**Lab 6 kNN应用实践**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号 | 姓名 | 年级 | 专业 |
| 18353070 | 谭嘉伟 | 2018 | 软件工程 |

1. 实验目的：
2. 利用Python实现kNN分类器
3. 进一步熟练用Python实现算法的能力
4. 进一步理解掌握kNN算法
5. 实验环境：

1、硬件环境：PC 1.60GHz四核处理器 8.00G内存

2、软件环境：Windows10 64位操作系统

Python3.5.6 tensorflow1.4.0

1. 实验步骤：
2. 实现kNN分类器

导入knn1代码后稍加修改，编辑调用函数得到以下代码

#knn1+

import numpy as np

import operator

#创建简单数据集

def createDataSet():

group = np.array([[1.0,1.1],[1.0,1.0],[0,0],[0,0.1]])

labels = ['A','A','B','B']

return group,labels

# k近邻算法

def classify(inX,dataSet,labels,k):

# inX用于分类的输入向量

# dataSet输入的训练样本集

# labels为标签向量

# k为最近的邻居数目

# 计算距离

dataSetSize = dataSet.shape[0]

diffMat = np.tile(inX,(dataSetSize,1)) - dataSet

sqDiffMat = diffMat \*\* 2

sqDistances = sqDiffMat.sum(axis=1)

distances = sqDistances \*\* 0.5

sortedDistIndicies = distances.argsort()

classCount = {}

# 选择距离最小的k个点

for i in range(k):

voteIlabel = labels[sortedDistIndicies[i]]

classCount[voteIlabel] = classCount.get(voteIlabel,0) +1

# 排序

sortedClassCount = sorted(classCount.items(),key = operator.itemgetter(1),reverse= True)

return sortedClassCount[0][0]

group,labels=createDataSet()

print(group)

print(labels)

print(classify([0,0],group,labels,3))

print(classify([0,0],group,labels,2))

print(classify([0,0],group,labels,1))

print(classify([0.9,0.3],group,labels,3))

1. 使用kNN改进约会网站的配对效果

导入knn2代码，使用规范的调用库语句，得到下面代码。

#knn2+

import numpy as np

import operator

import time

# 将文本记录转换为NumPy的解析程序

def file2matrix(filename):

fr = open(filename)

#得到文件行数

arrayOfLines = fr.readlines()

numberOfLines = len(arrayOfLines)

#创建返回的Numpy矩阵

returnMat = np.zeros((numberOfLines,3))

classLabelVector = []

#解析文件数据到列表

index = 0

for line in arrayOfLines:

line = line.strip() #注释1

listFromLine = line.split('\t') #注释2

returnMat[index,:] = listFromLine[0:3]

classLabelVector.append(int(listFromLine[-1]))

index +=1

return returnMat,classLabelVector

# 归一化特征值

def autoNorm(dataSet):

minVals = dataSet.min(0)

maxVals = dataSet.max(0)

ranges = maxVals - minVals

#创建一个与dataSet同大小的零矩阵

normDataSet = np.zeros(np.shape(dataSet))

#数据行数

m = dataSet.shape[0]

#tile()函数将变量内容复制成输入矩阵同等的大小

normDataSet = dataSet - np.tile(minVals,(m,1))

#特征值相除

normDataSet = normDataSet/np.tile(ranges,(m,1))

return normDataSet,ranges,minVals

# k近邻算法

def classify0(inX,dataSet,labels,k):

#inX用于分类的输入向量

#dataSet输入的训练样本集,

#labels为标签向量,

#k用于选择最近的邻居数目

#计算距离

dataSetSize = dataSet.shape[0]

diffMat = np.tile(inX,(dataSetSize,1)) - dataSet

sqDiffMat = diffMat \*\* 2

sqDistances = sqDiffMat.sum(axis=1)

distances = sqDistances \*\* 0.5

sortedDistIndicies = distances.argsort()

classCount = {}

#选择距离最小的k个点

for i in range(k):

voteIlabel = labels[sortedDistIndicies[i]]

classCount[voteIlabel] = classCount.get(voteIlabel,0) +1

#排序

sortedClassCount = sorted(classCount.items(),key = operator.itemgetter(1),reverse= True)

return sortedClassCount[0][0]

# 分类器针对约会网站的测试代码

def datingClassTest():

#测试样本的比例

hoRatio = 0.10

datingDataMat,datingLabels = file2matrix("D:\\datingTestSet2.txt")

normMat,ranges,minVals = autoNorm(datingDataMat)

m = normMat.shape[0]

