# Python & 机器学习之项目实践 | 赠书

2017-12-26 魏贞原 人工智能头条

文章节选自《机器学习——Python实践》

**文末评论赠送本书，欢迎留言！**

机器学习是一项经验技能，经验越多越好。在项目建立的过程中，实践是掌握机器学习的最佳手段。在实践过程中，通过实际操作加深对分类和回归问题的每一个步骤的理解，达到学习机器学习的目的。

## 预测模型项目模板

不能只通过阅读来掌握机器学习的技能，需要进行大量的练习。本文将介绍一个通用的机器学习的项目模板，创建这个模板总共有六个步骤。通过本文将学到：

- 端到端地预测（分类与回归）模型的项目结构。
- 如何将前面学到的内容引入到项目中。
- 如何通过这个项目模板来得到一个高准确度的模板。

机器学习是针对数据进行自动挖掘，找出数据的内在规律，并应用这个规律来预测新数据，如图19-1所示。

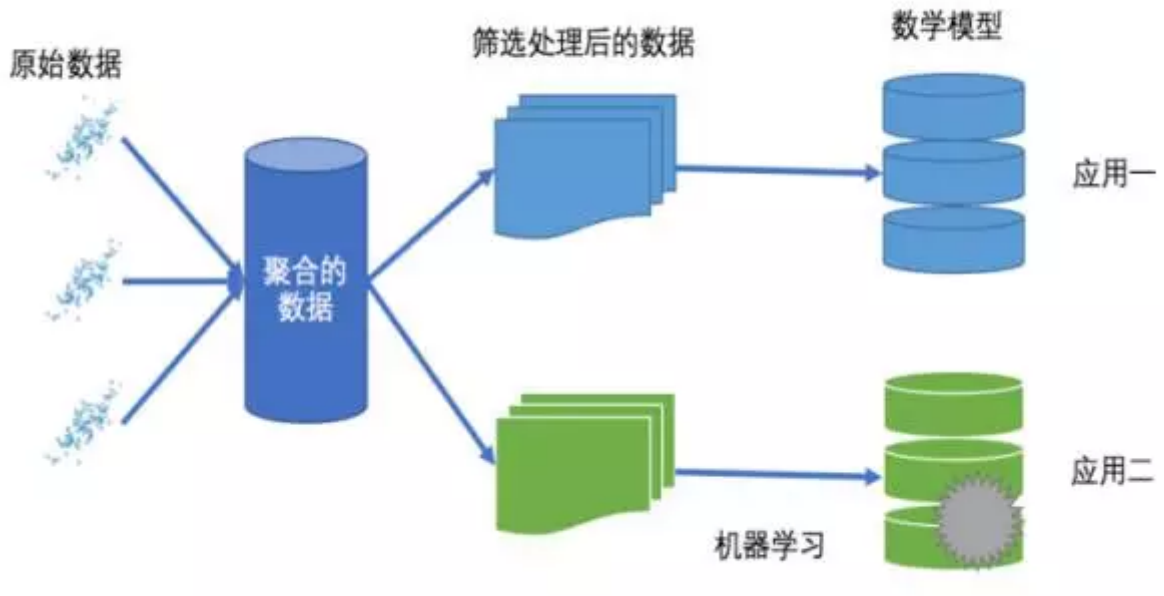



图19-1

## 在项目中实践机器学习

端到端地解决机器学习的问题是非常重要的。可以学习机器学习的知识，可以实践机器学习的某个方面，但是只有针对某一个问题，从问题定义开始到模型部署为止，通过实践机器学习的各个方面，才能真正掌握并应用机器学习来解决实际问题。

在部署一个项目时，全程参与到项目中可以更加深入地思考如何使用模型，以及勇于尝试用机器学习解决问题的各个方面，而不仅仅是参与到自己感兴趣或擅长的方面。一个很好的实践机器学习项目的方法是，使用从UCI机器学习仓库（http://archive.ics.uci.edu/ml/datasets.html）获取的数据集开启一个机器学习项目。如果从一个数据集开始实践机器学习，应该如何将学到的所有技巧和方法整合到一起来处理机器学习的问题呢？

分类或回归模型的机器学习项目可以分成以下六个步骤：

（1）定义问题。
（2）理解数据。
（3）数据准备。
（4）评估算法。
（5）优化模型。
（6）结果部署。

有时这些步骤可能被合并或进一步分解，但通常是按上述六个步骤来开展机器学习项目的。为了符合Python的习惯，在下面的Python项目模板中，按照这六个步骤分解整个项目，在接下来的部分会明确各个步骤或子步骤中所要实现的功能。

## 机器学习项目的Python模板

下面会给出一个机器学习项目的Python模板。代码如下：

```
# Python机器学习项目的模板

# 1. 定义问题
# a) 导入类库
# b) 导入数据集

# 2. 理解数据
# a) 描述性统计
# b) 数据可视化

# 3. 数据准备
# a) 数据清洗
# b) 特征选择
# c) 数据转换

# 4. 评估算法
# a) 分离数据集
# b) 定义模型评估标准
# c) 算法审查
# d) 算法比较

# 5. 优化模型
# a) 算法调参
# b) 集成算法

# 6. 结果部署
# a) 预测评估数据集
# b) 利用整个数据集生成模型
# c) 序列化模型
```

当有新的机器学习项目时，新建一个Python文件，并将这个模板粘贴进去，再按照前面章节介绍的方法将其填充到每一个步骤中。

## 各步骤的详细说明

接下来将详细介绍项目模板的各个步骤。

**步骤1：定义问题**

主要是导入在机器学习项目中所需要的类库和数据集等，以便完成机器学习的项目，包括导入Python的类库、类和方法，以及导入数据。同时这也是所有的配置参数的配置模块。当数据集过大时，可以在这里对数据集进行瘦身处理，理想状态是可以在1分钟内，甚至是30秒内完成模型的建立或可视化数据集。

**步骤2：理解数据**

这是加强对数据理解的步骤，包括通过描述性统计来分析数据和通过可视化来观察数据。在这一步需要花费时间多问几个问题，设定假设条件并调查分析一下，这对模型的建立会有很大的帮助。

**步骤3：数据准备**

数据准备主要是预处理数据，以便让数据可以更好地展示问题，以及熟悉输入与输出结果的关系。包括：

- 通过删除重复数据、标记错误数值，甚至标记错误的输入数据来清洗数据。
- 特征选择，包括移除多余的特征属性和增加新的特征属性。
- 数据转化，对数据尺度进行调整，或者调整数据的分布，以便更好地展示问题。

要不断地重复这个步骤和下一个步骤，直到找到足够准确的算法生成模型。

**步骤4：评估算法**

评估算法主要是为了寻找最佳的算法子集，包括：

- 分离出评估数据集，以便于验证模型。
- 定义模型评估标准，用来评估算法模型。
- 抽样审查线性算法和非线性算法。
- 比较算法的准确度。

在面对一个机器学习的问题的时候，需要花费大量的时间在评估算法和准备数据上，直到找到3~5种准确度足够的算法为止。

**步骤5：优化模型**

当得到一个准确度足够的算法列表后，要从中找出最合适的算法，通常有两种方法可以提高算法的准确度：

- 对每一种算法进行调参，得到最佳结果。
- 使用集合算法来提高算法模型的准确度。

**步骤6：结果部署**

一旦认为模型的准确度足够高，就可以将这个模型序列化，以便有新数据时使用该模型来预测数据。

- 通过验证数据集来验证被优化过的模型。
- 通过整个数据集来生成模型。
- 将模型序列化，以便于预测新数据。

做到这一步的时候，就可以将模型展示并发布给相关人员。当有新数据产生时，就可以采用这个模型来预测新数据。

## 使用模板的小技巧

快速执行一遍：首先要快速地在项目中将模板中的每一个步骤执行一遍，这样会加强对项目每一部分的理解并给如何改进带来灵感。

循环：整个流程不是线性的，而是循环进行的，要花费大量的时间来重复各个步骤，尤其是步骤3或步骤4（或步骤3～步骤5），直到找到一个准确度足够的模型，或者达到预定的周期。

