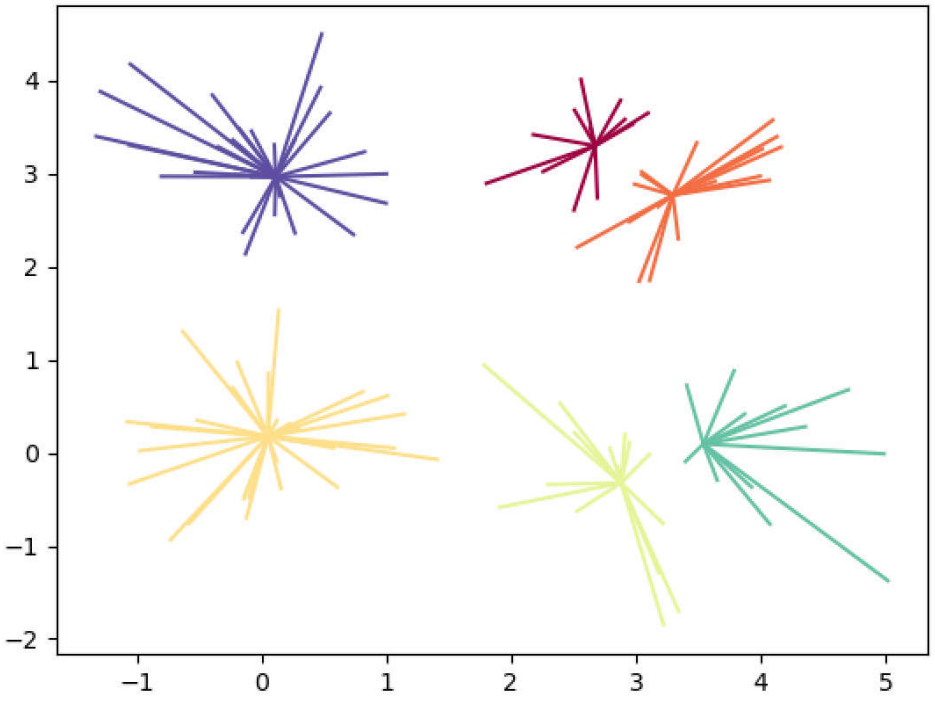
Python编程与人工智能实践



算法篇：

AP聚类算法
Affinity-Propagation

近邻传播聚类

于泓
鲁东大学
信息与电气工程学院

2022.6.30

AP（Affinity-Propagation）

AP聚类一般翻译为近邻传播聚类，07年被提出，其优点有：

- 1. 不需要制定最终聚类簇的个数
- 2. 已有的数据点作为最终的聚类中心，而不是新生成一个簇中心。
- 3. 模型对数据的初始值不敏感。
- 4. 对初始相似度矩阵数据的对称性没有要求。

几个概念：

相似度矩阵 S (**similarity**)： S(i，k)（**i !=k**）表示点i和点k的负距离，越大两点越相似（S矩阵可以非对阵）
S(k, k)（对角线元素）表示 点k适合做聚类中心的程度
初始化时，一般S(k，k)设置为 S矩阵中除对角线元素外的中值获最小值。
S(k，k) 设置越大，表示 k点越适合做聚类中心

责任矩阵 R (**responsibility**)： R(i，k) 表示点**k** 适合做点**i**的中心的程度

可用性矩阵 A (**Availability**)： A(i，k) 表示点**k** 适合做，**除了点i外**的**其他**点的聚类中心的程度

E=R+A E 中：**第i行最大值的位置为k，表示样本i的聚类中心为样本k**

参数更新过程

其他点适合做聚类中心的程度

$$r_{t+1}(i,k)=\begin{cases}S(i,k)-\max_{j\neq k}\left\{a_t(i,j)+r_t(i,j)\right\},i\neq k\\ S(i,k)-\max_{j\neq k}\{S(i,j)\},i=k\end{cases}$$

$$a_{t+1}(i,k)=\begin{cases}\min\left\{0,r_{t+1}(k,k)+\sum_{j\neq i,k}\max\left\{r_{t+1}(j,k),0\right\}\right\},i\neq k\\ \sum_{j\neq k}\max\left\{r_{t+1}(j,k),0\right\},i=k\end{cases}$$

更新权重

$$r_{t+1}(i,k)=\lambda*r_t(i,k)+(1-\lambda)*r_{t+1}(i,k)$$
$$a_{t+1}(i,k)=\lambda*a_t(i,k)+(1-\lambda)*a_{t+1}(i,k)$$

具体训练过程

（1） 初始化 S，R，A（R,A为全0矩阵）

（2.1）E_old = R+A

（2.2） 更新R

（2.3） 更新A

（2.4） E_new = R+A
比较E_old，E_new 查看是否有变化
比较E_old，E_new 每行最大值，查看是否有变化
长期无变化：退出
否则 跳转到（2.1）

