暨南大學
JINAN UNIVERSITY

本科实验报告

课 程 名 称：数据挖掘

课 程 编 号：08060116

学 生 姓 名：郑泽琪

学　　号：2016051514

学　　院：信息科学技术学院

系：计算机科学系

专　　业：计算机科学与技术

指 导 教 师：刘波

教 师 单 位：暨南大学

开 课 时 间：2018 ~ 2019 学年度第二学期

暨南大学教务处

2019 年 6 月 12 日

暨南大学本科实验报告专用纸

数据挖掘 **课程实验项目目录**

学生姓名： 学号：

序号	实验项目编号	实验项目名称	*实验项目类型	成绩	指导教师
1	01	**关联分析**	综合性		刘波
2	02	**分类**	综合性		刘波
3	03	**聚类**	综合性		刘波

*实验项目类型：演示性、验证性、综合性、设计性实验。

*此表由学生按顺序填写。

暨南大学本科实验报告专用纸

课程名称 数据挖掘 成绩评定

实验项目名称 关联分析 指导教师 刘波

实验项目编号 01 实验项目类型 综合性 实验地点 N116

学生姓名 郑泽琪 学号 2016051514

学院 信息科学技术学院 系 计算机科学系 专业 计算机科学与技术

实验时间 2019 年 6 月 5 日下午～ 6 月 5 日 下 午

一. 实验内容

使用 Apriori 算法，分析一个超市用户购买记录，通过分析用户购买物品，计算出相关的商品关联度。为简化问题，将 200 项购物记录标记为 T1~T00，同时，为了使得算法在短时间内能够得到结果，我们只摘取分析六种不同的商品，分别记为 1, 2, 3, 4, 5, 6。

