得分的平均值。分割次数由 k 表示。3 次、4 次、5 次或 10 次分割是标准配置。

图 1.11 描述了 K 折交叉验证的工作原理。

交叉验证的工作原理是：在第一个训练集上拟合机器学习模型，并根据第一个测试集对其进行评分；然后为第二次分割提供不同的训练集和测试集，从而得到一个新的机器学习模型及其自己的评分；第三次分割会得到一个新模型，并根据另一个测试集对其评分，余次拆分以此类推。

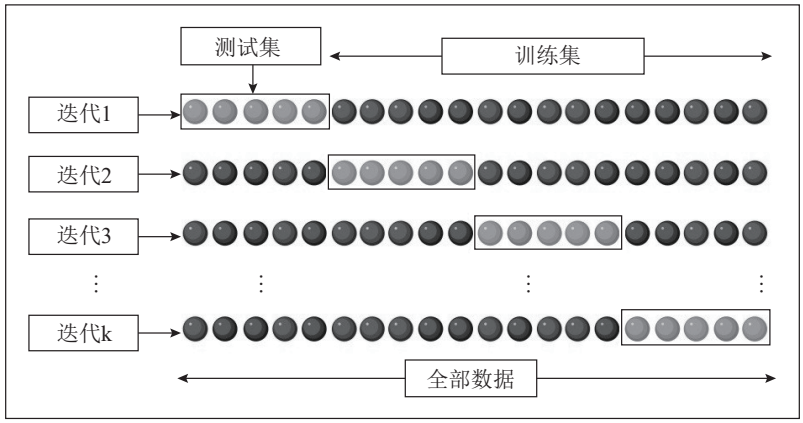

图 1.11　K 折交叉验证工作原理示意图

在数据集分割过程中，训练集将会有重叠，但测试集却不会。

选择拆分次数取决于数据本身。5 折交叉验证是标准的做法，因为每次都会保留测试集的 20%。在 10 折交叉验证中，只有 10% 的数据被保留作为测试集，有 90% 的数据可以用于训练，均值不易受到异常值的影响。对于较小的数据集，3 次分割可能更为适宜。

最终，k 个不同的测试集，将有 k 个不同的得分用于对模型进行评估。将 K 折交叉验证的平均得分作为最终评估指标得分，比单独使用某次分割更加可靠。

cross_val_score 是一种方便的交叉验证实现方式。它的输入包括一个机器学习算法以及预测列和目标列，还可以选择性地添加一些参数，如评分指标和所需的分割数。

1. 线性回归交叉验证

接下来以 LinearRegression 为例来学习使用交叉验证。

首先，从 cross_val_score 库中导入 cross_val_score，代码如下：

```
from sklearn.model_selection import cross_val_score
```

然后使用交叉验证来构建机器学习模型并对其评分，具体步骤如下。

（1）初始化机器学习模型，代码如下：

```
model = LinearRegression()
```

（2）使用模型、X、y、scoring='neg_mean_squared_error' 和拆分数 cv=10 作为输入来实现 cross_val_score，代码如下：

```
scores = cross_val_score(model, X, y, scoring='neg_mean_ squared_
error', cv=10)
```

> **提示**
>
> 之所以使用 scoring='neg_mean_squared_error' 参数，是因为 scikit-learn 旨在训练模型时选择最高评分，这对于精确度来说很有效，但对于误差来说并不最优。通过取每个均方误差的负值，最低值变成了最高值，并通过 rmse = np.sqrt(-scores) 进行补偿，因此最终的结果为正值。

（3）通过对负评分求平方根，找到均方根误差，代码如下：

```
rmse = np.sqrt(-scores)
```

（4）显示上述代码执行结果，代码如下：

```
print('Reg rmse:', np.round(rmse, 2))
print('RMSE mean: %0.2f' % (rmse.mean()))
```

上述代码执行结果如下：

```
Reg rmse: [ 504.01 840.55 1140.88   728.39   640.2   969.95
1133.45 1252.85 1084.64  1425.33]
RMSE mean: 972.02
```

线性回归模型的平均误差为 972.06，比之前获得的 980.38 稍好一些。但这里的重点并不在于评分更好或更差，关键在于这个预估方法显然更优秀，可以更好地预测线性回归在未知数据上的表现。

