机器学习入门

既然咱们学 Python，也肯定要讲讲机器学习。因为它的很多库都是用 Python 写的。开始先装上它们玩一下。

Tensorflow

安装一下。

```
$ pip install tensorflow
ERROR: Could not find a version that satisfies the requirement tensorflow
ERROR: No matching distribution found for tensorflow
```

```
$ type python
python is aliased to `/usr/local/Cellar/python@3.9/3.9.1_6/bin/python3'
```

然而 Tensorflow 2 只支持 Python 3.5–3.8。我们用的是 3.9 。

```
 % type python3
python3 is /usr/bin/python3
% python3 -V
Python 3.8.2
```

注意到我系统里的 python3 是 3.8.2 版本。这个 Python 版本对应的 pip 安装到哪儿呢。

```
% python3 -m pip -V
pip 21.0.1 from /Users/lzw/Library/Python/3.8/lib/python/site-packages/pip (python 3.8)
```

对应的 pip 在这里。那我更改一下.zprofile 文件里。最近我更改了我的 shell。.zprofile 就相当于之前的.bash_profile。加入一行。

```
alias pip3=/Users/lzw/Library/Python/3.8/bin/pip3
```

这样，我们用 python3 和 pip3 来玩 Tensorflow。

```
% pip3 install tensorflow
...
Successfully installed absl-py-0.12.0 astunparse-1.6.3 cachetools-4.2.1 certifi-2020.12.5 chardet-4.0.0
```

安装了很多库。用上官网的一个例子。

```
import tensorflow as tf


mnist = tf.keras.datasets.mnist
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])

predictions = model(x_train[:1]).numpy()
print(predictions)
```

运行一下。

```
$ /usr/bin/python3 tf.py
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 10s 1us/step
[[ 0.15477428 -0.3877643   0.0994779   0.07474922 -0.26219758 -0.03550266
   0.32226565 -0.37141111  0.10925996 -0.0115255 ]]
```

可见下载了数据集，接着输出了结果。

接下来，看看图片分类的例子。

```
# TensorFlow and tf.keras
import tensorflow as tf

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

报错。

```
ModuleNotFoundError: No module named 'matplotlib'
```

安装一下。

```
% pip3 install matplotlib
```

正确了。

```
$ /usr/bin/python3 image.py
2.4.1
```

进行复制粘贴例子代码。

```
# TensorFlow and tf.keras
import tensorflow as tf

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
print(train_images.shape)
print(len(train_labels))
```

输出了结果。注意到这里有 train_images、train_labels、test_images、test_labels。就是分为训练数据集和测试数据集。

```
(60000, 28, 28)
60000
```

接着试试打印出图片来。

```
print(train_images[0])
```

看下结果。

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  13  73   0
    0   1   4   0   0   0   0   1   1   0]
```

```
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62
   54   0   0   0   1   3   4   0   0   3]
 [  0   0   0   0   0   0   0   0   0   0   0   0   6   0 102 204 176 134
  144 123  23   0   0   0   0  12  10   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 155 236 207 178
  107 156 161 109  64  23  77 130  72  15]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0  69 207 223 218 216
  216 163 127 121 122 146 141  88 172  66]]
  ....
```

这里节选了部分结果。

```
print(len(train_images[0][0]))
```

输出 28。所以很清楚，这是一个横宽为 28 的矩阵。继续打印。

```
    print(len(train_images[0][0][0])
TypeError: object of type 'numpy.uint8' has no len()
```

所以很明白。每张图片都是 28*28*3 的数组。最后一维数组保存的是 rgb 值。然而发现我们的想法可能是错的。

```
print(train_images[0][1][20])
```

```
0
```

```
print(train_images[0][1])
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

说明每张图片是 28*28 的数组。捣鼓了一阵。我们终于知道了秘密。

先来看看输出的图。

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```

看到右边的颜色条吗。0 到 250。原来这是在两种颜色里的渐变。可是它怎么知道是哪两种颜色。我们哪里告诉它了。

接着我们把第二张图也打印出来。

```
plt.imshow(train_images[1])
```

很有意思。这难道是 pyplot 依赖库默认的吗。继续运行官网给的代码。

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

注意到这里显示了图片以及它们的分类。终于我们知道了 cmp 参数。如果 cmp 什么都不写，一定会是刚刚我们那种色彩的。果然。

```
    plt.imshow(train_images[i])
```

这会我们搜索 pyplot cmap。找到一些资料。

```
    plt.imshow(train_images[i], cmap=plt.cm.PiYG)
```

改一下代码。

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(2,5,i+1)    ## 改这行
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.Blues)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

然而报错了。