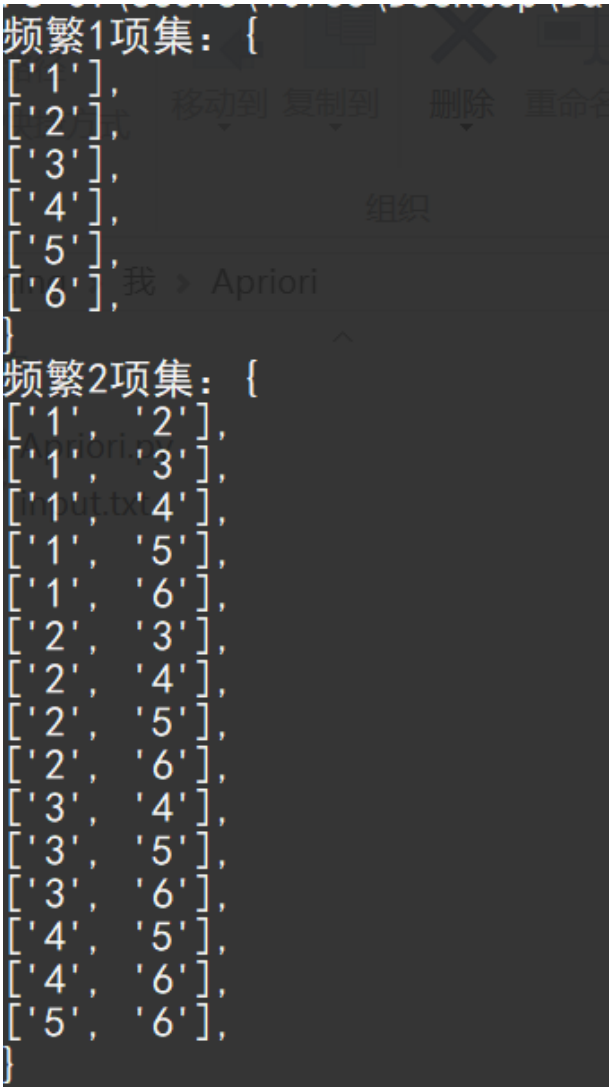
二. 数据样例以及实验结果

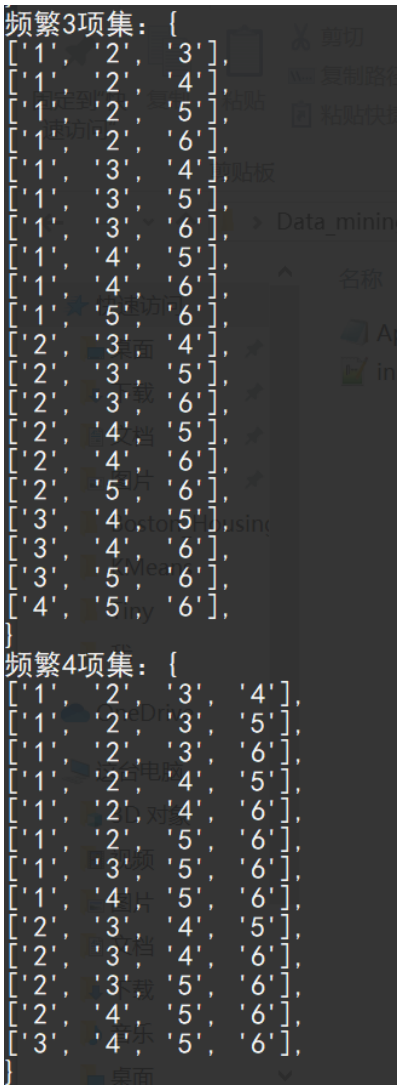
（1）截取前 15 和后 15 条购物记录：

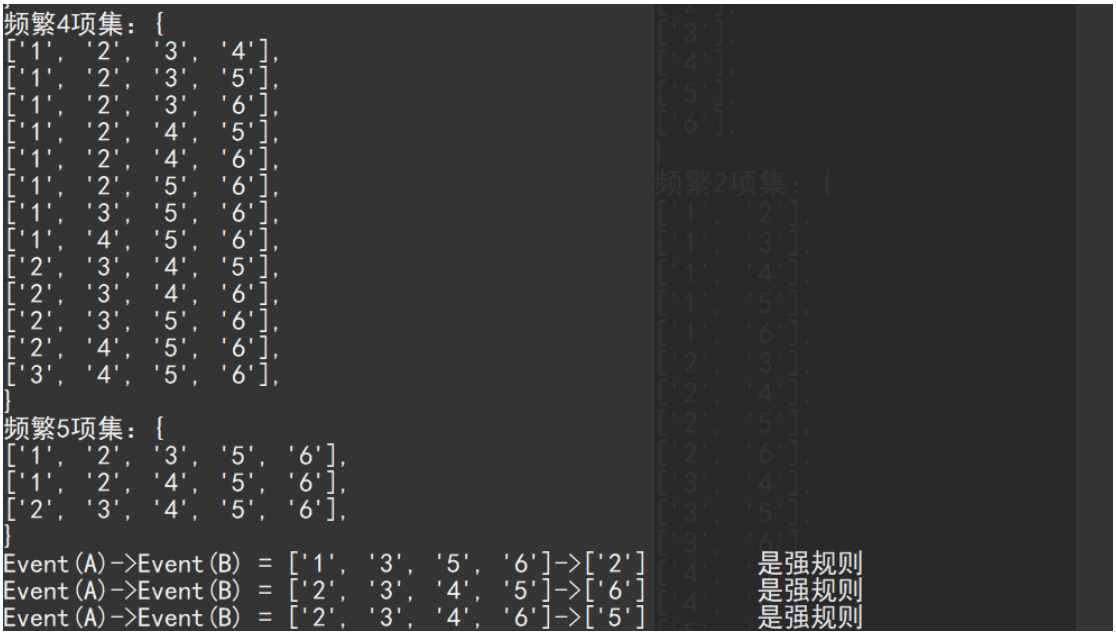
```
1   T1 1 2 5
2   T2 2 4
3   T3 2 3
4   T4 1 2 4
5   T5 1 3
6   T6 2 3
7   T7 1 3
8   T8 1 2 3 5
9   T9 1 2 3
10  T10 1 2 4 5
11  T11 1 2 4 6
12  T12 2 3 5 6
13  T13 1 2 4 6
14  T14 1 3 5
15  T15 2 3
```

```
185  T185 1 2 4
186  T186 1 3 6
187  T187 2 3
188  T188 1 3 5
189  T189 1 2 3 5
190  T190 1 2 3
191  T191 1 2 5
192  T192 2 4
193  T193 2 3
194  T194 1 2 4
195  T195 1 3
196  T196 2 3
197  T197 1 3
198  T198 1 2 3 5
199  T199 1 2 3 6
200  T200 1 2 5 5 6
```

