本书中，我已经介绍了Python数据分析的编程基础。因为数据分析师和科学家总是在数据规整和准备上花费大量时间，这本书的重点在于掌握这些功能。

开发模型选用什么库取决于应用本身。许多统计问题可以用简单方法解决，比如普通的最小二乘回归，其它问题可能需要复杂的机器学习方法。幸运的是，Python已经成为了运用这些分析方法的语言之一，因此读完此书，你可以探索许多工具。

本章中，我会回顾一些pandas的特点，在你胶着于pandas数据规整和模型拟合和评分时，它们可能派上用场。然后我会简短介绍两个流行的建模工具，statsmodels和scikit-learn。这二者每个都值得再写一本书，我就不做全面的介绍，而是建议你学习两个项目的线上文档和其它基于Python的数据科学、统计和机器学习的书籍。

13.1 pandas与模型代码的接口

模型开发的通常工作流是使用pandas进行数据加载和清洗，然后切换到建模库进行建模。开发模型的重要一环是机器学习中的“特征工程”。它可以描述从原始数据集中提取信息的任何数据转换或分析，这些数据集可能在建模中有用。本书中学习的数据聚合和GroupBy工具常用于特征工程中。

优秀的特征工程超出了本书的范围，我会尽量直白地介绍一些用于数据操作和建模切换的方法。

pandas与其它分析库通常是靠NumPy的数组联系起来的。将DataFrame转换为NumPy数组，可以使用.values属性：

```
In [10]: import pandas as pd

In [11]: import numpy as np

In [12]: data = pd.DataFrame({
   ....:     'x0': [1, 2, 3, 4, 5],
   ....:     'x1': [0.01, -0.01, 0.25, -4.1, 0.],
   ....:     'y': [-1.5, 0., 3.6, 1.3, -2.]})

In [13]: data
Out[13]:
   x0    x1    y
0   1  0.01 -1.5
1   2 -0.01  0.0
2   3  0.25  3.6
3   4 -4.10  1.3
4   5  0.00 -2.0

In [14]: data.columns
Out[14]: Index(['x0', 'x1', 'y'], dtype='object')

In [15]: data.values
Out[15]:
array([[ 1.  ,  0.01, -1.5 ],
       [ 2.  , -0.01,  0.  ],
       [ 3.  ,  0.25,  3.6 ],
       [ 4.  , -4.1 ,  1.3 ],
       [ 5.  ,  0.  , -2.  ]])
```

要转换回DataFrame，可以传递一个二维ndarray，可带有列名：

```
In [16]: df2 = pd.DataFrame(data.values, columns=['one', 'two', 'three'])

In [17]: df2
Out[17]:
   one   two  three
0  1.0  0.01   -1.5
1  2.0 -0.01    0.0
2  3.0  0.25    3.6
3  4.0 -4.10    1.3
4  5.0  0.00   -2.0
```

> 笔记：最好当数据是均匀的时候使用.values属性。例如，全是数值类型。如果数据是不均匀的，结果会是Python对象的ndarray：
>
> ```
> In [18]: df3 = data.copy()
> ```

```
In [19]: df3['strings'] = ['a', 'b', 'c', 'd', 'e']

In [20]: df3
Out[20]:
  x0    x1    y strings
0   1  0.01 -1.5       a
1   2 -0.01  0.0       b
2   3  0.25  3.6       c
3   4 -4.10  1.3       d
4   5  0.00 -2.0       e

In [21]: df3.values
Out[21]:
array([[1, 0.01, -1.5, 'a'],
       [2, -0.01, 0.0, 'b'],
       [3, 0.25, 3.6, 'c'],
       [4, -4.1, 1.3, 'd'],
       [5, 0.0, -2.0, 'e']], dtype=object)
```

对于一些模型，你可能只想使用列的子集。我建议你使用loc，用values作索引：

```
In [22]: model_cols = ['x0', 'x1']

In [23]: data.loc[:, model_cols].values
Out[23]:
array([[ 1.  ,  0.01],
       [ 2.  , -0.01],
       [ 3.  ,  0.25],
       [ 4.  , -4.1 ],
       [ 5.  ,  0.  ]])
```

