3.7. Python 编程实践

▲3.6.机器学习面临的挑战

▼3.8.继续学习本章知识

Python 编程实践

【分析对象】
CSV 文件—文件名为“bc_data.csv”，数据内容来自“威斯康星乳腺癌数据库 (Wisconsin Breast Cancer Database)”，该数据集主要记录了569个病例的32个属性。主要属性如下。id:病例的id; diagnosis(诊断结论):M 为恶性，B 为良性。该数据集共包含357 个良性病 例和212 个恶性病例; 细胞核的10 个特征值，包括半径(Radius)、纹理(Texture)、周长(Perimeter)、面积(Area)、平滑度(Smoothness)、紧凑度(Compactness)、凹面(Concavity)、凹点(Concave points)、对称性(Symmetry)和分形维数(Fractal Dimension) 等。同时，为上述10个特征值分别供了三种统计量，分别为均值(Mean)、 标准差(Standard Error)和 “最大值”(Worst or Largest)。

【分析目的与任务】
理解机器学习算法在数据科学中的应用—以 KNN 算法进行分类分析。
首先，以随机选择的部分记录为训练集进行学习“诊断结论(diagnosis)”的概念。其次，以剩余记录为测试集，进行KNN 建模。接着，按KNN 模型预测测试集的dignosis 类型。最后，将KNN 模型给出的diagnosis“预测类型”与数据集bc_data.csv 自带的“实际类型”进行对比分析，验证KNN 建模的有效性。

【分析方法及工具】 Python 及scikit-learn 包。

Python 编程实践

【主要步骤】

- 基于机器学习的数据科学项目也需要进行、数据读入、数据理解和数据准备活动于本书第二章的“2.7 Python 编程实践”相似。因此，本例中不再详细述以上活动的编程方法。需要了解相关知识的读者可以参考本书第2章的例题。

- 实现该数据科学项目的步骤除以上活动外，还需有算法选择及其超级参数的设置、具体模型的训练、用模型进行预测、模型的评价和模型的应用优化等步骤。

Step 1:数据读入

数据读入操作需要设置或(和)查看当前工作目录，而其的实现通常分别采用Python 模块os 中的函数chdir()和getcwd()。示例如下。

```
In[1] #导入所需 Python 包

      import pandas as pd
      import numpy as np import os

      #设置/更改当前工作目录 os.chdir(r'C:\Users\soloman\clm‘)
      #【提示】此处，路径“C:\Users\soloman\clm”可以用户自行设置

      #查看当前工作目录
      print(os.getcwd())
```

对应输出结果为：

```
C:\Users\soloman\clm
```

接着，可以用 Python 第三方包 Pandas 提供的函数 read_csv()将当前工作目录中的数据文件 bc_data.csv 读入至 Pandas 数据框 bc_data，并用 head()函数显示其前 5 行。

In[2]:
```
#将当前工作目录中的数据文件 bc_data.csv 读入至 Pandas 数据框
 bc_data 中 bc_data = pd.read_csv('bc_data.csv', header=0)
#【提示】由于目标数据'bc_data.csv'中没有列名信息，header 设置为 0



#查看数据框 bc_data 的前 5 行
 bc_data.head(5)
```

对应输出结果为：

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280

5 rows × 32 columns

Step 2:数据理解

基于机器学习方法进行数据科学任务也需要以数据理解为前提条件。在数据科学中，常用于数据理解的方法有查看数据形状、查看列名、进行描述性统计等，详见本书第2 章的“2.7 Python 编程实践”中的讲解。
其中**查看形状**的代码示例如下。

```
In[3]|#查看形状
     | print(bc_data.shape)
```

对应输出结果为：

```
(569, 32)
```

从输出结果可以看出，数据框 bc_data 的行数和列数分别为 569 和 32。

查看列名

接下来， 我们可以用 Pandas 数据框的 columns 属性查看其列名。示例如下。

```
In[4]: #查看列名

       print(bc_data.columns)
```

