2、端到端的机器学习项目

2019年1月5日　　20:38

2、端到端的机器学习项目

- 观察大局
    - 框架问题

      我们首先应该明白业务目标是什么，因为建立模型本身肯能不是最终的目标。公司期望知道如何使用这个模型，如何从中获益。这些信息决定了你怎样去设定问题，选择什么算法，使用什么测量方式来评估模型的性能，以及花多少时间来进行调整。
    - 流水线

      一个序列的数据处理组件被称为一个数据流水线。组件通常是异步运行。每个组件拉取大量的数据，然后进行处理，再将结果传输给另一个数据仓库；一段时间后，流水线中的先一个组件会拉取前面的数据，并给出自己的输出，以此类推。每个组件都很独立：组件和组件之间的连接只有数据仓库。
    - 选择性能指标

      回归问题的典型性能衡量指标是均方根误差(RMSE)，它测量的是预测过程中，预测错误的标准偏差。
    - 检查假设

      最后，列举和验证到目前为止做出的假设（比如说你假定这个模型是回归模型、是有监督的等等）。在这种情况下我们会和流水线上下游的工作人员进行交流，确保不会出错。
- 获取数据
    - 创建工作区

      需要为机器学习的代码和数据集创建一个工作区目录。如果需要在一个隔离的环境里操作，可以通过运行如下pip命令来安装virtualenv：

      pip3 install --user --upgrade virtualenv #这里的--user是指非root用户的安装方式，这样就不需要sudo权限

      安装好virtualen之后，进入工作目录，然后执行如下操作

      cd $ML_PATH

      virtualenv env #这里好像有点问题，应该用virtualenv -p /usr/bin/python3 env，否则默认是python2的虚拟环境

      现在开始，每当想要激活这个环境时，只需要打开终端并输入

      cd $ML_PATH

      source env/bin/active

      退出用deactivate命令

      当这个环境处于激活状态时，你使用pip安装的任何软件包都将被安装在这个隔离的环境中，python只拥有这些包的访问权限（如果还需要访问系统站点的软件包，则需要使用virtualenv的--system-site-packages命令选项）

      接下来就为这个虚拟环境安装必要的python库
    - 下载数据
    - 快速查看数据结构

      1）使用dataframe的head()方法来查看数据的前五行

- 快速查看数据结构

1) 使用dataframe的head()方法来查看数据的前五行

```
In [5]: housing = load_housing_data()
        housing.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populatio
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

2) 通过info()方法可以快速获取数据集的简单描述，特别是总行数、每个属性的类型以及空值的数量

```
In [6]: housing.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude           20640 non-null float64
latitude            20640 non-null float64
housing_median_age  20640 non-null float64
total_rooms         20640 non-null float64
total_bedrooms      20433 non-null float64
population          20640 non-null float64
households          20640 non-null float64
median_income       20640 non-null float64
median_house_value  20640 non-null float64
ocean_proximity     20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

3) 对于非数值型属性（某一列），可以使用value_counts()方法查看有多少种分类存在，每个种类有多少个example。

```
>>> housing["ocean_proximity"].value_counts()
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

4) 可以通过describe()方法显示数值属性的摘要

```
In [8]: housing.describe()
```

Out[8]:

	longitude	latitude	housing_median_age	total_rooms	total_bedr
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000(
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553
std	2.003532	2.135952	12.585558	2181.615252	421.38507(
min	-124.350000	32.540000	1.000000	2.000000	1.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.00000(
50%	-118.490000	34.260000	29.000000	2127.000000	435.00000(
75%	-118.010000	37.710000	37.000000	3148.000000	647.00000(
max	-114.310000	41.950000	52.000000	39320.000000	6445.0000(

某个属性

另一种快速了解数据类型的方法时绘制每个数值属性的直方图。直方图用来显示给定值范围（横轴）的实例数量（纵轴）

```
%matplotlib inline   # only in a Jupyter notebook
import matplotlib.pyplot as plt
```

这里可以发现一些属性具有重尾现象，为了将分布改成

```
%matplotlib inline   # only in a Jupyter notebook
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```

