菊安酱的机器学习第9期

菊安酱的直播间：https://live.bilibili.com/14988341

每周一晚8:00 菊安酱和你不见不散哦~(^o^)/~

更新日期：2018-12-28

作者：菊安酱

课件内容说明：

- 本文为作者原创，转载请注明作者和出处
- 如果想获得此课件及录播视频，可扫描左边二维码，回复"k"进群
- 如果想获得2小时完整版视频，可扫描右边二维码或点击如下链接
- 若有任何疑问，请给作者留言。

交流群二维码

完整版视频及课件

直播视频及课件：http://www.peixun.net/view/1278.html

完整版视频及课件：http://edu.cda.cn/course/966

12期完整版课纲

直播时间：每周一晚8:00

直播内容：

时间	期数	算法
2018/11/05	第1期	k-近邻算法
2018/11/12	第2期	决策树
2018/11/19	第3期	朴素贝叶斯
2018/11/26	第4期	Logistic回归
2018/12/03	第5期	支持向量机
2018/12/10	第6期	AdaBoost 算法
2018/12/17	第7期	线性回归
2018/12/24	第8期	树回归
2018/12/28	第9期	K-均值聚类算法
2019/01/07	第10期	Apriori 算法
2019/01/14	第11期	FP-growth 算法
2019/01/21	第12期	奇异值分解SVD

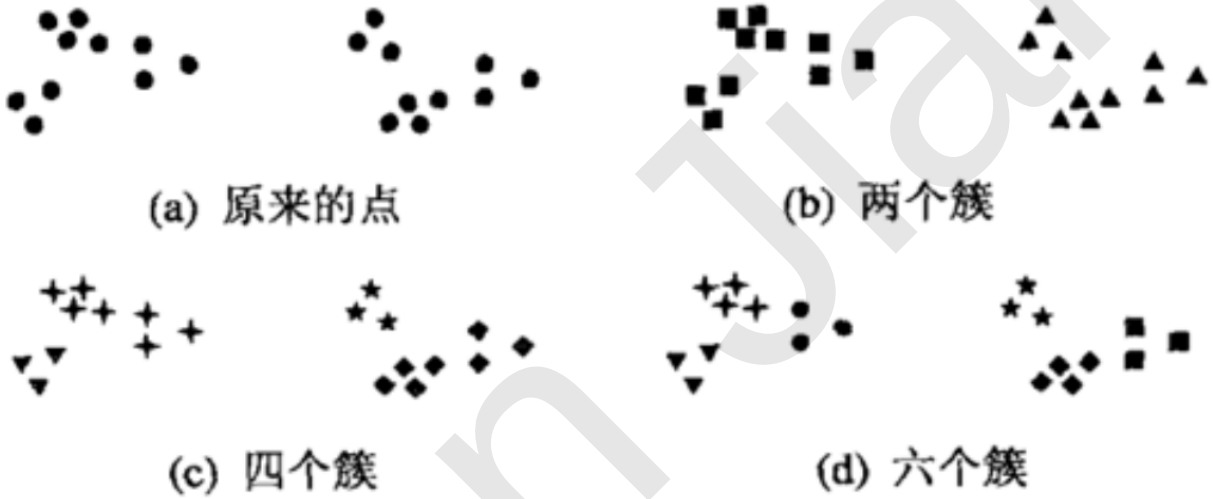
k-均值聚类算法

一、聚类分析概述

聚类分析是无监督类机器学习算法中最常用的一类，其目的是将数据划分成有意义或有用的组（也被称为**簇**）。组内的对象相互之间是相似的（相关的），而不同组中的对象是不同的（不相关的）。组内的相似性 （同质性）越大，组间差别越大，聚类就越好。

1. 簇的定义

简单来说，簇就是分类结果中的类，但实际上簇并没有明确的定义，并且簇的划分没有客观标准，我们可以利用下图来理解什么是簇。该图显示了 20个点和将它们划分成簇的3种不同方法。标记的形状指示簇的隶属关系。下图分别将数据划分成两部分、四部分和六部分。将2个较大的簇每一个都划分成3个子簇可能是人的视觉系统造成的假象。此外，说这些点形成4个簇可能也不无道理。该图表明簇的定义是不精确的，而最好的定义依赖于数据的特性和期望的结果。



