一、算法介绍

1.knn 算法（K 最近邻算法)

kNN 算法的核心思想是如果一个样本在特征空间中的 k 个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。该方法在确定分类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。 kNN 方法在类别决策时，只与极少量的相邻样本有关。由于 kNN 方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，kNN 方法较其他方法更为适合。

2.朴素贝叶斯分类算法

朴素贝叶斯分类器(Naive Bayes Classifier 或 NBC)发源于古典数学理论，有着坚实的数学基础，以及稳定的分类效率。同时，NBC 模型所需估计的参数很少，对缺失数据不太敏感，算法也比较简单。理论上，NBC 模型与其他分类方法相比具有最小的误差率。但是实际上并非总是如此，这是因为 NBC 模型假设属性之间相互独立，这个假设在实际应用中往往是不成立的，这给 NBC 模型的正确分类带来了一定影响。

3.层次聚类算法

原理

层次聚类(Hierarchical Clustering)是聚类算法的一种，通过计算不同类别数据点间的相似度来创建一棵有层次的嵌套聚类树。在聚类树中，不同类别的原始数据点是树的最低层，树的顶层是一个聚类的根节点。

算法流程

1. 层次聚类的初始状态是每个样本点自成一个类。
2. 计算现有的所有类别的两两之间的类间距离，形成距离矩阵 D。
3. 根据 D 找到类间距离最小的两个类——I 和 J。

4. 重复 2-3 直到最小的类间距离已经达到设定的距离阈值。

4.K-Means 聚类算法

原理

K 均值聚类算法（k-means clustering algorithm）是一种迭代求解的聚类分析算法，其步骤是随机选取 K 个对象作为初始的聚类中心，然后计算每个对象与各个种子聚类中心之间的距离，把每个对象分配给距离它最近的聚类中心。聚类中心以及分配给它们的对象就代表一个聚类。样本全部分配完毕之后，将会根据类中的样本计算这个类的新的（理想）聚类中心，新的（理想）聚类中心是类中样本的均值。这个过程将不断重复直到满足某个终止条件。

终止条件可以是 K 个聚类中心都不再变化或者没有（或最小数目）对象被重新分配给不同的聚类，没有（或最小数目）聚类中心再发生变化，误差平方和局部最小。

算法流程

1. 随机化选取 k 个点或者直接选取前 k 个样本作为初始聚类中心。
2. 根据现有的聚类中心，将每个样本分配到距离最近的聚类中心所代表的类中。
3. 根据聚类情况，计算各类的样本的均值向量得到新的 k 个聚类中心。
4. 重复迭代 2-3 直到新的 k 个聚类中心和旧的 k 个聚类中心重合。

二、数据集介绍

整体数据介绍

来源：https://archive.ics.uci.edu/ml/datasets/seeds
描述：考察的组包括属于三种不同小麦品种的籽粒：卡马，罗莎和加拿大小麦，每种都随机选择 70 种元素进行试验。
属性信息：为了构建数据，测量了小麦籽粒的七个几何参数：
1.面积 A
2.周长 P
3.紧密度 C = 4 * pi * A / P ^ 2
4.籽粒长度

5. 籽粒宽度
6. 不对称系数
7. 籽粒槽的长度
所有这些参数都是连续的。

训练集与测试集的说明

从网站获取的数据各类种子各有 70 条数据。

对于分类算法

对于分类算法（knn 分类与朴素贝叶斯分类），训练集和测试集的划分采用随机将每一类种子中的 50%抽取出来作为训练集，剩余的 50%的种子作为测试集。
每做一次训练集与测试集的划分，进行一次分类测试，并记录预测结果和预测正确率，这记作 1 次测试。
分类算法程序中要求输入的测试次数。为了避免第 2 次训练时记录了第 1 次训练时的训练集而导致正确率偏高，每次划分训练集和测试集之后会清空上一次的训练集及训练结果。

对于聚类算法

对于聚类算法，将 210 条种子的数据全部输入，之后进行聚类。

三、实验结果

算法	运行命令
KNN 分类算法	python judge.py
贝叶斯分类算法	python judge_bayes.py
层次聚类算法	python hierarchical_clustering.py
K-Means 聚类算法	python k_means.py

1.knn 算法（K 最近邻算法)

