模型-机器学习-聚类-SOM聚类

1. 模型名称

自组织特征图（Self-Organizing Feature Map，SOM）

2. 模型评价

2.1 缺点

1. 网络结构是固定的，不能动态改变
2. 网络训练时,有些神经元始终不能获胜，成为“死神经元”
3. SOM 网络在没有经过完整的重新学习之前，不能加入新的
4. 当输入数据较少时，训练的结果通常依赖于样本的输入顺序
5. 网络连接权的初始状态、算法中的参数选择对网络的收敛性能有较大影响

2.2 优点

1. 简明性
2. 实用性

3. 基本算法

1. 输入**输入层**：假设一个输入样本$X=[x_1,x_2,x_3,...,x_n]$，则输入层神经元的个数为n个
2. 用较小的随机值初始化**输出层（竞争层）**：通常输出层的神经元以矩阵方式排列在二维空间中，m个神经元有m个**权值向量**$w_i=[w_{i1},w_{i2},...,w_{in}]$，$i=1,2,..,m$
3. 对输入向量做**归一化**：$||X||$为输入的样本向量的欧几里得范数
$$X'=\frac{X}{||X||}$$
4. 对权值向量做**归一化**：$||w_i||$为权值向量的欧几里得范数

$$w_i' = \frac{w_i}{||w_i||}$$

5. 得到**获胜神经元**

   方法一：样本X和每个竞争层的神经元的权值向量w_i**点积**，值最大的为**获胜神经元**

   方法二：计算样本X和每个竞争层的神经元的权值向量的**欧几里得距离**，距离最小的为**获胜神经元**

6. 得到获胜神经元**拓扑邻域**N内的神经元

7. 对获胜神经元**拓扑邻域**N内的每个神经元进行**权值更新**和**归一化**，并更新**学习速率**η和**拓扑邻域**N

$$w_i(t+1) = w_i(t) + \eta(t,d) * (X - w_i(t))$$

$$\eta(t,d):\eta\text{为学习速率，}t\text{为训练时间，}d\text{为该神经元与获胜神经元之间的拓扑距离}$$

$$\eta(t,d) = \eta(t)e^{-d},\quad \eta(t)\text{一般取迭代次数的倒数}$$

8. 判断是否收敛：如果**学习率**$\eta < \eta_{min}$或者**迭代次数**$t > T$，则结束算法

4. 实例

4.1 数据介绍

来源于西瓜书，一共有30个西瓜，每个西瓜有2个不同属性

```
0.697,0.46
0.774,0.376
0.634,0.264
0.608,0.318
0.556,0.215
0.403,0.237
0.481,0.149
0.437,0.211
0.666,0.091
0.243,0.267
0.245,0.057
0.343,0.099
0.639,0.161
0.657,0.198
0.36,0.37
0.593,0.042
0.719,0.103
0.359,0.188
0.339,0.241
0.282,0.257
0.748,0.232
0.714,0.346
0.483,0.312
0.478,0.437
0.525,0.369
0.751,0.489
0.532,0.472
0.473,0.376
```

```
0.725,0.445
0.446,0.459
```

