机器学习导论
第二章

王小航

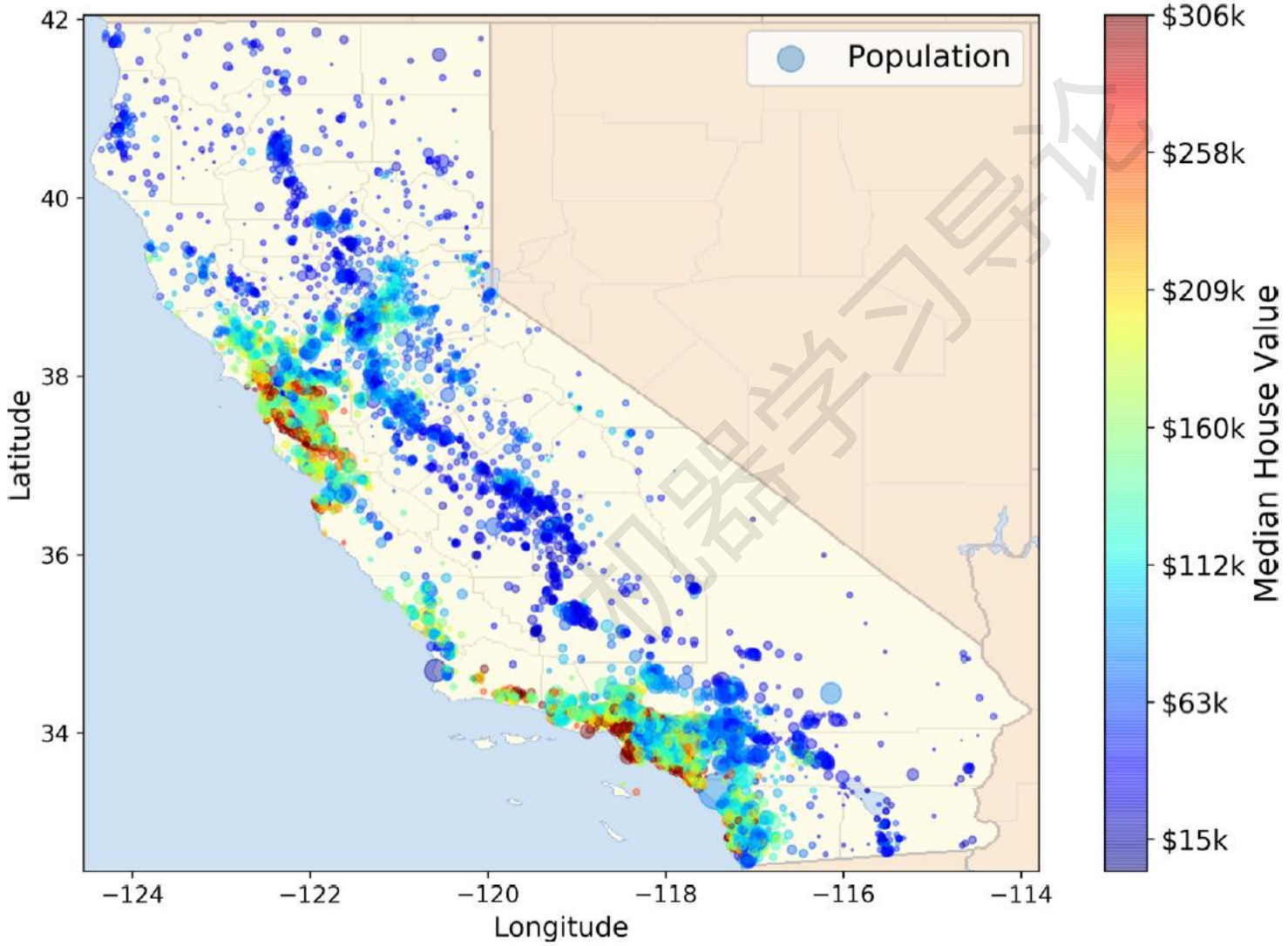
端到端的机器学习项目

- 假设你是一个房地产公司最近新雇用的数据科学家，以下是你将会经历的主要步骤：
  1. 观察大局
  2. 获得数据
  3. 从数据探索和可视化中获得洞见
  4. 机器学习算法的数据准备
  5. 选择并训练模型
  6. 微调模型
  7. 展示解决方案
  8. 启动、监控和维护系统

使用真实数据

- UC Irvine Machine Learning Repository（http://archive.ics.uci.edu/ml/）
- Kaggle datasets（https://www.kaggle.com/datasets）
- Amazon’s AWS datasets（http://aws.amazon.com/fr/datasets/）
- Data Portals（http://dataportals.org/）
- OpenDataMonitor（http://opendatamonitor.eu/）
- Quandl（http://quandl.com/）
- Wikipedia’s list of Machine Learning datasets（https://goo.gl/SJHN2k）
- Quora.com（http://goo.gl/zDR78y）
- The datasets subreddit（https://www.reddit.com/r/datasets）

- 本章我们从StatLib库中选择了加州住房价格的数据集。该数据集基于1990年加州人口普查的数据。



观察大局

- 数据总共有10个属性（在表中可以看到前6个）：longitude、latitude、housing_median_age、total_rooms、total_bedrooms、population、households、median_income、median_house_value以及ocean_proximity。
- 目标：对一个区域房价中位数的预测

	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

观察大局

- 是有监督学习、无监督学习？是分类任务、回归任务还是其他任务？应该使用批量学习还是在线学习技术？
- 这是一个典型的**有监督学习任务**，因为已经给出了标记的训练示例（每个实例都有预期的产出，也就是该区域的房价中位数）。
- 并且这也是一个典型的**回归任务**，因为你要对某个值进行预测。
- 最后，我们没有一个连续的数据流不断流进系统，所以不需要针对变化的数据做出特别调整，所以简单的**批量学习**应该就能胜任。