输入：

100

以下内容是程序输出结果：

一共重复测试了 100 次

每次测试了 105 个种子。

平均正确率：90.9%

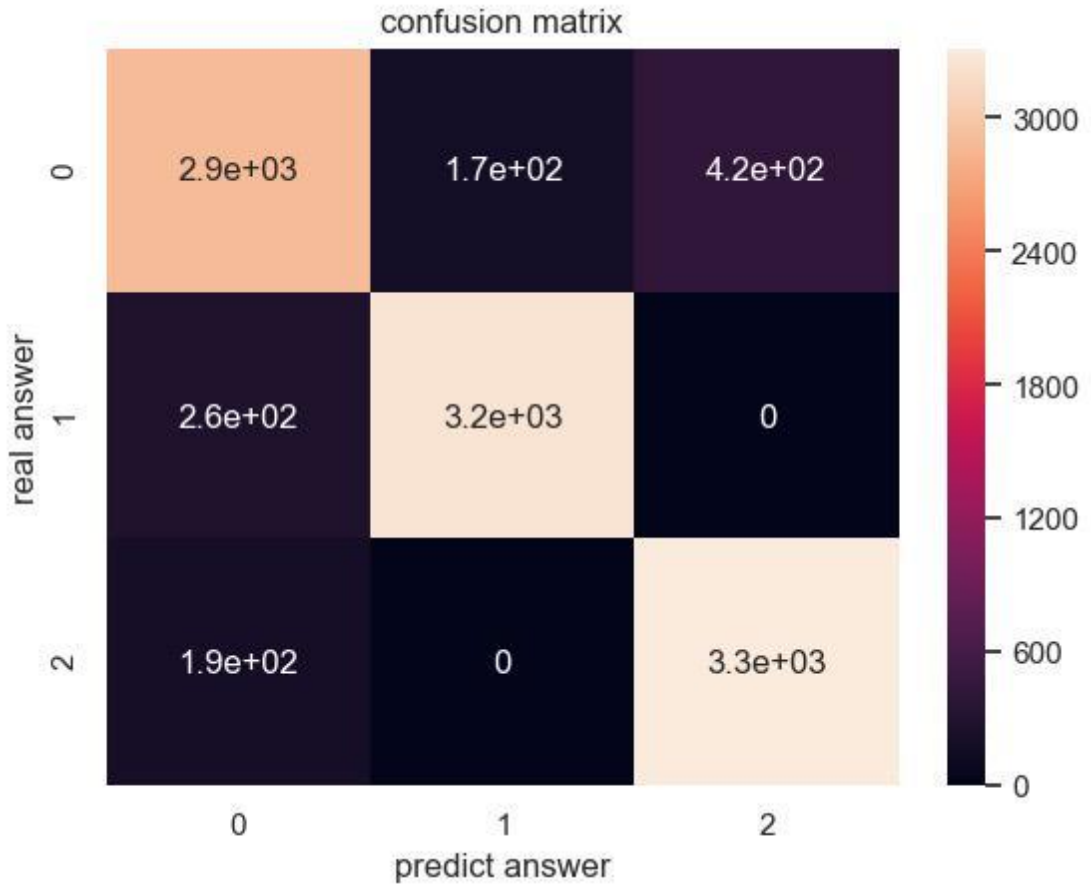
混淆矩阵：

[[2954 166 380]

 [240 3260   0]

 [169   0 3331]]