#测试样本的数量

numTestVecs = int(m\*hoRatio)

errorCount = 0.0

for i in range(numTestVecs):

classifierResult = classify0(normMat[i,:],normMat[numTestVecs:m,:],

datingLabels[numTestVecs:m],4)

print("the classifier came back with: %d, the real answer is :%d" %(classifierResult,datingLabels[i]))

if (classifierResult != datingLabels[i]): errorCount +=1.0

print("the total error rate is: %f" % (errorCount/float(numTestVecs)))

def classifyPerson():

resultList = ['not at all','in small doses','in large doses']

percentTats = float(input("percentage of time spent playing video games?"))

ffMiles = float(input("frequent flier miles earned per year?"))

iceCream = float(input("liters of ice cream consumed per year?"))

datingDataMat,datingLabels = file2matrix('D:\\datingTestSet2.txt')

normMat,ranges,minVals = autoNorm(datingDataMat)

inArr = np.array([ffMiles,percentTats,iceCream])

classifierResult = classify0((inArr - minVals)/ranges,normMat,datingLabels,3)

print("You will probably like this person:",resultList[classifierResult - 1])

datingDataMat,datingLabels=file2matrix("D:\\datingTestSet2.txt")

print(datingDataMat)

print(datingLabels[0:20])

import matplotlib

import matplotlib.pyplot as plt

plt.scatter(datingDataMat[:,1],datingDataMat[:,2])

plt.show()

plt.scatter(datingDataMat[:,1],datingDataMat[:,2],15.0\*np.array(datingLabels),15.0\*np.array(datingLabels))

normMat,ranges,minVals=autoNorm(datingDataMat)

normMat

ranges

minVals

datingClassTest()

classifyPerson()

可以通过输入框输入约会网站用户的信息，输入信息后可以得到其魅力程度的分类

1. 使用kNN识别手写字体

对代码改用调用库的规范语句和文件路径的少量修改，得到以下代码：

#knn3+

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

import operator

import time

from os import listdir

# knn实现

def classify(inputPoint,dataSet,labels,k):

'''

inputPoint:待判断的点

dataSet:数据集合

labels:标签向量，维数和dataSet行数相同，比如labels[1]代表dataSet[1]的类别

k:邻居数目

输出：inputPoint的标签

'''

dataSetSize = dataSet.shape[0]

# 先用tile函数将输入点拓展成与训练集相同维数的矩阵，再计算欧氏距离

diffMat = np.tile(inputPoint,(dataSetSize,1))-dataSet

sqDiffMat = diffMat \*\* 2

# 计算每一行元素之和

sqDistances = sqDiffMat.sum(axis=1)

# 开方得到欧拉距离矩阵

distances = sqDistances \*\* 0.5

# argsort返回的是数组从小到大的元素的索引

# 按distances中元素进行升序排序后得到的对应下标的列表

sortedDistIndicies = distances.argsort()

# 选择距离最小的k个点，统计每个类别的点的个数

classCount = {}

for i in range(k):

voteIlabel = labels[ sortedDistIndicies[i] ]

classCount[voteIlabel] = classCount.get(voteIlabel,0)+1

# 按classCount字典的第2个元素（即类别出现的次数）从大到小排序

sortedClassCount = sorted(classCount.items(), key = operator.itemgetter(1), reverse = True)

return sortedClassCount[0][0]

# 将图片矩阵转换为向量

def img2vector(filename):

'''

filename:文件名字

将这个文件的所有数据按照顺序写成一个一维向量并返回

'''

returnVect = []

fr = open(filename)

for i in range(32):

lineStr = fr.readline()

for j in range(32):

returnVect.append(int(lineStr[j]))

return returnVect

# 从文件名中解析分类数字

def classnumCut(fileName):

'''

filename:文件名

返回这个文件数据代表的实际数字

'''

fileStr = fileName.split('.')[0]

classNumStr = int(fileStr.split('\_')[0])

return classNumStr

# 构建训练集数据向量及对应分类标签向量

def trainingDataSet():

'''

从trainingDigits文件夹下面读取所有数据文件，返回：

trainingMat：所有训练数据，每一行代表一个数据文件中的内容

hwLabels：每一项表示traningMat中对应项的数据到底代表数字几

'''

hwLabels = []

# 获取目录traningDigits内容(即数据集文件名)，并储存在一个list中

trainingFileList = listdir('D:\\trainingDigits')

m = len(trainingFileList) #当前目录文件数

# 初始化m维向量的训练集，每个向量1024维

trainingMat = np.zeros((m,1024))

for i in range(m):

fileNameStr = trainingFileList[i]

