Chapter 2
End to End Machine Learning

DuLi

2019 年 4 月 8 日

目录

1 Outline

Here are main steps you will go through:

1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data to gain insights.
4. Prepare the data fpr machine learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.
8. Launch,monitor,and maintain your system.

2 Working with Real Data

Here are a afew places you can look to get data:

- Popular open data open repositories:
- UC Irvine Machine Learning Repositories.
- Kaggle datasets.
- Amazon's AWS datasets.
- Meta Portals(they list open data repositories)
- dataportals.org
- opendatamonitor.eu
- quandl.com
- Other pages listing many popular open data repositories

- Wikipedia's list of Machine Learning datasets.
- Quora.com question.
- Datasets subreddit.

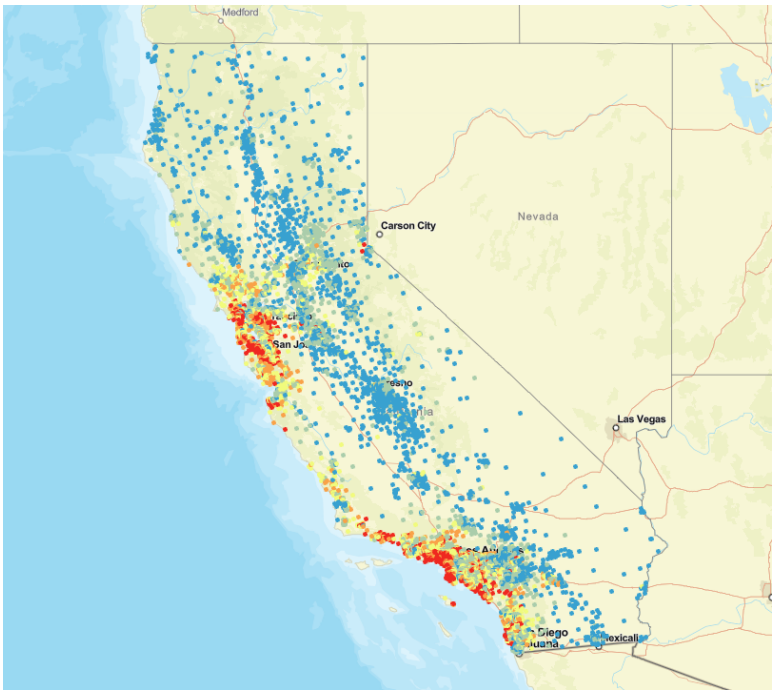


图 1: California Housing Prices Databases

3 Look at the Big Picture

3.1 Frame the Problem

The first question to ask you is what exactly is the bussiness objective;building a model is probably not the end goal.How do you expect use and benefit from this model?This is improtant because it will determine how you frame the problem,what algorithms you will select,what performance measure you will use to evaluate your model,and how much effort you should spend tweaking it. Your model output (a predicting of a district's median housing price) will be fed to another Machine Learning system along with

many other signals.This downstream system will determine whether it is worth investing in a given area or not.

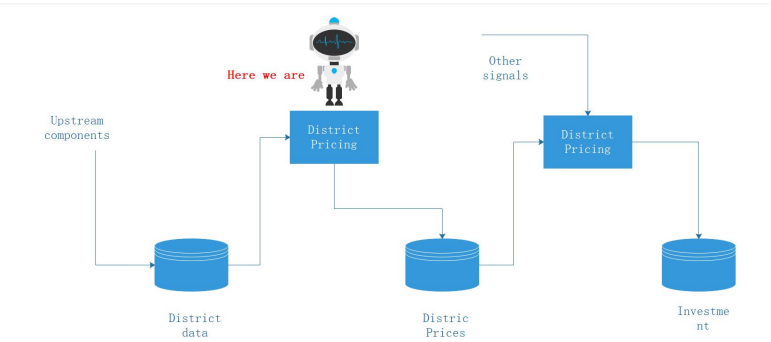


图 2: A machine learning pipeline for real estate investment

The next step is framing the problem:is it supervised,unsupervised,or Reinforcement learning?Is it a classification task,a regression task,or something else?Should we use batch learning or online learning technique? Clearly it is supervised,we need historical data to train the model.Moreover it is a typical regression task,more specifically,this is a multivariate regression since the system will use mutiple features to make a prediction.In first chapter,we predicted life satisfiction based on just one feature,the GDP per captia.Finally,there is no continuous flow of data coming in the system,there is no particular need to adjust to changing data rapidly,and the data is small enough to fit in memory,so plain batch learning should do just fine.

3.2 Select a Performance Measure

The next step is to select a performance measure.A typical performance measure for regression problems is the Root Mean Square Error(均方根误差),it measures the standard deviation of the errors the system makes in its prediction.

$$RMSE(X,h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(X^i) - y^i)^2} \tag{1}$$

Even though the RMSE is generally the perferred performance measure for regression tasks,in some

contexts you may prefer to use another function.For example,suppose that there are many outlier districts.In that case,we may consider using the Mean Absolute Error(平均绝对误差):

$$MAE(X,h) = \frac{1}{m}\sum_{i=1}^{m}|h(X^i) - y^i| \tag{2}$$

Both the RMSE and MAE are ways to measure the *distance* between two vectors.Various distance measures or norms are possible:

- Euclidean norm(欧几里得距离)。
- Manhattan norm(曼哈顿距离),it measures the distance between two points in a city if you can only travel along orthogonal city blocks.也就是指城市中两点之间沿着街区边缘走路的距离。
- More generally,the l_k norm of a vector v containing n elements is defined as:

$$\| v \| = (\mid v_o \mid^k + \mid v_1 \mid^k + \cdots + \mid v_n \mid^k)^{\frac{1}{k}} \tag{3}$$

4 Get the data

It's time to get your hands dirty.