对应输出结果为：

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean',
'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'texture_ se', 'perimeter_se', 'area_se',
'smoothness_se', 'compactness_se', 'concavity_ se', 'concave points_se',
'symmetry_se','fractal_dimension_se', 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst',
'smoothness_worst', 'compactness_ worst', 'concavity_worst', 'concave_points_worst', 'symmetry_
worst', 'fractal_
dimension_worst'], dtype='object')
```

以上数据结果依次显示了**数据框 bc_data 的 32 个列的名称**，其中，列名 id 的含义为“病例 id”，Diagnosis 的含义为“诊断结论其余 30 个列名的含义为细胞核的 10个特征值的均值（Mean）、标准差（Standard Error）和最大值（Worst or Largest）3 种 统计量。

进行描述性统计

接着，采用 Pandas 数据框的 describe()方法对数据框 bc_data 进行描述性统计。示例如下。

```
In[5] #进行描述性统计
      print(bc_data.describe())
```

对应输出结果为：

```
                 id  radius_mean  texture_mean  perimeter_mean    area_mean  \
count  5.690000e+02   569.000000    569.000000      569.000000   569.000000
mean   3.037183e+07    14.127292     19.289649       91.969033   654.889104
std    1.250206e+08     3.524049      4.301036       24.298981   351.914129
min    8.670000e+03     6.981000      9.710000       43.790000   143.500000
25%    8.692180e+05    11.700000     16.170000       75.170000   420.300000
50%    9.060240e+05    13.370000     18.840000       86.240000   551.100000
75%    8.813129e+06    15.780000     21.800000      104.100000   782.700000
max    9.113205e+08    28.110000     39.280000      188.500000  2501.000000

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200

       symmetry_mean            ...             radius_worst  texture_worst  \
count     569.000000            ...               569.000000     569.000000
mean        0.181162            ...                16.269190      25.677223
std         0.027414            ...                 4.833242       6.146258
min         0.106000            ...                 7.930000      12.020000
25%         0.161900            ...                13.010000      21.080000
50%         0.179200            ...                14.970000      25.410000
```

上述输出结果依次显示了数据框 bc_data 中每个列所对应的行数（Count）、均值（Mean）、标准差（Std）、最小值（Min）、上四分位数（25%）、中位数（50%）、下四分位数（75%）和最大值（max）。

Step 3:数据准备

- 与基于统计学的数据科学项目类似，基于机器学习的数据科学项目也需要进行**数据准备或数据预处理活动**。
- 但是，其区别为二者的数据准备方法不同。在统计学中需要将数据分为特征矩阵和目标向量。
- 然而，在机器学习中需要将数据集划分为训练集和测试集两部分或划分成**训练集、测试集和验证集**3部分。
- 此外，数据准备中还需要进行**清洗和ETL转换**等。

(1)数据清洗

数据清洗代码示例如下。

In[6]
```
#数据清洗：本例中没有实际意义的 id 项数据，可以考虑删除
data = bc_data.drop(['id'], axis=1)
#显示数据框 data 的前 5 行
print(data.head())
```

对应输出结果为：

```
  diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0         M        17.99         10.38          122.80     1001.0
1         M        20.57         17.77          132.90     1326.0
2         M        19.69         21.25          130.00     1203.0
3         M        11.42         20.38           77.58      386.1
4         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   symmetry_mean            ...             radius_worst  texture_worst  \
0         0.2419            ...                    25.38          17.33
1         0.1812            ...                    24.99          23.41
2         0.2069            ...                    23.57          25.53
3         0.2597            ...                    14.91          26.50
4         0.1809            ...                    22.54          16.67

   perimeter_worst  area_worst  smoothness_worst  compactness_worst  \
0           184.60      2019.0            0.1622             0.6656
1           158.80      1956.0            0.1238             0.1866
2           152.50      1709.0            0.1444             0.4245
3            98.87       567.7            0.2098             0.8663
4           152.20      1575.0            0.1374             0.2050
```

以上输出结果显示了bc_data 的前5 行数据，其中省略号(...)表示由于显示空间所限，此处省略了部分列的显示。

（2）定义特征矩阵 X_data

此外，从该输出结果还可以看出，id 列已经从数 据框bc_data 删除。示例如下。

In[7]:
```
#定义特征矩阵 X_data
X_data = data.drop(['diagnosis'], axis=1)
#【提示】axis=1 的含义为：①行数不变；②按行为单位计算；③逐行计算