建议读者始终使用交叉验证来更好地估计评分。

> **关于打印函数**
>
> 当运行机器学习代码时，全局的打印函数并非总是必要的，但如果想打印出多行并按照所示格式输出，则会有帮助。

2. XGBoost 的交叉验证

接下来使用 XGBRegressor 进行交叉验证。除了初始化模型之外，其余步骤与 LinearRegression 相同。

（1）初始化机器学习模型，代码如下：

```
model = XGBRegressor()
```

（2）使用模型、X、y、评分方法（scoring），以及拆分数（cv）作为输入，实现交叉验证评分（cross_val_score），代码如下：

```
scores = cross_val_score(model, X, y, scoring='neg_mean_ squared_
error', cv=10)
```

（3）通过对负评分求平方根，找到均方根误差，代码如下：

```
rmse = np.sqrt(-scores)
```

（4）打印执行结果，代码如下：

```
print('Reg rmse:', np.round(rmse, 2))
print('RMSE mean: %0.2f' % (rmse.mean()))
```

上述代码执行结果如下：

```
Reg rmse: [ 717.65 692.8  520.7  737.68 835.96
1006.24     991.34 747.61 891.99 1731.13]
RMSE mean: 887.31
```

由上述执行结果可知，XGBRegressor 比线性回归好了约 10%。

1.4　分类预测

由 1.3 节可见，XGBoost 在回归方面有优势，但它也具有分类模型，下面将其与经过充分验证的分类模型（如逻辑回归）进行评估比较。

1.4.1　什么是分类?

与回归不同的是，当使用有限的输出预测目标列时，机器学习算法被归类为分类算法。分类可能的输出结果包括以下内容：

（1）Yes、No。

（2）Spam、Not Spam。

（3）0、1。

（4）Red、Blue、Green、Yellow、Orange。

1.4.2　数据集 2：人口普查数据集

接下来将快速浏览第 2 个数据集：人口普查收入数据集。我们打算通过该数据集来预测个人收入。

1. 数据整理

在实现机器学习之前，必须对数据集进行预处理。当测试新算法时，必须要保证所有数值列都不存在空值。

1）数据加载

由于此数据集直接托管在 UCI 机器学习网站上，因此直接下载 pd.read_csv 文件，代码如下：

```
df_census=pd.read_csv(https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data')
df_census.head()
```

图 1.12 是人口普查收入数据集前 5 行数据输出结果。

	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	<=50K

图 1.12　人口普查收入数据集

如图 1.12 所示，列标题只与第一行的条目相关。当发生这种情况时，可以使用 header=None 参数重新加载数据，代码如下：

```
df_census=pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data', header=None)
df_census.head()
```

上述代码执行结果如图 1.13 所示。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

图 1.13　使用参数 header=None 后的输出结果

但是列标题仍然缺失。这些标题可以从人口普查收入数据集网站的属性信息页面中查看。

列标题可以按如下方式更改：

```
df_census.columns=['age', 'workclass', 'fnlwgt', 'education',
  'education-num', 'marital-status', 'occupation',
  'relationship', 'race', 'sex', 'capital-gain', 'capital-
loss', 'hours-per-week', 'native-country', 'income']
df_census.head()
```

图 1.14 是带有列标题的人口普查收入数据集前 5 行输出。

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

图 1.14　包括列标题的人口普查收入数据集

2）空值

要想便捷地检查空值，可以利用 DataFrame.info() 函数，代码如下：

```
df_census.info()
```

上述代码输出如下：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column              Non-Null Count      Dtype
---  ------              --------------      -----
 0   age                 32561 non-null      int64
 1   workclass           32561 non-null      object
 2   fnlwgt              32561 non-null      int64
 3   education           32561 non-null      object
 4   education-num       32561 non-null      int64
 5   marital-status      32561 non-null      object
 6   occupation          32561 non-null      object
 7   relationship        32561 non-null      object
 8   race                32561 non-null      object
 9   sex                 32561 non-null      object
10   capital-gain        32561 non-null      int64
```