4.2 实验目的

根据西瓜的2个属性，对30个西瓜进行分类

4.3 代码实现

4.3.1 Python自写代码

SOM.py

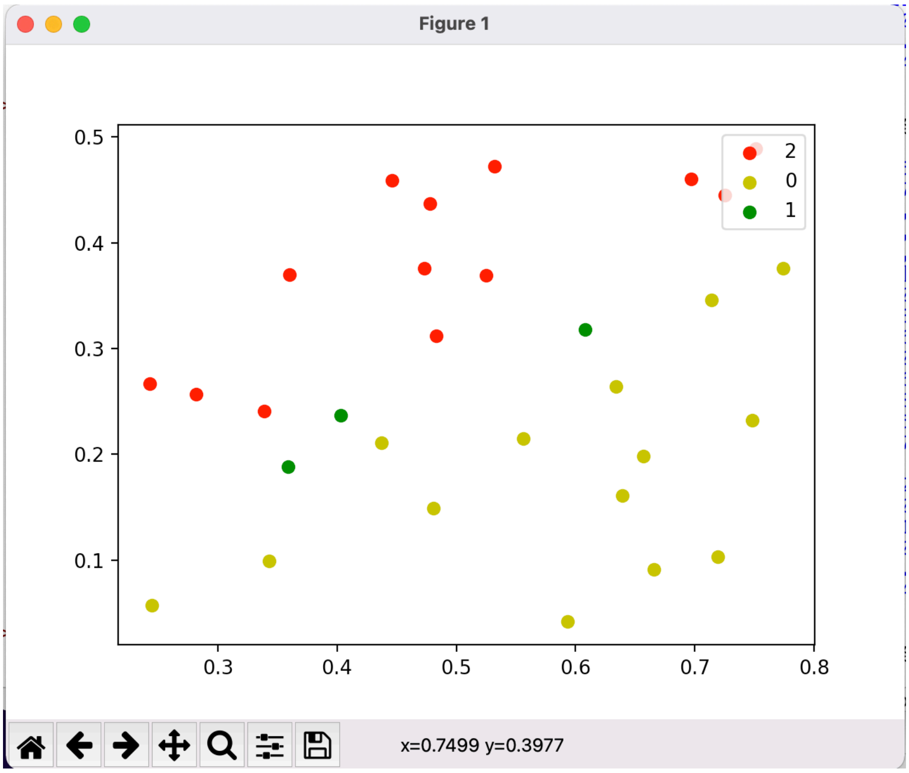
```
# 导入numpy库用于矩阵变换
import numpy as np
from numpy import shape
# 导入math库
import math
# 导入matplotlib库用于画图
import matplotlib.pyplot as plt
# 导入copy库用于实现deepcopy
import copy

# 数据处理
"""
输入：原始数据文件路径名
输出：含所有样本的list，每个元素为一个样本（含所有属性的list）
"""
def loadDataSet(fileName):
    all_data = []
    fr = open(fileName)
    for line in fr.readlines():
        oneLine = line.strip().split(',')
        all_data.append([float(oneLine[0]), float(oneLine[1])])
    return all_data

# 初始化输入层与竞争层神经元的连接权值矩阵
"""
输入：竞争层矩阵的行数n，竞争层矩阵的列数m，输入层每个样本的神经元数（属性数）d
输出：n*m*d的权值矩阵
"""
def initCompetition(n, m, d):
    array = np.random.random(size=n*m*d)
    com_weight = array.reshape(n,m,d)
    return com_weight

# 计算向量的二范数
def cal2NF(X):
    res = 0
    for x in X:
```

```
        res += x*x
    return res ** 0.5

# 对数据集进行归一化处理
def normalize(dataSet):
    for data in dataSet:
        two_NF = cal2NF(data)
        for i in range(len(data)):
            data[i] = data[i] / two_NF
    return dataSet

# 对权值矩阵进行归一化处理
def normalize_weight(com_weight):
    for x in com_weight:
        for data in x:
            two_NF = cal2NF(data)
            for i in range(len(data)):
                data[i] = data[i] / two_NF
    return com_weight

# 获得获胜神经元的索引值（利用方法一：样本和权值向量的点积）
def getWinner(data, norm_weight):
    max_sim = 0
    n,m,d = np.shape(norm_weight)
    mark_n = 0
    mark_m = 0
    for i in range(n):
        for j in range(m):
            result = sum(data * norm_weight[i][j])
            if result > max_sim:
                max_sim = result
                mark_n = i
                mark_m = j
    return (mark_n, mark_m)

# 得到拓扑邻域N中的所有神经元（计算距获胜神经元与每个神经元之间的距离，小于拓扑邻域N则是）
def getNeibor(N_neibor, com_weight):
    res = []
    n,m,_ = shape(com_weight)
    for i in range(n):
        for j in range(m):
            N_float = ((i-n)**2 + (j-m)**2) ** 0.5
            N = int(N_float)
            if N <= N_neibor:
                res.append((i,j,N))
    return res

# 学习率函数
def eta(t,N):
```