（2）实验结果：


频繁1项集：{
['1'],
['2'],
['3'],
['4'],
['5'],
['6'],
}
频繁2项集：{
['1', '2'],
['1', '3'],
['1', '4'],
['1', '5'],
['1', '6'],
['2', '3'],
['2', '4'],
['2', '5'],
['2', '6'],
['3', '4'],
['3', '5'],
['3', '6'],
['4', '5'],
['4', '6'],
['5', '6'],
}


频繁3项集：{
['1', '2', '3'],
['1', '2', '4'],
['1', '2', '5'],
['1', '2', '6'],
['1', '3', '4'],
['1', '3', '5'],
['1', '3', '6'],
['1', '4', '5'],
['1', '4', '6'],
['1', '5', '6'],
['2', '3', '4'],
['2', '3', '5'],
['2', '3', '6'],
['2', '4', '5'],
['2', '4', '6'],
['2', '5', '6'],
['3', '4', '5'],
['3', '4', '6'],
['3', '5', '6'],
['4', '5', '6'],
}
频繁4项集：{
['1', '2', '3', '4'],
['1', '2', '3', '5'],
['1', '2', '3', '6'],
['1', '2', '4', '5'],
['1', '2', '4', '6'],
['1', '2', '5', '6'],
['1', '3', '5', '6'],
['1', '4', '5', '6'],
['2', '3', '4', '5'],
['2', '3', '4', '6'],
['2', '3', '5', '6'],
['2', '4', '5', '6'],
['3', '4', '5', '6'],
}


频繁4项集：{
['1', '2', '3', '4'],
['1', '2', '3', '5'],
['1', '2', '3', '6'],
['1', '2', '4', '5'],
['1', '2', '4', '6'],
['1', '2', '5', '6'],
['1', '3', '5', '6'],
['1', '4', '5', '6'],
['2', '3', '4', '5'],
['2', '3', '4', '6'],
['2', '3', '5', '6'],
['2', '4', '5', '6'],
['3', '4', '5', '6'],
}
频繁5项集：{
['1', '2', '3', '5', '6'],
['1', '2', '4', '5', '6'],
['2', '3', '4', '5', '6'],
}
Event(A)->Event(B) = ['1', '3', '5', '6']->['2'] 是强规则
Event(A)->Event(B) = ['2', '3', '4', '5']->['6'] 是强规则
Event(A)->Event(B) = ['2', '3', '4', '6']->['5'] 是强规则