# 从文件名中解析分类数字，作为分类标签

hwLabels.append(classnumCut(fileNameStr))

# 将图片矩阵转换为向量并储存在新的矩阵中

trainingMat[i,:] = img2vector('D:\\trainingDigits/%s' % fileNameStr)

return hwLabels,trainingMat

def createDataSet():

'''

从trainingDigits文件夹下面读取所有数据文件，返回：

trainingMat：所有训练数据，每一行代表一个数据文件中的内容

hwLabels：每一项表示traningMat中对应项的数据到底代表数字几

'''

hwLabels = []

# 获取目录traningDigits内容(即数据集文件名)，并储存在一个list中

trainingFileList = listdir('D:\\trainingDigits')

m = len(trainingFileList) #当前目录文件数

# 初始化m维向量的训练集，每个向量1024维

trainingMat = np.zeros((m,1024))

for i in range(m):

fileNameStr = trainingFileList[i]

# 从文件名中解析分类数字，作为分类标签

hwLabels.append(classnumCut(fileNameStr))

# 将图片矩阵转换为向量并储存在新的矩阵中

trainingMat[i,:] = img2vector('D:\\trainingDigits/%s' % fileNameStr)

return trainingMat,hwLabels

# 测试函数

def handwritingTest():

# 构建训练集

hwLabels,trainingMat = trainingDataSet()

#从testDigits里面拿到测试集

testFileList = listdir('D:\\testDigits')

# 错误数

errorCount = 0.0

# 测试集总样本数

mTest = len(testFileList)

# 获取程序运行到此处的时间（开始测试）

t1 = time.time()

for i in range(mTest):

# 得到当前文件名

fileNameStr = testFileList[i]

# 从文件名中解析分类数字

classNumStr = classnumCut(fileNameStr)

# 将图片矩阵转换为向量

vectorUnderTest = img2vector('D:\\testDigits/%s' % fileNameStr)

# 调用knn算法进行测试

classifierResult = classify(vectorUnderTest, trainingMat, hwLabels, 5)

print ("the classifier came back with: %d, the real answer is: %d" % (classifierResult, classNumStr))

# 预测结果不一致，则错误数+1

if (classifierResult != classNumStr): errorCount += 1.0

print("\nthe total number of tests is: %d" % mTest)

print("the total number of errors is: %d" % errorCount )

print("the total error rate is: %f" % (errorCount/float(mTest)))

# 获取程序运行到此处的时间（结束测试）

t2 = time.time()

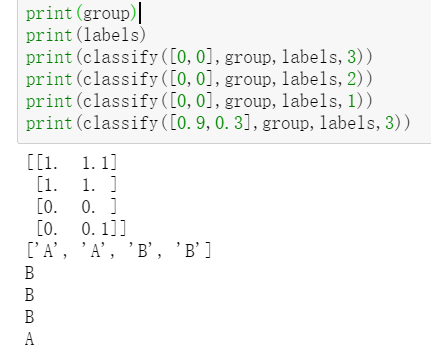
# 测试耗时

print ("Cost time: %.2fmin, %.4fs."%((t2-t1)//60,(t2-t1)%60) )

if \_\_name\_\_ == "\_\_main\_\_":

handwritingTest()

