2022-2023 学年秋季学期《模式识别》作业二

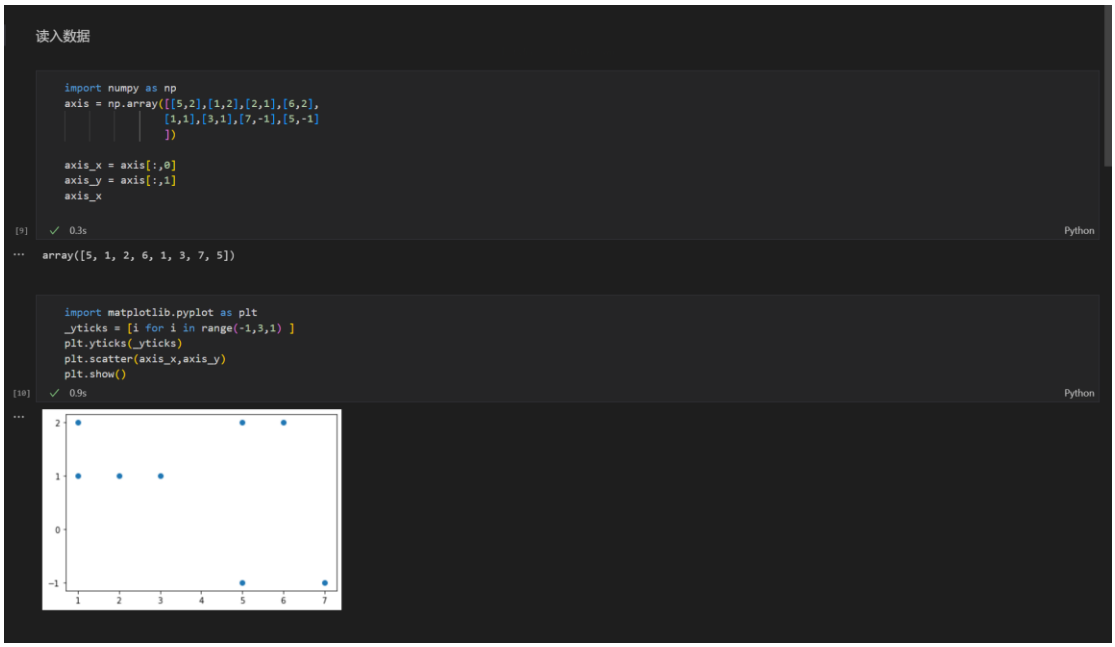
（聚类分析、特征选择和特征提取）

教师：王玮

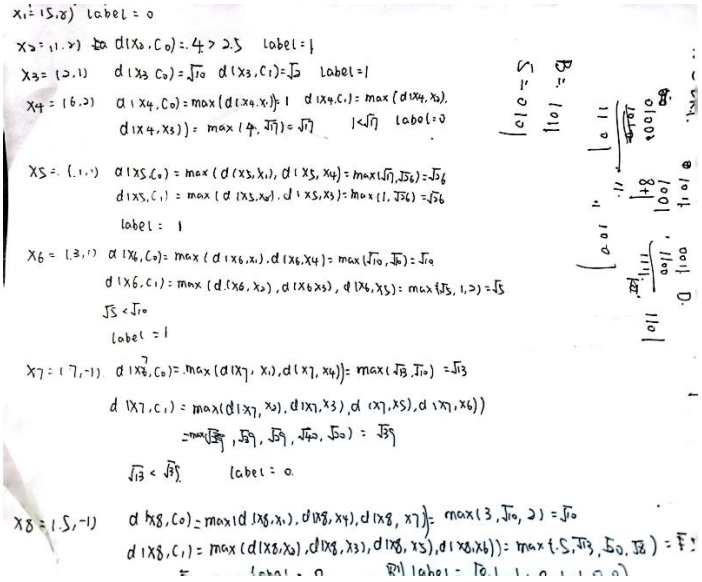
姓名：张奕桢 学号：200111102

1. 分别采用顺序聚类，谱系聚类和 K-均值聚类算法将下列 8 个样本聚成两个类别，并用 Dunn 指数和 Davies-Bouldin 指数评估聚类效果。

$$x_1 = (5,2)^T, x_2 = (1,2)^T, x_3 = (2,1)^T, x_4 = (6,2)^T$$
$$x_5 = (1,1)^T, x_6 = (3,1)^T, x_7 = (7,-1)^T, x_8 = (5,-1)^T$$



