支持向量机SVM编程作业

支持向量机SVM编程作业
TASK1:对地理数据应用二分K-均值算法聚类
一、软件设计
1. 设计目标
2. 算法设计
① K-均值_K-means原理
③ 代码实现
(1)K-means普通算法
(2)二分K-均值算法
(3)主函数+数据可视化
二、运行结果
三、总结体会
TASK2: 根据采集的WiFi信息对用户进行聚类
一、软件设计
1. 设计目标
2. 算法设计
① K-均值_K-means
② 代码实现
(1) 数据读取
(2)主函数+数据可视化
二、运行结果
三、总结体会
程序设计心得:

TASK1:对地理数据应用二分K-均值算法聚类

一、软件设计

1. 设计目标

(1) 问题概述:

你的朋友Drew希望你带他去城里庆祝他的生日。由于其他一些朋友也会过来，所以需要你提供一个大家都可行的计划。Drew给了你希望去的69个地址和相应的经纬度。你要决定将这些地方进行聚类的最佳策略，这样可以安排交通工具抵达这些簇的质心，然后步行到每个簇内地址

(2) 问题分析:

- 准备数据：用Python解析文本文件，根据经纬度信息计算球面距离。
- 分析数据：使用Matplotlib构建一个二维数据图，其中包含簇与地图
- 聚类算法：应用二分k-均值算法，最后的输出是包含簇及簇中心的地图。

部分数据集内容如下，仅提取最后两列即可：

```
Dolphin II  10860 SW Beaverton-Hillsdale Hwy    Beaverton, OR   45.486502   -122.788346
Hotties 10140 SW Canyon Rd. Beaverton, OR   45.493150   -122.781021
Pussycats   8666a SW Canyon Road    Beaverton, OR   45.498187   -122.766147
Stars Cabaret   4570 Lombard Ave    Beaverton, OR   45.485943   -122.800311
Sunset Strip    10205 SW Park Way   Beaverton, OR   45.508203   -122.781853
Vegas VIP Room  10018 SW Canyon Rd  Beaverton, OR   45.493398   -122.779628
Full Moon Bar and Grill 28014 Southeast Wally Road  Boring, OR  45.430319   -122.376304
505 Club    505 Burnside Rd Gresham, OR 45.507621   -122.425553
Dolphin 17180 McLoughlin Blvd   Milwaukie, OR   45.399070   -122.618893
Dolphin III 13305 SE McLoughlin BLVD    Milwaukie, OR   45.427072   -122.634159
Acropolis   8325 McLoughlin Blvd    Portland, OR    45.462173   -122.638846
Blush   5145 SE McLoughlin Blvd Portland, OR    45.485396   -122.646587
```

(3) 目标要求

需要完成聚类代码编写，输出包含簇及簇中心的地图，并将其展示在实验报告中，编写实验报告需要简述实验代码编写思路、实验结果、算法流程等

（由于簇中心具有随机性，因此每次运行结果可能不同）

2. 算法设计

> **所用项目环境: IDE：VS code　　Python版本：python3.7.5**

在“无监督学习”(unsupervised learning)中，训练样本的标记信息是未知的,目标是通过对无标记训练样本的学习来揭示数据的内在性质及规律,为进一步的数据分析提供基础.此类学习任务中研究最多、应用最广的是“聚类”(clustering).

K-均值属于原型聚类，原型聚类亦称“基于原型的聚类”(prototype-based clustering),此类算法假设聚类结构能通过一组原型刻画,在现实聚类任务中极为常用.通常情形下，算法先对原型进行初始化,然后对原型进行迭代更新求解.采用不同的原型表示、不同的求解方式,将产生不同的算法.

① K-均值_K-means原理

- **普通k-均值算法**:
  K-means 是我们最常用的基于欧式距离的聚类算法，其认为两个目标的距离越近，相似度越大，主要算法步骤如下：

  1. 选择初始化的 k 个样本作为初始聚类中心$a = a_1, a_2, \ldots, a_k$;
  2. 针对数据集中每个样本x_i 计算它到 k 个聚类中心的距离并将其分到距离最小的聚类中心所对应的类中;
  3. 针对每个类别a_j ，重新计算它的聚类中心$a_j = \frac{1}{|c_j|}\sum_{x \in c_i} x$（即属于该类的所有样本的质心）；
  4. 重复上面 2 3 两步操作，直到达到某个中止条件（迭代次数、最小误差变化等）。

  转化为伪代码如下：

```
获取数据 n 个 m 维的数据
随机生成 K 个 m 维的点
while(t)
    for(int i=0;i < n;i++)
        for(int j=0;j < k;j++)
            计算点 i 到类 j 的距离
    for(int i=0;i < k;i++)
        1. 找出所有属于自己这一类的所有数据点
        2. 把自己的坐标修改为这些数据点的中心点坐标
end
```

