排名，实时显示在 Public Leaderboard 上，用于给选手提供及时的反馈和动态展示比赛的进行情况；测试集的剩余部分数据（例如 75%或 70%）用于计算参赛选手的最终得分和排名，在比赛结束后才会揭晓，显示在 Private Leaderboard 上。用于计算 Public Leaderboard 和 Private Leaderboard 的数据有不同的划分方式，具体视比赛和数据的类型而定，一般有随机划分、按时间或按一定规则划分等。

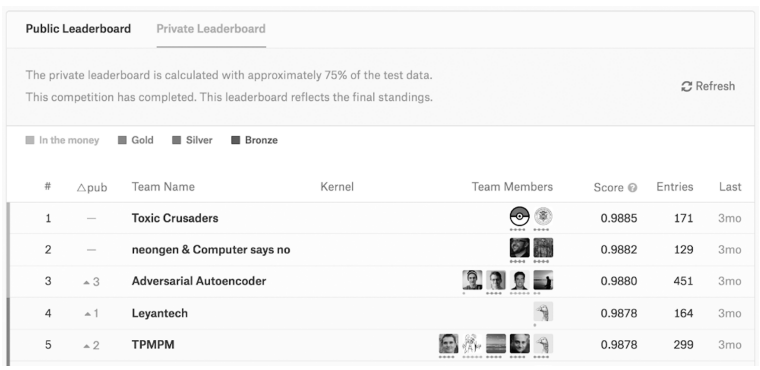


图 15-3　竞赛的 Public Leaderboard 和 Private Leaderboard

Kaggle 竞赛允许单人或者多人组成团队参赛。竞赛通常会限制每队的人数上限，以及每队每日最多可以提交的结果文件的数量（例如不能超过 5 个）。关于组队，建议先单人参加比赛，进行数据探索和模型构建，在比赛后期（如比赛结束前 2～3 周，注意必须在比赛所设置的团队合并最后期限前）合并成一个团队，以充分发挥群智的效果（类似于模型集成，模型差异性越大，越有可能提升效果，超越单个人的模型）。当然也可以在参加比赛前就组好队，这样方便分工协作、讨论问题和碰撞火花等。

15.3　Kaggle 竞赛案例分析：泰坦尼克号乘客生还预测

本节通过最经典的 Kaggle 入门竞赛，泰坦尼克号乘客生还预测（Titanic: Machine Learning from Disaster，https://www.kaggle.com/c/titanic），讲解如何综合运用前面所学来完成一个 Kaggle 竞赛任务。

完成一个竞赛的完整流程通常包括以下几个步骤：

（1）问题理解。理解问题的设定，明确任务目标。

（2）数据理解。包括导入数据，查看数据的基本信息，使用可视化进行探索性数据分析（exlporatory data analysis，EDA）。

（3）数据预处理。包括处理缺失值、异常值，进行数据类型转换和数值缩放等。

（4）特征工程。选择出重要的特征，也可以构造一些新的特征。

（5）基本模型构建。选择某种机器学习模型，构建并训练模型，以及调节最优参数。

（6）模型集成。为进一步改善单模型的效果，可以将多个模型集成起来，降低最终模型的偏差和方差。

（7）模型使用。使用模型对测试数据进行预测，并按照要求的格式生成预测结果文件。

15.3.1　问题理解

泰坦尼克号的沉没是历史上最著名的沉船事件之一。1912 年 4 月 15 日，泰坦尼克号在处女航中与一座冰山相撞后沉入海底。船上共有 2 224 名乘客和船员，其中 1 502 人遇难。这一悲剧举世震惊，沉船导致如此多的人遇难的一个主要原因是泰坦尼克号上没有配备足够多的救生艇。尽管能否生还有运气的成分，但某些类别的人比其他类别的人生还的可能性更大些，如妇女、儿童和上层阶级等。

泰坦尼克号乘客生还的预测是数据科学竞赛平台 Kaggle 上最经典的入门竞赛之一。它是一个二分类问题，要求根据乘客的相关信息，如性别、年龄、船舱等级等，来预测其是否会生还。

15.3.2　数据理解

一、导入数据。该竞赛用的数据（https://www.kaggle.com/c/titanic/data）分为训练数据和测试数据，分别存储在两个 csv 文件中，即 train.csv 和 test.csv，可以使用 Pandas 库中的 read _ csv 读入。

```
# 先导入 pandas 库
import pandas as pd
# 读入数据。有两个数据集:训练数据集和测试数据集,假设下载后存放在当前目录下
trainData = pd.read_csv("train.csv")
testData = pd.read_csv("test.csv")
```

要将在训练集上训练好的模型最后应用在测试集上，需要对训练数据和测试数据做相同的预处理，即做相同的特征转换和抽取等。为了方便后面的预处理，可以先把两个数据集合成一个数据集。

```
# 合并数据集,方便同时对两个数据集进行相同的预处理
fullData = trainData.append(testData, ignore_index = True)
```

二、查看数据信息。在使用数据训练模型之前需要先了解数据，包括查看一些样例数据，以及数据集的基本信息，如包含多少行和多少列、每列的数据类型和值的分布等。

```
# 显示前 5 行数据
fullData.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

另外，可以用于显示样例数据的函数还有 tail() 和 sample()。例如，可以使用 fullData.tail(5) 显示最后 5 行，fullData.sample(10) 显示随机抽取的 10 行等。

```
# 显示行数和列数
print('训练数据集:', trainData.shape, '测试数据集:', testData.shape)
print ('合并后的数据集:', fullData.shape)
```

```
训练数据集:(891, 12) 测试数据集:(418, 11)
合并后的数据集:(1309, 12)
```

可以看到训练集中有 891 行数据，即 891 名乘客的信息。每名乘客由 12 列特征来描述：

- PassengerId：乘客编号
- Survived：是否生还（0 代表未生还，1 代表生还）
- Pclass：船舱等级
- Name：乘客姓名
- Sex：乘客性别
- Age：乘客年龄
- SibSp：乘客在船上的兄弟姐妹及配偶数量

- Parch：乘客在船上的父母和子女数量
- Ticket：船票编号
- Fare：票价
- Cabin：舱位
- Embarked：登船港口

测试集中有 418 行（即 418 名乘客）。训练集和测试集合起来总共有 1 309 行。测试集的所有列与训练集都一样，除了不含有 Survived 列，即我们要预测的那一列，所以测试集比训练集少一列。

可以使用 info() 显示每列的数据类型和非空值的个数等。

```
# 显示所有列的数据类型和非空值的个数
fullData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns(total 12 columns):
Age                  1046 non-null float64
Cabin                295 non-null object
Embarked             1307 non-null object
Fare                 1308 non-null float64
Name                 1309 non-null object
Parch                1309 non-null int64
PassengerId          1309 non-null int64
Pclass               1309 non-null int64
Sex                  1309 non-null object
SibSp                1309 non-null int64
Survived             891 non-null float64
Ticket               1309 non-null object
dtypes: float64(3), int64(4), object(5)
memory usage: 122.8+ KB
```

可以看到，12 列中有 7 列是数值型特征（其中 PassengerId，Pclass，SibSp，Parch 列是整数类型，Age，Survived 和 Fare 列是小数类型），5 列是非数值型特征（即 Name，Sex，Ticket，Cabin 和 Embarked）。一些列，如 Age，Cabin，Fare 和 Embarked，非空值的个数小于 1 309，说明这些列中存在缺失值。其中 Age 列和 Cabin 列的缺失值较多，Embarked 列有两个缺失值，Fare 列只有一个缺失值。我们后面进行数据预处理时再讨论如何处理缺失数据。

对于数值型特征（整数或小数），可以使用 describe() 显示每列的基本描述统计信息，如最大值、最小值、均值、标准差和计数等。