顺序聚类是手算的



```
axis
```

[11] ✓ 0.4s Python

```
array([[ 5,  2],
       [ 1,  2],
       [ 2,  1],
       [ 6,  2],
       [ 1,  1],
       [ 3,  1],
       [ 7, -1],
       [ 5, -1]])
```

顺序聚类

Markdown

+ 代码 + Markdown

```
label=[0,1,1,0,1,1,0,0]
label
```

[20] ✓ 0.3s Python

```
[0, 1, 1, 0, 1, 1, 0, 0]
```

Kmeans

Markdown

+ 代码 + Markdown

```
from sklearn.cluster import KMeans
clf = KMeans(n_clusters=2)
y_pred_kmeans = clf.fit_predict(axis)
# 输出完整Kmeans函数，包括很多省略参数
print('k均值模型:\n',clf)
# 输出聚类预测结果，20行数据，每个y_pred对应X一行成一个球员，聚成3类，类标为0、1、2
print('聚类结果:\n',y_pred_kmeans)
```

[17] ✓ 0.6s Python

```
k均值模型:
 KMeans(n_clusters=2)
聚类结果:
 [0 1 1 0 1 1 0 0]
```

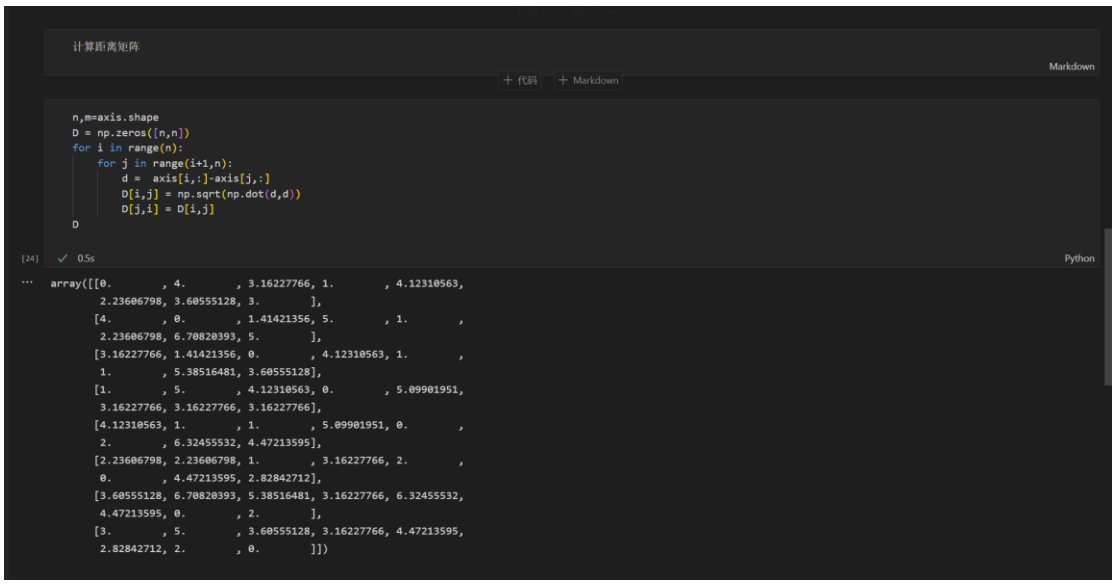
谱聚类

Markdown

```
from sklearn.cluster import SpectralClustering as sc
clf= sc(gamma=1,n_clusters=2)
y_pred_sc = clf.fit_predict(axis)
# 输出完整Kmeans函数，包括很多省略参数
print('谱聚类模型:\n',clf)
# 输出聚类预测结果，20行数据，每个y_pred对应X一行成一个球员，聚成3类，类标为0、1、2
print('聚类结果:\n',y_pred_sc)
```

[18] ✓ 0.5s Python

```
谱聚类模型:
 SpectralClustering(gamma=1, n_clusters=2)
聚类结果:
 [0 1 1 0 1 1 0 0]
```



```
计算距离矩阵
```