选择性能指标（Performance Measure）

- 回归问题的典型**性能指标（Performance Measure）**是**均方根误差（RMSE）**：

$$\mathrm{RMSE}(\boldsymbol{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h(\boldsymbol{x}^{(i)}) - y^{(i)}\right)^2}$$

- m是要在其上测量RMSE的数据集中的实例数。
- $\boldsymbol{x}^{(i)}$是数据集中第i个实例的所有特征值（不包括标签）的向量，而$y^{(i)}$是其标签（该实例的期望输出值）。
- h是系统的预测函数，也称为假设。当给系统输入一个实例的特征向量$\boldsymbol{x}^{(i)}$时，它会为该实例输出一个预测值$\hat{y}^{(i)} = h\left(\boldsymbol{x}^{(i)}\right)$。

举例

- 如果你要在2000个区域的验证集上评估RMSE，则m=2000。
- 如果数据集中的第一个区域位于经度-118.29°，纬度33.91°，居民1416人，收入中位数为38 372美元，房屋价值中位数为156 400美元（忽略其他特征），那么

$$x^{(1)} = \begin{pmatrix} -118.29 \\ 33.91 \\ 1416 \\ 38372 \end{pmatrix}, y^{(1)} = 15640$$

- 如果系统预测第一个区域的房价中位数为158 400美元，则$\hat{y}^{(1)} = h\left(\boldsymbol{x}^{(1)}\right) = 158400$。该区域的预测误差为$\hat{y}^{(1)} - y^{(1)} = 2000$。

选择性能指标（Performance Measure）

- 另外一种度量方式为**平均绝对误差**（Mean Absolute Error，MAE，也称为平均绝对偏差）：

$$\mathrm{MAE}(\boldsymbol{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right|$$

- RMSE对异常值比MAE更敏感。假设有许多异常区域，在这种情况下，你可以考虑使用MAE

获取数据

- 用python下载数据
- 查看数据集的简单描述：
  - 数据集中包含20 640个实例.
  - total_bedrooms这个属性只有20 433个非空值，这意味着有207个区域缺失这个特征。
  - 所有属性的字段都是数字，除了ocean_proximity。该列中的值是重复的，这意味着它有可能是一个分类属性。

```
housing.info()
```

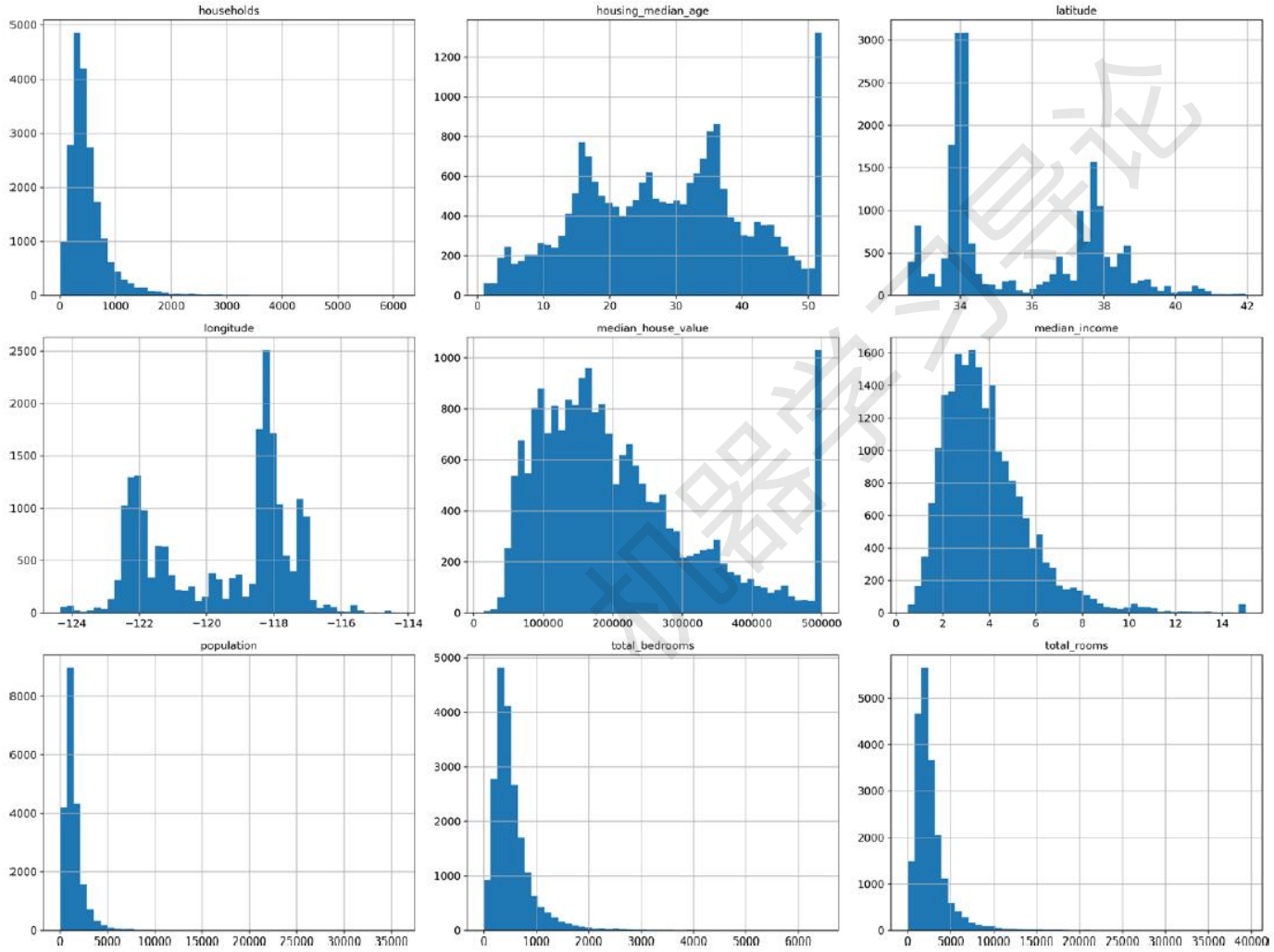
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude           20640 non-null float64
latitude            20640 non-null float64
housing_median_age  20640 non-null float64
total_rooms         20640 non-null float64
total_bedrooms      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
```

数值属性的摘要

