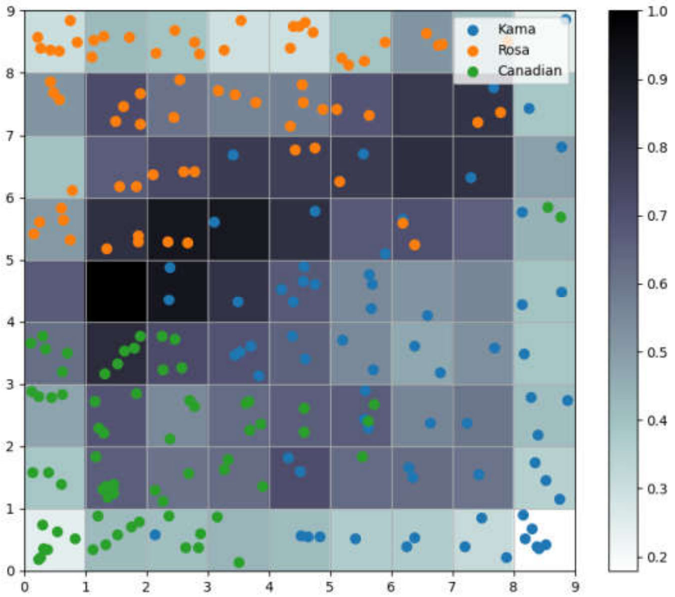
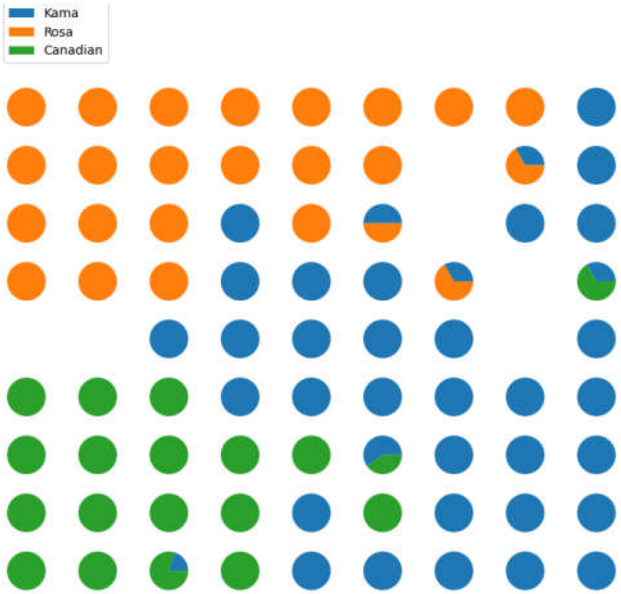
# Python编程与人工智能实践