时间复杂度：$O(t*k*n*m)$，其中，t 为迭代次数，k 为簇的数目，n 为样本点数，m 为样本点维度。

空间复杂度：$O(m(n+k))$，其中，k 为簇的数目，m 为样本点维度，n 为样本点数。

- ◦ 优点
  a. 容易理解，聚类效果不错，虽然是局部最优， 但往往局部最优就够了；
  b. 处理大数据集的时候，该算法可以保证较好的伸缩性；
  c. 当簇近似高斯分布的时候，效果非常不错；
  d. 算法复杂度低。
- ◦ 缺点
  a. K 值需要人为设定，不同 K 值得到的结果不一样；
  b. 对初始的簇中心敏感，不同选取方式会得到不同结果；
  c. 对异常值敏感；
  d. 样本只能归为一类，不适合多分类任务；
  e. 不适合太离散的分类、样本类别不平衡的分类、非凸形状的分类。

- **改进二分k-均值算法**
  为克服K-均值算法收敛于局部最小值的问题，因此有人提出二分k-均值算法，该算法首先将所有点作为一个簇，然后将该簇一分为二。之后选择其中一个簇继续进行划分，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误差平方和(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

```
将所有点看成一个簇
当簇数目小于k时
对于每一个簇
   计算总误差
   在给定的簇上面进行k-均值聚类（k=2）
   计算将该簇一分为二之后的总误差
选择使得误差最小的那个簇进行划分操作
```

③ 代码实现

> **具体代码已经已经给出demo--py文件，只需写好调用函数即可，下面粗略分析给出的demo算法自己的理解**

(1)K-means普通算法

算法介绍：按照算法原理中分析的K-means普通算法的伪代码用python实现

- **数据读取函数：**

```
def kMeans(dataSet, k, distMeas=distEclud, createCent=randCent):
    m = shape(dataSet)[0]
    clusterAssment = mat(zeros((m, 2)))
    centroids = createCent(dataSet, k)
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        for i in range(m):
            minDist = inf
            minIndex = -1
            for j in range(k):
                distJI = distMeas(centroids[j, :], dataSet[i, :])
                if distJI < minDist:
                    minDist = distJI
                    minIndex = j
            if clusterAssment[i, 0] != minIndex: clusterChanged = True
            clusterAssment[i, :] = minIndex, minDist**2
```

```
        print(centroids)
        for cent in range(k):
            ptsInClust = dataSet[nonzero(clusterAssment[:, 0].A == cent)[0]]
            centroids[cent, :] = mean(ptsInClust, axis=0)
    return centroids, clusterAssment
```

- ◦ **代码细节**:

> **输入**:
> dataSet：数据集　k：分簇数目
> distEclud：距离函数　randCent：随机选择簇中心方法
> **输出**:
> centroids:簇中心　clusterAssment:簇数据矩阵，[簇索引值，存储误差]

(1) 使用**clusterChanged**作为是否继续循环迭代的标志变量，每次循环默认将标志位设为false
(2) 使用for循环将每个样本点分配到与其最近的簇中心所在的簇，若存在有样本被划入不同的簇，这说明数据聚类没有完成，将标志位设为true：**[if clusterAssment[i, 0] != minIndex: clusterChanged = True]**
(3) 每次循环之后都意味着簇划分发生了变化，使用for循环计算每一个簇的新中心，使用np.mean计算均值即可计算中心 **[centroids[cent, :] = mean(ptsInClust, axis=0)]**

- **距离函数**

```
def distEclud(vecA, vecB):
    return sqrt(sum(power(vecA - vecB, 2)))
```

- ◦ **代码细节:**
  (1) 代码简单，距离函数采用欧氏距离，直接将$\sqrt{(A-B)^2}$转化为代码即可

- **随机生成簇中心函数**

```
def randCent(dataSet, k):
    n = shape(dataSet)[1]
    centroids = mat(zeros((k, n)))
    for j in range(n):
        minJ = min(dataSet[:, j])
        rangeJ = float(max(dataSet[:, j]) - minJ)
        centroids[:, j] = mat(minJ + rangeJ * random.rand(k, 1))
    return centroids
```

- ◦ **代码细节:**
  (1) 随机产生k个簇中心，保证簇中心的每个维度的取值都在这个纬度所有值的最小值与最大值的左闭右开区间内
  (2) minJ每个维度的最小值，rangeJ每个维度最小值与最大值的差值，使用 **(minJ + rangeJ * random.rand(k, 1)** 实现中心点的选取