```
# 显示数值型特征列的描述统计信息
fullData.describe()
```

	Age	Fare	Parch	PassengerId	Pclass	SibSp	Survived
count	1046.000000	1308.000000	1309.000000	1309.000000	1309.000000	1309.000000	891.000000
mean	29.881138	33.295479	0.385027	655.000000	2.294882	0.498854	0.383838
std	14.413493	51.758668	0.865560	378.020061	0.837836	1.041658	0.486592
min	0.170000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000
25%	21.000000	7.895800	0.000000	328.000000	2.000000	0.000000	0.000000
50%	28.000000	14.454200	0.000000	655.000000	3.000000	0.000000	0.000000
75%	39.000000	31.275000	0.000000	982.000000	3.000000	1.000000	1.000000
max	80.000000	512.329200	9.000000	1309.000000	3.000000	8.000000	1.000000

如上所示，乘客中年龄最大的 80 岁，年龄最小的不满半岁。通过描述统计信息还可以观测出一些明显的异常值，例如大于 200 的年龄肯定是异常值。

对于类别特征（即取值范围是有限个离散值，可以是数值型的离散值也可以是字符串），我们可以使用 value_counts() 查看该特征列所有不同取值的个数。首先可以查看一下训练集中待预测的目标列（即 Survived 列）中各种类别的出现次数，以判断是否存在类别非常不平衡的问题，比如正例的发生次数非常少。对类别分布非常不均衡的情况，需要采取一些措施，例如过采样或欠采样，才能取得较好的预测效果。

```
# 显示待预测列中各种类别值及其出现次数
trainData["Survived"].value_counts()
```

```
0    549
1    342
Name: Survived,dtype: int64
```

训练集的 891 名乘客中，549 人没有生还，342 人生还，类别分布基本均衡。也可以如下直接显示出每种类别所占的比例。

```
# 显示待预测列中各种类别值及其出现次数占总次数的比例
trainData["Survived"].value_counts(normalize = True)
```

```
0    0.616162
1    0.383838
Name: Survived, dtype: float64
```

三、探索性数据分析。除用以上方法概览数据的一些基本信息之外，还可以使用可视化技术对数据进行更深入的探索性分析，发现数据中的模式、异常或关系等。例如，绘制直方图或密度图来观察单个特征列的值的分布；绘制散点图来发现两个特征列或者一个特征列和目标列之间的关联关系。分析结果可用作进一步处理数据和特征选择的依据。

常用的可视化 Python 库有 Matplotlib 和 Seaborn，其中 Seaborn 是在 Matplotlib 的基础上进行了更高级的封装，作图更方便快捷，代码更加简单，图形更加美观。下面主要以 Seaborn 库示例如何对数据进行可视化分析。使用 Seaborn 之前要先导入该库。

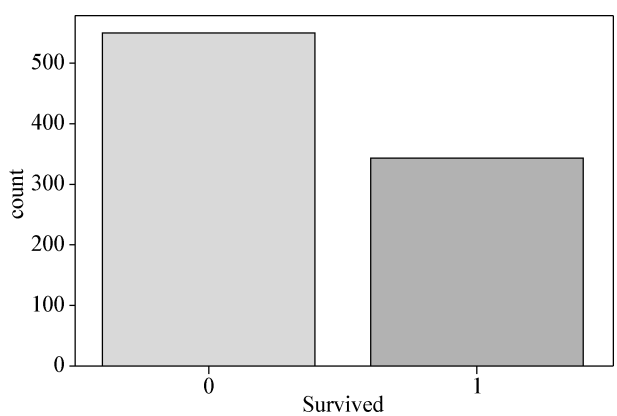
```
# 先导入 seaborn 库
import seaborn as sns
% matplotlib inline
```

在泰坦尼克号训练数据集的 12 列中，除了要预测的目标列 Survived 和用于标识每个乘客的编号列 PassengerId，其他 10 个特征列中，Age 和 Fare 是连续变量，Pclass，SibSp，Parch，Name，Sex，Ticket，Cabin 和 Embarked 是离散变量（其中 Pclass，SibSp，Parch 列是整数型，Name，Sex，Ticket，Cabin 和 Embarked 是字符串型）。

（1）单变量可视化。对于单变量，我们主要想了解其值的分布情况。

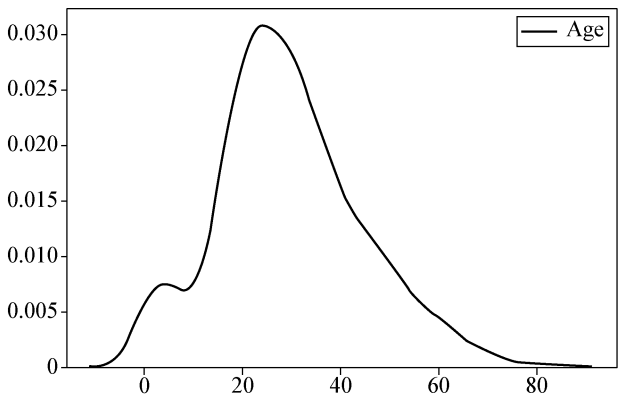
分类变量值的分布情况可以使用 countplot()（计数图，类似一个分类直方图）来展示变量每个取值的出现次数。下面使用计数图将 Survived 列值的分布情况展示出来。

```
# 可视化 Survived 列值的分布情况
sns.countplot(trainData.Survived)
```



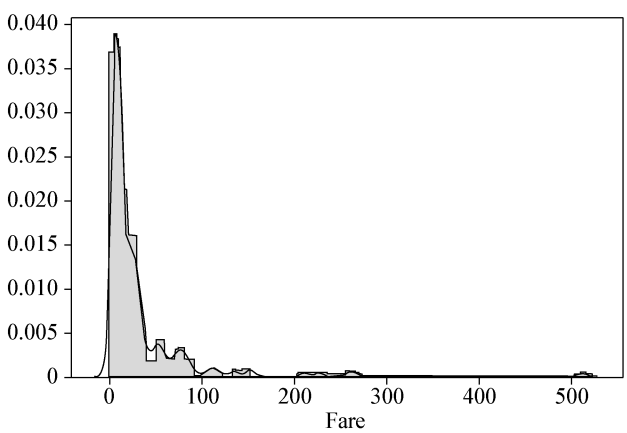
连续变量值的分布情况，可以使用直方图，也可以用概率密度函数来刻画。例如，可以使用 kdeplot()（核密度估计图）来展示一个连续变量的数据分布情况。下面使用核密度估计图展示 Age 列数据的分布情况。可以看到乘客的年龄基本呈一个均值为 20～30 的正态分布。