```
housing.describe()
```

	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	421.385070
min	-124.350000	32.540000	1.000000	2.000000	1.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.00000

数值属性的直方图


households
housing_median_age
latitude
longitude
median_house_value
median_income
population
total_bedrooms
total_rooms

数值属性的直方图

1. 收入中位数这个属性看起来不像是用美元（USD）在衡量。经与收集数据的团队核实，你得知数字后的单位为万美元。
2. 房龄中位数和房价中位数也被设定了上限。而由于后者正是你的目标属性（标签），因此这可能是个大问题。可以选择：
   a) 对那些标签值被设置了上限的区域，重新收集标签值。
   b) 将这些区域的数据从训练集中移除。
3. 这些属性值的量级不同。
4. 最后，许多直方图都表现出**重尾（tail-heavy）**：图形在中位数右侧的延伸比左侧要远得多。

创建测试集

- 随机抽样：

随机选择一些实例，通常是数据集的20%（如果数据集很大，比例将更小）。

- Scikit-Learn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。最简单的函数是train_test_split()

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

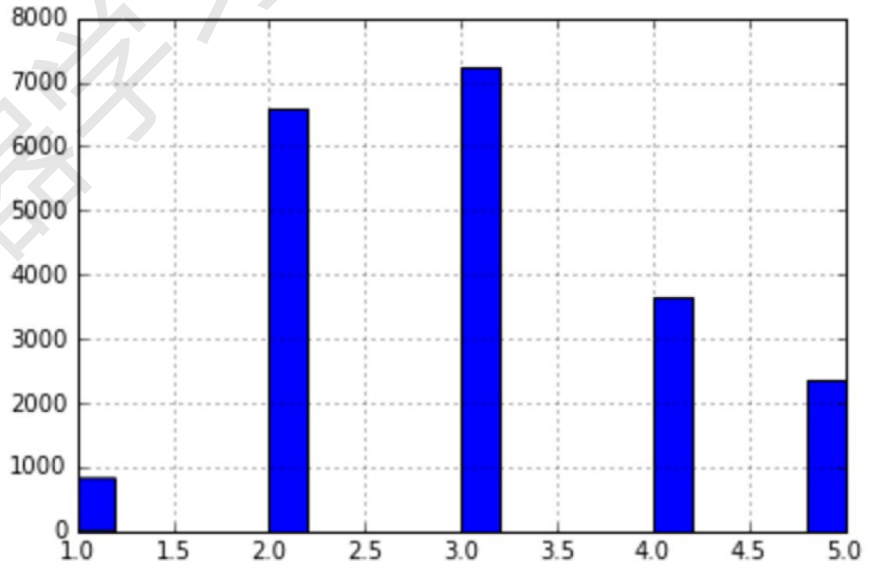
创建测试集

- 分层抽样：

如果一家调查公司想要打电话给1000个人来调研几个问题，他们不会在电话簿中纯随机挑选1000个人。他们试图确保让这1000人能够代表全体人口。例如，美国人口组成为51.3%的女性和48.7%的男性，所以，要想在美国进行一场有效的调查，样本中应该试图维持这一比例，即513名女性和487名男性。

分层抽样

- 按照收入中位数这个属性进行分层抽样：
  - 先创建一个收入类别的属性：0～1.5是类别1，1.5～3是类别2，以此类推
  - 查看收入类别比例分布
  - 根据收入类别进行分层抽样



分层抽样

- 创建一个收入类别的属性：