2.常用的聚类算法

最常用的聚类算法有以下三种：

•**K-means聚类**：也称为**K均值聚类**，它试图发现k（用户指定个数）个不同的簇，并且每个簇的中心采用簇中所含值的均值计算而成。

•**层次聚类**：层次聚类(hierarchical clustering)试图在不同层次对数据集进行划分，从而形成树形的聚类结构。

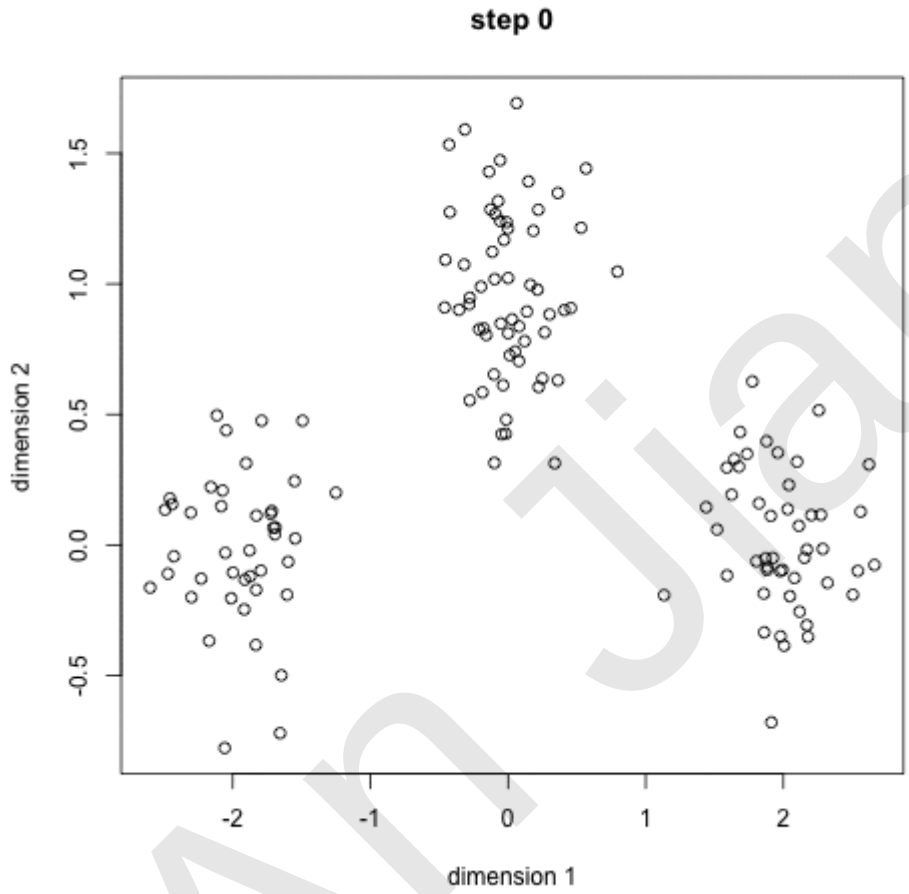
•**DBSCAN**：这是一种基于密度的聚类算法，簇的个数由算法自动地确定。低密度区域中的点被视为噪声而忽略，因此DBSCAN不产生完全聚类。

二、K-均值聚类算法

K均值是发现给定数据集的k个簇的算法。簇个数k是用户给定的，每一个簇通过其质心（centroid）来描述。

K均值工作流程是这样的：首先，随机选择K个初始质心，其中K是用户指定的参数，即所期望的簇的个数。然后将数据集中每个点指派到最近的质心，而指派到一个质心的点即为一个簇。然后，根据指派到簇的点，将每个簇的质心更新为该簇所有点的平均值。重复指派和更新步骤，直到簇不发生变化，或等价地，直到质心不发生变化。

该过程如下图所示，该图显示如何从3个质心出发，通过12次指派和更新，找出最后的簇。质心用符号“x”指示；属于同一个簇的所有点具有相同颜色的标记。



1. K-均值算法的python实现

根据k-均值算法的工作流程，我们可以写出伪代码：

```
创建k个点作为初始质心（通常是随机选择）
当任意一个点的簇分配结果发生改变时：
    对数据集中的每个点：
        对每个质心：
            计算质心与数据点之间的距离
        将数据点分配到据其最近的簇
    对每个簇，计算簇中所有点的均值并将均值作为新的质心
直到簇不再发生变化或者达到最大迭代次数
```