这里可以发现一些属性具有重尾现象，为了将分布改成钟形，可能需要对这些属性做处理（如计算其对数）

- 创建测试集

  sklearn提供了一些函数，可以通过多种方式将数据集分成多个子集。最简单的是trian_test_split，并且可以把行数相同的多个数据集一次性发送给它，它会根据相同的索引将其拆分（这里是指可以把X和y同时输入）

```
In [10]: a = rng.randn(50,2)

In [11]: b = rng.randn(50)

In [15]: X,test_X,y,test_y = train_test_split(a,b,test_size=0.2)
```

- 从数据探索和数据可视化中获得洞见

  之前只是快速浏览数据，现阶段的目的是再深入一点了解。

  首先，把测试集放在一边，能探索的只有训练集。此外，如果训练集非常庞大，你可以抽样一个探索集，这样后面的操作更简单快捷一些。探索的时候，注意要先创建一个副本，这样可以随便尝试而不损害训 练集：

  housing = train_set.copy()

  - 将地理数据可视化

    由于存在地理位置信息（经度和纬度），因此建立一个各区域的 分布图以便于数据可视化是一个很好的想法

    housing.plot(kind='scatter',x='longtitude',y='latitude')

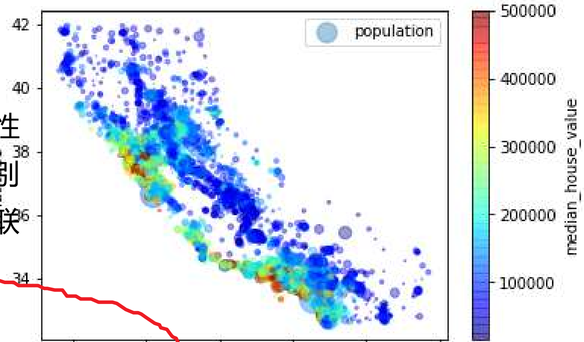
    将alpha选项设置为0.1，可以更清楚地看出高密度数据 点的位置

    housing.plot(kind='scatter',x='longitude',y='latitude',alpha=0.1)

    现在，再来看看房价（见图2-13）。每个圆的半径大小代表了每 个地区的人口数量（选项s），颜色代表价格（选项c）。我们使用一 个名叫jet的预定义颜色表（选项cmap）来进行可视化，颜色范围从 蓝（低）到红（高）

```
housing.plot(kind='scatter',x='longitude',y='latitude',alpha=0.4,
             s=housing['population']/100,label='population',c="median_house_value",
            cmap=plt.get_cmap('jet'),colorbar=True)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f43a93578d0>
```



这里可以发现不同属性之间的某些联系，特别是跟目标属性相关的联系。

  - 寻找相关性

    由于数据集不大，你可以使用corr（）方法轻松计算出每对属性 之间的标准相关系数（也称为皮尔逊相关系数），然后看看每个属性与房屋中位数的相关性分别是多少。

    corr_matrix = housing.corr()

由于数据集不大，你可以使用corr()方法轻松计算出每对属性之间的标准相关系数（也称为皮尔逊相关系数），然后看看每个属性与房屋中位数的相关性分别是多少。

```
corr_matrix = housing.corr()
corr_matrix['median_house_value'].sort_values(ascending=False)
```

相关系数仅测量线性相关性（“如果x上升，则y上升/下 降”）。所以它有可能彻底遗漏非线性相关性（例如“如果x接近于 零，则y会上升”）。需要注意的是，这个相关性跟斜率完全无关。这就好比是说，你本人用英寸来计量的身高与你用英尺甚至是纳米来计 量的身高之间的相关系数等于1。

还有一种方法可以检测属性之间的相关性，就是使用Pandas的 scatter_matrix函数，它会绘制出每个数值属性相对于其他数值属性的相关性。

```
from pandas.tools.plotting import scatter_matrix
attributes = ['median_house_value','median_income','total_rooms','housing_median_age']
scatter_matrix(housing[attributes],figsize=(12,8))
#这里只画出标黄的这几个属性之间的联系的散点图。
```