```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
```

- 查看收入类别比例分布

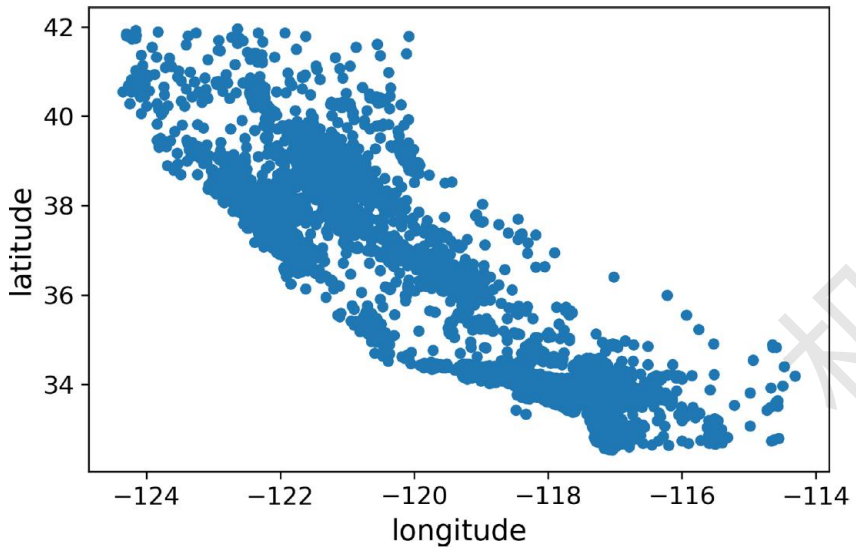
```
housing["income_cat"].hist()
```

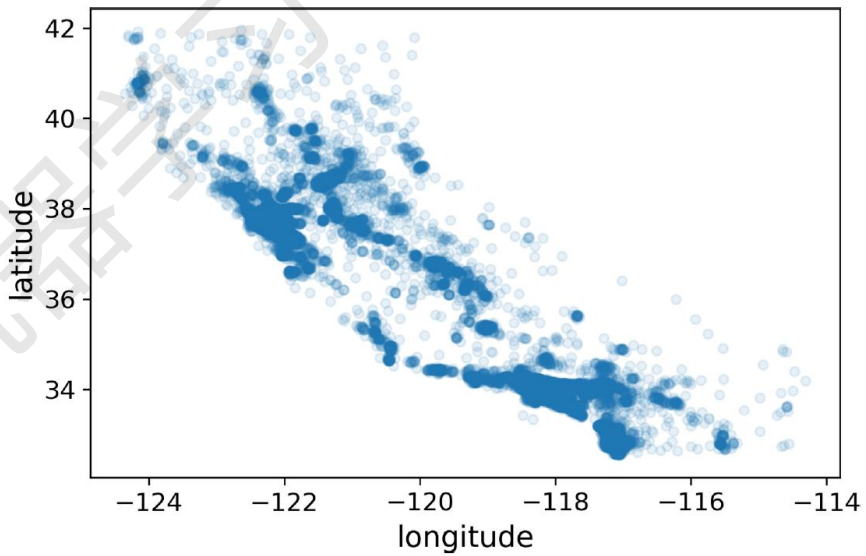
- 根据收入类别进行分层抽样

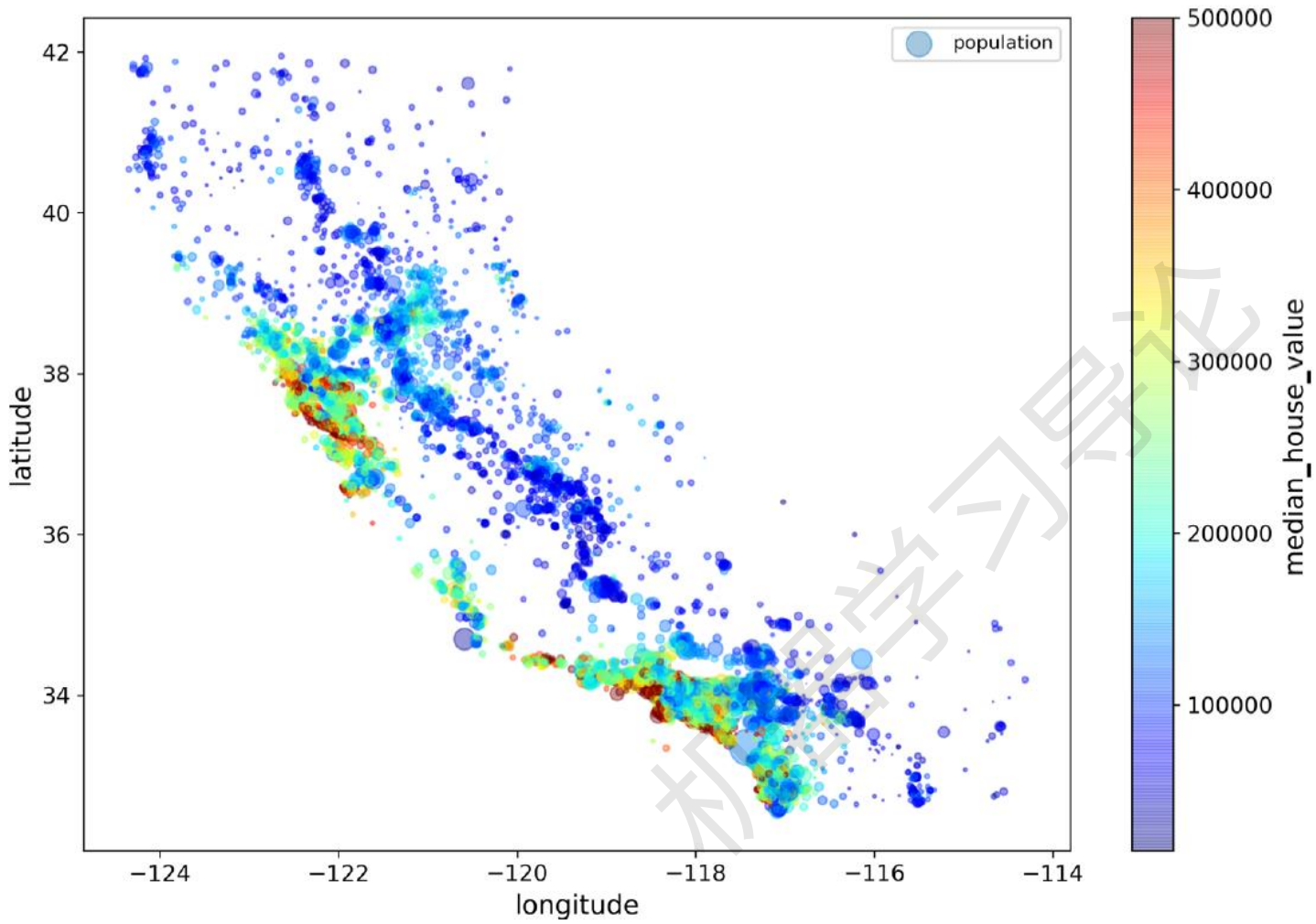
```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