```
ValueError: num must be 1 <= num <= 10, not 11
```

这意味着什么。之前的 5,5,i+1 到底什么意思。为什么改成 2 就不行了。尽管我们直观地知道大概是 5 行 5 列的意思。但为什么会报这个错误。11 是怎么计算出来的。num 又是什么意思。10 是什么意思。注意到 2*5=10。所以也许当 i=11 的时候出错了。当改成 for i in range(10): 时，得到了以下结果。

这会稍微看一下文档，得知 subplot(nrows, ncols, index, **kwargs)。嗯，到此我们很明白了。

plot_scale

Figure 1: plot_scale

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    # plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.Blues)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

注意到 0 25 这种就叫 xticks。当我们放大缩小这个框的时候，会有不同的展示。

注意到放大缩小框，xticks 和 xlabels 会有不同的显示。

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10)

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('\nTest accuracy:', test_acc)
```

注意到了这里定义 model 的方式，用到了类 Sequential。注意这些参数，28,28、128、relu、10。注意到需要 compile 和 fit。fit 是拟合的意思。注意到 28,28 就是图形大小。

```
Epoch 1/10
1875/1875 [==============================] - 2s 928us/step - loss: 0.6331 - accuracy: 0.7769
Epoch 2/10
1875/1875 [==============================] - 2s 961us/step - loss: 0.3860 - accuracy: 0.8615
```

```
Epoch 3/10
1875/1875 [==============================] - 2s 930us/step - loss: 0.3395 - accuracy: 0.8755
Epoch 4/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3071 - accuracy: 0.8890
Epoch 5/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2964 - accuracy: 0.8927
Epoch 6/10
1875/1875 [==============================] - 2s 985us/step - loss: 0.2764 - accuracy: 0.8955
Epoch 7/10
1875/1875 [==============================] - 2s 961us/step - loss: 0.2653 - accuracy: 0.8996
Epoch 8/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2549 - accuracy: 0.9052
Epoch 9/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2416 - accuracy: 0.9090
Epoch 10/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2372 - accuracy: 0.9086
313/313 - 0s - loss: 0.3422 - accuracy: 0.8798

Test accuracy: 0.879800021648407
```

模型已经训练出来了。来改下参数。

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(28, activation='relu'),    # 128 -> 28
    tf.keras.layers.Dense(10)
])
```

修改一下 `Dense` 的第一个参数。

```
Epoch 1/10
1875/1875 [==============================] - 2s 714us/step - loss: 6.9774 - accuracy: 0.3294
Epoch 2/10
1875/1875 [==============================] - 1s 715us/step - loss: 1.3038 - accuracy: 0.4831
Epoch 3/10
1875/1875 [==============================] - 1s 747us/step - loss: 1.0160 - accuracy: 0.6197
Epoch 4/10
1875/1875 [==============================] - 1s 800us/step - loss: 0.7963 - accuracy: 0.6939
Epoch 5/10
```

```
1875/1875 [==============================] - 2s 893us/step - loss: 0.7006 - accuracy: 0.7183
Epoch 6/10
1875/1875 [==============================] - 1s 747us/step - loss: 0.6675 - accuracy: 0.7299
Epoch 7/10
1875/1875 [==============================] - 1s 694us/step - loss: 0.6681 - accuracy: 0.7330
Epoch 8/10
1875/1875 [==============================] - 1s 702us/step - loss: 0.6675 - accuracy: 0.7356
Epoch 9/10
1875/1875 [==============================] - 1s 778us/step - loss: 0.6508 - accuracy: 0.7363
Epoch 10/10
1875/1875 [==============================] - 1s 732us/step - loss: 0.6532 - accuracy: 0.7350
313/313 - 0s - loss: 0.6816 - accuracy: 0.7230

Test accuracy: 0.7229999899864197
```

注意到 Test accuracy 前后发生了变化。Epoch 这样的是 fit 函数输出的日志。注意到当是 128 时，accuracy 从 0.7769 变到 0.9086。而当是 28 时，accuracy 从 0.3294 变到 0.7350。这会注意到，我们先是用训练集，来调优 loss 和 accuracy。接着用测试数据集来测试。先来看看 train_labels。

```
print(train_labels)
[9 0 0 ... 3 0 5]
print(len(train_labels))
60000
```

这意味着用 0 到 9 来表示这些类别。刚好 class_names 也是有 10 个。

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

再来改一改。

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(28, activation='relu'),
    tf.keras.layers.Dense(5)   # 10 -> 5
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.fit(train_images, train_labels, epochs=10)
```

出错了。

```
tensorflow.python.framework.errors_impl.InvalidArgumentError:  Received a label value of 9 which is outs
         [[node sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEn

Function call stack:
train_function
```