## 算法篇：SOM（Self Organizing Maps，自组织映射）

## 应用：聚类、分类





于泓
鲁东大学
信息与电气工程学院

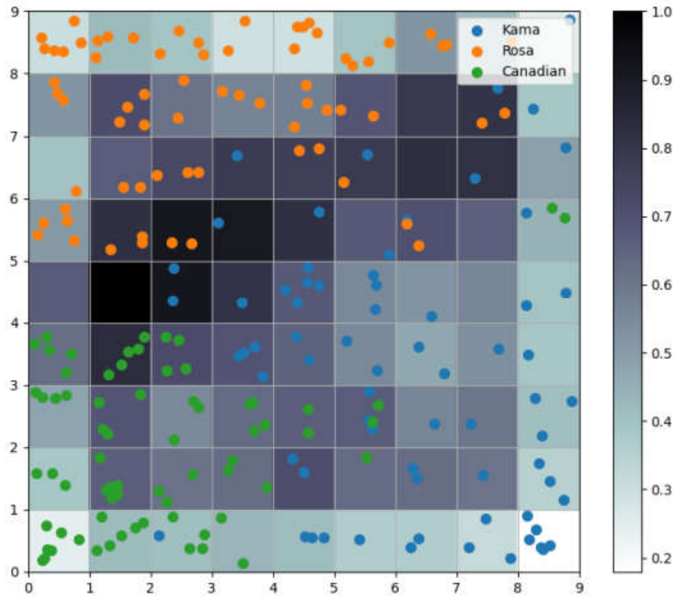
2021.12.24

# SOM应用

- (1)  映射效果展示
- (2)  聚类
- (3)  分类

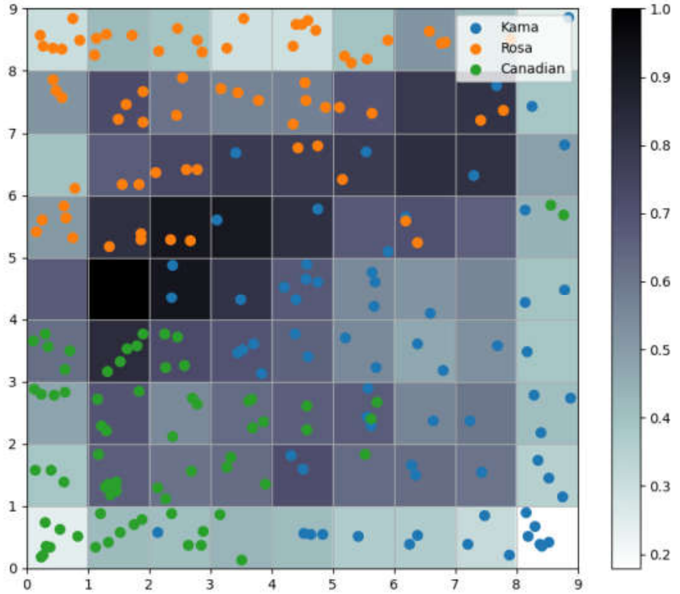
## 效果展示

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from SOM import train_SOM,feature_normalization,get_U_Matrix,get_winner_index,weights_PCA
from collections import defaultdict, Counter
import matplotlib.gridspec as gridspec
if __name__ == "__main__":

    # seed 数据展示
    columns=['area', 'perimeter', 'compactness', 'length_kernel', 'width_kernel',
                   'asymmetry_coefficient', 'length_kernel_groove', 'target']
    data = pd.read_csv('seeds_dataset.txt',
                    names=columns,
                   sep='\t+', engine='python')
    labs = data['target'].values
    label_names = {1:'Kama', 2:'Rosa', 3:'Canadian'}
    datas = data[data.columns[:-1]].values
    N,D = np.shape(datas)
    print(N,D)