4.1 Create the workspace

First,you need to have Python enviornment installed.We recommand you installed anaconda on your computer.https://www.anaconda.com/

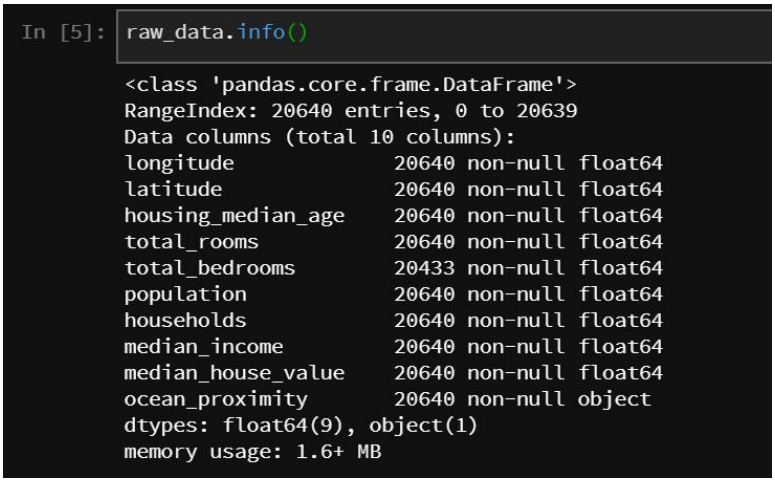
4.2 Take a Quick Look at the Data Structure

Let's take a look a the Data Structure.I download the database from Kaggle.https://www.kaggle.com/camnugent/california-housing-prices.Let's take a glance of the data structure. Each row represent one district.There are 10 attributes(Figure 3):longitude,lattidute,housing_median_age,total_rooms,total_bedrooms,population,households,median_income,median_house_value,ocean_proximity.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	269700.0	NEAR BAY
6	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0	NEAR BAY
7	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.1200	241400.0	NEAR BAY
8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	NEAR BAY
9	-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0	NEAR BAY
10	-122.26	37.85	52.0	2202.0	434.0	910.0	402.0	3.2031	281500.0	NEAR BAY
11	-122.26	37.85	52.0	3503.0	752.0	1504.0	734.0	3.2705	241800.0	NEAR BAY
12	-122.26	37.85	52.0	2491.0	474.0	1098.0	468.0	3.0750	213500.0	NEAR BAY
13	-122.26	37.84	52.0	696.0	191.0	345.0	174.0	2.6736	191300.0	NEAR BAY

图 3: Data Structure

The info method is useful to get a quick description of the data,in particular the total number of rows,and each attribute's type and number of non-null values.There are 20640 instances in the dataset,which means that it is fairly small by Machine Learning standards,but it's perfect to get started.Notice that the total_bedrooms attribute has only 20433 non-null values,meaning that 207 districts are missing this feature.We will need to take care of this later.



```
In [5]: raw_data.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
```

图 4: Data Info

All attribute are numercial,except the ocean_proximity field.We can find out what categories exist and how many districts belong to each category by using the value_counts() method:

```
In [6]: raw_data['ocean_proximity'].value_counts()

Out[6]: <1H OCEAN     9136
        INLAND        6551
        NEAR OCEAN    2658
        NEAR BAY      2290
        ISLAND           5
        Name: ocean_proximity, dtype: int64
```

图 5: Data Counts

Let's look at the other fields.The describe() method shows a summary of the numercial attributes.(Figure 6).The count,mean,min and max rows are self-explanatory.

```
In [7]: raw_data.describe()
```

Out[7]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

图 6: Data Describe

Another quick way to get a feel of the type of data you are dealing with is to plot a histogram for each numercial attribute.

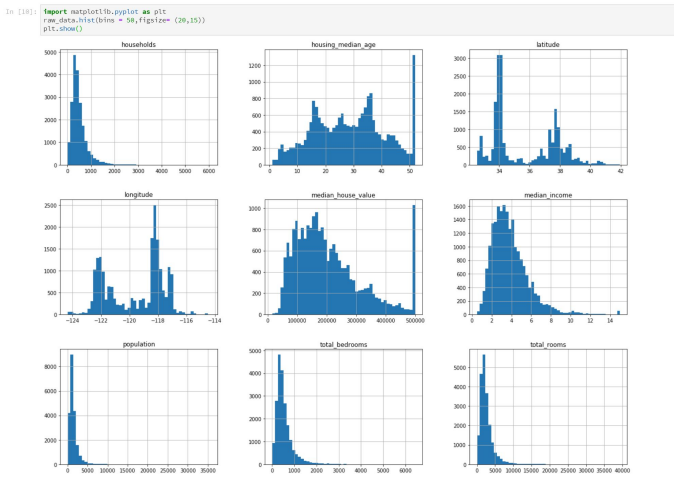


图 7: A histogram for each numercial attribute

4.3 Create a Testset

Create a test set is theoretically quite simple:just pick some instances randomly,typically 20% of the dataset,and set them aside.

```
import numpy as np

def split_train_test(data,test_ratio):
  shuffled_indices = np.random.permutation(len(data))
  test_set_size = int(len(data)*test_ratio)
  test_indices = shuffled_indices[:test_size]
  train_indices = shuffled_indices[test_size:]
  return data.iloc[train_indices],data.iloc[test_indices]
```

We can then using the function like this:

```
train_set,test_set = split_train_test(housing,0.2)
```

This works,but it not perfect:if you run the program again,it will generate a different test set.One solution is to save the test set on the first run and then load it in subsequent runs.Another option is to set the random number generate's seed(eg. np.random.seed(52))before calling np.permutation().But both these solution will break next time you fetch an updated dataset.A common solution is to use each instance's identifier to decide whether or not it should go in the test set.For example,you could compute a hash of each instance's identifier.

```
import hashlib

def test_set_check(identifier,test_ratio,hash):
    return hash(np.int64(identifier).digest()[-1]<256*test_ratio)

def split_train_test_by_id(data,test_ratio,id_column,hash=hashlib.md5):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_:test_set_check(id_,test_ratio,hash))
    return data.iloc[~in_test_set],data.iloc[in_test_set]
```

Unfortunately,the hosuing dataset does not have an identifier column.The simplest solution is to use the row index as the ID:

```
housing_with_id = housing.reset_index()
train_set,test_set = split_train_test_by_id(hosuing_with_id,0.2,"index")
```

We can utilize the train_testsplit function to reah the same effect of the split function above.

```
from sklearn.model_selection import train_test_split

train_set,test_set = train_test_split(housing,test_size = 0.25,random_state = 0)
```

This generally fine if your dataset is large enough,bu if it is not,you run the risk of introducing a significant sampling bias.For example,when a survey company decides to call 1000 people to ask them a few questions ,they don't just pick 1000 people randomly in a phone booth.They try to ensure that these 1000 people are representative of the whole population,eg. the US population is composed of 51.3% female and 48.7% male,so a survey in the US should try to maintain this ratio in the sample:513 female and 487 male.This is called *stratified sampling.*

