机器学习实验报告2

kmeans random

设置分类数量，类中心点的最大容许误差

```
def __init__(self, k = 2, tolerance = 0.0000001):
    self.k_ = k                         #类别数量
    self.tolerance_ = tolerance     #最大容许误差
```

初始化类中心，使用随机选取k个数据作为类中心点

```
def run(self, data):
    #初始化类中心，使用随机选取
    col = data.shape[1]
    self.center_ = {}
    for i in range(self.k_):
        index = random.randint(0, data.shape[0] - 1)
        self.center_[i] = data[index]
```

然后开始迭代计算类中心点，选择距离数据最小的类中心点作为该数据的类标，根据新的分类，计算新的类中心点，直到类中心点不发生变换

```
#迭代计算类中心
while 1:
    self.class_ = {}    #分类后的数据
    for i in range(self.k_):
        self.class_[i] = []
    #根据当前所获得的类中心，对数据进行分类
    for feature in data:
        distances = []
        #计算该数据对所有类中心的欧式距离
        for center in self.center_:
            distances.append(np.linalg.norm(feature[0 : col - 1] -
self.center_[center][0 : col - 1]))
        #选择距离最小的类中心作为当前数据的类
        classification = distances.index(min(distances))
        #将该数据划入类中
        self.class_[classification].append(feature)

    #记录当前的类中心点
    old_center = dict(self.center_)
    #根据当前的分类，计算新的类中心点
    for i in self.class_:
        self.center_[i] = np.average(self.class_[i], axis = 0)

    #判断当前的分类是否最优化
    optimized = True
    for center in self.center_:
        old = old_center[center][0 : col - 1]
        new = self.center_[center][0 : col - 1]
```

```
            #计算新中心点与旧中心点的误差，误差过大，则说明当前分类仍未最优化
            if np.sum((new - old) / old * 100.0) > self.tolerance_:
                optimized = False
        #如果当前分类为最优化，则停止迭代
        if optimized:
            break
```

kmeans分类后，给数据分配新的类标

```
#统计所有数据的原本类标和kmeans分类后的新类标
old_label = np.zeros(data.shape[0])
new_label = np.zeros(data.shape[0])
x = 0
for i in kmeans.class_:
    col = data.shape[1]
    distances = []
    center = kmeans.center_[i]

    #获取在该类中，离中心点距离最近的元素
    for feature in kmeans.class_[i]:
        distances.append(np.linalg.norm(feature[0 : col - 1] - center[0 : col -
1]))
    min_index = distances.index(min(distances))

    #将该元素所属的类别，作为该类所有元素的类别
    label = int(kmeans.class_[i][min_index][col - 1])
    for feature in kmeans.class_[i]:
        old_label[x] = int(feature[col - 1])
        new_label[x] = label
        x += 1
```

根据数据新旧类标，计算Normalized Mutual Information值

```
#计算Normalized Mutual Information
def NMI(A,B):
    total = len(A)
    A_ids = set(A)
    B_ids = set(B)
    MI = 0
    eps = 1.4e-45
    for idA in A_ids:
        for idB in B_ids:
            idAOccur = np.where(A == idA)
            idBOccur = np.where(B == idB)
            idABOccur = np.intersect1d(idAOccur, idBOccur)
            px = 1.0 * len(idAOccur[0]) / total
            py = 1.0 * len(idBOccur[0]) / total
            pxy = 1.0 * len(idABOccur) / total
            MI = MI + pxy * math.log(pxy / (px * py) + eps, 2)
    Hx = 0
    for idA in A_ids:
        idAOccurCount = 1.0 * len(np.where(A == idA)[0])
        Hx = Hx - (idAOccurCount / total) * math.log(idAOccurCount / total +
eps, 2)
    Hy = 0
    for idB in B_ids:
```

```
        idBOccurCount = 1.0 * len(np.where(B == idB)[0])
        Hy = Hy - (idBOccurCount / total) * math.log(idBOccurCount / total +
eps, 2)
    MIhat = 2.0 * MI / (Hx + Hy)
    return MIhat
```

