

数据挖掘实验报告

|  |  |
| --- | --- |
| 院 别 | 管理学院 |
| 专业名称 | 电子商务 |
| 班级学号 | 160111 |
| 学生姓名 | 杨一鸣 |
| 指导教师 | 刘冰玉 |

**2018**年**11**月**23**日

# 实验一 分类算法的实现

## 1、实验目的

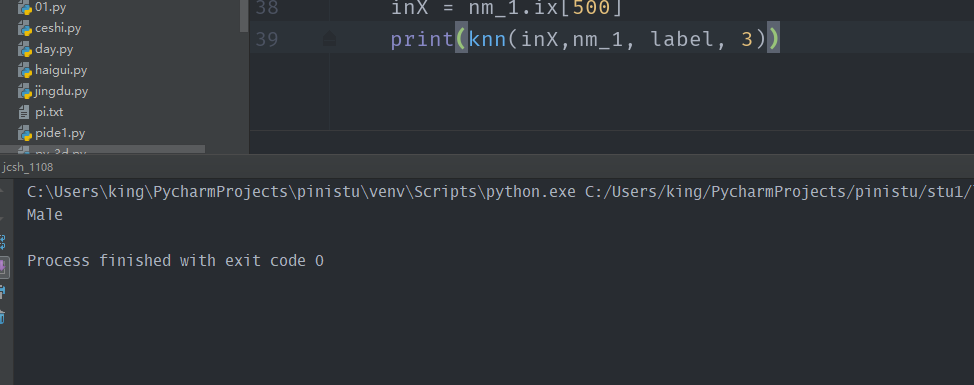
1. 掌握分类算法的思想，尤其是决策树算法、朴素贝叶斯算法。
2. 掌握 PYTHON 语言分类算法库

## 2、实验内容与要求

1. 对给定数据集进行整理，对数据集中的空缺值进行处理
2. 利用某种分类算法对数据集进行分类
3. 对分类结果进行评价，选择某种评价指标对分类结果进行评价

## 3、实验程序

import pandas as pd  
import numpy as np  
  
def getdata(path):  
 data = pd.read\_csv(path, header=0)  
 #print(data)  
 #character = data.iloc[:, 1]  
 #print(character)  
 character = data.drop(['S'],axis=1)  
 chara\_max = character.max()  
 chara\_min = character.min()  
 chara\_range = chara\_max - chara\_min  
 normal\_chara = (character - chara\_min) / chara\_range  
 #print(normal\_chara.head(5))  
 return normal\_chara  
  
def getdat(path):  
 data = pd.read\_csv(path, header=0)  
 label = data.iloc[:, 1]  
 return label  
  
def knn(inX, normal\_chara, label, k):  
 data\_sub = normal\_chara - inX  
 data\_square = data\_sub.applymap(np.square)  
 data\_sum = data\_square.sum(axis=1)  
 data\_sqrt = data\_sum.map(np.sqrt)  
 dis\_sort = data\_sqrt.argsort()  
 k\_label = label[dis\_sort[:k]]  
 label\_sort = k\_label.value\_counts()  
 res\_label = label\_sort.index[0]  
 return res\_label  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 label = getdat('.\indian.csv')  
 nm\_1 = getdata('.\indian.csv')  
 #print(label)  
 nm\_2 = nm\_1.iloc[0:400,:]  
 inX = nm\_1.ix[500]  
 print(knn(inX,nm\_1, label, 3))