Since the median income is a very important attribute to predict median housing prices. You may want to ensure that the test set is representative of the various categories of incomes in the whole dataset

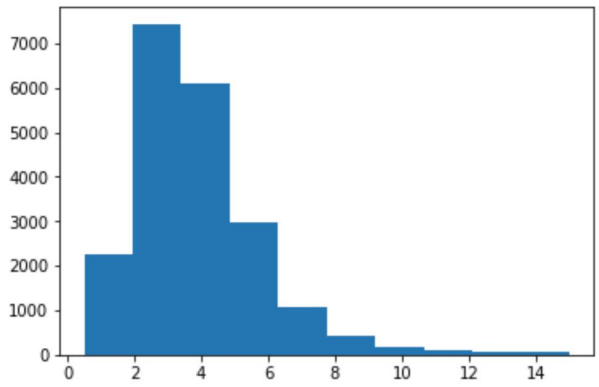


图 8: Histogram of income categories

Let's look at the histogram of income categories,most median income values are clustered around 2-5(tens of thousands of dallors),but some median incomes go far beyond 6.It is important to have a sufficient number of instances in your dataset for each stratum,or else the estimate of the stratum'simportance may br biased,so we should not have too many strata(dividing the median income by 1.5 and rounding up using cel and then merge all the categories greater than 5 into category 5).

```
raw_data["income_cat"] = np.ceil(raw_data["median_income"]/1.5)
raw_data["income_cat"].where(raw_data["income_cat"]>5,5.0,inplace = False)
```

Now you are ready to do stratified sampling based on the income category.For this we can use Scikit-learn's StratifiedShuffleSplit class:

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits = 1,test_size = 0.2, random_state = 17)

for train_index, test_index in split.split(raw_data, raw_data["income_cat"]):
    strat_train_set = raw_data.loc[train_index]
    strat_test_set = raw_data.loc[test_index]
```

By this way,the category proportions in the test_set which generated with stratified sampling almost identical to those in the full dataset.

We spent quite a bit of time on the test set generation for a good reason:this is an often neglected but critical part of a Machine Learning project. At last,we should remove the income_cat attribute so the data is back to its original state:

```
for set in(strat_train_set,strat_test_set):
  set.drop(["income_cat"],axis = 1,inplace = True)
```

这个部分看似平淡无奇，其实还蛮有用的，之前在选数据集的时候，从来没有考虑过这个问题。回头想想，选择有代表性的数据集对于一个监督学习的系统来说，还是非常重要的。使得在训练阶段也能提高精度，这个道理我想是不言自明的。

5 Discover and Visualize the Data to Gain Insights

So far we have only taken a quick glance at the data to get a general understanding of the kind of data we are manipulating.In our case,the set is quite small so you can just work directly on the full set.Let's create a copy so you can play with it without harming the training set.

```
housing = strat_train_set.copy()
```

5.1 Visualizing Geographical Data

Since there is geographical information(latitude and longitude),let's plot it!

```
1 housing.plot(kind="scatter",x="longitude",y="latitude")
```

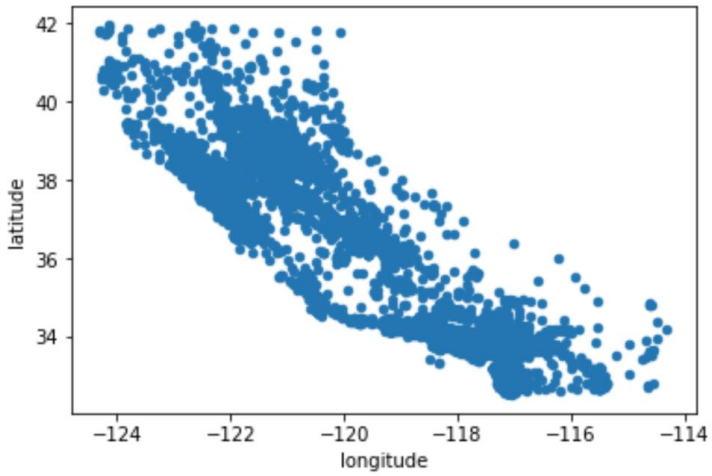


图 9: A geographical scatterplot of the data

Setting the alpha option to 0.1 makes it much easier to visualize the places where there is a high density of data points.

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

alpha:float (0.0 transparent through 1.0 opaque),这里的alpha指的是透明度,所以密度越大的地方，因为重叠的原因，颜色就会越深。

It's better now,let's look at the housing prices.The radius of each circle represents the district's population(option s),and the color represents the price(option c).We will use a predefined color map(option cmap)called jet,which ranges from blue(low values) to red(high prices):

```
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()
```

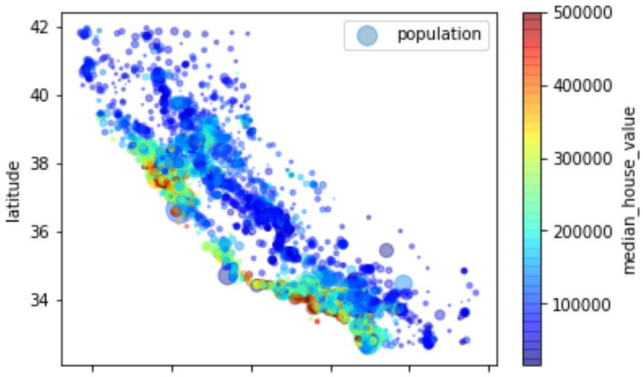


图 10: California housing prices

观察最后得到的图我们大概能看得出一些结论，这些结论也与我们的常识相符。首先，房价和位置的关系很大，临海的房价普遍要高一些，还有某些位置的房价普遍要高一些，这些地方应该是加州沿海的一些城市，旧金山，洛杉矶等等。当然这也不是全部，北部沿海的房价也会低一些。

5.2 Looking for Correlations

Since the data set is not too large ,we can easily compute the *Standard correlation coefficient*(also called Pearson's r 皮尔森相关系数) between every pair of attributes using the corr() method:

```
corr_matrix = housing.corr()
```

The matrix is big,so let's look at a specific attribute(eg.median house value).