（3）E=R+A
计算每行最大值的位置，实现聚类

S,R,A物理意义的解释

将聚类过程看成选举：所有人都参加选举（大家都是选民也都是参选人），要选出几个作为代表

s(i,k)就相当于i对选k这个人的一个固有的偏好程度

r(i,k)表示用s(i,k)减去最强竞争者的评分，可以理解为k在对i这个选民的竞争中的优势程度r(i,k)的更新过程对应选民i的各个参选人的挑选（越出众越有吸引力）

a(i,k)：从公式里可以看到，所有r(i‘,k)>0的值都对a有正的加成。对应到我们这个比喻中，就相当于选民i通过网上关于k的民意调查看到：有很多人（即i’们）都觉得k不错（r(i‘,k)>0），那么选民i也就会相应地觉得k不错，是个可以相信的选择。a(i,k)的更新过程对应关于参选人k的民意调查对于选民i的影响（已经有了很多跟随者的人更有吸引力）

两者交替的过程也就可以理解为选民在各个参选人之间不断地比较和不断地参考各个参选人给出的民意调查。r(i,k)的思想反映的是竞争，a(i,k)则是为了让聚类更成功。

作者：kael链接：https://www.zhihu.com/question/25384514/answer/47636054来源：知乎著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

代码实现：

```
def compute_R(S,R,A,dampfac):

    to_max = A+R
    N = np.shape(to_max)[0]

    max_AS = np.zeros_like(S)
    for i in range(N):
        for k in range(N):
            if not i ==k:
                temp = to_max[i,:].copy()
                temp[k] = -np.inf
                max_AS[i,k] = max(temp)
            else:
                temp = S[i,:].copy()
                temp[k] = -np.inf
                max_AS[i,k] = max(temp)


    return (1-dampfac) * (S - max_AS) + dampfac * R
```

$$r_{t+1}(i,k) = \begin{cases} S(i,k) - \max_{j \neq k} \{a_t(i,j) + r_t(i,j)\}, i \neq k \\ S(i,k) - \max_{j \neq k} \{S(i,j)\}, i = k \end{cases}$$

$$r_{t+1}(i,k) = \lambda * r_t(i,k) + (1 - \widehat{\lambda}) * r_{t+1}(i,k)$$

```python
def compute_A(R,A,dampfac):
    max_R = np.zeros_like(R)
    N = np.shape(max_R)[0]

    for i in range(N):
        for k in range(N):
            max_R[i,k] = np.max([0,R[i,k]])

    min_A = np.zeros_like(A)

    for i in range(N):
        for k in range(N):
            if not i == k:
                temp = max_R[:,k].copy()
                temp[i] =0
                min_A[i,k] = np.min([0,R[k,k]+np.sum(temp)])

            else:
                temp = max_R[:,k].copy()
                temp[k] =0
                min_A[i,k] = np.sum(temp)

    return (1-dampfac)*min_A + dampfac*A
```

$$a_{t+1}(i,k) = \begin{cases} \min\left\{0, r_{t+1}(k,k) + \sum_{j\neq i,k} \max\left\{r_{t+1}(j,k),0\right\}\right\}, i \neq k \\ \sum_{j\neq k} \max\left\{r_{t+1}(j,k),0\right\}, i = k \end{cases}$$

$$a_{t+1}(i,k) = \lambda * a_t(i,k) + (1-\lambda) * a_{t+1}(i,k)$$

S矩阵初始化 → 对角线元素初始化方法

```python
def compute_S_init(datas,preference="median"):
    N,D = np.shape(datas)
    tile_x = np.tile(np.expand_dims(datas,1),[1,N,1]) # N, N,D
    tile_y = np.tile(np.expand_dims(datas,0),[N,1,1]) # N, N,D
    S = -np.sum((tile_x-tile_y)*(tile_x-tile_y),axis=-1)
    indices = np.where(~np.eye(S.shape[0],dtype=bool))

    if preference == "median":
        m = np.median(S[indices])
    elif preference == "min":
        m = np.min(S[indices])
    elif type(preference) == np.ndarray:
        m = preference

    np.fill_diagonal(S, m)
    return S
```

聚类过程

```
def affinity_prop(datas,maxiter=100,preference='median',dampfac =0.7,display=False):
    # 判断更新前后 R+A是否有显著变化
    message_thresh = 1e-5

    # 判断聚类结果是否多轮不变
    local_thresh = 10

    # 计算S
    S= compute_S_init(datas,preference)

    # A 和 R 的初始化
    A = np.zeros_like(S)
    R = np.zeros_like(S)

    # 加上较小的值防止震荡
    S = S+1e-12*np.random.normal(size=A.shape) * (np.max(S)-np.min(S))

    count_equal = 0
    i = 0
    converged = False
```