#显示数据框 X_data 的前 5 行
X_data.head()
```

对应输出结果：

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 30 columns

从以上输出结果可以看出，数据框X_data 中已将因变量diagnosis 删除。

（3）定义目标向量

接下来， 用 Numpy 的 ravel()方法将因变量 dianosis 对应的列进行降维处理，将其转换为目标向量。示例如下。

In[8]:
```
#定义目标向量
y_data = np.ravel(data[['diagnosis']])
#【提示】在数据分析与数据科学项目中，可以用 np.ravel()进行降维处理
y_data[0:6]
```

对应输出结果：

```
array(['M', 'M', 'M', 'M', 'M', 'M'], dtype=object)
```

（4）测试数据与训练数据的拆分方法

通常，我们采用 Python 第三方包 sklearn.model_selection 中的函数 train_test_split()可以轻松实现数据集的划分工作。示例如下。

In[9]:
```
#测试数据与训练数据的拆分方法：用第三方包 sklearn.model_selection 中的函数 train_test_split()
from sklearn.model_selection import train_test_split
X_trainingSet, X_testSet, y_trainingSet, y_testSet = train_test_
split(X_data, y_data, random_state=1)
#【提示】X_trainingSet 和 y_trainingSet 分别为训练集的特征矩阵和目标向量
#【提示】X_testSet 和 y_testSet 分别为测试集的特征矩阵和目标向量
```

（5）查看训练集的形状

接着，可以采用 Pandas 包中的 shape 属性查看训练集和测试集的形状，即所包含的行数和列数。示例如下。

```
In[10] | # 查看训练集的形状
       | print(X_trainingSet.shape)
```

对应输出结果为：

```
(426, 30)
```

从以上输出结果可以看出，训练集（X_trainingSet）的行数和列数分别为 426 个和 30 个。接着，我们继续查看测试集（X_testSet）的行数和列数。示例如下。

```
In[11] | # 查看训练集的形状
       | print(X_testSet.shape)
```

对应输出结果为：

```
(143, 30)
```

Step 4:算法选择及其超级参数的设置

Python 的第三方包 scikit-learn 提供了能够支持不同机器学习算法的具体函数。以 KNN 算法为例，scikit-learn 包中提供了函数 KNeighborsClassifier()。因此，我们可以通过调用函数KNeighborsClassifier 实现KNN 算法的调用。示例如下。

In[12]
```
#选择算法：本例选用 KNN 算法，需要导入 KNeighborsClassifier 分类器
from sklearn.neighbors import KNeighborsClassifier
```

导入KNeighborsClassifier 分类器后，可通过调用函数KNeighborsClassifier()，生成一个KNN 模型的实例，如cancerModel。示例如下。

In[13]
```
#实例化 KNN 模型，并设置超级参数 algorithm='kd_tree'

cancerModel= KNeighborsClassifier(algorithm='kd_tree')
#【提示】：algorithm 为计算最邻近的算法，可取值如 ball_tree、kd_tree、brute 或自动选择 auto
```

Step 5:具体模型的训练

与第2 章的“2.7 Python 编程实践”中介绍的统计模型的训练方式类似，基于scikit- learn 包的机器学习中**模型的训练**是调用同名函数fit()来实现。示例如下。

In[14]

```
#基于训练集训练出新的具体模型

myModel.fit(X_trainingSet, y_trainingSet)
#【提示】训练集的特征矩阵：X_trainingSet
#【提示】训练集的目标向量：y_trainingSet
```

对应输出结果为：

```
KNeighborsClassifier(algorithm='kd_tree', leaf_size=30, metric='minkowski’,
metric_params=None, n_jobs=None, n_neighbors=5, p=2, weights='uniform')
```

Step 6:用模型进行预测

与第2章的“2.7 Python 编程实践”中介绍的统计模型的应用(预测)方式类似，基于scikit-learn 包的机器学习中模型的应用(预测)是调用同名函数predict()来实现预测。示例如下。