一些库原生支持pandas，会自动完成工作：从DataFrame转换到NumPy，将模型的参数名添加到输出表的列或Series。其它情况，你可以手工进行“元数据管理”。

在第12章，我们学习了pandas的Categorical类型和pandas.get_dummies函数。假设数据集中有一个非数值列：

```
In [24]: data['category'] = pd.Categorical(['a', 'b', 'a', 'a', 'b'],
   ....:                                   categories=['a', 'b'])

In [25]: data
Out[25]:
   x0    x1    y category
0   1  0.01 -1.5        a
1   2 -0.01  0.0        b
2   3  0.25  3.6        a
3   4 -4.10  1.3        a
4   5  0.00 -2.0        b
```

如果我们想替换category列为虚变量，我们可以创建虚变量，删除category列，然后添加到结果：

```
In [26]: dummies = pd.get_dummies(data.category, prefix='category')

In [27]: data_with_dummies = data.drop('category', axis=1).join(dummies)

In [28]: data_with_dummies
Out[28]:
   x0    x1    y  category_a  category_b
0   1  0.01 -1.5           1           0
1   2 -0.01  0.0           0           1
2   3  0.25  3.6           1           0
3   4 -4.10  1.3           1           0
4   5  0.00 -2.0           0           1
```

用虚变量拟合某些统计模型会有一些细微差别。当你不只有数字列时，使用Patsy（下一节的主题）可能更简单，更不容易出错。

13.2 用Patsy创建模型描述

Patsy是Python的一个库，使用简短的字符串“公式语法”描述统计模型（尤其是线性模型），可能是受到了R和S统计编程语言的公式语法的启发。

Patsy适合描述statsmodels的线性模型，因此我会关注于它的主要特点，让你尽快掌握。Patsy的公式是一个特殊的字符串语法，如下所示：

```
y ~ x0 + x1
```

a+b不是将a与b相加的意思，而是为模型创建的设计矩阵。patsy.dmatrices函数接收一个公式字符串和一个数据集（可以是DataFrame或数组的字典），为线性模型创建设计矩阵：

```
In [29]: data = pd.DataFrame({
   ....:     'x0': [1, 2, 3, 4, 5],
   ....:     'x1': [0.01, -0.01, 0.25, -4.1, 0.],
   ....:     'y': [-1.5, 0., 3.6, 1.3, -2.]})

In [30]: data
Out[30]:
   x0    x1    y
0   1  0.01 -1.5
1   2 -0.01  0.0
2   3  0.25  3.6
3   4 -4.10  1.3
4   5  0.00 -2.0

In [31]: import patsy

In [32]: y, X = patsy.dmatrices('y ~ x0 + x1', data)
```

现在有：

```
In [33]: y
Out[33]:
DesignMatrix with shape (5, 1)
     y
  -1.5
   0.0
   3.6
   1.3
  -2.0
  Terms:
    'y' (column 0)

In [34]: X
Out[34]:
DesignMatrix with shape (5, 3)
  Intercept  x0     x1
          1   1   0.01
          1   2  -0.01
          1   3   0.25
          1   4  -4.10
          1   5   0.00
  Terms:
    'Intercept' (column 0)
    'x0' (column 1)
    'x1' (column 2)
```

这些Patsy的DesignMatrix实例是NumPy的ndarray，带有附加元数据：

```
In [35]: np.asarray(y)
Out[35]:
array([[-1.5],
       [ 0. ],
       [ 3.6],
       [ 1.3],
       [-2. ]])

In [36]: np.asarray(X)
Out[36]:
array([[ 1.  ,  1.  ,  0.01],
       [ 1.  ,  2.  , -0.01],
       [ 1.  ,  3.  ,  0.25],
       [ 1.  ,  4.  , -4.1 ],
       [ 1.  ,  5.  ,  0.  ]])
```

你可能想Intercept是哪里来的。这是线性模型（比如普通最小二乘回归）的惯例用法。添加 +0 到模型可以不显示intercept：

```
In [37]: patsy.dmatrices('y ~ x0 + x1 + 0', data)[1]
Out[37]:
DesignMatrix with shape (5, 2)
  x0     x1
   1   0.01
   2  -0.01
   3   0.25
   4  -4.10
```

```
   5   0.00
  Terms:
    'x0' (column 0)
    'x1' (column 1)
```