改成把 Sequential 的第三个参数 Dense 的参数改成 15 就可以了。结果区别不大。试试改改 Epoch。

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(28, activation='relu'),
    tf.keras.layers.Dense(15)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=15)  # 10 -> 15

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('\nTest accuracy:', test_acc)
```

```
Epoch 1/15
1875/1875 [==============================] - 2s 892us/step - loss: 6.5778 - accuracy: 0.3771
Epoch 2/15
1875/1875 [==============================] - 2s 872us/step - loss: 1.3121 - accuracy: 0.4910
Epoch 3/15
1875/1875 [==============================] - 2s 909us/step - loss: 1.0900 - accuracy: 0.5389
Epoch 4/15
1875/1875 [==============================] - 1s 730us/step - loss: 1.0422 - accuracy: 0.5577
Epoch 5/15
```

```
1875/1875 [==============================] - 1s 709us/step - loss: 0.9529 - accuracy: 0.5952
Epoch 6/15
1875/1875 [==============================] - 1s 714us/step - loss: 0.9888 - accuracy: 0.5950
Epoch 7/15
1875/1875 [==============================] - 1s 767us/step - loss: 0.8678 - accuracy: 0.6355
Epoch 8/15
1875/1875 [==============================] - 1s 715us/step - loss: 0.8247 - accuracy: 0.6611
Epoch 9/15
1875/1875 [==============================] - 1s 721us/step - loss: 0.8011 - accuracy: 0.6626
Epoch 10/15
1875/1875 [==============================] - 1s 711us/step - loss: 0.8024 - accuracy: 0.6622
Epoch 11/15
1875/1875 [==============================] - 1s 781us/step - loss: 0.7777 - accuracy: 0.6696
Epoch 12/15
1875/1875 [==============================] - 1s 724us/step - loss: 0.7764 - accuracy: 0.6728
Epoch 13/15
1875/1875 [==============================] - 1s 731us/step - loss: 0.7688 - accuracy: 0.6767
Epoch 14/15
1875/1875 [==============================] - 1s 715us/step - loss: 0.7592 - accuracy: 0.6793
Epoch 15/15
1875/1875 [==============================] - 1s 786us/step - loss: 0.7526 - accuracy: 0.6792
313/313 - 0s - loss: 0.8555 - accuracy: 0.6418

Test accuracy: 0.6417999863624573
```

注意改成 15。区别也不大。`tf.keras.layers.Dense(88, activation='relu')`，是重要的。试着 128 改成 88。得到了 `Test accuracy: 0.824999988079071`。128 时，是 `0.879800021648407`。28 时，是 `0.7229999899864197`。是不是越大越好，然而当改成 256 时，是 `Test accuracy: 0.8409000039100647`。这不禁让我们思考 loss 和 accuracy 的含义。

```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
```

接下来预测一下。注意到 Sequential 和上面的一样。注意到参数 model 和 `tf.keras.layers.Softmax()`。

```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
predictions = probability_model.predict(test_images)
```

```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)

def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')

i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
```

```
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```

这说明这个图片 99% 的可能是 Ankle boot。注意到 plot_image 是显示左边的图。plot_value_array 是输出右边的图。

```
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```

注意到这里只是显示更多的测试结果。所以使用流程我们大致很清楚。所以我们还不知道背后怎么计算的。但我们知道如何使用它们。它们背后是微积分。如何理解微积分呢。

比如说有个数字，1 到 100 让你猜。每次你猜多少。我告诉你小了还是大了。你猜 50。我说小了。你猜 80。我说大了。你猜 65。我说大了。你猜 55。我说小了。你猜 58。我说，嗯，猜对了。

机器学习，就是在背后模拟类似的过程。只不过复杂一些。可能是很多的 1 到 100，要猜很多数。同时每次猜都要进行很多运算。以及每次判断是否大了还是小了，要计算很多。

PyTorch

安装一下。这个支持 3.9 版本的 Python。

```
$ pip install torch torchvision
Collecting torch
  Downloading torch-1.8.0-cp39-none-macosx_10_9_x86_64.whl (120.6 MB)
     |                 | 120.6 MB 224 kB/s
Collecting torchvision
  Downloading torchvision-0.9.0-cp39-cp39-macosx_10_9_x86_64.whl (13.1 MB)
     |                 | 13.1 MB 549 kB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.9/site-packages (from torch) (1.20.1)
Collecting typing-extensions
  Downloading typing_extensions-3.7.4.3-py3-none-any.whl (22 kB)
```

```
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.9/site-packages (from torchvisio
Installing collected packages: typing-extensions, torch, torchvision
Successfully installed torch-1.8.0 torchvision-0.9.0 typing-extensions-3.7.4.3
```

检验一下。