**测试备注**

**为了方便处理。提前处理了数据集，增加了索引项。以性别为标签进行分类，随机抽取400个做训练集，随机选出一个进行分类，分类结果如上。**

# 实验二 关联规则算法的实现

主窗体

菜单条

工具条

绘图面板

## 1、实验目的

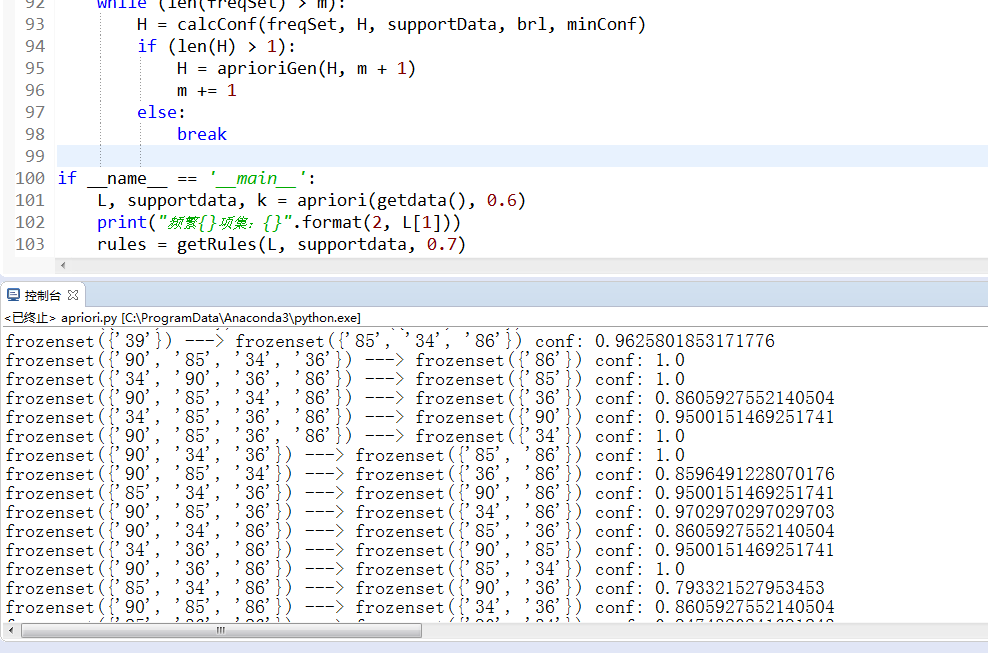
1. 掌握关联规则算法的思想，尤其是 aprior 算法
2. 掌握 PYTHON 语言关联规则算法库

## 2、实验内容与要求

1. 对给定数据集进行整理
2. 利用某种关联规则算法对数据集进行关联规则分析
3. 对结果进行评价，选择某种评价指标对实验结果进行评价

## 3、实验程序

import operator  
  
def getdata():  
 mushDatSet = [line.split() for line in open("mushroom.dat").readlines()]  
 return mushDatSet  
  
def createc1(data):  
 c1 = []  
 aa = set()  
 bb = []  
 for transaction in data:  
 for item in transaction:  
 if item not in c1:  
 c1.append(item)  
 c1.sort()  
 for i in c1:  
 aa.add(i)  
 bb.append(frozenset(aa))  
 aa.remove(i)  
 return bb  
  
def scanD(ck, d, minsuport):  
 ssCnt = {}  
 for item in d:  
 for can in ck:  
 if can.issubset(item):  
 if can in ssCnt.keys():  
 ssCnt[can] += 1  
 else:  
 ssCnt[can] = 1  
 relist = []  
 supportdata = {}  
 for key in ssCnt:  
 support = ssCnt[key] / int(len(d))  
 if support >= minsuport:  
 supportdata[key] = support  
 relist.append(key)  
 return relist, supportdata  
  
def aprioriGen(Lk, k):  
 relist = []  
 for i in range(len(Lk)):  
 for j in range(i + 1, len(Lk)):  
 l1 = list(Lk[i])[:k - 2]  
 l2 = list(Lk[j])[:k - 2]  
 if operator.eq(l1, l2):  
 relist.append(Lk[i] | Lk[j])  
 return relist  
  
def apriori(data, minsupport):  
 c1 = createc1(data)  
 d = list(map(set, data))  
 l1, supportdata = scanD(c1, d, minsupport)  
 L = [l1]  
 k = 2  
 while (len(L[k - 2]) > 0):  
 ck = aprioriGen(L[k - 2], k)  
 lk, supk = scanD(ck, d, minsupport)  
 supportdata.update(supk)  
 L.append(lk)  
 k += 1  
 return L, supportdata, k - 1  
  
def getRules(L, supportData, minConf):  
 rulelist = []  
 for i in range(1, len(L)):  
 for freqSet in L[i]:  
 freqSet2 = list(freqSet)  
 H1 = []  
 ff = set()  
 for item in freqSet2:  
 ff.add(item)  
 H1.append(frozenset(ff))  
 ff.clear()  
 rulesFromConseq(freqSet, H1, supportData, rulelist, minConf)  
 return rulelist  
  
def calcConf(freqSet, H, supportData, brl, minConf):  
 prunedH = []  
 for conseq in H:  
 conf = supportData[freqSet] / supportData[freqSet - conseq]  
 if conf > minConf:  
 print(freqSet - conseq, "--->", conseq, "conf:", conf)  
 brl.append((freqSet - conseq, conseq, conf))  
 prunedH.append((conseq))  
 return prunedH  
  
def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):  
 m = len(H[0])  
 while (len(freqSet) > m):  
 H = calcConf(freqSet, H, supportData, brl, minConf)  
 if (len(H) > 1):  
 H = aprioriGen(H, m + 1)  
 m += 1  
 else:  
 break  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 L, supportdata, k = apriori(getdata(), 0.6)  
 print("频繁{}项集：{}".format(2, L[1]))  
 rules = getRules(L, supportdata, 0.7)