```
1 corr_matrix["median_house_value"].sort_values(ascending=False)
```

The correlation coefficient ranges from 1 to -1.接近1代表正相关，接近-1代表负相关，我们可以看出平均房价和收入是正相关关系，而和经度是负相关的，也就是内陆房价低，沿海高。Finally,coefficients close to zero mean there is no linear correlation.

```
median_house_value    1.000000
median_income         0.689774
total_rooms           0.137847
housing_median_age    0.098007
households            0.069283
total_bedrooms        0.053267
population           -0.023161
longitude            -0.048948
latitude             -0.140303
Name: median_house_value, dtype: float64
```

下面这张图来自维基百科，从左到右依次表示皮尔森相关系数在不同值时的情况。

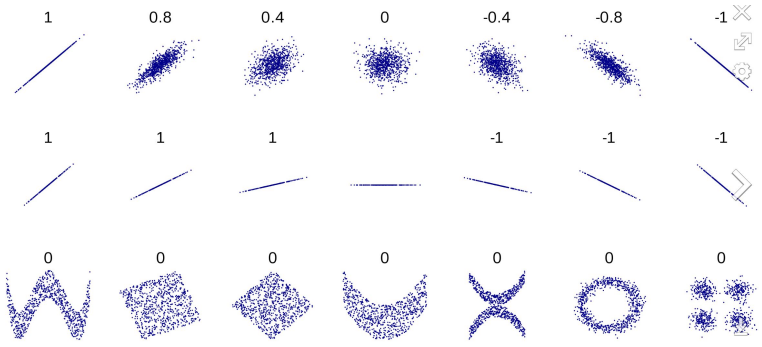


图 11: Scatter Matrix

The correlation coefficient only measures linear correlations.

Another way to check for correlations between attributes is to use Pandas' scatter_matrix function.Since there are now 11 numerical attributes,you would get 11^2 plots,so let's just focus on a few promising attributes that seem most correlated with the median housing value.

```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value","median_income","total_rooms","housing_median_age"]

scatter_matrix(housing[attributes],figsize=(12,8))
```

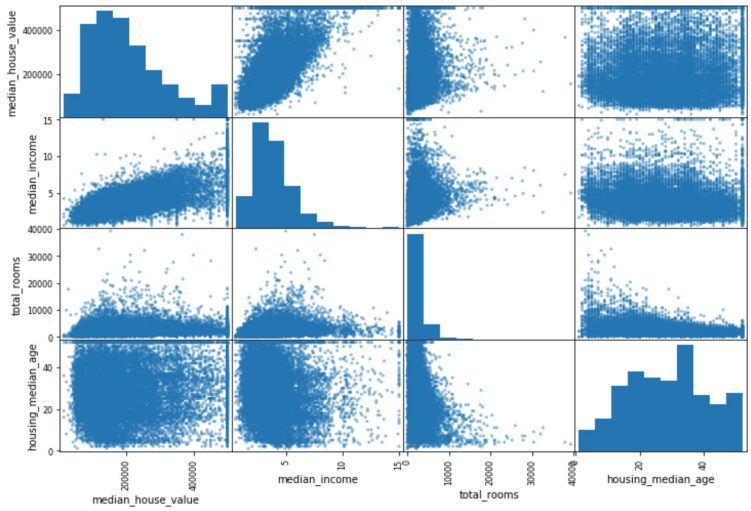


图 12: Scatter Matrix

这里简单做一个说明，之几张图是将数据按照各自的位置两两进行绘图，按照皮尔森系数的意义，两个数据集在一起越接近一条直线，相关性越好，所以在这张图上可以明显的看出，median_income参数和median_house_相关性是最好的。这个也和之前计算的结果相符。接下来重点关注这个两个参数之间的关系。

```
housing.plot(kind="scatter",x="median_income",y="median_house_value",alpha = 0.1)
```

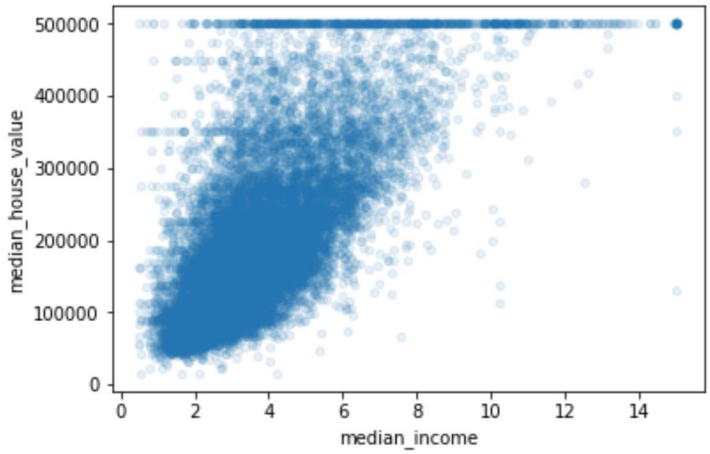


图 13: Median income versus median house value

This plot reveals a few things.First,the correlation is indeed very strong;we can clearly see the upward trend and the points are not too dispersed.Second,the price cap is clearly visible as a horizontial line at 500000，and this plot reveals other less obvious straight lines:450000,350000.

5.3 Experinment with attribute Combinations

One last thingwe may want to do before actually preparing the data for Machine Learning algorithms is try out various attribute combinations.简单的举个例子，就是说单单看一个地区卧室的总数并不是非常有用的，还必须结合房间数或者人口数才真正的有意义。接下来在数据集上，我们把这几个属性添上。

```
housing["roms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_room"] = housing["population"]/housing["total_rooms"]
```

接下来再算一次Pearson's r:

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

```
median_house_value    1.000000
median_income         0.689774
total_rooms           0.137847
housing_median_age    0.098007
households            0.069283
total_bedrooms        0.053267
population           -0.023161
longitude            -0.048948
latitude             -0.140303
Name: median_house_value, dtype: float64
```

(a) Original correlation