混淆矩阵热力图如下图所示。其中行代表真实的种子种类，列表示分类算法识别的种子种类。



2.朴素贝叶斯分类算法

输入：

100

以下内容是程序输出结果：

一共重复测试了 100 次

每次测试了 105 个种子。

平均正确率：90.6%

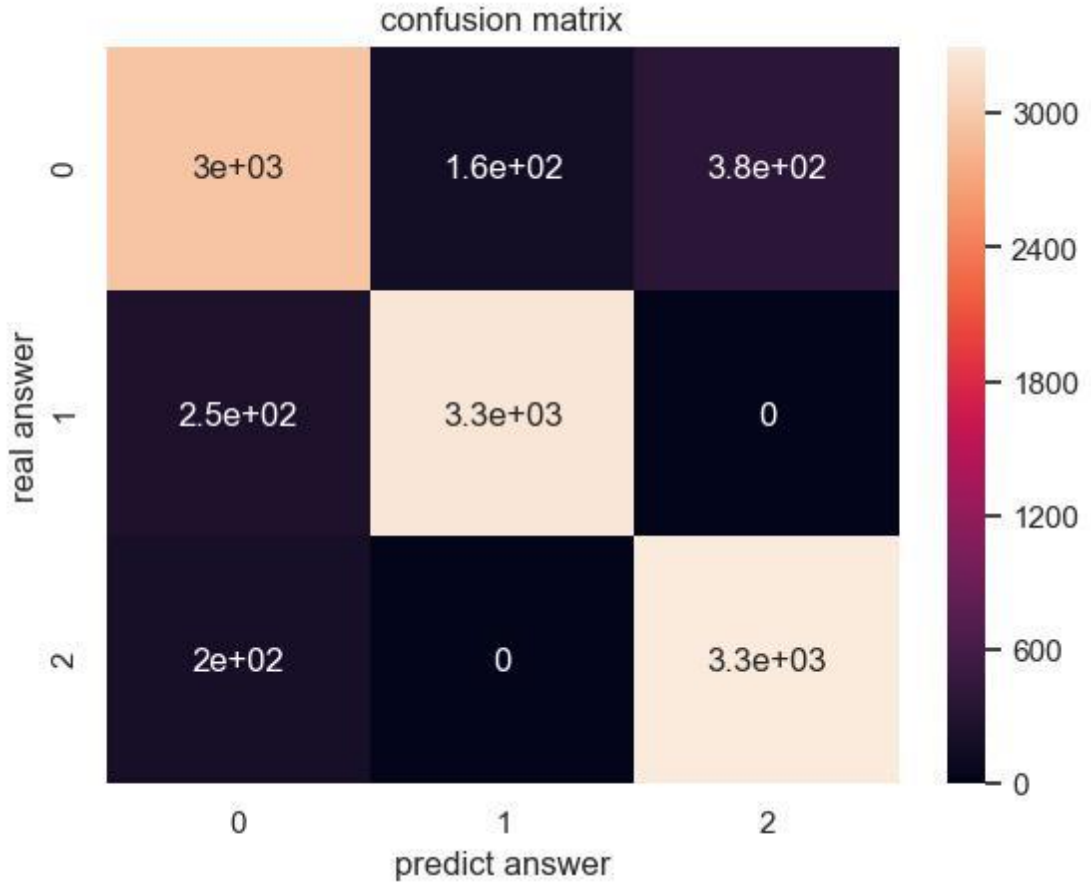
混淆矩阵：

[[2964 157 379]

 [247 3253   0]

 [200   0 3300]]

混淆矩阵热力图如下图所示。其中行代表真实的种子种类，列表示分类算法识别的种子种类。



3.层次聚类算法

以下是程序输出结果：

mean = 4.41729285579846

min = 0.0

max = 11.927155939703313

threshold = 7.72

下面展示分类结果的情况。

行是聚类出来的一个类，列是表示真实的种子的一个类别。

值是个数。例如第一行第一列表示聚类出来的类 1 中实际上是 1 类种子的种子数。

分类结果展示：

[[0. 47. 0.]
 [52. 23. 0.]
 [18. 0. 70.]]

种子真实分类各类的种子数目

[70. 70. 70.]