尝试每一个步骤：跳过某个步骤很简单，尤其是不熟悉、不擅长的步骤。坚持在这个模板的每一个步骤中做些工作，即使这些工作不能提高算法的准确度，但也许在后面的操作就可以改进并提高算法的准确度。即使觉得这个步骤不适用，也不要跳过这个步骤，而是减少该步骤所做的贡献。

定向准确度：机器学习项目的目标是得到一个准确度足够高的模型。每一个步骤都要为实现这个目标做出贡献。要确保每次改变都会给结果带来正向的影响，或者对其他的步骤带来正向的影响。在整个项目的每个步骤中，准确度只能向变好的方向移动。

按需适用：可以按照项目的需要来修改步骤，尤其是对模板中的各个步骤非常熟悉之后。需要把握的原则是，每一次改进都以提高算法模型的准确度为前提。

## 总结

本章介绍了预测模型项目的模板，这个模板适用于分类或回归问题。接下来将介绍机器学习中的一个回归问题的项目，这个项目比前面介绍的鸢尾花的例子更加复杂，会利用到本章介绍的每个步骤。

# 回归项目实例

机器学习是一项经验技能，实践是掌握机器学习、提高利用机器学习解决问题的能力的有效方法之一。那么如何通过机器学习来解决问题呢？本章将通过一个实例来一步一步地介绍一个回归问题。本章主要介绍以下内容：

- 如何端到端地完成一个回归问题的模型。
- 如何通过数据转换提高模型的准确度。
- 如何通过调参提高模型的准确度。
- 如何通过集成算法提高模型的准确度。

## 定义问题

在这个项目中将分析研究波士顿房价（Boston House Price）数据集，这个数据集中的每一行数据都是对波士顿周边或城镇房价的描述。数据是1978年统计收集的。数据中包含以下14个特征和506条数据（UCI机器学习仓库中的定义）。

- CRIM：城镇人均犯罪率。
- ZN：住宅用地所占比例。
- INDUS：城镇中非住宅用地所占比例。
- CHAS：CHAS虚拟变量，用于回归分析。
- NOX：环保指数。
- RM：每栋住宅的房间数。
- AGE：1940年以前建成的自住单位的比例。
- DIS：距离5个波士顿的就业中心的加权距离。
- RAD：距离高速公路的便利指数。
- TAX：每一万美元的不动产税率。
- PRTATIO：城镇中的教师学生比例。
- B：城镇中的黑人比例。
- LSTAT：地区中有多少房东属于低收入人群。
- MEDV：自住房屋房价中位数。

通过对这些特征属性的描述，我们可以发现输入的特征属性的度量单位是不统一的，也许需要对数据进行度量单位的调整。

## 导入数据

首先导入在项目中需要的类库。代码如下：

```
# 导入类库
import numpy as np
from numpy import arange
from matplotlib import pyplot
from pandas import read_csv
from pandas import  set_option
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error
```

接下来导入数据集到Python中，这个数据集也可以从UCI机器学习仓库下载，在导入数据集时还设定了数据属性特征的名字。代码如下：

```
# 导入数据
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
        'RAD', 'TAX', 'PRTATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=names, delim_whitespace=True)
```

在这里对每一个特征属性设定了一个名称，以便于在后面的程序中使用它们。因为CSV文件是使用空格键做分隔符的，因此读入CSV文件时指定分隔符为空格键（delim_whitespace=True）。

### 理解数据

对导入的数据进行分析，便于构建合适的模型。

首先看一下数据维度，例如数据集中有多少条记录、有多少个数据特征。代码如下：

```
# 数据维度print(dataset.shape)
```

执行之后我们可以看到总共有506条记录和14个特征属性，这与UCI提供的信息一致。

```
(506, 14)
```

再查看各个特征属性的字段类型。代码如下：

```
# 特征属性的字段类型
print(dataset.dtypes)
```

可以看到所有的特征属性都是数字，而且大部分特征属性都是浮点数，也有一部分特征属性是整数类型的。执行结果如下：

```
CRIM       float64
ZN         float64
INDUS      float64
CHAS         int64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD          int64
TAX        float64
PRTATIO    float64
B          float64
LSTAT      float64
MEDV       float64
dtype: object
```

接下来对数据进行一次简单的查看，在这里我们查看一下最开始的30条记录。代码如下：

```
# 查看最开始的30条记录
set_option('display.line_width', 120)
print(dataset.head(30))
```

这里指定输出的宽度为120个字符，以确保将所有特征属性值显示在一行内。而且这些数据不是用相同的单位存储的，进行后面的操作时，也许需要将数据整理为相同的度量单位。执行结果如图20-1所示。