Patsy对象可以直接传递到算法（比如numpy.linalg.lstsq）中，它执行普通最小二乘回归：

```
In [38]: coef, resid, _, _ = np.linalg.lstsq(X, y)
```

模型的元数据保留在design_info属性中，因此你可以重新附加列名到拟合系数，以获得一个Series，例如：

```
In [39]: coef
Out[39]:
array([[ 0.3129],
       [-0.0791],
       [-0.2655]])

In [40]: coef = pd.Series(coef.squeeze(), index=X.design_info.column_names)

In [41]: coef
Out[41]:
Intercept    0.312910
x0          -0.079106
x1          -0.265464
dtype: float64
```

用Patsy公式进行数据转换

你可以将Python代码与patsy公式结合。在评估公式时，库将尝试查找在封闭作用域内使用的函数：

```
In [42]: y, X = patsy.dmatrices('y ~ x0 + np.log(np.abs(x1) + 1)', data)

In [43]: X
Out[43]:
DesignMatrix with shape (5, 3)
  Intercept  x0  np.log(np.abs(x1) + 1)
          1   1                 0.00995
          1   2                 0.00995
          1   3                 0.22314
          1   4                 1.62924
          1   5                 0.00000
  Terms:
    'Intercept' (column 0)
    'x0' (column 1)
    'np.log(np.abs(x1) + 1)' (column 2)
```

常见的变量转换包括标准化（平均值为0，方差为1）和中心化（减去平均值）。Patsy有内置的函数进行这样的工作：

```
In [44]: y, X = patsy.dmatrices('y ~ standardize(x0) + center(x1)', data)

In [45]: X
Out[45]:
DesignMatrix with shape (5, 3)
  Intercept  standardize(x0)  center(x1)
          1         -1.41421        0.78
          1         -0.70711        0.76
          1          0.00000        1.02
          1          0.70711       -3.33
          1          1.41421        0.77
  Terms:
    'Intercept' (column 0)
    'standardize(x0)' (column 1)
    'center(x1)' (column 2)
```

作为建模的一步，你可能拟合模型到一个数据集，然后用另一个数据集评估模型。另一个数据集可能是剩余的部分或是新数据。当执行中心化和标准化转变，用新数据进行预测要格外小心。因为你必须使用平均值或标准差转换新数据集，这也称作状态转换。

patsy.build_design_matrices函数可以使用原始样本数据集的保存信息，来转换新数据，：

```
In [46]: new_data = pd.DataFrame({
   ....:     'x0': [6, 7, 8, 9],
   ....:     'x1': [3.1, -0.5, 0, 2.3],
   ....:     'y': [1, 2, 3, 4]})
```

```
In [47]: new_X = patsy.build_design_matrices([X.design_info], new_data)

In [48]: new_X
Out[48]:
[DesignMatrix with shape (4, 3)
   Intercept  standardize(x0)  center(x1)
           1          2.12132        3.87
           1          2.82843        0.27
           1          3.53553        0.77
           1          4.24264        3.07
   Terms:
     'Intercept' (column 0)
     'standardize(x0)' (column 1)
     'center(x1)' (column 2)]
```

因为Patsy中的加号不是加法的意义，当你按照名称将数据集的列相加时，你必须用特殊I函数将它们封装起来：

```
In [49]: y, X = patsy.dmatrices('y ~ I(x0 + x1)', data)

In [50]: X
Out[50]:
DesignMatrix with shape (5, 2)
  Intercept  I(x0 + x1)
          1        1.01
          1        1.99
          1        3.25
          1       -0.10
          1        5.00
  Terms:
    'Intercept' (column 0)
    'I(x0 + x1)' (column 1)
```

