大数据分析第二次作业：

SA19225404 吴语港

任务：

k-means算法优化

要求提交一份以“学号+姓名”命名的压缩文档，里面包含代码（与k-means比较）、说明文档、参考资料等。

1. 研究问题：

主要的聚类算法可以划分为如下几类：划分方法、层次方法、基于密度的方法、基于网格的方法以及基于模型的方法。

目前,聚类问题的研究不仅仅局限于上述的硬聚类（即每一个数据只能被归为一类，数据集中每一个样本都是被100%确定得分到某一个类别中），模糊聚类也是聚类分析中

研究较为广泛的一个分支。模糊聚类（通过隶属函数来确定每个数据隶属于各个簇的程度，而不是将一个数据对象硬性地归类到某一簇中，可以理解为每个样本是以一定的概率被分到某一个类别中）。

以下介绍几种常见的K-means算法的改进版本的思想

1、K-means++

假设已经选取了n个初始聚类中心(0<n<k), 则在选取第n+1个聚类中心时: 距离当前n个聚类中心越远的点会有更高的概率被选为第n+1个聚类中心, 在选取第一个聚类中心(n=1)时同样通过随机的方法.可以说这也符合我们的直觉:聚类中心当然是互相离得越远越好

2、ISODATA

类别数目随着聚类过程而变化，对类别数的'合并'(当聚类结果某一类中样本数太少, 或两个类间的距离太近时)'分裂' (当聚类结果中某一类的内方差太大, 将该类进行分裂)

3、Kernel K-means

kernel k-means,就是将每个样本进行一个投射到高维空间的处理,然后再将处理后的数据使用普通的k-means算法进行聚类

4、二分K-means

首先将所有点作为一个簇, 然后将该簇一分为二. 之后选择能最大限度降低聚类代价函数(也就是误差平方和)和簇划分为两个簇. 一次进行下去, 直到簇的数目等于用户给定的数目K为止.

隐含的一个原则是: 因为聚类的误差平方和能够衡量聚类性能, 该值越小表示数据点越接近于他们的质心, 聚类效果就越好. 所以我们就需要对误差平方和最大的簇进行再一次划分, 因为误差平方和越大, 表示该簇聚类效果越不好, 越有可能是多个簇被当成了一个簇, 所以我们首先需要对这个簇进行划分.

二分K均值算法可以加速K-means算法的执行速度, 因为它的相似度计算少了不受初始化问题的影响, 因为这里不存在随机点的选取, 且每一步都保证了误差最小

5、Mini Batch K-Means（适合大数据的聚类算法）

大数据量是什么量级？通过当样本量大于1万做聚类时，就需要考虑选用MiniBatch K-Means算法。

Mini Batch KMeans使用了一个种叫做Mini Batch（分批处理）的方法对数据点之间的距离进行计算。Mini Batch的好处是计算过程中不必使用所有的数据样本，而是从不同类别的样本中抽取一部分样本来代表各自类型进行计算。由于计算样本量少，所以会相应的减少运行时间，但另一方面抽样也必然会带来准确度的下降。

该算法的迭代步骤有两步

1. 从数据集中随机抽取一些数据形成小批量, 把他们分配给最近的质心

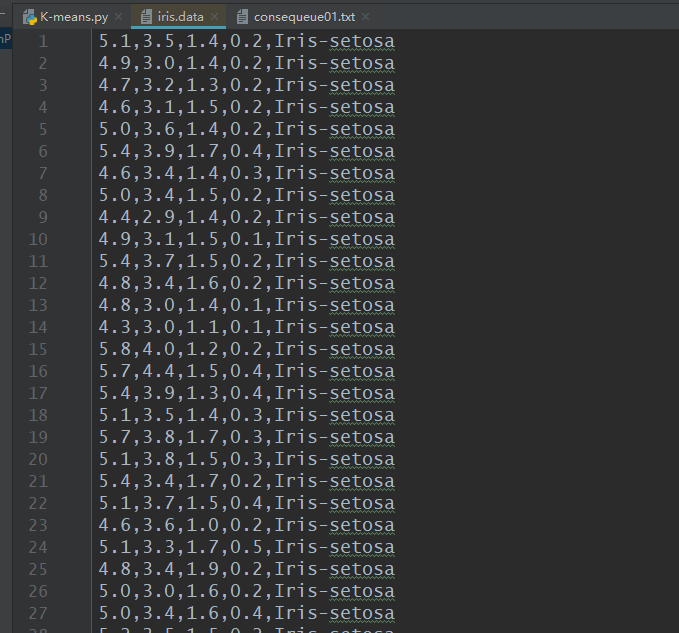
2. 更新质心

与K均值算法相比, 数据的更新是在每个小的样本集上. 对于每一个小批量, 通过计算平均值得到更新质心, 并把小批量里的试卷分配给该质心, 随着迭代次数的增加, 这些质心的变化是逐渐减小的, 直到质心稳定或达到指定的迭代次数, 停止计算

