### 实验五 近邻分类器的设计

### 一、实验目的

1.掌握最近邻分类和k近邻分类算法

2.掌握近邻法的优化算法：剪辑近邻和压缩近邻

3.掌握分类器错误率的估算方法

### 二、实验内容及要求

1.使用IRIS数据集,每个类别选45个训练样本,5个测试样本,分别用最近邻法、k近邻法对测试样本进行分类，比较分类错误率;

2.调整k的值,比较不同的k值对错误率的影响.

3.采用剪辑近邻法和压缩近邻法对训练数据进行处理，用优化后的训练样本集对测试集进行分类，比较分类错误率

4.使用python实现

### 三、实验原理

1.近邻法原理及其决策规则

最小距离分类器：将各类训练样本划分成若干子类，并在每个子类中确定代表点。测试样本的类别则以其与这些代表点距离最近作决策。该法的缺点：所选择的代表点并不一定能很好地代表各类，其后果将使错误率增加。

以全部训练样本作为“代表点”，计算测试样本与这些“代表点”，即所有样本的距离，并以最近邻者的类别作为决策。

最近邻法决策规则：将与测试样本最近邻样本的类别作为决策的方法称为最近邻法。

对一个C类别问题，每类有Ni个样本，i＝1，…，C，则第i类ωi的判别函数



判别函数的决策规则：

若： ；则：X∈ωj

表示是ωi类的第k个样本

2.k-近邻法决策规则

在所有N个样本中找到与测试样本的k个最近邻者，其中各类别所占个数表示成ki,i＝1，…，c。

则决策规划是：

如果：

则：X∈ωj

3.剪辑近邻法

原理：从训练数据集中剔除两类交界处容易错分的样本，从而减少训练样本集，降低错误率。

步骤：

（1）将样本集X随机划分为多个子集合X={X1,X2,…,Xs}；

（2）用最近邻法,X(i+1)mod(s) 作为训练集对集合Xi进行分类；

（3）去掉步骤（2）中被错分的样本；

（4）用所有留下的样本，构成新的样本集

（5）经过k次操作，若没有样本被剪辑掉则停止。

4.压缩近邻法

原理：在剪辑近邻法的基础上，利用现有样本集，逐渐生成一个新的样本集。使该样本集在保留最少量样本的条件下, 仍能对原有样本的全部用最近邻法正确分类，那末该样本集也就能对待识别样本进行分类, 并保持正常识别率。它定义两个存储器，一个用来存放即将生成的样本集，称为Store；另一存储器则存放原样本集，称为Grabbag。

步骤：

1. 初始化： Store是空集，原样本集存入Grabbag；从Grabbag中任意选择一样本放入Store中作为新样本集第一个样本。
2. 样本集生成：在Grabbag中取出第i个样本用Store中的当前样本集按最近邻法分类。若分类错误，则将该样本从Grabbag转入Store中，若分类正确，则将该样本放回Grabbag中，对Grabbag中所有样本重复上述过程。
3. 若Grabbag中所有样本在执行第二步时没有发生转入Store的现象，或Grabbag已成空集，则算法终止，否则转入第二步。

**四、实验代码及结果**

#-\*- coding: utf-8 -\*-

import numpy as np

from matplotlib import pyplot as plt

from timeit import timeit

# 将列表中的数据切片读入矩阵

def Read(lines,m,n):

    A = np.zeros((m, n))

    A\_row = 0  # 表示矩阵的行，从0行开始

    for line in lines:  # 把lines中的数据逐行读取出来

        list = line.strip('\n').split('\t')  # 处理逐行数据：strip表示把头尾的'\n'去掉，split表示以空格来分割行数据，然后把处理后的行数据返回到list列表中

        A[A\_row:] = list[0:5]  # 把处理后的数据放到方阵A中。list[0:4]表示列表的0,1,2,3列数据放到矩阵A中的A\_row行

        A\_row += 1  # 然后方阵A的下一行接着读

    return A

# 数据集初始化

def DataInit():