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PRTATIO | B | LSTAT | MEDV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00632 | 18.0 | 2.31 | 0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1 | 296.0 | 15.3 | 396.90 | 4.98 | 24.0 |
| 1 | 0.02731 | 0.0 | 7.07 | 0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2 | 242.0 | 17.8 | 396.90 | 9.14 | 21.6 |
| 2 | 0.02729 | 0.0 | 7.07 | 0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2 | 242.0 | 17.8 | 392.83 | 4.03 | 34.7 |
| 3 | 0.03237 | 0.0 | 2.18 | 0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3 | 222.0 | 18.7 | 394.63 | 2.94 | 33.4 |
| 4 | 0.06905 | 0.0 | 2.18 | 0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3 | 222.0 | 18.7 | 396.90 | 5.33 | 36.2 |
| 5 | 0.02985 | 0.0 | 2.18 | 0 | 0.458 | 6.430 | 58.7 | 6.0622 | 3 | 222.0 | 18.7 | 394.12 | 5.21 | 28.7 |
| 6 | 0.08829 | 12.5 | 7.87 | 0 | 0.524 | 6.012 | 66.6 | 5.5605 | 5 | 311.0 | 15.2 | 395.60 | 12.43 | 22.9 |
| 7 | 0.14455 | 12.5 | 7.87 | 0 | 0.524 | 6.172 | 96.1 | 5.9505 | 5 | 311.0 | 15.2 | 396.90 | 19.15 | 27.1 |
| 8 | 0.21124 | 12.5 | 7.87 | 0 | 0.524 | 5.631 | 100.0 | 6.0821 | 5 | 311.0 | 15.2 | 386.63 | 29.93 | 16.5 |
| 9 | 0.17004 | 12.5 | 7.87 | 0 | 0.524 | 6.004 | 85.9 | 6.5921 | 5 | 311.0 | 15.2 | 386.71 | 17.10 | 18.9 |
| 10 | 0.22489 | 12.5 | 7.87 | 0 | 0.524 | 6.377 | 94.3 | 6.3467 | 5 | 311.0 | 15.2 | 392.52 | 20.45 | 15.0 |
| 11 | 0.11747 | 12.5 | 7.87 | 0 | 0.524 | 6.009 | 82.9 | 6.2267 | 5 | 311.0 | 15.2 | 396.90 | 13.27 | 18.9 |
| 12 | 0.09378 | 12.5 | 7.87 | 0 | 0.524 | 5.889 | 39.0 | 5.4509 | 5 | 311.0 | 15.2 | 390.50 | 15.71 | 21.7 |
| 13 | 0.62976 | 0.0 | 8.14 | 0 | 0.538 | 5.949 | 61.8 | 4.7075 | 4 | 307.0 | 21.0 | 396.90 | 8.26 | 20.4 |
| 14 | 0.63796 | 0.0 | 8.14 | 0 | 0.538 | 6.096 | 84.5 | 4.4619 | 4 | 307.0 | 21.0 | 380.02 | 10.26 | 18.2 |
| 15 | 0.62739 | 0.0 | 8.14 | 0 | 0.538 | 5.834 | 56.5 | 4.4986 | 4 | 307.0 | 21.0 | 395.62 | 8.47 | 19.9 |
| 16 | 1.05393 | 0.0 | 8.14 | 0 | 0.538 | 5.935 | 29.3 | 4.4986 | 4 | 307.0 | 21.0 | 386.85 | 6.58 | 23.1 |
| 17 | 0.78420 | 0.0 | 8.14 | 0 | 0.538 | 5.990 | 81.7 | 4.2579 | 4 | 307.0 | 21.0 | 386.75 | 14.67 | 17.5 |
| 18 | 0.80271 | 0.0 | 8.14 | 0 | 0.538 | 5.456 | 36.6 | 3.7965 | 4 | 307.0 | 21.0 | 288.99 | 11.69 | 20.2 |
| 19 | 0.72580 | 0.0 | 8.14 | 0 | 0.538 | 5.727 | 69.5 | 3.7965 | 4 | 307.0 | 21.0 | 390.95 | 11.28 | 18.2 |
| 20 | 1.25179 | 0.0 | 8.14 | 0 | 0.538 | 5.570 | 98.1 | 3.7979 | 4 | 307.0 | 21.0 | 376.57 | 21.02 | 13.6 |
| 21 | 0.85204 | 0.0 | 8.14 | 0 | 0.538 | 5.965 | 89.2 | 4.0123 | 4 | 307.0 | 21.0 | 392.53 | 13.83 | 19.6 |
| 22 | 1.23247 | 0.0 | 8.14 | 0 | 0.538 | 6.142 | 91.7 | 3.9769 | 4 | 307.0 | 21.0 | 396.90 | 18.72 | 15.2 |
| 23 | 0.98843 | 0.0 | 8.14 | 0 | 0.538 | 5.813 | 100.0 | 4.0952 | 4 | 307.0 | 21.0 | 394.54 | 19.88 | 14.5 |
| 24 | 0.75026 | 0.0 | 8.14 | 0 | 0.538 | 5.924 | 94.1 | 4.3996 | 4 | 307.0 | 21.0 | 394.33 | 16.30 | 15.6 |
| 25 | 0.84054 | 0.0 | 8.14 | 0 | 0.538 | 5.599 | 85.7 | 4.4546 | 4 | 307.0 | 21.0 | 303.42 | 16.51 | 13.9 |
| 26 | 0.67191 | 0.0 | 8.14 | 0 | 0.538 | 5.813 | 90.3 | 4.6820 | 4 | 307.0 | 21.0 | 376.88 | 14.81 | 16.6 |
| 27 | 0.95577 | 0.0 | 8.14 | 0 | 0.538 | 6.047 | 88.8 | 4.4534 | 4 | 307.0 | 21.0 | 306.38 | 17.28 | 14.8 |
| 28 | 0.77299 | 0.0 | 8.14 | 0 | 0.538 | 6.495 | 94.4 | 4.4547 | 4 | 307.0 | 21.0 | 387.94 | 12.80 | 18.4 |
| 29 | 1.00245 | 0.0 | 8.14 | 0 | 0.538 | 6.674 | 87.3 | 4.2390 | 4 | 307.0 | 21.0 | 380.23 | 11.98 | 21.0 |

图20-1

接下来看一下数据的描述性统计信息。代码如下：

```
# 描述性统计信息
set_option('precision', 1)
print(dataset.describe())
```

在描述性统计信息中包含数据的最大值、最小值、中位值、四分位值等，分析这些数据能够加深对数据分布、数据结构等的理解。结果如图20-2所示。

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PRTATIO | B | LSTAT | MEDV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 5.1e+02 | 506.0 | 506.0 | 5.1e+02 | 506.0 | 506.0 | 506.0 | 506.0 | 506.0 | 506.0 | 506.0 | 506.0 | 506.0 | 506.0 |
| mean | 3.6e+00 | 11.4 | 11.1 | 6.9e−02 | 0.6 | 6.3 | 68.6 | 3.8 | 9.5 | 408.2 | 18.5 | 356.7 | 12.7 | 22.5 |
| std | 8.6e+00 | 23.3 | 6.9 | 2.5e−01 | 0.1 | 0.7 | 28.1 | 2.1 | 8.7 | 168.5 | 2.2 | 91.3 | 7.1 | 9.2 |
| min | 6.3e−03 | 0.0 | 0.5 | 0.0e+00 | 0.4 | 3.6 | 2.9 | 1.1 | 1.0 | 187.0 | 12.6 | 0.3 | 1.7 | 5.0 |
| 25% | 8.2e−02 | 0.0 | 5.2 | 0.0e+00 | 0.4 | 5.9 | 45.0 | 2.1 | 4.0 | 279.0 | 17.4 | 375.4 | 6.9 | 17.0 |
| 50% | 2.6e−01 | 0.0 | 9.7 | 0.0e+00 | 0.5 | 6.2 | 77.5 | 3.2 | 5.0 | 330.0 | 19.1 | 391.4 | 11.4 | 21.2 |
| 75% | 3.7e+00 | 12.5 | 18.1 | 0.0e+00 | 0.6 | 6.6 | 94.1 | 5.2 | 24.0 | 666.0 | 20.2 | 396.2 | 17.0 | 25.0 |
| max | 8.9e+01 | 100.0 | 27.7 | 1.0e+00 | 0.9 | 8.8 | 100.0 | 12.1 | 24.0 | 711.0 | 22.0 | 396.9 | 38.0 | 50.0 |

图20-2

接下来看一下数据特征之间的两两关联关系，这里查看数据的皮尔逊相关系数。代码如下：

```
# 关联关系
set_option('precision', 2)
print(dataset.corr(method='pearson'))
```