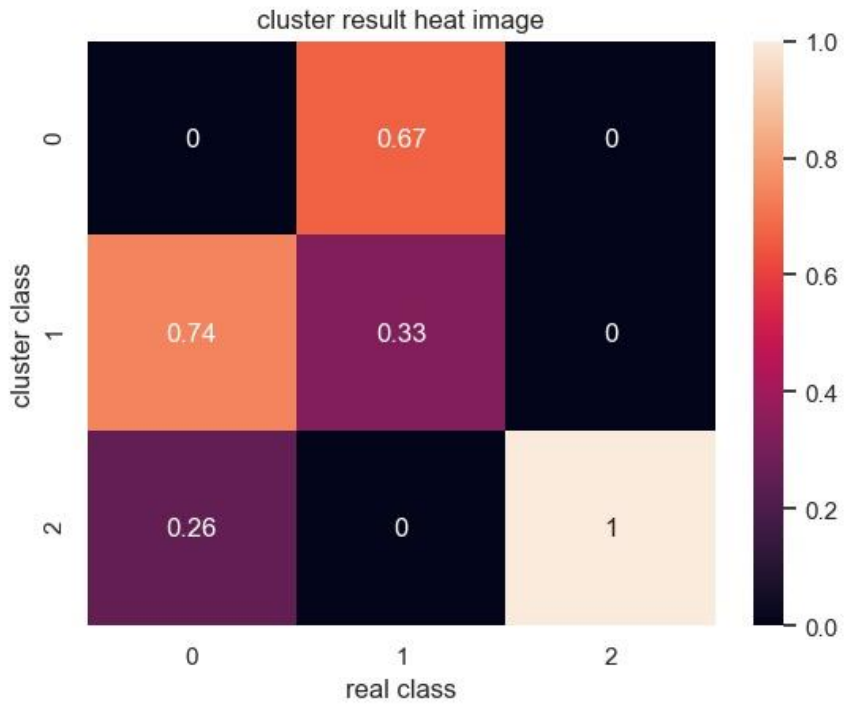
下面展示的是上面的矩阵转换为占比的情况

[[0.   0.67142857 0.   ]
 [0.74285714 0.32857143 0.   ]
 [0.25714286 0.   1.   ]]

请输入你要保存的矩阵图的文件名：

层次聚类_heat_img.jpeg

层次聚类分类情况热力图如下图：



4.K-Means 聚类算法

迭代次数是 5

新的类中心已经和旧的类中心重合了

下面展示分类结果的情况。

行是聚类出来的一个类，列是表示真实的种子的一个类别。

值是个数。例如第一行第一列表示聚类出来的类 1 中实际上是 1 类种子的种子数。

分类结果展示：

[[1. 60. 0.]

 [57. 10. 0.]

 [12. 0. 70.]]

种子真实分类各类的种子数目

[70. 70. 70.]

下面展示的是上面的矩阵转换为占比的情况

[[0.01428571 0.85714286 0.   ]

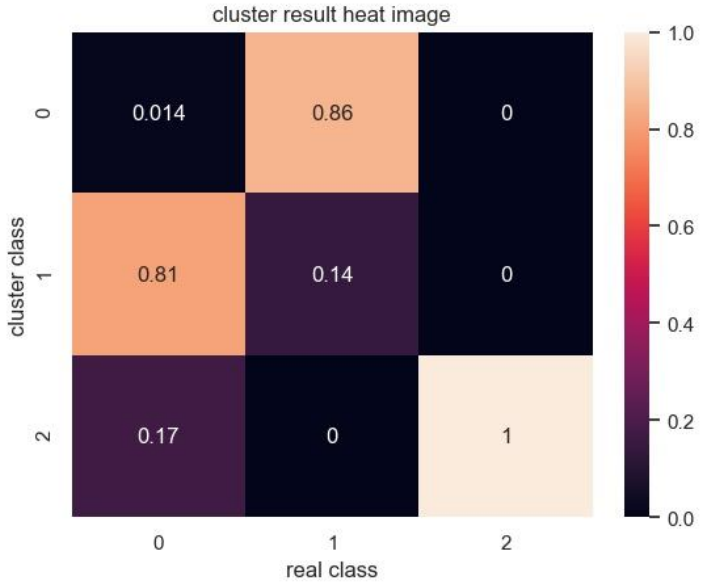
 [0.81428571 0.14285714 0.   ]

 [0.17142857 0.   1.   ]]

请输入你要保存的矩阵图的文件名：

k_Means 聚类_heat_img.jpeg

K-Means 聚类分析聚类结果热力图展示：



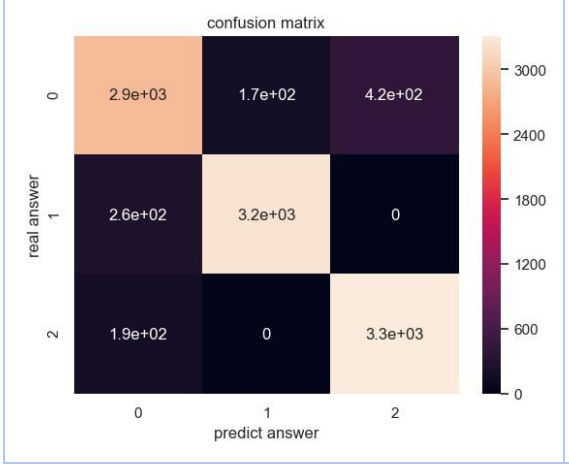
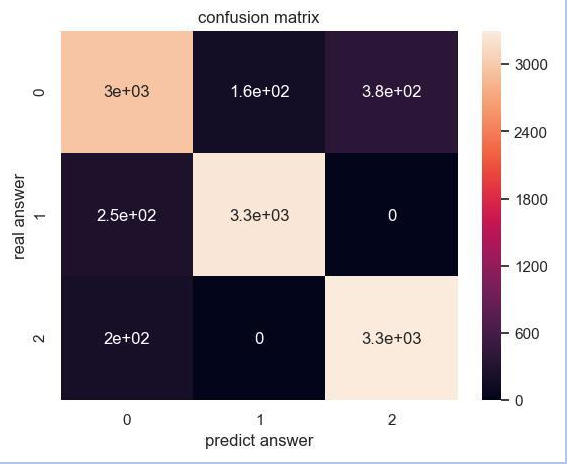
四、算法比较