(2)二分K-均值算法

算法介绍: 对K-means算法进行改进,避免收敛于局部最小

- **二分K-means算法**

```
def biKmeans(dataSet, k, distMeas=distEclud):
    m = shape(dataSet)[0]
    clusterAssment = mat(zeros((m, 2)))
    centroid0 = mean(dataSet, axis=0).tolist()[0]
```

```
    centList = [centroid0]
    for j in range(m):
        clusterAssment[j, 1] = distMeas(mat(centroid0), dataSet[j, :])**2
    while (len(centList) < k):
        lowestSSE = inf
        for i in range(len(centList)):
            ptsInCurrCluster = dataSet[nonzero(clusterAssment[:, 0].A == i)[0],
:]
            centroidMat, splitClustAss = kMeans(ptsInCurrCluster, 2, distMeas)
            sseSplit = sum(splitClustAss[:, 1])
            sseNotSplit = sum(clusterAssment[nonzero(clusterAssment[:, 0].A !=
i)[0], 1])
            print("sseSplit, and notSplit: ", sseSplit, sseNotSplit)
            if (sseSplit + sseNotSplit) < lowestSSE:
                bestCentToSplit = i
                bestNewCents = centroidMat
                bestClustAss = splitClustAss.copy()
                lowestSSE = sseSplit + sseNotSplit
        bestClustAss[nonzero(bestClustAss[:, 0].A == 1)[0], 0] = len(centList)
        bestClustAss[nonzero(bestClustAss[:, 0].A == 0)[0], 0] = bestCentToSplit
        print('the bestCentToSplit is: ', bestCentToSplit)
        print('the len of bestClustAss is: ', len(bestClustAss))
        centList[bestCentToSplit] = bestNewCents[0, :].tolist()[0]
        centList.append(bestNewCents[1, :].tolist()[0])
        clusterAssment[nonzero(clusterAssment[:, 0].A == bestCentToSplit)[0], :]
= bestClustAss
    return mat(centList), clusterAssment
```

- ◦ **代码细节：**

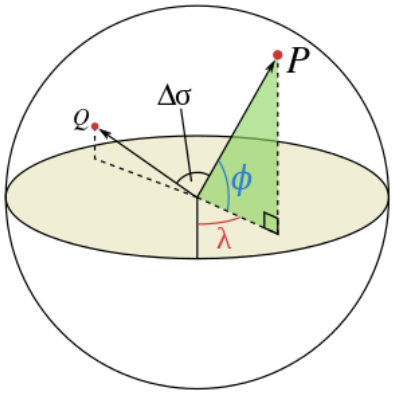
> 输入:
> dataSet：数据集　k：分簇数目　distEclud：距离函数
> 输出:
> centroids:簇中心　clusterAssment:簇数据矩阵，[簇索引值，存储误差]

(1) 代码实现下列的伪代码，参考之前叙述的原理及伪代码，这里不再叙述，跳转回二分K-均值算法原理
主要实现为参考伪代码，在普通k-均值的基础上增加簇二拆分功能即可
(2) 首先初始化第一个初始簇中心 **{centroid0 = mean(dataSet, axis=0).tolist()[0]}** ，然后设置循环，根据距离函数计算每一个样本的误差作为初始化
(3) 采用一个while循环 **[while (len(centList) < k):]** 作为整个簇划分主体，每一次循环中再次使用for循环 **[for i in range(len(centList)):]** 对当前每一个簇进行拆分每个簇，并计算拆分后的SSE，选择拆分后SSE最小的簇，保存拆分
(4)当有新的簇拆分完成后，给一个簇增加新的索引，另一个索引不变，索引保存在bestClustAss中，循环的最后与k-means一样重置簇中心，重新计算误差。

- **经纬度距离函数**:

```
def distSLC(vecA, vecB):
    a = sin(vecA[0, 1] * pi / 180) * sin(vecB[0, 1] * pi / 180)
    b = cos(vecA[0,1]*pi/180) * cos(vecB[0,1]*pi/180) * \
                    cos(pi * (vecB[0,0]-vecA[0,0]) /180)
    return arccos(a + b) * 6371.0
```

- ◦ **代码细节**:
  参考维基百科上的球面计算公式文章编写代码



(1) 代码使用Great-circle distance球面距离计算公式简单计算，距离为弧线距离
(2) 让λ_1, ϕ_1和λ_2, ϕ_2表示为两点1，2以弧度为单位的经纬度，用$\Delta\lambda, \Delta\phi$表示他们之间的误差，可以得到两点之间的角度$\Delta\sigma$

$$\Delta\sigma = arccos(sin\phi_1 sin\phi_2 + cos\phi_1 cos\phi_2 cos(\Delta\lambda))$$