```
# 可视化 Age 列值的分布情况
sns.kdeplot(trainData.Age)
```



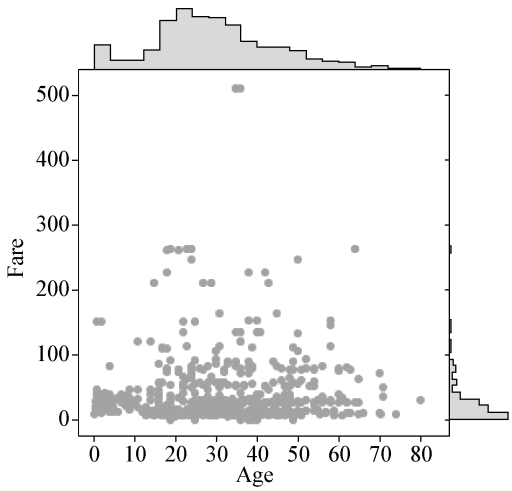
另外，Seaborn 中还提供了功能更强大的 distplot()(分布图)，可以绘制出许多其他的近似分布。缺省地，distplot 直接绘制出需要的直方图及对应的核密度估计图。如下所示，用 distplot 绘制出 Fare 列值的分布情况。可以看到大部分船票价格在 50 美元以下。

```
# 可视化 Fare 列值的分布情况
sns.distplot(trainData.Fare)
```



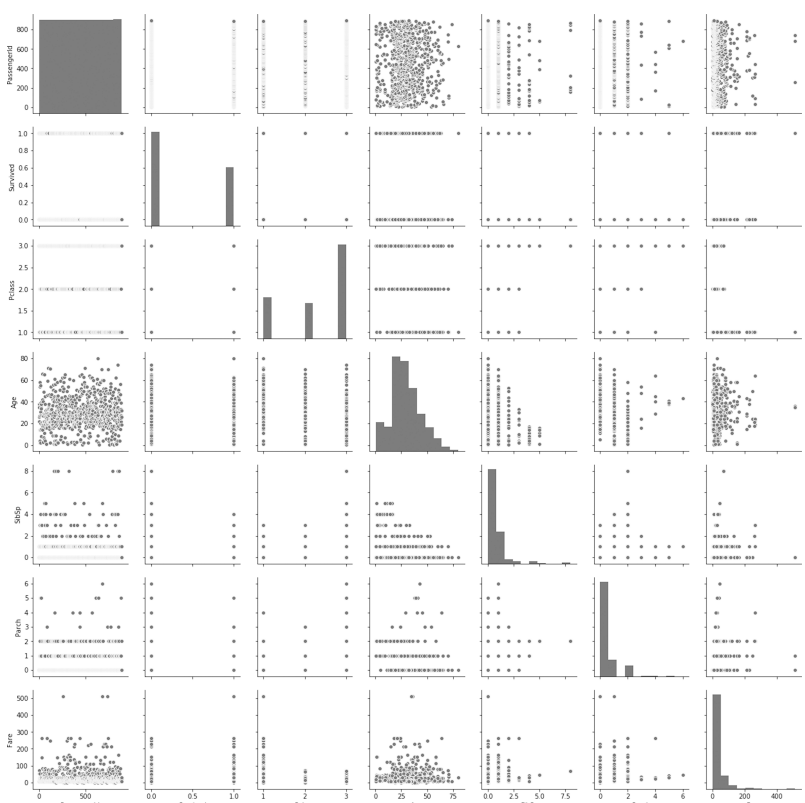
(2) 两个变量的可视化。两个变量的联合分布最常用散点图的形式进行可视化。Seaborn 里提供了 jointplot()(联合图) 可以方便地可视化两个数值型变量的联合分布。例如，下面可视化 Age 和 Fare 两个变量的联合分布情况。如果在 jointplot 函数调用中增加一个参数 **kind = "kde"**，则会展示为二维核密度图而不是散点图的形式。

```
# 可视化 Age 和 Fare 两列值的联合分布情况
sns.jointplot(x = "Age", y = "Fare", data = trainData)
```



Seaborn 里提供的 pairplot() 可以同时绘制出数据集中所有数值型变量两两之间的联合分布图，如下所示，其中对角线上是单变量的分布直方图。

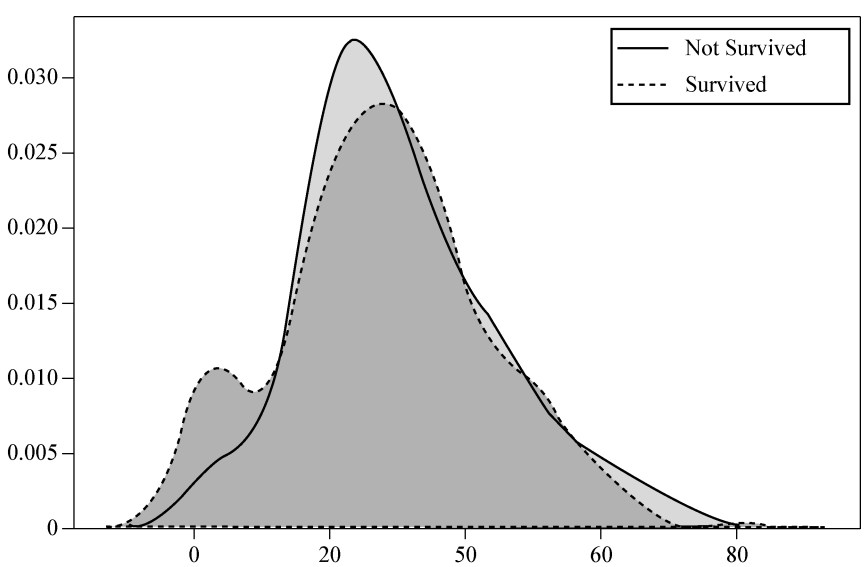
```
# 可视化所有数值型列之间的两两联合分布情况
sns.pairplot(trainData)
```



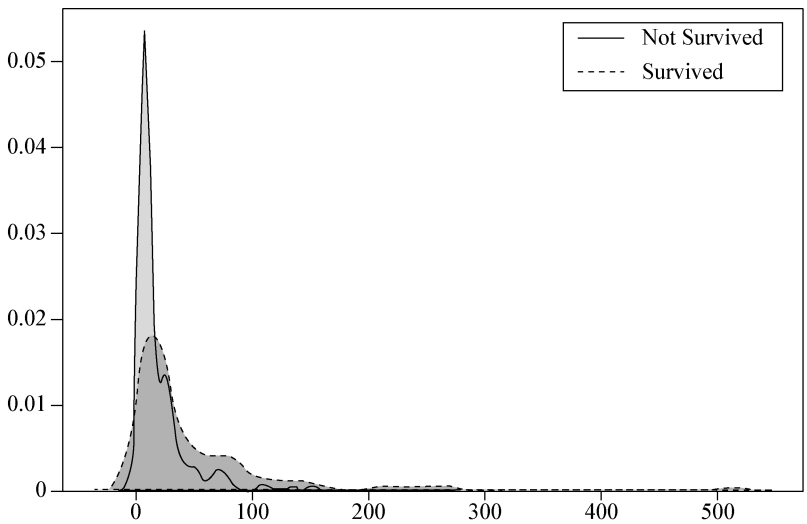
除了解变量值的分布情况，更重要的是要了解变量之间的关系，有助于做特征变换和特征选择。在解决预测问题中，我们更想知道每个特征变量和目标变量之间的相关性，从而选择可能会对预测产生较大影响的特征进行建模。通过两个变量的散点图可以观察到变量之间的一些关系，比如是否具有线性相关性等。泰坦尼克号乘客生还预测是个二分类任务，目标变量 Survived 是一个二值类别型变量。通过上面显示的散点图（即第二行）很难观察出其他变量和它的相关性，或者说对其的预测能力。下面使用一些其他可视化图形来进一步观察每个特征变量对预测 Survived 的影响。

首先，对于连续变量 Age 和 Fare，可以绘制出它们在不同预测类别中（即 Survived 的不同取值时）值的分布情况（核密度估计图），来观察这些变量在不同类别中的分布是否有较大差异。