1. knn 分类算法与贝叶斯分类算法比较

正确率比较

	KNN 分类算法	Bayes 分类算法
100 次测试平均正确率	90.9%	90.6%

混淆矩阵比较

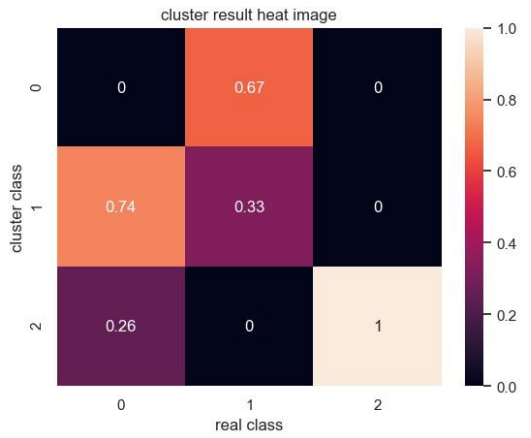
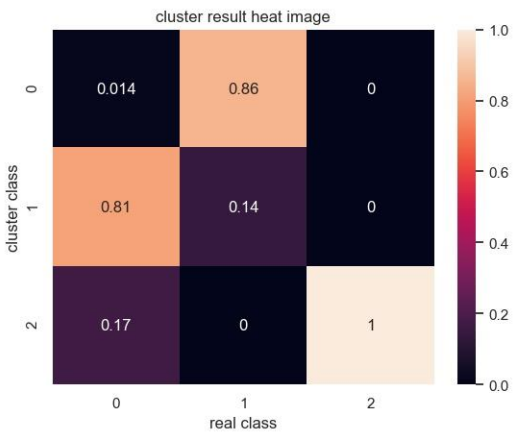
KNN 分类算法	Bayes 分类算法
	

比较结果评价

可以看到，两种算法的正确率差异不大，正确率都大概是 90%，分类效果良，但可以看到的是和已经存在的一些更加复杂的分类算法的 95%及以上的正确率有不小的差距。
即两种算法都是思想简单，实现简单快捷，准确率良好，但是不够优秀。
从混淆矩阵的热力图可以看出，两者的分类效果都比较良好。

2. 层次聚类算法与 K-Means 聚类算法比较

聚类情况占比构成（分布）热力图对比

层次聚类算法	K-Means 聚类算法
	

比较结果评价

从上图可以看出，层次聚类分析的聚类效果较差，真实种子的类别 1 约 3/4 和类别 2 中的 1/3 难以区分，类别 2 中的约 1/4 和类别 3 中的种子难以区分。即存在着把不同类别的种子聚成一类。这已经是通过调整距离阈值参数后所得的比较优秀的结果，但依旧聚类效果较差。而对于 K—Means 算法，程序运行结果显示，经过少数迭代（10 次左右），即将真实的种子类别 1,2,3 基本区分开来了，分类效果相对较好。

五、核心代码

1.knn 算法（K 最近邻算法)

```
class TR:
    def __init__(self,a=[],i=-1):
        self.vector=a
        self.answer=i

class RE:
    def __init__(self,a=0.0,i=-1):
        self.distance=a
```

```
        self.answer=i

    def __lt__(self,other):
        return self.distance<other.distance

def e_distance(x,y):
    nx = len(x)
    a=0
    for i in range(nx):
        a+=(x[i]-y[i])**2

    return a**0.5


trainingset=[]

def clear():
    trainingset.clear()


def recognize(vector,k=15):
    """
    识别种子类别
    :param vector: 要识别的种子的向量
    :return: 返回识别的种子类别的编号（1,2,3）
    """
    num=[0 for i in range(3)]

    testset=[]
    for i in range(len(trainingset)):
        distance=e_distance(vector,trainingset[i].vector)
        testset.append(RE(distance,trainingset[i].answer))
    testset.sort()
    #print(testset)
    for i in range(k):
        num[testset[i].answer-1]+=1
```

```
    maxnum=0
    for i in range(3):
        if(num[i]>num[maxnum]):
            maxnum=i
    return maxnum+1


def train(vectors, answers):
    """
    种子类别
    :param vectors: 一个列表，每个 vector[i]代表一个种子的向量
    :param answers: answers[i]代表 vectors[i]种子类别（1，2，3）
    :return: 无返回值
    """
    num=[0 for i in range(3)]
    for i in range(len(vectors)):
        if(num[answers[i]-1]<50):
            num[answers[i]-1]+=1
            trainingset.append(TR(vectors[i],answers[i]))
```