定义这个球的半径为r，则实际弧线距离为

$$d = r\Delta\sigma$$

(3)主函数+数据可视化

算法介绍： 调用上述编写的k-means算法并可视化数据

```
def clusterClubs(numClust=5):
    datList = []
    for line in open('Task1/places.txt').readlines():
        lineArr = line.split('\t')
        datList.append([float(lineArr[4]), float(lineArr[3])])
    datMat = mat(datList)
    myCentroids, clustAssing = biKmeans(datMat, numClust, distMeas=distSLC)
    fig = plt.figure()
    rect = [0.0, 0.0, 1.0, 1.0]
    scatterMarkers = ['s', 'o', '^', '8', 'p', 'd', 'v', 'h', '>', '<']
    axprops = dict(xticks=[], yticks=[])
    ax0 = fig.add_axes(rect, label='ax0', **axprops)
    imgP = plt.imread('Task1/Portland.png')
    ax0.imshow(imgP)
    ax1 = fig.add_axes(rect, label='ax1', frameon=False)
    for i in range(numClust):
        ptsInCurrCluster = datMat[nonzero(clustAssing[:, 0].A == i)[0], :]
        markerStyle = scatterMarkers[i % len(scatterMarkers)]
        ax1.scatter(ptsInCurrCluster[:, 0].flatten().A[0], ptsInCurrCluster[:,
1].flatten().A[0], marker=markerStyle, s=90)
    ax1.scatter(myCentroids[:, 0].flatten().A[0], myCentroids[:,
1].flatten().A[0], marker='+', s=300)
    plt.show()
```

- **代码细节**:

  (1) 数据读取代码较为简单，python文件操作使用readline读取每一行即可。
  (2) 调用上述实现的二分k-均值算法biKmeans()得到算法结果数据，可视化使用 **import matplotlib.pyplot as plt** 库函数
  (3) 可视化过程使用ax0显示画布图片，使用ax1显示聚类后的数据集，数据只使用 **ptsInCurrCluster[:, 0]、ptsInCurrCluster[:, 1]** 两个特征方便二维可视化，使用

scatterMarkers = ['s', 'o', '^', '8', 'p', 'd', 'v', 'h', '>', '<'] 使不同的数据有不同的显示图形；同时使用 **ax1.scatter** 显示簇中心为"+"