```
11   capital-loss         32561 non-null      int64
12   hours-per-week       32561 non-null      int64
13   native-country       32561 non-null      object
14   income               32561 non-null      object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

因为所有列都有相同数量的非空行，所以可以推断没有空值。

3）非数值列

含有 dtype 对象的所有列都必须转换为数值列。pandas 的 get_dummies 函数会取每一列的非数字唯一值，并将其转换为它们自己的列，1 表示存在，0 表示不存在。例如，一个名为 Book Types 的 DataFrame 的列值是 hardback、paperback 或 ebook。使用 pd.get_dummies 后，将创建 3 个名为 hardback、paperback 和 ebook 的新列，来替换 Book Types 列。

图 1.15 展示的是名为 Book Types 的 DataFrame 列值。

	Book Types
0	hardback
1	paperback
2	ebook

图 1.15　Book Types 的 DataFrame 列值

图 1.16 展示的是经过 pd.get_dummies 处理后的相同 DataFrame 列值。

	hardback	paperback	ebook
0	1	0	0
1	0	1	0
2	0	0	1

图 1.16　经过处理后的新 DataFrame 列值

pd.get_dummies 会创建很多新的列，因此值得检查是否有任何列可以被删除。快速查看 df_census 数据会发现 'education' 列和 'education_num' 列。很明显，'education_num' 列是对 'education' 列进行数值转换的结果。因为信息是相同的，所以可以删除 'education' 列，代码如下：

```
df_census = df_census.drop(['education'], axis=1)
```

使用 pd.get_dummies 将非数值列转换为数值列的代码如下：

```
df_census = pd.get_dummies(df_census)
df_census.head()
```

如图 1.17 所示，将人口普查收入数据集中的非数值列转换为数值列的输出。

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	workclass_ ?	workclass_ Federal-gov	workclass_ Local-gov	workclass_ Never-worked	...	native-country_ Scotland	native-country_ South	native-country_ Thailand	native-cc Trinadad&T
0	39	77516	13	2174	0	40	0	0	0	0	...	0	0	0	
1	50	83311	13	0	0	13	0	0	0	0	...	0	0	0	
2	38	215646	9	0	0	40	0	0	0	0	...	0	0	0	
3	53	234721	7	0	0	40	0	0	0	0	...	0	0	0	
4	28	338409	13	0	0	40	0	0	0	0	...	0	0	0	

5 rows × 94 columns

图 1.17　使用 pd.get_dummies 将人口普查收入数据集中的非数值列转换为数值列

可以看出，新列是使用引用原始列的 'column_value' 语法创建的。例如，native-country 是一个原始列，Thailand 是其中的一个值。如果此人来自泰国，则新的 native-country_Thailand 列的值为 1，否则为 0。

> **提示**
>
> 使用 pd.get_dummies 可能会增加内存使用量，可以使用 DataFrame 的 .info() 方法来验证，检查最后一行即可。稀疏矩阵可节省内存，因为仅存储值为 1 的元素，而无须存储值为 0 的元素。有关稀疏矩阵的更多信息，可以参阅第 10 章内容或访问 SciPy 的官方文档。

4）目标列和预测列

由于所有列都是数值型且没有空值，那么接下来将数据拆分为目标列和预测列。

目标列是指某个人年收入是否有 5 万美元。在执行 pd.get_dummies 代码之后，使用 df_census['income_<=50K'] 和 df_census['income_>50K'] 两列来确定某人的年收入是否达到 5 万美元。由于这两列中任意一列均可使用，所以保留一个即可，删除 df_census['income_ <=50K'] 列，代码如下：

```
df_census = df_census.drop('income_ <=50K', axis=1)
```

将数据拆分为 X（预测列）和 y（目标列）。由于最后一列是目标列，因此使用 -1 进行索引，代码如下：

```
X = df_census.iloc[:,:-1]
y = df_census.iloc[:,-1]
```

接下来将构建机器学习分类模型。

2. 逻辑回归

逻辑回归是最基本的分类算法。在数学上，逻辑回归的工作方式类似于线性回归。对于每一列，逻辑回归寻找一个适当的权重或系数，以最大化模型的准确度。主要区别在于，逻辑回归使用 Sigmoid 函数，而不像线性回归那样对每个项进行求和。