1. 实例及代码

数据集：

下载地址：http://archive.ics.uci.edu/ml/machine-learning-databases/iris/



一共四行数，一行字符串，最后一行多了一个回车，在py程序里面处理要删掉iris.data最后一行的空格。不然会出现str转float的错误。

代码：（二分K-means 与 K-means的对比）

# -\*- coding:utf-8 -\*-

# kmeans : Bisecting k-means cluster(二分K-means算法)

import numpy as np

import matplotlib.pyplot as plt

def readfile(filename):

    """

    读取数据集

    W：特征向量数组，只取前两个特征

    label：标签（类别）列表

    :param filename:

    :return:特征向量数组和标签集合列表

    """

    save\_path = "G:\\"

    with open(save\_path + filename, 'r') as f:

        length = len(f.readlines())

        print(filename, "length: %d" % length)

        W = np.zeros((length, 2))

        label = []

        i = 0

        f.seek(0, 0)

        for line in f.readlines():

            linestr = line.strip()

            linestrlist = line.split(',')

            print(linestrlist)

            # 鸢尾属植物数据集的特征共有四个，我们这里只取前两个特征作为特征向量。

            number\_data = [float(j) for j in linestrlist[0:2]]

            W[i, :] = np.array(number\_data)

            label.append(linestrlist[4].strip('\n'))

            i += 1

    return W, label

def createDataset(filename):

    """

    创建待分类数据集

    """

    data\_vector, label\_str = readfile(filename)

    # print(data\_vector,"\n",label)

    # 将原始数据集中非字符串标签改为用数字代表，方便后续画图

    label\_num = []

    for i in label\_str:

        if i == "Iris-setosa":

            label\_num.append(0)

        elif i == "Iris-versicolor":

            label\_num.append(1)

        else:

            label\_num.append(2)

    return data\_vector, label\_num

# 计算欧式距离

def euclDistance(vector1, vector2):

    return np.sqrt(sum(pow(vector2 - vector1, 2)))  # pow()是自带函数

# 使用随机样例初始化质心

def initCentroids(dataSet, k):

    numSamples, dim = dataSet.shape

    # numSample - 行，此处代表数据集数量  dim - 列，此处代表维度，例如只有xy轴的，dim=2

    centroids = np.zeros((k, dim))  # 产生k行，dim列零矩阵

    for i in range(k):

        index = int(np.random.uniform(0, numSamples))  # 给出一个服从均匀分布的在0~numSamples之间的整数

        # print("ok",index,numSamples)

        centroids[i, :] = dataSet[index, :]  # 第index行作为簇心

    # print(centroids)

    return centroids

# k均值聚类

def kmeans(dataSet, k):

    numSamples = dataSet.shape[0]

    # print(numSamples)

    # frist column stores which cluster this sample belongs to,

    # second column stores the error between this sample and its centroid

    clusterAssment = np.zeros((numSamples, 2))

    clusterChanged = True

    ## step 1: init centroids

    centroids = initCentroids(dataSet, k)

    while clusterChanged:

        clusterChanged = False

        ## for each sample

        for i in range(numSamples):

            minDist = 1000000.0  # 最小距离

            minIndex = 0  # 最小距离对应的点群

            ## for each centroid

            ## step2: find the centroid who is closest

            for j in range(k):

                distance = euclDistance(centroids[j, :], dataSet[i, :])  # 计算每个数据到每个簇中心的欧式距离

                if distance < minDist:  # 如果距离小于当前最小距离

                    minDist = distance  # 则最小距离更新

                    minIndex = j  # 对应的点群也会更新

            ## step 3: update its cluster

            if clusterAssment[i, 0] != minIndex:  # 如当前数据不属于该点群

                # 此处与书本上算法步骤稍微有点不同：当有一个数据的分类错误时就clusterChanged = True ，便会重新计算簇心。而书本上的终止条件是是新簇心等于上一次迭代后的簇心

                clusterChanged = True  # 聚类操作需要继续

                clusterAssment[i, :] = minIndex, minDist \*\* 2

                ## step 4: update centroids

        for j in range(k):

            # 提取同一类别的向量

            pointsInCluster = dataSet[np.nonzero(clusterAssment[:, 0] == j)[0]]

