机器学习的目标是训练一个模型

$$\boldsymbol{y} = f(\boldsymbol{X})$$

其中$\boldsymbol{X}$为数据，$\boldsymbol{y}$为标签，f为模型
经过一组数据的训练，对该模型输入类似的数据可以得到合适的标签
输入为数据和输出为标签只是习惯上的说法，只要输入输出符合模型的要求即可
例如AI图像识别中输入为图片输出为标签，而在AI绘画中输入为标签输出为图片

机器学习可以分为以下几个步骤：
1.整理数据和标签
2.定义和初始化模型
3.训练模型
4.测试模型
如果模型表现不好，可以尝试修改以改善效果
模型训练好之后可以用来对同类的问题进行预测
模型中的参数可以保存下来以方便使用
这些内容在代码中基本上可以写成类似以下模板的形式

In [1]:

```
'''
# Data code
X_train, y_train, X_test, y_test = train_test_split(X, y)

# Model code
model = Model(para)

# Train code
model.fit(X_train, y_train)

# Test code
model.score(X_test, y_test)

# Modify
model.set_params(para)

# Predict
model.predict(...)
''';
```

使用不同的模型会略有区别
具体情况用几个简单的例子展示

下面展示用KMeans对点分簇

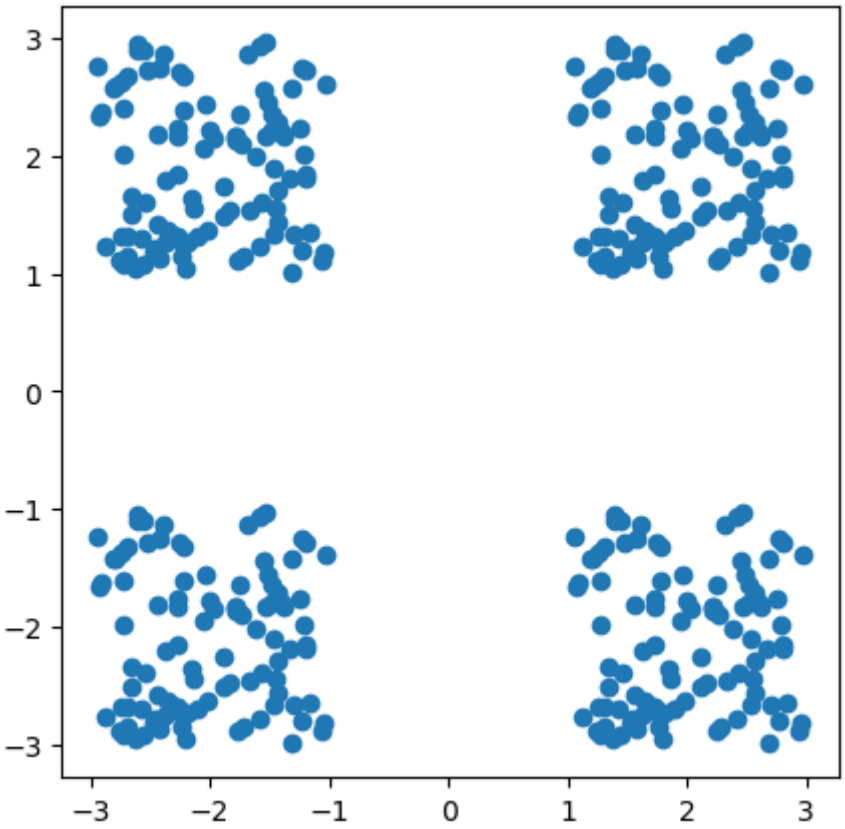
In [1]:

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
```

生成一些数据

In [2]:

```
points = np.random.uniform(-1, 1, (100, 2))
p1 = points + np.array((2, 2))
p2 = points + np.array((-2, 2))
p3 = points + np.array((-2, -2))
p4 = points + np.array((2, -2))
X = np.concatenate((p1, p2, p3, p4))