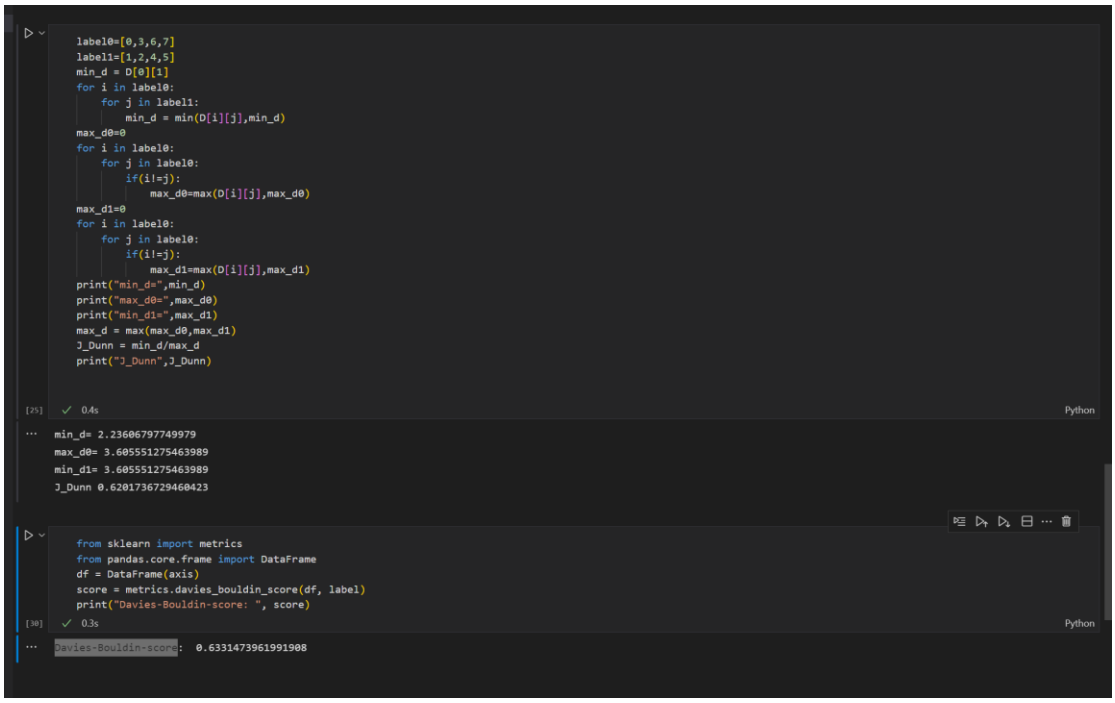
Markdown

+ 代码 + Markdown

```
n,m=axis.shape
D = np.zeros([n,n])
for i in range(n):
    for j in range(i+1,n):
        d =  axis[i,:]-axis[j,:]
        D[i,j] = np.sqrt(np.dot(d,d))
        D[j,i] = D[i,j]
D
```

[24] ✓ 0.5s Python

```
array([[0.        , 4.        , 3.16227766, 1.        , 4.12310563,
        2.23606798, 3.60555128, 3.        ],
       [4.        , 0.        , 1.41421356, 5.        , 1.        ,
        2.23606798, 6.70820393, 5.        ],
       [3.16227766, 1.41421356, 0.        , 4.12310563, 1.        ,
        1.        , 5.38516481, 3.60555128],
       [1.        , 5.        , 4.12310563, 0.        , 5.09901951,
        3.16227766, 3.16227766, 3.16227766],
       [4.12310563, 1.        , 1.        , 5.09901951, 0.        ,
        2.        , 6.32455532, 4.47213595],
       [2.23606798, 2.23606798, 1.        , 3.16227766, 2.        ,
        0.        , 4.47213595, 2.82842712],
       [3.60555128, 6.70820393, 5.38516481, 3.16227766, 6.32455532,
        4.47213595, 0.        , 2.        ],
       [3.        , 5.        , 3.60555128, 3.16227766, 4.47213595,
        2.82842712, 2.        , 0.        ]])
```



```
label0=[0,3,6,7]
label1=[1,2,4,5]
min_d = D[0][1]
for i in label0:
    for j in label1:
        min_d = min(D[i][j],min_d)
max_d0=0
for i in label0:
    for j in label0:
        if(i!=j):
            max_d0=max(D[i][j],max_d0)
max_d1=0
for i in label0:
    for j in label0:
        if(i!=j):
            max_d1=max(D[i][j],max_d1)
print("min_d=",min_d)
print("max_d0=",max_d0)
print("min_d1=",max_d1)
max_d = max(max_d0,max_d1)
J_Dunn = min_d/max_d
print("J_Dunn",J_Dunn)
```