Patsy的patsy.builtins模块还有一些其它的内置转换。请查看线上文档。

分类数据有一个特殊的转换类，下面进行讲解。

分类数据和Patsy

非数值数据可以用多种方式转换为模型设计矩阵。完整的讲解超出了本书范围，最好和统计课一起学习。

当你在Patsy公式中使用非数值数据，它们会默认转换为虚变量。如果有截距，会去掉一个，避免共线性：

```
In [51]: data = pd.DataFrame({
   ....:     'key1': ['a', 'a', 'b', 'b', 'a', 'b', 'a', 'b'],
   ....:     'key2': [0, 1, 0, 1, 0, 1, 0, 0],
   ....:     'v1': [1, 2, 3, 4, 5, 6, 7, 8],
   ....:     'v2': [-1, 0, 2.5, -0.5, 4.0, -1.2, 0.2, -1.7]
   ....: })

In [52]: y, X = patsy.dmatrices('v2 ~ key1', data)

In [53]: X
Out[53]:
DesignMatrix with shape (8, 2)
  Intercept  key1[T.b]
          1          0
          1          0
          1          1
          1          1
          1          0
          1          1
          1          0
          1          1
  Terms:
    'Intercept' (column 0)
    'key1' (column 1)
```

如果你从模型中忽略截距，每个分类值的列都会包括在设计矩阵的模型中：

```
In [54]: y, X = patsy.dmatrices('v2 ~ key1 + 0', data)

In [55]: X
Out[55]:
DesignMatrix with shape (8, 2)
  key1[a]  key1[b]
        1        0
```

```
        1         0
        0         1
        0         1
        1         0
        0         1
        1         0
        0         1
  Terms:
    'key1' (columns 0:2)
```

使用C函数，数值列可以截取为分类量：

```
In [56]: y, X = patsy.dmatrices('v2 ~ C(key2)', data)

In [57]: X
Out[57]:
DesignMatrix with shape (8, 2)
  Intercept  C(key2)[T.1]
          1             0
          1             1
          1             0
          1             1
          1             0
          1             1
          1             0
          1             0
  Terms:
    'Intercept' (column 0)
    'C(key2)' (column 1)
```

当你在模型中使用多个分类名，事情就会变复杂，因为会包括key1:key2形式的相交部分，它可以用在方差（ANOVA）模型分析中：

```
In [58]: data['key2'] = data['key2'].map({0: 'zero', 1: 'one'})

In [59]: data
Out[59]:
  key1  key2  v1   v2
0    a  zero   1 -1.0
1    a   one   2  0.0
2    b  zero   3  2.5
3    b   one   4 -0.5
4    a  zero   5  4.0
5    b   one   6 -1.2
6    a  zero   7  0.2
7    b  zero   8 -1.7

In [60]: y, X = patsy.dmatrices('v2 ~ key1 + key2', data)

In [61]: X
Out[61]:
DesignMatrix with shape (8, 3)
  Intercept  key1[T.b]  key2[T.zero]
          1          0             1
          1          0             0
          1          1             1
          1          1             0
          1          0             1
          1          1             0
          1          0             1
          1          1             1
  Terms:
    'Intercept' (column 0)
    'key1' (column 1)
    'key2' (column 2)

In [62]: y, X = patsy.dmatrices('v2 ~ key1 + key2 + key1:key2', data)

In [63]: X
Out[63]:
DesignMatrix with shape (8, 4)
  Intercept  key1[T.b]  key2[T.zero]
key1[T.b]:key2[T.zero]
          1          0             1                        0
          1          0             0                        0
          1          1             1                        1
          1          1             0                        0
          1          0             1                        0
          1          1             0                        0
          1          0             1                        0
          1          1             1                        1
  Terms:
    'Intercept' (column 0)
```

```
    'key1' (column 1)
    'key2' (column 2)
    'key1:key2' (column 3)
```