如图 1.18 所示，Sigmoid 函数及其对应的曲线。

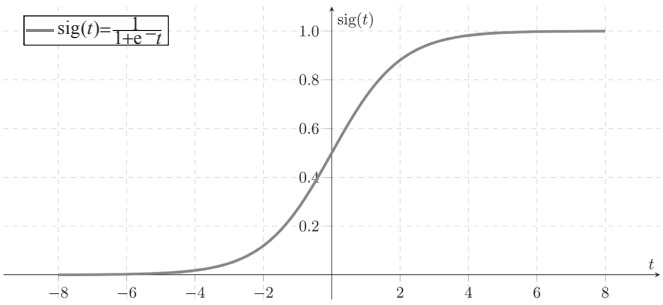

图 1.18　Sigmoid 函数曲线

Sigmoid 通常用于分类。该函数所有大于 0.5 的值都匹配为 1，所有小于 0.5 的值都匹配为 0。

用 scikit-learn 实现逻辑回归与实现线性回归几乎相同。主要的区别在于预测列应该分为不同的类别，而误差应该以准确度为衡量标准。由于误差默认使用精度来表示，因此不需要显式的评分参数。

按如下方式导入逻辑回归：

```
from sklearn.linear_model import LogisticRegression
```

3. 交叉验证函数

使用交叉验证来进行逻辑回归，预测某人年收入是否超过 5 万美元。

构建一个交叉验证分类函数，该函数以机器学习算法作为输入，并使用 cross_val_score 输出准确性评分，代码如下：

```
def cross_val(classifier, num_splits=10):
    model = classifier
    scores = cross_val_score(model, X, y, cv=num_splits) print('Accuracy:',
    np.round(scores, 2))
    print('Accuracy mean: %0.2f' % (scores.mean()))
```

用逻辑回归调用交叉验证函数，代码如下：

```
cross_val(LogisticRegression())
```

上述代码输出如下：

```
Accuracy: [0.8  0.8  0.79 0.8  0.79 0.81 0.79 0.79 0.8  0.8 ]
Accuracy mean: 0.80
```

上述结果显示的 80% 在目前情况下已经很好了。 接下来尝试 XGBoost 是否能够做得更好。

> **提示**
>
> 任何时候，当发现自己在复制和粘贴代码时，都可以寻找更好的方法，计算机科学的目标之一是避免重复。编写自己的数据分析和机器学习程序，将会让生活更加轻松；从长远来看，也会让工作更加高效。

1.4.3　XGBoost 分类器

XGBoost 有回归器和分类器。若要使用分类器，则按如下方式导入分类器：

```
from xgboost import XGBClassifier
```

在 cross_val 函数中运行分类器，并增加一个重要的内容。由于有 94 列数据，而 XGBoost 是一种集成方法，这意味着它在每次运行时会组合许多模型，每个模型包括 10 个拆分，因此将限制 n_estimators（模型数量）为 5。通常 XGBoost 执行非常快。事实上，它是目前最快的提升集成方法。然而，就最初的目的而言，5 个估计量虽不如默认的 100 个稳健，但已足够。关于选择 n_estimators 的细节，第 4 章将重点介绍。代码如下：

```
cross_val(XGBClassifier(n_estimators=5))
```

上述代码输出如下：

```
Accuracy: [0.85 0.86 0.87 0.85 0.86 0.86 0.86 0.87 0.86 0.86]
Accuracy mean: 0.86
```

由上述执行结果可知，XGBoost 在开箱即用的情况下比逻辑回归得分更高。

1.5　总结

本章首先介绍了数据整理和 pandas 的基础知识，这是所有机器学习从业者的基本技能，修复空值是重点。然后，通过比较线性回归和 XGBoost，介绍了如何在 scikit-learn 中构建机器学习模型。接着，准备了一个分类所需的数据集，并将逻辑回归与 XGBoost 进行了比较。在这两种情况下，XGBoost 都是胜者。

通过构建第一个 XGBoost 模型，读者应该掌握了使用 pandas、NumPy 和 scikit-learn 库进行数据处理和机器学习的入门知识。