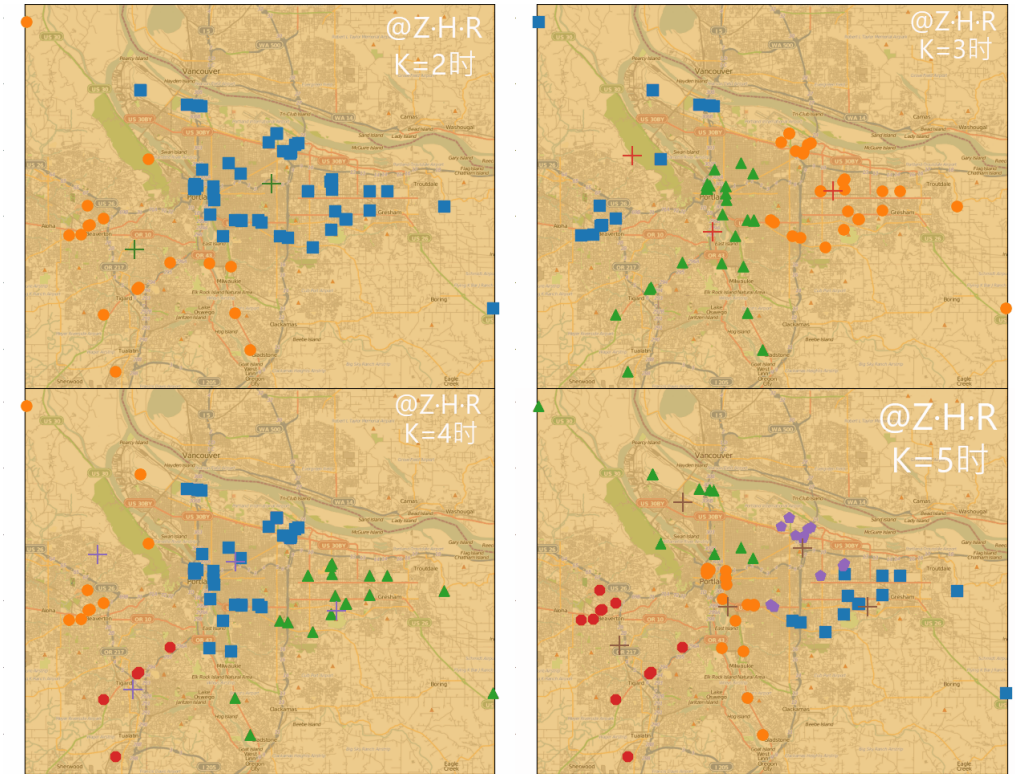
二、运行结果

1. 结果一_终端显示

```
PS F:\own_file\课程文件\机器学习导论\聚类\聚类_编程作业> python -u "f:\ow
习导论\聚类\聚类_编程作业\Task1\Kmeansch10.py"
[[-122.40306647   45.58433457]
 [-122.76854203   45.51063528]]
[[-122.53014919   45.518213  ]
 [-122.67609427   45.50972269]]
[[-122.54432282   45.52042118]
 [-122.69133217   45.5067651 ]]
[[-122.54868607   45.51882187]
 [-122.69551477   45.50729503]]
sseSplit, and notSplit:  3043.2633161055337 0.0
the bestCentToSplit is:  0
the len of bestClustAss is:  69
[[-122.54917107   45.54994804]
 [-122.51562129   45.45919858]]
[[-122.56775189   45.53629784]
 [-122.51575418   45.48863609]]
[[-122.57677259   45.53831618]
 [-122.51195754   45.49332931]]
[[-122.57979735   45.53623741]
 [-122.50800208   45.49604769]]
[[-122.57989794   45.53332622]
 [-122.50186825   45.49706533]]
[[-122.57768158   45.53263337]
 [-122.49860291   45.49496564]]
sseSplit, and notSplit:  464.7205983452951 2191.824427523823
[[-122.70907672   45.48104634]
 [-122.64082495   45.59215887]]
[[-122.69544887   45.48630247]
```


- 图片分析：可以看出程序运行时信息打印正常，由于打印信息较多，上图只显示一部分。

2. 结果二_不同K数据可视化



- 图片分析：上图K=2，3，4，5时不同的输出结果，不同的簇用不用的形状表示，用+显示簇中心，可以看出数据聚类的结果正常。

三、总结体会

1. 软件工程量较大，建议采用模块化设计，同时不同的内容可以分为不同的文件，优先设计函数头，确定参数及返回值后具体设计逻辑
2. 聚类算法的构建，可以参考《机器学习-西瓜书》 中的相关算法的伪代码，然后将伪代码实现即可。
3. 参考给出的样例demo算法时，可以使用ide的debug功能进行调试，加深对于代码的理解，方便自己进行修改，也加深对于聚类的k-means算法的理解。

TASK2: 根据采集的WiFi信息对用户进行聚类

一、软件设计

1. 设计目标

(1) 数据集讲解：

- **数据集**：数据集存于DataSetKMeans1.csv与DataSetKMeans2.csv中，两个数据集相互独立。
- **BSSIDLabel**：SSID标识符，每个AP（接入点，如路由器）拥有1个或多个不同的BSSID，但1个BSSID只属于1个AP；
- **RSSLabel**：该BSSID的信号强度，单位dbm；
- **RoomLabel**：该BSSID被采集时所属的房间号；
- **SSIDLabel**：该BSSID的名称，不唯一；
- **finLabel**： finLabel标号相同，表示这部分BSSID在同一时刻被采集到；我们将在同一时刻采集的所有BSSID及其相应RSS构成的矢量称为一个指纹$f_i = [BSSID_1 : RSS_1, BSSID_2 : RSS_2, \dots]$；由于BSSID的RSS在不同位置大小不同，因此指纹可以唯一的标识一个位置

BSSIDLabel	RSSLabel	RoomLabel	SSIDLabel	finLabel
06:69:6c:0a:bf:02	-56	1	HUST_WIRELESS	1
20:76:93:3a:ae:78	-69	1	HC5761	1
4a:69:6c:07:a1:e7	-69	1	HUST_WIRELESS_AUTO	1
2a:69:6c:05:c5:25	-71	1	HUST_WIRELESS_AUTO	1
24:05:0f:9b:e7:63	-80	1	zqxll2003	1
08:57:00:7b:63:16	-86	1	TL-WTR9500	1
94:53:30:11:ac:12	-86	1	HP-Print-12-LaserJet Pro MFP	1
88:25:93:58:06:63	-86	1	TP-LINK_580663	1
00:0f:e2:db:26:20	-70	1	ChinaNet	1

(2) 问题分析：

- **样本构造**: 一个指纹$f_i = [BSSID_1 : RSS_1, BSSID_2 : RSS_2, \dots]$为一个样本，共包含N个样本；有可能$BSSID_2$未在该样本中出现，但属于特征构造中得到的一个特征，则RSS值采用缺失值处理方法进行处理。
- **特征构造**: 不同样本中$BSSID$集合不尽相同，因此可以采用所有样本$BSSID$集合的并集作为特征，如指纹f_i的$BSSID$集合为$B_i = BSSID_j | BSSID_j \in f_i$，则特征可表示为$B_u = \cup_{i=1}^{N} B_i$.
- **缺失值处理**: 使用特殊值填充，如0，-100等。
- **样本间距离计算**: 可以采用欧式距离；可以仅计算两个样本中同一特征均为接收到的RSS值的平均距离