三.源代码

```
# -*- coding:utf-8 -*-

# @Author zzq
import itertools


class FrequentItem:
    # 频繁项集的集合 ID
    idArray = []
    # 频繁项集的支持度计数
    count = 0
    # 频繁项集的长度，1 项集或是 2 项集，亦或是 3 项集
    length = 0

    def __init__(self, id_array, item_count):
        self.idArray = id_array
        self.count = item_count
        self.length = len(id_array)

    def get_id_array(self):
        return self.idArray

    def set_id_array(self, id_array):
        self.idArray = id_array

    def get_count(self):
        return self.count

    def set_count(self, item_count):
        self.count = item_count

    def get_length(self):
        return self.length

    def set_length(self, array_length):
        self.length = array_length


class AprioriTool:
    # 最小支持度计数
    min_support_count = 0
    # 文件地址
```

```
file_path = ''
# 每个事务中的商品 ID
total_goods_ids = []
# 过程中计算出来的所有频繁项集列表
result_item = []
# 过程中计算出来频繁项集的 ID 集合
result_item_id = []
# 剪枝后的新频繁项集
new_item = []

def __init__(self, path, support_count):
    self.path = path
    self.min_support_count = support_count
    self.read_data_file(file_path)

# 从文件中读取数据
def read_data_file(self, path):
    data_array = []

    file = open(path)
    for line in file:
        if line != '':
            temp_array = line.split()
            data_array.append(temp_array)
    file.close()

    for array in data_array:
        temp = array[1:].copy()
        # 将事务 ID 加入列表
        self.total_goods_ids.append(temp)

# 项集连接步
def compute_link(self):
    # 当前已经进行连接运算到几项集,开始时就是 1 项集
    current_num = 1
    # 初始化列表
    init_list = []
    temp_item = {}
    # 商品 id 的种类
    id_type = []
    for array in self.total_goods_ids:
        for goods_id in array:
            if goods_id in id_type:
                temp_value = temp_item[goods_id]
```

```
            temp_item[goods_id] = temp_value + 1
        else:
            id_type.append(goods_id)
            temp_item[goods_id] = 1
# 得到初始频繁项集，并将初始频繁项集存入 result_item， result_item_id 中
id_type.sort()
for goods_id in id_type:
    temp_id_array = []
    temp_id_array.append(goods_id)
    temp_node = FrequentItem(temp_id_array, temp_item[goods_id])
    # 根据最小支持度，判断初始项集是否频繁
    if temp_node.get_count() >= self.min_support_count:
        init_list.append(temp_node)
        self.result_item.append(temp_node)
        self.result_item_id.append(temp_id_array)

# 连接计算的终止数，k 项集必须算到 k-1 子项集为止
end_num = len(init_list) - 1
while current_num < end_num:
    result_container = []
    i = 0
    while i < len(init_list) - 1:
        array1 = init_list[i].get_id_array()
        j = i + 1
        while j < len(init_list):
            array2 = init_list[j].get_id_array()

            temp_ids = []
            # 连接两个 array，比较对应位置值是否相等
            k = 0
            while k < len(array1):
                if array1[k] == array2[k]:
                    temp_ids.append(array1[k])
                else:
                    temp_ids.append(array1[k])
                    temp_ids.append(array2[k])
                k += 1

            is_contain = False
            if len(temp_ids) == len(array1) + 1:
                is_contain = self.is_id_array_contains(
                    result_container, temp_ids)
                if not is_contain:
                    result_container.append(temp_ids)
```

```
                j += 1
            i += 1
        init_list = self.pruning(result_container)

        # 将每步连接得到的频繁项集存入 result_item, result_item_id
        for item in init_list:
            self.result_item.append(item)
            self.result_item_id.append(item.get_id_array())

        current_num += 1

    # 输出频繁项集
    l = 1
    while l <= current_num:
        print("频繁" + str(l) + "项集：{")
        for frequent_set in self.result_item:
            if frequent_set.get_length() == l:
                print(str(frequent_set.get_id_array()) + ", ")
        print("}")
        l += 1

# 判断列表结果中是否已经包含此数组
@staticmethod
def is_id_array_contains(container, array):
    is_contain = True
    if len(container) == 0:
        is_contain = False
        return is_contain

    for element in container:
        if len(element) != len(array):
            continue

        is_contain = True
        i = 0
        while i < len(element):
            if element[i] not in array:
                is_contain = False
                break
            i += 1
        # 如果判断出包含，直接退出
        if is_contain:
            break
```

```
    return is_contain

# 对频繁项集做剪枝步骤，必须保证新的频繁项集的子项集也必须是频繁项集
def pruning(self, middle_result_ids):
    # 剪枝后的新频繁项集
    self.new_item = []
    # 需要删除的不符合要求的项集 id
    delete_id_array = []
    # 判断子项集是否是频繁项集
    is_contain = True
    for array in middle_result_ids:
        i_index = 0
        while i_index < len(array):
            is_contain = True
            temp = []
            j = 0
            while j < len(array):
                if j != i_index:
                    temp.append(array[j])
                j += 1

            if not self.is_id_array_contains(self.result_item_id, temp):
                is_contain = False
                break
            i_index += 1

            if not is_contain:
                delete_id_array.append(array)

    # 移除子项集不是频繁项集的 id 组合
    for array in delete_id_array:
        del middle_result_ids[middle_result_ids.index(array)]

    # 移除不满足最小支持度的 id 组合
    for array1 in middle_result_ids:
        temp_num = 0
        for array2 in self.total_goods_ids:
            if self.is_str_array_contain(array2, array1):
                temp_num += 1
        # 将支持度 >= 最小支持度的频繁项集加入 result_item, result_item_id
        if temp_num >= self.min_support_count:
            temp_item = FrequentItem(array1, temp_num)
            self.new_item.append(temp_item)
```

```
    return self.new_item

# 数组 array2 是否包含于 array1 中，不需要完全一样
@staticmethod
def is_str_array_contain(array1, array2):
    is_contain = True
    for element in array2:
        if element not in array1:
            is_contain = False
    return is_contain

# 根据产生的频繁项集输出关联规则
# @param min_conf
#        最小置信度阈值
def print_attach_rule(self, min_conf):
    self.compute_link()
    # 计算得到的频繁项集 id_array 的最大长度
    max_length = len(self.result_item_id[-1])
    for item in self.result_item:
        item_array = item.get_id_array()
        if len(item_array) == max_length:
            l = max_length - 1
            # 最长频繁项集的子集
            sub_set = []
            while l > 0:
                for sub_item in itertools.combinations(item_array, l):
                    sub_set.append(list(sub_item))
                l -= 1
            # A->B, conf(A->B) = P(B|A) = P(AB)/P(A)
            for sub_a in sub_set:
                pos_ab = self.result_item_id.index(item_array)
                pos_a = self.result_item_id.index(sub_a)
                sub_ab_item = self.result_item[pos_ab]
                sub_a_item = self.result_item[pos_a]
                conf_ab = sub_ab_item.get_count() / sub_a_item.get_count()
                sub_b = list(set(item_array).difference(set(sub_a)))
                if conf_ab > min_conf:
                    print("Event(A)->Event(B) = "
                          + str(sub_a) + "->" + str(sub_b)
                          + "\t" + "是强规则")
                else:
                    # print("Event(A)->Event(B) = "
                    #       + str(sub_a) + "->" + str(sub_b)
                    #       + "\t" + "不是强规则")
```

```
                    pass

if __name__ == '__main__':
    file_path = './input.txt'
    tool = AprioriTool(file_path, 2)
    tool.print_attach_rule(0.7)
```