执行结果如图20-3所示。

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PRTATIO | B | LSTAT | MEDV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CRIM | 1.00 | -0.20 | 0.41 | -5.59e-02 | 0.42 | -0.22 | 0.35 | -0.38 | 6.26e-01 | 0.58 | 0.29 | -0.39 | 0.46 | -0.39 |
| ZN | -0.20 | 1.00 | -0.53 | -4.27e-02 | -0.52 | 0.31 | -0.57 | 0.66 | -3.12e-01 | -0.31 | -0.39 | 0.18 | -0.41 | 0.36 |
| INDUS | 0.41 | -0.53 | 1.00 | 6.29e-02 | 0.76 | -0.39 | 0.64 | -0.71 | 5.95e-01 | 0.72 | 0.38 | -0.36 | 0.60 | -0.48 |
| CHAS | -0.06 | -0.04 | 0.06 | 1.00e+00 | 0.09 | 0.09 | 0.09 | -0.10 | -7.37e-03 | -0.04 | -0.12 | 0.05 | -0.05 | 0.18 |
| NOX | 0.42 | -0.52 | 0.76 | 9.12e-02 | 1.00 | -0.30 | 0.73 | -0.77 | 6.11e-01 | 0.67 | 0.19 | -0.38 | 0.59 | -0.43 |
| RM | -0.22 | 0.31 | -0.39 | 9.13e-02 | -0.30 | 1.00 | -0.24 | 0.21 | -2.10e-01 | -0.29 | -0.36 | 0.13 | -0.61 | 0.70 |
| AGE | 0.35 | -0.57 | 0.64 | 8.65e-02 | 0.73 | -0.24 | 1.00 | -0.75 | 4.56e-01 | 0.51 | 0.26 | -0.27 | 0.60 | -0.38 |
| DIS | -0.38 | 0.66 | -0.71 | -9.92e-02 | -0.77 | 0.21 | -0.75 | 1.00 | -4.95e-01 | -0.53 | -0.23 | 0.29 | -0.50 | 0.25 |
| RAD | 0.63 | -0.31 | 0.60 | -7.37e-03 | 0.61 | -0.21 | 0.46 | -0.49 | 1.00e+00 | 0.91 | 0.46 | -0.44 | 0.49 | -0.38 |
| TAX | 0.58 | -0.31 | 0.72 | -3.56e-02 | 0.67 | -0.29 | 0.51 | -0.53 | 9.10e-01 | 1.00 | 0.46 | -0.44 | 0.54 | -0.47 |
| PRTATIO | 0.29 | -0.39 | 0.38 | -1.22e-01 | 0.19 | -0.36 | 0.26 | -0.23 | 4.65e-01 | 0.46 | 1.00 | -0.18 | 0.37 | -0.51 |
| B | -0.39 | 0.18 | -0.36 | 4.88e-02 | -0.38 | 0.13 | -0.27 | 0.29 | -4.44e-01 | -0.44 | -0.18 | 1.00 | -0.37 | 0.33 |
| LSTAT | 0.46 | -0.41 | 0.60 | -5.39e-02 | 0.59 | -0.61 | 0.60 | -0.50 | 4.89e-01 | 0.54 | 0.37 | -0.37 | 1.00 | -0.74 |
| MEDV | -0.39 | 0.36 | -0.48 | 1.75e-01 | -0.43 | 0.70 | -0.38 | 0.25 | -3.82e-01 | -0.47 | -0.51 | 0.33 | -0.74 | 1.00 |

图20-3

通过上面的结果可以看到，有些特征属性之间具有强关联关系（ >0.7或<-0.7 ），如：

- NOX与INDUS之间的皮尔逊相关系数是0.76。
- DIS与INDUS之间的皮尔逊相关系数是-0.71。
- TAX与INDUS之间的皮尔逊相关系数是0.72。
- AGE与NOX之间的皮尔逊相关系数是0.73。
- DIS与NOX之间的皮尔逊相关系数是-0.77。

## 数据可视化

### 单一特征图表

首先查看每一个数据特征单独的分布图，多查看几种不同的图表有助于发现更好的方法。我们可以通过查看各个数据特征的直方图，来感受一下数据的分布情况。代码如下：

```
# 直方图
dataset.hist(sharex=False, sharey=False, xlabelsize=1, ylabelsize=1)
pyplot.show()
```

执行结果如图20-4所示，从图中可以看到有些数据呈指数分布，如CRIM、ZN、AGE和B；有些数据特征呈双峰分布，如RAD和TAX。

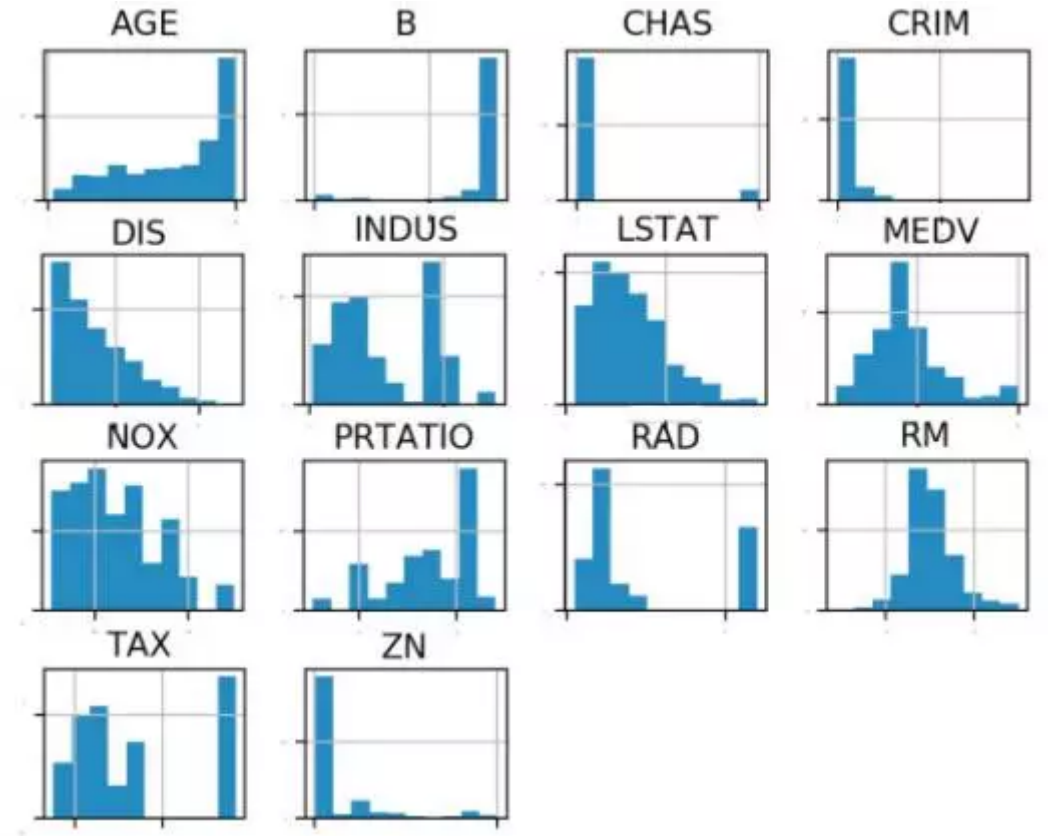



图20-4

通过密度图可以展示这些数据的特征属性，密度图比直方图更加平滑地展示了这些数据特征。代码如下：

```
# 密度图
dataset.plot(kind='density', subplots=True, layout=(4,4), sharex=False, fontsize=1)
pyplot.show()
```