            # print("s",pointsInCluster.shape)

            # nonzeros返回的是矩阵中非零的元素的[行号]和[列号]

            # 将所有等于当前点群j的，赋给pointsInCluster，之后计算该点群新的中心

            if len(pointsInCluster) > 0:

                centroids[j, :] = np.mean(pointsInCluster, axis=0)  # 对每列求均值

    # print("center",centroids)

    return centroids, clusterAssment

# Bisecting k-means cluster(二分k-means算法)

def bisect\_kmeans(dataSet, k):

    numSamples = dataSet.shape[0]

    clusterAssment = np.zeros((numSamples, 2))  # 保存数据点信息，所属类别和误差

    centrol\_address = []  # 簇心坐标，注意是列表形式

    # step1 将所有点看做一个簇

    # print(np.mean(dataSet,axis=0).shape,np.mean(dataSet,axis=0))

    centrol\_address.append(np.mean(dataSet, axis=0).tolist())  # numpy.ndarray.tolist()将数组变为列表形式

    for j in range(numSamples):

        clusterAssment[j, 1] = euclDistance(dataSet[j, :], centrol\_address[0]) \*\* 2  # 计算SSE

    classes = len(centrol\_address)

    while classes < k:

        lowestSSE = float("inf")  # python 使用float("inf")表示正无穷大，float("-inf")表示负无穷大

        for i in range(classes):

            # 获得属于某一类的数据

            second\_kmeans\_verctor = dataSet[np.nonzero(clusterAssment[:, 0] == i)[0], :]

            # 调用上一节写的k-means方法，此时k=2，二分法。输出簇心坐标和数据聚类信息

            second\_centroids, second\_clusterAssment = kmeans(second\_kmeans\_verctor, k=2)

            # 划分数据的SSE，加上未划分数据的SSE，得到总的SSE

            SSE = sum(second\_clusterAssment[:, 1]) + sum(clusterAssment[np.nonzero(clusterAssment[:, 0] != i)[0], 1])

            # 找出最小的SSE以及相关数据信息

            if SSE < lowestSSE:

                lowestSSE = SSE

                bestCentToSplit = i  # 当前最适合继续二分法划分的类别

                bestNewCentrols = second\_centroids  # 二分法划分后的两个簇心向量

                bestClustAss = second\_clusterAssment.copy()  # 二分法划分后的数据信息

        # 由于是二分法划分，只有0，1两个类别，将属于1类别的所属信息转为下一个新的中心点的的类别号，即等于len(centrol\_address)=i+1

        # 注意这里两次类别信息的替换顺序不能错位！！！！

        bestClustAss[np.nonzero(bestClustAss[:, 0] == 1)[0], 0] = len(centrol\_address)

        # 将属于0类别的类别信息替换为本次被划分的数据信息，bestCentToSplit=i

        bestClustAss[np.nonzero(bestClustAss[:, 0] == 0)[0], 0] = bestCentToSplit

        # 这里替换中心点信息

        centrol\_address[bestCentToSplit] = bestNewCentrols[0, :].tolist()

        centrol\_address.append(bestNewCentrols[1, :].tolist())

        # 替换本次被用来重新划分数据的所属类别和误差信息

        clusterAssment[np.nonzero(clusterAssment[:, 0] == bestCentToSplit)[0], :] = bestClustAss

        classes = len(centrol\_address)

    # 返回中心点信息和数据类别和误差信息

    return np.array(centrol\_address), clusterAssment

def showCluster(dataSet, k, centroids, clusterAssment, old\_label):

    numSamples, dim = dataSet.shape  # numSample - 样例数量  dim - 数据的维度

    if dim != 2:

        print(" not two-dimensional data")

        return 1

    mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr']

    if k > len(mark):

        print("the k is too large! the max k is 10")

        return 1

    # draw all samples     对k-means聚类后的结果对数据进行绘图

    for i in range(numSamples):

        markIndex = int(clusterAssment[i, 0])

        plt.plot(dataSet[i, 0], dataSet[i, 1], mark[markIndex])

    plt.title(" The classification results of Bisecting k-means cluster")

    mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb']

    # draw the centroids

    for i in range(k):

        plt.plot(centroids[i, 0], centroids[i, 1], mark[i], ms=12.0)

    # 按照原始数据集自带的类别画图，用于与新分类后的数据进行对比

    plt.figure()  # 打开第二个窗口显示图片，而不是分屏显示

    for i in range(numSamples):

        markIndex = int(old\_label[i])

        plt.plot(dataSet[i, 0], dataSet[i, 1], mark[markIndex])

    plt.title("Original classification result")

    plt.show()

'''

函数功能： 二分 k-means聚类

采用UCI的数据集：鸢尾属植物数据库，Link：http://archive.ics.uci.edu/ml/machine-learning-databases/iris/

为了绘图方便，只取特征空间中的部分维度，即萼片长度、萼片宽度特征

'''

if \_\_name\_\_ == "\_\_main\_\_":

    # 数据集导入

    filename = "iris.data"

    # data\_vector, label分别为特征向量和原始标签

    data\_vector, label = createDataset(filename)