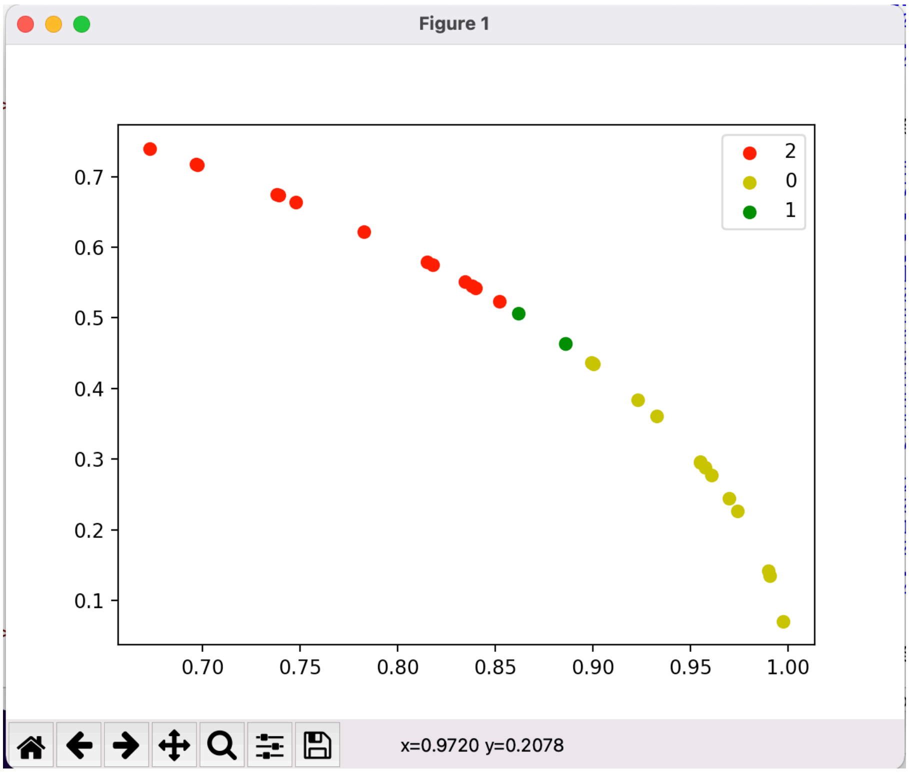
```
    return (0.3/(t+1)) * (math.e ** -N)

# 画图方法（C为聚类之后每个样本的label名称组成的list，用于分类画图）
def draw(C, dataSet):
    color = ['r', 'y', 'g', 'b', 'c', 'k', 'm', 'd']
    count = 0
    for i in C.keys():
        X = []
        Y = []
        datas = C[i]
        for j in range(len(datas)):
            X.append(dataSet[datas[j]][0])
            Y.append(dataSet[datas[j]][1])
        plt.scatter(X, Y, marker = 'o', color = color[count % len(color)], label=i)
        count += 1
    plt.legend(loc='upper right')
    plt.show()


# SOM算法的实现（T为最大迭代次数，N_neibor是初始近邻数）
def do_som(dataSet, com_weight, T, N_neibor):
    for t in range(T-1):
        com_weight = normalize_weight(com_weight)
        for data in dataSet:
            n, m = getWinner(data, com_weight)
            neibor = getNeibor(N_neibor, com_weight)
            for x in neibor:
                j_n = x[0]; j_m = x[1]; N = x[2]
                com_weight[j_n][j_m] = com_weight[j_n][j_m] + eta(t,N)*(data -
com_weight[j_n][j_m])
            N_neibor = N_neibor + 1 - (t + 1) / 200
    res = {}
    N, M, _ = shape(com_weight)
    for i in range(len(dataSet)):
        n, m = getWinner(dataSet[i], com_weight)
        key = n*M + m
        if key in res.keys():
            res[key].append(i)
        else:
            res[key] = []
            res[key].append(i)
    return res

# SOM算法主函数
def SOM(dataSet, com_n, com_m, T, N_neibor):
    old_dataSet = copy.deepcopy(dataSet)
    dataSet = normalize(dataSet)
    com_weight = initCompetition(com_n, com_m, shape(dataSet)[1])
    C_res = do_som(dataSet, com_weight, T, N_neibor)
```

```
    draw(C_res, old_dataSet)
    draw(C_res, dataSet)
    print(old_dataSet)
    print(dataSet)

fileName = '/Users/xinyuanhe/Desktop/working/2021美赛/模型/【正式】模型-机器学习-聚类-SOM聚类【hxy】/dataset.txt'
dataSet = loadDataSet(fileName)
# n=2为竞争层神经元的行数，m=2为竞争层神经元的列数，T=4为最大迭代次数，N_neibor=2为初始邻域
# 可改变参数做敏感性分析
# 一般n=m=math.ceil(np.sqrt(5 * np.sqrt(样本个数)))比较合适
# 迭代次数一般可选200次
SOM(dataSet, 2, 2, 4, 2)
```