```
# 可视化 Age 与 Survived 不同取值的关系
sns.kdeplot(trainData["Age"][trainData.Survived == 0], color = "Red", shade = True, label = 'Not Survived')
sns.kdeplot(trainData["Age"][trainData.Survived == 1], color = "Blue", shade = True, label = 'Survived')
```




```
# 可视化 Fare 与 Survived 不同取值的关系
sns.kdeplot(trainData["Fare"][trainData.Survived == 0], color = "Red", shade = True, label = 'Not Survived')
sns.kdeplot(trainData["Fare"][trainData.Survived == 1], color = "Blue", shade = True, label = 'Survived')
```



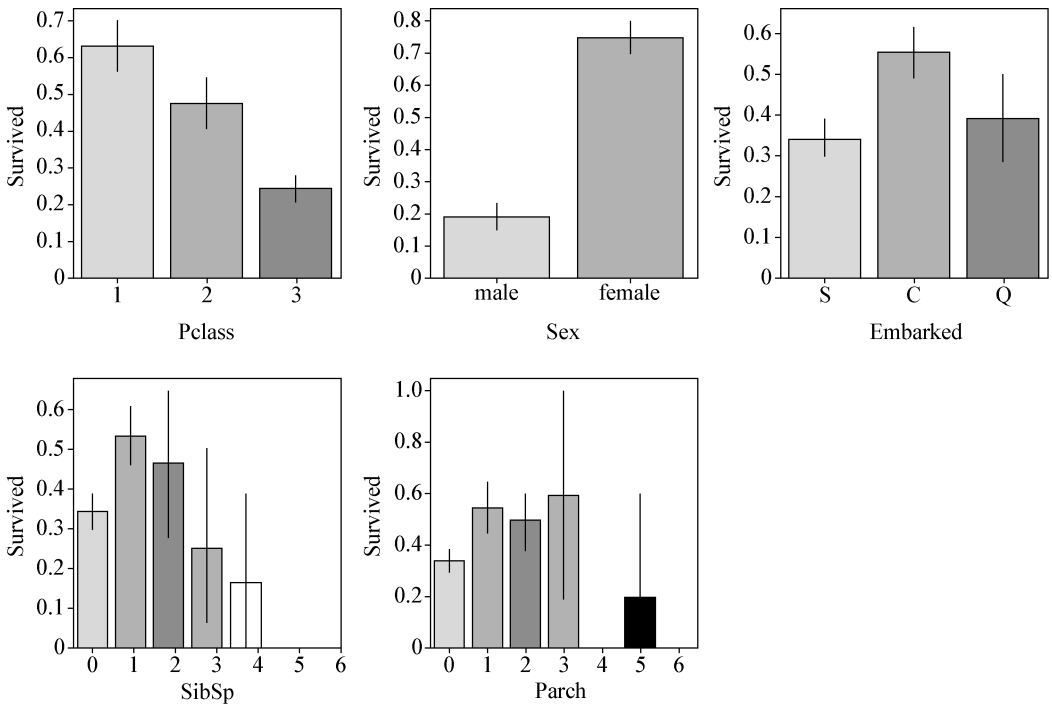
从上面两张图中可以看出：

- 生还的乘客和未生还的乘客的年龄分布具有明显的不同，大部分乘客的年龄都集中在 20～40 岁之间，而生还的乘客年龄有两个峰值段，年龄较小（0～8 岁）和年龄中等（20～35 岁）。
- 票价（Fare）在生还的乘客和未生还的乘客中的分布较为相似，只是未生还的乘客的 Fare 值更集中些。

其次，对于取值范围是少数有限个值的离散变量 Pclass，Sex，Embarked，SibSp 和 Parch，可以用 barplot()(分类条形图）来可视化它们不同取值下的生还率是否有较大差异。缺省地，barplot 对 y 轴数据使用的统计估计函数是求均值函数。因为 Survived 列只有两个可能的取值，0 表示未生还，1 表示生还，所以 Survived 列的平均值恰好就是生还人数占总人数的比例，即生还率。下面我们将五个变量的条形图放在一起展示。

```
import matplotlib.pyplot as plt

fig,axes = plt.subplots(2, 3, figsize = (15,10))
# Pclass 列的不同取值下的生还率
sns.barplot(x = 'Pclass', y = 'Survived', data = trainData, ax = axes[0,0])
# Sex 列的不同取值下的生还率
sns.barplot(x = 'Sex', y = 'Survived', data = trainData, ax = axes[0,1])
# Embarked 列的不同取值下的生还率
sns.barplot(x = 'Embarked', y = 'Survived', data = trainData, ax = axes[0,2])
# SibSp 列的不同取值下的生还率
sns.barplot(x = 'SibSp', y = 'Survived', data = trainData, ax = axes[1,0])
# Parch 列的不同取值下的生还率
sns.barplot(x = 'Parch', y = 'Survived', data = trainData, ax = axes[1,1])
```

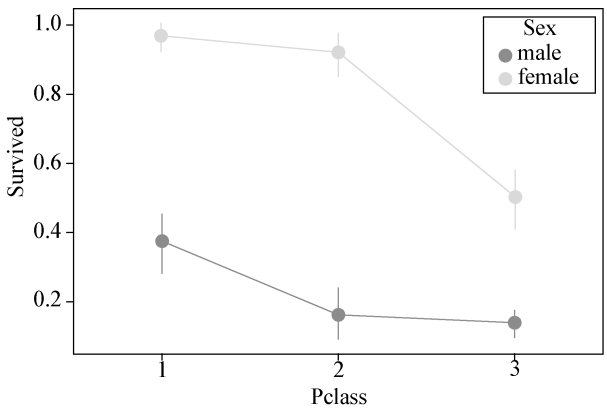


从上面五幅分类条形图中可以看出：

- 不同阶层（Pclass）乘客之间的生还率差别较大，阶层越高，乘客的生还率越大。
- 女性的生还率远大于男性，说明性别（Sex）对生还率的影响很大。
- 登船港口（Embarked）有三个值：S 是出发港口英国南安普敦（South Amp-ton），C 和 Q 分别是两个途经的港口，即法国瑟堡（Cherbourg）和爱尔兰昆士敦（Queens town），在途经港口上船的乘客生还率更高些。
- SibSp 和 Parch 对生还率有一定的影响，船上父母子女或者兄弟姐妹和配偶数量较少的乘客生还率更高。

（3）两个以上变量的可视化。在 Seaborn 提供的许多绘图函数中，除了传入参数 x 和 y 之外，还可以传入参数 hue，就可以多增加一个可视化数据的维度。如下所示，我们使用 pointplot()(点图）可视化特征变量 Pclass、Sex 和目标变量 Survived 之间的关系。点图与条形图的主要差别在于，点图不绘制完整的条形，只绘制点估计值和置信区间（误差棒），并连接同一个类别变量的不同值的点，这样方便通过斜率来识别变化趋势的差异。