```
median_house_value     1.000000
median_income          0.689774
roms_per_household     0.148177
total_rooms            0.137847
housing_median_age     0.098007
households             0.069283
total_bedrooms         0.053267
population            -0.023161
population_per_room   -0.030419
longitude             -0.048948
latitude              -0.140303
bedrooms_per_room     -0.257873
Name: median_house_value, dtype: float64
```

(b) Combination correlation

图 14: Comparation of two cases

值得注意的是，在相关性的计算上，最坏的情况是0，无论是正相关还是负相关都是好的，也就是说越接近1或者-1都好，所以在这里看出，计算后的bedrooms_room是好于卧室数以及房间数的。同时，rooms_household也是同样的情况。在后续我们运行系统之后，会得到比现在更加深入的信息，这是一个迭代并不断优化的过程。

6 Prepare the Data for Machine Learning Algorithms

数据预处理是一个重要的过程，是常见的处理过程，在项目和项目之间大概是可以通用的。首先要做的是数剧的分离，即原数据集和label的分离。

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

Note that drop() creates a copy of the data and does not affect strat_train_set.

6.1 Data Cleaning

Most Machine Learning algorithms cannot work with missing features.We noticed that the total_bedrooms attribute has some missing values,let's fix this:

- Get rid of corresponding districts.
- Get rid of the whole attribute.
- Set the value to some value(0,mean,median,etc)

We can accomplish these easily using DataFrame's dropna(),drop(),and fillna() methods:

```
housing.dropna(subset=["total_bedrooms"])
housing.drop("total_bedrooms",axis=1)
hosuing["total_bedrooms"].fillna(median)
```

Scikit-Learn provide a handy class to take care of missing values:Imputer.Here is how to use it.First,we need to crate an Imputer instance,specifying that you want to replace each attribute's missing values with the median of that attribute:

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

Since the median can only be computed on numercial attributes,we need to create a copy of the data without the text attribute ocean_proximity:

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

Now we can fit the imputer instance to the training data using the fit() method:

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

The imputer has simply computed the median of each attribute and stored the result in the statistics_ instacne varible.

```
imputer.statistics_
housing_num.median().values
```

Now we can use this imputer to transform the training set by replacing missing values by the learned medaians:

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

The result is a plain Numpy array containing the transformed features.If you want to put it back into a Pandas DataFrame,it's simply:

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

6.2 Handling Text and Categrocial Attributes

Earlier we left out the categorical attribute ocean_proximity beacuse it is a text attribute,let's convert these text labels to numbers.

Scikit-Learn provides a transformer for this task called LabelEncoder:

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
housing_cat = housing["ocean_proximity"]
housing_cat_encoded = encoder.fit_transform(housing_cat)
housing_cat_encoded
```

We can look at the mapping that this encoder has learned using the class attribute:

```
In [53]: print(encoder.classes_)
['<1H OCEAN' 'INLAND' 'ISLAND' 'NEAR BAY' 'NEAR OCEAN']
```

这样的处理有一个问题：对上面的例子而言，0和4的距离大于0和1的距离，这个在实际中并没有体现。所以常见的处理方式是，把非数值属性的数值表示由单独向量转化为一个多维数组，用不同位置的0和1来表示不同的属性。Scikit-Learn恰好提供了这样的方法。也就是OneHotEncoder.

```
from sklearn.preprocessing import OneHotEncoder
encoder= OneHotEncoder()
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
housing_cat_1hot.toarray()
```

```
housing_cat_1hot.toarray()
```

```
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.]])
```

We can apply both transformations(text to integer,integer to one_hot vectors) using the LabelBinarizer class

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

LabelBinarier method are recommended.

6.3 Feature Scaling

Machine Learning algorithms do not perform well when the input numerical attributes have very different scales.The total number of rooms ranges from 6 to 39320 while the median income ranges from 0 to 15.

There are two common ways to get all attribute to have the same scale:min-max scaling and standardization. Min-max scaling(很多人叫normaliza)：把数据根据最大最小值限定在0到1的范围内。Scikit—learn中的MinMaxScalar就是用来实现这个功能的。

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4}$$

Standardization：First,it substracts the mean value and divide by unit variance.Standardization does not bound values to a specific range,which may be problems in some algorithms(neural networks).However standardization is much less affected by outliers.简单说就是，当一组数据中大部分数在1-100之间，当有一个离群点1000时，大量的数据都在一个很小的范围内。Scikit-learn provides a transform called StandardScaler for standardization.

$$x_{new} = \frac{x - \mu}{\sigma} \tag{5}$$

6.4 Custom Transformers

通常我们把添加属性的过程写成一个类，而不是大段大段的写成凑得。里面设计好具体的方法，于是就有了下面的代码

```
from sklearn.base import BaseEstimator,TransformerMixin

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[:,rooms_ix]/X[:,household_ix]
        population_per_household = X[:,population_ix]/X[:,household_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:,bedrooms_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 = adder.transform(housing.values)
```

6.5 Transform pipelines

Scikit-learn provides the *Pipeline* class to help with such sequences of transformations.Here is a small pipeline for numeerical attribute:

```
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)
```