2.朴素贝叶斯分类算法

```
import math
import csv
import random

class Bayes:
    def __init__(self,vectors,answers):
        self.vectors=vectors
        self.answers=answers
        # model_par 以字典形式存放每一个类别的方差
        self.model_para={}

    def tarin_bayesModel(self):
```

```
        # 将训练集按照类别进行提取
        separated_class=self.separateByClass()
        # vectors 是列表，包含的是每个类别对应的向量集
        for classValue, vectors in separated_class.items():
            # 将每一个类别的均值和方差保存在对应的键值对中
            self.model_para[classValue] = self.summarize(vectors)
        return self.model_para

    # 计算均值
    def mean(self,numbers):
        return sum(numbers) / float(len(numbers))

    # 计算方差，注意是分母是 n-1
    def stdev(self,numbers):
        avg = self.mean(numbers)
        variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(
numbers) - 1)
        return math.sqrt(variance)

    # 对每一类样本的每个特征计算均值和方差，结果保存在列表中，依次为第一维特征、第二维特征等...的均值和方差
    def summarize(self,vectors):
        # zip 利用 * 号操作符，可以将不同元组或者列表压缩为为列表集合。用来提取每类样本下的每一维的特征集合
        summaries = [(self.mean(attribute), self.stdev(attribute)) for
attribute in zip(*vectors)]
        return summaries


    # 将训练集按照类别进行提取，以字典形式存放，Key 为类别，value 为列表，列表中包含的是每个类别对应的向量集
    def separateByClass(self):
        #字典用于存放分类后的向量集合
        separated_class = {}
        for i in range(len(self.vectors)):
            vector = self.vectors[i]
```

```
            if ((self.answers[i]-1) not in separated_class):
                separated_class[self.answers[i]-1] = []
            #     将每列数据存放在对应的类别下，列表形式
            separated_class[self.answers[i]-1].append(vector)
        return separated_class

    # 假定服从正态分布，对连续属性计算概率密度函数,公式参考周志华老师的西瓜书P151
    def calProbabilityDensity(self,x, mean, stdev):
        # x 为待分类数据
        exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
        return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

    # 计算待分类数据的联合概率
    def calClassProbabilities(self, inputVector):
        # summaries 为训练好的贝叶斯模型参数, inputVector 为待分类数据(单个)
        # probabilities 用来保存待分类数据对每种类别的联合概率
        probabilities = {}
        # classValue 为字典的 key(类别) ,classSummaries 为字典的 vlaue(每个类别每维特征的均值和方差),列表形式
        for classValue, classSummaries in self.model_para.items():
            probabilities[classValue] = 1
            # len(classSummaries)表示有多少特征维度
            for i in range(len(classSummaries)):
                # mean, stdev 分贝别表示每维特征对应的均值和方差
                mean, stdev = classSummaries[i]
                # 提取待分类数据的 i 维数据值
                x = inputVector[i]
                # 计算联合概率密度
                probabilities[classValue] *= self.calProbabilityDensity(x, mean, stdev)
        # 返回概率最大的类别
        prediction=max(probabilities,key=probabilities.get)
        return prediction+1
```

```
bayes=Bayes([],[])

def train(vectors, answers):
    """
    种子类别
    :param vectors: 一个列表，每个 vector[i]代表一个种子的向量
    :param answers: answers[i]代表 vectors[i]种子类别（1，2，3）
    :return: 无返回值
    """
    global bayes
    bayes=Bayes(vectors, answers)
    bayes.tarin_bayesModel()

def recognize(vector):
    """
    识别种子类别
    :param vector: 要识别的种子的向量
    :return: 返回识别的种子类别的编号（1,2,3）
    """

    return bayes.calClassProbabilities(vector)
```

