数据挖掘课程报告



姓名： 廖松

班级： 191141

学号： 20141000235

学院： 计算机学院

专业：计算机科学与技术

**目录**

**Lab1 1**

Project1: Numeric Data Analysis 3

python核心代码和运行结果 3

**Lab2 9**

Project3:Density Based Clustering 9

DENCLUE算法思想及基本操作 9

python核心代码和运行结果 10

**Lab3 12**

Project4:Decision Tree 12

python核心代码和运行结果 12

# Lab 1

## Project1:Numeric Data Analysis：

操作步骤及问题：

1. 先将数据集magic04.txt文件读入python程序中

2 问题1-计算多元平均向量

3 问题2-计算属性1和2之间的相关性，计算中心属性向量之间夹角的余弦值。绘制这两个属性之间的散点图。

4 问题3-假设属性1为正态分布，则绘制其概率密度函数。

5 问题 4-哪个属性的方差最大，哪个属性的方差最小?打印这些值。

6 问题5：哪一对属性有最大的协方差，哪一对属性的协方差最小?打印这些值。

**Python核心代码及运行结果：**

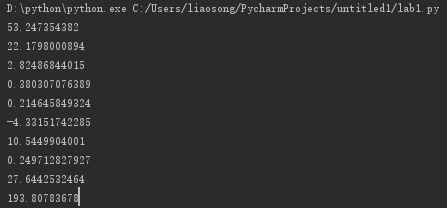
导入数据集的python代码如下：

#coding:utf-8  
  
import os  
import numpy as np  
import matplotlib  
import matplotlib.pyplot as chart  
  
matrix0 = []  
matrix1 = []  
matrix2 = []  
matrix3 = []  
matrix4 = []  
matrix5 = []  
matrix6 = []  
matrix7 = []  
matrix8 = []  
matrix9 = []  
numb = 0 # 记录总个数  
filename = 'magic04.txt'  
  
def fun\_a(numb):  
 with open(filename, 'r') as file\_to\_read:  
 while True:  
 lines = file\_to\_read.readline()  
 numb += 1  
  
 if not lines:  
 break  
  
 mat0\_tmp, mat1\_tmp, mat2\_tmp, \  
 mat3\_tmp, mat4\_tmp, mat5\_tmp, \  
 mat6\_tmp, mat7\_tmp, mat8\_tmp, \  
 mat9\_tmp, attr = [i for i in lines.split(",")]  
 matrix0.append(float(mat0\_tmp))  
 matrix1.append(float(mat1\_tmp))  
 matrix2.append(float(mat2\_tmp))  
 matrix3.append(float(mat3\_tmp))  
 matrix4.append(float(mat4\_tmp))  
 matrix5.append(float(mat5\_tmp))  
 matrix6.append(float(mat6\_tmp))  
 matrix7.append(float(mat7\_tmp))  
 matrix8.append(float(mat8\_tmp))  
 matrix9.append(float(mat9\_tmp))  
  
 print sum(matrix0) / numb  
 print sum(matrix1) / numb  
 print sum(matrix2) / numb  
 print sum(matrix3) / numb  
 print sum(matrix4) / numb  
 print sum(matrix5) / numb  
 print sum(matrix6) / numb  
 print sum(matrix7) / numb  
 print sum(matrix8) / numb  
 print sum(matrix9) / numb

问题一代码如下：

print sum(matrix0) / numb  
print sum(matrix1) / numb  
print sum(matrix2) / numb  
print sum(matrix3) / numb  
print sum(matrix4) / numb  
print sum(matrix5) / numb  
print sum(matrix6) / numb  
print sum(matrix7) / numb  
print sum(matrix8) / numb  
print sum(matrix9) / numb