从数据探索和可视化中获得洞见

- 建立一个各区域的分布图：





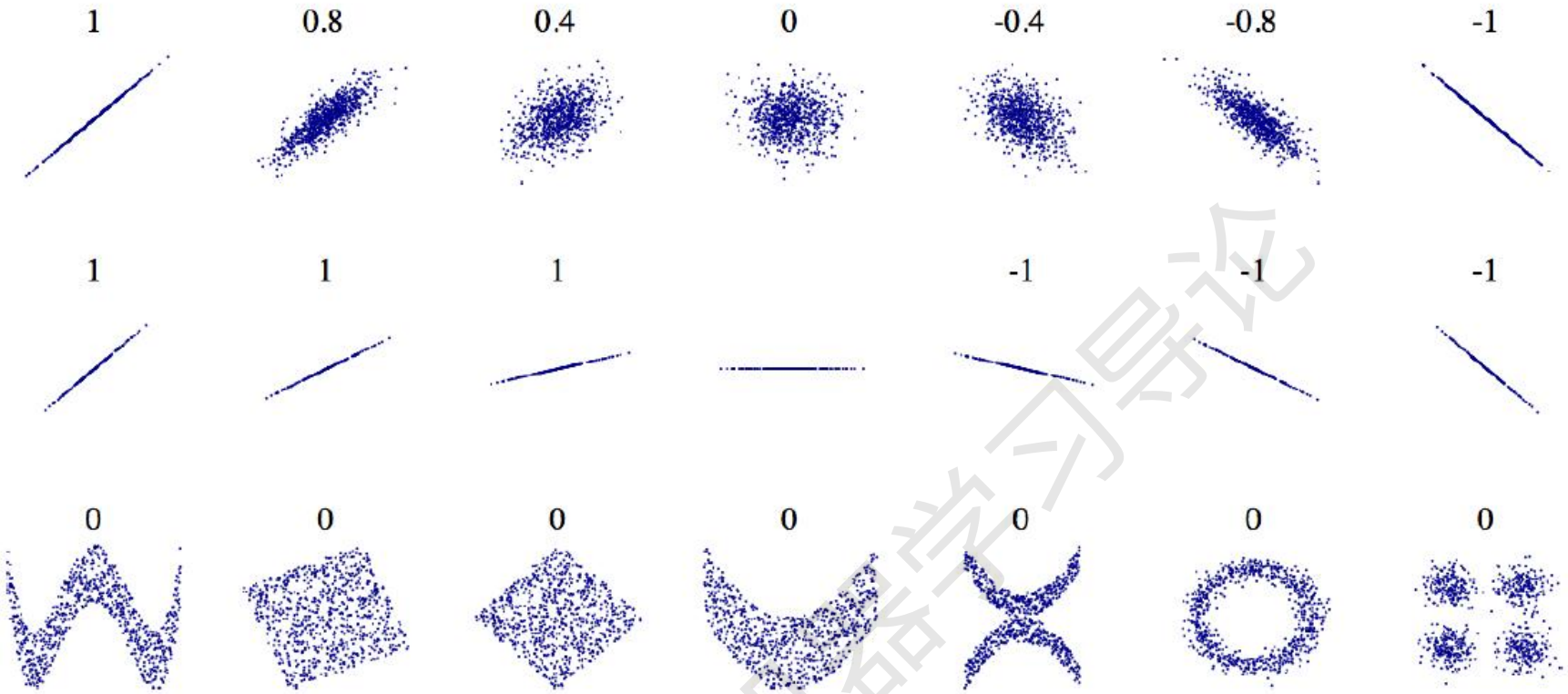


- 每个圆的半径大小代表了每个区域的人口数量，颜色代表价格
- 图表明房价与地理位置（例如靠近海）和人口密度息息相关

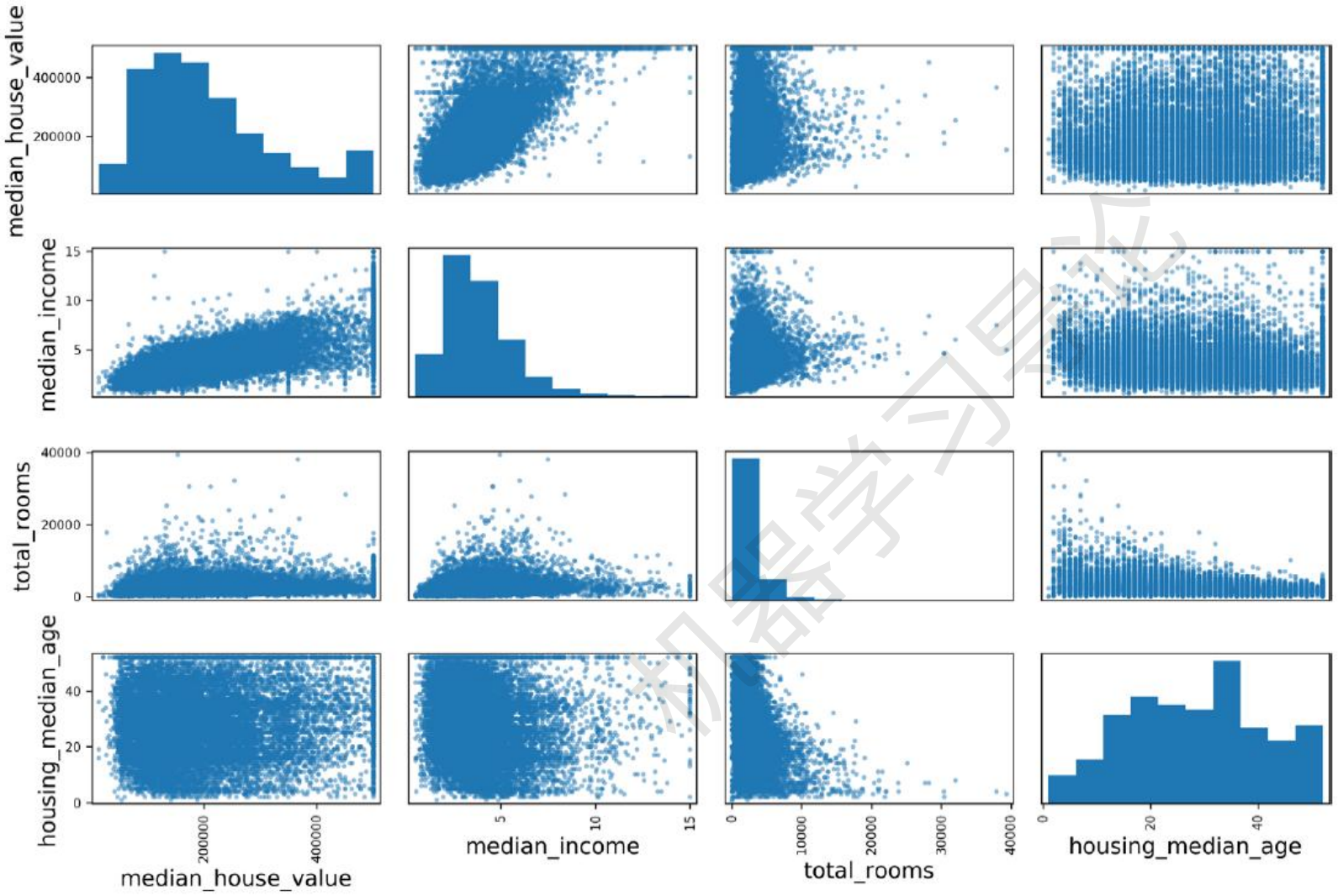
寻找相关性

- 相关系数的范围从-1变化到1。越接近1，表示有越强的正相关。例如，当收入中位数上升时，房价中位数也趋于上升。
- 当系数接近于-1时，表示有较强的负相关。我们可以看到纬度和房价中位数之间呈现出轻微的负相关（也就是说，越往北走，房价倾向于下降。
