**模式识别 K近邻法**

**一、最近邻、k近邻算法介绍**

**1.1 介绍**

以每个训练样本为一个子类，不同类的两个样本之间用最小距离作为分类准则。显然这时就没有必要事先用所有两两样本间的分类面构造出分段线性分类面，而是可以在拿到一个待分类的样本后，通过判断它到两类样本的距直来进行决策。这就是最近邻法。

最近邻法就是源于这样一种直观的想法：对于一个新样本，把它逐一与已知样本比较，找出距离新样本最近的已知样本，该样本的类别作为新样本的类别。

在很多情况下，把决策建立在一个最近的样本上有一定风险，当数据分布复杂或数据中噪声严重时尤其如此。一种很自然的改进就是引人投票机制选择前若干个离新样本最近的已知样本，用它们的类别投票来决定新样本的类别，这种方法称作k-近邻法，因为人们习惯上把参加投票的近邻样本的个数记作k。显然，最近邻法可以看作是k近邻法的特例。

**1.2 近邻法的形式化表示**

**（1）最近邻**

已知样本集，其中，是样本i的特征向量，是它对应的类别，设c个类，即。定义两个样本简得距离量度，比如可以采用欧氏距离。对未知样本，求中与之距离最近的样本，设为（对应的类别为），即：

将决策为类。

如果写成判别函数的形式，类的判别函数可以写作：

决策规则为各类的判别函数比较大小，即：

**错误率：**最坏不会超过两倍的贝叶斯错误率，而最好则有可能接近或达到贝叶斯错误率。

注：根据实际问题，可以采用不同的距离度量。

**（2）k近邻**

设有N个已知样本分属于c个类，，考察新样本在这些样本中的前个近邻，设有个属于类，则类的判别函数就是：

决策规则是：

**错误率：**当k趋于无穷大时，k近邻法就达到了贝叶斯错误率。

注：在两分类问题中，通常需要选择k为奇数，以避免出现两类的票相等的情况

**二、实验数据集介绍**

**2.1 Iris数据集介绍**

|  |  |  |  |
| --- | --- | --- | --- |
| Iris 数据集 | | | |
| 类别 | 3 | | |
| 维度 | 4 | | |
| 数据长度 | 150 | 50 | Iris-setosa |
| 50 | Iris-versicolor |
| 50 | Iris-setosa |

**2.2 Sonar数据集介绍**

|  |  |  |  |
| --- | --- | --- | --- |
| Sonar 数据集 | | | |
| 类别 | 2 | | |
| 维度 | 60 | | |
| 数据长度 | 208 | 97 | R |
| 111 | M |

**三、实验设置**

对两类数据集，均分别采用最近邻、k近邻两种方法进行分类，计算准确率，最后将k值、测试比例进行变换，观察分类结果。

两类数据集处理方法具有相似性，因此实验设置基本一致，具体如下：

**1、读取数据信息。**首先从数据集文件（iris.data、sonar.all-data）中读取数据，将读取到的数据按照一定比例（例如：1/5的数据用于测试，4/5的数据用于训练）**随机**存放到训练集和测试集中，再将训练集中数据按标签分类（iris分三类、sonar分两类）。

