云南大学软件学院
实　验　报　告

姓名：王子陈 学号：20171050008 专业：电子科学与技术 日期：2019/11/6 成绩：_____
任课教师：谢仲文

数据挖掘技术　实验四

一、实验目的

1. 掌握聚类的概念，熟练运用 K 均值算法进行聚类分析。
2. 选择一种编程语言实现 K 均值算法。

二、实验内容

1. 在一个简单的、虚拟的数据集（训练集）上应用划分方法进行聚类分析。该数据集如下：

国家	2019年国际排名	2018世界杯	2015亚洲杯
中国	73	40	7
日本	60	15	5
韩国	61	19	2
伊朗	34	18	6
沙特	67	26	10
伊拉克	91	40	4
卡塔尔	101	40	13
阿联酋	81	40	6
乌兹别克斯坦	88	40	8
泰国	122	40	17
越南	102	50	17
阿曼	87	50	12
巴林	116	50	11
朝鲜	110	50	14
印尼	164	50	17
澳洲	40	30	1
叙利亚	76	40	17
约旦	118	50	9
科威特	160	50	15
巴勒斯坦	96	50	16

2. 基本要求：使用 K 均值算法。（必做）
3. 提高要求：使用 K 中心点算法。（选做）
4、比较和讨论 K 均值算法与 K 中心点算法的优劣。

三、实验要求

1. 完成实验内容，源码作为实验报告附件一起打为一个压缩包提供。该压缩包要包含实验报告、代码文件。
2. 关键部分要求有注释，注释量不低于 20%

3. 要求独立完成，不得抄袭代码。

四、关键实验步骤（请粘贴关键步骤、代码、实验结果）

```
#data_list 包含所有的点,每一个点的形式为:[0, '中国', 73, 40, 7]: 第一位是序号,后三位是 3 个排名
#cluster_centers 包含 3 个质心,即簇中心的数目 k=3

def k_means(data_list,cluster_centers):
    flag=1    #标志位用来判断目标函数 E 是否发生变化
    while (flag) :
        cluster0=[] #存放簇 0
        cluster1=[] #存放簇 1
        cluster2=[] #存放簇 2
        total_clusters=[cluster0,cluster1,cluster2] #包含 3 个簇
        distance=[] #存放距离

        #每个国家都要计算到 3 个中心点的距离
        for country in data_list:
            for center in cluster_centers:
                sum_sqrt_error=0
                for i in range(2,5): #计算到某一质心的距离
                    sum_sqrt_error+=pow(country[i]-center[i-2],2)
                distance.append(math.sqrt(sum_sqrt_error))#存储了 3 个距离
            cluster_num=distance.index(min(distance)) #根据序号找最小距离对应的质心
            distance.clear()
            if cluster_num==0:      #如果序号=0，就归到簇 0
                cluster0.append(country)
            elif cluster_num==1:    #如果序号=1，就归到簇 1
                cluster1.append(country)
            else:                   #如果序号=2，就归到簇 2
                cluster2.append(country)

        #计算目标函数 E：
        #簇内的点与簇质心的欧几里得距离
        E_prim=0
        for cluster_idx in range(0,3):
            for country in total_clusters[cluster_idx]:
                for idx in range(2,5):
                    E_prim+=pow(country[idx]-cluster_centers[cluster_idx][idx-2],2)

        #重新计算 3 个簇的质心(均值)
        cen0=[] #质心 0   (3 个坐标)
```

```
        cen1=[] #质心 1    (3 个坐标)
        cen2=[] #质心 2    (3 个坐标)
        centers=[cen1,cen2,cen3]    #包含 3 个质心

        for cluster_idx in range(0,3): #遍历 3 个簇
            for idx in range(2,5):    #x,y,z,3 个坐标分别求平均值
                sum_idx=0
                for country in total_clusters[cluster_idx]:
                    sum_idx+=country[idx]
                centers[cluster_idx].append(sum_idx/len(total_clusters[cluster_idx]))
        cluster_centers=centers #更新 3 个簇的质心

        #重新计算目标函数，看是否发生改变
        E_new=0
        for cluster_idx in range(0,3): #分开计算 3 个簇
            for country in total_clusters[cluster_idx]:
                for idx in range(2,5):
                    E_new+=pow(country[idx]-cluster_centers[cluster_idx][idx-2],2)

        #更新质点后,如果目标函数没有变化,就停止循环
        if(E_prim==E_new):
            flag=0
        else:
            return(E_new,cluster0,cluster1,cluster2)
```