Patsy提供转换分类数据的其它方法，包括以特定顺序转换。请参阅线上文档。

13.3 statsmodels介绍

statsmodels是Python进行拟合多种统计模型、进行统计试验和数据探索可视化的库。Statsmodels包含许多经典的统计方法，但没有贝叶斯方法和机器学习模型。

statsmodels包含的模型有：

- 线性模型，广义线性模型和健壮线性模型
- 线性混合效应模型
- 方差（ANOVA）方法分析
- 时间序列过程和状态空间模型
- 广义矩估计

下面，我会使用一些基本的statsmodels工具，探索Patsy公式和pandasDataFrame对象如何使用模型接口。

估计线性模型

statsmodels有多种线性回归模型，包括从基本（比如普通最小二乘）到复杂（比如迭代加权最小二乘法）的。

statsmodels的线性模型有两种不同的接口：基于数组和基于公式。它们可以通过API模块引入：

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

为了展示它们的使用方法，我们从一些随机数据生成一个线性模型：

```
def dnorm(mean, variance, size=1):
    if isinstance(size, int):
        size = size,
    return mean + np.sqrt(variance) * np.random.randn(*size)

# For reproducibility
np.random.seed(12345)

N = 100
X = np.c_[dnorm(0, 0.4, size=N),
          dnorm(0, 0.6, size=N),
          dnorm(0, 0.2, size=N)]
eps = dnorm(0, 0.1, size=N)
beta = [0.1, 0.3, 0.5]

y = np.dot(X, beta) + eps
```

这里，我使用了“真实”模型和可知参数beta。此时，dnorm可用来生成正态分布数据，带有特定均值和方差。现在有：

```
In [66]: X[:5]
Out[66]:
array([[-0.1295, -1.2128,  0.5042],
       [ 0.3029, -0.4357, -0.2542],
       [-0.3285, -0.0253,  0.1384],
       [-0.3515, -0.7196, -0.2582],
       [ 1.2433, -0.3738, -0.5226]])

In [67]: y[:5]
Out[67]: array([ 0.4279, -0.6735, -0.0909, -0.4895,-0.1289])
```

像之前Patsy看到的，线性模型通常要拟合一个截距。sm.add_constant函数可以添加一个截距的列到现存的矩阵：

```
In [68]: X_model = sm.add_constant(X)

In [69]: X_model[:5]
Out[69]:
array([[ 1.    , -0.1295, -1.2128,  0.5042],
       [ 1.    ,  0.3029, -0.4357, -0.2542],
       [ 1.    , -0.3285, -0.0253,  0.1384],
       [ 1.    , -0.3515, -0.7196, -0.2582],
       [ 1.    ,  1.2433, -0.3738, -0.5226]])
```

sm.OLS类可以拟合一个普通最小二乘回归：

```
In [70]: model = sm.OLS(y, X)
```

这个模型的fit方法返回了一个回归结果对象，它包含估计的模型参数和其它内容：

```
In [71]: results = model.fit()

In [72]: results.params
Out[72]: array([ 0.1783,  0.223 ,  0.501 ])
```

对结果使用summary方法可以打印模型的详细诊断结果：

```
In [73]: print(results.summary())
OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.430
Model:                            OLS   Adj. R-squared:                  0.413
Method:                 Least Squares   F-statistic:                     24.42
Date:                Mon, 25 Sep 2017   Prob (F-statistic):           7.44e-12
Time:                        14:06:15   Log-Likelihood:                -34.305
No. Observations:                 100   AIC:                             74.61
Df Residuals:                      97   BIC:                             82.42
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.1783      0.053      3.364      0.001       0.073       0.283
x2             0.2230      0.046      4.818      0.000       0.131       0.315
x3             0.5010      0.080      6.237      0.000       0.342       0.660
==============================================================================
Omnibus:                        4.662   Durbin-Watson:                   2.201
Prob(Omnibus):                  0.097   Jarque-Bera (JB):                4.098
Skew:                           0.481   Prob(JB):                        0.129
Kurtosis:                       3.243   Cond. No.
1.74
==============================================================================
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

这里的参数名为通用名x1, x2等等。假设所有的模型参数都在一个DataFrame中：

```
In [74]: data = pd.DataFrame(X, columns=['col0', 'col1', 'col2'])

In [75]: data['y'] = y

In [76]: data[:5]
Out[76]:
       col0      col1      col2         y
0 -0.129468 -1.212753  0.504225  0.427863
1  0.302910 -0.435742 -0.254180 -0.673480
2 -0.328522 -0.025302  0.138351 -0.090878
3 -0.351475 -0.719605 -0.258215 -0.489494
4  1.243269 -0.373799 -0.522629 -0.128941
```

现在，我们使用statsmodels的公式API和Patsy的公式字符串：