**2、求解样本间距离。**依次遍历测试集中的元素，将测试数据分别与训练集中的每一组数据求解欧氏距离（还可更换其他距离），将所有距离数据保存。

**3、寻找最小距离/分类别投票，求分类结果。**

（1）最近邻法：直接遍历即可求出最小距离，该最小距离对应的样本标签即为分类结果。

（2）k近邻法：将训练集数据的标签和对应距离依据距离进行排序（递增），取前k项数据，统计出现次数最多的标签数，该标签即为分类结果。

**4、 统计所有测试数据分类的正确、错误数量，计算准确率。**

**5、重复1-4步20次，计算平均准确率，绘图显示。**

**6、更改k值，观察准确率变化。**

**四、实验结果展示与分析**

**4.1 Iris数据集分类结果分析**

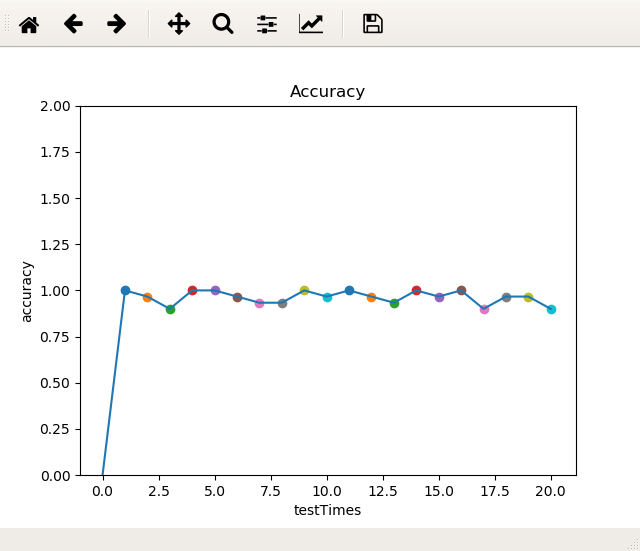
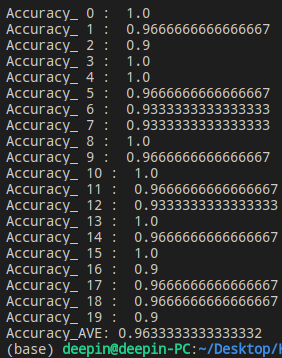
1、首先使用**最近邻**法将iris测试集进行分类，默认使用1/5的数据用于测试，4/5的数据用于训练，重复20次后，分类精确度如下：

图4.1：20次最近邻法iris数据集分类结果

上述结果可以看出：20次最近邻法iris数据集分类平均准确率96.3%，由于测试样本较少，还出现了几次准确率为1的情况。

2、使用**k近邻**法将iris数据集分类，设置k=5，其余设置与上述一致，分类结果如下：

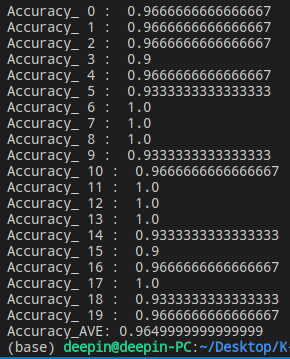
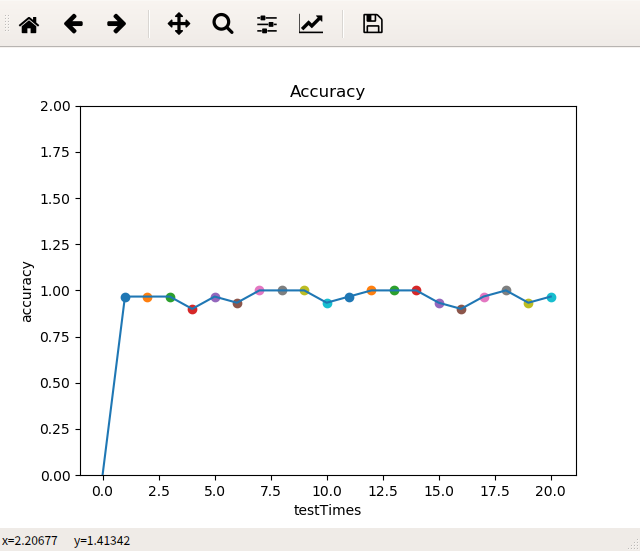


图4.2：20次k近邻法(k=5)iris数据集分类结果

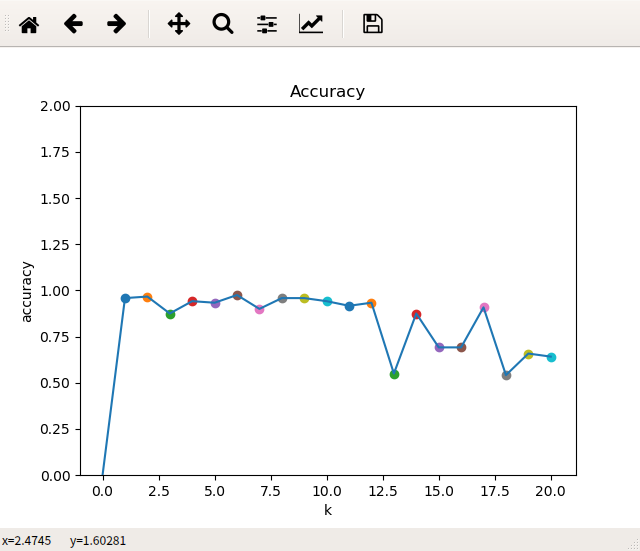
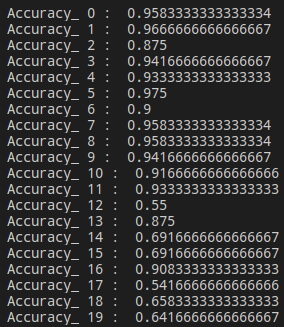
上述结果可以看出：使用k近邻法（k=5）的分类结果与最近邻法相比，精准度略有提高，但相差不多。平均准确率为96.5%。