- 试验不同属性的组合

  在准备给机器学习算法输入数据之前，你要做的最后一件事应该是尝试各种属性的组合。比如，如果你不知道一个地区有多少个家 庭，那么知道一个地区的“房间总数”也没什么用。你真正想要知道的 是一个家庭的房间数量。同样地，单看“卧室总数”这个属性本身，也 没什么意义，你可能是想拿它和“房间总数”来对比，或者拿来同“每 个家庭的人口数”这个属性结合也似乎挺有意思。我们来试着创建这 些新属性：

```
housing['rooms_per_household'] = housing['total_rooms']/housing['households']
housing['bedrooms_per_room'] = housing['total_bedrooms']/housing['total_rooms']
housing['population_per_household'] = housing['population']/housing['households']
```

```
corr_matrix = housing.corr()
corr_matrix['median_house_value'].sort_values(ascending=False)
```

```
median_house_value          1.000000
median_income               0.690647
rooms_per_household         0.158485
total_rooms                 0.133989
housing_median_age          0.103706
households                  0.063714
total_bedrooms              0.047980
population_per_household   -0.022030
population                 -0.026032
longitude                  -0.046349
latitude                   -0.142983
bedrooms_per_room          -0.257419
Name: median_house_value, dtype: float64
```

- 机器学习算法的数据准备

  现在，终于是时候给你的机器学习算法准备数据了。这里你应该 编写函数来执行，而不是手动操作，原因如下：

  ·你可以在任何数据集上轻松重现这些转换（例如，获得更新的 数据库之后）。

  ·你可以逐渐建立起一个转换函数的函数库，在以后的项目中可 以重用。

  ·你可以在实时系统（live system）中使用这些函数来转换新数 据，再喂给算法。

  ·你可以轻松尝试多种转换方式，查看哪种转换的组合效果最佳。

·你可以轻松尝试多种转换方式，查看哪种转换的组合效果最佳。

但是现在，让我们先回到一个干净的数据集（再次复制 train_set），然后将预测器和标签分开，因为这里我们不一定对 它们使用相同的转换方式（需要注意drop（）会创建一个数据副本， 但是不影响train_set）：

```
housing = train_set.drop('median_house_value',axis=1)
housing_labels = train_set['median_house_value'].copy()
```

- 数据清理

  大部分的机器学习算法无法在缺失的特征上工作，所以我们要创 建一些函数来辅助它。前面我们已经注意到total_bedrooms属性有部 分值缺失，所以我们要解决它。有以下三种选择：

  ·放弃这些相应的地区（放弃缺失的行）

  ·放弃这个属性

  ·将缺失的值设置为某个值（0、平均数或者中位数等都可以）

  通过DataFrame的dropna（）、drop（）和fillna（）方法，可以 轻松完成这些操作：

  ```
  housing.dropna(subset=["total_bedrooms"])    # option 1
  housing.drop("total_bedrooms", axis=1)       # option 2
  median = housing["total_bedrooms"].median()
  housing["total_bedrooms"].fillna(median)     # option 3
  ```

  Scikit-Learn提供了一个非常容易上手的教程来处理缺失值：imputer。使用方法如下，首先，你需要创建一个imputer实例，指定你要用属性的中位数值替换该属性的缺失值：

  ```
  from sklearn.preprocessing import Imputer
  imputer = Imputer(strategy='median')
  ```

  由于中位数值只能在数值属性上计算，所以我们需要创建一个没 有文本属性的数据副本ocean_proximity：

  ```
  housing_num = housing.drop('ocean_proximity',axis=1)
  ```

  使用fit（）方法将imputer实例适配到训练集：

  ```
  imputer.fit(housing_num)
  ```

  这里imputer仅仅只是计算了每个属性的中位数值，并将结果存 储在其实例变量statistics_中。现在，你可以使用这个“训练有素”的imputer将缺失值替换成中位 数值完成训练集转换。

  ```
  x = imputer.transform(housing_num)
  ```

  结果是一个包含转换后特征的Numpy数组。如果你想要将它放回 Pandas DataFrame，也很简单：

  ```
  housing_tr = pd.DataFrame(X, columns=housing_num.columns)
  ```