计算kmeans的目标函数，即距离平方误差

```
#计算目标函数，平方误差
square_error = 0
for i in kmeans.class_:
    avg = np.average(kmeans.class_[i], axis = 0)
    for feature in kmeans.class_[i]:
        square_error += pow(np.linalg.norm(feature[0 : col - 1] - avg[0 : col -
1]), 2)
print("Square Error: ", square_error)
```

kmeans distance based

使用distance based来初始化类中心点，第一个类中心点随机选择，然后计算所有数据与当前所有类中心的最小距离，再从这些最小距离选出最大的那个，该数据作为下一个类中心点

```
def run(self, data):
    #初始化类中心，使用distance-based
    col = data.shape[1]
    self.center_ = {}
    #第一个类中心随机选择
    self.center_[0] = data[random.randint(0, data.shape[0] - 1)]
    #剩余的类中心根据distance-based
    for i in range(1, self.k_):
        max_dist = -10000000
        #遍历所有数据
        for j in range(0, data.shape[0]):
            #如果当前数据已经存在类中心中，则直接跳过
            flag = True
            for center in self.center_:
                if (data[j] == self.center_[center]).all():
                    flag = False
            if flag == False:
                continue

            #计算当前数据与当前存在的所有类中心的最小距离
            min_dist = 10000000
            for k in range(0, i):
                dist = np.linalg.norm(data[j][0 : col - 1] - self.center_[k][0 :
col - 1])
                #选择最小距离
                if dist < min_dist:
                    min_dist = dist

            #选择所有最小距离中最大的，作为新的类中心点，更新最大值
            if min_dist > max_dist:
                self.center_[i] = data[j]
                max_dist = min_dist
```

kmeans random distance

使用random distance来初始化类中心点，第一个类中心点随机选择，然后计算所有数据与当前所有类中心的最小距离，将所有最小距离取和得到sum，在0到sum区间随机选取一个数random，random不断减去每个数据的最小距离，当random小于等于0时，该数据作为下一个类中心点

```
def run(self, data):
    #初始化类中心，使用distance-based
    col = data.shape[1]
    self.center_ = {}
    #第一个类中心随机选择
    self.center_[0] = data[random.randint(0, data.shape[0] - 1)]
    #剩余的类中心根据random-distance
    for i in range(1, self.k_):
        #所有数据到当前存在的所有类中心的最小距离
        dist = {}
        #所有类中心的最小距离的和
        sum_dist = 0
        #遍历所有数据
        for j in range(0, data.shape[0]):
            dist[j] = 0
            #如果当前数据已经存在类中心中，则直接跳过
            flag = True
            for center in self.center_:
                if((data[j] == self.center_[center]).all()):
                    flag = False
            if flag == False:
                continue

            #计算当前数据与当前存在的所有类中心的最小距离
            min_dist = 10000000
            for k in range(0, i):
                tmp = np.linalg.norm(data[j][0 : col - 1] - self.center_[k][0 :
col - 1])
                #选择最小距离
                if tmp < min_dist:
                    min_dist = tmp
            dist[j] = min_dist
            sum_dist += dist[j]

        #从0到sum_dist中选取一个随机数
        Random = random.uniform(0, sum_dist)
        #Random不断减去每个数据的dist值
        for j in range(0, data.shape[0]):
            Random -= dist[j]
            #如果Random小于等于0，则该数据被选为下一个类中心点
            if Random <= 0:
                self.center_[i] = data[j]
                break
```