在伪代码中，有提到“最近”的说法，那就意味着要进行某种距离的计算。**附录1**中给大家总结了几种距离度量方法。这里我们使用的是最简单的欧式距离。

1.1 导入数据集

此处先以经典的鸢尾花数据集为例，来帮助我们建模，数据存放在iris.txt中

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

#导入数据集
iris = pd.read_csv('iris.txt',header = None)
iris.head()
iris.shape
```

1.2 构建距离计算函数

我们需要定义一个两个长度相等的数组之间欧式距离计算函数，在不直接应用计算距离计算结果，只比较距离远近的情况下，我们可以用距离平方和代替距离进行比较，化简开平方运算，从而减少函数计算量。

此外需要说明的是，涉及到距离计算的，一定要注意量纲的统一。如果量纲不统一的话，模型极易偏向量纲大的那一方。**附录2**给大家总结了一些归一化的方法，供大家参考。此处选用鸢尾花数据集，基本不需要考虑量纲问题。

```
"""
函数功能：计算两个数据集之间的欧式距离
输入：两个array数据集
返回：两个数据集之间的欧氏距离（此处用距离平方和代替距离）
"""
def distEclud(arrA, arrB):
    d = arrA - arrB
    dist = np.sum(np.power(d, 2), axis=1)
    return dist
```

1.3 编写自动生成随机质心的函数

在定义随机质心生成函数时，首先需要计算每列数值的范围，然后从该范围中随机生成指定个数的质心。此处我们使用numpy.random.uniform()函数生成随机质心。

```
"""
函数功能：随机生成k个质心
参数说明：
    dataSet:包含标签的数据集
    k：簇的个数
返回：
    data_cent：K个质心
"""
def randCent(dataSet, k):
    n = dataSet.shape[1]
    data_min = dataSet.iloc[:, :n-1].min()
    data_max = dataSet.iloc[:, :n-1].max()
```

```
    data_cent = np.random.uniform(data_min,data_max,(k, n-1))
    return data_cent
```

验证上述定义函数，在iris中随机生成三个质心

```
iris_cent = randCent(iris, 3)
iris_cent
```

1.4 编写K-Means聚类函数

在执行K-Means的时候，需要不断的迭代质心，因此我们需要两个可迭代容器来完成该目标：

第一个容器用于存放和更新质心，该容器可考虑使用list来执行，list不仅是可迭代对象，同时list内不同元素索引位置也可用于标记和区分各质心，即各簇的编号；

第二个容器则需要记录、保存和更新各点到质心之间的距离，并能够方便对其进行比较，该容器考虑使用一个三列的数组来执行，其中第一列用于存放最近一次计算完成后某点到各质心的最短距离，第二列用于记录最近一次计算完成后根据最短距离得到的代表对应质心的数值索引，即所属簇，即质心的编号，第三列用于存放上一次某点所对应质心编号（某点所属簇），后两列用于比较质心发生变化后某点所属簇的情况是否发生变化。

```
"""
函数功能：k-均值聚类算法
参数说明：
    dataSet：带标签数据集
    k:簇的个数
    distMeas:距离计算函数
    createCent:随机质心生成函数
返回：
    centroids:质心
    result_set:所有数据划分结果
"""
def kMeans(dataSet, k, distMeas=distEclud, createCent=randCent):
    m,n = dataSet.shape
    centroids = createCent(dataSet, k)
    clusterAssment = np.zeros((m,3))
    clusterAssment[:, 0] = np.inf
    clusterAssment[:, 1: 3] = -1
    result_set = pd.concat([dataSet, pd.DataFrame(clusterAssment)], axis=1,
ignore_index = True)
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        for i in range(m):
            dist = distMeas(dataSet.iloc[i, :n-1].values, centroids)
            result_set.iloc[i, n] = dist.min()
            result_set.iloc[i, n+1] = np.where(dist == dist.min())[0]
        clusterChanged = not (result_set.iloc[:, -1] == result_set.iloc[:, -2]).all()
        if clusterChanged:
            cent_df = result_set.groupby(n+1).mean()
            centroids = cent_df.iloc[:,:n-1].values
            result_set.iloc[:, -1] = result_set.iloc[:, -2]
    return centroids, result_set
```

鸢尾花数据集带进去，查看模型运行效果：