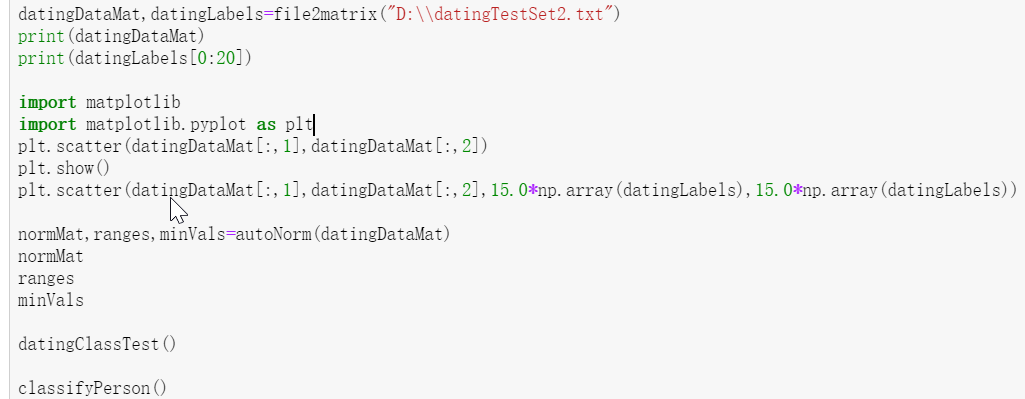
1. 实验结果与分析：
2. 实现kNN分类器



由于(0,0)与训练集中的一个B点重合，因此k=1时返回B，其他时候由于相对两个B点的距离还是较近，因此k=1,2,3均返回B

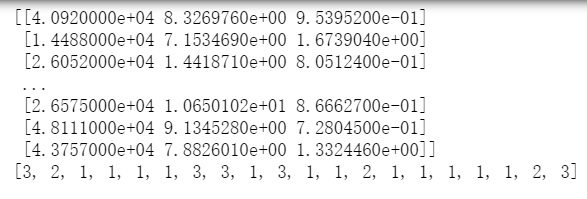
查找(0.9,0.3)时则附近有较多的A点，k=3时返回A

1. 改进约会网站配对效果

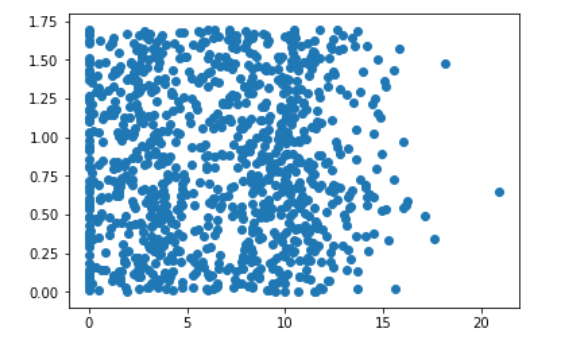


调用运行的代码见上图

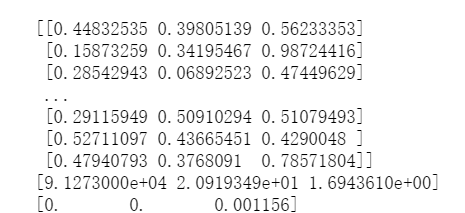
输出的数据结果为



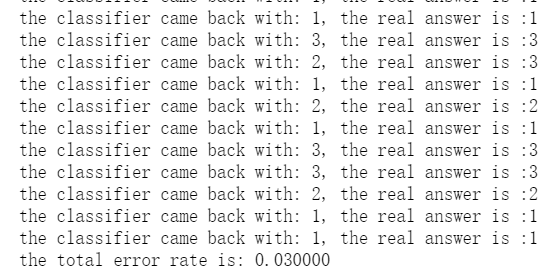
约会数据可视化后得到



数据归一化后的数据为：

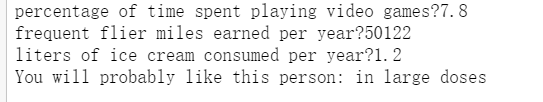


调用测试系统

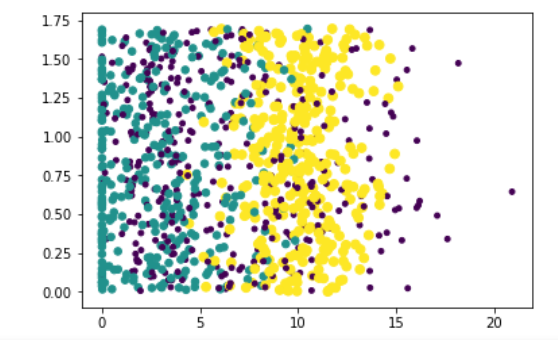


可见错误率为0.03，错误概率较低，分类效果较好

输入需要预测的人的信息：

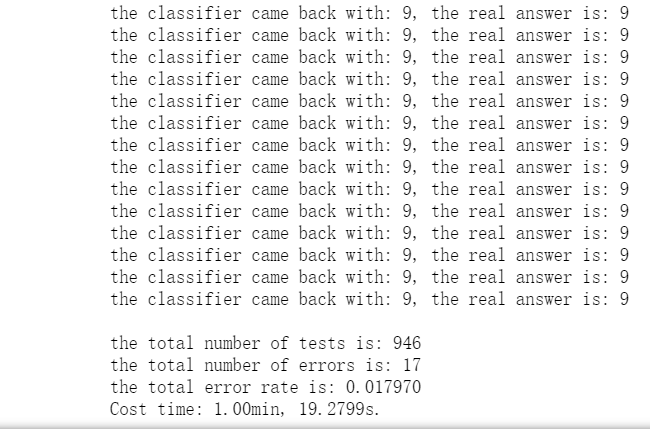


即可得到预测结论：具有很大概率

对数据点分类后进行可视化的图像如下：  


1. 使用kNN识别手写体：

程序需要比较长时间运行，成功运行后得到以下运行结果



程序时间复杂度较高，运行1min19s才完成运行。这是由于计算时每个测试向量都需要进行1934次的欧氏距离计算，而总共有946个测试向量。因此需要进行1934\*946=约两百万次欧氏距离计算，而每次欧式距离计算需要计算1024个维度，因此需要约两亿次运算，这是这个算法执行的瓶颈，是执行的主要计算量。如果需要优化速度的话，可以考虑进行这样的优化：将1024\*1的01向量压缩成32个整数向量，用32个整数的异或运算代替1024个布尔量的比较运算，可能可以使算法的执行速度有所提升。

测试集有946个向量，运行结果有17个错误结果，总错误率为1.79%，执行效果较好。

数据集和训练集的数字比较规范，它们都在位图的正中央附近，而且大小相近，这样识别效果较好。如果识别真正的书写不太规范的手写数字，可能识别效果不会太好。所以面对真实的手写数字位图，可能需要一定的归一化处理。

空间复杂度方面，需要保存所有位图信息，即需要保存个2000个1\*1024的向量。由于现代计算机的内存都较大，因此空间复杂度还可以接受。

1. 操作练习：
2. 对于knn2，测试不同k值对错误率的影响：

用下面的代码循环枚举k=1..149，测试不同的k值下的错误率

datingDataMat,datingLabels = file2matrix("D:\\datingTestSet2.txt")

normMat,ranges,minVals = autoNorm(datingDataMat)

kx=[]

ky=[]

for k in range(1,150):

#测试样本的比例

hoRatio = 0.10

m = normMat.shape[0]

#测试样本的数量