在密度图中，指定layout=(4, 4)，这说明要画一个四行四列的图形。执行结果如图20-5所示。

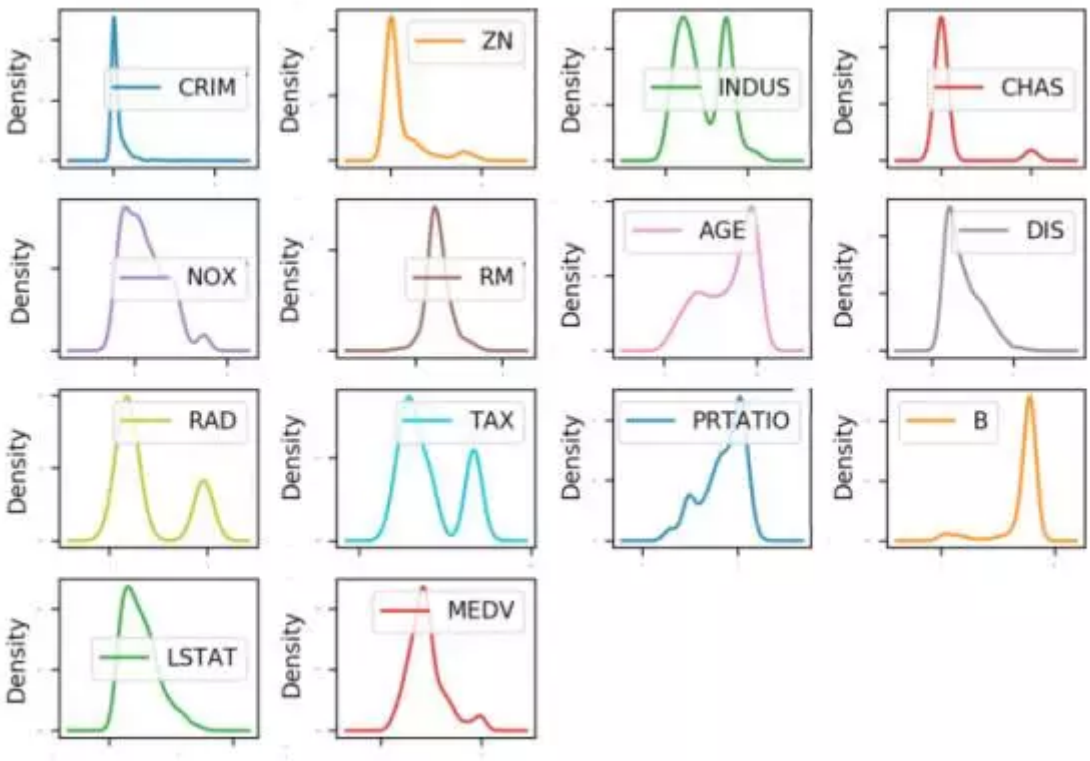



图20-5

通过箱线图可以查看每一个数据特征的状况，也可以很方便地看出数据分布的偏态程度。代码如下：

```
#箱线图
dataset.plot(kind='box', subplots=True, layout=(4,4), sharex=False, sharey=False, fontsize=8)
pyplot.show()
```

执行结果如图20-6所示。

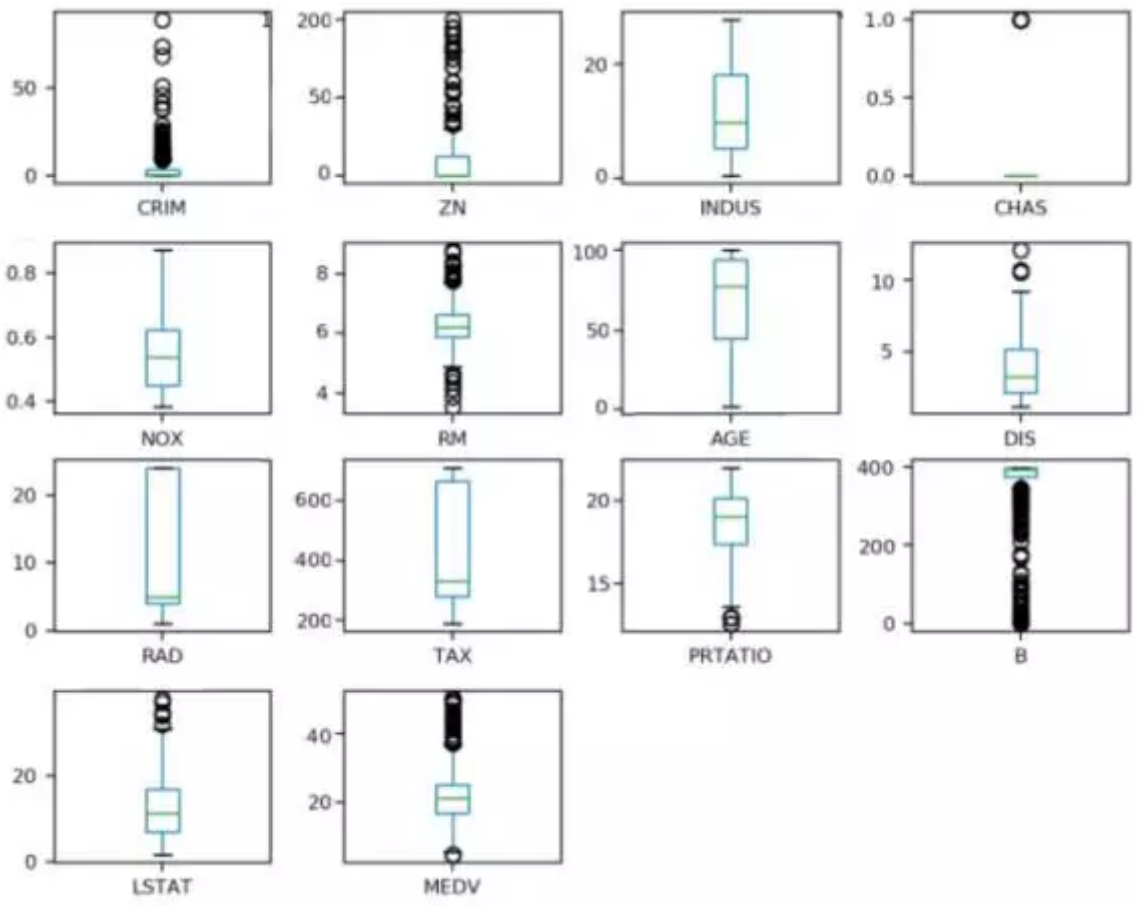



图20-6

**多重数据图表**

接下来利用多重数据图表来查看不同数据特征之间的相互影响关系。首先看一下散点矩阵图。代码如下：

```
# 散点矩阵图
scatter_matrix(dataset)
pyplot.show()
```

通过散点矩阵图可以看到，虽然有些数据特征之间的关联关系很强，但是这些数据分布结构也很好。即使不是线性分布结构，也是可以很方便进行预测的分布结构，执行结果如图20-7所示。

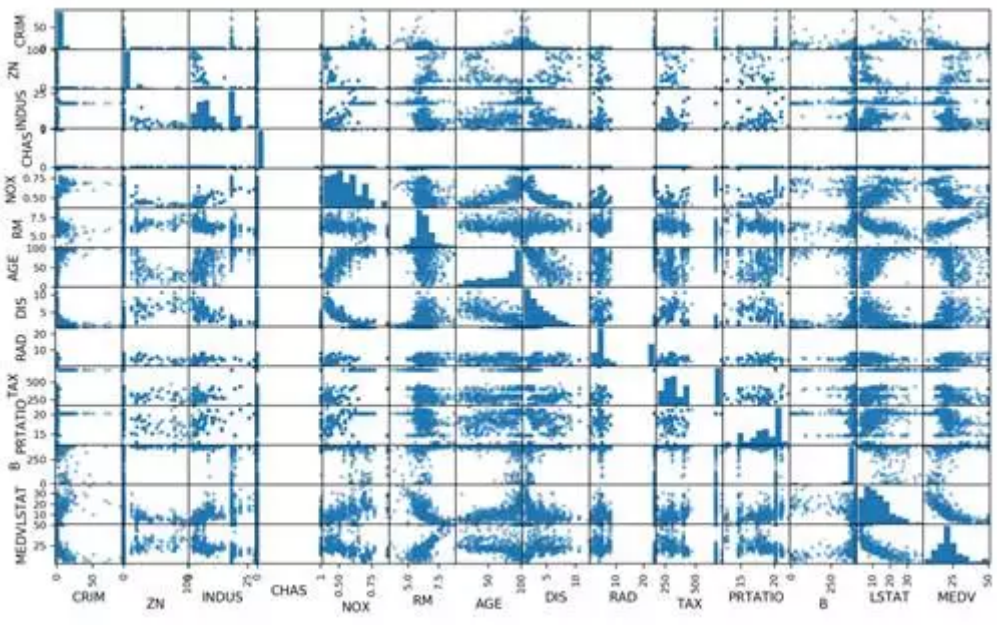



图20-7

再看一下数据相互影响的相关矩阵图。代码如下：

```
# 相关矩阵图
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(dataset.corr(), vmin=-1, vmax=1, interpolation='none')
fig.colorbar(cax)
ticks = np.arange(0, 14, 1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
pyplot.show()
```