原始数据聚类图：



归一化后数据聚类图：



原始数据和归一化后数据：

```
>>>
= RESTART: /Users/xinyuanhe/Desktop/working/2021美赛/模型/【正式】模型-机器学习-聚类-SOM聚类【hxy】/SOM.py
[[0.697, 0.46], [0.774, 0.376], [0.634, 0.264], [0.608, 0.318], [0.556, 0.215],
[0.403, 0.237], [0.481, 0.149], [0.437, 0.211], [0.666, 0.091], [0.243, 0.267],
[0.245, 0.057], [0.343, 0.099], [0.639, 0.161], [0.657, 0.198], [0.36, 0.37], [0
.593, 0.042], [0.719, 0.103], [0.359, 0.188], [0.339, 0.241], [0.282, 0.257], [0
.748, 0.232], [0.714, 0.346], [0.483, 0.312], [0.478, 0.437], [0.525, 0.369], [0
.751, 0.489], [0.532, 0.472], [0.473, 0.376], [0.725, 0.445], [0.446, 0.459]]
[[0.8346204166330251, 0.5508255260418817], [0.8994820591298843, 0.43695769280728
23], [0.9231630592739966, 0.3844085925052604], [0.8861166297173975, 0.4634623162
008757], [0.9326955612109484, 0.3606646504682624], [0.8619891954620765, 0.506926
6484479209], [0.9552190874043465, 0.2958994678237996], [0.9005238141415731, 0.43
48066928692721], [0.9907939212169795, 0.13537874899511282], [0.67308629425086, 0
.7395639529423029], [0.9739876256570963, 0.2266012027038959], [0.960780544458333
, 0.27730983644715734], [0.9696945283534095, 0.24432053061799522], [0.9574645398
596203, 0.2885509572179678], [0.6973549598034537, 0.7167259309091053], [0.997501
2143215075, 0.07064932715261943], [0.9898943479864515, 0.14180683983672393], [0.
8858798293249888, 0.4639147852732532], [0.815031016285887, 0.5794173301619432],
[0.7391093431221455, 0.6735854651857852], [0.9551139801898226, 0.296238560700586
7], [0.8999040475480857, 0.43608795581461857], [0.8399898376135325, 0.5426021311
292384], [0.7380507810452066, 0.6747451701187349], [0.818132254308594, 0.5750300
987426118], [0.8380099508390431, 0.5456549480163675], [0.7480296030887282, 0.663
6653621388716], [0.7828025838220598, 0.6222701300572823], [0.8522631744260054, 0
.5231132587856172], [0.6968775520464895, 0.7171901264334948]]
```

4.3.2 github-minisom库的调用

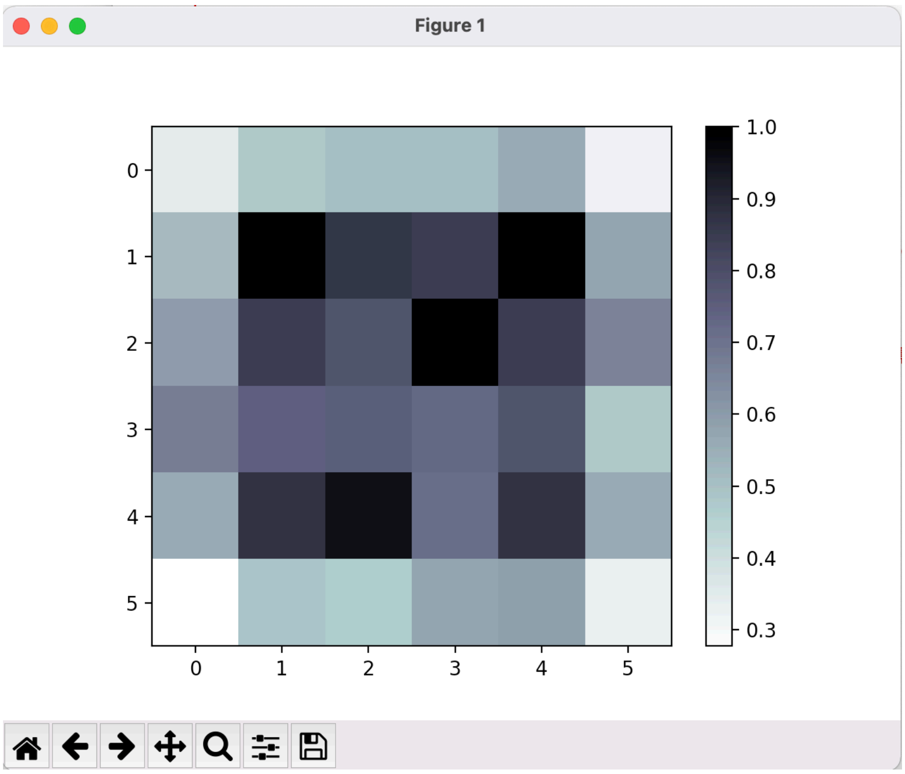
[minisom.py]