- 处理文本和分类属性

  之前我们排除了分类属性ocean_proximity，因为它是一个文本属 性，我们无法计算它的中位数值。大部分的机器学习算法都更易于跟 数字打交道，所以我们先将这些文本标签转化为数字。Scikit-Learn为这类任务提供了一个转换器LabelEncoder：

  ```
  >>> from sklearn.preprocessing import LabelEncoder
  >>> encoder = LabelEncoder()
  >>> housing_cat = housing["ocean_proximity"]
  >>> housing_cat_encoded = encoder.fit_transform(housing_cat)
  >>> housing_cat_encoded
  array([1, 1, 4, ..., 1, 0, 3])
  ```

  这种代表方式产生的一个问题是，机器学习算法会以为两个相近 的数字比两个离得较远的数字

array([1, 1, 4, ..., 1, 0, 3])

这种代表方式产生的一个问题是，机器学习算法会以为两个相近 的数字比两个离得较远的数字更为相似一些。显然，真实情况并非如 此（比如，类别0和类别4之间就比类别0和类别1之间的相似度更 高）。为了解决这个问题，常见的解决方案是给每个类别创建一个二 进制的属性：当类别是“<1H OCEAN”时，一个属性为1（其他为 0），当类别是“INLAND”时，另一个属性为1（其他为0），以此类 推。这就是独热编码，因为只有一个属性为1（热），其他均为 0（冷）。Scikit-Learn提供了一个OneHotEncoder编码器，可以将整数分类 值转换为独热向量。我们用它来将类别编码为独热向量。值得注意的 是，fit_transform（）需要一个二维数组，但是housing_cat_encoded是一个一维数组，所以我们需要将它重塑：

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> encoder = OneHotEncoder()
>>> housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
>>> housing_cat_1hot
<16513x5 sparse matrix of type '<class 'numpy.float64'>' with 16513 stored elements in Compressed Sparse Row format>
```

注意到这里的输出是一个SciPy稀疏矩阵，而不是一个NumPy数 组。当你有成千上万种类别的分类属性时，这个函数会非常有用。因 为当独热编码完成之后，我们会得到一个几千列的矩阵，并且全是 0，每行仅有一个1。占用大量内存来存储0是一件非常浪费的事情， 因此稀疏矩阵选择仅存储非零元素的位置。而你依旧可以像使用一个 普通的二维数组那样来使用它，当然如果你实在想把它转换成一个 （密集的）NumPy数组，只需要调用toarray（）方法即可。

使用LabelBinarizer类可以一次性完成两个转换（从文本类别转化 为整数类别，再从整数类别转换为独热向量）：

```
from sklearn.preprocessing import LabelBinarizer
encoder = LabelBinarizer()
housing_cat_1hot = encoder.fit_transform(housing_cat)
housing_cat_1hot
```

注意，这时默认返回的是一个密集的NumPy数组。通过发送 sparse_output=True给LabelBinarizer构造函数，可以得到稀疏矩阵。

- 自定义转换器

虽然Scikit-Learn已经提供了许多有用的转换器，但是你仍然需要为一些诸如自定义清理操作或是组合特定属性等任务编写自己的转换 器。你当然希望让自己的转换器与Scikit-Learn自身的功能（比如流水 线）无缝衔接，而由于Scikit-Learn依赖于鸭子类型（duck typing）的 编译，而不是继承，所以你所需要的只是创建一个类，然后应用以下 三个方法：fit（）（返回自身）、transform（）、fit_transform（）。 如果添加TransformerMixin作为基类，就可以直接得到最后一个方 法。同时，如果添加BaseEstimator作为基类（并在构造函数中避免 *args和**kargs），你还能额外获得两个非常有用的自动调整超参数 的方法（get_params（）和set_params（））。例如，我们前面讨论 过的组合属性，这里有个简单的转换器类，用来添加组合后的属性：

```
from sklearn.base import BaseEstimator,TransformerMixin
import numpy as np

rooms_ix,bedrooms_ix,population_ix,household_ix = 3,4,5,6

class CombinedAttributesAdder(BaseEstimator,TransformerMixin):
```

concatenate 的意思