3.层次聚类算法（使用最大距离作为类间距离）

```
from prepare_data import Data
import numpy as np
from mtr_to_img import *


def get_element(dis, results, n):
    """
    根据 dis 和 results 寻找类间距离最大的两个“类”(种子 r,c)的编号
    :param dis: 类间距离矩阵
    :param results: 描述每个种子归属哪个类
```

```
    :param n: 初始的种子数
    :return: r,c
    """
    r = c = None
    for i in range(n):
        if results[i] != i:
            continue
        for j in range(i):
            if j != results[j]:
                continue
            if r is None:
                r, c = i, j
            elif dis[r, c] > dis[i, j]:
                r, c = i, j
    if r is None:
        raise ValueError("全部变成一类了，无法找到距离最大的两个类！")
    return r, c


def merge(dis, results, n, r, c):
    """
    合并两个类间距离最大的类 r, c
    更新类间距离举证和 results
    :param dis: 类间距离矩阵
    :param results: 描述每个种子归属哪个类
    :param n: 初始的种子数
    :param r: 要合并的类 1
    :param c: 要合并的类 2
    :return:
    """
    # 类编号选择较小的数
    # 保证 r<c
    if r > c:
        r, c = r, c
    results[c] = r  # 让 c 类归属与 r 类
    # 更新距离矩阵
```

```
        # c 类不存在了，故 dis[,c]和 dis[c,]不再有意义，不管即可
        # 更新 dis[r,]与 dis[,r]即可
        for i in range(n):
            if i != results[i] or i == r:
                continue
            t = max(dis[i, r], dis[i, c])
            dis[r, i] = dis[i, r] = t


def get_root(results, i):
    if i == results[i]:
        return i
    else:
        results[i] = get_root(results, results[i])
        return results[i]


def hierarchical_cluster(seeds:list, threshold=7.72):
    """
    层次聚类
    :param seeds: 输入的种子数组(每个种子都是一个 list 列表)
    :param threhold: 停止的聚类的距离阈值
    :return: results, classes
    一个列表 group。group 和 vectors 等长。group[i]表示 vectors[i]的分类编
号
    classes 是一个列表，classes[i]是编号为 ip 的成员的列表
    """
    vectors = []
    for seed in seeds:
        vectors.append(np.array(seed))
    n = len(vectors)
    results = [i for i in range(n)]
    dis = np.zeros([n, n])
    # 求初始的距离矩阵 dis
    for i in range(n):
        for j in range(i):
```

```
            dis[i, j] = dis[j, i] = np.linalg.norm(vectors[i] - vectors[j])
    print("mean = ", dis.mean())
    print("min = ", dis.min())
    print("max = ", dis.max())
    print("threshold = ", threshold)
    # 重复以下迭代过程直到最大距离已经达到了阈值
    # 1. 从距离矩阵中找类间距离最大的两个类
    # 2. 将这两个类合并
    merge_count = 0
    while True:
        i, j = get_element(dis, results, n)
        tmp = dis[i, j]
        if dis[i, j] >= threshold:
            break
        merge(dis, results, n, i, j)
        merge_count += 1
        # if merge_count%10 == 0:
        #     print("已经 merge{}次了,这次 merge 之前的最大类间距离是{}".
        #           format(merge_count, tmp))
    class_index_dict = dict()
    j = 0
    for i in range(n):
        if i == results[i]:
            class_index_dict[i] = j
            j += 1
    classes = [[] for i in range(j)]
    for i in range(n):
        ri = get_root(results, i)
        classes[class_index_dict[ri]].append(i)
    return results, classes


def judge(real_classes, classes):
    """
    聚类评价
    :return: 输出聚类结果的构成 例如聚类成了三个类 1,2,3 输出
```

```
    聚类 1=0.9 真类 1+0.2 的真类 2+0.1 的真类 3
    """
    mtr = np.zeros([len(classes), 3])
    for i in range(len(classes)):
        for j in classes[i]:
            mtr[i, real_classes[j]-1] += 1
    print("""
下面展示分类结果的情况。
行是聚类出来的一个类，列是表示真实的种子的一个类别。
值是个数。例如第一行第一列表示聚类出来的类 1 中实际上是 1 类种子的种子数。
分类结果展示：
""")
    print(mtr)
    a = mtr.sum(axis=0)
    print("种子真实分类各类的种子数目")
    print(a)
    mtr = mtr/a
    print("下面展示的是上面的矩阵转换为占比的情况")
    print(mtr)
    filename = input("请输入你要保存的矩阵图的文件名：\n")
    mtr_to_img(mtr, filename)


def main():
    data = Data()
    data.read_data()
    # 下面的参数用于找出比较好的阈值
    # l = 7
    # r = 11
    # step = 0.25
    # t = l
    # while t < r:
    #     print("T = ", t)
    #     results, classes =
hierarchical_cluster(data.all_vector,threshold=t)
    #     judge(data.all_answer, classes)
```

```
    #     t += step
    #     os.system("pause")
    results, classes = hierarchical_cluster(data.all_vector)
    judge(data.all_answer, classes)


if __name__ == '__main__':
    main()
```