图4.3：20次k取不同值时iris数据集分类结果

3、随后**改变k**的值（1-20，对应图中0-19）进行测试，分类结果如下：

上述结果可以看出：k取值在12之前准确率均在较高水平，由于测试集只有30个样本（1/5的数据用于测试），而且包含了3类，所以当k超过一定值（实验结果12）时，距离测试样本前k近的已知样本中必然包含错误类的样本，导致分类错误，可见，k不能设置太大。

尤其，当k=1时，k近邻法就等价于最近邻法，实验结果也可以看到，k=1时的准确率与最近邻相似（有区别原因在于每次分类都会重新分配训练、测试数据，导致结果不同）。

k很小时，准确率也很高，同时也说明iris数据集的噪声很小，易分类。

**4.2 sonar数据集分类结果分析**

由于两类数据集处理具有相似性，这里只简要描述。

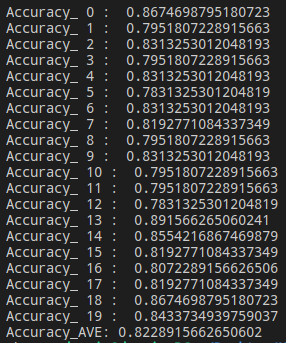
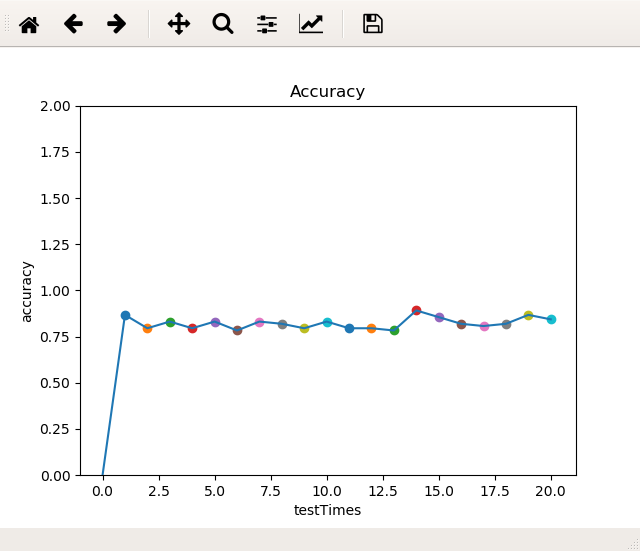
1、首先使用**最近邻**法将sonar测试集进行分类（1/5测试）：

图4.4：20次最近邻法sonar数据集分类结果

上述结果可以看出：20次最近邻法sonar数据集分类平均准确率为82.3%。

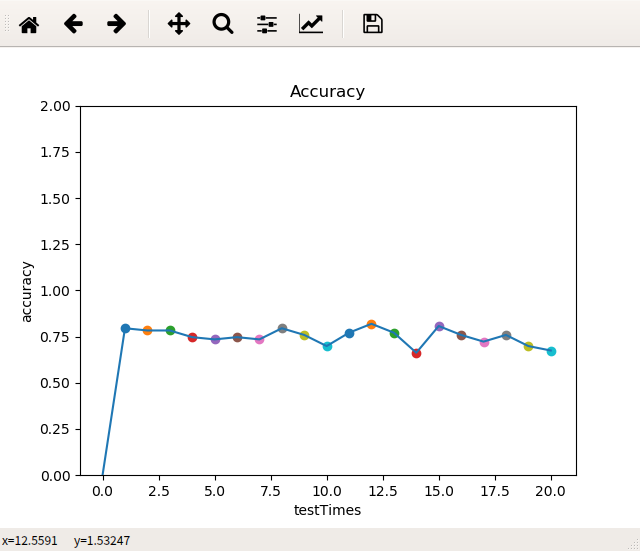
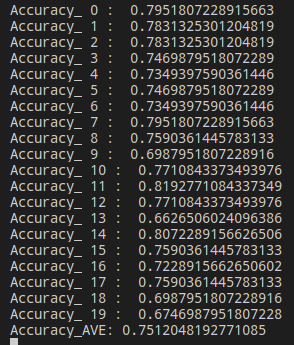
2、使用**k近邻**法将sonar数据集分类，设置k=5，其余设置与上述一致，分类结果如下：