numTestVecs = int(m\*hoRatio)

errorCount = 0.0

for i in range(numTestVecs):

classifierResult = classify0(normMat[i,:],normMat[numTestVecs:m,:],

datingLabels[numTestVecs:m],k)

if (classifierResult != datingLabels[i]): errorCount +=1.0

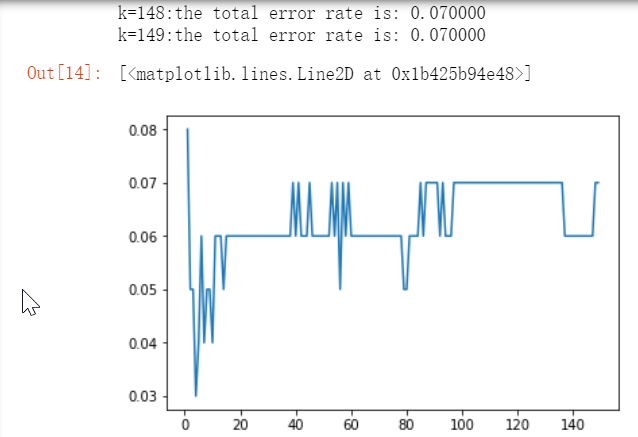
print("k=%d:the total error rate is: %f" % (k,errorCount/float(numTestVecs)))

kx.append(k)

ky.append(errorCount/float(numTestVecs))

plt.plot(kx,ky)

运行结果可视化后的结果为：



横坐标为kNN算法k的选择，纵坐标为错误率

可以发现k=4时，错误率为极小值0.03，k变小或变大时，错误率均会上升，因此k=4为这个例子合适的参数

1. 对于knn2，测试不同参数的闵可夫斯基距离对错误率的影响：

修改距离函数与测试代码如下

import numpy as np

import operator

import time

def classify0new(inX,dataSet,labels,k,dim):

#inX用于分类的输入向量

#dataSet输入的训练样本集,

#labels为标签向量,

#k用于选择最近的邻居数目

#计算距离

dataSetSize = dataSet.shape[0]

diffMat = np.tile(inX,(dataSetSize,1)) - dataSet

sqDiffMat = abs(diffMat) \*\* dim

sqDistances = sqDiffMat.sum(axis=1)

distances = sqDistances \*\* (1.0/dim)

sortedDistIndicies = distances.argsort()

classCount = {}

#选择距离最小的k个点

for i in range(k):

voteIlabel = labels[sortedDistIndicies[i]]

classCount[voteIlabel] = classCount.get(voteIlabel,0) +1

#排序

sortedClassCount = sorted(classCount.items(),key = operator.itemgetter(1),reverse= True)

return sortedClassCount[0][0]

datingDataMat,datingLabels = file2matrix("D:\\datingTestSet2.txt")

normMat,ranges,minVals = autoNorm(datingDataMat)

kx=[]

ky=[]

for k in range(1,30):