实验结果与分析

本次实验所选取的五个数据集，按顺序展示：

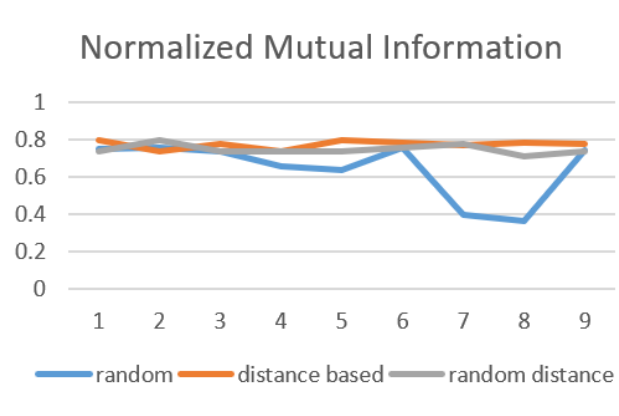
- https://archive.ics.uci.edu/ml/datasets/Iris 3类
- https://archive.ics.uci.edu/ml/datasets/banknote+authentication 2类
- https://archive.ics.uci.edu/ml/datasets/Wine 3类
- https://archive.ics.uci.edu/ml/datasets/Glass+Identification 7类
- http://archive.ics.uci.edu/ml/datasets/pen-based+recognition+of+handwritten+digits 10类

正确分类时的Normalized Mutual Information:

表格最后一行数据为平均值

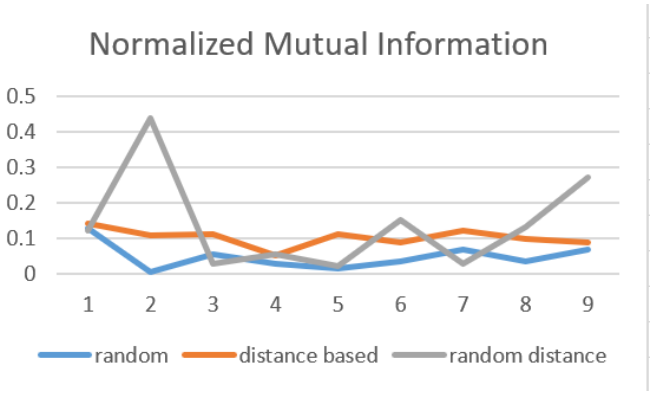
data1:

random	distance based	random distance
0.7507	0.7979	0.7419
0.7614	0.7419	0.7979
0.7419	0.7771	0.7419
0.6565	0.7419	0.7419
0.6398	0.7979	0.7419
0.7581	0.7837	0.7581
0.3978	0.7709	0.7771
0.3645	0.7837	0.7102
0.7479	0.7771	0.7419
0.64651	0.774677778	0.750311111



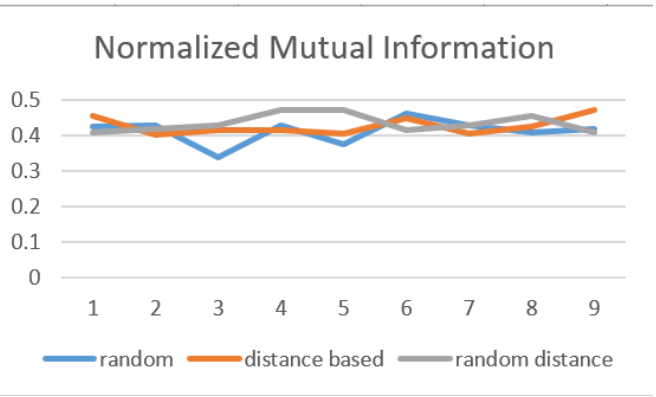
data2:

random	distance based	random distance
0.1276	0.1425	0.1217
0.0041	0.1077	0.4394
0.0565	0.1116	0.0299
0.0303	0.0516	0.0572
0.0161	0.1135	0.0227
0.0372	0.0877	0.1509
0.0679	0.1219	0.0303
0.0343	0.0988	0.1333
0.0706	0.089	0.2729
0.0494	0.1027	0.139811111