[25] ✓ 0.4s Python

```
min_d= 2.23606797749979
max_d0= 3.605551275463989
min_d1= 3.605551275463989
J_Dunn 0.620173672946042
```

```
from sklearn import metrics
from pandas.core.frame import DataFrame
df = DataFrame(axis)
score = metrics.davies_bouldin_score(df, label)
print("Davies-Bouldin-score: ", score)
```

[30] ✓ 0.3s Python

```
Davies-Bouldin-score:  0.6331473961991908
```

由于三种方法聚类结果是一样的，因此算出的两种评分标准是一样的

2. 根据下列 3 组高斯分布参数分别生成 200 个 2 维随机样本，使用 K-均值算法将所有样本聚成若干类别，采用类内距离准则确定合理聚类数。

$$\mu_1 = (-1,-1)^T, \mu_2 = (6,2)^T, \mu_3 = (0.5,7)^T$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$$

```python
import numpy as np
mean1=(-1,1)
mean2=(6,2)
mean3=(0.5,7)
cov1=[[1,0.5],[0.5,1]]
cov2=[[1,0],[0,1]]
cov3=[[1,-0.5],[-0.5,1]]
A1 = np.random.multivariate_normal(mean1,cov1,200)
A2 = np.random.multivariate_normal(mean2,cov2,200)
A3 = np.random.multivariate_normal(mean3,cov3,200)
A = np.concatenate((A1,A2,A3),axis=0)
```

[5] Python

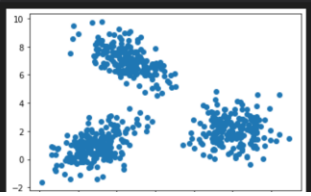
```python
A
```

[9] Python

```
array([[-1.18022477,  1.36555331],
       [-0.10221122,  0.42537953],
       [-3.35682714,  0.33110084],
       ...,
       [ 1.64514988,  6.04739258],
       [-0.03800919,  7.51121969],
       [ 2.04602329,  5.29914342]])
```

```python
x = A[:,0]
y = A[:,1]
import matplotlib.pyplot as plt
plt.scatter(x,y)
plt.show()
```

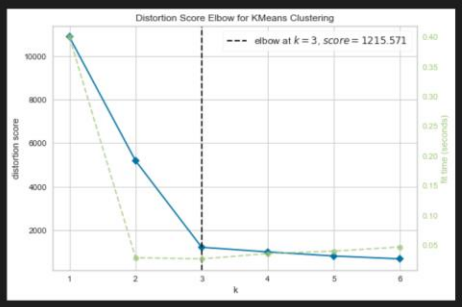
[12] Python



```python
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer

model = KMeans()
visualizer = KElbowVisualizer(model,k=(1,7))
visualizer.fit(A)
visualizer.show()
```

[13] Python

```
h:\ANACONDA\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than
available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=3.
  warnings.warn(
```




```
<AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>
```

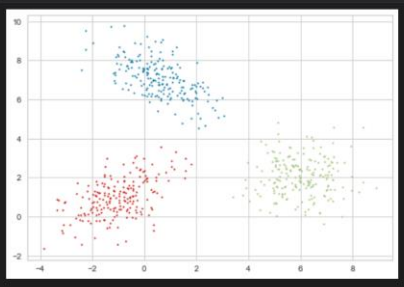
```python
kmeans = KMeans(n_clusters=3).fit(A)
colors = ['b', 'g', 'r']
A=A.tolist()
for id,label in enumerate(kmeans.labels_):
    plt.scatter(A[id][0],A[id][1],color =colors[label],marker='o',s=3)
plt.show()
```