执行结果如图20-8所示，根据图例可以看到，数据特征属性之间的两两相关性，有些属性之间是强相关的，建议在后续的处理中移除这些特征属性，以提高算法的准确度。

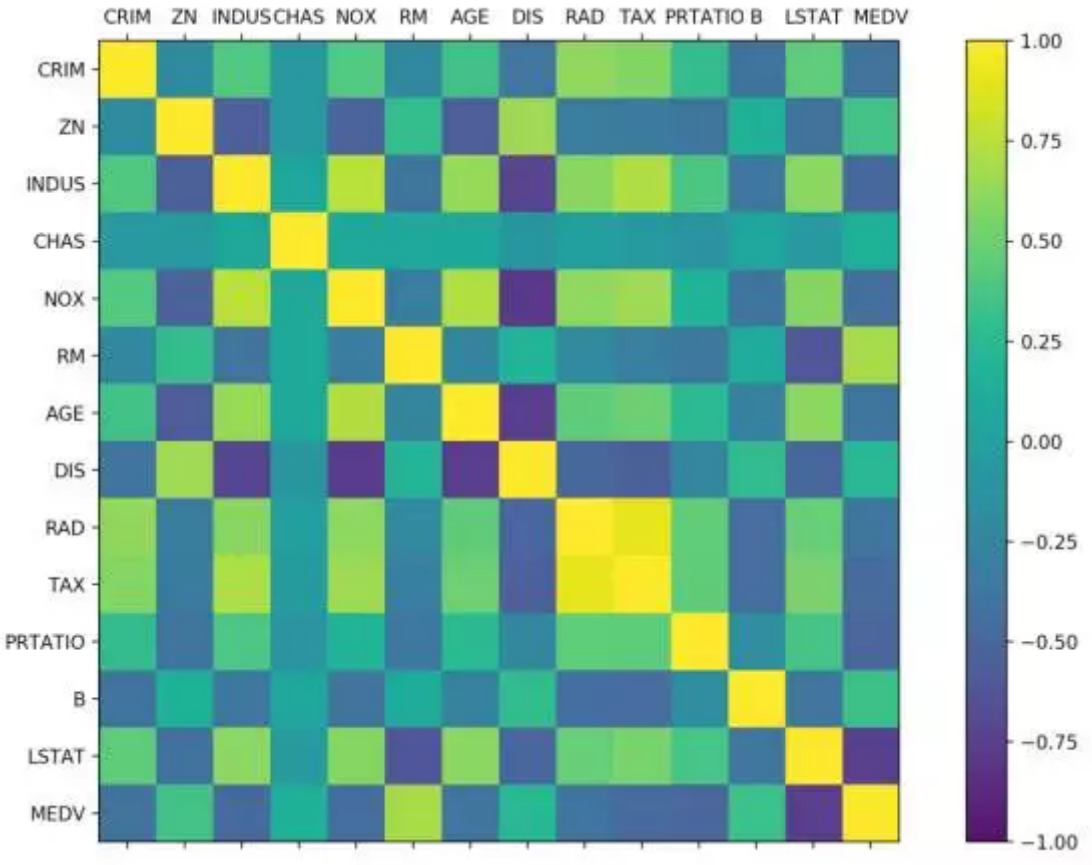



图20-8

**思路总结**

通过数据的相关性和数据的分布等发现，数据集中的数据结构比较复杂，需要考虑对数据进行转换，以提高模型的准确度。可以尝试从以下几个方面对数据进行处理：

- 通过特征选择来减少大部分相关性高的特征。
- 通过标准化数据来降低不同数据度量单位带来的影响。
- 通过正态化数据来降低不同的数据分布结构，以提高算法的准确度。

可以进一步查看数据的可能性分级（离散化），它可以帮助提高决策树算法的准确度。

## 分离评估数据集

分离出一个评估数据集是一个很好的主意，这样可以确保分离出的数据集与训练模型的数据集完全隔离，有助于最终判断和报告模型的准确度。在进行到项目的最后一步处理时，会使用这个评估数据集来确认模型的准确度。这里分离出20%的数据作为评估数据集，80%的数据作为训练数据集。代码如下：

```
# 分离数据集
array = dataset.values
X = array[:, 0:13]
Y = array[:, 13]
```

```
validation_size = 0.2
seed = 7
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y,test_size=validation_size,
random_state=seed)
```

## 评估算法

### 评估算法——原始数据

分析完数据不能立刻选择出哪个算法对需要解决的问题最有效。我们直观上认为，由于部分数据的线性分布，线性回归算法和弹性网络回归算法对解决问题可能比较有效。另外，由于数据的离散化，通过决策树算法或支持向量机算法也许可以生成高准确度的模型。到这里，依然不清楚哪个算法会生成准确度最高的模型，因此需要设计一个评估框架来选择合适的算法。我们采用10折交叉验证来分离数据，通过均方误差来比较算法的准确度。均方误差越趋近于0，算法准确度越高。代码如下：

```
# 评估算法 —— 评估标准
num_folds = 10
seed = 7
scoring = 'neg_mean_squared_error'
```

对原始数据不做任何处理，对算法进行一个评估，形成一个算法的评估基准。这个基准值是对后续算法改善优劣比较的基准值。我们选择三个线性算法和三个非线性算法来进行比较。

线性算法：线性回归（LR）、套索回归（LASSO）和弹性网络回归（EN）。
非线性算法：分类与回归树（CART）、支持向量机（SVM）和K近邻算法（KNN）。

算法模型初始化的代码如下：

```
# 评估算法 - baseline
models = {}
models['LR'] = LinearRegression()
models['LASSO'] = Lasso()
models['EN'] = ElasticNet()
models['KNN']  = KNeighborsRegressor()
models['CART'] = DecisionTreeRegressor()
models['SVM'] = SVR()
```

对所有的算法使用默认参数，并比较算法的准确度，此处比较的是均方误差的均值和标准方差。代码如下：

```
# 评估算法
results = []
```

```
for key in models:
    kfold = KFold(n_splits=num_folds, random_state=seed)
    cv_result = cross_val_score(models[key], X_train, Y_train, cv=kfold, scoring=scoring)
    results.append(cv_result)
    print('%s: %f (%f)' % (key, cv_result.mean(), cv_result.std()))
```

从执行结果来看，线性回归（LR）具有最优的MSE，接下来是分类与回归树 （CART）算法。执行结果如下：

```
LR: -21.379856 (9.414264)
LASSO: -26.423561 (11.651110)
EN: -27.502259 (12.305022)
KNN: -41.896488 (13.901688)
CART: -26.608476 (12.250800)
SVM: -85.518342 (31.994798)
```

再查看所有的10折交叉分离验证的结果。代码如下：

```
#评估算法——箱线图
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(models.keys())
pyplot.show()
```