暨南大学本科实验报告专用纸

课程名称 数据挖掘 成绩评定

实验项目名称 分类 指导教师 刘波

实验项目编号 02 实验项目类型 综合性 实验地点 N116

学生姓名 郑泽琪 学号 2016051514

学院 信息科学技术学院 系 计算机科学系 专业 计算机科学与技术

实验时间 2019 年 6 月 5 日下午～ 6 月 5 日 下 午

一．实验内容

海伦一直在使用在线约会网站寻找合适自己的约会对象，经过一番总结，海伦整理了以下数据，希望我们的分类软件可以更好地帮助她将匹配对象划分到确切的分类中。数据如下：截取前 10 行：

```
40920	8.326976	0.953952	largeDoses
14488	7.153469	1.673904	smallDoses
26052	1.441871	0.805124	didntLike
75136	13.147394	0.428964	didntLike
38344	1.669788	0.134296	didntLike
72993	10.141740	1.032955	didntLike
35948	6.830792	1.213192	largeDoses
42666	13.276369	0.543880	largeDoses
67497	8.631577	0.749278	didntLike
35483	12.273169	1.508053	largeDoses
```

第一列：每年获得的飞行常客里程数

第二列：玩视频游戏所耗时间百分比

第三列：每周消费的冰淇淋公升数

第四列：海伦对数据的分类

largeDoses 表示对海伦极具魅力的人

smallDoses 表示对海伦魅力一般的人

didntlike 表示海伦不喜欢的人
为简化代码处理，我们将标签数字化：
largeDoses：用 3 表示

smallDoses：用 2 表示

didntlike：用 1 表示

二. 选用的算法与实验结果：

算法上选用了 knn 最后统计错误率
实验结果：

```
    print("error rate:",(test_data_len - index)/test_data_len)
    print(counts)
    print(test_data_len)


test()
```

```
error rate: 0.315
137
200
```