```
rooms_ix,bedrooms_ix,population_ix,household_ix = 3,4,5,6

class CombinedAttributesAdder(BaseEstimator,TransformerMixin):
  def __init__(self,add_bedrooms_per_room=True):
    self.add_bedrooms_per_room = add_bedrooms_per_room
  def fit(self,X,y=None):
    return self
  def transform(self,X,y=None):
    rooms_per_household = X[:,bedrooms_ix]/X[:,rooms_ix]
    population_per_household = X[:,population_ix]/X[:,household_ix]
    if self.add_bedrooms_per_room:
      bedrooms_per_room = X[:,bedrooms_ix]/X[:,rooms_ix]
      return np.c_[X,rooms_per_household,population_per_household,
          bedrooms_per_room]
    else:
      return np.c_[X,rooms_per_household,population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

在本例中，转换器有一个超参数add_bedrooms_per_room默认设 置为True（提供合理的默认值通常是很有帮助的）。这个超参数可以 让你轻松知晓添加这个属性是否有助于机器学习的算法。更广泛地 说，如果你对数据准备的步骤没有充分的信心，就可以添加这个超参 数来进行把关。这些数据准备步骤的执行越自动化，你自动尝试的组 合也就越多，从而有更大可能从中找到一个重要的组合（还节省了大 量时间）。

- 特征缩放

最重要也最需要应用到数据上的转换器，就是特征缩放。如果输 入的数值属性具有非常大的比例差异，往往导致机器学习算法的性能 表现不佳，当然也有极少数特例。案例中的房屋数据就是这样：房间 总数的范围从6到39320，而收入中位数的范围是0到15。注意，目标 值通常不需要缩放。同比例缩放所有属性，常用的两种方法是：最小-最大缩放和标准化。

最小-最大缩放（又叫作归一化）很简单：将值重新缩放使其最 终范围归于0到1之间。实现方法是将值减去最小值并除以最大值和最 小值的差。对此，Scikit-Learn提供了一个名为MinMaxScaler的转换 器。如果出于某种原因，你希望范围不是0～1，你可以通过调整超参 数feature_range进行更改。

标准化则完全不一样：首先减去平均值（所以标准化值的均值总 是零），然后除以方差，从而使得结果的分布具备单位方差。不同于 最小-最大缩放的是，标准化不将值绑定到特定范围，对某些算法而 言，这可能是个问题（例如，神经网络期望的输入值范围通常是0到 1）。但是标准化的方法受异常值的影响更小。例如，假设某个地区 的平均收入等于100（错误数据）。最小-最大缩放会将所有其他值从 0～15降到0～0.15，而标准化则不会受到很大影响。Scikit-Learn提供 了一个标准化的转换器StandadScaler。

- 转换流水线

正如你所见，许多数据转换的步骤需要以正确的顺序来执行。而 Scikit-Learn正好提供了Pipeline来支持这样的转换。下面是一个数值 属性的流水线例子：

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([('imputer',Imputer(strategy='median')),('attribs_adder',
                                        CombinedAttributesAdder()),
                ('std_scaler',StandardScaler())])
housing_num_tr = num_pipeline.fit_transform(housing_num)
```

Pipeline构造函数会通过一系列名称/估算器的配对来定义步骤的 序列。除了最后一个是估算器之外，前面都必须是转换器（也就是 说，必须有fit_transform（）方法）。至于命名，可以随意，你喜欢 就好。

当调用流水线的fit（）方法时，会在所有转换器上按照顺序依次 调用fit_transform（），将一个调用的输出作为参数传递给下一个调 用方法，直到传递到最终的估算器，则只会调用fit（）方法。流水线的方法与最终的估算器的方法相同。在本例中，最后一个 估算器是StandardScaler，这是个转换器，因此Pipeline有 transform（）方法可以按顺序将所有的转换应用到数据中（如果不希 望先调用fit（）再调用transform（），也可以直接调用 fit_transform（）方法）。

现在，你已经有了一个处理数值的流水线，接下来你需要在分类值上应用LabelBinarizer：不然怎么将这些转换加入单个流水线中？ Scikit-Learn为此特意提供了一个FeatureUnion类。你只需要提供一个转换器列表（可以是整个转换器流水线），当transform（）方法被调用时，它会并行运行每个转换器的transform（）方法，等待它们的输出，然后将它们连结起来，返回结果（同样地，调用fit（）方法也会调用每个转换器的fit（）方法）。一个完整的处理数值和分类属性的 流水线可能如下所示：