其中 distortion_score 是没有求类内距离平均之前的值。（和平均之后是等价的）

```
<AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>
```

```
kmeans = KMeans(n_clusters=3).fit(A)
colors = ['b', 'g', 'r']
A=A.tolist()
for id,label in enumerate(kmeans.labels_):
    plt.scatter(A[id][0],A[id][1],color =colors[label],marker='o',s=3)
plt.show()
```

Python



3. 有两类样本集合:

• 类别 1 中有 5 个样本$c_1 = [(1,2),(2,3),(3,3),(4,5),(5,5)]$

• 类别 2 中有 6 个样本$\boldsymbol{c_2} = [(1,0),(2,1),(3,1),(3,2),(5,3),(6,5)]$

（1）使用主成分分析将特征降为 1 维，在二维空间中画出样本和投影坐标轴。

（2）使用 Fisher 判别分析将特征降为 1 维，在二维空间中画出样本和投影坐标轴。

（3）在使用 Fisher 判别分析降维后的 1 维空间中构建线性分类器区分两类样本。

（4）如果在主成分分析降维后的 1 维空间中构建线性分类器，效果是否理想？为什么？

（1）

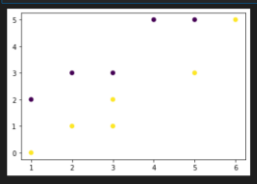
```
import pandas as pd
data =[[1,2,1],
       [2,3,1],
       [3,3,1],
       [4,5,1],
       [5,5,1],
       [1,0,2],
       [2,1,2],
       [3,1,2],
       [3,2,2],
       [5,3,2],
       [6,5,2]]
df = pd.DataFrame(data,columns=['X','Y','target'])
print(df)
```

0.4s

Python

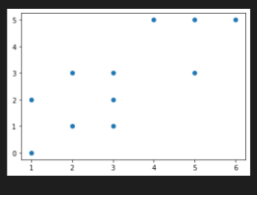
```
    X  Y  target
0   1  2       1
1   2  3       1
2   3  3       1
3   4  5       1
4   5  5       1
5   1  0       2
6   2  1       2
7   3  1       2
8   3  2       2
9   5  3       2
10  6  5       2
```

```python
import matplotlib.pyplot as plt
plt.scatter(df.X,df.Y,c=df.target)
plt.show()
```

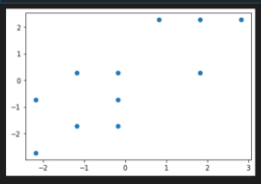


```python
plt.scatter(X[:,0],X[:,1])
```

```
<matplotlib.collections.PathCollection at 0x1d10031646>
```



```python
import numpy as np
def demean(X):
    # axis=0按列计算均值，即每个属性的均值，1则是计算行的均值
    return (X - np.mean(X, axis=0))

X_demean = demean(X)
# 注意看数据分布没变，但是坐标已经以原点为中心了
plt.scatter(X_demean[:, 0], X_demean[:, 1])
plt.show()
```



```python
def f(w,X):
    return np.sum((X.dot(w)**2))/len(X)
```

```python
def df_math(w,X):
    return X.T.dot(X.dot(w))*2./len(X)
```

+ 代码 + Markdown

```python
def f(w,X):
    return np.sum((X.dot(w)**2))/len(X)
```

```python
def df_math(w,X):
    return X.T.dot(X.dot(w))*2./len(X)
```

```python
# 验证梯度求解是否正确，使用梯度调试方法。
def df_debug(w, X, epsilon=0.0001):
    # 先创建一个与参数组等长的向量
    res = np.empty(len(w))
    # 对于每个特征，求值
    for i in range(len(w)):
        w_1 = w.copy()
        w_1[i] += epsilon
        w_2 = w.copy()
        w_2[i] -= epsilon
        res[i] = (f(w_1, X) - f(w_2, X)) / (2 * epsilon)
    return res
```

```python
def direction(w):
    return w / np.linalg.norm(w)
```

```python
# 梯度上升法代码
def gradient_ascent(df, X, initial_w, eta, n_iters=1e4, epsilon=1e-8):
    w = direction(initial_w)
    cur_iter = 0
    while cur_iter < n_iters:
        gradient = df_math(w,X)
        last_w = w
        w = last_w + eta * gradient
        w = direction(w)    # 注意时换成单位向量
        if (abs(f(w,X) - f(last_w,X)) < epsilon):
            break
        cur_iter += 1
```

```
initial_w = np.random.random(X.shape[1])
eta = 0.001
```