plt.figure(figsize=(5, 5))
plt.scatter(*X.T)
plt.show()
```

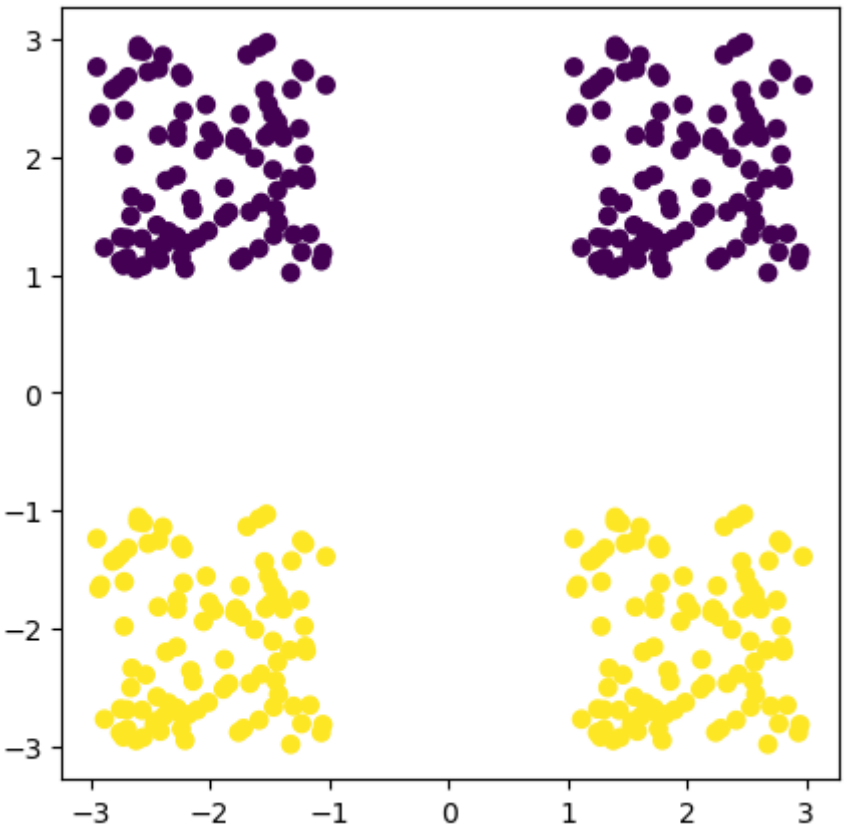


In [3]:

```
# model = Model(para)
kmeans = KMeans(2)

# model.fit(X_train, y_train)
y = kmeans.fit_predict(X)
# 由于是无监督学习，训练时没有标签
# 不仅需要训练模型，而且要得到标签
# 所以用fit_predict而不是fit
```

In [4]:

```
plt.figure(figsize=(5, 5))
plt.scatter(*X.T, c=y)
plt.show()
```

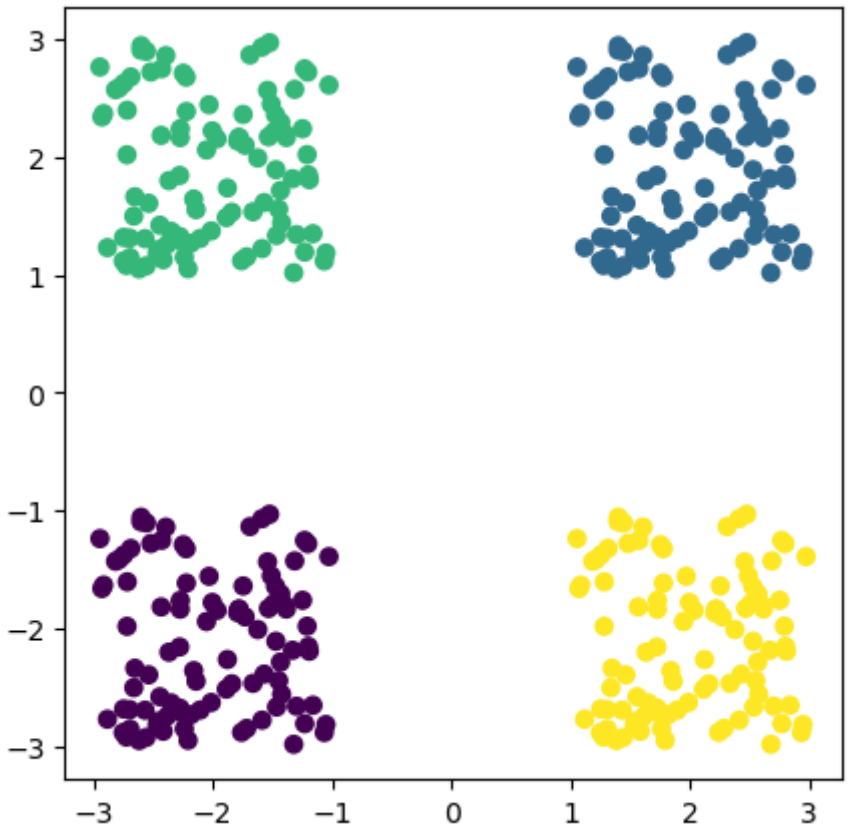


In [5]:

```
# model.set_params(para)
kmeans.set_params(n_clusters=4)
# 分成4簇看起来更合适

y = kmeans.fit_predict(X)
```

In [6]:

```
plt.figure(figsize=(5, 5))
plt.scatter(*X.T, c=y)
plt.show()
```



测试分簇结果

In [7]:

```
test_points = ((2.0, 2.0),
               (-2.0, 2.0),
               (-2.0, -2.0),
               (2.0, -2.0))

# model.predict(X_test)
kmeans.predict(test_points)
```

Out[7]:

```
array([1, 2, 0, 3], dtype=int32)
```

下面展示用pytorch实现用神经网络对点分类

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt

import torch
from torch import nn
from torch.utils.data import DataLoader
```

产生一些点，并根据xy的正负号确定标签

In [2]:

```
points = np.random.uniform(-1, 1, (10000, 2))
p1 = points + np.array((2, 2))
p2 = points + np.array((-2, 2))
p3 = points + np.array((-2, -2))
p4 = points + np.array((2, -2))
X = np.concatenate((p1, p2, p3, p4), dtype=np.float32)

Y = np.array((X > 0), dtype=np.float32)
# pytorch要求运算过程中的数据类型一致，因此在这里指定数据类型
```

上面产生的数据是numpy的数据结构
需要转换为pytorch的数据结构才能在pytorch中使用
可以使用以下方式

In [3]:

```
# 定义一个数据库对象
class CustomDataset:
    def __init__(self, data, labels):
        self.data = data
        self.labels = labels

    # 数据的数量
    def __len__(self):
        return len(self.labels)

    # 访问每一个数据的方式
    def __getitem__(self, idx):
        return self.data[idx], self.labels[idx]

# 读取前面产生的数据
train_data = CustomDataset(X, Y)

# 通过pytorch提供的DataLoader实现批量读取数据
# 训练时会用到
batch_size = 128
train_dataloader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
# shuffle=True 表示数据不是按顺序读取的，而是随机读取的
```

接下来是搭建神经网络模型

In [4]:

```
# 输入层连接到2个线性神经元，计算结果经过激活函数Sigmoid输出
net = nn.Sequential(nn.Linear(2, 2), nn.Sigmoid())
# Sequential是顺序连接模型，表示不同层之间顺序连接
# 如果要搭建循环神经网络等不同类型的神经网络
# 就要使用有对应功能的模型

# 初始化参数
for layer in net:
    if type(layer) == nn.Linear:
        layer.weight.data.normal_(0, 0.01)
        layer.bias.data.fill_(0)
```

数据和模型都准备好之后就可以训练了
下面使用pytorch教程中的方法进行训练

In [5]:

```
# 批量读取训练集进行训练
def train_loop(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    for batch, (X, y) in enumerate(dataloader):
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

训练用的train_loop函数需要4个参数
其中dataloader和model已经准备好了
还需要loss_fn和optimizer

In [6]:

```
# 分类问题使用交叉熵损失函数
loss = nn.CrossEntropyLoss()

# 使用随机梯度下降进行优化
trainer = torch.optim.SGD(net.parameters(), lr=0.1)
# lr为学习率
```

遍历训练集10次进行优化

In [7]:

```
epochs = 10
for t in range(epochs):
    train_loop(train_dataloader, net, loss, trainer)
```

```
loss: 0.647338  [    0/40000]
loss: 0.497046  [12800/40000]
loss: 0.525587  [25600/40000]
loss: 0.588541  [38400/40000]
loss: 0.516009  [    0/40000]
loss: 0.600300  [12800/40000]
loss: 0.497979  [25600/40000]
loss: 0.525918  [38400/40000]
loss: 0.551277  [    0/40000]
loss: 0.542556  [12800/40000]
loss: 0.561078  [25600/40000]
loss: 0.469761  [38400/40000]
loss: 0.504010  [    0/40000]
loss: 0.444945  [12800/40000]
loss: 0.550959  [25600/40000]
loss: 0.474935  [38400/40000]
loss: 0.482399  [    0/40000]
loss: 0.451033  [12800/40000]
loss: 0.483964  [25600/40000]
loss: 0.557776  [38400/40000]
loss: 0.575127  [    0/40000]
loss: 0.519535  [12800/40000]
loss: 0.559151  [25600/40000]
loss: 0.584140  [38400/40000]
loss: 0.588718  [    0/40000]
loss: 0.531470  [12800/40000]
loss: 0.590531  [25600/40000]
loss: 0.489080  [38400/40000]
loss: 0.484553  [    0/40000]
loss: 0.513621  [12800/40000]
loss: 0.485553  [25600/40000]
loss: 0.494500  [38400/40000]
loss: 0.511040  [    0/40000]
loss: 0.581465  [12800/40000]
loss: 0.426382  [25600/40000]
loss: 0.495086  [38400/40000]
loss: 0.474214  [    0/40000]
loss: 0.435742  [12800/40000]
loss: 0.544968  [25600/40000]
loss: 0.509170  [38400/40000]
```

然后简单地检查分类是否正确

In [8]:

```
test_points = ((2.0, 2.0),
               (-2.0, 2.0),
               (-2.0, -2.0),
               (2.0, -2.0))

print(net(torch.tensor(test_points)))
```

```
tensor([[0.9331, 0.9332],
        [0.0076, 0.9956],
        [0.1089, 0.1094],
        [0.9956, 0.0076]], grad_fn=<SigmoidBackward0>)
```

上面的内容只是展示代码的写法
神经网络和超参数都是随意设置的
有时候结果不对

下面展示用tensorflow实现用神经网络对点分类

In [1]:

```
# tensorflow会输出很多日志信息
# 用以下设置可以不输出日志
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
```

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.layers import Input, Dense, Dropout
from keras.models import Model, Sequential, load_model
```

In [3]:

```
# 产生数据
points = np.random.uniform(-1, 1, (10000, 2))
p1 = points + np.array((2, 2))
p2 = points + np.array((-2, 2))
p3 = points + np.array((-2, -2))
p4 = points + np.array((2, -2))
X = np.concatenate((p1, p2, p3, p4))

Y = np.array(X > 0, dtype=float)
```

In [4]:

```
# 搭建神经网络模型
inputs = Input(shape=(2,))
outputs = Dense(2, activation='sigmoid')(inputs)
model = Model(inputs, outputs)
```

In [5]:

```
# 设置优化算法、损失函数
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
# 损失函数有时不够直观
# 可以设置更合适的量来评价训练过程
```

In [6]:

```
# 按照给定的batch_size和epochs训练模型
model.fit(X, Y, batch_size=128, epochs=10)
```

```
Epoch 1/10
313/313 [==============================] - 1s 679us/step - loss: 0.5880 - accuracy:
0.7324
Epoch 2/10
313/313 [==============================] - 0s 662us/step - loss: 0.3981 - accuracy:
0.7487
Epoch 3/10
313/313 [==============================] - 0s 654us/step - loss: 0.3968 - accuracy:
0.7487
Epoch 4/10
313/313 [==============================] - 0s 653us/step - loss: 0.3966 - accuracy:
0.7478
Epoch 5/10
313/313 [==============================] - 0s 625us/step - loss: 0.3966 - accuracy:
0.7483
Epoch 6/10
313/313 [==============================] - 0s 654us/step - loss: 0.3966 - accuracy:
0.7479
Epoch 7/10
313/313 [==============================] - 0s 667us/step - loss: 0.3966 - accuracy:
0.7476
Epoch 8/10
313/313 [==============================] - 0s 841us/step - loss: 0.3966 - accuracy:
0.7490
Epoch 9/10
313/313 [==============================] - 0s 674us/step - loss: 0.3966 - accuracy:
0.7483
Epoch 10/10
313/313 [==============================] - 0s 694us/step - loss: 0.3966 - accuracy:
0.7481
```

Out[6]:

```
<keras.callbacks.History at 0x7fabfefa8eb0>
```

In [7]:

```
# 检验分类效果
test_points = ((2.0, 2.0),
               (-2.0, 2.0),
               (-2.0, -2.0),
               (2.0, -2.0))

model.predict(test_points)
```

```
1/1 [==============================] - 0s 38ms/step
```

Out[7]:

```
array([[1.0000000e+00, 1.0000000e+00],
       [1.4224477e-01, 8.3764815e-01],
       [7.4592097e-29, 7.2304321e-29],
       [8.5778093e-01, 1.6240819e-01]], dtype=float32)
```

上面用到的神经网络十分简单
实际使用的神经网络往往很复杂

但是搭建的难度并没有变大
要做的只是把各模块组装起来
只要熟悉神经网络的每一个部分
看到示意图就能搭建出来
例如：https://keras.io/examples/vision/image_classification_from_scratch/#build-a-model (https://keras.io/examples/vision/image_classification_from_scratch/#build-a-model)