```
In[15]|#用上一步中已训练出的具体模型，并基于测试集中的特征矩阵，预测对应的目标向量
      |y_predictSet = myModel.predict(X_testSet)
      |#【提示】测试集的特征矩阵为 X_testSet
```

上一行代码的含义为，采用所训练出的新模型 myModel 对测试集中的自变量X_testSet 进行预测，并获得所对应的因变量diagnosis 的预测值y_predictSet。示例如下。

查看预测结果

```
In[16]: #查看预测结果
        print(y_predictSet)
```

对应输出结果：

```
['M' 'M' 'B' 'M' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B'
 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'B' 'M' 'B' 'B' 'B'
 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B'
 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B'
 'B' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M'
 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'B' 'B' 'M' 'M' 'B']
```

上面的输出结果显示的是根据所训练出的新模型对测试集 X_testSet 中的每一个样本依次预测的因变量 diagnosis 的值，“M”代表的是“恶性肿瘤”，“B”代表的是“良性肿瘤”。

查看真实值

接着，我们查看在原始数据集 bc_data 中与测试集 X_testSet 对应的真实诊断结论，即 y_testSet 的值。示例如下。

```
In[17] #查看真实值
       print(y_testSet)
```

对应输出结果：

```
['B' 'M' 'B' 'M' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B'
 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'B' 'M' 'M' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B'
 'M' 'M' 'B' 'M' 'M' 'M' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B'
 'B' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M'
 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B']
```

从输出结果可以看出，个别病例的模型预测的结论与原数据集中的标签信息，即诊断结论不一致，如第一个病例的预测结论为“M”，但原始数据中其标注的诊断结论为“B”。为此，我们需要引入**模型评估方法**计算预测结果的**准确率**。

Step 7:模型评估

与本书第2章“2.7 Python 编程实践”中介绍的统计模型的优度检验类似，基于机器学习的数据科学项目也需要对所训练出模型进行评估。
二者的区别在于，前者依据的是统计指标(如 R 方等)，而后者采用的是交叉验证方法(如基于混淆矩阵计算准确率)。
Python 第三方包 sklearn.metrics 中提供了可用于计算准确率的函数 accaccuracy_ score()。因此，我们可以调用该函数轻松实现评价。示例如下。

```
In[18] #导入 accuracy_score()函数用于计算模型的准确率
       from sklearn.metrics import accuracy_score
       #查看模型的准确率
       print(accuracy_score(y_testSet, y_predictSet))
        #【提示】y_testSet 和 y_predictSet 分别为测试集和预测集
```

对应输出结果为：