```
from sklearn.pipeline import FeatureUnion
num_attribs = list(housing_num) cat_attribs = ["ocean_proximity"]
num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_attribs)),
        ('imputer', Imputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
         ('std_scaler', StandardScaler()), ])
cat_pipeline = Pipeline([('selector', DataFrameSelector(cat_attribs)),
        ('label_binarizer', LabelBinarizer()),])
full_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline),
        ("cat_pipeline", cat_pipeline),  ])
```

运行整条流水线：

```
>>> housing_prepared = full_pipeline.fit_transform(housing)
```

每条子流水线从选择器转换器开始：只需要挑出所需的属性（数 值或分类），删除其余的数据，然后将生成的DataFrame转换为 NumPy数组，数据转换就完成了。Scikit-Learn中没有可以用来处理 Pandas DataFrames的[5]，因此我们需要为此任务编写一个简单的自定 义转换器：

```
from sklearn.base import BaseEstimator, TransformerMixin
class DataFrameSelector(BaseEstimator, TransformerMixin):
  def __init__(self, attribute_names):
    self.attribute_names = attribute_names
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    return X[self.attribute_names].values
```

- 选择和训练模型
    - 培训和评估训练集

      先训练一个线性回归模型

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared,housing_labels)
```

      我们可以使用Scikit-Learn的mean_squared_error函数来 测量整个训练集上回归模型的RMSE：

```
from sklearn.metrics import mean_squared_error
housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels,housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
67839.07351617553
```

      结果很差，换decision tree

```
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels,housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
0.0
```

      等等，什么！完全没有错误？这个模型真的可以做到绝对完美 吗？当然，更有可能的是这个模型对数据严重过度拟合了。我们怎么 确认呢？前面提到过，在你有信心启动模型之前，都不要触碰测试 集，所以这里，你需要拿训练集中的一部分用于训练，另一部分用于 模型的验证。

    - 使用交叉验证来更好地进行评估

      以下是执 行K-折（K-fold）交叉验证的代码：它将训练集随机分割成10个不同 的子集，每个子集称为一个折叠（fold），然后对决策树模型进行10 次训练和评估——每次挑选1个折叠进行评估，使用另外的9个折叠进行训练。产出的结果是一个包含10次评估分数的数组：

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg,housing_prepared,housing_labels,
                         scoring='neg_mean_squared_error',cv=10)
scores
array([-4.55044189e+09, -5.33009634e+09, -4.47923445e+09, -5.44920400e+09,
       -5.35217023e+09, -4.93554732e+09, -4.44931329e+09, -4.37591499e+09,
       -4.55396782e+09, -4.87626998e+09])
```

      Scikit-Learn的交叉验证功能更倾向于使用效用函数（越大越好）而不是成本函数（越小越好），所以计算分数的函数实际上是负 的MSE（一个负值）函数，这就是为什么上面的代码在计算平方根之 前会先计算出-scores

      我们再来试试最后一个模型：RandomForestRegressor。在后面的 第7章中，我们将会了解到随机森林的工作原理，通过对特征的随机 子集进行许多个决策树的训练，然后对其预测取平均。在多个模型的 基础之上建立模型，称为集成学习，这是进一步推动机器学习算法的 好方法。这里我们将跳过大部分代码，因为与其他模型基本相同：

```
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared,housing_labels)
housing_predictions = forest_reg.predict(housing_prepared)
```

```
forest_mse = mean_squared_error(housing_labels,housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```
22073.912880820284

```
scores = cross_val_score(forest_reg,housing_prepared,housing_labels,
                 scoring='neg_mean_squared_error',cv=10)
forest_rmse_scores = np.sqrt(-scores)
display_scores(forest_rmse_scores)
```
Scores: [50101.96513946 53930.60847561 51128.97007585 54425.15325041
 55589.97328509 50349.67083652 49866.45949881 53673.04850772
 53543.47211773 52145.80800093]
Mean: 52475.512918812936
Standard deviation: 1925.4835377462196

n_estimators，max_features 以及 bootstrap 均为模型构造要素的参数。

哇，这个就好多了：随机森林看起来很有戏。但是，请注意，训 练集上的分数仍然远低于验证集，这意味着该模型仍然对训练集过度 拟合。过度拟合的可能解决方案包括简化模型、约束模型（即使其正 规化），或是获得更多的训练数据。不过在深入探索随机森林之前， 你应该先尝试一遍各种机器学习算法的其他模型（几种具有不同内核 的支持向量机，比如神经网络模型等），但是记住——别花太多时间 去调整超参数。我们的目的是筛选出几个（2～5个）有效的模型。