图4.5：20次k近邻法(k=5)sonar数据集分类结果

上述结果可以看出：20次k近邻法（k=5）sonar数据集分类平均准确率为75.1%，相比最近邻有所下降。

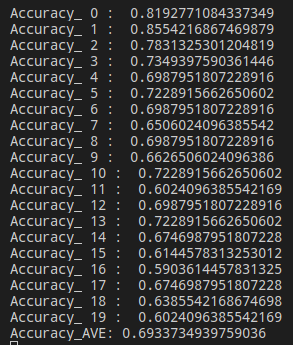
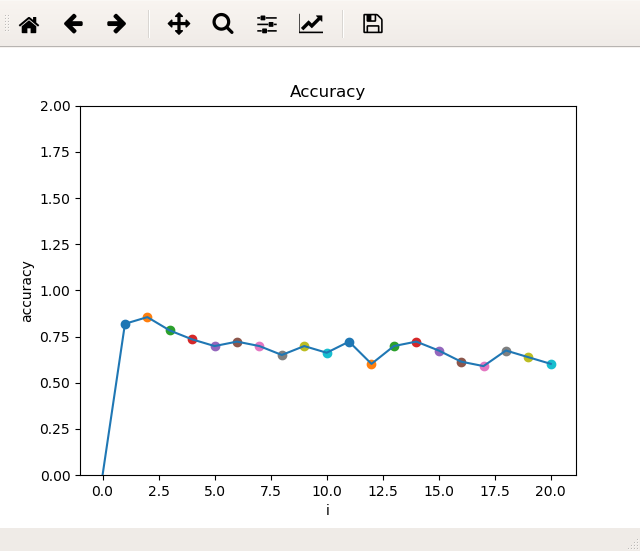
3、随后**改变k**的值（1-20，对应图中0-19）进行测试，分类结果如下：

图4.6：20次k取不同值时sonar数据集分类结果

上述结果可以看出：k=2时准确率最高，随k增加准确率下降，由于样本较多，没有iris数据集那样变化剧烈。

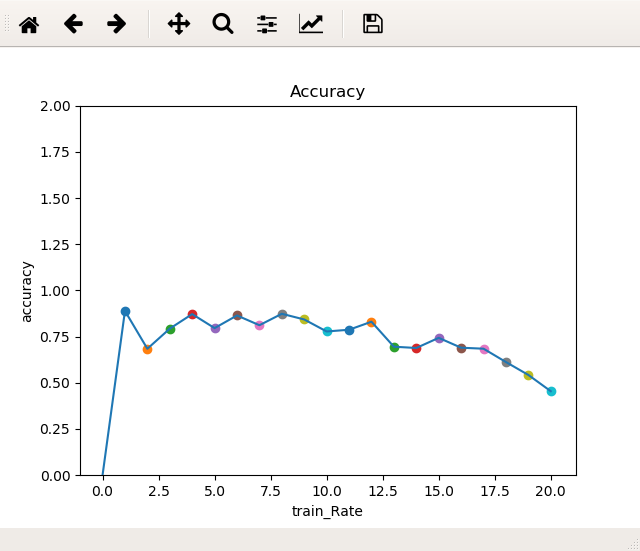
4、随后**改变测试集和训练集的比例**，可以看到随测试样本增加、已知样本数量减少，准确率逐渐降低：

图4.7：减少已知样本对分类准确率的影响

**五、Fisher与k近邻（最近邻）对比**

首先，给出每种算法的分类准确率如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | fisher | 最近邻 | k近邻(k=2) | k近邻(k=5) |
| iris数据集 | 96.70% | 96.30% | 96.60% | 96.50% |
| sonar数据集 | 72.30% | 82.30% | 85.50% | 75.10% |