    # 对训练数据进行正则化处理
    datas = feature_normalization(datas)

    # SOM的训练
    X=9
    Y=9
    weights = train_SOM(X=X,Y=Y,N_epoch=4,datas=datas,sigma=1.5,init_weight_fun=weights_PCA)

    # 获取UMAP
    UM = get_U_Matrix(weights)
```

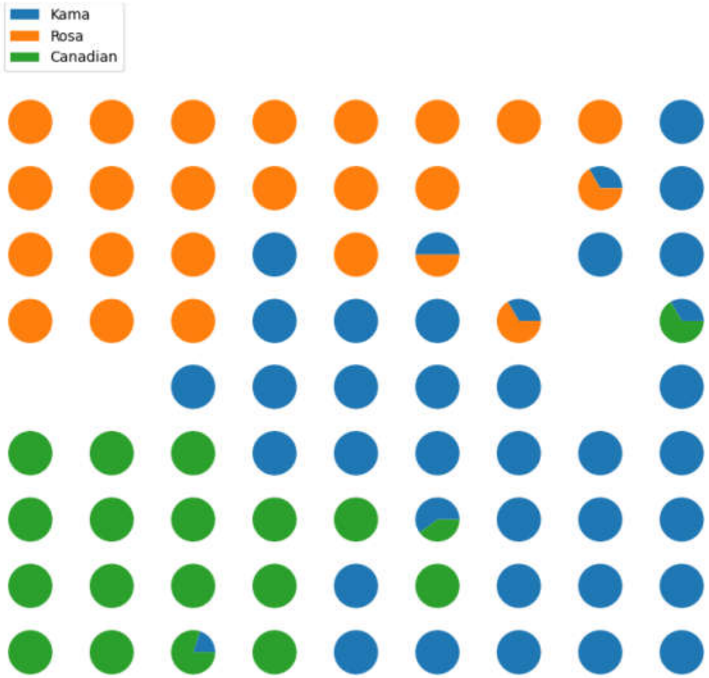


→ 计算聚类中心之间的关系

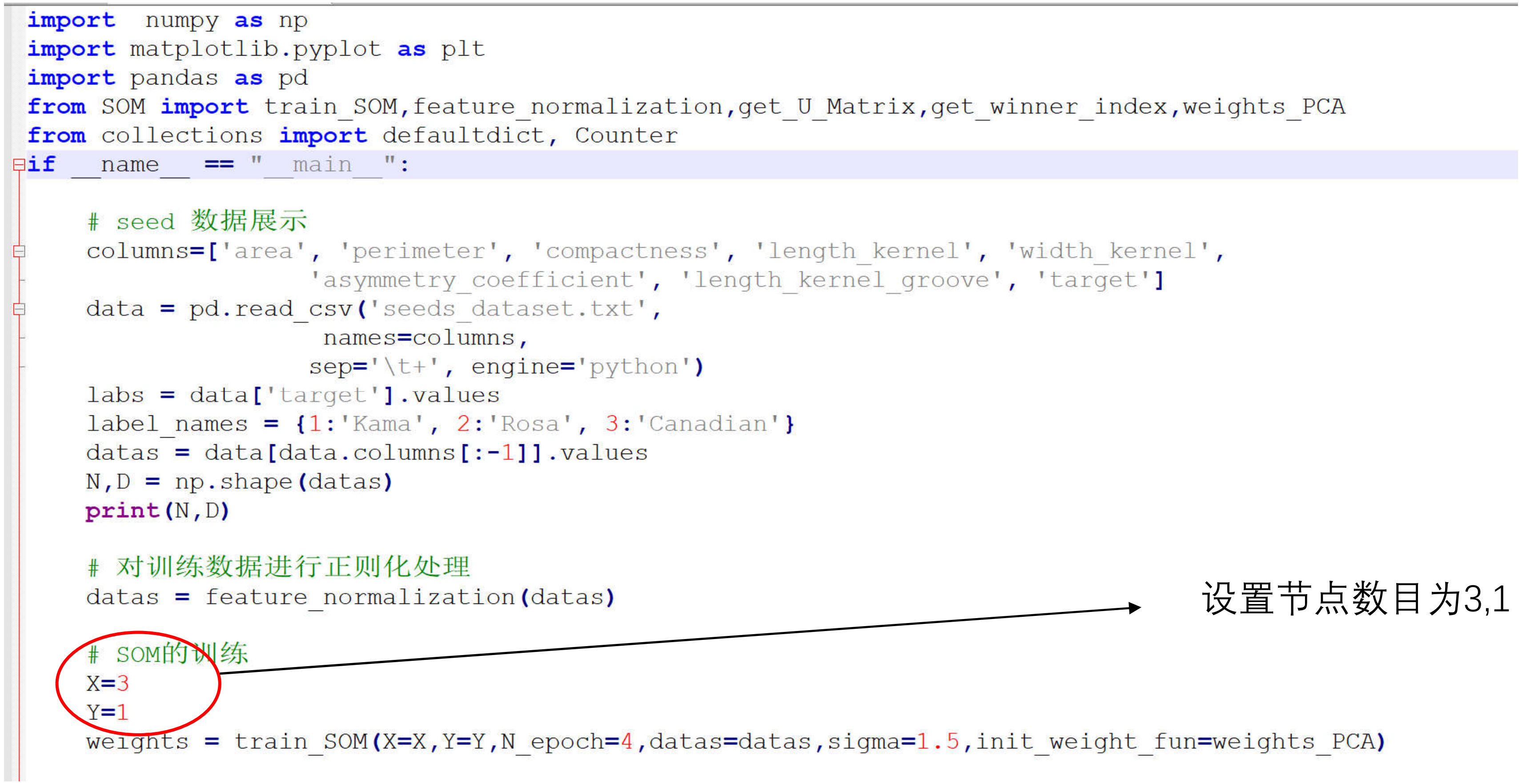
```
# '''画散点图'''