# 实验三 聚类算法的实现

主窗体

菜单条

工具条

绘图面板

## 1、实验目的

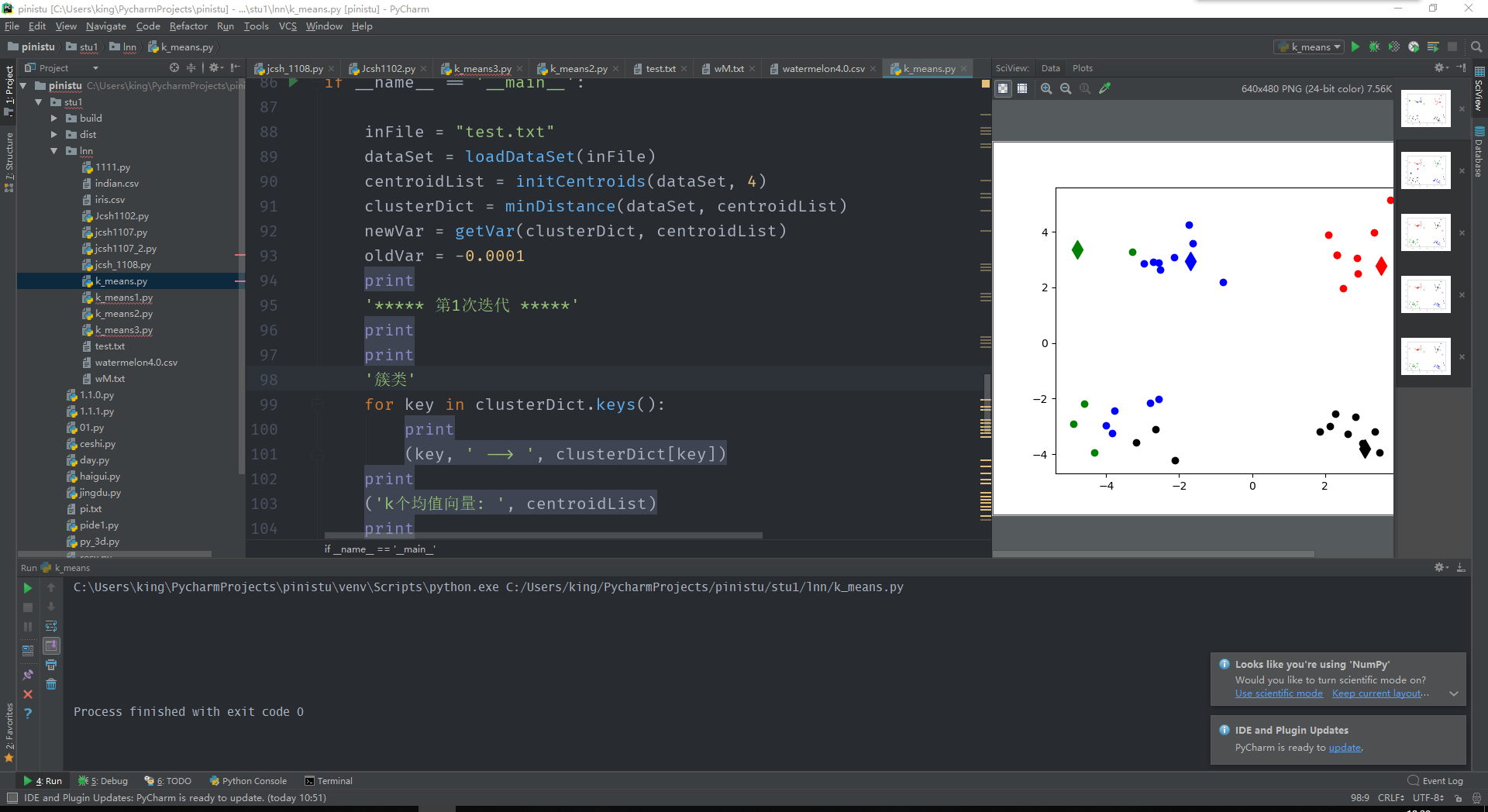
1. 掌握聚类算法的思想，尤其是 k-means,k 中心点算法。
2. 掌握 PYTHON 语言聚类算法库