可以看出，iris数据集使用fisher分类效果最好，sonar使用k近邻或最近邻分类效果好，可见，不同数据集应根据本身特性选用不同的分类方法。

两大类算法具体对比如下：

1、**Fisher算法**利用训练样本求出的权向量对测试数据进行降维进行分类。

该算法需求出每两类间的判别函数，对两分类问题较为方便，但对多分类问题较为复杂；

当训练样本中有少量噪声时，对分类结果影响不大，但样本有重叠，难以通过投影方式分开时，该方法错误率较高；

随训练样本的数目增加，训练时间会有所增加，但并不影响测试数据的分类时间。

2、**近邻算法**以新样本与的已知样本的距离为依据，寻找距离最近的已知样本或前k个距离最近样本类别投票进行分类。

该算法对二分类、多分类问题处理方法相同，方便适用于多分类问题。

当训练样本中有少量噪声时，对最近邻的分类结果影响很大，通过k近邻法有所改善，但需要选取合适的k值，k近邻法较fisher还改善了样本重叠造成的影响。

近邻法相比于fisher，没有了训练过程，但如果已知样本数量巨大，由于需要将测试样本与所有已知样本对比一遍，测试时间将非常漫长，分类效率很低，需要通过近邻法的快速算法进行改善。

**六、Python代码**

**5.1 iris数据集**

import os

import sys

import numpy as np

from numpy import \*

import random

import matplotlib.pyplot as plt

def readData(DATA\_PATH,DATA\_SIZE,DATA\_DIMENSION,test\_rate):

    test\_Labels = random.sample(range(0,DATA\_SIZE-1),int(DATA\_SIZE\*test\_rate))

    test\_Data = [[0 for i in range(DATA\_DIMENSION+1)] for j in range(len(test\_Labels))]

    train\_Data = [[0 for i in range(DATA\_DIMENSION+1)] for j in range(DATA\_SIZE-len(test\_Labels))]

    #随机提取训练、测试数据

    f = open(DATA\_PATH)

    line = f.readline()

    i = 0

    train\_Label = 0

    test\_Label = 0

    while line:

        temp = line.split(",",DATA\_DIMENSION)

        if i in test\_Labels:

            test\_Data[test\_Label][0:DATA\_DIMENSION] = list(map(float, temp[0:DATA\_DIMENSION]))

            test\_Data[test\_Label][DATA\_DIMENSION] = temp[DATA\_DIMENSION]

            test\_Label = test\_Label+1

        else:

            train\_Data[train\_Label][0:DATA\_DIMENSION] = list(map(float, temp[0:DATA\_DIMENSION]))

            train\_Data[train\_Label][DATA\_DIMENSION] = temp[DATA\_DIMENSION]

            train\_Label = train\_Label+1

        i = i+1

        line = f.readline()

    #将要训练的数据分类保存

    \_1\_Num = \_2\_Num = \_3\_Num = 0

    for i in range(len(train\_Data)):

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-setosa\n':

            \_1\_Num = \_1\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-versicolor\n':

            \_2\_Num = \_2\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-virginica\n':

            \_3\_Num = \_3\_Num+1

    \_1\_Data = mat(zeros((\_1\_Num,DATA\_DIMENSION)))

    \_2\_Data = mat(zeros((\_2\_Num,DATA\_DIMENSION)))

    \_3\_Data = mat(zeros((\_3\_Num,DATA\_DIMENSION)))

    \_1\_Num = \_2\_Num = \_3\_Num = 0

    for i in range(len(train\_Data)):

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-setosa\n':

            \_1\_Data[\_1\_Num] = train\_Data[i][0:DATA\_DIMENSION]

            \_1\_Num = \_1\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-versicolor\n':

            \_2\_Data[\_2\_Num] = train\_Data[i][0:DATA\_DIMENSION]

            \_2\_Num = \_2\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-virginica\n':

            \_3\_Data[\_3\_Num] = train\_Data[i][0:DATA\_DIMENSION]

            \_3\_Num = \_3\_Num+1

    f.close()

    return train\_Data,test\_Data,\_1\_Data.T,\_2\_Data.T,\_3\_Data.T

def calcu\_Eucli\_Dis(x,y):

    dis = np.sqrt(np.sum(np.square(x-y)))

    return dis

def Nearest\_Neighbors(dataPath,dataSize,dataDimension,dataTypeNum,testRate = 2/5):

    #数据准备

    train\_Data,test\_Data,group1,group2,group3 = readData(dataPath,dataSize,dataDimension,testRate)

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    print("                    train\_Size:",len(train\_Data))

    print("                    test\_Size:",len(test\_Data))

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    wrongNum = correctNum = 0

    #遍历每一个测试元素

    for testElement in range(len(test\_Data)):

        minDis = 1024

        minLabel = -1

        print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*testElement\_",testElement,"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

        #每一个测试元素与训练集中的每一个数据进行距离计算

        for trainElement in range(len(train\_Data)):

            testArray = np.array(test\_Data[testElement][0:dataDimension])

            trainArray = np.array(train\_Data[trainElement][0:dataDimension])