- 系数靠近0则说明二者之间没有线性相关性。

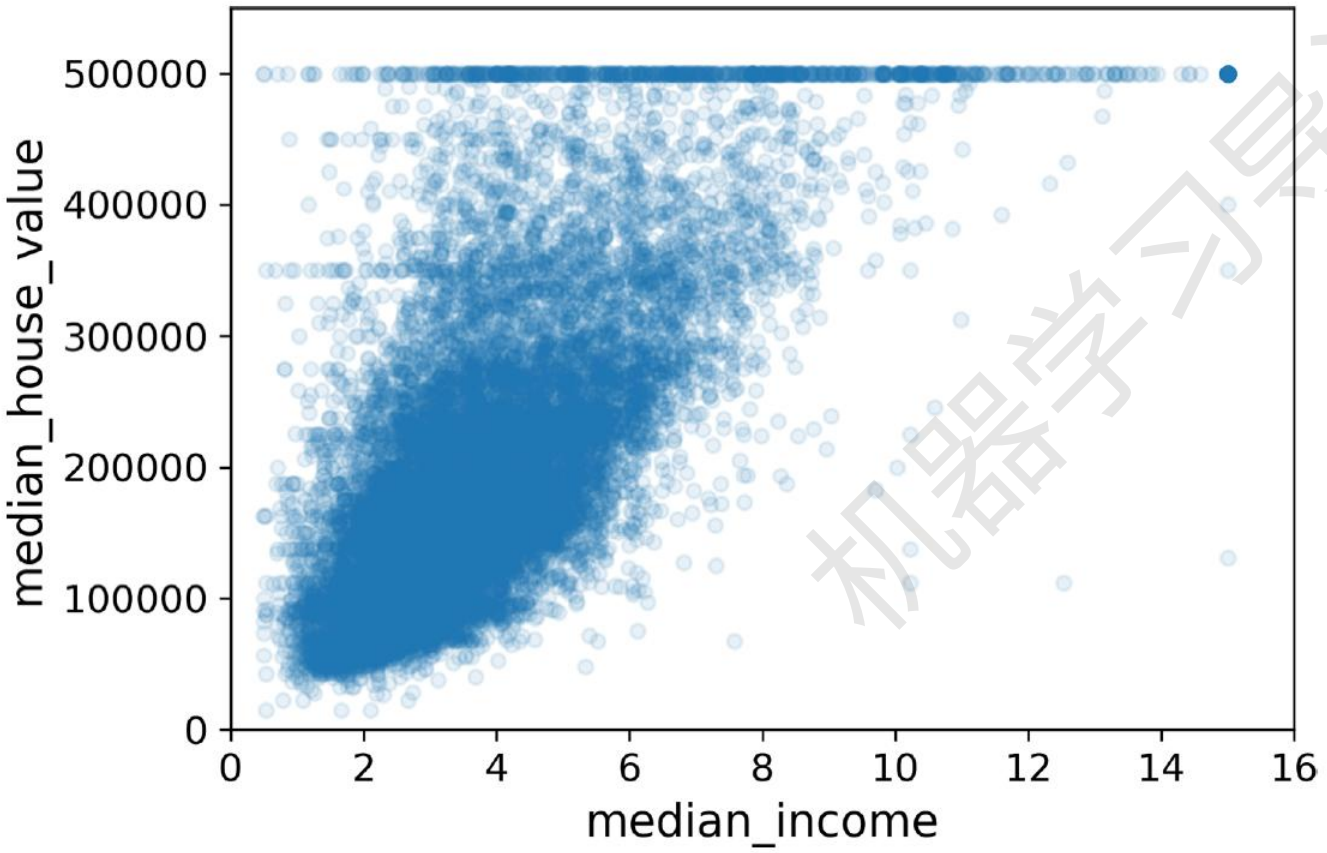
median_house_value	1.000000
median_income	0.687170
total_rooms	0.135231
housing_median_age	0.114220
households	0.064702
total_bedrooms	0.047865
population	-0.026699
longitude	-0.047279
latitude	-0.142826