0.3s Python

```
w = gradient_ascent(df_debug, X_demean, initial_w, eta)
w
```

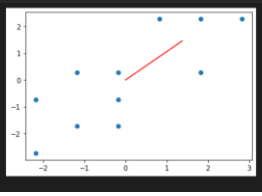
0.4s Python

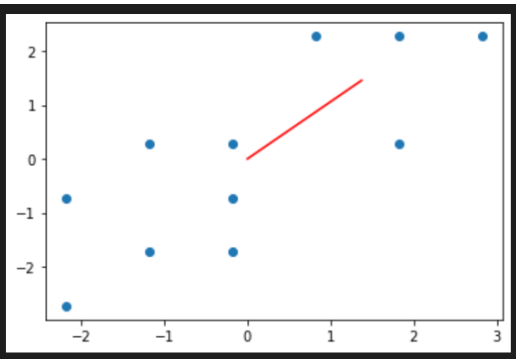
```
array([0.68716927, 0.72649735])
```

+ 代码 + Markdown

```
plt.scatter(X_demean[:,0],X_demean[:,1])
plt.plot([0,w[0]*2],[0,w[1]*2], color='red')
plt.show()
```

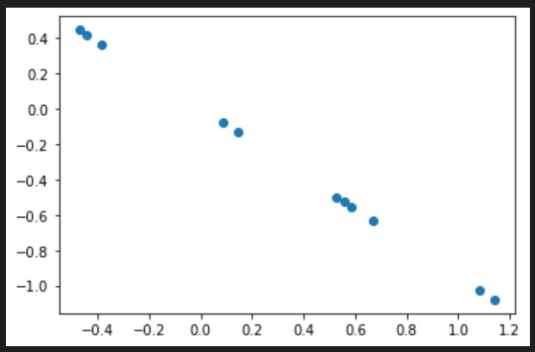
0.1s Python





```
X_new = X - X.dot(w).reshape(-1,1) * w
plt.scatter(X_new[:,0], X_new[:,1])
plt.show()
```

[30] ✓ 0.1s



```python
import pandas as pd
import numpy as np
data =[[1,2,1],
       [2,3,1],
       [3,3,1],
       [4,5,1],
       [5,5,1],
       [1,0,2],
       [2,1,2],
       [3,1,2],
       [3,2,2],
       [5,3,2],
       [6,5,2]]
df = pd.DataFrame(data,columns=['X','Y','target'])
df_X=df[['X','Y']]
X=df_X.to_numpy()
df_Y=df[['target']]
Y=df_Y.to_numpy()
c_1=X[0:5][:]
c_2=X[5:-1][:]
```

[17] ✓ 0.3s Python

```python
def cal_cov_and_avg(samples):
    """
    给定一个类别的数据，计算协方差矩阵和平均向量
    :param samples:
    :return:
    """
    u1 = np.mean(samples, axis=0)
    cov_m = np.zeros((samples.shape[1], samples.shape[1]))
    for s in samples:
        t = s - u1
        cov_m += t * t.reshape(2, 1)
    return cov_m, u1


def fisher(c_1, c_2):
    """
    fisher算法实现(请参考上面推导出来的公式，那个才是精华部分)
    :param c_1:
    :param c_2:
    :return:
    """
    cov_1, u1 = cal_cov_and_avg(c_1)
    cov_2, u2 = cal_cov_and_avg(c_2)
    s_w = cov_1 + cov_2
    u, s, v = np.linalg.svd(s_w)  # 奇异值分解
    s_w_inv = np.dot(np.dot(v.T, np.linalg.inv(np.diag(s))), u.T)
    return np.dot(s_w_inv, u1 - u2)
```

[18] ✓ 0.3s Python

```python
def judge(sample, w, c_1, c_2):
    """
    true 属于1
    false 属于2
    :param sample:
```