```
iris_cent,iris_result = kMeans(iris, 3)
iris_cent
iris_result.head()
```

有以下几点需要特别注意：

- 设置统一的操作对象result_set

  为了调用和使用的方便，此处将clusterAssment容易转换为了DataFrame并与输入DataFrame合并，组成的对象可作为后续调用的统一对象，该对象内即保存了原始数据，也保存了迭代运算的中间结果，包括数据所属簇标记和数据质心距离等，该对象同时也作为最终函数的返回结果；

- 判断质心发生是否发生改变条件

  注意，在K-Means中判断质心是否发生改变，即判断是否继续进行下一步迭代的依据并不是某点距离新的质心距离变短，而是某点新的距离向量（到各质心的距离）中最短的分量位置是否发生变化，即质心变化后某点是否应归属另外的簇。在质心变化导致各点所属簇发生变化的过程中，点到质心的距离不一定会变短，即判断条件不能用下述语句表示

  ```
  if not (result_set.iloc[:, -1] == result_set.iloc[:, -2]).all()
  ```

- 合并DataFrame后索引值为n的列

  这里有个小技巧，能够帮助迅速定位DataFrame合并后列的索引，即两个DF合并后后者的第一列在合并后的DF索引值为n，第二列索引值为n+1

- 质心和类别一一对应

  即在最后生成的结果中，centroids的行标即为result_set中各点所属类别

2. 算法验证

数编写完成后，先以testSet数据集测试模型运行效果（为了可以直观看出聚类效果，此处采用一个二维数据集进行验证）。testSet数据集是一个二维数据集，每个观测值都只有两个特征，且数据之间采用空格进行分隔，因此可采用pd.read_table()函数进行读取

```
testSet = pd.read_table('testSet.txt', header=None)
testSet.head()
testSet.shape
```

然后利用二维平面图形观察其分布情况

```
plt.scatter(testSet.iloc[:,0], testSet.iloc[:,1]);
```

可以大概看出数据大概分布在空间的四个角上，后续我们将对此进行验证。然后利用我们刚才编写的K-Means算法对其进行聚类，在执行算法之前需要添加一列虚拟标签列（算法是从倒数第二列开始计算特征值）

```
ze = pd.DataFrame(np.zeros(testSet.shape[0]).reshape(-1, 1))
test_set = pd.concat([testSet, ze], axis=1, ignore_index = True)
test_set.head()
```

然后带入算法进行计算，根据二维平面坐标点的分布特征，我们可考虑设置四个质心，即将其分为四个簇，并简单查看运算结果

```
test_cent, test_cluster = kMeans(test_set, 4)
test_cent
test_cluster.head()
```

将分类结果进行可视化展示，使用scatter函数绘制不同分类点不同颜色的散点图，同时将质心也放入同一张图中进行观察

```
plt.scatter(test_cluster.iloc[:,0], test_cluster.iloc[:, 1], c=test_cluster.iloc[:,
-1])
plt.scatter(test_cent[:, 0], test_cent[:, 1], color='red',marker='x',s=80);
```

能够看出，K-Means模型判别结果和我们初步判别结果类似，二维空间内偏向于分布在四个角上。

3.误差平方和SSE计算

误差平方和（SSE）是聚类算法模型最重要评估指标，接下来，在手动编写的K-Means快速聚类函数基础上计算结果集中的误差平方和。在kMeans函数生成的结果result_set中，第n列，也就是我们定义的clusterAssment容器中的第一列就保留了最近一次分类结果的某点到对应所属簇质心的距离平方和，因此我们只需要对result_set中的第n列进行简单求和汇总，即可得聚类模型误差平方和。