三. 源代码

使用 **jupyter notebook**

Block 1:

```
import numpy as np
from operator import *
import operator
import os
```

Block 2:

```
def KNN(data,rawdata,label,k):
    '''
    data:未知数据
    rawdata:数据集
    label:标签
    k:选取前 k 个数据
    '''
    m = rawdata.shape[0]
    differ = (np.tile(data,(m,1))-rawdata)**2
    differ_matrix = np.sum(differ,axis=0)
    distance = differ_matrix**0.2

    ##返回获得距离的索引
    sortlabelindex = distance.argsort()

    distance_label = {}

    for i in range(k):
        index = label[sortlabelindex[i]]
        distance_label[index] = distance_label.get(index,0)+1

    sorted_distance_label                                                    =
```

```
sorted(distance_label.items(),key=operator.itemgetter(1),reverse=True)
    return sorted_distance_label[0][0]
```

Block 3:

```
    def get_data(file):
        f = open(file)
        lines = f.readlines()
        cols = len(lines)
        data_matirx = np.zeros((cols,3))
        label_matrix = []
        index = 0
        for line in lines:
            line = line.strip().split('\t')
            data_matirx[index,:]=line[0:3]
            label_matrix.append(line[3])
            index=index+1
        return data_matirx,label_matrix

    def label(label):
        new_label = list(set(label))
        new_label.sort(key=label.index)
        items = list(range(0,3))
        label_dict={}
        index=0
        for i in new_label:
            label_dict[i]=index
            index=index+1
        label_new = [label_dict[x] for x in label]
        return label_new


    data_matirx,label_matrix=get_data('./data/datingTestSet.txt')
    label_matrix[0:20]
    classfy = list(set(label_matrix))
    classfy.sort(key=label_matrix.index)
    label=label(label_matrix)
```

Block 4:

```
    import matplotlib.pyplot as plt
    import matplotlib
    def draw_scatter(data,label):
        fig = plt.figure()
        #ax = fig.add_subplot(111)
        plt.scatter(data[:,1],data[:,2],15.0*np.array(label),15.0*np.array(label))
```

```
        plt.xlabel('x')
        plt.ylabel('y')
        plt.show()

    draw_scatter(data_matirx,label)
```

Block 5:

```
    def norm(data):
        min_data = data.min(axis=0)
        max_data = data.max(axis=0)
        delta = np.tile(max_data-min_data,(data.shape[0],1))
        #print(np.tile(min_data,(1000,1)))
        #print(max_data)
        min_data_tile = np.tile(min_data,(data.shape[0],1))
        #min_data_tile = np.tile(min_data,(3,1))
        return((data-min_data_tile)/delta)
    norm(data_matirx)
```

Block 6:

```
from sklearn.utils import shuffle
def test():
    test_ratio =0.2
    data,label=get_data('./data/datingTestSet2.txt')
    #data=norm(data)
    m = data.shape[0]
    test_data_len =int(test_ratio*m)
    index =0
    counts=0
    for i in range(test_data_len):

        predict = KNN(data[i,:],data[test_data_len:m,:],label[test_data_len:m],3)
        # print('predict label is {} true label is {}'.format(predict,label[i]))
        if predict!=label[i]:
            index=index+1
            counts+=1
    print("error rate:",(test_data_len - index)/test_data_len)
    print(counts)
    print(test_data_len)

test()
```

