机器学习实验 12. 聚类（三）近邻传播

近邻传播聚类算法(Affinity Propagation)原理概述

Affinity Propagation (AP) 聚类是2007年在Science杂志上提出的一种新的聚类算法。它根据 N 个数据点之间的相似度进行聚类，这些相似度可以是对称的，即两个数据点互相之间的相似度一样(如欧氏距离)；也可以是不对称的，即两个数据点互相之间的相似度不等。这些相似度组成 $n \times n$ 的相似度矩阵 S(其中 n 为有 n 个数据点)。

AP 算法不需要事先指定聚类数目,相反它将所有的数据点都作为潜在的聚类中心，称之为 exemplar。以 S 矩阵的对角线上的数值 $S(k, k)$作为 k 点能否成为聚类中心的评判标准，这意味着该值越大，这个点成为聚类中心的可能性也就越大，这个值又称作参考度 p(preference)。聚类的数量受到参考度 p 的影响，如果认为每个数据点都有可能作为聚类中心，那么 p 就应取相同的值。如果取输入的相似度的均值作为 p 的值,得到聚类数量是中等的。如果取最小值，得到类数较少的聚类。

AP 算法中传递两种类型的消息，Responsibility 和 Availability。$R(i,k)$表示从点 i 发送到候选聚类中心 k 的数值消息，反映 k 点是否适合作为 i 点的聚类中心。$A(i,k)$则从候选聚类中心 k 发送到 i 的数值消息，反映 i 点是否选择 k 作为其聚类中心。$R(i,k)$与 $A(i,k)$越强，则 k 点作为聚类中心的可能性就越大，并且 i 点隶属于以 k 点为聚类中心的聚类的可能性也越大。AP 算法通过迭代过程不断更新每一个点的吸引度和归属度值,直到产生 m 个高质量的 exemplar，同时将其余的数据点分配到相应的聚类中。

AP 聚类算法是将每个数据看成图中的一个节点，迭代的过程即是在图中通过传播信息来找到聚类集合。本文计算两个数据点的相似度采用距离的负数，也就是说距离越近，相似度越大。相似矩阵 S 中 i 到 j 的相似度就是刚刚所说的距

离的负数。但是主对角线上的那些数表示的是某个点和自身的相似度，但是这里我们不能直接用 0 来表示。根据算法要求，主对角线上的值 $S(k,k)$一般称为偏向参数，一般情况下对所有 k，$S(k,k)$都相等，取非主对角线上的所有数的中位数。这个值很重要，其大小与最后得到的类数目有关，一般而言这个数越大，得到的类的数目就越多。

值得注意的是，在许多机器学习文献中，也将近邻传播作为**半监督学习**，即去除前一阶段的聚类中心选取阶段，而直接代之以部分少量已知标签的训练样本，并利用这些样本完成标签传播，从而实现对未知样本的标签预测。聚类中心选取阶段本身也可以作为一项重要的机器学习任务，即**主动学习**。

AP 聚类算法描述

1）设置实验数据；

2）计算相似度矩阵，并且设置参考度，这里使用相似度矩阵的中值；

3）计算吸引度矩阵，即 R 值；

4）计算归属度矩阵，即 A 值；

5）迭代更新 R 值和 A 值；

6）根据求出的聚类中心，对数据进行分类

AP 聚类算法在模拟数据中的一般实现

1）设置实验数据；

```
def init_sample():
    ## 生成的测试数据的中心点
    centers = [[1, 1], [-1, -1], [1, -1]]
    ##生成数据
    Xn, labels_true = make_blobs(n_samples=150, centers=centers, cluster_std=0.5,
                                 random_state=0)
    #3数据的长度，即：数据点的个数
    dataLen = len(Xn)

    return Xn,dataLen
```

我们使用 sklearn 包中提供的函数，随机生成以[1, 1], [-1, -1], [1, -1]三个点为

中心的 150 个数据。

2）计算相似度矩阵，并且设置参考度，这里使用相似度矩阵的中值；

```
def cal_simi(Xn):
    ##这个数据集的相似度矩阵，最终是二维数组
    simi = []
    for m in Xn:
        ##每个数字与所有数字的相似度列表，即矩阵中的一行
        temp = []
        for n in Xn:
            ##采用负的欧式距离计算相似度
            s =-np.sqrt((m[0]-n[0])**2 + (m[1]-n[1])**2)
            temp.append(s)
        simi.append(temp)

    ##设置参考度，即对角线的值，一般为最小值或者中值
    #p = np.min(simi)   ##11个中心
    #p = np.max(simi)  ##14个中心
    p = np.median(simi)  ##5个中心
    for i in range(dataLen):
        simi[i][i] = p
    return simi
```