此函数可以实现: 给定初始簇中心点的聚类结果, 若初始簇中心点不同,则聚类结果不同, 为了找到最佳的聚类结果可以把所有的初始簇中心点代入,比较目标函数 E, 取最小的情况. C_{20}^{3}

```
#找出 20 个数所有 3 个数的组合
combine=[0,0,0] #初始化
copy_combine=[] #拷贝
total_comb=[]     #包含所有 3 点组合
for i in range(18):
    combine[0]=i
    for j in range(i+1,19):
        combine[1]=j
        for k in range(j+1,20):
            combine[2]=k
            copy_combine=copy.copy(combine)
            total_comb.append(copy_combine)

times=0 #次数
for comb_idx in total_comb: #遍历每一种初始中心点组合
```

```
cluster_centers=data_list[comb_idx[0]][2:5],data_list[comb_idx[1]][2:5],data_list[co
mb_idx[2]][2:5]
    result=k_means(data_list,cluster_centers) #计算聚类结果
    if times==0: #保留第一次得到的 E 和质心作为最佳方案,后面与之做比较
        min_E=result[0]
        best_clusters=[result[1],result[2],result[3]]
        times+=1
    if result[0]<min_E: #选出最小的 E
        min_E=result[0]
        best_clusters=[result[1],result[2],result[3]]
```

最小目标函数为: 4761.25

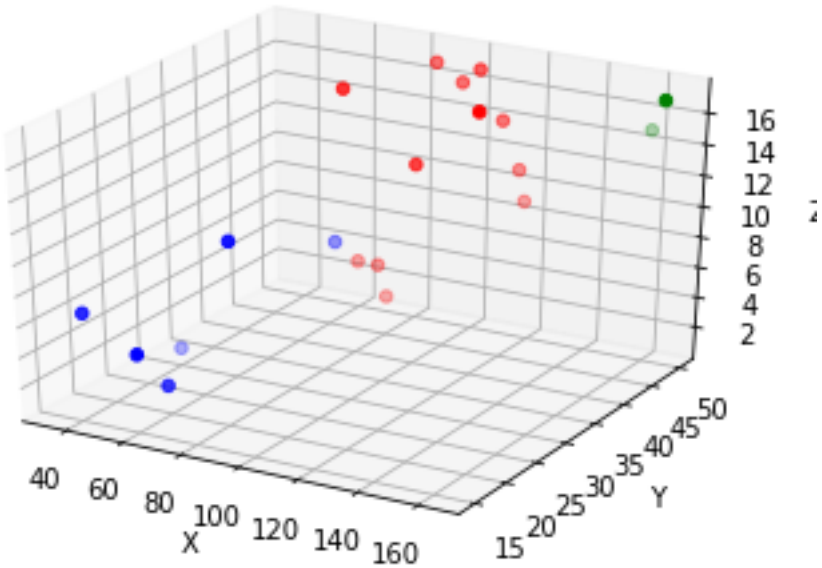
最终得到的 3 个簇为:

[[0, '中国', 73, 40, 7], [1, '日本', 60, 15, 5], [2, '韩国', 61, 19, 2], [3, '伊朗', 34, 18, 6], [4, '沙特', 67, 26, 10], [15, '澳洲', 40, 30, 1]]

[[5, '伊拉克', 91, 40, 4], [6, '卡塔尔', 101, 40, 13], [7, '阿联酋', 81, 40, 6], [8, '乌兹别克斯坦', 88, 40, 6], [9, '泰国', 122, 40, 17], [10, '越南', 102, 50, 17], [11, '阿曼', 87, 50, 17], [12, '巴林', 116, 50, 11], [13, '朝鲜', 110, 50, 14], [16, '叙利亚', 76, 40, 17], [17, '约旦', 118, 50, 9], [19, '巴勒斯坦', 96, 50, 16]]