```
0.9370629370629371
```

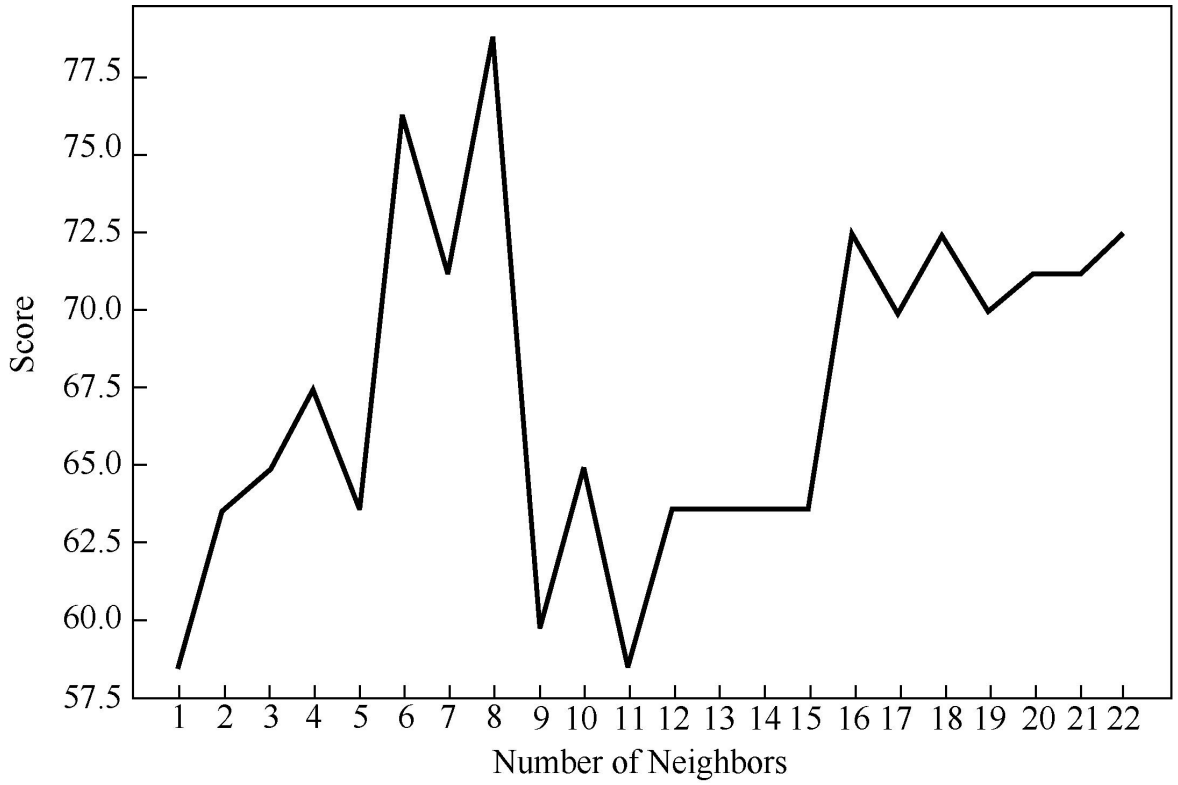
Step 8:模型的应用与优化

- 不管是基于统计学还是基于机器学习的数据科学项目，我们的数据科学项目很难做到“一次成功”。通常需要对已有训练模型进行**优化**，甚至需要改变机器学习算法的方式来训练新模型。
- 当已训练模型的准确率可以满足业务需求时，可以用这个模型进行预测(Predict)新数据或更多数据。
- 如果该模型的准确率可以满足业务需求，那么需要进一步**优化模型参数**，甚至替换成其他算法/模型。
- 对于KNN 算法而言，**K 值的选择**是该算法的难点所在。
- 以上代码中，我们直接用了 scikit-learn 包中提供的函数KNeighborsClassifier()的可选参数 K 的默认值。
- 为了找到更优的 K 值，我们采用可以数据可视化的方法画出模型的学习曲线(Learning Curve)，观察K 值的变化规律，选择更优的K 值。
- 示例如下。

（1）画出模型的学习曲线

```
In[19] | from sklearn.neighbors import
       | KNeighborsClassifier
       | NumberOfNeighbors = range(1,23)
       | KNNs = [KNeighborsClassifier(n_neighbors=i) for i in
       | NumberOfNeighbors]
       | scores = [KNNs[i].fit(X_trainingSet,
       | y_trainingSet).score(X_testSet, y_testSet) for i in
       | range(len(KNNs))]
       | import matplotlib.pyplot as plt
       | %matplotlib inline
       | plt.plot(NumberOfNeighbors,scores)
       | plt.xlabel('Number of Neighbors’)
       | plt.ylabel('Score’)
       | plt.title('Elbow Curve’)
       | plt.xticks(NumberOfNeighbors)
```

对应输出结果为：



（2）重新预测

从上图可以看出，K=4时模型的预测准确率出现了一个拐点。所以，我们可以将算法的K参数调整为7，并重新预测和计算准确率，查看准确率是否有提高。

In[18]:

```
#重新预测
from sklearn.neighbors import KNeighborsClassifier
cancerModel=KNeighborsClassifier(algorithm='kd_tree',n_neighbors=4)
myModel.fit(X_trainingSet, y_trainingSet)
y_predictSet = myModel.predict(X_testSet)
from sklearn.metrics import accuracy_score
print(accuracy_score(y_testSet, y_predictSet))
```

对应输出结果为：

```
0.944055944055944
```

可见，K=4 时，模型的预测准确率从 0.9370629370629371 提高至 0.944055944055944。