```
# Pclass 和 Sex 对生还率的联合影响
sns.pointplot(x = 'Pclass', y = 'Survived', hue = 'Sex', data = trainData)
```



从上图中可以看出，无论是在哪个阶层，女性的生存率都远大于男性，而且随着阶层的下降（即 Pclass 值的增加），男性和女性的生存率都是递减的。

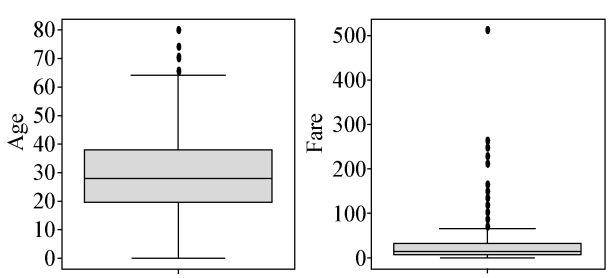
15.3.3　数据预处理

理解完数据之后，在构建机器学习模型之前，还需要对数据进行预处理，即对数据做清洗和转换，以便输入到机器学习模型中。常见的数据预处理包括：异常值处理、缺失值处理、特征编码转换和特征值缩放等。

（1）异常值处理。异常值，即数据集中不合理的值，又称离群点。

检测异常值的方法有很多。前面对列进行简单的描述统计分析，可以看出一列中是否存在异常值。另外，用于可视化数据分布概况的箱线图提供了一个识别异常值的标准，即大于或小于设定的上下界的数值为异常值。箱线图中上界和下界的估计一般分别为高于上四分位数和低于下四分位数 1.5 倍的上下四分位数间差距的值（1.5 是缺省的参数值，可以设为其他值）。如下所示，箱线图中最上面和最下面的两条直线就是估计的上下界，高于上界和低于下界的点被视为异常值。

```
# 用箱线图检查 Age 和 Fare 列中的异常值
fig,axes = plt.subplots(1, 2, figsize = (10,5))
sns.boxplot(y = 'Age', data = trainData, ax = axes[0])
sns.boxplot(y = 'Fare', data = trainData, ax = axes[1])
```



从上面分别关于 Age 列和 Fare 列数据分布的箱线图可以看出：

- 相比于 Age，Fare 列的数据更集中一些；
- Age 列有几个异常值，是年龄较大的乘客，但这也是正常的，不用处理；
- Fare 中有一个大于 500 的极端值，可以考虑处理。

如果发现有异常值，可以考虑删除该样例；或者把异常值视为缺失值，用下面所讲的填充缺失值的方法填充一个新值；也可以不处理。具体到 Fare 列中的大于 500 的异常值，我们可以不处理，或者将其替换为 Fare 列的中位数。

(2) 缺失值处理。大部分机器学习模型要求输入的特征列中不含空值，即缺失值。所以在数据清洗时，需要检查数据集中是否有缺失值，并做相应的处理。

前面我们通过 info() 函数查看每列的类型和非空值的个数等基本信息时，可以从每列中非空值的个数推断出该列是否有缺失值。也可用下面的方法直接显示每列缺失值的情况。

```
# 显示每列缺失值的个数
fullData.isnull().sum()
```

```
Age            263
Cabin         1014
Embarked         2
Fare             1
Name             0
Parch            0
PassengerId      0
Pclass           0
Sex              0
SibSp            0
Survived       418
Ticket           0
dtype: int64
```

可以看到，训练集和测试集中共有 4 列有缺失值，即 Age，Cabin，Embarked 和 Fare。其中，Cabin 中大部分值缺失，Embarked 和 Fare 只有一两个值缺失。

对于缺失值，一般有以下几种处理方法：

- 如果样本量很大，且只有很少的缺失值，可以考虑删除含缺失值的样例。
- 如果一个特征列包含太多缺失值，对预测又不是特别重要，可以考虑删除该特征列，例如本例中的 Cabin 列。
- 填充缺失值。缺失值的填充策略有很多种：

填充为随机值。这是最简单的一种策略。在一些应用中，将空值填充为随机值之后，并不改变其整体的数值分布。

填充为该列的均值、中位数或众数（即最频繁出现的值）。本例中，我们可以分别

使用均值和众数来填充 Age 和 Embarked 列的缺失值，即将缺失的年龄填充为所有乘客的平均年龄，缺失的登船港口填充为大部分乘客登船的港口。

填充为一个表示缺失的值，因为缺失本身也可能代表一些隐含信息。例如，本例中，舱位 Cabin 的缺失可能代表就没有船舱，所以我们可以用一个该列中不存在的值“U”来填充这些空值，来代表没有船舱这一信息。

填充为相似样例的该列值。这一策略比直接填充为所有样例的均值、中位数或众数更合理准确，但也更复杂。例如，本例中，我们可以从乘客的姓名中提取出其称谓，如 Mr.，Mrs.，Miss，Dr. 等，然后用具有相同称谓的乘客的平均年龄来填充缺失的 Age 值，这样肯定会更准确些。甚至还可以使用一种机器学习模型（如随机森林）来预测缺失的 Age 值，即考虑更多（甚至所有）特征的相似性来得到尽可能准确的 Age 值。

本例中对缺失值的处理如下所示：用一个特定值“U”填充 Cabin 列的缺失值；用均值填充 Age 列的缺失值；用中位数填充 Fare 列的缺失值；用众数填充 Embarked 列的缺失值。也可以尝试其他的处理策略。

```
# 用一个特定值'U'填充 Cabin 列的缺失值
fullData.Cabin.fillna('U', inplace = True)
```

```
# 用均值填充 Age 列的缺失值
fullData.Age.fillna(fullData.Age.mean(), inplace = True)
```

```
# 用中位数填充 Fare 列的缺失值
fullData.Fare.fillna(fullData.Fare.median(), inplace = True)
```

```
# 先用 value_counts 查看 Embarked 列中出现最频繁的值
fullData.Embarked.value_counts()
```

```
S    914
C    270
Q    123
Name: Embarked,dtype: int64
```

```
# 用最频繁的值'S'填充 Embarked 列的缺失值
fullData.Embarked.fillna('S', inplace = True)
```

最后，可以再使用 fullData.info() 或者 fullData.isnull().sum() 查看一下，确定已经没有缺失值了。

（3）特征编码转换。大部分机器学习模型只能处理数值型数据，所以非数值型特征需要转换为可供模型计算的数值型编码。

根据特征列的值是否为数字，可以将其分为数值型（包括整数和小数）和非数值型（如字符类型）两大类。另外，根据取值范围是有限的几个值还是无限个可能值，可以将特征列分为类别型（或者称为离散型）和连续型两种。例如，在泰坦尼克号数

据集中，除去 PassengerId 和 Survived 列，其余 10 列可以划分到下面四类中：

- 类别数值型：Pclass，SibSp，Parch；
- 类别非数值型：Sex，Embarked；
- 连续数值型：Age，Fare；
- 连续非数值型：Name，Cabin，Ticket。

连续非数值型特征常常是一些取值为字符串的变量，如这里的 Name，Cabin 和 Ticket，甚至有可能是长文本。要把这种数据转换为机器学习模型能接受的数值类型，一般要从这些列中进一步提取出一些离散特征，并编码成数字。我们将在下面的特征工程部分介绍转换 Name，Cabin 和 Ticket 列的具体方法。对于长文本类型列的转换，可以参考前面垃圾短信分类案例中的特征提取和转换方法。

连续数值型特征，如 Age 和 Fare，可以不做任何转换。如果所使用的机器学习模型是基于距离计算的，则需要做数值缩放，或者采用数值离散化（即分组）方法转换成类别型特征，以使其和其他特征列的取值范围相近。具体讨论见下面的特征值缩放部分。

类别非数值型特征，可以使用 Pandas 中的映射函数 map() 或者 Scikit-learn 中提供的 LabelEncoder 转换器先将其转换成数字。如下所示，Sex 列的两个值“male”和“female”分别转换成了 1 和 0。