            #计算距离

            distance = calcu\_Eucli\_Dis(testArray,trainArray)

            #寻找最近邻元素

            if distance < minDis:

                minDis = distance

                minLabel = trainElement

        print("minDis:",minDis)

        print("Real:",test\_Data[testElement][dataDimension],"Forecast:",train\_Data[minLabel][dataDimension])

        if(test\_Data[testElement][dataDimension] == train\_Data[minLabel][dataDimension]):

            correctNum += 1

        else:

            wrongNum += 1

    print("------------------------------------")

    print("|WrongNums:",wrongNum,"correctNums:",correctNum)

    print("------------------------------------\n\n\n")

    Accuracy = correctNum/(correctNum+wrongNum)

    return  Accuracy

def dis\_Sort(allDisInfo):

    for i in range(0,len(allDisInfo)):

        for j in range(0,len(allDisInfo)-i-1):

            if allDisInfo[j][1]>allDisInfo[j+1][1]:

                temp = allDisInfo[j]

                allDisInfo[j] = allDisInfo[j+1]

                allDisInfo[j+1] = temp

    return allDisInfo

def k\_Judge(allDisInfo,k):

    type\_1 = type\_2 = type\_3 = 0

    for i in range(k):

        if allDisInfo[i][0] == "Iris-setosa\n":

            type\_1 += 1

        if allDisInfo[i][0] == "Iris-versicolor\n":

            type\_2 += 1

        if allDisInfo[i][0] == "Iris-virginica\n":

            type\_3 += 1

    if type\_1>=type\_2 and type\_1>=type\_3:

        return "Iris-setosa\n"

    if type\_2>=type\_1 and type\_2>=type\_3:

        return "Iris-versicolor\n"

    if type\_3>=type\_1 and type\_3>=type\_2:

        return "Iris-virginica\n"

def K\_Nearest\_Neighbors(dataPath,dataSize,dataDimension,dataTypeNum,testRate,k=5):

    #数据准备

    train\_Data,test\_Data,group1,group2,group3 = readData(dataPath,dataSize,dataDimension,testRate)

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    print("                    train\_Size:",len(train\_Data))

    print("                    test\_Size:",len(test\_Data))

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    wrongNum = correctNum = 0

    #遍历每一个测试元素

    for testElement in range(len(test\_Data)):

        allDisInfo = [[0 for i in range(2)] for j in range(len(train\_Data))]  #[forcast\_label][distance]

        print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*testElement\_",testElement,"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

        #每一个测试元素与训练集中的每一个数据进行距离计算

        for trainElement in range(len(train\_Data)):

            testArray = np.array(test\_Data[testElement][0:dataDimension])

            trainArray = np.array(train\_Data[trainElement][0:dataDimension])

            #计算距离

            dis = calcu\_Eucli\_Dis(testArray,trainArray)

            #保存所有label的距离信息

            allDisInfo[trainElement][0] = train\_Data[trainElement][dataDimension]

            allDisInfo[trainElement][1] = dis

        #对距离信息进行排序

        dis\_Sort(allDisInfo)

        for i in range(len(allDisInfo)):

            print(allDisInfo[i])

        #选择K值进行判别

        forcast\_label = k\_Judge(allDisInfo,k)

        print("------------------------------------> Forcast Label:",forcast\_label,"-----------------------------------> Real Label:",test\_Data[testElement][dataDimension])

        if(test\_Data[testElement][dataDimension] == forcast\_label):

            correctNum += 1

        else:

            wrongNum += 1

    print("------------------------------------")

    print("|WrongNums:",wrongNum,"correctNums:",correctNum)

    print("------------------------------------\n\n\n")

    Accuracy = correctNum/(correctNum+wrongNum)

    return  Accuracy

if \_\_name\_\_ == "\_\_main\_\_":

    Accuracy\_AVE = 0

    Accuracy = []

    Accuracy.append(0)

    testTimes = 20

    for i in range(testTimes):

        temp = K\_Nearest\_Neighbors("iris.data",150,4,3,1/5,5)

        # temp = Nearest\_Neighbors("iris.data",150,4,3,1/5)

        Accuracy.append(temp)

        plt.scatter(i+1,temp)