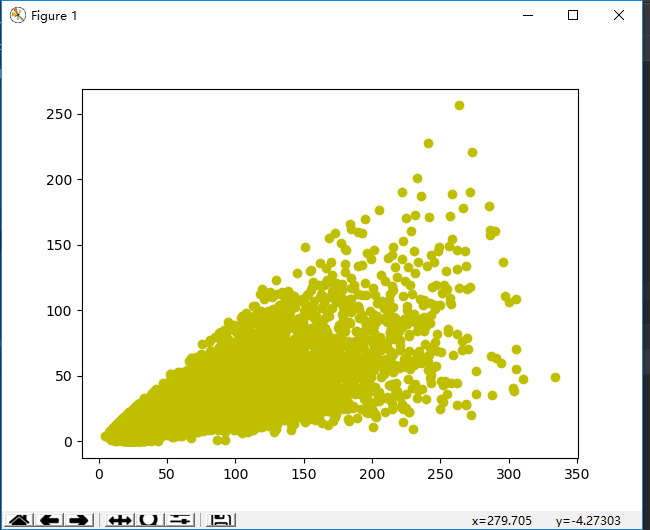
运行结果如图所示：



问题二代码如下：

y = [matrix0, matrix1, matrix2, matrix3, matrix4, matrix5, matrix6, matrix7, matrix8, matrix9]  
print np.cov(y)  
a = np.array(matrix0)  
b = np.array(matrix1)  
chart.plot(a, b, 'yo')  
chart.show()

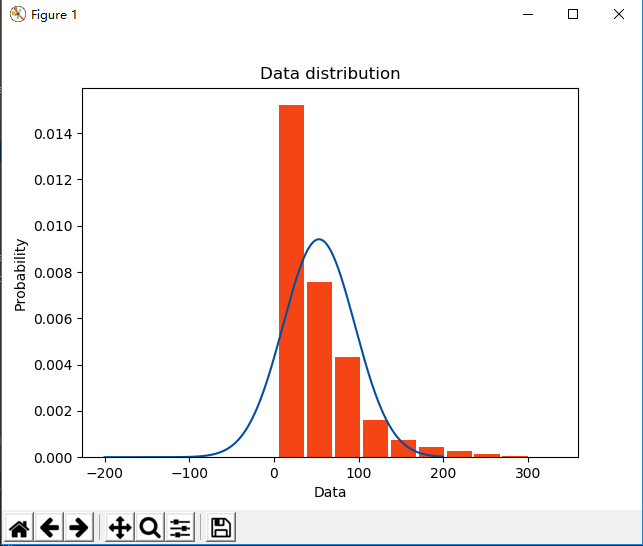
运行结果如图所示（属性1和属性2的散点图）：



问题三代码如下：

data = a  
 mean = data.mean()  
 std = data.std()  
 x = np.arange(-200, 200, 0.1)  
 y = normal\_fun(x, mean, std)  
 chart.plot(x, y, color='#054E9F')  
 chart.hist(data, color='#F54515', bins=10, rwidth=0.9, normed=True)  
 chart.title('Data distribution')  
 chart.xlabel('Data')  
 chart.ylabel('Probability')  
 chart.show()  
  
def normal\_fun(x,mu,sigma):  
 pdf = np.exp(-((x - mu)\*\*2)/(2\*sigma\*\*2)) / (sigma \* np.sqrt(2\*np.pi))  
 return pdf  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 fun\_a(numb)

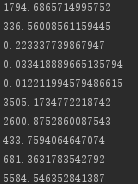
根据题设属性1的正太分布图像如图所示：



问题四代码如下：

def variancefun(a):  
 array = np.array(a)  
 var = array.var()  
 return var  
print variancefun(matrix0)  
print variancefun(matrix1)  
print variancefun(matrix2)  
print variancefun(matrix3)  
print variancefun(matrix4)  
print variancefun(matrix5)  
print variancefun(matrix6)  
print variancefun(matrix7)  
print variancefun(matrix8)  
print variancefun(matrix9)