```
# 将 Sex 列的值转换为 0 和 1
fullData.Sex = fullData.Sex.map({'female':0, 'male':1})
```

同样 Embarked 列也可以用类似的方法将三个字符值“S”“C”“Q”分别转换成三个数字。但是直接将类别值转换成几个数字通常并不足够，因为数字之间有大小和顺序关系，而它们所代表的类别值之间是无序的，也没有大小关系。因而，为了使模型中计算的距离更合理，一般应该将其转换为独热编码，即一个 N 值特征列被转换为 N 个二值（取值为 0 或 1）特征列，每列对应一个类别值。

可以使用 Pandas 中的 get_dummies() 函数或者 Scikit-learn 中提供的 OneHotEncoder 或 LabelBinarizer 转换器将类别型的特征列直接转换成独热编码。如下所示，Embarked 列被转换为独热编码。Embarked 列有三个不同取值，转换为独热编码后，生成了三个二值特征列 Embarked_C，Embarked_Q 和 Embarked_S，可以去掉第一列，因为它是冗余的，其值可以由其他两列值推断出。然后再把新生成的两列 Embarked_Q 和 Embarked_S 拼接到原数据集中，则原数据集变成了 14 列。

```
# 将 Embarked 列的值转换为独热编码
embDf = pd.get_dummies(fullData.Embarked, prefix = 'Embarked', drop_first = True)
# 将独热编码后的多个列拼接到原数据中
fullData = pd.concat([fullData, embDf], axis = 1)
```

Sex 列因为是二值的，直接被映射成 0 和 1 就已经是独热编码了。对于其他类别数

值型特征，要更准确地计算其类别间的距离，也需要将它们转换成独热编码，例如，Pclass 也可以按如下方法转换成独热编码。

```
# 将 Pclass 列的值转换为独热编码
pclassDf = pd.get_dummies(fullData.Pclass, prefix = 'Pclass', drop_first = True)
# 将独热编码后的多个列拼接到原数据中
fullData = pd.concat([fullData, pclassDf], axis = 1)
```

SibSp 和 Parch 列值都是同船亲人的个数，具有大小数量关系，所以不对它们进行独热编码转换了。

（4）特征值缩放。不同特征通常具有不同的取值范围，例如年龄一般在 0～100 岁，性别取值只有 0 和 1，而年收入可以是几万元、几十万元甚至几百万元。在计算距离时，取值范围大的特征会主导距离的计算结果，因而需要将各特征的取值范围缩放到相近的范围。特征缩放对一些基于距离计算的机器学习模型非常关键，如使用径向基核函数的支持向量机模型和 K 近邻模型等。使用决策树模型，一般无须对特征值进行缩放。

本例中，Age 和 Fare 列的取值范围基本都在几十左右，处于同一量级，相差不大，所以不对它们进行数值缩放。如果考虑到其他类别特征的取值都在 10 以内，可以考虑通过数值离散化（即分组）的方法将 Age 和 Fare 列的值也转换成 10 以内的离散值，不过这里不做这样的处理。

15.3.4　特征工程

处理好数据之后，需要选择出有助于预测生还率的一些特征作为机器学习模型的输入。可以从已有的特征中选择，也可以创建一些新的对预测有帮助的特征。

一、特征创建。通过对数据的仔细观察，我们创建出以下一些可能对预测生还率有帮助的新特征。

（1）称谓（Title）。首先，每个乘客姓名（Name）中都包含了具体的称谓或头衔，例如，“Braund, Mr. Owen Harris”“Heikkinen, Miss. Laina”“Fermina, Master. Michael J”等。称谓可以在一定程度上反映一个人的职业和年龄等。我们将这部分信息提取出来，作为非常有用的一个新变量“Title”。

```
# 从每个乘客的姓名中提取其称谓
fullData['Title'] = fullData['Name'].map(lambda x: x.split(',')[1].split('.')[0].strip())
fullData['Title'].value_counts()
```

```
Mr              757
Miss            260
Mrs             197
```

```
Master                61
Dr                     8
Rev                    8
Col                    4
Major                  2
Ms                     2
Mlle                   2
Mme                    1
Don                    1
Sir                    1
Jonkheer               1
Dona                   1
Lady                   1
Capt                   1
the Countess           1
Name: Title, dtype: int64
```

提取出来的称谓中包含英、法、西班牙语的不同叫法，我们把它们归成如下六类：Officer：政府官员，Royalty：王室，Mr：已婚男士，Mrs：已婚妇女，Miss：年轻未婚女子，Master：有技能的人/教师。

```
# 使用 map 函数将各种称谓转换成其相应的称谓类别
title_mapDict = {"Capt":      "Officer",    "Col":      "Officer",
                 "Major":     "Officer",    "Jonkheer":"Royalty",
                 "Don":       "Royalty",    "Sir":      "Royalty",
                 "Dr":        "Officer",    "Rev":      "Officer",
                 "the Countess":"Royalty",  "Dona":     "Royalty",
                 "Mme":       "Mrs",        "Mlle":     "Miss",
                 "Ms":        "Mrs",        "Mr":       "Mr",
                 "Mrs":       "Mrs",        "Miss":     "Miss",
                 "Master":    "Master",     "Lady":     "Royalty"}
fullData['Title'] = fullData['Title'].map(title_mapDict)
```

最后我们将具有六个类别值的 Title 列进行独热编码，并拼接到原数据集中。

```
# 使用 get_dummies 函数进行独热编码
titleDf = pd.get_dummies(fullData['Title'], prefix = 'Title', drop_first = True)
# 将独热编码后的多个列拼接到原数据中
fullData = pd.concat([fullData, titleDf], axis = 1)
```

（2）船舱类型。舱位（Cabin）列值的格式一般为一个大写字母，如 A 或 C，再加

上一个整数值（即具体的座位号）。第一个字母可以视为船舱类型，可能有助于预测乘客的生还率，所以我们把 Cabin 列值的首字母提取出来，将其映射为一个类别特征。

```
# 将 Cabin 列值映射为其首字母的值
fullData['Cabin'] = fullData['Cabin'].map(lambda x : x[0])
```

同样对映射后的 Cabin 列进行独热编码，并拼接到原数据集中。

```
# 使用 get_dummies 函数进行独热编码
cabinDf = pd.get_dummies(fullData['Cabin'], prefix = 'Cabin', drop_first = True)
# 将独热编码后的多个列拼接到原数据中
fullData = pd.concat([fullData, cabinDf], axis = 1)
```

（3）家庭人数和家庭类别。Parch 和 SibSp 两列都是表示家庭成员的数量的，将它们的值相加合成一列“FamilySize”，并且按照家庭人数的多少创建三个二值特征“Family_Single”（家庭人数等于 1）、“Family_Small”（家庭人数大于 1 小于 5）和“Family_Large”（家庭人数大于等于 5）。