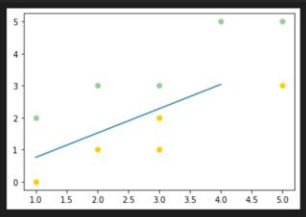
```
    :param center_2:
    :return:
    """
    u1 = np.mean(c_1, axis=0)
    u2 = np.mean(c_2, axis=0)
    center_1 = np.dot(w.T, u1)
    center_2 = np.dot(w.T, u2)
    pos = np.dot(w.T, sample)
    return abs(pos - center_1) < abs(pos - center_2)


w = fisher(c_1, c_2)  # 调用函数，得到参数w
out = judge(c_1[1], w, c_1, c_2)  # 判断所属的类别
print(out)
```

[19] ✓ 0.2s Python

```
True
```

```
import matplotlib.pyplot as plt

plt.scatter(c_1[:, 0], c_1[:, 1], c='#99CC99')
plt.scatter(c_2[:, 0], c_2[:, 1], c='#FFCC00')
line_x = np.arange(min(np.min(c_1[:, 0]), np.min(c_2[:, 0])),
                   max(np.max(c_1[:, 0]), np.max(c_2[:, 0])),
                   step=1)

line_y = - (w[0] * line_x) / w[1]
plt.plot(line_x, line_y)
plt.show()
```

[20] ✓ 0.1s Python



第3问

Markdown

```
from sklearn import discriminant_analysis
#定义Fisher分类器对象fisher_clf
fisher_clf = discriminant_analysis.LinearDiscriminantAnalysis()
#调用该对象的训练方法
fisher_clf.fit(X,Y)
```

[4] ✓ 0.8s Python

```
h:\ANACONDA\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape
```

第3问

Markdown

```
from sklearn import discriminant_analysis
#定义Fisher分类器对象fisher_clf
fisher_clf = discriminant_analysis.LinearDiscriminantAnalysis()
#调用该对象的训练方法
fisher_clf.fit(X,Y)
```

[4] ✓ 0.8s Python

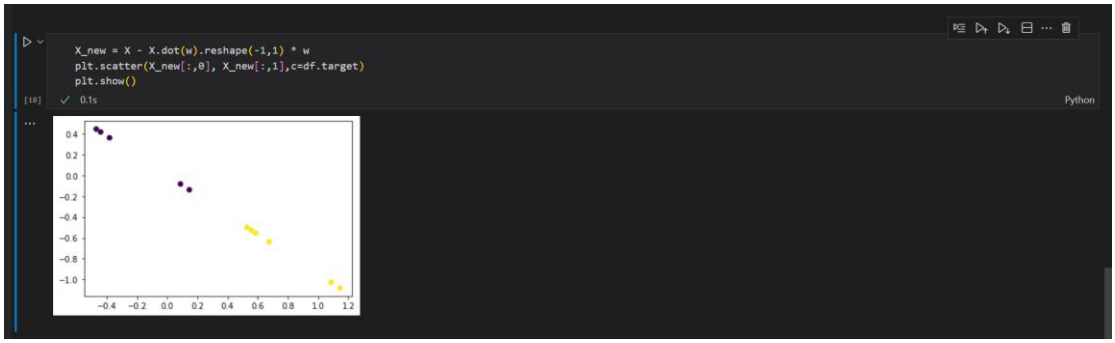
```
h:\ANACONDA\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape
of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

LinearDiscriminantAnalysis()
```

```
y_pred=fisher_clf.predict(X)
print('测试数据集的正确标签为:',Y)
print('测试数据集的预测标签为:',y_pred)
from sklearn.metrics import accuracy_score
testing_acc=accuracy_score(Y, y_pred)*100
print('Fisher线性分类器测试准确率: {:.2f}%'.format(testing_acc))
```

[5] ✓ 0.3s Python

```
测试数据集的正确标签为: [[1]
 [1]
 [1]
 [1]
 [1]
 [2]
 [2]
 [2]
 [2]
 [2]
 [2]]
测试数据集的预测标签为: [1 1 1 1 1 2 2 2 2 2]
Fisher线性分类器测试准确率: 100.00%
```

（4）



比较理想因为各个点之间没有相互交叉，特征比较明显