```
test_cluster.iloc[:, 3].sum()
```

```
iris_result.iloc[:,5].sum()
```

```
In [170]: test_cluster.iloc[:,3].sum()
Out[170]: 149.95430467642635
In [171]: iris_result.iloc[:,5].sum()
Out[171]: 78.94506582597731
```

当然，对于聚类算法而言，误差平方和仍然有一定的局限性，主要体现在以下几点：

- 对于任意数据集而言，聚类误差平方和和质心数量高度相关，随着质心增加误差平方和将逐渐下降，虽然下降过程偶尔会有小幅起伏，不是严格递减，极限情况是质心数量和数据集行数保持一致，此时误差平方和将趋于零（思考为何不为0）；
- 同时，误差平方和还与数据集本身数据量大小、量纲大小、数据维度高度相关，数据量越大、量纲越大、维度越高则在相同质心数量情况下误差平方和也将更大；

因此，模型误差平方和没有绝对意义，比较不同数据集聚类结果的误差平方和没有任何意义，误差平方和在聚类分析中主要作用有以下两点：

- 确定模型最优化目标，结合距离计算方法进而推导质心选取方法；
- 对于确定数据集可绘制横轴为质心数量、纵轴为误差平方和的曲线，可以判断，曲线整体将呈现下降趋势，我们可从中找到“骤然下降”的某个拐点，则该点可作为聚类分类数量的参考值，即可利用SSE绘制聚类数量学习曲线。当然，由于聚类分析本身属于探索性质的算法，曲线拐点给出的值只是当前数据维度和数据量的情

况，数据在高维空间内的分布可能呈现的集中分布趋势，并不一定代表客观事物的自然规律，也不是所有数据集的聚类分类数量学习曲线都存在拐点；

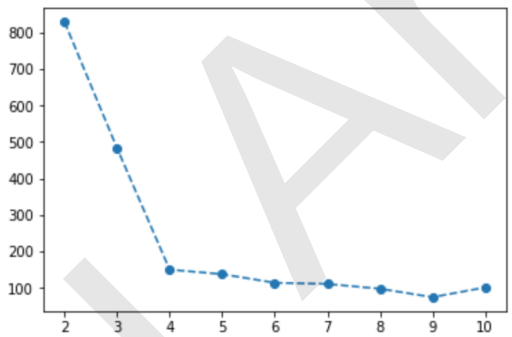
接下来，尝试绘制聚类分类数量的学习曲线，这里仍然考虑自定义一个函数来进行绘制，此处需要注意，质心数量选取建议从2开始，质心为1的时候SSE数值过大，对后续曲线显示效果有较大影响

```
"""
函数功能：聚类学习曲线
参数说明：
    dataSet：原始数据集
    cluster：Kmeans聚类方法
    k：簇的个数
返回：误差平方和SSE
"""
def kcLearningCurve(dataSet, cluster = kMeans, k=10):
    n = dataSet.shape[1]
    SSE = []
    for i in range(1, k):
        centroids, result_set = cluster(dataSet, i+1)
        SSE.append(result_set.iloc[:,n].sum())
    plt.plot(range(2, k+1), SSE, '--o')
    return SSE
```

然后在已有数据集上进行测试

```
kcLearningCurve(test_set)
```

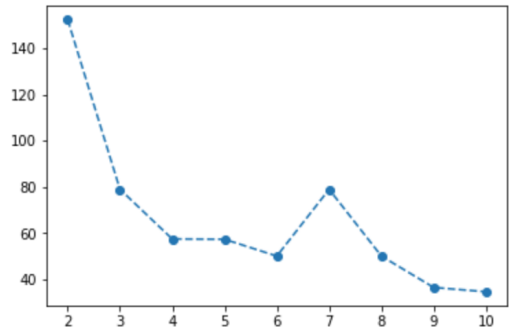
```
In [172]: kcLearningCurve(test_set);
```



此时能看出，质心由2增加到4的时候SSE下降速度较快，而从4开始，随着质心增加SSE下降速度有所递减，因此当质心为4的时候聚类效果较好