    '''1.读取训练集和测试集'''

    f1,f2 = open('train.txt'),open('test.txt') # 打开训练集和测试集

    lines1,lines2 = f1.readlines(),f2.readlines() # 把全部数据文件读到一个列表lines中

    Line1,Line2 = len(lines1),len(lines2) # 读取训练集合测试集的行数

    A,B = Read(lines1,Line1,5),Read(lines2,Line2,5)

    return A,B

# k近邻法函数

def k\_nearest(A,B,show,k):

    train\_sum = A.shape[0]

    test\_sum = B.shape[0]

    train\_id = A[:,0] # 训练集样本类别

    test\_id = B[:,0] # 测试集样本类别

    A,B = np.delete(A,[0],axis=1),np.delete(B,[0],axis=1) # 删除标签

    true,false = 0,0

    for i in range(test\_sum):

        num = np.zeros(4) # 记录存放距离最近的k个点各自类别出现次数

        dis = np.zeros((train\_sum,2))

        for j in range(train\_sum):

            dis[j,0] = train\_id[j] # 训练集原属标签

            dis[j,1] = np.sqrt(sum(np.power(A[j,:]-B[i,:],2))) # type: ignore # 欧式距离

        order = dis[np.lexsort(dis.T)] # 按距离从近到远升序排序

        for top in range(k): # 找到距离该测试集前k近的训练集点

            index = order[top,0].astype(int)

            num[index] += 1

        Id = num.argmax()

        if Id == test\_id[i]: # 如果预测标号等于我们的实际测试集标号

            true += 1

        else:

            false += 1

            if show == True:

                print("错将测试集中的第%d组数据分成第%d类,正确类别为第%d类"%(i+1,Id,test\_id[i]))

    if k==1:

        str = "最近邻法"

    else:

        str = "k近邻法"

    if show == True:

        print("基于%s对三类样本进行分类："%str)

        print("正确个数：",true)

        print("错误个数：",false)

        print("准确率：",true/(true+false),'\n')

    else:

        return true/(true+false)

# 对训练集进行剪切

def cut\_train(A,B,ok):

    # ok = True

    k = 1

    train\_sum = A.shape[0]

    test\_sum = B.shape[0]

    train\_id = A[:,0] # 训练集样本类别

    test\_id = B[:,0] # 测试集样本类别

    wrong = [] # 记录错误样本的标号

    for i in range(test\_sum):

        num = np.zeros(4) # 记录存放距离最近的k个点各自类别出现次数

        dis = np.zeros((train\_sum,2))

        for j in range(train\_sum):

            dis[j,0] = train\_id[j] # 训练集原属标签

            dis[j,1] = np.sqrt(sum(np.power(A[j,1:]-B[i,1:],2))) # type: ignore # 欧式距离

        order = dis[np.lexsort(dis.T)] # 按距离从近到远升序排序

        for top in range(k): # 找到距离该测试集前k近的训练集点

            index = order[top,0].astype(int)

            num[index] += 1

        Id = num.argmax()

        if Id != test\_id[i]: # 如果预测标号不等于我们的实际测试集标号

            wrong.append(i)

            ok = False

    B = np.delete(B,wrong,axis=0)

    return A,B,ok

# 训练校验过程

def get\_train\_cut(A\_random):

    # 将打乱的训练集拆分成五组进行迭代训练

    ok = False # 全部剪辑完毕则为yes

    count = 0

    '''在这里进行一个创新改进，连续十轮随机剪辑后没有错分对象才可以认为剪辑完毕'''

    while(not ok):

        ok = True

        np.random.shuffle(A\_random) # 随机打乱训练集

        train\_sum = A\_random.shape[0]

        divide = train\_sum // 5

        A1,A2,A3,A4,A5 = A\_random[0:divide,:],A\_random[divide:2\*divide,:],A\_random[2\*divide:3\*divide,:],A\_random[3\*divide:4\*divide,:],A\_random[4\*divide:train\_sum ,:]

        # 修正的过程

        A2,A1,ok = cut\_train(A2,A1,ok)

        A3,A2,ok = cut\_train(A3,A2,ok)

        A4,A3,ok = cut\_train(A4,A3,ok)

        A5,A4,ok = cut\_train(A5,A4,ok)

        A1,A5,ok = cut\_train(A1,A5,ok)

        A\_random = np.concatenate((A1,A2,A3,A4,A5),axis=0)

        if ok==True:

            count +=1

            if count == 10:

                ok = True

            else:

                ok = False

        else:

            count = 0

    return A\_random

# 对剪切过的训练集进行压缩

def get\_train\_compress(A\_cut):

    length = A\_cut.shape[0]