```
# 创建新的特征 Family_Size
fullData['FamilySize'] = fullData['Parch'] + fullData['SibSp'] + 1
# 创建三个新的二值特征:Family_Single, Family_Small, Family_Large
fullData['Family_Single'] = fullData['FamilySize'].map( lambda s: 1 if s == 1 else 0)
fullData['Family_Small'] = fullData['FamilySize'].map( lambda s: 1 if 1<s<5 else 0)
fullData['Family_Large'] = fullData['FamilySize'].map( lambda s: 1 if s>= 5 else 0)
```

二、特征选择。在创建和提取特征时，我们尽可能多地提取出对预测有帮助的特征，但过多的特征会造成冗余、噪声、容易过拟合等问题，因此需要进行特征选择。特征选择可以加快模型的训练速度，同时还可以提升效果。

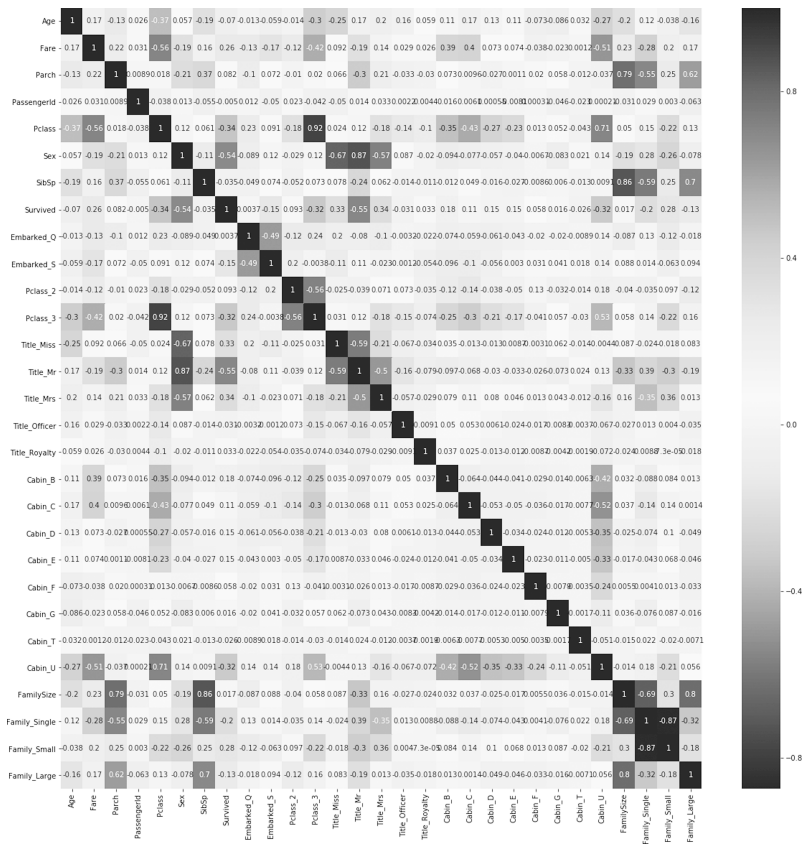
特征选择的方法有多种，最简单的是使用相关系数（correlation coefficient）。

Pandas 中提供的 corr() 方法可以计算出一个 DataFame 中所有数值型列两两之间的相关系数，缺省的是 Pearson 相关系数（即线性相关系数），还可以计算 Kendall 或 Spearman 等非线性相关系数，甚至是用户自定义的相关系数。

```
# 计算所有数值型特征列两两之间的线性相关性
corrDf = fullData.corr()
```

我们可以用 Seaborn 中的 heatmap()（热力图）将计算出的相关系数矩阵可视化出来。

```
# 可视化相关系数矩阵
plt.figure(figsize = (20, 20))
sns.heatmap(corrDf, cmap = plt.cm.RdBu, linecolor = 'white', annot = True)
```



Pearson 相关系数主要衡量两个变量之间的线性相关性，数值在－1 到 1 之间，绝对值越大，两个变量的线性相关性越大；越接近 0，线性相关性越小。相关系数矩阵是一个对称矩阵，对角线以下和以上的内容是完全一样的。特征选择时，我们主要关注两种情况：

- 特征变量和目标变量之间的相关度，可以看作该特征的重要度，绝对值越大越好。
- 特征和特征之间的相关度应该低，如果两个特征之间的相关性很高，就有可能存在冗余。

从上面显示的热力图我们可以观察到，整体上特征变量之间的相关性较低，除了几个比较显然的情况，例如性别（Sex）和称谓（Mr.，Mrs.，Miss）之间存在较强的相关性，船舱等级（Pclass）与票价（Fare）之间具有相关性，船舱等级（Pclass）与有无舱位号（Cabin_U）之间存在较强相关性，以及 Parch 和 SibSp 与它们所生成的几个有关家庭大小的变量（FamilySize，Family_Single，Family_Small，Family_Large）之间存在强相关性。我们在选择特征时可以考虑这些相关性。

本例中的目标变量是 Survived 列，我们着重查看其他变量与该变量之间的相关性，并按绝对值降序排列。

```
# 查看每个特征与 Survived 的相关系数,并按绝对值的降序排列
corrDf['Survived'].map(abs).sort_values(ascending = False)
```

```
Survived                1.000000
Title _ Mr              0.549199
Sex                     0.543351
Title _ Mrs             0.344935
Pclass                  0.338481
Title _ Miss            0.332795
Pclass _ 3              0.322308
Cabin _ U               0.316912
Family _ Small          0.279855
Fare                    0.257307
Family _ Single         0.203367
Cabin _ B               0.175095
Cabin _ D               0.150716
Embarked _ S            0.149683
Cabin _ E               0.145321
Family _ Large          0.125147
Cabin _ C               0.114652
Pclass _ 2              0.093349
Parch                   0.081629
Age                     0.070323
Cabin _ F               0.057935
SibSp                   0.035322
Title _ Royalty         0.033391
Title _ Officer         0.031316
Cabin _ T               0.026456
FamilySize              0.016639
Cabin _ G               0.016040
PassengerId             0.005007
Embarked _ Q            0.003650
Name: Survived, dtype: float64
```

一般地，Pearson 相关系数绝对值小于 0.02 表示极弱相关甚至不相关，所以这里只选择相关系数绝对值在 0.02 以上的特征。另外，考虑到特征之间的相关性，去掉一

些冗余列，如去掉 Pclass_2 和 Pclass_3，保留 Pclass；去掉 Parch 和 SibSp，保留 Family_Single，Family_Small 和 Family_Large。最后我们选择了 20 个特征作为输入机器学习模型的特征列。

```
# 筛选出最终输入模型的特征列和目标列
cols = corrDf['Survived'](abs(corrDf['Survived'])>0.02].index
fullNew = fullData[cols].drop(columns = ['Pclass_2', 'Pclass_3', 'Parch', 'SibSp'])
```

除了使用相关系数之外，还可以通过训练机器学习模型来筛选特征，比如逻辑回归、决策树、随机森林、梯度提升树等模型都可以输出特征的重要度。

最后，将准备好的数据集重新拆分成原来给定的训练集和测试集，并准备好供机器学习模型使用的特征列（X）和目标列（y）两部分。Survived 列是待预测的目标列，竞赛给定的测试数据集中没有 Survived 列。