- 各种数据集的标准相关系数。
- 相关系数仅测量线性相关性（“如果x上升，则y上升/下降”）。
- 相关性跟斜率完全无关。


median_house_value
median_income
total_rooms
housing_median_age
400000
200000
15
10
5
40000
20000
0
40
20
0
200000
400000
5
10
15
0
10000
20000
30000
40000
0
20
40

- 最有潜力能够预测房价中位数的属性是收入中位数，所以我们放大来看看其相关性的散点图：



- 二者的相关性确实很强
- 前面我们提到过50万美元的价格上限在图中是一条清晰的水平线，不过除此之外，图还显示出几条不那么明显的直线。为了避免你的算法学习之后重现这些怪异数据，你可能会尝试删除这些相应区域。

试验不同属性的组合

- 产生三个新的属性：每个家庭的平均房间数量；每个房间的平均卧室数量；每个家庭的平均人口数。

- 新属性bedrooms_per_room较之“房间总数”或“卧室总数”与房价中位数的相关性都要高得多。卧室/房间比例更低的房屋往往价格更贵。
- “每个家庭的房间数量”也比“房间总数”更具信息量——房屋越大，价格越贵。

median_house_value	1.000000
median_income	0.687160
rooms_per_household	0.146285
total_rooms	0.135097
housing_median_age	0.114110
households	0.064506
total_bedrooms	0.047689
population_per_household	-0.021985
population	-0.026920
longitude	-0.047432
latitude	-0.142724
bedrooms_per_room	-0.259984

机器学习算法的数据准备

- 你应该编写函数来执行你的机器学习算法准备数据，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，获得更新的数据集之后）。
  - 你可以逐渐建立起一个转换函数的函数库，可以在以后的项目中重用。
  - 你可以在实时系统中使用这些函数来转换新数据，再输入给算法。
  - 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。

将预测器和标签分开

- 让我们先回到一个干净的训练集（再次复制strat_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strat_train_set）：

```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

数据清理

- 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：
  1. 放弃这些相应的区域。
  2. 放弃整个属性。
  3. 将缺失的值设置为某个值（0、平均数或者中位数等）。
- 如果选择方法3，你需要计算出训练集的中位数值，然后用它填充训练集中的缺失值。
- Scikit-Learn提供了一个非常容易上手的类来处理缺失值：SimpleImputer。

处理缺失值

- SimpleImputer使用方法如下：首先，你需要创建一个SimpleImputer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.impute import SimpleImputer

imputer = SimpleImputer(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ocean_proximity的数据副本：

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

处理缺失值

- 使用fit（）方法将imputer实例适配到训练数据：

```
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：

```
>>> imputer.statistics_
array([ -118.51 , 34.26 , 29. , 2119.5 , 433. , 1164. , 408. , 3.5409])
>>> housing_num.median().values
array([ -118.51 , 34.26 , 29. , 2119.5 , 433. , 1164. , 408. , 3.5409])
```

处理缺失值

- 虽然只有total_bedrooms这个属性存在缺失值，但是我们无法确认系统启动之后新数据中是否一定不存在任何缺失值，所以稳妥起见，还是将imputer应用于所有的数值属性：

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

- 结果是一个包含转换后特征的NumPy数组。将它放回pandas DataFrame：

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

处理文本和分类属性

- 在此数据集中只有一个文本属性：ocean_proximity属性。我们看看前10个实例的值：

```
>>> housing_cat = housing[["ocean_proximity"]]
>>> housing_cat.head(10)
       ocean_proximity
17606        <1H OCEAN
18632        <1H OCEAN
14650       NEAR OCEAN
3230            INLAND
3555         <1H OCEAN
19480           INLAND
8879         <1H OCEAN
13685           INLAND
4937         <1H OCEAN
4861         <1H OCEAN
```

处理文本和分类属性

- 它不是任意文本，而是有限个可能的取值，每个值代表一个类别。因此，此属性是**分类属性**。
- 使用Scikit-Learn的OrdinalEncoder类将文本转到数字：

```
>>> from sklearn.preprocessing import OrdinalEncoder
>>> ordinal_encoder = OrdinalEncoder()
>>> housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
>>> housing_cat_encoded[:10]
array([[0.],
       [0.],
       [4.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.]])
```

处理文本和分类属性

- 以上的表征方式产生的一个问题是，机器学习算法会认为两个相近的值比两个离得较远的值更为相似一些。即针对**有序类别**，如“坏”“平均”“好”“优秀。
- 对于**无序类别**，使用Scikit-Learn的OneHotEncoder编码器创建独热向量：

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> cat_encoder = OneHotEncoder()
>>> housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
>>> housing_cat_1hot
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
  with 16512 stored elements in Compressed Sparse Row format>
```

处理文本和分类属性

- 为了节省内存，这里的输出是一个SciPy稀疏矩阵，而不是一个NumPy数组。
- 此稀疏矩阵选择仅存储非零元素的位置。
- 如果你想把它转换成一个（密集的）NumPy数组，只需要调用toarray（）方法即可：

```
>>> housing_cat_1hot.toarray()
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       ...,
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

自定义转换器

- 虽然Scikit-Learn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组合特定属性等任务编写自己的转换器。
- 你所需要的只是创建一个类，然后应用以下三种方法：fit（）（返回self）、transform（）、fit_transform（）。

```python
from sklearn.base import BaseEstimator, TransformerMixin

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

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_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)
```