    # initCentroids(data\_vector,3)

    k = 3

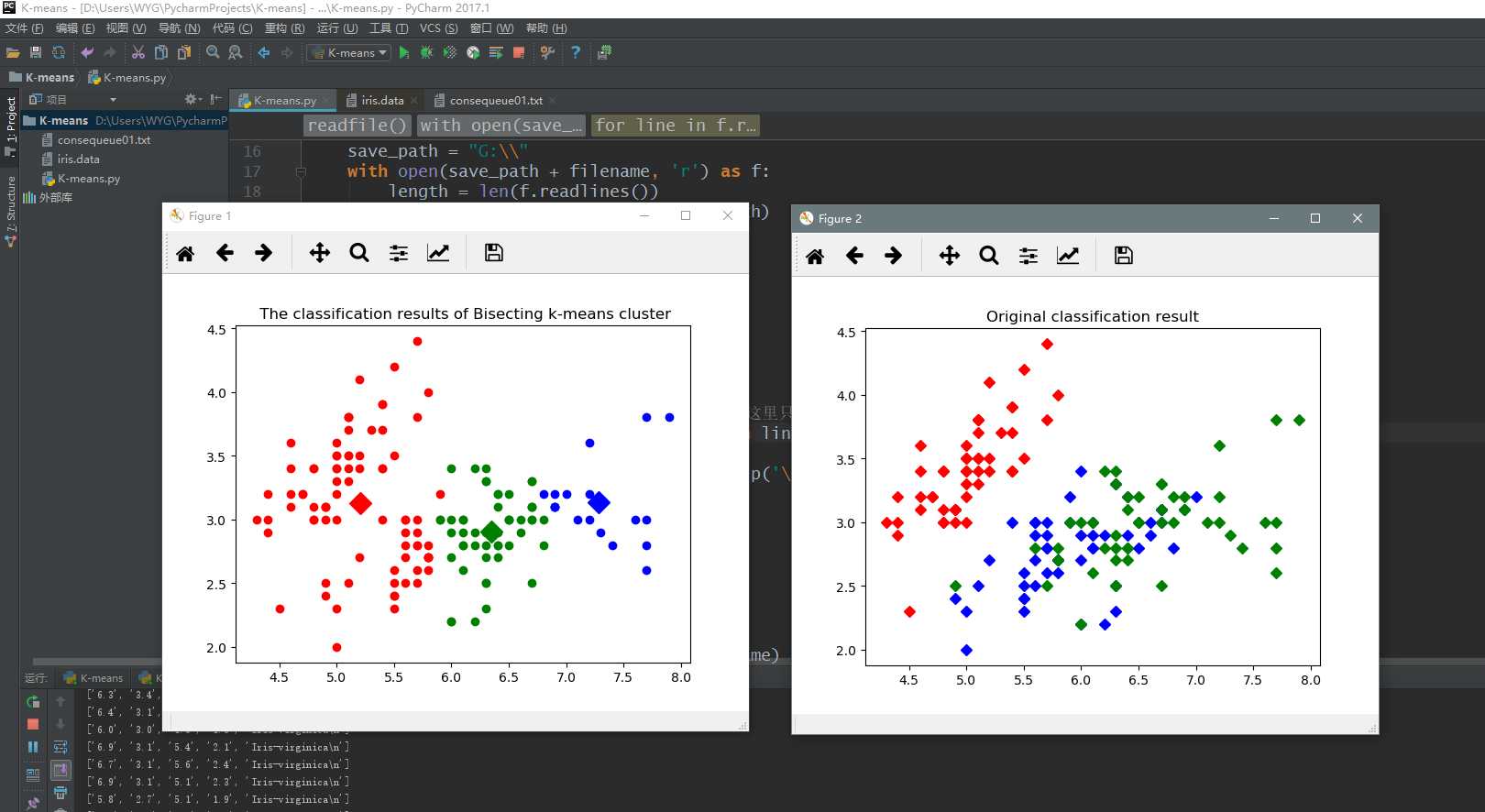
    # centroids, clusterAssment=kmeans(data\_vector,k)

    centroids, clusterAssment = bisect\_kmeans(data\_vector, k)

    # 按照原始标签和k-means聚类后的分类分别对数据进行绘图

    showCluster(data\_vector, k, centroids, clusterAssment, label)

运行结果及其对比：



右图为传统K-means的效果图，左图为二分K-means的效果图。

二分k-means算法放弃了随机选取簇中心的方法，而是从一个簇出发，根据聚类效果度量指标SSE来判断下一次应该对哪一个簇进行划分，因此该方法不会收敛到局部最小值，而是收敛到全局最小值。通过肉眼可以分辨出二分K-means的效果相比传统K-means好了很多。

1. 参考资料

<https://blog.csdn.net/datoutong_/article/details/78804151>

<https://blog.csdn.net/weixin_42555080/article/details/90401382>

<https://zhuanlan.zhihu.com/p/20445085>

<http://archive.ics.uci.edu/ml/machine-learning-databases/iris/>