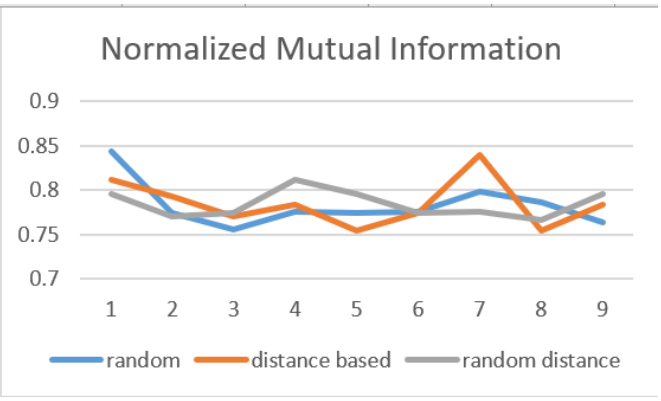
data3:

random	distance based	random distance
0.4263	0.4569	0.411
0.4287	0.4025	0.421
0.3381	0.4165	0.4287
0.4287	0.4165	0.4714
0.3772	0.4073	0.4714
0.4637	0.4497	0.4165
0.4287	0.4073	0.4287
0.4097	0.4259	0.4569
0.418	0.4714	0.411
0.41323	0.428222222	0.435177778



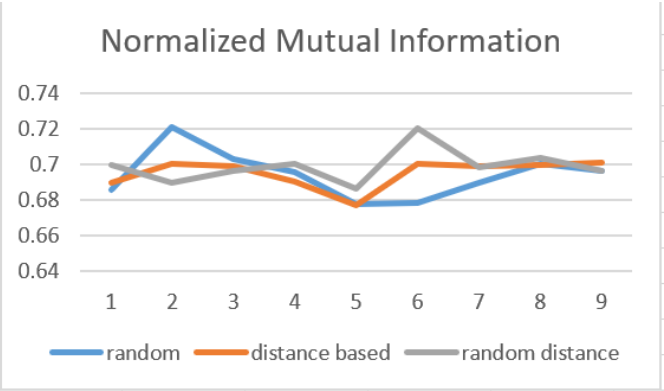
data4:

random	distance based	random distance
0.8443	0.8123	0.7961
0.7737	0.7931	0.7707
0.7554	0.7707	0.7737
0.7755	0.7833	0.8123
0.7737	0.7539	0.7957
0.7763	0.7737	0.7737
0.7983	0.8401	0.775
0.7861	0.7545	0.7664
0.7638	0.7833	0.7961
0.78301	0.784988889	0.784411111



data5:

random	distance based	random distance
0.686	0.6896	0.7
0.7209	0.7002	0.6896
0.7029	0.6989	0.6964
0.6961	0.6902	0.7007
0.6781	0.6774	0.6866
0.6787	0.7008	0.7208
0.6899	0.6989	0.6986
0.7007	0.6998	0.7039
0.6966	0.7013	0.6966
0.69443	0.695233333	0.699244444



分析:

从以上五个数据集得到的实验结果，可以看出random distance的准确度和distance based的准确度是十分相似的，这两种初始化方法得到的结果也十分稳定，也比random的准确度稍微高点，而且random初始化得到的结果十分不稳定，会出现较大的波动。

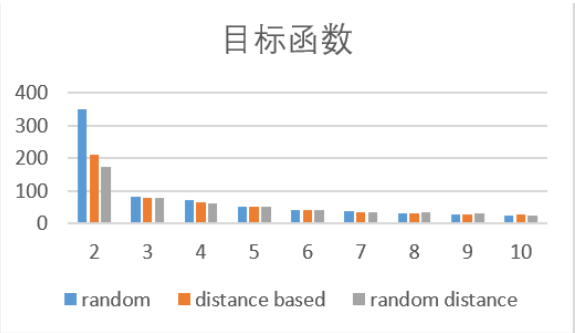
数据集1、4、5的nmi值都在0.6~0.8之间，分类效果较好，而数据集2和3的nmi值都比较低，这些数据集可能不适合kmeans分类。

不同分类个数的目标函数:

横坐标为分类数量，纵坐标为目标函数的值

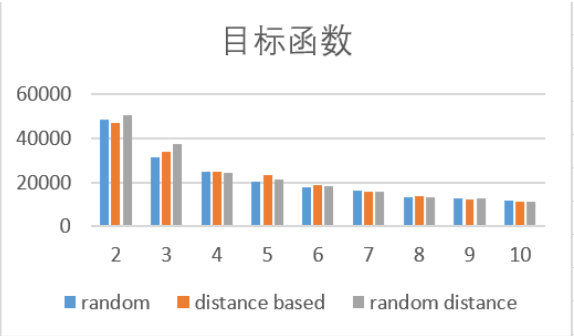
data1:

	random	distance based	random distance
2	351	211	175
3	82	79	78
4	71	65	62
5	50	50	50
6	42	42	43
7	37	35	36
8	32	32	34
9	28	29	32
10	26	27	26