[[14, '印尼', 164, 50, 17], [18, '科威特', 160, 50, 15]]

画出散点图观察:



K-means 算法每次计算的质心为簇内点平均值, 质心不在一个具体的点上, 结果可能会是把不应该在某簇里的点”拉扯”过来了, 所以在 K-means 算法里, 极端值会扭曲数据分布, k-means 对离群点敏感.

K 中心点算法:

```
#data_list 包含所有的点,每一个元素形式为:[0, '中国', 73, 40, 7],第一位是序号,后三位是 3 个排名
#cluster_centers 包含 3 个代表对象
def k_medoids(data_list,cluster_centers):
    flag=1    #标志位用于判断和上次的目标函数是否相同
    while (flag) :
        cluster0=[] #存放 1 簇
        cluster1=[] #存放 2 簇
        cluster2=[] #存放 3 簇
        total_clusters=[cluster0,cluster1,cluster2] #包含 3 个簇
        distance=[]   #存放距离
        #每个国家都要计算到 3 个代表对象的距离
        for country in data_list:
            for center in cluster_centers:
                sum_sqrt_error=0
                for i in range(2,5): #计算曼哈顿距离
                    #曼哈顿距离取消了欧式距离的平方,因此使得离群点的影响减弱。
                    sum_sqrt_error+=abs(country[i]-center[i])
                distance.append(sum_sqrt_error)#存储了 3 个距离
            cluster_num=distance.index(min(distance))#找最小距离对应点序号
            distance.clear()
            if cluster_num==0: #序号为 0,就归到第一个簇
                cluster0.append(country)
            elif cluster_num==1:
                cluster1.append(country)
            else:
                cluster2.append(country)

        #计算目标函数 E：
        #簇内的点到簇质心的曼哈顿距离的和
        E_prim=0
        for cluster_idx in range(0,3):
            for country in total_clusters[cluster_idx]:
                for idx in range(2,5):
                    E_prim+=abs(country[idx]-cluster_centers[cluster_idx][idx])

        #每个簇里重新选老大(遍历除了 c1,c2,c3 点外的所有点分别替代 c1,c2,c3)
        #先把中心点排除,得到子集
        copy_data_list=copy.copy(data_list)
        for n in range(3):
            copy_data_list.pop(cluster_centers[n][0])   #删除中心点
```

```
        copy_cluster_centers=copy.copy(cluster_centers)#用拷贝,否则元素会一起
变化
        #把所有非中心点拿来分别替换 c1,c2,c3,计算目标函数,取最小的作为新
老大
        E_min=E_prim
        for center_idx in range(3): #每次只换一个中心点
            for country in copy_data_list:
                copy_cluster_centers[center_idx]=country #替换
                #以新的 3 个中心点聚类
                clust1=[] #簇 1
                clust2=[] #簇 2
                clust3=[] #簇 3
                all_clusters=[clust1,clust2,clust3]   #包含 3 个簇

                #每个国家都要计算到 3 个代表对象的距离
                for country in copy_data_list:
                    for center in copy_cluster_centers:
                        sum_sqrt_error=0
                        for i in range(2,5): #计算曼哈顿距离
                            #曼哈顿距离取消了欧式距离的平方，因此使得
离群点的影响减弱。
                            sum_sqrt_error+=abs(country[i]-center[i])
                        distance.append(sum_sqrt_error)#存储了 3 个距离
                    cluster_num=distance.index(min(distance))  #找最小距离对
应的点的序号
                    distance.clear() #清空距离
                    if cluster_num==0:     #序号为 0,就归到第一个簇
                        clust1.append(country)
                    elif cluster_num==1:   #序号为 1,就归到第二簇
                        clust2.append(country)
                    else:                  #序号为 2,就归到第三簇
                        clust3.append(country)

                #重新计算目标函数，看是否发生改变
                E_new=0
                for cluster_idx in range(0,3): #分开计算 3 个簇
                    for country in all_clusters[cluster_idx]:
                        for idx in range(2,5):
                            E_new+=abs(country[idx]-cluster_centers[cluster_idx][idx])

                #如果这次替换使目标函数变小了，就更新代表对象
                if(E_new<E_min):
                    E_min=E_new
                    cluster_centers=copy_cluster_centers
```