    Store = np.array([A\_cut[0,:]]) # 剪辑后的训练集第一组数据默认放入Store

    Grabbag = A\_cut[1:length,:] # 剪辑后的训练集除第一组以外均放入Grabbag

    count = 0 # 计数连续多少个样本测试正确

    Len = Grabbag.shape[0] # 当前Grabbag有多少数据

    while 1:

        test = np.array([Grabbag[0,:]]) # 取出待测试样本

        Grabbag = np.delete(Grabbag,0,axis=0) # 从Grabbag中去除

        cur = k\_nearest(Store,test,False,1)

        if cur == 1: # 分类正确

            Grabbag = np.insert(Grabbag,Grabbag.shape[0],test,axis=0) # 尾插回去

            count += 1

        else : # 分类错误

            Store = np.insert(Store,Store.shape[0],test,axis=0) # 转入Store

            count = 0

            Len = Grabbag.shape[0] # 更新

        if Grabbag.shape[0] == 0 or count == Len:

            break

    return Store

'''

    一、数据集初始化：

    其中A,B分别为训练集和测试集

    并将训练集扩充成5倍，增大训练集数目

'''

A,B = DataInit()

# t = timeit('DataInit(k)', 'from \_\_main\_\_ import DataInit', number=1000)

# print(t)

A = np.concatenate((A,A,A,A,A),axis=0) # 将训练集扩充成5倍

B = np.concatenate((B,B,B),axis=0) # 测试集扩充成3倍

'''

    二、最近邻法：

    计算测试集点到训练集每一个点的距离，

    距离测试点最近的训练集点的类别作为该测试集的预估值。

'''

k\_nearest(A,B,True,1) # k=1的k近邻就是最近邻法

'''

    三、k近邻法：

    计算测试集点到训练集每一个点的距离，

    统计距离测试点最近的k个训练集点,

    这k个训练集点出现的类别次数最多的那类作为该测试集的预估值。

'''

k\_nearest(A,B,True,30)

'''

    四、通过图表可视化观察k从1到200变化时

    对应的准确率的变化

'''

plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode\_minus']=False #用来正常显示负号

x = list(range(1,201))

y = []

for i in range(1,201):

    point = k\_nearest(A,B,False,i)

    y.append(point)

plt.plot(x,y,label="测试准确率")

plt.xlabel("近邻法的k值")

plt.ylabel("测试准确率")

plt.title("k从1到200变化时近邻法识别准确率的变化")

plt.show()

'''

    五、剪辑近邻法

    先对训练集进行多重剪辑处理，得到一个新的训练集

'''

A\_cut = get\_train\_cut(A) # 剪辑训练集

cut = A.shape[0]-A\_cut.shape[0]

print("训练集本有%d组数据,剪辑掉%d组,最终训练集剩余%d组"%(A.shape[0],cut,A\_cut.shape[0]))

k\_nearest(A\_cut,B,True,1) # 使用最近邻法进行分类

'''

    六、压缩近邻法

    在Grabbag中取出第i个样本用Store中的当前样本集按最近邻法分类。

    若分类错误，则将该样本从Grabbag转入Store中，

    若分类正确，则将该样本放回Grabbag中，对Grabbag中所有样本重复上述过程。

'''

A\_compress = get\_train\_compress(A\_cut)

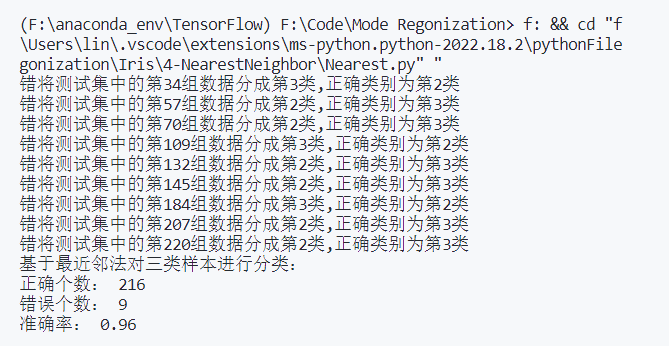
compress = A\_cut.shape[0]-A\_compress.shape[0]

print("剪辑后的训练集本有%d组数据,压缩掉%d组,最终训练集剩余%d组"%(A\_cut.shape[0],compress,A\_compress.shape[0]))

k\_nearest(A\_compress,B,True,1)

