数据挖掘

姓名： 易韬邦

班级： 191141

学号： 20141001788

学院： 计算机学院

专业：计算机科学与技术

教师： 万林

目录

[Lab1:Numeric Data Analysis 2](#_Toc3770)

[Lab2:Density Based Clustering 8](#_Toc26686)

[算法思想 8](#_Toc14035)

[实现代码 8](#_Toc11430)

[结果 10](#_Toc3499)

[Lab3:Decision Tree 11](#_Toc11660)

[算法思想 11](#_Toc9673)

[实现代码 11](#_Toc4966)

[结果 12](#_Toc28100)

# Lab1:Numeric Data Analysis

读取数据文件

fLength = []

fWidth = []

fSize = []

fConc = []

fConc1 = []

fAsym = []

fM3Long = []

fM3Trans = []

fAlpha = []

fDist = []

num=0

filename = 'magic04.txt'

with open(filename) as file\_to\_read:

while (True):

lines = file\_to\_read.readline()

num += 1

if not lines:

break

fLt, fWt, fst, fct, fc1t, fat, fmlt, fmtt, fat, fdt,\

attr = [i for i in lines.split(",")]

fLength.append(float(fLt))

fWidth.append(float(fWt))

fSize.append(float(fst))

fConc.append(float(fct))

fConc1.append(float(fc1t))

fAsym.append(float(fat))

fM3Long.append(float(fmlt))

fM3Trans.append(float(fmtt))

fAlpha.append(float(fat))

fDist.append(float(fdt))

**计算均值向量**

print (sum(fLength)/num)

print (sum(fWidth)/num)

print (sum(fSize)/num)

print (sum(fConc)/num)

print (sum(fConc1)/num)

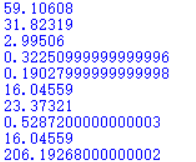
print (sum(fAsym)/num)

print (sum(fM3Long)/num)

print (sum(fM3Trans)/num)

print (sum(fAlpha)/num)

print (sum(fDist)/num)



**计算数据矩阵的属性之间的内积的协方差矩阵**

y = [fLength,fWidth,fSize,fConc,fConc1,

fAsym,fM3Long,fM3Trans,fAlpha,fDist]

k=np.array(y, dtype = float)

r=np.ndarray.dot(k,k.T)

print(np.cov(r))



**计算数据矩阵的样本之间的外积的协方差矩阵**

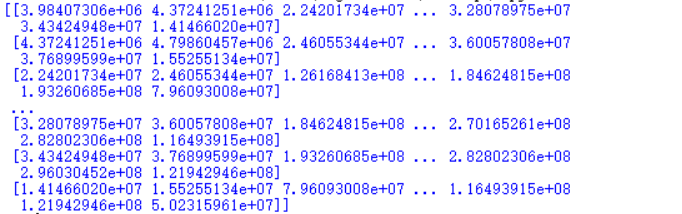
y = [fLength,fWidth,fSize,fConc,fConc1,

fAsym,fM3Long,fM3Trans,fAlpha,fDist]

k=np.array(y, dtype = float)

r=np.outer(k,k)(数据集过大报：MemoryError，以下是前十个数据的计算结果)

print(np.cov(r))



**属性1和2之间的散点图**

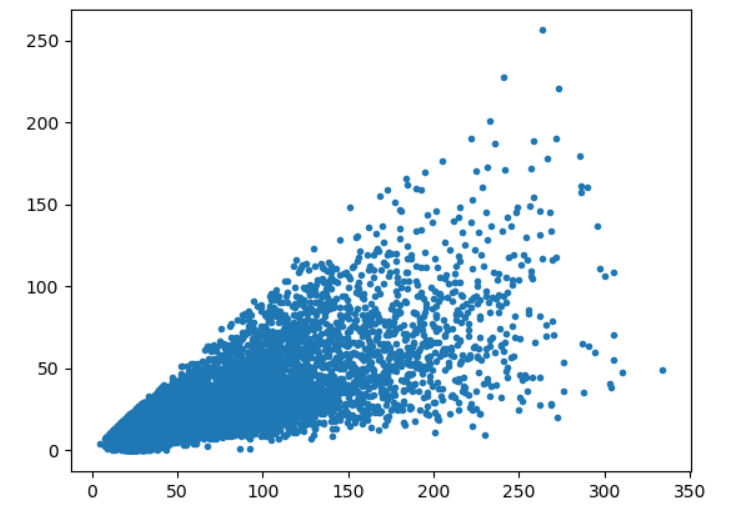
print(np.cov(y))