# 显示UMAP
plt.figure(1,figsize=(9, 9))
plt.pcolor(UM.T, cmap='bone_r')  # plotting the distance map as background
plt.colorbar()

markers = ['o', 's', 'D']
colors = ['C0', 'C1', 'C2']

# 计算每个样本点投射后的坐标
w_x, w_y = zip(*[get_winner_index(d,weights) for d in datas])
w_x = np.array(w_x)
w_y = np.array(w_y)

# 分别把每一类的散点在响应的方格内进行打印（+随机位置偏移）
for c in np.unique(labs):
    idx_target = (labs==c)
    plt.scatter(w_x[idx_target]+.5+(np.random.rand(np.sum(idx_target))-.5)*.8,
                w_y[idx_target]+.5+(np.random.rand(np.sum(idx_target))-.5)*.8,
                s=50, c=colors[c-1], label=label_names[c])
plt.legend(loc='upper right')
plt.grid()
```



```
''' 画饼图'''
# 计算输出层的每个节点上映射了哪些数据
win_map = defaultdict(list)
for x,lab in zip(datas,labs):
    win_map[get_winner_index(x,weights)].append(lab)

# 统计每个输出节点上，映射了各类数据、各多少个
for pos in win_map:
    win_map[pos] = Counter(win_map[pos])

fig = plt.figure(2,figsize=(9, 9))
# 按照 X,Y对画面进行分格
the_grid = gridspec.GridSpec(Y, X, fig)
print(the_grid)

# 在每个格子里面画饼图
for pos in win_map.keys():
    label_fracs = [win_map[pos][l] for l in label_names.keys()]

    plt.subplot(the_grid[Y-1-pos[1],
                         pos[0]], aspect=1)
    patches, texts = plt.pie(label_fracs)


plt.legend(labels = label_names.values(), loc='upper left',bbox_to_anchor=(-6, 10))
# plt.savefig('resulting_images/som_seed_pies.png')
plt.show()
```



# 聚类

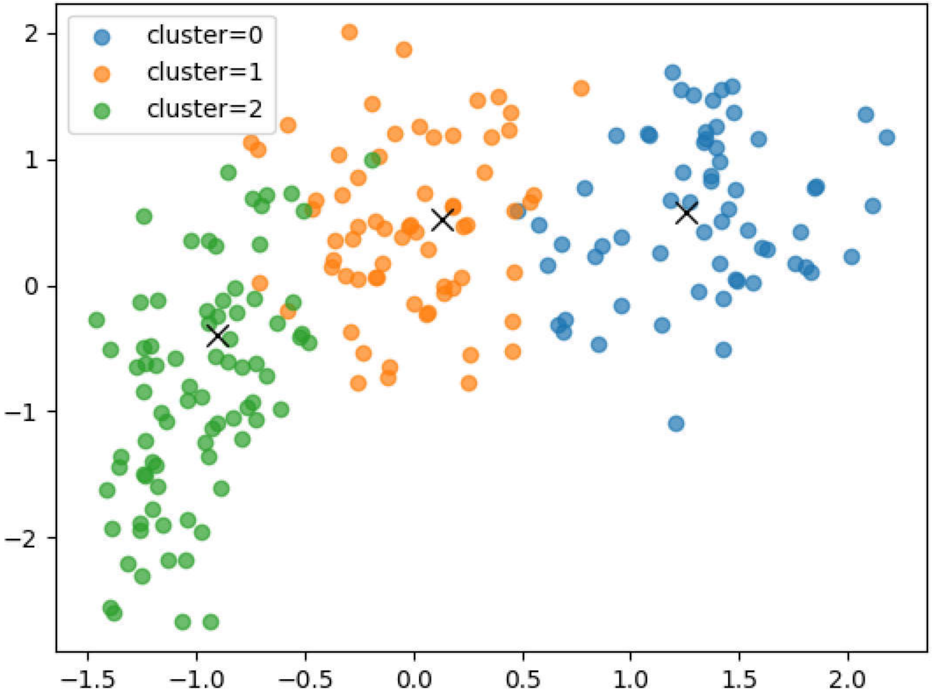


```python
import  numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from SOM import train_SOM,feature_normalization,get_U_Matrix,get_winner_index,weights_PCA
from collections import defaultdict, Counter
if __name__ == "__main__":