4.k-Means 聚类算法

```
from prepare_data import Data
import numpy as np
import hierarchical_clustering as hc


def get_nearest_index(centers, p):
    """
    返回离点 p 最近的中心点 centers[i]的下标 i
    :param centers: 中心点的数组。
    :param param: 点 p
    :return: i
    """
    x, dis = None, None
    for i in range(len(centers)):
        d = np.linalg.norm(centers[i] - p)
        if x is None or dis > d:
            x = i
            dis = d
    assert(x is not None)
    return x


def k_means_cluster(k, now_centers, seeds, enough_less_dis=1e-8,
max_iterator_count=1e6):
```

```
    """
    kj均值聚类算法
    :param k: k
    :param now_centers: 初始的k个类中心
    :param seeds: 种子的向量数组（list）
    :param enough_less_dis: 足够小的距离阈值，距离小于该阈值认为点重合
    :param max_iterator_count: 最大迭代次数
    :return: centers, group, classes
    group是一个列表。group[i]表示seeds[i]的分类编号（从0开始）。
    classes是列表，classes[i]是存储了所有类别i的种子的下标。
    """
    # 根据最小距离原则将点划分到k个聚类中心所代表的类中
    # 对划分出来的k个类，计算类的“理想”中心（均值向量）
    # 如果现实和“理想”全部重合了，那么停止迭代
    # 由于都是实数，所以是否“重合”不能直接各个维度直接比较。
    # 本程序采用欧式距离度量，当小于一定的阈值即认为重合
    # 同时采取最大迭代次数
    vectors = []
    for seed in seeds:
        vectors.append(np.array(seed))
    n = len(vectors)
    m = len(seeds[0])  # 种子向量的维度数目
    results = [0] * n
    iterator_count = 0
    is_same = False
    while iterator_count < max_iterator_count:
        # 计算每个点所属的类别
        for i in range(n):
            results[i] = get_nearest_index(now_centers, vectors[i])
        # 求类中的向量和及向量个数
        next_centers = [np.zeros(m) for i in range(k)]
        count = [0] * k
        for i in range(n):
            next_centers[results[i]] += vectors[i]
            count[results[i]] += 1
        # 计算新的聚类中心并判断是否重合
```

```
            is_same = True
            for i in range(k):
                next_centers[i] /= count[i]  # 理论上应该不至于划分出来某个类一个点都没不可能发生除 0 错误
                d = np.linalg.norm(now_centers[i] - next_centers[i])
                if d >= enough_less_dis:
                    is_same = False
            if is_same:
                break
            now_centers = next_centers
            iterator_count += 1
        if is_same:
            print("迭代次数是"+str(iterator_count))
            print("新的类中心已经和旧的类中心重合了")
        else:
            print("已经迭代了"+str(max_iterator_count)+"次了，但依旧没有达到新的类中心和旧的类中心重合的情况。")
        classes = [[] for i in range(k)]
        for i in range(n):
            classes[results[i]].append(i)
        return now_centers, results, classes


def main():
    data = Data()
    data.read_data()
    k = 3
    # 选取 k 个点作为初始的 k 个类的中心
    # init_center_indexes = data.get_m_lucky_index(len(data.all_answer), k)
    # init_center_indexes = [30+i*70 for i in range(k)]
    init_center_indexes = [i for i in range(k)]
    # 获取 k 个中心点的向量
    init_centers = [np.array(data.all_vector[i]) for i in init_center_indexes]

    centers, results, classes = \
```

```
        k_means_cluster(
            k, init_centers,data.all_vector)
    # print("最终确定的"+str(k)+"聚类中心如下所示：")
    # print(centers)
    hc.judge(data.all_answer, classes)


if __name__ == '__main__':
    main()
```