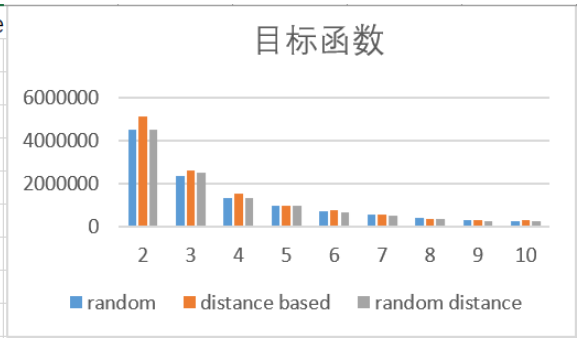
data2:

	random	distance based	random distance
2	48498	47323	50512
3	31402	33701	37618
4	24773	24772	24564
5	20482	23485	21095
6	17994	18879	18425
7	16123	15687	15807
8	13463	13661	13343
9	12567	12189	12577
10	11772	11002	11364



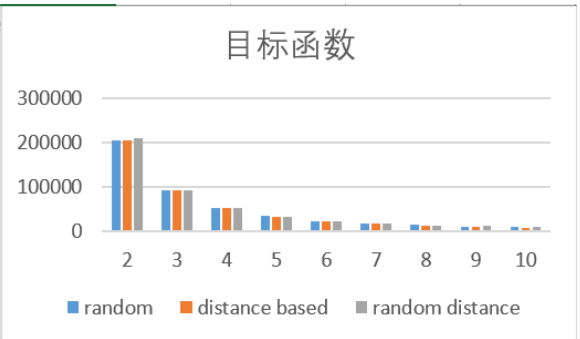
data3:

	random	distance based	random distance
2	4545746	5170560	4554666
3	2370689	2649229	2533951
4	1337437	1532985	1351749
5	988245	989238	990455
6	720459	753783	700035
7	598763	552905	531682
8	441742	385862	356788
9	301467	332473	280541
10	264255	291428	250085



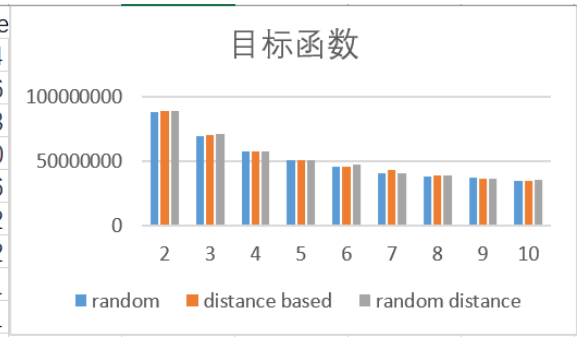
data4:

	random	distance based	random distance
2	205391	205349	209714
3	91939	91939	91879
4	52182	52182	52106
5	35731	33759	33830
6	23875	23731	23955
7	17888	18161	17740
8	14324	13767	13783
9	11413	11026	11943
10	9694	9155	9371



data5:

	random	distance based	random distance
2	88766185	89459102	89451244
3	70051789	70261848	71403116
4	58182341	58182317	58182303
5	51455224	51455405	51455100
6	45630825	45630825	47944146
7	41282781	43035659	41282702
8	38772488	38900265	39341622
9	37186078	37067601	36854711
10	34894764	35159358	36056131



分析：

从以上五个数据集得到的实验结果，可以看出随着分类数量的逐渐增大，目标函数会不断减少，当分类数量从2变为3时，减少的幅度最大，但分类数量从3开始，目标函数减少的幅度越来越少，符合理论上的情况。三种不同类型的初始化，在分类数量相同时，目标函数的值都十分相似。