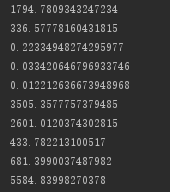
结果如图：



问题五代码如下：

def covfun(a):  
 array = np.array(a)  
 cov = np.cov(array)  
 return cov  
print covfun(matrix0)  
print covfun(matrix1)  
print covfun(matrix2)  
print covfun(matrix3)  
print covfun(matrix4)  
print covfun(matrix5)  
print covfun(matrix6)  
print covfun(matrix7)  
print covfun(matrix8)  
print covfun(matrix9)

结果如图：



**Lab 2**

**Project3:Density Based Clustering:**

要求：

编写代码实现基于密度的聚类算法。

脚本应作为数据集D的输入，最小密度Ze，公差

收敛性和带宽H。

不要对数据进行任何假设（即列名等），除非

最后一列给出了“true”簇ID。

问题：

在IRIS数据集上运行脚本，用＝0.0001。您的脚本应该输出以下内容：

1。簇的数目和每个簇的大小。

2。密度吸引子，然后是该簇中的点集。

三。聚类的纯度，基于真实ID。

为了加快估算一个点的密度，你可能要先计算。

识别K最近邻居，只使用那些邻居。

**DENCLUE算法思想及基本操作：**

基本思想：使用核密度函数用个体数据对象影响之和对点集总密度建模，尽管DENCLUE本质上不是基于网格的技术，但是它使用基于网格的方法提高性能。DENCLUE也是一种基于密度的方法。结果总密度函数具有局部尖峰（称作局部吸引点），并且这些局部尖峰用来以自然的方式定义簇。集体的说，对于每个数据点，一个爬山过程找出与该点相关联的最近的尖峰，并且与一个特定的尖峰相关联的所有数据称为一个簇。