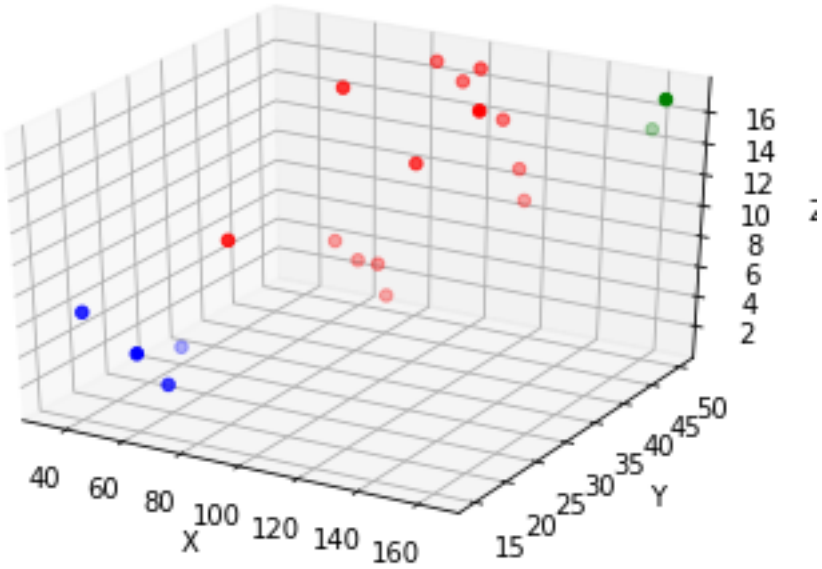
```
        #判断如果把c1,c2,c3都替换了一边之后,目标函数没有变化,就停止循环
        if(E_prim==E_new):
            flag=0
        else:
            return(E_new,cluster0,cluster1,cluster2)
```

得到最小的目标函数: 702
最终得到的 3 个簇:
[[1, '日本', 60, 15, 5], [2, '韩国', 61, 19, 2], [3, '伊朗', 34, 18, 6], [15, '澳洲', 40, 30,]],

[[0, '中国', 73, 40, 7], [4, '沙特', 67, 26, 10], [5, '伊拉克', 91, 40, 4], [6, '卡塔尔', 101, 40, 13], [7, '阿联酋', 81, 40, 6], [8, '乌兹别克斯坦', 88, 40, 6], [9, '泰国', 122, 40, 17], [10, '越南', 102, 50, 17], [11, '阿曼', 87, 50, 17], [12, '巴林', 116, 50, 11], [13, '朝鲜', 110, 50, 14], [16, '叙利亚', 76, 40, 17], [17, '约旦', 118, 50, 9], [19, '巴勒斯坦', 96, 50, 16]],

[[14, '印尼', 164, 50, 17], [18, '科威特', 160, 50, 15]])

画出散点图观察:



可以看到相比 K-means,这次聚类的簇间距离更大, 没有交叠.

K-中心点算法是对 K-means 算法的优化, 簇中心不选择均值, 而是每次都以”代表对象”作为簇的中心, 把靠近”代表对象”的点归到一类, 反复地用非代表对象来替换代表对象, 找到具有最小目标函数的”老大”. 对于孤立点更鲁棒.
在 K-means 算法执行过程中，可以通过随机的方式选择初始质心，也只有初始时通过随机方式产生的质心才是实际需要聚簇集合的中心点，而后面通过不断迭代产生的新的质心很可能并不是在聚簇中的点。如果某些异常点距离质心相对较大时，很可能导致重新计算得到的质心偏离了聚簇的真实中心。