```
kcLearningCurve(iris)
```

```
In [173]: kcLearningCurve(iris);
```



而在iris数据集中，质心选取3个或4个为佳，其中质心选取4个时的聚类效果比选取3个质心要更好。这里虽然生物学上数据集对象本身包含了三个不同类别的鸢尾花，但就采集的数据和字段而言，从数据高维空间分布来说更加倾向于可分为4个簇，从侧面也说明数据量和数据集特征将影响模型判别效果，换而言之就是数据本身将影响模型对数据背后客观规律的探索。

三、模型收敛稳定性探讨

在执行前面聚类算法的过程中，好像虽然初始质心是随机生成的，但最终分类结果均保持一致。若初始化参数是随机设置（如此处初始质心位置是随机生成的），但最终模型在多次迭代之后稳定输出结果，则可认为最终模型收敛，所谓迭代是指上次运算结果作为下一次运算的条件参与计算，kMeans的每次循环本质上都是迭代，我们利用上次生成的中心点参与到下次距离计算中。

同时，在之前我们编写kMeans聚类算法中我们设置的收敛条件比较简单，就是最近两次迭代各点归属簇的划分结果，若结果不发生变化，则表明结果收敛，停止迭代。但实际上这种收敛条件并不稳定，尤其是在我们采用均值作为质心带入计算的过程中，实际上是采用了梯度下降作为优化手段，但梯度下降本质上是无约束条件下局部最优手段，局部最优手段最终不一定导致全局最优，即我们使用均值作为质心带入迭代，最终依据收敛判别结果计算的最终结果一定是基于初始质心的局部最优结果，但不一定是全局最优的结果，因此其实刚才的结果并不稳定，最终分类和计算的结果会收到初始质心选取的影响。

接下来验证初始质心选取最终如何影响K-means聚类结果的。这里我们可提前设置随机数种子，由于我们的初始质心由np.random类函数生成，所以随机数种子也要用np.random类方法生成

```
np.random.seed(123)
```

然后执行算法，注意，这里我们每执行一次kMeans函数，初始质心就会随机生成一次，我们需要观察的是重复执行算法过程会不会最终输出不同的分类结果。为方便更直观的表达，此处仅以test_set数据集验证过程为例进行讨论

- 验证test_set数据集三分类结果

  同时，利用多子图功能进行可视化结果展示

```
np.random.seed(123)
for i in range(1, 5):
    plt.subplot(2, 2, i)
    test_cent, test_cluster = kMeans(test_set, 3)
    plt.scatter(test_cluster.iloc[:,0], test_cluster.iloc[:, 1],
c=test_cluster.iloc[:, -1])
    plt.plot(test_cent[:, 0], test_cent[:, 1], 'o', color='red')
    print(test_cluster.iloc[:, 3].sum())
```

- 验证test_set数据集四分类结果

```
np.random.seed(123)
for i in range(1, 5):
    plt.subplot(2, 2, i)
    test_cent, test_cluster = kMeans(test_set, 4)
    plt.scatter(test_cluster.iloc[:,0], test_cluster.iloc[:, 1],
c=test_cluster.iloc[:, -1])
    plt.plot(test_cent[:, 0], test_cent[:, 1], 'o', color='red')
    print(test_cluster.iloc[:, 3].sum())
```

- 验证test_set数据集五分类结果

```
np.random.seed(123)
for i in range(1, 5):
    plt.subplot(2, 2, i)
    test_cent, test_cluster = kMeans(test_set, 5)
    plt.scatter(test_cluster.iloc[:,0], test_cluster.iloc[:, 1],
c=test_cluster.iloc[:, -1])
    plt.plot(test_cent[:, 0], test_cent[:, 1], 'o', color='red')
    print(test_cluster.iloc[:, 3].sum())
```

大家可以继续对iris数据集和更多分类的test_set数据集进行测试，最终我们能得出以下结论：

- 初始质心的随机选取在kMeans中将最终影响聚类结果；
- 质心数量从某种程度上将影响这种随机影响的程度，如果质心数量的选取和数据的空间集中分布情况越相类似，则初始质心的随机选取对聚类结果可能造成影响的概率越小，当然，这也和质心随机生成的随机方法也高度相关。

接下来，我们仍以test_set三分类为例，稍微修改kMeans函数，使其每一步都生成一张点图，借以查看随机初始化质心是如何一步步最终影响聚类结果的