The Pipeline constructor takes a list of name/estimator pairs defining a sequence of steps.顾名思义就是按顺序transform输入，一个的输出作为另一个的输入，按顺序执行。We now have a pipeline for numercial values,we also need to apply the LabelBinarizer on the categorical values.在这里要使用的是FeatureUnion().这里有一点改动，就是用CategoricalEncoder代替LabelBinarier.

```
# Definition of the CategoricalEncoder class, copied from PR #9151.
# Just run this cell, or copy it to your code, do not try to understand it (yet).

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.utils import check_array
from sklearn.preprocessing import LabelEncoder
from scipy import sparse

class CategoricalEncoder(BaseEstimator, TransformerMixin):
    The input to this transformer should be a matrix of integers or strings,
    denoting the values taken on by categorical (discrete) features.
    The features can be encoded using a one-hot aka one-of-K scheme
    (``encoding='onehot'``, the default) or converted to ordinal integers
    (``encoding='ordinal'``).
    This encoding is needed for feeding categorical data to many scikit-learn
    estimators, notably linear models and SVMs with the standard kernels.
    Read more in the :ref:`User Guide <preprocessing_categorical_features>`.
    Parameters
    ----------
    encoding : str, 'onehot', 'onehot-dense' or 'ordinal'
        The type of encoding to use (default is 'onehot'):
        - 'onehot': encode the features using a one-hot aka one-of-K scheme
          (or also called 'dummy' encoding). This creates a binary column for
          each category and returns a sparse matrix.
        - 'onehot-dense': the same as 'onehot' but returns a dense array
          instead of a sparse matrix.
        - 'ordinal': encode the features as ordinal integers. This results in
          a single column of integers (0 to n_categories - 1) per feature.
    categories : 'auto' or a list of lists/arrays of values.
```

```
        Categories (unique values) per feature:
        - 'auto' : Determine categories automatically from the training data.
        - list : ``categories[i]`` holds the categories expected in the ith
          column. The passed categories are sorted before encoding the data
          (used categories can be found in the ``categories_`` attribute).
    dtype : number type, default np.float64
        Desired dtype of output.
    handle_unknown : 'error' (default) or 'ignore'
        Whether to raise an error or ignore if a unknown categorical feature is
        present during transform (default is to raise). When this is parameter
        is set to 'ignore' and an unknown category is encountered during
        transform, the resulting one-hot encoded columns for this feature
        will be all zeros.
        Ignoring unknown categories is not supported for
        ``encoding='ordinal'``.
    Attributes
    ----------
    categories_ : list of arrays
        The categories of each feature determined during fitting. When
        categories were specified manually, this holds the sorted categories
        (in order corresponding with output of `transform`).
    Examples
    --------
    Given a dataset with three features and two samples, we let the encoder
    find the maximum value per feature and transform the data to a binary
    one-hot encoding.
    >>> from sklearn.preprocessing import CategoricalEncoder
    >>> enc = CategoricalEncoder(handle_unknown='ignore')
    >>> enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])
    ... # doctest: +ELLIPSIS
    CategoricalEncoder(categories='auto', dtype=<... 'numpy.float64'>,
              encoding='onehot', handle_unknown='ignore')
    >>> enc.transform([[0, 1, 1], [1, 0, 4]]).toarray()
    array([[ 1.,  0.,  0.,  1.,  0.,  0.,  1.,  0.,  0.],
           [ 0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.]])
    See also
    --------
    sklearn.preprocessing.OneHotEncoder : performs a one-hot encoding of
      integer ordinal features. The ``OneHotEncoder assumes`` that input
      features take on values in the range ``[0, max(feature)]`` instead of
      using the unique values.
    sklearn.feature_extraction.DictVectorizer : performs a one-hot encoding of
      dictionary items (also handles string-valued features).
    sklearn.feature_extraction.FeatureHasher : performs an approximate one-hot
      encoding of dictionary items or strings.
    """

```

```
    def __init__(self, encoding='onehot', categories='auto', dtype=np.float64,
78                 handle_unknown='error'):
        self.encoding = encoding
80      self.categories = categories
        self.dtype = dtype
82      self.handle_unknown = handle_unknown

84  def fit(self, X, y=None):
        """Fit the CategoricalEncoder to X.
86      Parameters
        ----------
88      X : array-like, shape [n_samples, n_feature]
            The data to determine the categories of each feature.
90      Returns
        -------
92      self
        """
94
        if self.encoding not in ['onehot', 'onehot-dense', 'ordinal']:
96          template = ("encoding should be either 'onehot', 'onehot-dense' "
                        "or 'ordinal', got %s")
98          raise ValueError(template % self.handle_unknown)

100     if self.handle_unknown not in ['error', 'ignore']:
            template = ("handle_unknown should be either 'error' or "
102                     "'ignore', got %s")
            raise ValueError(template % self.handle_unknown)
104
        if self.encoding == 'ordinal' and self.handle_unknown == 'ignore':
106         raise ValueError("handle_unknown='ignore' is not supported for"
                             " encoding='ordinal'")
108
        X = check_array(X, dtype=np.object, accept_sparse='csc', copy=True)
110     n_samples, n_features = X.shape

112     self._label_encoders_ = [LabelEncoder() for _ in range(n_features)]

114     for i in range(n_features):
            le = self._label_encoders_[i]
116         Xi = X[:, i]
            if self.categories == 'auto':
118             le.fit(Xi)
            else:
120             valid_mask = np.in1d(Xi, self.categories[i])
                if not np.all(valid_mask):
122                 if self.handle_unknown == 'error':
                        diff = np.unique(Xi[~valid_mask])
```

```
                        msg = ("Found unknown categories {0} in column {1}"
                               " during fit".format(diff, i))
                        raise ValueError(msg)
                le.classes_ = np.array(np.sort(self.categories[i]))

        self.categories_ = [le.classes_ for le in self._label_encoders_]

        return self

    def transform(self, X):
        """Transform X using one-hot encoding.
        Parameters
        ----------
        X : array-like, shape [n_samples, n_features]
            The data to encode.
        Returns
        -------
        X_out : sparse matrix or a 2-d array
            Transformed input.
        """
        X = check_array(X, accept_sparse='csc', dtype=np.object, copy=True)
        n_samples, n_features = X.shape
        X_int = np.zeros_like(X, dtype=np.int)
        X_mask = np.ones_like(X, dtype=np.bool)

        for i in range(n_features):
            valid_mask = np.in1d(X[:, i], self.categories_[i])

            if not np.all(valid_mask):
                if self.handle_unknown == 'error':
                    diff = np.unique(X[~valid_mask, i])
                    msg = ("Found unknown categories {0} in column {1}"
                           " during transform".format(diff, i))
                    raise ValueError(msg)
                else:
                    # Set the problematic rows to an acceptable value and
                    # continue `The rows are marked `X_mask` and will be
                    # removed later.
                    X_mask[:, i] = valid_mask
                    X[:, i][~valid_mask] = self.categories_[i][0]
            X_int[:, i] = self._label_encoders_[i].transform(X[:, i])

        if self.encoding == 'ordinal':
            return X_int.astype(self.dtype, copy=False)

        mask = X_mask.ravel()
        n_values = [cats.shape[0] for cats in self.categories_]
```

```
        n_values = np.array([0] + n_values)
        indices = np.cumsum(n_values)

        column_indices = (X_int + indices[:-1]).ravel()[mask]
        row_indices = np.repeat(np.arange(n_samples, dtype=np.int32),
                                n_features)[mask]
        data = np.ones(n_samples * n_features)[mask]

        out = sparse.csc_matrix((data, (row_indices, column_indices)),
                                shape=(n_samples, indices[-1]),
                                dtype=self.dtype).tocsr()
        if self.encoding == 'onehot-dense':
            return out.toarray()
        else:
            return out
```

```
from sklearn.pipeline import FeatureUnion

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

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

num_pipeline = Pipeline([('selector', DataFrameSelector(num_attribs)), ('imputer', Imputer(
    strategy="median")), ('attr_adder', CombinedAttributesAdder()), ('std_scaler', StandardScaler
    ()),])
cat_pipeline = Pipeline([('selector', DataFrameSelector(cat_attribs)), ('label_binarizer',
    CategoricalEncoder(encoding="onehot-dense")),])
full_pipeline = FeatureUnion(transformer_list=[("num_pipeline", num_pipeline), ("cat_pipeline",
    cat_pipeline),])
```