    # seed 数据展示
    columns=['area', 'perimeter', 'compactness', 'length_kernel', 'width_kernel',
                   'asymmetry_coefficient', 'length_kernel_groove', 'target']
    data = pd.read_csv('seeds_dataset.txt',
                    names=columns,
                   sep='\t+', engine='python')
    labs = data['target'].values
    label_names = {1:'Kama', 2:'Rosa', 3:'Canadian'}
    datas = data[data.columns[:-1]].values
    N,D = np.shape(datas)
    print(N,D)

    # 对训练数据进行正则化处理
    datas = feature_normalization(datas)

    # SOM的训练
    X=3
    Y=1
    weights = train_SOM(X=X,Y=Y,N_epoch=4,datas=datas,sigma=1.5,init_weight_fun=weights_PCA)
```

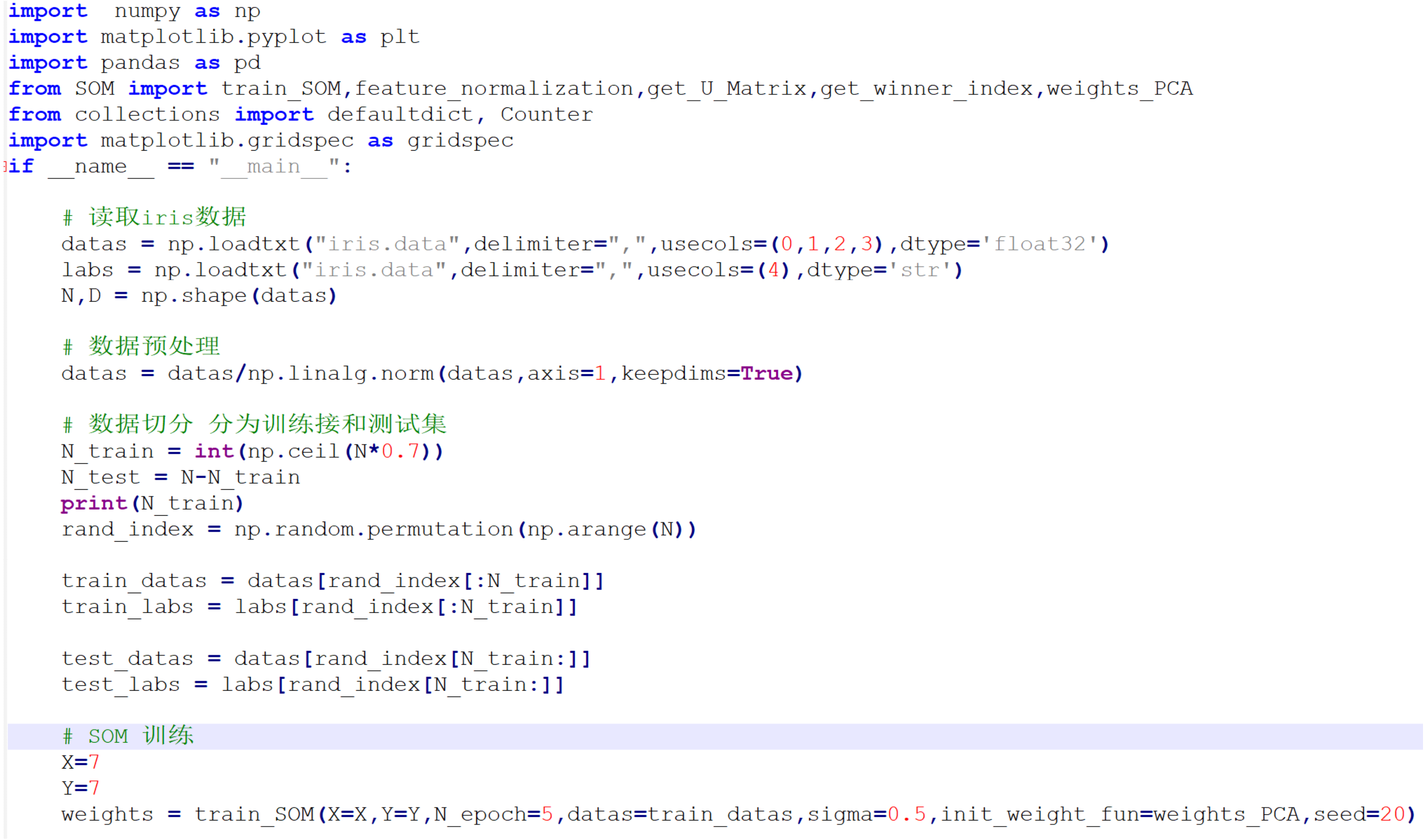
设置节点数目为3,1