执行结果如图20-9所示，从图中可以看到，线性算法的分布比较类似，并且K近邻算法的结果分布非常紧凑。

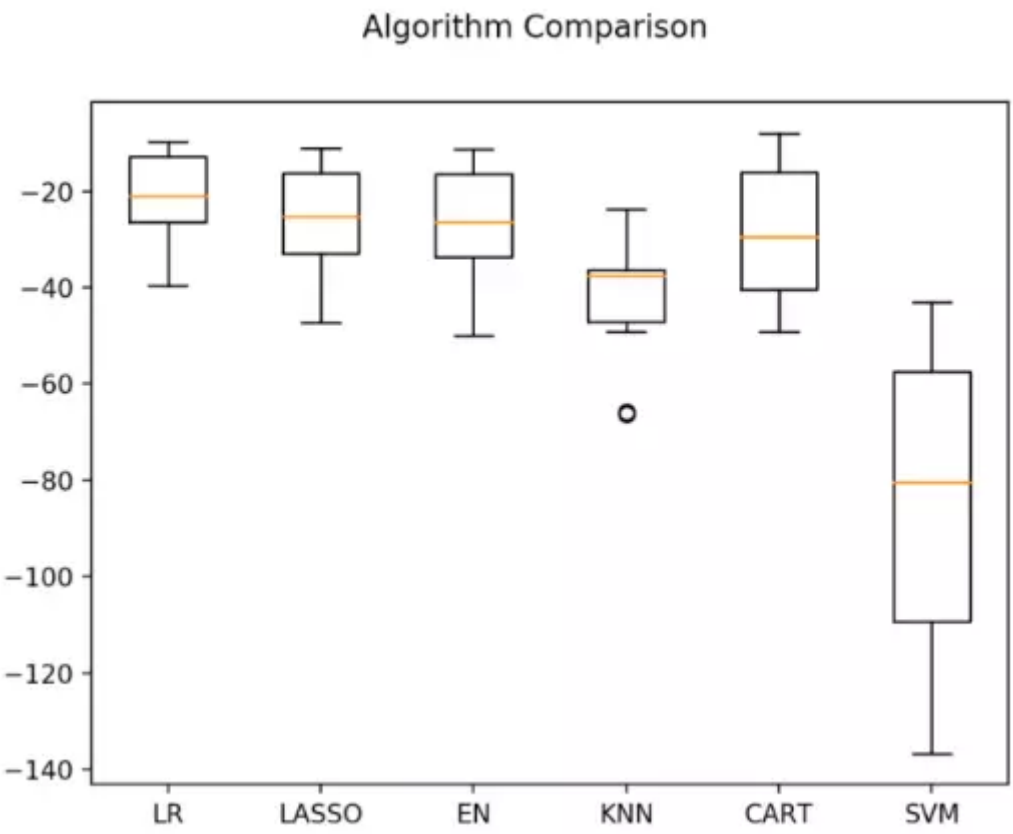



图20-9

不同的数据度量单位，也许是K近邻算法和支持向量机算法表现不佳的主要原因。下面将对数据进行正态化处理，再次比较算法的结果。

**评估算法——正态化数据**

在这里猜测也许因为原始数据中不同特征属性的度量单位不一样，导致有的算法的结果不是很好。接下来通过对数据进行正态化，再次评估这些算法。在这里对训练数据集进行数据转换处理，将所有的数据特征值转化成“0”为中位值、标准差为“1”的数据。对数据正态化时，为了防止数据泄露，采用Pipeline来正态化数据和对模型进行评估。为了与前面的结果进行比较，此处采用相同的评估框架来评估算法模型。代码如下：

```
# 评估算法——正态化数据
pipelines = {}
pipelines['ScalerLR'] = Pipeline([('Scaler', StandardScaler()), ('LR', LinearRegression())])
pipelines['ScalerLASSO'] = Pipeline([('Scaler', StandardScaler()), ('LASSO', Lasso())])
pipelines['ScalerEN'] = Pipeline([('Scaler',
StandardScaler()), ('EN', ElasticNet())])
pipelines['ScalerKNN'] = Pipeline([('Scaler',
StandardScaler()), ('KNN', KNeighborsRegressor())])
pipelines['ScalerCART'] = Pipeline([('Scaler',
StandardScaler()), ('CART', DecisionTreeRegressor())])
pipelines['ScalerSVM'] = Pipeline([('Scaler',
StandardScaler()), ('SVM', SVR())])
```

```
results = []
for key in pipelines:
    kfold = KFold(n_splits=num_folds, random_state=seed)
    cv_result = cross_val_score(pipelines[key], X_train, Y_train, cv=kfold, scoring=scoring)
    results.append(cv_result)
    print('%s: %f (%f)' % (key, cv_result.mean(), cv_result.std()))
```

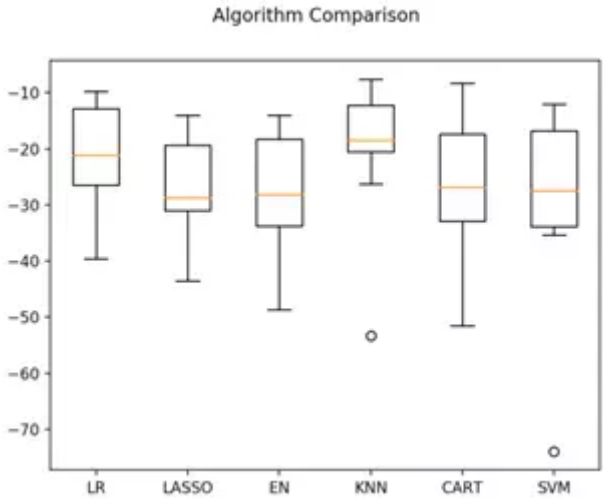
执行后发现K近邻算法具有最优的MSE。执行结果如下：

```
ScalerLR: -21.379856 (9.414264)
ScalerLASSO: -26.607314 (8.978761)
ScalerEN: -27.932372 (10.587490)
ScalerKNN: -20.107620 (12.376949)
ScalerCART: -26.978716 (12.164366)
ScalerSVM: -29.633086 (17.009186)
```

接下来看一下所有的10折交叉分离验证的结果。代码如下：

```
#评估算法——箱线图
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(models.keys())
pyplot.show()
```

执行结果，生成的箱线图如图20-10所示，可以看到K近邻算法具有最优的MSE和最紧凑的数据分布。




## 调参改善算法

目前来看，K近邻算法对做过数据转换的数据集有很好的结果，但是是否可以进一步对结果做一些优化呢？K近邻算法的默认参数近邻个数（n_neighbors）是5，下面通过网格搜索算法来优化参数。代码如下：

```
# 调参改善算法——KNN
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = {'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]}
model = KNeighborsRegressor()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model,
param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(X=rescaledX, y=Y_train)

print('最优：%s 使用%s' % (grid_result.best_score_, grid_result.best_params_))
cv_results =
zip(grid_result.cv_results_['mean_test_score'],

grid_result.cv_results_['std_test_score'],
                grid_result.cv_results_['params'])
for mean, std, param in cv_results:
    print('%f (%f) with %r' % (mean, std, param))
```

最优结果——K近邻算法的默认参数近邻个数（n_neighbors）是3。执行结果如下：

```
最优：-18.1721369637 使用{'n_neighbors': 3}
-20.208663 (15.029652) with {'n_neighbors': 1}
-18.172137 (12.950570) with {'n_neighbors': 3}
-20.131163 (12.203697) with {'n_neighbors': 5}
-20.575845 (12.345886) with {'n_neighbors': 7}
-20.368264 (11.621738) with {'n_neighbors': 9}
-21.009204 (11.610012) with {'n_neighbors': 11}
-21.151809 (11.943318) with {'n_neighbors': 13}
-21.557400 (11.536339) with {'n_neighbors': 15}
-22.789938 (11.566861) with {'n_neighbors': 17}
-23.871873 (11.340389) with {'n_neighbors': 19}
-24.361362 (11.914786) with {'n_neighbors': 21}
```