a=np.array(fLength)

b=np.array(fWidth)

plt.plot(a,b,'.')

plt.show()



**属性1的概率密度**

data = a

mean = data.mean()

std = data.std()

def profun(x,mu,sigma):

pdf = np.exp(-((x - mu)\*\*2)/(2\*sigma\*\*2)) / (sigma \* np.sqrt(2\*np.pi))

return pdf

x = np.arange(-200,200,0.1)

y = profun(x, mean, std)

plt.plot(x,y)

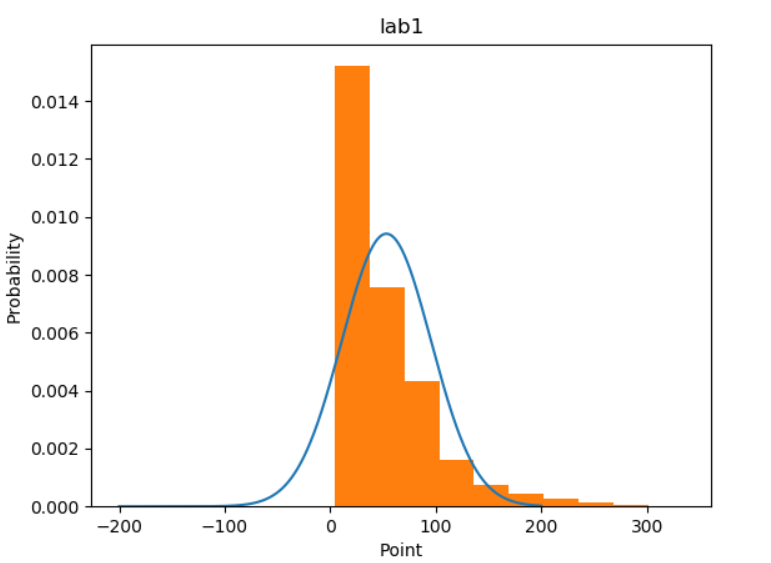
plt.hist(data, bins=10, rwidth=10, density=True)

plt.title('lab1')

plt.xlabel('Point')

plt.ylabel('Probability')

plt.show()



**哪个属性的方差最大，哪个属性的方差最小**

def varifun(a):

array = np.array(a)

var = array.var()

return var

var1=varifun(fLength)

var2=varifun(fWidth)

var3=varifun(fSize)

var4=varifun(fConc)

var5=varifun(fConc1)

var6=varifun(fAsym)

var7=varifun(fM3Long)

var8=varifun(fM3Trans)

var9=varifun(fAlpha)

var10=varifun(fDist)

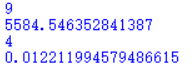
c1=[var1,var2,var3,var4,var5,var6,var7,var8,var9,var10]

print (c1.index(max(c1)))

print (c1[c1.index(max(c1))])

print (c1.index(min(c1)))

print (c1[c1.index(min(c1))])



**哪一对属性的协方差最大，哪一对属性的协方差最小**

def covfun(a):

array = np.array(a)

cov = np.cov(array)

return cov

cov1=covfun(fLength)

cov2=covfun(fWidth)

cov3=covfun(fSize)

cov4=covfun(fConc)

cov5=covfun(fConc1)

cov6=covfun(fAsym)

cov7=covfun(fM3Long)

cov8=covfun(fM3Trans)

cov9=covfun(fAlpha)

cov10=covfun(fDist)