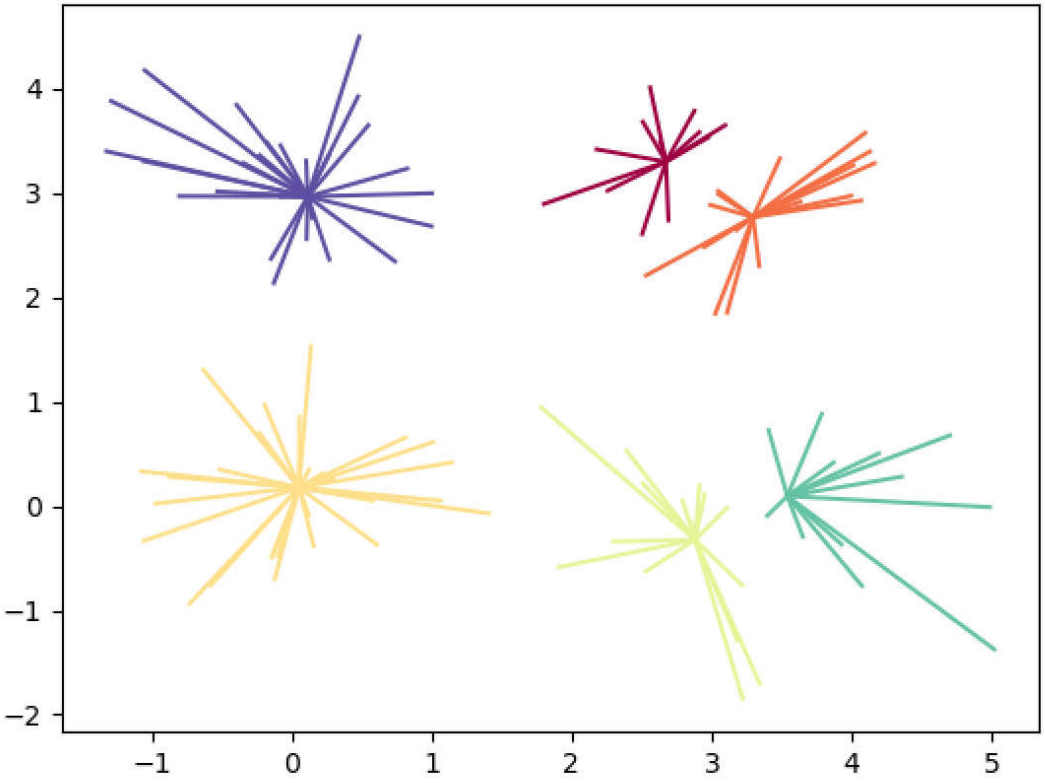
加上随机数，避免出现相同的数值

```
while i<maxiter:
    print(i)
    E_old = R+A
    labels_old = np.argmax(E_old, axis=1)

    R = compute_R(S,R,A,dampfac)
    A = compute_A(R,A,dampfac)

    E_new = R+A

    labels_cur = np.argmax(E_new, axis=1)

    # 判断更新前后 label是否一致
    if np.all(labels_cur == labels_old):
            count_equal += 1
    else:
            count_equal = 0

    if (message_thresh != 0 and np.allclose(E_old, E_new, atol=message_thresh)) or\
            (local_thresh != 0 and count_equal > local_thresh):
            converged = True
            break
    i = i+1

if converged:
    print("%d 轮后收敛."%(i))
else:
    print("%d 轮后迭代结束"%(maxiter))


E = R+A # Pseudomarginals
labels = np.argmax(E, axis=1)
exemplars = np.unique(labels)
centers = datas[exemplars]

return labels,exemplars,centers
```

迭代过程

```python
def cplot(datas,labels,str_title=""):
    plt.cla()
    index_center = np.unique(labels).tolist()

    colors={}
    for i,each in zip(index_center,np.linspace(0, 1,len(index_center))):
        colors[i]=plt.cm.Spectral(each)

    N,D = np.shape(datas)
    for i in range(N):
        i_center = labels[i]
        center = datas[i_center]
        data = datas[i]

        color = colors[i_center]
        plt.plot([center[0],data[0]],[center[1],data[1]],color=color)
    plt.title(str_title)
```

```python
if __name__ == "__main__":

    a = np.random.multivariate_normal([3,3], [[.4,0],[0,.4]],30)
    b = np.random.multivariate_normal([0,0], [[0.4,0],[0,0.4]], 30)
    c = np.random.multivariate_normal([3,0], [[0.4,0],[0,0.4]], 30)
    d = np.random.multivariate_normal([0,3], [[0.4,0],[0,0.4]], 30)
    data = np.r_[a,b,c,d]
    labels,exemplars,centers = affinity_prop(data,dampfac=0.7,preference='median',display=True)
    print(labels)

    cplot(data,labels)
    plt.show()
```



聚类效果