(3) 目标要求

编写代码分别对DataSetMeans1.csv和DataSetMeans2.csv两个数据集完成聚类实验，k（k>=2）取不同的值，评估聚类的内部指标DB指数

$$DBI = \frac{1}{k}\sum_{i=1}^{k}\max_{j\neq i}\left(\frac{avg\left(C_i\right)+avg\left(C_j\right)}{d_{cen}\left(\mu_i,\mu_j\right)}\right)$$

其中，μ代表簇C的中心点，avg(C)对应于簇C内样本间的平均距离，$d_{cen}\left(\mu_i, \mu_j\right)$对应于簇$C_i$和中i心点$C_j$间的距离。显然，DBI的值越小越好。

（由于簇中心具有随机性，因此每次运行结果可能不同）

2. 算法设计

> **所用项目环境: IDE：VS code　　Python版本：python3.7.5**

① K-均值_K-means

K-mens手写算法实现参考task1中的内容，这里不再阐述，跳转回二分K-均值算法原理

下列代码实现，使用sklearn库函数Kmeans，下面为搜索到的sklearn中kmeans的帮助文档

sklearn.cluster.KMeans¶

class sklearn.cluster.**KMeans**(*n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='deprecated', verbose=0, random_state=None, copy_x=True, n_jobs='deprecated', algorithm='auto'*) [source]

K-Means clustering.

Read more in the User Guide.

Parameters:

n_clusters : *int, default=8*
The number of clusters to form as well as the number of centroids to generate.

init : *{'k-means++', 'random'}, callable or array-like of shape (n_clusters, n_features), default='k-means++'*
Method for initialization:

'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence. See section Notes in k_init for more details.

'random': choose `n_clusters` observations (rows) at random from data for the initial centroids.

If an array is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.

If a callable is passed, it should take arguments X, n_clusters and a random state and return an initialization.

n_init : *int, default=10*
Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.

max_iter : *int, default=300*

下面为在Sklearn官方文档中找到的DBI的库函数：计算数据的DBI指数，评价聚类的好坏，DB越小越好

sklearn.metrics.davies_bouldin_score

sklearn.metrics.**davies_bouldin_score**(*X, labels*) [source]

Computes the Davies-Bouldin score.

The score is defined as the average similarity measure of each cluster with its most similar cluster, where similarity is the ratio of within-cluster distances to between-cluster distances. Thus, clusters which are farther apart and less dispersed will result in a better score.

The minimum score is zero, with lower values indicating better clustering.

Read more in the User Guide.

New in version 0.20.

Parameters:

X : *array-like of shape (n_samples, n_features)*
A list of `n_features`-dimensional data points. Each row corresponds to a single data point.

labels : *array-like of shape (n_samples,)*
Predicted labels for each sample.

Returns:

score: float
The resulting Davies-Bouldin score.

下面为在Sklearn官方文档中找到的MDS的库函数：数据降维，用于可视化数据

sklearn.manifold.MDS

class sklearn.manifold.**MDS**(*n_components=2, *, metric=True, n_init=4, max_iter=300, verbose=0, eps=0.001, n_jobs=None, random_state=None, dissimilarity='euclidean'*) [source]

Multidimensional scaling.

Read more in the User Guide.

Parameters:

n_components : *int, default=2*
Number of dimensions in which to immerse the dissimilarities.

metric : *bool, default=True*
If `True`, perform metric MDS; otherwise, perform nonmetric MDS.

n_init : *int, default=4*
Number of times the SMACOF algorithm will be run with different initializations. The final results will be the best output of the runs, determined by the run with the smallest final stress.

max_iter : *int, default=300*
Maximum number of iterations of the SMACOF algorithm for a single run.

verbose : *int, default=0*
Level of verbosity.

eps : *float, default=1e-3*

② 代码实现

引用库函数代码如下：

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.manifold import MDS
from sklearn.metrics import davies_bouldin_score
import matplotlib.pyplot as plt
```

(1) 数据读取

算法介绍：使用pandas读取训练数据的csv文件，并将所需要信息提取出来转化为需要的$f_1 = [BSSID_1 : -30, BSSID_2 : 0, BSSID_3 : -45, BSSID_4 : -70]$数据格式

- **数据读取函数：**读取csv数据

```
def data_Get(data_Path):
    traincsv = pd.read_csv(data_Path, usecols=['BSSIDLabel', 'RSSLabel',
'RoomLabel', 'finLabel'])