每一个尝试过的模型你都应该妥善保存，这样将来你可以轻 松回到你想要的模型当中。记得还要同时保存超参数和训练过的参 数，以及交叉验证的评分和实际预测的结果。这样你就可以轻松地对比不同模型类型的评分，以及不同模型造成的错误类型。通过Python 的pickel模块或是sklearn.externals.joblib，你可以轻松保存Scikit-Learn 模型，这样可以更有效地将大型NumPy数组序列化：

```
from sklearn.externals import joblib
joblib.dump(forest_reg,"forest_reg.pkl")
```
['forest_reg.pkl']

```
fr = joblib.load('forest_reg.pkl')
housing_predictions = fr.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels,housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```
22073.912880820284

- 模型微调

1) 网格搜索

可以用Scikit-Learn的GridSearchCV来替你进行探索。你 所要做的只是告诉它你要进行实验的超参数是什么，以及需要尝试的 值，它将会使用交叉验证来评估超参数值的所有可能的组合。例如， 下面这段代码搜索RandomForestRegressor的超参数值的最佳组合：

```
from sklearn.model_selection import GridSearchCV
param_grid = [{'n_estimators':[3,10,30],'max_features':[2,4,6,8]},
        {'bootstrap':[False],'n_estimators':[3,10],'max_features':[2,3,4]}]
forest_reg = RandomForestRegressor()
grid_search = GridSearchCV(forest_reg,param_grid,cv=5,scoring='neg_mean_squared_error')
grid_search.fit(housing_prepared,housing_labels)
```

这个param_grid告诉Scikit-Learn，首先评估第一个dict中的n_estimator和max_features的所有3×4 = 12种超参数值组合（先不要担 心这些超参数现在意味着什么，我们将在第7章中进行解释），接 着，尝试第二个dict中超参数值的所有2×3 = 6种组合，但这次超参数bootstrap需要设置为False而不是True（True是该超参数的默认值）。

总而言之，网格搜索将探索RandomForestRegressor超参数值的12 + 6 = 18种组合，并对每个模型进行五次训练（因为我们使用的是5折交叉验证）。换句话说，总共会完成18×5 = 90次训练！这可能需要 相当长的时间，但是完成后你就可以获得最佳的参数组合：

```
grid_search.best_params_
```

```
{'max_features': 6, 'n_estimators': 30}
```

因为被评估的n_estimator最大值是30，你还可以试试更高的 值，评分可能还会继续改善。

你可以直接得到最好的估算器

```
grid_search.best_estimator_
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=6, max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=30, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

- 分析最佳模型及其错误

  通过检查最佳模型，你总是可以得到一些好的洞见。例如在进行 准确预估时，Random ForestRegressor可以指出每个属性的相对重要 程度：

  ```
  feature_importances = grid_search.best_estimator_.feature_importances_
  feature_importances
  ```

  array([7.46444380e-02, 6.90607253e-02, 4.12439207e-02, 1.74848995e-02, 1.70039348e-02, 1.84675311e-02, 1.71933555e-02, 3.21707013e-01, 7.76931883e-02, 1.10317957e-01, 7.59693568e-02, 1.69572341e-02, 1.32693368e-01, 1.71789110e-04, 3.19525901e-03, 6.19602958e-03])

- 通过测试集评估系统

  ```
  final_model = grid_search.best_estimator_

  X_test = test_set.drop('median_house_value',axis=1)
  y_test = test_set['median_house_value'].copy()
  X_test_prepared = full_pipeline.transform(X_test)
  final_predictions = final_model.predict(X_test_prepared)

  final_mse = mean_squared_error(y_test,final_predictions)
  final_rmse = np.sqrt(final_mse)
  print(final_mse,final_rmse)
  ```

  2551316257.3341813 50510.55590007084

- 启动、监控和维护系统