```
In [77]: results = smf.ols('y ~ col0 + col1 + col2', data=data).fit()

In [78]: results.params
Out[78]:
Intercept    0.033559
col0         0.176149
col1         0.224826
col2         0.514808
dtype: float64

In [79]: results.tvalues
```

```
Out[79]:
Intercept    0.952188
col0         3.319754
col1         4.850730
col2         6.303971
dtype: float64
```

观察下statsmodels是如何返回Series结果的，附带有DataFrame的列名。当使用公式和pandas对象时，我们不需要使用add_constant。

给出一个样本外数据，你可以根据估计的模型参数计算预测值：

```
In [80]: results.predict(data[:5])
Out[80]:
0   -0.002327
1   -0.141904
2    0.041226
3   -0.323070
4   -0.100535
dtype: float64
```

statsmodels的线性模型结果还有其它的分析、诊断和可视化工具。除了普通最小二乘模型，还有其它的线性模型。

估计时间序列过程

statsmodels的另一模型类是进行时间序列分析，包括自回归过程、卡尔曼滤波和其它态空间模型，和多元自回归模型。

用自回归结构和噪声来模拟一些时间序列数据：

```
init_x = 4

import random
values = [init_x, init_x]
N = 1000

b0 = 0.8
b1 = -0.4
noise = dnorm(0, 0.1, N)
for i in range(N):
    new_x = values[-1] * b0 + values[-2] * b1 + noise[i]
    values.append(new_x)
```

这个数据有AR(2)结构（两个延迟），参数是0.8和-0.4。拟合AR模型时，你可能不知道滞后项的个数，因此可以用较多的滞后量来拟合这个模型：

```
In [82]: MAXLAGS = 5

In [83]: model = sm.tsa.AR(values)

In [84]: results = model.fit(MAXLAGS)
```

结果中的估计参数首先是截距，其次是前两个参数的估计值：

```
In [85]: results.params
Out[85]: array([-0.0062,  0.7845, -0.4085, -0.0136,  0.015 ,  0.0143])
```

更多的细节以及如何解释结果超出了本书的范围，可以通过statsmodels文档学习更多。

13.4 scikit-learn介绍

scikit-learn是一个广泛使用、用途多样的Python机器学习库。它包含多种标准监督和非监督机器学习方法和模型选择和评估、数据转换、数据加载和模型持久化工具。这些模型可以用于分类、聚合、预测和其它任务。

机器学习方面的学习和应用scikit-learn和TensorFlow解决实际问题的线上和纸质资料很多。本节中，我
scikit-learn API

会简要介绍 的风格。

写作此书的时候，scikit-learn并没有和pandas深度结合，但是有些第三方包在开发中。尽管如此，pandas非常适合在模型拟合前处理数据集。

举个例子，我用一个Kaggle竞赛的经典数据集，关于泰坦尼克号乘客的生还率。我们用pandas加载测试和训练数据集：

```
In [86]: train = pd.read_csv('datasets/titanic/train.csv')

In [87]: test = pd.read_csv('datasets/titanic/test.csv')

In [88]: train[:4]
Out[88]:
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
```

statsmodels和scikit-learn通常不能接收缺失数据，因此我们要查看列是否包含缺失值：

```
In [89]: train.isnull().sum()
Out[89]:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64

In [90]: test.isnull().sum()
Out[90]:
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

在统计和机器学习的例子中，根据数据中的特征，一个典型的任务是预测乘客能否生还。模型现在训练数据集中拟合，然后用样本外测试数据集评估。

我想用年龄作为预测值，但是它包含缺失值。缺失数据补全的方法有多种，我用的是一种简单方法，用训练数据集的中位数补全两个表的空值：

```
In [91]: impute_value = train['Age'].median()

In [92]: train['Age'] = train['Age'].fillna(impute_value)

In [93]: test['Age'] = test['Age'].fillna(impute_value)
```