```
import torch
x = torch.rand(5, 3)
print(x)
```

出错了。

```
Traceback (most recent call last):
  File "torch.py", line 1, in <module>
    import torch
  File "torch.py", line 2, in <module>
    x = torch.rand(5, 3)
AttributeError: partially initialized module 'torch' has no attribute 'rand' (most likely due to a circ
```

谷歌一下这个错误信息。原来是因为我们的文件也叫 `torch`。重名了。改一下然后就正确了。

```
tensor([[0.5520, 0.9446, 0.5543],
        [0.6192, 0.0908, 0.8726],
        [0.0223, 0.7685, 0.9814],
        [0.4019, 0.5406, 0.3861],
        [0.5485, 0.6040, 0.2387]])
```

找到一个例子。

```
# -*- coding: utf-8 -*-

import torch
import math
dtype = torch.float
device = torch.device("cpu")
# device = torch.device("cuda:0") # Uncomment this to run on GPU

# Create random input and output data
x = torch.linspace(-math.pi, math.pi, 2000, device=device, dtype=dtype)
y = torch.sin(x)
```

```
# Randomly initialize weights
a = torch.randn((), device=device, dtype=dtype)
b = torch.randn((), device=device, dtype=dtype)
c = torch.randn((), device=device, dtype=dtype)
d = torch.randn((), device=device, dtype=dtype)

learning_rate = 1e-6
for t in range(2000):
    # Forward pass: compute predicted y
    y_pred = a + b * x + c * x ** 2 + d * x ** 3

    # Compute and print loss
    loss = (y_pred - y).pow(2).sum().item()
    if t % 100 == 99:
        print(t, loss)

    # Backprop to compute gradients of a, b, c, d with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_a = grad_y_pred.sum()
    grad_b = (grad_y_pred * x).sum()
    grad_c = (grad_y_pred * x ** 2).sum()
    grad_d = (grad_y_pred * x ** 3).sum()

    # Update weights using gradient descent
    a -= learning_rate * grad_a
    b -= learning_rate * grad_b
    c -= learning_rate * grad_c
    d -= learning_rate * grad_d
print(f'Result: y = {a.item()} + {b.item()} x + {c.item()} x^2 + {d.item()} x^3')
```

运行一下。

```
99 1273.537353515625
199 849.24853515625
299 567.4786987304688
399 380.30291748046875
499 255.92752075195312
```

```
599 173.2559814453125
699 118.2861328125
799 81.72274780273438
899 57.39331817626953
999 41.198158264160156
1099 30.41307830810547
1199 23.227672576904297
1299 18.438262939453125
1399 15.244369506835938
1499 13.113286972045898
1599 11.690631866455078
1699 10.740333557128906
1799 10.105220794677734
1899 9.6804780960083
1999 9.39621353149414
Result: y = -0.011828352697193623 + 0.8360244631767273 x + 0.002040589228272438 x^2 + -0.09038365632295
```

看看只用 numpy 库的代码。

```python
# -*- coding: utf-8 -*-
import numpy as np
import math

# Create random input and output data
x = np.linspace(-math.pi, math.pi, 2000)
y = np.sin(x)

# Randomly initialize weights
a = np.random.randn()
b = np.random.randn()
c = np.random.randn()
d = np.random.randn()

learning_rate = 1e-6
for t in range(2000):
    # Forward pass: compute predicted y
    # y = a + b x + c x^2 + d x^3
```

```
    y_pred = a + b * x + c * x ** 2 + d * x ** 3

    # Compute and print loss
    loss = np.square(y_pred - y).sum()
    if t % 100 == 99:
        print(t, loss)

    # Backprop to compute gradients of a, b, c, d with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_a = grad_y_pred.sum()
    grad_b = (grad_y_pred * x).sum()
    grad_c = (grad_y_pred * x ** 2).sum()
    grad_d = (grad_y_pred * x ** 3).sum()

    # Update weights
    a -= learning_rate * grad_a
    b -= learning_rate * grad_b
    c -= learning_rate * grad_c
    d -= learning_rate * grad_d

print(f'Result: y = {a} + {b} x + {c} x^2 + {d} x^3')
```

注意到这是两种方式来计算。

这两个例子，是先生成了一组 x 和 y。接着假设是三次方程。再接着用些方法把系数迭代计算出来。这些算法是怎样的呢。注意到是循环了 2000 次。每次拟合地精确一些。这里先不细究。

最后

目前，我们不懂机器学习背后是怎么计算的。然而，暂时不重要。我们用上面类似的知识已经可以来干很多事情了。还可以用机器学习来处理文本、音频等的。等我们试探了几十个例子，再学原理也不迟。

练习

- 学生像上面那样探索一遍。