最后这部分代码比较多，其实主要是书上的代码稍稍有点问题，所以就稍稍做了改动，具体可以查看code部分的实现，但是这都不是最重要的，最重要的其实是对pipeline的掌握，其实这里无非就是把之前的step by step的内容改成了pipeline的形式。接下来，我们就要开始最核心部分的学习了，我们花了很

长的时间去做准备工作，真正的模型构建及数据处理，由于sklearn封装的原因，显得比较单薄。

7 Select and Train a Model

7.1 Training and Evaluating on the Training Set

Let's first train a Linear Regression Model.

```
from sklearn.linear_model import LinearRegression

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

It's simple!Let's try it out on a few instance from the training set:

```
some_data = housing.iloc[:5]
some_labels = housing_labels[:5]
some_data_prepared = full_pipeline.transform(some_data)
print(lin_reg.predict(some_data_prepared))
print(list(some_labels))
```

It works!Although the prediction are not exactly accurate.Let's evaluate the RMSE(标准差) on the whole training set using Scikit-Learn's mean_squared_error function.

```
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
```

The result is 67842.Well,this is better than nothing but clearly not a great score.As we said before,underfitting is the reason why leading to the low accuracy.We can selecting a more powerful model,feeding

better feature or reducing the constraints on the model.Let's first try a more complex model to see how it does.

Let's train a DecisionTreeRegressor.This is a more powerful model,capable of finding complex nonlinear relationship in the data.Then,let's evaluate it on the training set:

```
from sklearn.tree import DecisionTreeRegressor

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

```
In [62]: from sklearn.tree import DecisionTreeRegressor

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

Out[62]: DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                    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,
                    presort=False, random_state=None, splitter='best')

In [63]: housing_predictions = tree_reg.predict(housing_prepared)
         tree_mse  =mean_squared_error(housing_predictions,housing_labels)
         tree_rmse = np.sqrt(tree_mse)
         tree_rmse

Out[63]: 0.0
```

图 15: bizarre result

Of course it is much more likely that the model has badly overfitting problem.

7.2 Better Evaluation Using Cross-Validation

One way to evaluate the decision Tree would be to use the train_test_split function to split the training set into a smaller training set and a vaildation set,then train your models against the smaller training set and evaluate them against the vaildation set.

A great alternative is to use Scikit-Learn's cross-vaildation feature.The following code performs *K-fold* cross-vaildation:简而言之，就是训练集分成10份，分别用一份来验证，其余9份来训练，一共做十次。输

出结果是一个array，把十次的结果输出。

```
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)
rmse_scores = np.sqrt(-scores)
```

在这里使用的是均方根误差，也就是把预测值和实际值之间求一个MSE。

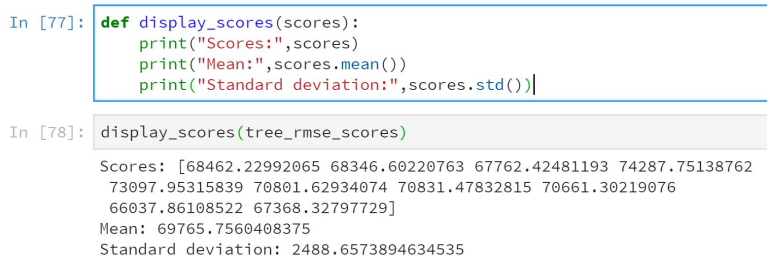


图 16: Cross-Vaildation of DecisionTreeRegressor

The decision tree doesn't look good as it did earlier.In fact,it seems to perform worse than the Linear Regression Model.Let's compute the same scores for the Linear Regression model:

```
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)
```

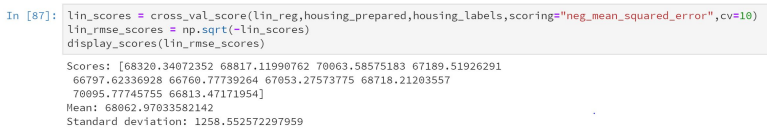


图 17: Cross-Vaildation of Linear Regression

可以看出，决策树模型的精度比起线性回归更差。

Let's try one last model:RandomForestRegressor.更多内容，后面章节会细讲。

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

```
In [88]: from sklearn.ensemble import RandomForestRegressor
         forest_reg = RandomForestRegressor()
         forest_reg.fit(housing_prepared,housing_labels)
         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: [52332.70442842 52559.42946366 53856.97030171 52908.47754638
          52240.36928912 52500.91341798 52909.37390224 53228.02241597
          52207.98172025 52810.72518843]
         Mean: 52755.49676741497
         Standard deviation: 483.0935087775267
```

图 18: Cross-Vaildation of RandomForestRegressor

通过结果可以看出，预测的结果和实际还是有着较大的差距。因此，这里还是Overfitting了。我们要不就是简化模型，在要不就是获取更多数据。在这里仅仅测试这几个方法，当然可以尝试的方法还有很多。

8 Fine-Tune Your Model

Let's assume we have a list of promising models.We need to fine-tune them.