        Accuracy\_AVE += temp

    Accuracy\_AVE /= testTimes

    for i in range(testTimes):

        print("Accuracy\_",i,": ",Accuracy[i+1])

    print("Accuracy\_AVE:",Accuracy\_AVE)

    plt.title("Accuracy")

    plt.xlabel("testTimes")

    plt.ylabel("accuracy")

    plt.ylim(0,2)

    plt.plot(Accuracy)

    plt.show()

**5.2 sonar数据集（代码通用，与iris相差无几）**

import os

import sys

import numpy as np

from numpy import \*

import random

import matplotlib.pyplot as plt

def readData(DATA\_PATH,DATA\_SIZE,DATA\_DIMENSION,test\_rate):

    test\_Labels = random.sample(range(0,DATA\_SIZE-1),int(DATA\_SIZE\*test\_rate))

    test\_Data = [[0 for i in range(DATA\_DIMENSION+1)] for j in range(len(test\_Labels))]

    train\_Data = [[0 for i in range(DATA\_DIMENSION+1)] for j in range(DATA\_SIZE-len(test\_Labels))]

    #随机提取训练、测试数据

    f = open(DATA\_PATH)

    line = f.readline()

    i = 0

    train\_Label = 0

    test\_Label = 0

    while line:

        temp = line.split(",",DATA\_DIMENSION)

        if i in test\_Labels:

            test\_Data[test\_Label][0:DATA\_DIMENSION] = list(map(float, temp[0:DATA\_DIMENSION]))

            test\_Data[test\_Label][DATA\_DIMENSION] = temp[DATA\_DIMENSION]

            test\_Label = test\_Label+1

        else:

            train\_Data[train\_Label][0:DATA\_DIMENSION] = list(map(float, temp[0:DATA\_DIMENSION]))

            train\_Data[train\_Label][DATA\_DIMENSION] = temp[DATA\_DIMENSION]

            train\_Label = train\_Label+1

        i = i+1

        line = f.readline()

    #将要训练的数据分类保存

    \_1\_Num = \_2\_Num = \_3\_Num = 0

    for i in range(len(train\_Data)):

        if train\_Data[i][DATA\_DIMENSION] == 'R\n':

            \_1\_Num = \_1\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'M\n':

            \_2\_Num = \_2\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-virginica\n':

            \_3\_Num = \_3\_Num+1

    \_1\_Data = mat(zeros((\_1\_Num,DATA\_DIMENSION)))

    \_2\_Data = mat(zeros((\_2\_Num,DATA\_DIMENSION)))

    \_3\_Data = mat(zeros((\_3\_Num,DATA\_DIMENSION)))

    \_1\_Num = \_2\_Num = \_3\_Num = 0

    for i in range(len(train\_Data)):

        if train\_Data[i][DATA\_DIMENSION] == 'R\n':

            \_1\_Data[\_1\_Num] = train\_Data[i][0:DATA\_DIMENSION]

            \_1\_Num = \_1\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'M\n':

            \_2\_Data[\_2\_Num] = train\_Data[i][0:DATA\_DIMENSION]

            \_2\_Num = \_2\_Num+1

        if train\_Data[i][DATA\_DIMENSION] == 'Iris-virginica\n':

            \_3\_Data[\_3\_Num] = train\_Data[i][0:DATA\_DIMENSION]

            \_3\_Num = \_3\_Num+1

    f.close()

    return train\_Data,test\_Data,\_1\_Data.T,\_2\_Data.T,\_3\_Data.T

def calcu\_Eucli\_Dis(x,y):

    dis = np.sqrt(np.sum(np.square(x-y)))

    return dis

def Nearest\_Neighbors(dataPath,dataSize,dataDimension,dataTypeNum,testRate = 2/5):

    #数据准备

    train\_Data,test\_Data,group1,group2,group3 = readData(dataPath,dataSize,dataDimension,testRate)

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    print("                    train\_Size:",len(train\_Data))

    print("                    test\_Size:",len(test\_Data))

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    wrongNum = correctNum = 0

    #遍历每一个测试元素

    for testElement in range(len(test\_Data)):

        minDis = 1024

        minLabel = -1

        print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*testElement\_",testElement,"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

        #每一个测试元素与训练集中的每一个数据进行距离计算

        for trainElement in range(len(train\_Data)):

            testArray = np.array(test\_Data[testElement][0:dataDimension])

            trainArray = np.array(train\_Data[trainElement][0:dataDimension])

            #计算距离

            distance = calcu\_Eucli\_Dis(testArray,trainArray)