    BSSID_List = traincsv['BSSIDLabel'].tolist()
    fin_List = traincsv['finLabel'].tolist()
    # RSS_List = traincsv['RSSLabel'].tolist()
    Room_List = list(traincsv['RoomLabel'].tolist())
    BSSID_List = list(set(BSSID_List))
    fin_List = list(set(fin_List))
    # RSS_List = list(set(RSS_List))

    return traincsv, BSSID_List, fin_List, Room_List
```

- ◦ **代码细节**：

> 输入:
> data_Path: 数据文件路径
> 输出:
> traincsv: csv整体数据　　BSSID_List: BBSID信息去重list
> fin_List: 采集时间fin去重List　　Room_List: 标签Room数据list

(1) 使用**pd.read_csv**读取csv数据，使用参数**usecols**只使用['BSSIDLabel', 'RSSLabel', 'RoomLabel', 'finLabel']这几列数据
(2) 对每读取后的数据，与字典操作类似通过key获取每一列的数据，同时使用tolist()函数将其转化为数组，例 **[BSSID_List = traincsv['BSSIDLabel'].tolist()]**

- **数据处理函数**：$f_1 = [BSSID_1 : -30, BSSID_2 : 0, BSSID_3 : -45, BSSID_4 : -70]$矩阵构建

```
def data_trans(data_Path):
    datacsv, BSSID_List, fin_List, Room_List = data_Get(data_Path)
    train_data = [[0 for i in range(len(BSSID_List))] for i in
range(len(fin_List))]
    for i in range(len(datacsv)):
        times = datacsv['finLabel'][i]
        bssid = datacsv['BSSIDLabel'][i]
        rss = datacsv['RSSLabel'][i]
        try:
            index = BSSID_List.index(bssid)
            train_data[times - 1][index] = rss
        except Exception:
            print("ERROR: unexpected BSSID")
    print(data_Path, "数据处理完毕")
    return np.mat(train_data), Room_List
```

- ◦ **代码细节：**

(1) 按照数据的大小初始化一个二维list，使用for循环，循环读取每一个time，bssid，rss值，并按照bssid的索引在对应位置填入rss
(2) 使用try，except的差错处理，避免数据读取错误，出现没有set去重过的bssid

(2)主函数+数据可视化

算法介绍： 使用k-means算法训练上述处理过的数据并可视化数据

```
if __name__ == "__main__":
    datapath1 = "Task2/DataSetKMeans1.csv"
    datapath2 = "Task2/DataSetKMeans2.csv"
    # X_train, Y_train = data_trans(datapath1)
    X_train, Y_train = data_trans(datapath2)

    numClust = len(set(Y_train))
    Y_pred = KMeans(n_clusters=numClust, random_state=0).fit_predict(X_train)

    dbi_score = davies_bouldin_score(X_train, Y_pred)
    print("Kmeans聚类结果的DBI指数为：%f" % dbi_score)

    mds = MDS(n_components=2)
    X_transformed = np.mat(mds.fit_transform(X_train))