```
# 拆分出原来的训练数据集和测试数据集
train_X = fullNew[:891].drop(columns = ['Survived'])
train_y = fullNew[:891].Survived
test_X = fullNew[891:].drop(columns = ['Survived'])
train_X.shape, train_y.shape, test_X.shape
```

```
((891,20),(891,),(418,20))
```

15.3.5 基本模型构建

用于分类的机器学习模型有很多种，这里实验四种常用的模型：逻辑回归、支持向量机、随机森林和梯度提升树模型。

我们使用交叉验证的方法来评估每个模型的泛化性能，即在未知数据上的表现。和单次划分训练集和测试集的评估方法不同，交叉验证是将数据集进行多次不同的划分，形成多个不同的训练集和测试集对，分别用它们训练模型并评估，然后取这多个评估结果的平均值作为该模型的评估结果，因而会更准确些。Scikit-learn 库中实现的最简单的交叉验证的方法是 cross_val_score。

下面我们先评估每种模型使用缺省参数的效果，然后再使用网格搜索方法自动寻找最佳的超参数值。首先导入要用到的所有类。

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
# 导入各个分类模型类
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

（1）使用缺省或固定参数。

```
# 用 5 折交叉验证一次评估四种模型的效果
models = [LogisticRegression(), SVC(), RandomForestClassifier(), GradientBoostingClassifier()]
names = ['LR: ', 'SVM: ', 'RF: ', 'GB: ']
for name, model in zip(names, models):
    score = cross_val_score(model, train_X, train_y, cv = 5)
    print(name, score.mean(), score)
```

```
LR:  0.823825842164674 [0.82122905 0.82122905 0.80337079 0.8258427 0.84745763]
SVM:  0.729643834800288 [0.6424581 0.75418994 0.71348315 0.75842697 0.77966102]
RF:  0.8216165263546502 [0.83798883 0.7877095 0.85393258 0.76404494 0.86440678]
GB:  0.8260539891842896 [0.82122905 0.82122905 0.83146067 0.8258427 0.83050847]
```

在使用缺省参数的各模型中，梯度提升树模型取得了最高分数：0.826。

（2）自动参数调优。每种模型都有一些超参数，可以通过调节超参数的值来寻找效果最优的模型。这一过程可以手动进行，即逐一设置不同的参数值，生成相应的模型，并得到其评估结果，最后再从中选出最优的模型。Scikit-learn 库中提供了一些方法可以自动化这一过程，如带交叉验证的网格搜索 GridSearchCV。给定多个参数的多个可能取值，GridSearchCV 自动为每种可能的参数值组合生成相应的模型，并用交叉验证的方法训练和评估该模型，然后选出最优的一个模型，并给出最优模型对应的参数值组合。

下面我们使用 GridSearchCV 来对每个模型自动寻找最优超参数，并显示最优模型的交叉验证得分。

```
# 逻辑回归模型
param_grid = {'C':[0.5,1.0,1.1,1.2,1.3,1.5], 'penalty':['l1', 'l2']}
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv = 5)
grid_search.fit(train_X, train_y)
grid_search.best_params_, grid_search.best_score_
```

```
({'C': 1.2, 'penalty': 'l1'}, 0.8271604938271605)
```

```
# 支持向量机模型
param_grid = {'C':[5,6,7,8], 'gamma':[0.005,0.01,0.015,0.02]}
grid_search = GridSearchCV(SVC(), param_grid, cv = 5)
grid_search.fit(train_X, train_y)
grid_search.best_params_, grid_search.best_score_
```

```
({'C': 7, 'gamma': 0.01}, 0.7845117845117845)
```

```
# 随机森林模型
param_grid = {'n_estimators':[450,480,485,490], 'max_depth':[7,8,9,10]}
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv = 5)
grid_search.fit(train_X, train_y)
grid_search.best_params_, grid_search.best_score_
```

```
({'max_depth': 8, 'n_estimators': 480}, 0.8395061728395061)
```

```
# 梯度提升树模型
param_grid = {'n_estimators':[110,120,130,140,150],
              'learning_rate':[0.1,0.11,0.12,0.15], 'max_depth':[3,4,5,6]}
grid_search = GridSearchCV(GradientBoostingClassifier(), param_grid, cv = 5)
grid_search.fit(train_X, train_y)
grid_search.best_params_, grid_search.best_score_
```

```
({'learning_rate': 0.11, 'max_depth': 4, 'n_estimators': 130}, 0.8439955106621774)
```

各模型经过调参后，性能都有所提升。相比之下，效果最好的还是梯度提升树模型，得分为 0.844。

15.3.6 模型集成

为了使预测的准确度更高一些，可以将多个不同的模型集成起来。集成多个异质学习器的方法通常有多数投票和堆叠。Scikit-learn 中提供了多数投票模型的实现，即 VotingClassifier。下面我们将前面四种最优模型用多数投票方式集成起来。

```
# 用多数投票模型集成前面四个最佳模型
from sklearn.ensemble import VotingClassifier

clf1 = LogisticRegression(C = 1.2, penalty = 'l1')
clf2 = SVC(C = 7, gamma = 0.01)
clf3 = RandomForestClassifier(n_estimators = 480, max_depth = 8)
clf4 = GradientBoostingClassifier(n_estimators = 110, learning_rate = 0.1, max_depth = 4)

vclf = VotingClassifier(estimators = [('LR', clf1), ('SVM', clf2), ('RF', clf3), ('GB', clf4)])

score = cross_val_score(vclf, train_X, train_y, cv = 5)
print(score.mean(), score)
```

```
0.8395122397542792 [0.84357542 0.83798883 0.84269663 0.8258427 0.84745763]
```

上面结果显示，用多数投票法融合多个不同机器学习模型的效果并没有超过单个模型，甚至低于单个模型的效果。除了多数投票法，还可以尝试多层堆叠的方法来集成多个模型。

15.3.7 模型使用

最后，将选择好的模型运用到竞赛给出的测试数据集上，得到预测结果，并将预测结果按照竞赛要求的格式存入一个外部 csv 文件中，就可以提交到 Kaggle 平台上做评测了。

```
# 用模型对测试数据进行预测
test_y = vclf.fit(train_X, train_y).predict(test_X).astype(int)
passenger_id = full[891:].PassengerId
# 将预测结果按所要求的格式,准备成一个用于提交的 csv 文件
submission = pd.DataFrame({'PassengerId': passenger_id, 'Survived': test_y})
submission.to_csv('titanic_pred.csv', index = False)
```

我们提交的用多数投票集成模型得到的预测结果在 Kaggle 平台上的得分为 0.775 11。我们也将其他四种网格搜索得到的最佳模型预测的结果提交到 Kaggle 平台上，得分如表 15－1 所示，可见在测试集上的得分不同于我们在训练集上做交叉验证得到的分数。

表 15－1　各模型得分

模型	Kaggle 评分
逻辑回归	0.775 11
支持向量机	0.717 70
随机森林	0.775 11
梯度提升树	0.736 84
多数投票模型	0.775 11

参考文献

[1] Titanic Data Science Solutions. https://www.kaggle.com/startupsci/titanic-data-science-solutions.

[2] An Interactive Data Science Tutorial. https://www.kaggle.com/helgejo/an-interactive-data-science-tutorial/data.

[3] https://blog.csdn.net/regina67/article/details/77940640.