DENCLUE算法操作步骤：

（1）对数据点占据的空间推导密度函数；

（2）识别局部最大点（这是局部吸引点）；

（3）通过沿密度增长最大的方向移动，将每个点关联到一个密度吸引点；

（4）定义与特定的密度吸引点相关联的点构成的簇；

（5）丢弃密度吸引点的密度小于用户指定阈值的簇；

（6）合并通过密度大于或等于的点路径连接的簇。

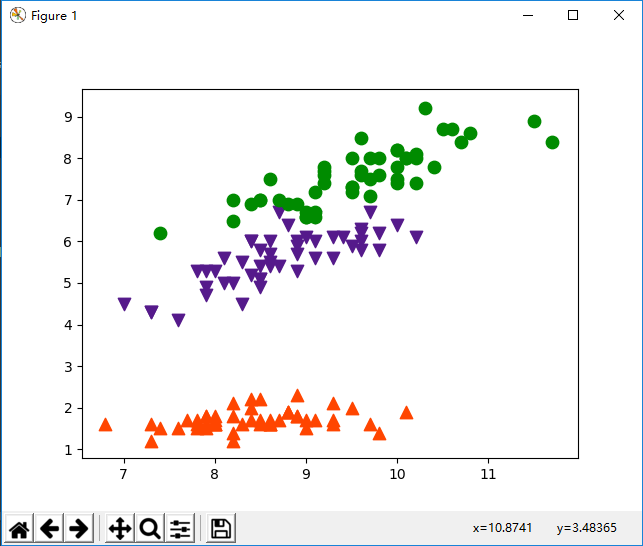
**Python核心代码和运行结果：**

1 Write a script to implement the DENCLUE density-based clustering algorithm.

编写代码实现基于密度的聚类算法：

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import DBSCAN  
from sklearn import datasets  
iris = datasets.load\_iris()  
data = iris.data  
def DENCLUE(data, eps=0.3, min\_samples=10):  
 db = DBSCAN(eps=eps, min\_samples=min\_samples).fit(data)  
 coreSampleMask = np.zeros\_like(db.labels\_, dtype = bool)  
 coreSampleMask[db.core\_sample\_indices\_] = True  
 clusterLabels = iris.target  
 uniqueClusterLabels = set(clusterLabels)  
 colors = ['#FF4500', '#551A8B', '#008B00']  
 markers = ['v', '^', 'o']  
 for i, cluster in enumerate(uniqueClusterLabels):  
 clusterIndex = (clusterLabels == cluster)  
 coreSamples = data[clusterIndex & coreSampleMask]  
 plt.scatter(coreSamples[:, 0] + coreSamples[:, 1], coreSamples[:, 2] + coreSamples[:, 3],c=colors[i], marker=markers[i], s=80)  
 noiseSamples = data[clusterIndex & ~coreSampleMask]  
 plt.scatter(noiseSamples[:, 0] + noiseSamples[:, 1],noiseSamples[:, 2] + noiseSamples[:, 3], c=colors[i], marker=markers[i], s=26)  
 plt.show()  
DENCLUE(data, 10, 10)

结果如图所示：



获取密度的函数如下：

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

# Lab3

## Project4:Decision Tree：

### 算法思想：

决策树(Decision Tree)是在已知各种情况发生概率的[基础](https://baike.baidu.com/item/%E5%9F%BA%E7%A1%80/32794" \t "_blank)上，通过构成决策树来求取净现值的[期望](https://baike.baidu.com/item/%E6%9C%9F%E6%9C%9B/35704" \t "_blank)值大于等于零的概率，评价项目风险，判断其可行性的决策分析方法，是直观运用概率分析的一种图解法。

**Python核心代码和运行结果：**

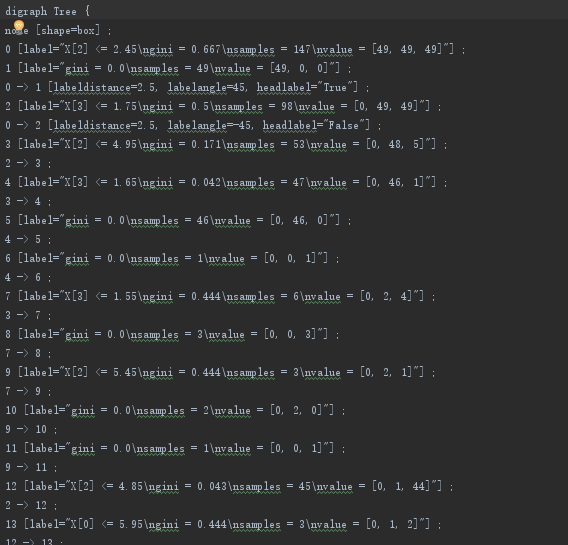
•Write a script to compute the decision tree for an input dataset.

#以下代码是读取鸢尾花数据集并将Decision Point 写入tree.dot 文件的python代码

import numpy as np  
from sklearn.datasets import load\_iris  
from sklearn import tree  
import matplotlib as plt  
iris = load\_iris()  
test\_idx = [0, 50, 100]  
train\_target = np.delete(iris.target, test\_idx)  
train\_data = np.delete(iris.data, test\_idx, axis=0)  
  
test\_target = iris.target[test\_idx]  
test\_data = iris.data[test\_idx]  
  
clf = tree.DecisionTreeClassifier()  
clf = clf.fit(train\_data, train\_target)  
tree.export\_graphviz(clf,out\_file='tree.dot')