现在我们需要指定模型。我增加了一个列IsFemale，作为“Sex”列的编码：

```
In [94]: train['IsFemale'] = (train['Sex'] == 'female').astype(int)

In [95]: test['IsFemale'] = (test['Sex'] == 'female').astype(int)
```

然后，我们确定一些模型变量，并创建NumPy数组：

```
In [96]: predictors = ['Pclass', 'IsFemale', 'Age']

In [97]: X_train = train[predictors].values

In [98]: X_test = test[predictors].values

In [99]: y_train = train['Survived'].values

In [100]: X_train[:5]
Out[100]:
array([[  3.,   0.,  22.],
       [  1.,   1.,  38.],
       [  3.,   1.,  26.],
       [  1.,   1.,  35.],
       [  3.,   0.,  35.]])

In [101]: y_train[:5]
Out[101]: array([0, 1, 1, 1, 0])
```

我不能保证这是一个好模型，但它的特征都符合。我们用scikit-learn的LogisticRegression模型，创建一个模型实例：

```
In [102]: from sklearn.linear_model import LogisticRegression

In [103]: model = LogisticRegression()
```

与statsmodels类似，我们可以用模型的fit方法，将它拟合到训练数据：

```
In [104]: model.fit(X_train, y_train)
Out[104]:
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

现在，我们可以用model.predict，对测试数据进行预测：

```
In [105]: y_predict = model.predict(X_test)

In [106]: y_predict[:10]
Out[106]: array([0, 0, 0, 0, 1, 0, 1, 0, 1, 0])
```

如果你有测试数据集的真是值，你可以计算准确率或其它错误度量值：

```
(y_true == y_predict).mean()
```

在实际中，模型训练经常有许多额外的复杂因素。许多模型有可以调节的参数，有些方法（比如交叉验证）可以用来进行参数调节，避免对训练数据过拟合。这通常可以提高预测性或对新数据的健壮性。

交叉验证通过分割训练数据来模拟样本外预测。基于模型的精度得分（比如均方差），可以对模型参数进行网格搜索。有些模型，如logistic回归，有内置的交叉验证的估计类。例如，logisticregressioncv类可以用一个参数指定网格搜索对模型的正则化参数C的粒度：

```
In [107]: from sklearn.linear_model import LogisticRegressionCV

In [108]: model_cv = LogisticRegressionCV(10)

In [109]: model_cv.fit(X_train, y_train)
Out[109]:
LogisticRegressionCV(Cs=10, class_weight=None, cv=None, dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
           refit=True, scoring=None, solver='lbfgs', tol=0.0001, verbose=0)
```

要手动进行交叉验证，你可以使用cross_val_score帮助函数，它可以处理数据分割。例如，要交叉验证我们的带有四个不重叠训练数据的模型，可以这样做：

```
In [110]: from sklearn.model_selection import cross_val_score
```

```
In [111]: model = LogisticRegression(C=10)

In [112]: scores = cross_val_score(model, X_train, y_train, cv=4)

In [113]: scores
Out[113]: array([ 0.7723,  0.8027,  0.7703,  0.7883])
```

默认的评分指标取决于模型本身，但是可以明确指定一个评分。交叉验证过的模型需要更长时间来训练，但会有更高的模型性能。

13.5 继续学习

我只是介绍了一些Python建模库的表面内容，现在有越来越多的框架用于各种统计和机器学习，它们都是用Python或Python用户界面实现的。

这本书的重点是数据规整，有其它的书是关注建模和数据科学工具的。其中优秀的有：

- Andreas Mueller and Sarah Guido (O'Reilly)的 《Introduction to Machine Learning with Python》
- Jake VanderPlas (O'Reilly)的 《Python Data Science Handbook》
- Joel Grus (O'Reilly) 的 《Data Science from Scratch: First Principles》
- Sebastian Raschka (Packt Publishing) 的《Python Machine Learning》
- Aurélien Géron (O'Reilly) 的《Hands-On Machine Learning with Scikit-Learn and TensorFlow》

虽然书是学习的好资源，但是随着底层开源软件的发展，书的内容会过时。最好是不断熟悉各种统计和机器学习框架的文档，学习最新的功能和API。