[test_minisom.py]

```
# install
pip3 install minisom
```

```python
# use
# import minisom
from minisom import MiniSom
import matplotlib.pyplot as plt
# input data
data = [[ 0.80,  0.55,  0.22,  0.03],
        [ 0.82,  0.50,  0.23,  0.03],
        [ 0.80,  0.54,  0.22,  0.03],
        [ 0.80,  0.53,  0.26,  0.03],
        [ 0.79,  0.56,  0.22,  0.03],
        [ 0.75,  0.60,  0.25,  0.03],
        [ 0.77,  0.59,  0.22,  0.03]]
# set parameters
som = MiniSom(6, 6, 4, sigma=0.3, learning_rate=0.5) # initialization of 6x6 SOM
# train
som.train(data, 100) # trains the SOM with 100 iterations
# draw U-Matrix
heatmap = som.distance_map()  #生成U-Matrix
plt.imshow(heatmap, cmap='bone_r')      #miniSom案例中用的pcolor函数,需要调整坐标
plt.colorbar()
plt.show()
# print
print(som.get_weights())
```

画U-Matrix图（用距离显示关联度）：



输出权值向量：

```
= RESTART: /Users/xinyuanhe/Desktop/working/2021美赛/模型/【正式】模型-机器学习-聚类-SOM聚类【hxy】/tes
t_minisom.py
[[[-0.08731956 -0.03937451 -0.92754531 -0.36122659]
  [-0.51380429  0.60111182  0.57331909  0.21441771]
  [-0.56180653 -0.59740279  0.56574392  0.0861229 ]
  [ 0.32044655 -0.48288396  0.57533139 -0.56392113]
  [ 0.79198174  0.55007185  0.23003397  0.03      ]
  [-0.60018344  0.56904868 -0.20301754 -0.50382567]]

 [[-0.65662469  0.01025707  0.32165003  0.68211441]
  [-0.47739548  0.52864132 -0.55323037 -0.43193527]
  [-0.25705387 -0.14509936  0.75752299  0.58226145]
  [ 0.6379476   0.31832619  0.60075603 -0.36162964]
  [-0.83425414 -0.4177136  -0.29784177  0.08716986]
  [-0.48607183  0.1357158   0.8569743  -0.10424233]]

 [[ 0.07067359 -0.35358921  0.92779935  0.09575116]
  [-0.37275594 -0.61686561 -0.53862755  0.43636016]
  [ 0.42299316 -0.66559279 -0.23051783  0.57002153]
  [ 0.24146536 -0.53481774 -0.58166535 -0.56332041]
  [ 0.4061066   0.07748638  0.75913043  0.50278652]
  [ 0.15259413  0.19974616 -0.81780427  0.51769941]]

 [[-0.28626006  0.5107969  -0.65312053 -0.48018255]
  [ 0.29161198 -0.88037114  0.30160659  0.22122968]
  [ 0.2927777  -0.32013065  0.7485585   0.50145564]
  [ 0.61730771 -0.4653399   0.60673286 -0.18510864]
  [ 0.42320204  0.56944044  0.61709782 -0.34033498]
  [ 0.02507307  0.86782278  0.30062063 -0.39481919]]

 [[ 0.76694272 -0.39951642 -0.39853108 -0.3055462 ]
  [-0.70796821 -0.6694303   0.21831143  0.05462792]
  [-0.51614665  0.58517195  0.18158092  0.59849377]
  [ 0.61251799 -0.3223832   0.61401195 -0.37931532]
  [ 0.27254296  0.84698196 -0.45144991  0.06734148]
  [-0.01375962  0.44060025 -0.42727016  0.78938096]]

 [[-0.39519211 -0.6191087   0.62337873 -0.26819128]
  [ 0.6933718  -0.25059698  0.61651009  0.27631868]
  [ 0.66748392  0.26487377  0.67701527  0.16111307]
  [-0.64447117  0.36064095  0.67180764 -0.05717954]
  [ 0.71067785 -0.65262965 -0.00660274  0.26261747]
  [ 0.03434904 -0.20930359 -0.87319605 -0.43879472]]]
```

5. 参考资料

1. SOM-github源码整理
2. github-minisom源码
3. SOM-可视化
4. 数模官网-SOM聚类
5. 机器学习实战——python实现SOM神经网络聚类算法（偏代码实现）
6. 机器学习实战——python实现DBSCAN密度聚类（用于2的数据处理参考）
7. SOM算法（偏数学理论）