    # 定义画布，背景
    fig = plt.figure("wifi信息聚类可视化")
    rect = [0.0, 0.0, 1.0, 1.0]
    # 不同图形标识
    scatterMarkers = ['s', 'o', '^', '8', 'p', 'd', 'v', 'h', '>', '<']
    ax1 = fig.add_axes(rect, label='ax1', frameon=False)
```

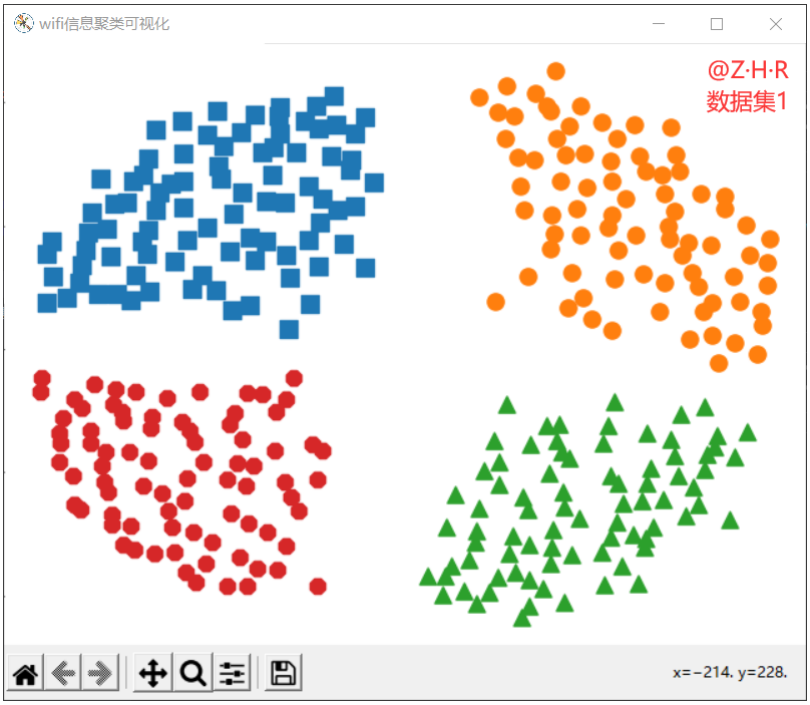
```
    # 采用不同图形标识不同簇
    for i in range(numClust):
        ptsInCurrCluster = X_transformed[np.nonzero(Y_pred[:] == i)[0], :]
        markerStyle = scatterMarkers[i % len(scatterMarkers)]
        ax1.scatter(ptsInCurrCluster[:, 0].flatten().A[0], ptsInCurrCluster[:,
1].flatten().A[0], marker=markerStyle, s=90)
    plt.show()
```

- **代码细节**:

  (1) 簇的数量使用**numClust = len(set(Y_train))**获取：对数据读取到的标签去重得到不同标签个数作为簇的数量。
  (2) **Y_pred = KMeans(n_clusters=numClust, random_state=0).fit_predict(X_train)**使用sklearn库函数聚类算法KMeans对训练数据进行训练，预测簇编号(标签)为Y_pred。
  (3) **dbi_score = davies_bouldin_score(X_train, Y_pred)**，使用sklearn库函数davies_bouldin_score计算聚类后的簇的DBI指数，评判聚类结果好坏
  (4) 先使用**mds = MDS(n_components=2)；X_transformed = np.mat(mds.fit_transform(X_train))**降维数据，可视化使用 **import matplotlib.pyplot as plt** 库函数，具体方法与task1中一样

二、运行结果

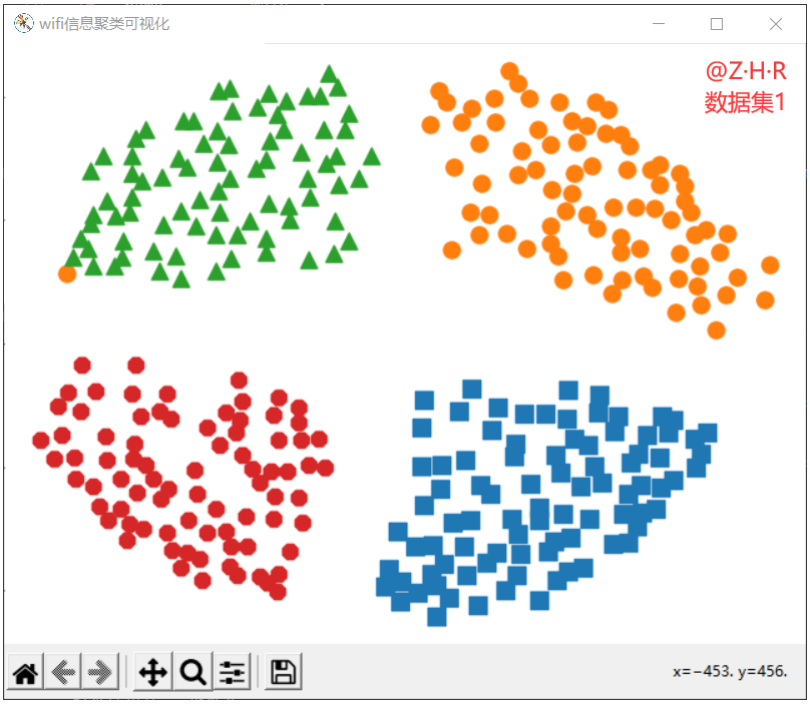
1. 结果一_数据集DataSetMeans1




```
PS F:\own_file\课程文件\机器学习导论\聚类\聚类_编程作业> python -u "f:\own_file\
Task2/DataSetKMeans1.csv 数据处理完毕
聚类结果Y_pred:
 [3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Kmeans聚类结果的DBI指数为：1.797185
```

- 图片分析：可以看出程序运行时信息打印正常，第一个数据集被聚类成4个簇，不同的簇用不同的图形显示，DBI=1.797，效果尚可。

2. 结果二__DataSetMeans2




```
PS F:\own_file\课程文件\机器学习导论\聚类\聚类_编程作业> python -u "f:\own_file\i
Task2/DataSetKMeans2.csv 数据处理完毕
聚类结果Y_pred:
 [3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Kmeans聚类结果的DBI指数为：1.790244
PS F:\own_file\课程文件\机器学习导论\聚类\聚类_编程作业>
```


- 图片分析：可以看出程序运行时信息打印正常，第一个数据集被聚类成4个簇，不同的簇用不同的图形显示，DBI=1.790，效果尚可。

三、总结体会

程序设计心得:

1. 软件工程量较大，建议采用模块化设计，同时不同的内容可以分为不同的文件，优先设计函数头，确定参数及返回值后具体设计逻辑
2. 聚类算法的构建，sklearn库函数的使用，需要多阅读其官方文档，理解给出的样例代码，熟悉各个参数的意思，在实际使用中，参数需要不断训练修改，以求最适合的参数。
3. 参考给出的样例demo算法时，可以使用ide的debug功能进行调试，加深对于代码的理解，方便自己进行修改，也加深对于聚类的k-means算法的理解。