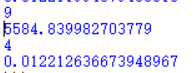
c2=[cov1,cov2,cov3,cov4,cov5,cov6,cov7,cov8,cov9,cov10]

print (c2.index(max(c2)))

print (c2[c2.index(max(c2))])

print (c2.index(min(c2)))

print (c2[c2.index(min(c2))])



# Lab2:Density Based Clustering

## 算法思想

其核心思想就是先发现密度较高的点，然后把相近的高密度点逐步都连成一片，进而生成各种簇。

算法实现上就是，对每个数据点为圆心，以eps为半径画个圈（称为邻域eps-neigbourhood），然后数有多少个点在这个圈内，这个数就是该点密度值。

输入：D-数据对象集合 ；Eps-邻域或称为半径 ； MinPts-密度阈值

输出：基于密度的簇的集合

方法：

Step1 读取D中任意一个未分类的对象p；

Step2 检索出与p的距离不大于Eps的所有对象Neps(p)；

Step3 如果 |Neps(p)|< MinPts （即p为非核心对象），则将p标记为噪声，并执行Step1；

Step4 否则（即p为核心对象），给 Neps(p)中的所有对象打上一个新的类标签 newid，然后将这些对象压入堆栈的Seeds中；

Step5 让CurrentObject = Seeds.top；然后检索属于Neps(CurrentObject) 的 所有对象；如果| Neps(CurrentObject) |>MinPts，则剔除已经打上标记的 对象，将余下的未分类对象打上类标签newid，然后压入堆栈；

Step6 Seeds.pop，判断Seeds是否为空，是，则执行Step1 ，否则执行Step5。

## 实现代码

filename = 'iris.txt'

datar=[]

with open(filename,'r') as file\_to\_read:

while (True):

datat=[]

lines = file\_to\_read.readline()

if not lines:

break

att1, att2, att3, att4,\

temp = [i for i in lines.split(",")]

datat.append(float(att1))

datat.append(float(att2))

datat.append(float(att3))

datat.append(float(att4))

datar.append(datat)

D=np.array(datar)

def get\_density(self, x, X, y=None, sample\_weight=None):

superweight = 0.

n\_samples = X.shape[0]

n\_features = X.shape[1]

if sample\_weight is None:

sample\_weight = np.ones((n\_samples, 1))

else:

sample\_weight = sample\_weight

for y in range(n\_samples):

kernel = kernelize(x, X[y], h=self.h, degree=n\_features)

kernel = kernel \* sample\_weight[y] / (self.h \*\* n\_features)

superweight = superweight + kernel

density = superweight / np.sum(sample\_weight)

return density

def kernelize(x, y, h, degree):

kernel = np.exp(-(np.linalg.norm(x - y) / h) \*\* 2. / 2.) / ((2. \* np.pi) \*\* (degree / 2))

return kernel

def DENCLUE(D, eps=0.3, min\_samples=10):

db = DBSCAN(eps=eps, min\_samples=min\_samples).fit(D)

coreSampleMask = np.zeros\_like(db.labels\_, dtype = bool)

coreSampleMask[db.core\_sample\_indices\_] = True

clusterLabels = iris.target

uniqueClusterLabels = set(clusterLabels)

colors = ['red', 'green', 'blue', 'grey', 'black']

markers = ['x', 'o', '+', '\*', '8', 'H', '3']

for i, cluster in enumerate(uniqueClusterLabels):

clusterIndex = (clusterLabels == cluster)

coreSamples = D[clusterIndex & coreSampleMask]