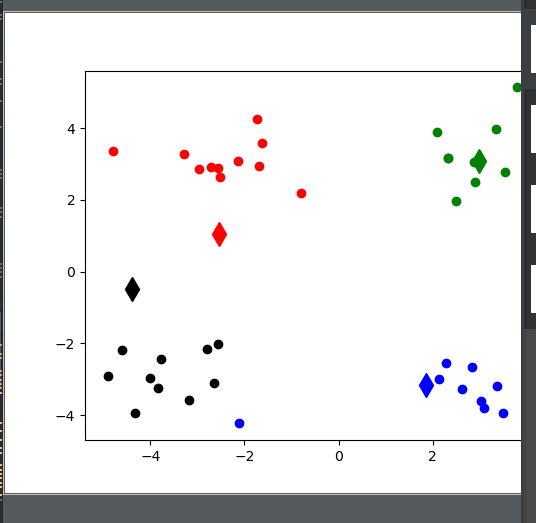
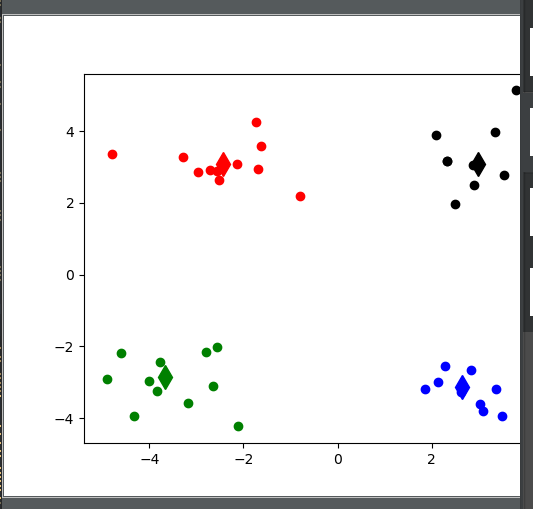
## 2、实验内容与要求

1. 对给定数据集进行整理
2. 利用某种聚类算法对数据集进行聚类分析
3. 对结果进行评价，选择某种评价指标对实验结果进行评价

## 3、实验程序

import numpy  
import pandas as pd  
import matplotlib.pyplot as plt  
  
  
def calcuDistance(vec1, vec2):  
 return numpy.sqrt(numpy.sum(numpy.square(vec1 - vec2)))  
  
def loadDataSet(path):  
 dataf = pd.read\_csv(path, header=0)  
 data = dataf.drop(['id'],axis=1)  
 return data  
  
def initCentroids(data,k):  
 cendata = data.iloc[0:k,:]  
 return cendata  
  
def initData(data,k):  
 data =data.iloc[k+1: ,:]  
 return data  
  
  
  
def minDistance(dataSet, centroidList):  
 clusterDict = dict()  
 for item in dataSet:  
 vec1 = numpy.array(item)  
 flag = 0  
 minDis = float("inf")  
  
 for i in range(len(centroidList)):  
 vec2 = numpy.array(centroidList[i])  
 distance = calcuDistance(vec1, vec2)  
 if distance < minDis:  
 minDis = distance  
 flag = i  
  
 if flag not in clusterDict.keys():  
 clusterDict[flag] = list()  
 clusterDict[flag].append(item)  
  
 return clusterDict  
  
  
def getCentroids(clusterDict):  
 centroidList = list()  
 for key in clusterDict.keys():  
 centroid = numpy.mean(numpy.array(clusterDict[key]), axis=0)  
 centroidList.append(centroid)  
  
 return numpy.array(centroidList).tolist()  
  
  
def getVar(clusterDict, centroidList):  
 sum = 0.0  
 for key in clusterDict.keys():  
 vec1 = numpy.array(centroidList[key])  
 distance = 0.0  
 for item in clusterDict[key]:  
 vec2 = numpy.array(item)  
 distance += calcuDistance(vec1, vec2)  
 sum += distance  
  
 return sum  
  
  
def showCluster(centroidList, clusterDict):  
 colorMark = ['or', 'ob', 'og', 'ok', 'oy', 'ow']  
 centroidMark = ['dr', 'db', 'dg', 'dk', 'dy', 'dw']  
 for key in clusterDict.keys():  
 plt.plot(centroidList[key][0], centroidList[key][1], centroidMark[key], markersize=12)  
 for item in clusterDict[key]:  
 plt.plot(item[0], item[1], colorMark[key])  
  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 #数据处理  
 path = ".\watermelon4.0.csv"  
 dataSet = loadDataSet(path)  
 centroidList = initCentroids(dataSet, 4)  
 clusterDict = minDistance(dataSet, centroidList)  
 newVar = getVar(clusterDict, centroidList)  
 oldVar = -0.0001  
 #第一次  
 for key in clusterDict.keys():  
 print(key, ' --> ', clusterDict[key])  
 print('k个均值向量: ', centroidList)  
 print('平均均方误差: ', newVar)  
 print()  
 showCluster(centroidList, clusterDict)  
  
 k = 2  
 while abs(newVar - oldVar) >= 0.0001:  
 centroidList = getCentroids(clusterDict)  
 clusterDict = minDistance(dataSet, centroidList)  
 oldVar = newVar  
 newVar = getVar(clusterDict, centroidList)  
  
 #迭代  
 for key in clusterDict.keys():  
 print (key, ' --> ', clusterDict[key])  
 print('k个均值向量: ', centroidList)  
 print ('平均均方误差: ', newVar)  
 print()  
 showCluster(centroidList, clusterDict)  
 k += 1

****



**测试备注**

**使用西瓜数据集，取出前四个，视为聚类中心。上图描述了聚类的过程。**