8.1 Grid Search

One way to do that is to fiddle with the hyperparameters,until you find the a great combinationof hyperparameter values.We should get Scikit-Learn's GridSearchCV to search for us.

```
from sklearn.model_selection import GridSearchCV

```

```
para_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,para_grid,cv=5,scoring='neg_mean_squared_error')
grid_search.fit(housing_prepared,housing_labels)
```

在这里简单说明一下，更详细的将会在后面说到。这里做的就是数据的组合，穷举所有可能的组合。在上面的例子中，首先再第一个字典中衡量$3*4=12$种组合。接下来在第二个字典中穷举$2*3=6$种组合。只是在这里的bootstrap参数被设为False。接下来这18种不同的组合中的每一种都会训练五次(5 times fold cross vaildation).共计90轮训练。

```
In [92]: grid_search.best_params_

Out[92]: {'max_features': 6, 'n_estimators': 30}

In [93]: grid_search.best_estimator_

Out[93]: 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)

In [94]: cvres = grid_search.cv_results_

In [95]: for mean_score,params in zip(cvres["mean_test_score"],cvres["params"]):
             print(np.sqrt(-mean_score),params)

         64225.05953347903 {'max_features': 2, 'n_estimators': 3}
         55769.669259561255 {'max_features': 2, 'n_estimators': 10}
         52744.90075339514 {'max_features': 2, 'n_estimators': 30}
         61166.08445796616 {'max_features': 4, 'n_estimators': 3}
         51929.83373710057 {'max_features': 4, 'n_estimators': 10}
         50100.85661187252 {'max_features': 4, 'n_estimators': 30}
         58054.14805566756 {'max_features': 6, 'n_estimators': 3}
         52240.13816610162 {'max_features': 6, 'n_estimators': 10}
         49463.49846846275 {'max_features': 6, 'n_estimators': 30}
         57621.90917522437 {'max_features': 8, 'n_estimators': 3}
         51154.96854651268 {'max_features': 8, 'n_estimators': 10}
         49552.83238019895 {'max_features': 8, 'n_estimators': 30}
         63698.705720632206 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
         54287.06306815 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
         59687.38245259647 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
         52484.055717093186 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
         57536.073330506515 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
         51595.69559914546 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

图 19: Grid Search

可以看出，最好的情况，RMSE为49959，比之前的结果略好一点(52634)。

8.2 Randomized Search

刚才的方法好归好，但是可以想见的是，当attributes过多的时候，这个循环的次数就会非常大，复杂度极高。这时候就会用到RandomizedSearchCV.用法类似，不再赘述。

8.3 Ensemble Methods

另一个提高精度的方法就是使用不同模型的组合。这个将在第七章讲。

8.4 Analyze the Best Models and Their Errors

通过上面的分析之后，我们看看这些重要的特性是什么

```
feature_importance = grid_search.best_estimator_.feature_importances_
extra_attribs = ["rooms_per_hhold","pop_per_hhold","bedrooms_per_room"]
cat_one_hot_attribs = list(encoder.classes_)
attributes = num_attribs+extra_attribs+cat_one_hot_attribs
sorted(zip(feature_importance,attributes),reverse=True)
```

```
In [96]: feature_importance = grid_search.best_estimator_.feature_importances_

In [97]: feature_importance

Out[97]: array([8.29392123e-02, 7.62847995e-02, 4.18505306e-02, 1.82010080e-02,
                1.60829081e-02, 1.82111914e-02, 1.57231445e-02, 3.56809498e-01,
                7.46777650e-02, 1.07102773e-01, 1.47192009e-02, 1.18921229e-02,
                1.56188232e-01, 1.93309029e-04, 2.09639038e-03, 7.02791422e-03])

In [99]: extra_attribs = ["rooms_per_hhold","pop_per_hhold","bedrooms_per_room"]
         cat_one_hot_attribs = list(encoder.classes_)
         attributes = num_attribs+extra_attribs+cat_one_hot_attribs
         sorted(zip(feature_importance,attributes),reverse=True)

Out[99]: [(0.35680949846837084, 'median_income'),
          (0.15618823175699562, 'INLAND'),
          (0.10710277305590452, 'pop_per_hhold'),
          (0.08293921225671655, 'longitude'),
          (0.07628479954544372, 'latitude'),
          (0.07467776501165445, 'rooms_per_hhold'),
          (0.04185053058883791, 'housing_median_age'),
          (0.018211191370133405, 'population'),
          (0.018201007970220267, 'total_rooms'),
          (0.01608290813105871, 'total_bedrooms'),
          (0.015723144450168883, 'households'),
          (0.014719200913763207, 'bedrooms_per_room'),
          (0.01189212285340688, '<1H OCEAN'),
          (0.007027914216497706, 'NEAR OCEAN'),
          (0.002096390381606388, 'NEAR BAY'),
          (0.00019330902922086812, 'ISLAND')]
```

图 20: Feature importance

8.5 Evaluate Your System on the Test Set

```
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_prediction = final_model.predict(X_test_prepared)

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

最后的结果是50014,评估结果通常要比交叉验证的效果差一点，如果你之前做过很多超参数微调（因

为你的系统在验证集上微调，得到了不错的性能，通常不会在未知的数据集上有同样好的效果）。这个例子不属于这种情况，但是当发生这种情况时，你一定要忍住不要调节超参数，使测试集的效果变好；这样的提升不能推广到新数据上。

9 Launch,Monitor and Maintain Your System

很好，你被允许启动系统了！你需要为实际生产做好准备，特别是接入输入数据源，并编写测试。

你还需要编写监控代码，以固定间隔检测系统的实时表现，当发生下降时触发报警。这对于捕获突然的系统崩溃和性能下降十分重要。做监控很常见，是因为模型会随着数据的演化而性能下降，除非模型用新数据定期训练。

评估系统的表现需要对预测值采样并进行评估。这通常需要人来分析。分析者可能是领域专家，或者是众包平台（比如 Amazon Mechanical Turk 或 CrowdFlower）的工人。不管采用哪种方法，你都需要将人工评估的流水线植入系统。

你还要评估系统输入数据的质量。有时因为低质量的信号（比如失灵的传感器发送随机值，或另一个团队的输出停滞），系统的表现会逐渐变差，但可能需要一段时间，系统的表现才能下降到一定程度，触发警报。如果监测了系统的输入，你就可能尽量早的发现问题。对于线上学习系统，监测输入数据是非常重要的。

最后，你可能想定期用新数据训练模型。你应该尽可能自动化这个过程。如果不这么做，非常有可能你需要每隔至少六个月更新模型，系统的表现就会产生严重波动。如果你的系统是一个线上学习系统，你需要定期保存系统状态快照，好能方便地回滚到之前的工作状态。

实践!

10 Try it Out

希望这一章能告诉你机器学习项目是什么样的，你能用学到的工具训练一个好系统。你已经看到，大部分的工作是数据准备步骤、搭建监测工具、建立人为评估的流水线和自动化定期模型训练，当然，最好能了解整个过程、熟悉三或四种算法，而不是在探索高级算法上浪费全部时间，导致在全局上的时间不够。

有机会去kaggle做做东西。