暨南大学本科实验报告专用纸

课程名称 数据挖掘 成绩评定

实验项目名称 聚类 指导教师 刘波

实验项目编号 03 实验项目类型 综合性 实验地点 N116

学生姓名 郑泽琪 学号 2016051514

学院 信息科学技术学院 系 计算机科学系 专业 计算机科学与技术

实验时间 2019 年 6 月 5 日下午～ 6 月 5 日 下 午

一．实验内容

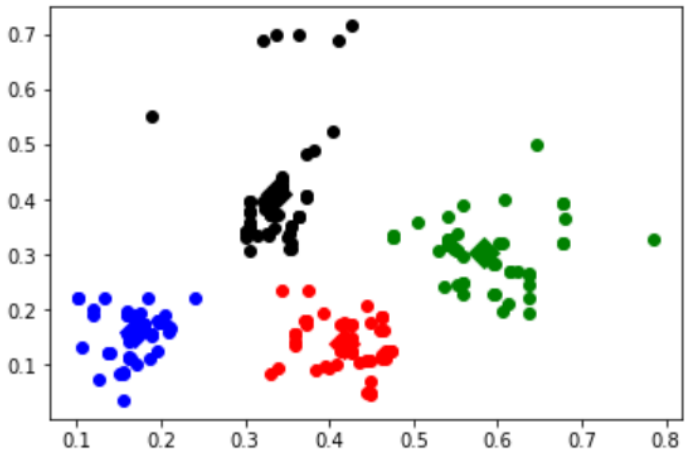
本实验简化现实生活中一些拥有两个维度的实例，简化为 x 和 y，通过对数据进行聚类分析，分析不同类型的质心，以便后期确认其他数据所属类别。

二. 数据样例以及实验结果

（1）数据样例：(数据为 200 条，仅截取前 10 行)

```
0.209752 0.169062
0.160104 0.190186
0.17048 0.191011
0.161867 0.111339
0.188379 0.155126
0.155035 0.0835798
0.17886 0.170226
0.139611 0.120509
0.101838 0.220112
0.196831 0.178135
```

（2） 实验结果：



三．源代码：

本代码在 Jupyter Notebook 上运行：

Block 1:

```
#导入数据
def load_dataset(file_path):
    data = []
    for line in open(file_path):
        line_tmp = line.split()
        x = float(line_tmp[0])
        y = float(line_tmp[1])
        point = [x,y]
        data.append(point)
    return data
```

Block 2：

```
#作图
import matplotlib.pyplot as plt

def show(dataSet, k, centroids, clusterAssment):
    from matplotlib import pyplot as plt
    numSamples, dim = dataSet.shape
    mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr']
    for i in range(numSamples):
        markIndex = int(clusterAssment[i, 0])
        plt.plot(dataSet[i, 0], dataSet[i, 1], mark[markIndex])
    mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb']
    for i in range(k):
        plt.plot(centroids[i, 0], centroids[i, 1], mark[i], markersize = 12)
    plt.show()
```

Block 3:

```
#K-means 核心算法
from numpy import *
import numpy as np

#随机生成初始 k 个质心
def create_cent(dataSet, k):
    n = shape(dataSet)[1] #  矩阵列长度
    centroids = mat(zeros((k,n))) #生成给定形状和类型的用 0 填充的矩阵
    #随机生成 k 个质心
    for j in range(n):
        minJ = min(dataSet[:,j])
        rangeJ = float(max(array(dataSet)[:,j]) - minJ)
```

```
            centroids[:,j] = minJ + rangeJ * random.rand(k,1)
        return centroids

#计算欧几里得距离
def dist_euclidean(vector_a, vector_b):
        return sqrt(sum(power(vector_a - vector_b, 2)))

def kMeans(dataSet, k, distMeas=dist_euclidean, create_cent=create_cent):
        m = shape(dataSet)[0] #  矩阵行长度
        clusterAssment = mat(zeros((m,2))) #生成给定形状和类型的用 0 填充的矩阵
        centroids = create_cent(dataSet, k) #随机生成初始 k 个质心
        clusterChanged = True
        while clusterChanged:
            clusterChanged = False
            for i in range(m): #循环每个点，以分配到最近的质心
                minDist = inf
                minIndex = -1
                for j in range(k): #循环每质心，找到距离 i 最近的那一个
                    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
            for cent in range(k):#重新计算质心
                ptsInClust = dataSet[nonzero(clusterAssment[:,0].A==cent)[0]] #重新给每个质心
分派点
                centroids[cent,:] = mean(ptsInClust, axis=0)
        return centroids, clusterAssment
```

Block 4:

```
from numpy import *
FILE_DIR = "./KMeans.txt"
k = 4

dataMat = mat(load_dataset(FILE_DIR))
myCentroids, clustAssing= kMeans(dataMat,k)
print("myCentroids:")
print(myCentroids)
print("clustAssing:")
print(clustAssing)
show(dataMat, k, myCentroids, clustAssing)
```