特征缩放

- 如果输入的数值属性具有非常大的比例差异，往往会导致机器学习算法的性能表现不佳。
- 案例中的房屋数据就是这样：房间总数的范围从6～39 320，而收入中位数的范围是0～15。
- 注意，目标值通常不需要缩放。
- 同比例缩放所有属性的两种常用方法是**最小-最大缩放（归一化）**和**标准化**。

归一化

- 将值重新缩放使其最终范围归于0～1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。
- Scikit-Learn提供了一个名为MinMaxScaler的转换器。

标准化

- 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。
- 但是标准化的方法受异常值的影响更小。
- Scikit-Learn提供了一个标准化的转换器StandadScaler。

转换流水线

- Pipeline构造函数会通过一系列名称/估算器的配对来定义步骤序列。

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

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

转换流水线

- 到目前为止，我们分别处理了类别列和数值列。
- 拥有一个能够处理所有列的转换器会更方便，将适当的转换应用于每个列。Scikit-Learn为此引入了ColumnTransformer。

```
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

选择和训练模型

- 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)
```

- 让我们用几个训练集的实例试试：

```
>>> some_data = housing.iloc[:5]
>>> some_labels = housing_labels.iloc[:5]
>>> some_data_prepared = full_pipeline.transform(some_data)
>>> print("Predictions:", lin_reg.predict(some_data_prepared))
Predictions: [ 210644.6045  317768.8069  210956.4333  59218.9888  189747.5584]
>>> print("Labels:", list(some_labels))
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

评估训练集

- 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
68628.19819848922
```

评估训练集

- 大多数区域的median_housing_values分布在120 000～265 000美元之间，所以典型的预测误差达到68 628美元只能算是差强人意。
- 这就是一个典型的模型对训练数据欠拟合的案例。
- 这种情况发生时，通常意味着这些特征可能无法提供足够的信息来做出更好的预测，或者是模型本身不够强大。
- 想要修正欠拟合，可以通过选择更强大的模型，或为算法训练提供更好的特征。

选择和训练模型

- 训练一个DecisionTreeRegressor。
- 这是一个非常强大的模型，它能够从数据中找到复杂的非线性关系（关于决策树在第6章会有更详细的介绍。）

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)
```

评估训练集

- 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：

```
>>> 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
```

- 这个模型对数据严重过拟合了
- 你需要拿训练集中的一部分用于训练，另一部分用于模型验证。

使用交叉验证来进行评估

- 一个不错的选择是使用Scikit-Learn的K-折交叉验证功能。
- 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，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。
- 产生的结果是一个包含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)
tree_rmse_scores = np.sqrt(-scores)
```

使用交叉验证来进行评估

- K-折交叉验证结果：

```
>>> def display_scores(scores):
...     print("Scores:", scores)
...     print("Mean:", scores.mean())
...     print("Standard deviation:", scores.std())
...
>>> display_scores(tree_rmse_scores)
Scores: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
 71115.88230639 75585.14172901 70262.86139133 70273.6325285
 75366.87952553 71231.65726027]
Mean: 71407.68766037929
Standard deviation: 2439.4345041191004
```

- 这次的决策树模型好像不如之前表现得好。

使用交叉验证来进行评估

- 让我们也计算一下线性回归模型的评分：

```
>>> lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
...                              scoring="neg_mean_squared_error", cv=10)
...
>>> lin_rmse_scores = np.sqrt(-lin_scores)
>>> display_scores(lin_rmse_scores)
Scores: [66782.73843989 66960.118071   70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
Mean: 69052.46136345083
Standard deviation: 2731.674001798348
```

- 决策树模型确实是严重过拟合了，以至于表现得比线性回归模型还要差。

选择和训练模型

- 我们再来试试最后一个模型：RandomForestRegressor。在第7章中，我们将会了解到随机森林的工作原理。
- 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)
>>> [...]
>>> forest_rmse
18603.515021376355
>>> display_scores(forest_rmse_scores)
Scores: [49519.80364233 47461.9115823  50029.02762854 52325.28068953
 49308.39426421 53446.37892622 48634.8036574  47585.73832311
 53490.10699751 50021.5852922 ]
Mean: 50182.303100336096
Standard deviation: 2097.0810550985693
```

微调模型

- 你可以用Scikit-Learn的GridSearchCV来调整超参数。
- 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组合。
- 例如，下面这段代码搜索RandomForestRegressor的超参数值的最佳组合。

网格搜索

```
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',
                           return_train_score=True)

grid_search.fit(housing_prepared, housing_labels)
```

网格搜索

- 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。
- 接着，尝试第二个dict中超参数值的所有2×3=6种组合。
- 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5次训练（因为我们使用的是5-折交叉验证）
- 总共会完成18×5=90次训练。

网格搜素

- 完成后你就可以获得最佳的参数组合：

```
>>> grid_search.best_params_
{'max_features': 8, 'n_estimators': 30}
```

- 在本例中，我们得到的最佳解决方案是将超参数max_features设置为8，将超参数n_estimators设置为30。
- 这个组合的RMSE分数为49 682，略优于之前使用默认超参数值的分数50 182。你成功地将模型调整到了最佳模式！

网格搜素

- 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=8, 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=None, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

通过测试集评估系统

- 从测试集中获取预测器和标签，运行full_pipeline来转换数据（调用transform（）而不是fit_transform（）），然后在测试集上评估最终模型：

```
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_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)   # => evaluates to 47,730.2
```

通过测试集评估系统

- 你可以使用scipy.stats.t.interval（）计算泛化误差的95%置信区间：

```
>>> from scipy import stats
>>> confidence = 0.95
>>> squared_errors = (final_predictions - y_test) ** 2
>>> np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
...                          loc=squared_errors.mean(),
...                          scale=stats.sem(squared_errors)))
...
array([45685.10470776, 49691.25001878])
```