```
# 实现聚类

# 获取聚类的编号
index_clusters = []
for i in range(N):
    x = datas[i]
    winner = get_winner_index(x,weights)
    index_clusters.append(winner[0]*Y+winner[1])


for c in np.unique(index_clusters):

    ii = np.where(index_clusters==c)[0]

    plt.scatter(datas[ii, 0],
                datas[ii, 2], label='cluster='+str(c), alpha=.7)
plt.legend()
for i in range(X):
    for j in range(Y):
        plt.scatter(weights[i,j,0], weights[i,j,2], marker='x',
            s=80, linewidths=1, color='k')
plt.legend()
plt.show()
```

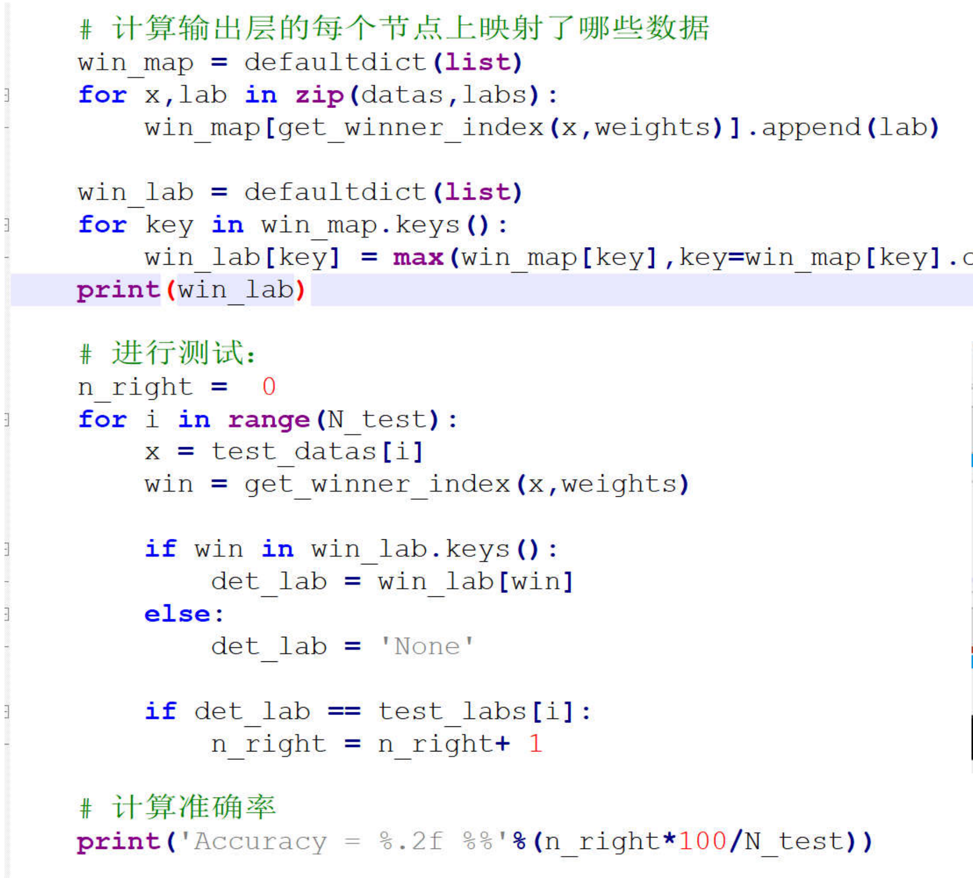
获得聚类的编号



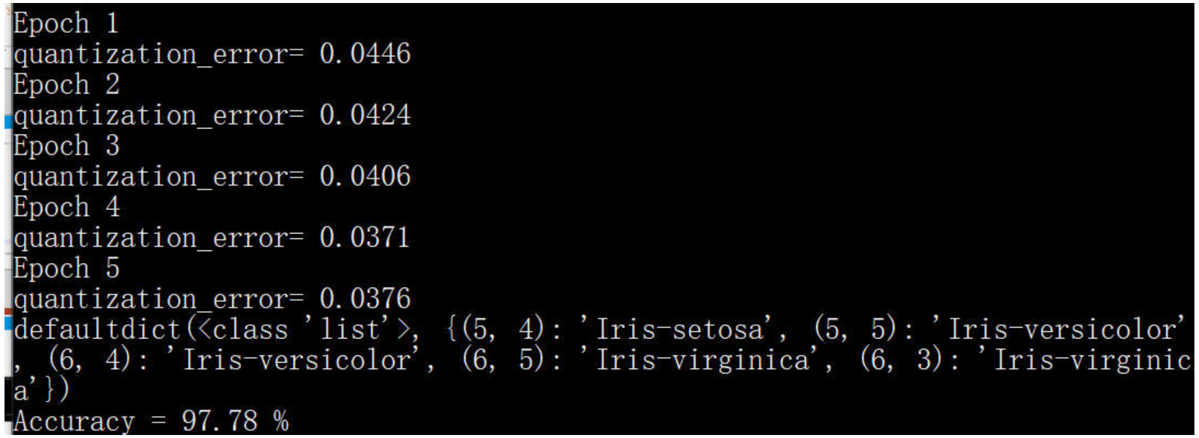
分类