**五、实验分析**

**（1）最近邻法**

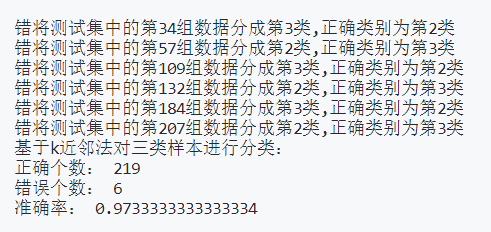


计算了测试集点到训练集每一个点的距离，距离测试点最近的训练集点的类别作为该测试集的预估值。

最终225个测试集正确216，错误9，准确率96%

**（2）k近邻法**

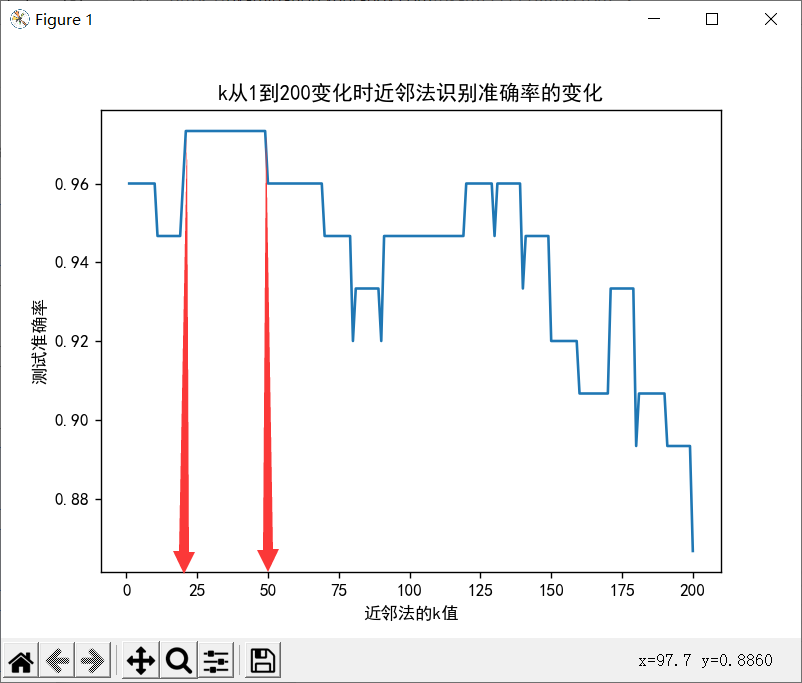
选取k=30：



最终225个测试集正确219，错误6，准确率97.33%

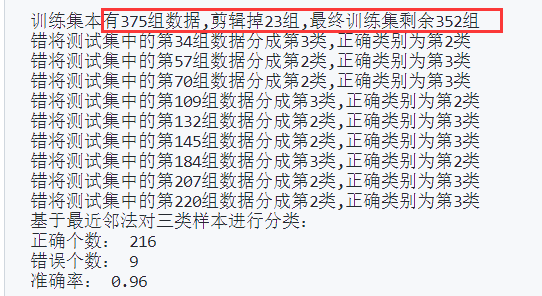
**（3）模拟不同k值对分类结果的影响**

特此进行k从1到200取值进行近邻法的检验，观察其准确率的变化



k在大约20~50之间取值能够达到最理想的效果。

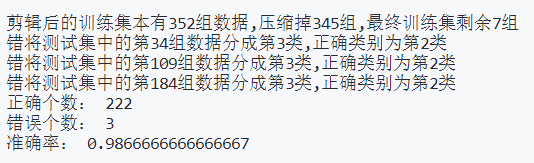
**（4）剪辑近邻法**



最终225个测试集正确219，错误9，准确率96%

【由于剪辑的训练集是随机迭代出来的，所以每一次运行结果均不同，我们可以运行多次最终找到最适合的剪辑训练集】

**（5）压缩近邻法**



效果格外神奇，仅用7组训练集实现了225组测试集的分类，且准确率达到了新的水准：225个测试集正确222，错误3，准确率98.67%