3）计算吸引度矩阵，即 R 值；

```
##初始化R矩阵、A矩阵
def init_R(dataLen):
    R = [[0]*dataLen for j in range(dataLen)]
    return R
def init_A(dataLen):
    A = [[0]*dataLen for j in range(dataLen)]
    return A
##迭代更新R矩阵
def iter_update_R(dataLen,R,A,simi):
    old_r = 0 ##更新前的某个r值
    lam = 0.5 ##阻尼系数,用于算法收敛
    ##此循环更新R矩阵
    for i in range(dataLen):
        for k in range(dataLen):
            old_r = R[i][k]
            if i != k:
                max1 = A[i][0] + R[i][0]  ##注意初始值的设置
                for j in range(dataLen):
                    if j != k:
                        if A[i][j] + R[i][j] > max1 :
                            max1 = A[i][j] + R[i][j]
                ##更新后的R[i][k]值
                R[i][k] = simi[i][k] - max1
                ##带入阻尼系数重新更新
                R[i][k] = (1-lam)*R[i][k] +lam*old_r
            else:
                max2 = simi[i][0] ##注意初始值的设置
                for j in range(dataLen):
                    if j != k:
                        if simi[i][j] > max2:
                            max2 = simi[i][j]
                ##更新后的R[i][k]值
                R[i][k] = simi[i][k] - max2
                ##带入阻尼系数重新更新
                R[i][k] = (1-lam)*R[i][k] +lam*old_r
    print("max_r:"+str(np.max(R)))
    #print(np.min(R))
    return R
```

4）计算归属度矩阵，即 A 值；

```
##迭代更新A矩阵
def iter_update_A(dataLen,R,A):
    old_a = 0 ##更新前的某个a值
    lam = 0.5 ##阻尼系数,用于算法收敛
    ##此循环更新A矩阵
    for i in range(dataLen):
        for k in range(dataLen):
            old_a = A[i][k]
            if i ==k :
                max3 = R[0][k] ##注意初始值的设置
                for j in range(dataLen):
                    if j != k:
                        if R[j][k] > 0:
                            max3 += R[j][k]
                        else :
                            max3 += 0
                A[i][k] = max3
                ##带入阻尼系数更新A值
                A[i][k] = (1-lam)*A[i][k] +lam*old_a
            else :
                max4 = R[0][k] ##注意初始值的设置
                for j in range(dataLen):
                    ##上图公式中的i!=k 的求和部分
                    if j != k and j != i:
                        if R[j][k] > 0:
                            max4 += R[j][k]
                        else :
                            max4 += 0
                ##上图公式中的min部分
                if R[k][k] + max4 > 0:
                    A[i][k] = 0
                else :
                    A[i][k] = R[k][k] + max4
                ##带入阻尼系数更新A值
                A[i][k] = (1-lam)*A[i][k] +lam*old_a
    print("max_a:"+str(np.max(A)))
    #print(np.min(A))
    return A
```

5）迭代更新 R 值和 A 值。终止条件是聚类中心在一定程度上不再更新或者达到最大迭代次数；

```
def cal_cls_center(dataLen,simi,R,A):
    max_iter = 100    ##最大迭代次数
    curr_iter = 0     ##当前迭代次数
    max_comp = 30     ##最大比较次数
    curr_comp = 0     ##当前比较次数
    class_cen = []    ##聚类中心列表，存储的是数据点在Xn中的索引
    while True:
        R = iter_update_R(dataLen,R,A,simi)
        A = iter_update_A(dataLen,R,A)
        for k in range(dataLen):
            if R[k][k] +A[k][k] > 0:
                if k not in class_cen:
                    class_cen.append(k)
                else:
                    curr_comp += 1
        curr_iter += 1
        print(curr_iter)
        if curr_iter >= max_iter or curr_comp > max_comp :
            break
    return class_cen
```

6）根据求出的聚类中心，对数据进行分类；

```
if __name__=='__main__':
    Xn,dataLen = init_sample()
    R = init_R(dataLen)
    A = init_A(dataLen)
    simi = cal_simi(Xn)
    class_cen = cal_cls_center(dataLen,simi,R,A)
    c_list = []
    for m in Xn:
        temp = []
        for j in class_cen:
            n = Xn[j]
            d = -np.sqrt((m[0]-n[0])**2 + (m[1]-n[1])**2)
            temp.append(d)
        c = class_cen[temp.index(np.max(temp))]
        c_list.append(c)
    colors = ['red','blue','black','green','yellow']
    plt.figure(figsize=(8,6))
    plt.xlim([-3,3])
    plt.ylim([-3,3])
    for i in range(dataLen):
        d1 = Xn[i]
        d2 = Xn[c_list[i]]
        c = class_cen.index(c_list[i])
        plt.plot([d2[0],d1[0]],[d2[1],d1[1]],color=colors[c],linewidth=1)
    plt.show()
```

这个步骤产生的是一个归类列表，列表中的每个数字对应着样本数据中对应位置的数据的分类。

从图中可以清楚地看出 AP 聚类算法的原理。