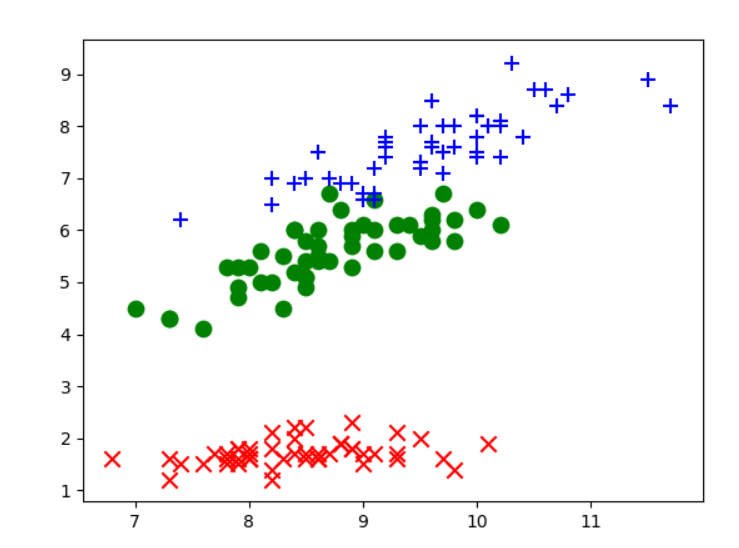
plt.scatter(coreSamples[:, 0] + coreSamples[:, 1], coreSamples[:, 2] + coreSamples[:, 3],c=colors[i], marker=markers[i], s=80)

noiseSamples = D[clusterIndex & ~coreSampleMask]

plt.scatter(noiseSamples[:, 0] + noiseSamples[:, 1],noiseSamples[:, 2] + noiseSamples[:, 3], c=colors[i], marker=markers[i], s=26)

plt.show()  
DENCLUE(D, 10, 10)

结果：



# Lab3:Decision Tree

## 算法思想

决策树输入是一组带有类别标记的点，构造的结果是一棵叉树。树的内部节点(非叶子节点)一般表示为一个逻辑判断，如形式为a=aj的逻辑判断，其中a是属性，aj是该属性的所有取值：树的边是逻辑判断的分支结果。多叉树(ID3)的内部结点是属性，边是该属性的所有取值，有几个属性值就有几条边。树的叶子节点都是类别标记。

算法实现：

1）树以代表训练样本的单个结点开始。

2）如果样本都在同一个类．则该结点成为树叶，并用该类标记。

3）否则，算法选择最有分类能力的属性作为决策树的当前结点．

4）根据当前决策结点属性取值的不同，将训练样本数据集tlI分为若干子集，每个取值形成一个分枝，有几个取值形成几个分枝。匀针对上一步得到的一个子集，重复进行先前步骤，递4'I形成每个划分样本上的决策树。一旦一个属性出现在一个结点上，就不必在该结点的任何后代考虑它。

5）递归划分步骤仅当下列条件之一成立时停止：

①给定结点的所有样本属于同一类。

②没有剩余属性可以用来进一步划分样本．在这种情况下．使用多数表决，将给定的结点转换成树叶，并以样本中元组个数最多的类别作为类别标记，同时也可以存放该结点样木的类别分布，

③如果某一分支没有样本，则以样本的多数类创建一个树叶。

## 实现代码

import numpy as np

from sklearn import tree

from sklearn.datasets import load\_iris

import matplotlib as plt

iris=load\_iris()

filename = 'iris.txt'

datar=[]

with open(filename,'r') as file\_to\_read:

while (True):

datat=[]

lines = file\_to\_read.readline()

if not lines:

break

att1, att2, att3, att4,\

temp = [i for i in lines.split(",")]

datat.append(float(att1))

datat.append(float(att2))

datat.append(float(att3))

datat.append(float(att4))

datar.append(datat)

D=np.array(datar)

clf = tree.DecisionTreeClassifier()

clf = clf.fit(D, iris.target)

with open("tree.dot",'w')as file:

file=tree.export\_graphviz(clf,out\_file=file)

import pydotplus

dota\_data=tree.export\_graphviz(clf,out\_file=None)

graph = pydotplus.graph\_from\_dot\_data(dota\_data)

print(graph)

graph.write\_pdf("result.pdf")

## 结果

生成节点信息与决策树如下：