            #寻找最近邻元素

            if distance < minDis:

                minDis = distance

                minLabel = trainElement

        print("minDis:",minDis)

        print("Real:",test\_Data[testElement][dataDimension],"Forecast:",train\_Data[minLabel][dataDimension])

        if(test\_Data[testElement][dataDimension] == train\_Data[minLabel][dataDimension]):

            correctNum += 1

        else:

            wrongNum += 1

    print("------------------------------------")

    print("|WrongNums:",wrongNum,"correctNums:",correctNum)

    print("------------------------------------\n\n\n")

    Accuracy = correctNum/(correctNum+wrongNum)

    return  Accuracy

def dis\_Sort(allDisInfo):

    for i in range(0,len(allDisInfo)):

        for j in range(0,len(allDisInfo)-i-1):

            if allDisInfo[j][1]>allDisInfo[j+1][1]:

                temp = allDisInfo[j]

                allDisInfo[j] = allDisInfo[j+1]

                allDisInfo[j+1] = temp

    return allDisInfo

def k\_Judge(allDisInfo,k):

    type\_1 = type\_2 = type\_3 = 0

    for i in range(k):

        if allDisInfo[i][0] == "R\n":

            type\_1 += 1

        if allDisInfo[i][0] == "M\n":

            type\_2 += 1

        if allDisInfo[i][0] == "Iris-virginica\n":

            type\_3 += 1

    if type\_1>=type\_2 and type\_1>=type\_3:

        return "R\n"

    if type\_2>=type\_1 and type\_2>=type\_3:

        return "M\n"

    if type\_3>=type\_1 and type\_3>=type\_2:

        return "Iris-virginica\n"

def K\_Nearest\_Neighbors(dataPath,dataSize,dataDimension,dataTypeNum,testRate,k=5):

    #数据准备

    train\_Data,test\_Data,group1,group2,group3 = readData(dataPath,dataSize,dataDimension,testRate)

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    print("                    train\_Size:",len(train\_Data))

    print("                    test\_Size:",len(test\_Data))

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    wrongNum = correctNum = 0

    #遍历每一个测试元素

    for testElement in range(len(test\_Data)):

        allDisInfo = [[0 for i in range(2)] for j in range(len(train\_Data))]  #[forcast\_label][distance]

        print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*testElement\_",testElement,"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

        #每一个测试元素与训练集中的每一个数据进行距离计算

        for trainElement in range(len(train\_Data)):

            testArray = np.array(test\_Data[testElement][0:dataDimension])

            trainArray = np.array(train\_Data[trainElement][0:dataDimension])

            #计算距离

            dis = calcu\_Eucli\_Dis(testArray,trainArray)

            #保存所有label的距离信息

            allDisInfo[trainElement][0] = train\_Data[trainElement][dataDimension]

            allDisInfo[trainElement][1] = dis

        #对距离信息进行排序

        dis\_Sort(allDisInfo)

        for i in range(len(allDisInfo)):

            print(allDisInfo[i])

        #选择K值进行判别

        forcast\_label = k\_Judge(allDisInfo,k)

        print("------------------------------------> Forcast Label:",forcast\_label,"-----------------------------------> Real Label:",test\_Data[testElement][dataDimension])

        if(test\_Data[testElement][dataDimension] == forcast\_label):

            correctNum += 1

        else:

            wrongNum += 1

    print("------------------------------------")

    print("|WrongNums:",wrongNum,"correctNums:",correctNum)

    print("------------------------------------\n\n\n")

    Accuracy = correctNum/(correctNum+wrongNum)

    return  Accuracy

if \_\_name\_\_ == "\_\_main\_\_":

    Accuracy\_AVE = 0

    Accuracy = []

    Accuracy.append(0)

    testTimes = 20

    for i in range(testTimes):

        temp = K\_Nearest\_Neighbors("sonar.all-data",208,60,2,2/5,5)

        # temp = Nearest\_Neighbors("sonar.all-data",208,60,2,2/5)

        Accuracy.append(temp)

        plt.scatter(i+1,temp)

        Accuracy\_AVE += temp

    Accuracy\_AVE /= testTimes

    for i in range(testTimes):

        print("Accuracy\_",i,": ",Accuracy[i+1])

    print("Accuracy\_AVE:",Accuracy\_AVE)

    plt.title("Accuracy")

    plt.xlabel("testTimes")

    plt.ylabel("accuracy")

    plt.ylim(0,2)

    plt.plot(Accuracy)

    plt.show()