运行完以上程序可在项目中找到tree.dot的文件，该文件存储了鸢尾花数据集的Decision Point；

Decision Point存入文件tree.dot中，文件部分截图如下

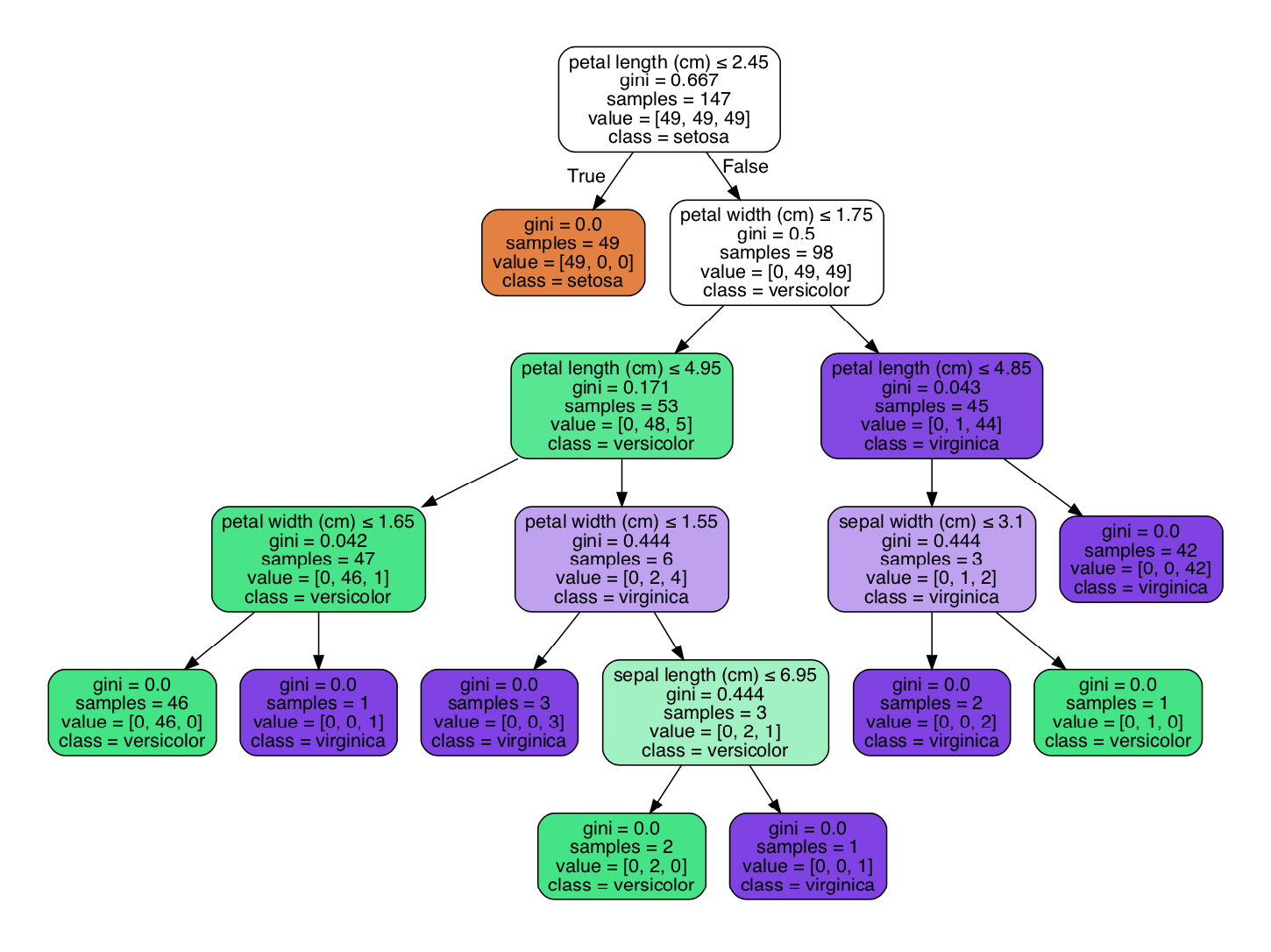


•Output your decision tree. Note that each internal node, print the decision followed by the Information Gain, and for each leaf, print the majority label, purity of the leaf, and the size.

将Decision Point绘制成决策树的代码如下：

from sklearn.externals.six import StringIO  
import pydotplus  
dot\_data = StringIO()  
tree.export\_graphviz(clf,out\_file=dot\_data,feature\_names=iris.feature\_names,class\_names=iris.target\_names,  
filled=True, rounded=True,  
special\_characters=True)  
graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())  
  
print(graph)  
graph.write\_pdf("iris.pdf")

决策树截图如下：



GitHub链接：<https://github.com/liaosong1/curriculumdesign>