### 集成算法

除调参之外，提高模型准确度的方法是使用集成算法。下面会对表现比较好的线性回归、K近邻、分类与回归树算法进行集成，来看看算法能否提高。

装袋算法：随机森林（RF）和极端随机树（ET）。
提升算法：AdaBoost（AB）和随机梯度上升（GBM）。

依然采用和前面同样的评估框架和正态化之后的数据来分析相关的算法。代码如下：

```
# 集成算法
ensembles = {}
ensembles['ScaledAB'] = Pipeline([('Scaler',
StandardScaler()), ('AB', AdaBoostRegressor())])
ensembles['ScaledAB-KNN'] = Pipeline([('Scaler',
StandardScaler()), ('ABKNN', AdaBoostRegressor
(base_estimator= KNeighborsRegressor(n_neighbors=3)))])
ensembles['ScaledAB-LR'] = Pipeline([('Scaler',
StandardScaler()), ('ABLR',
AdaBoostRegressor(LinearRegression()))])
ensembles['ScaledRFR'] = Pipeline([('Scaler',
StandardScaler()), ('RFR', RandomForestRegressor())])
ensembles['ScaledETR'] = Pipeline([('Scaler',
StandardScaler()), ('ETR', ExtraTreesRegressor())])
ensembles['ScaledGBR'] = Pipeline([('Scaler',
StandardScaler()), ('RBR', GradientBoostingRegressor())])

results = []
for key in ensembles:
    kfold = KFold(n_splits=num_folds, random_state=seed)
    cv_result = cross_val_score(ensembles[key], X_train, Y_train, cv=kfold, scoring=scoring)
    results.append(cv_result)
    print('%s: %f (%f)' % (key, cv_result.mean(), cv_result.std()))
```

与前面的线性算法和非线性算法相比，这次的准确度都有了较大的提高。执行结果如下：

```
ScaledAB: -15.244803 (6.272186)
ScaledAB-KNN: -15.794844 (10.565933)
ScaledAB-LR: -24.108881 (10.165026)
ScaledRFR: -13.279674 (6.724465)
ScaledETR: -10.464980 (5.476443)
ScaledGBR: -10.256544 (4.605660)
```

接下来通过箱线图看一下集成算法在10折交叉验证中均方误差的分布状况。代码如下：

```
# 集成算法——箱线图
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
```

```
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(ensembles.keys())
pyplot.show()
```

执行结果如图20-11所示，随机梯度上升算法和极端随机树算法具有较高的中位值和分布状况。

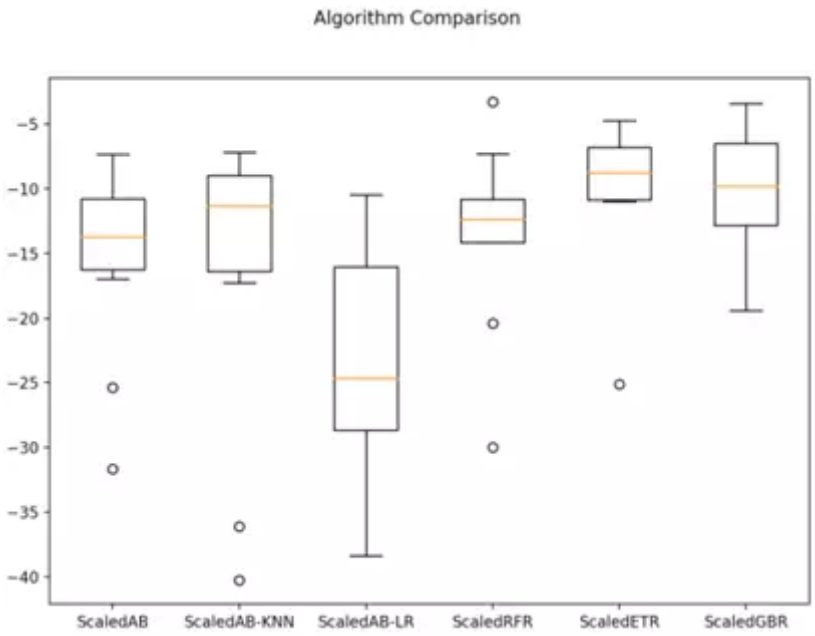



图20-11

## 集成算法调参

集成算法都有一个参数n_estimators，这是一个很好的可以用来调整的参数。对于集成参数来说，n_estimators会带来更准确的结果，当然这也有一定的限度。下面对随机梯度上升（GBM）和极端随机树（ET）算法进行调参，再次比较这两个算法模型的准确度，来确定最终的算法模型。代码如下：

```
# 集成算法GBM——调参
caler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = {'n_estimators': [10, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900]}
model = GradientBoostingRegressor()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model,
param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(X=rescaledX, y=Y_train)
print('最优：%s 使用%s' % (grid_result.best_score_,
grid_result.best_params_))

# 集成算法ET——调参
```

```
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
param_grid = {'n_estimators': [5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]}
model = ExtraTreesRegressor()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
grid_result = grid.fit(X=rescaledX, y=Y_train)
print('最优：%s 使用%s' % (grid_result.best_score_, grid_result.best_params_))
```

对于随机梯度上升（GBM）算法来说，最优的n_estimators是500；对于极端随机树（ET）算法来说，最优的n_estimators是80。执行结果，极端随机树（ET）算法略优于随机梯度上升（GBM）算法，因此采用极端随机树（ET）算法来训练最终的模型。执行结果如下：

```
最优：-9.3078229754 使用{'n_estimators': 500}
最优：-8.99113433246 使用{'n_estimators': 80}
```

也许需要执行多次这个过程才能找到最优参数。这里有一个技巧，当最优参数是param_grid的边界值时，有必要调整param_grid进行下一次调参。

### 确定最终模型

我们已经确定了使用极端随机树（ET）算法来生成模型，下面就对该算法进行训练和生成模型，并计算模型的准确度。代码如下：

```
#训练模型
caler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
gbr = ExtraTreesRegressor(n_estimators=80)
gbr.fit(X=rescaledX, y=Y_train)
```

再通过评估数据集来评估算法的准确度。

```
# 评估算法模型
rescaledX_validation = scaler.transform(X_validation)
predictions = gbr.predict(rescaledX_validation)
print(mean_squared_error(Y_validation, predictions))
```

执行结果如下：

```
14.077038511
```

### 总结

本项目实例从问题定义开始，直到最后的模型生成为止，完成了一个完整的机器学习项目。通过这个项目，理解了上一章中介绍的机器学习项目的模板，以及整个机器学习模型建立的流程。接下来会介绍一个机器学习的二分类问题，以进一步加深对这个模板的理解。

> **编者按：**《机器学习——Python实践》不同于很多讲解机器学习的书籍，本书以实践为导向，使用scikit-learn 作为编程框架，强调简单、快速地建立模型，解决实际项目问题。读者通过对《机器学习——Python实践》的学习，可以迅速上手实践机器学习，并利用机器学习解决实际问题。《机器学习——Python实践》非常适合于项目经理、有意从事机器学习开发的程序员，以及高校相关专业在的读学生阅读。

---

《机器学习——Python实践》订购链接（点击阅读原文订购）：https://item.jd.com/12252293.html


**赠书啦！！！**

**留言告诉头条菌你想获得这本书的理由，获点赞前5名就可获得本书。**

**开奖截止时间12月28日（本周四）中午12点！**

阅读原文