```
import  numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from SOM import train_SOM,feature_normalization,get_U_Matrix,get_winner_index,weights_PCA
from collections import defaultdict, Counter
import matplotlib.gridspec as gridspec
if __name__ == "__main__":

    # 读取iris数据
    datas = np.loadtxt("iris.data",delimiter=",",usecols=(0,1,2,3),dtype='float32')
    labs = np.loadtxt("iris.data",delimiter=",",usecols=(4),dtype='str')
    N,D = np.shape(datas)

    # 数据预处理
    datas = datas/np.linalg.norm(datas,axis=1,keepdims=True)

    # 数据切分 分为训练接和测试集
    N_train = int(np.ceil(N*0.7))
    N_test = N-N_train
    print(N_train)
    rand_index = np.random.permutation(np.arange(N))

    train_datas = datas[rand_index[:N_train]]
    train_labs = labs[rand_index[:N_train]]

    test_datas = datas[rand_index[N_train:]]
    test_labs = labs[rand_index[N_train:]]

    # SOM 训练
    X=7
    Y=7
    weights = train_SOM(X=X,Y=Y,N_epoch=5,datas=train_datas,sigma=0.5,init_weight_fun=weights_PCA,seed=20)
```



```
# 计算输出层的每个节点上映射了哪些数据
win_map = defaultdict(list)
for x,lab in zip(datas,labs):
    win_map[get_winner_index(x,weights)].append(lab)

win_lab = defaultdict(list)
for key in win_map.keys():
    win_lab[key] = max(win_map[key],key=win_map[key].count)
print(win_lab)

# 进行测试:
n_right =  0
for i in range(N_test):
    x = test_datas[i]
    win = get_winner_index(x,weights)

    if win in win_lab.keys():
        det_lab = win_lab[win]
    else:
        det_lab = 'None'

    if det_lab == test_labs[i]:
        n_right = n_right+ 1

# 计算准确率
print('Accuracy = %.2f %%'%(n_right*100/N_test))
```



```
Epoch 1
quantization_error= 0.0446
Epoch 2
quantization_error= 0.0424
Epoch 3
quantization_error= 0.0406
Epoch 4
quantization_error= 0.0371
Epoch 5
quantization_error= 0.0376
defaultdict(<class 'list'>, {(5, 4): 'Iris-setosa', (5, 5): 'Iris-versicolor'
, (6, 4): 'Iris-versicolor', (6, 5): 'Iris-virginica', (6, 3): 'Iris-virginic
a'})
Accuracy = 97.78 %
```