```
"""
函数功能：绘制迭代聚类点图
参数说明：
    dataSet：原始数据集
    k：簇个数k
```

```
    distMeas：距离计算函数
    createCent：随机质心生成函数
返回：每一步生成的聚类点图
"""
def kMeans_1(dataSet, k, distMeas=distEclud, createCent=randCent):
    m,n = dataSet.shape
    centroids = createCent(dataSet, k)
    clusterAssment = np.zeros((m,3))
    clusterAssment[:, 0] = np.inf
    clusterAssment[:, 1: 3] = -1
    result_set = pd.concat([dataSet, pd.DataFrame(clusterAssment)], axis=1,
ignore_index = True)
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        for i in range(m):
            dist = distMeas(dataSet.iloc[i, :n-1].values, centroids)
            result_set.iloc[i, n] = dist.min()
            result_set.iloc[i, n+1] = np.where(dist == dist.min())[0]
        clusterChanged = not (result_set.iloc[:, -1] == result_set.iloc[:, -2]).all()
        if clusterChanged:
            cent_df = result_set.groupby(n+1).mean()
            centroids = cent_df.iloc[:,:n-1].values
            result_set.iloc[:, -1] = result_set.iloc[:, -2]
        plt.scatter(result_set.iloc[:,0], result_set.iloc[:, 1], c=result_set.iloc[:,
-1])
        plt.plot(centroids[:, 0], centroids[:, 1], 'o', color='red')
        plt.show()
```

```
np.random.seed(123)
kMeans_1(test_set, 3)
```

从中我们可清楚地看到随机初始质心是如何影响最终聚类分类结果的。解决该问题的办法，即尽量降低初始化质心随机性对最后聚类结果造成影响的方案有以下几种：

- 在设始化质心随机生成的过程中，尽可能的让质心更加分散

  这点其实我们在利用np.random.random进行[0,1)区间内取均匀分布抽样而不是进行正态分布抽样，就是为了能够尽可能的让初始质心分散；

- 人工制定初始质心

  即在观察数据集分布情况后，手工设置初始质心，此举也能降低随机性影响，但需要人为干预；

- 增量的更新质心

  可以在点到簇的每次指派之后，增量地更新质心，而不是在所有的点都指派到簇中之后才更新簇质心。注意，每步需要零次或两次簇质心更新，因为一个点或者转移到一个新的簇（两次更新），或者留在它的当前簇（零次更新）。使用增量更新策略确保不会产生空簇，因为所有的簇都从单个点开始；并且如果一个簇只有单个点，则该点总是被重新指派到相同的簇。

此外，如果使用增量更新，则可以调整点的相对权值；例如，点的权值通常随聚类的进行而减小。尽管这可能产生更好的准确率和更快的收敛性，但是在千变万化的情况下，选择好的相对权值可能是困难的。这些更新问题类以于人工神经网络的权值更新。

接下来就介绍一种最常用，效果也非常好的增量更新质心的方法：二分K-均值法。

————【以下的内容，完整版课程给大家详细讲解】————

四、二分K-均值算法

k-means聚类算法通过不断的更新簇质心直到收敛为止，但是这个收敛是局部收敛到了最小值，并没有考虑全局的最小值。为了克服k-均值算法的局部最小值问题，有人提出了二分K-均值算法。该算法的思想是首先将所有点作为一个簇，然后讲该簇一分为二.之后选择其中一个簇继续划分，选择哪一个簇进行划分取决于对其划分是否可以最大程度降低SSE的值。上述基于SSE的划分过程不断重复,直到得到用户指定的簇数目为止。

1. 二分K均值法的python实现

2. 模型验证

五、聚类模型的评价指标

1. 误差平方和SSE

2. 轮廓系数

3. 轮廓系数的python实现

六、实例：对地图上的点进行聚类

使用Yahoo！PlaceFinder API收集数据，然后用二分K均值法构建模型

七、实例：K-Means算法之股票聚类

爬取标准普尔500指数的数据（2017/1/1~至今），计算其历史收益和波动率，然后继续使用K-Means聚类算法根据所述收益和波动率将股票划分为不同的组，以方便后续的投资组合构建。

【附录1】距离类模型中距离的确定

【附录2】归一化方法

其他

- 菊安酱的直播间：https://live.bilibili.com/14988341
- 下周一（2019/1/7）将讲解**使用Apriori算法进行关联分析**，欢迎各位进入菊安酱的直播间观看直播
- 如有问题，可以给我留言哦~