#测试样本的比例

hoRatio = 0.10

m = normMat.shape[0]

#测试样本的数量

numTestVecs = int(m\*hoRatio)

errorCount = 0.0

for i in range(numTestVecs):

classifierResult = classify0new(normMat[i,:],normMat[numTestVecs:m,:],

datingLabels[numTestVecs:m],4,k)

if (classifierResult != datingLabels[i]): errorCount +=1.0

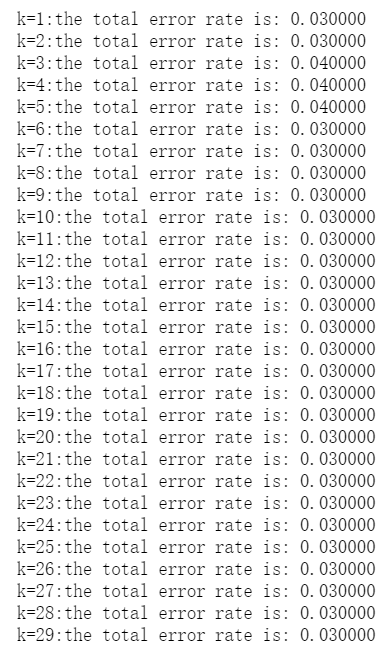
print("k=%d:the total error rate is: %f" % (k,errorCount/float(numTestVecs)))

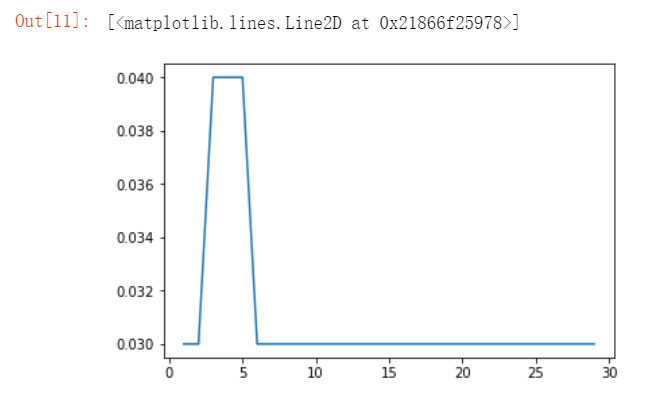
kx.append(k)

ky.append(errorCount/float(numTestVecs))

plt.plot(kx,ky)

运行结果为：





横坐标为闵可夫斯基距离指数，纵坐标为错误率

k对应为闵可夫斯基距离的不同次数，k=1为曼哈顿距离，k=2为欧几里得距离,k=∞为切比雪夫距离。

所以在k=1，距离函数为曼哈顿距离时，错误率仍然为3%，与k=2的欧几里得距离相同。对于k=1..49，错误率均在0.3~0.4之间，变化幅度不大。

1. 随机选取训练样本，测试不同样本数目对错误率的影响：

增加选择不同训练集比例的循环，在循环内部进行随机选取训练样本的处理。每次用numpy库生成一个随机排列，然后用这个随机排列构造一个排列顺序和原始数据集不同的数据集，然后将前0~numTrainVecs-1编号的数据划分为训练集，后面numTrainVecs~m-1编号的数据划分为测试集进行测试，使得训练集和测试集没有交集，可以得到不同样本比例对错误率的曲线。

代码如下：

datingDataMat,datingLabels = file2matrix("D:\\datingTestSet2.txt")

normMat,ranges,minVals = autoNorm(datingDataMat)

kx=[]

ky=[]

for k in range(1,95):

#测试样本的比例

hoRatio =k/100#不同训练集的比例

m = normMat.shape[0]

#测试样本的数量

numTrainVecs = int(m\*hoRatio)

errorCount = 0.0

tot=0

p=np.random.permutation(m)#生成随机排列

#p=[i for i in range(m)]

mat=[]

label=[]

for i in range(m):

mat.append(normMat[p[i],:])#用随机排列打乱数据顺序

label.append(datingLabels[p[i]])

mat=np.array(mat)#强制从list类型转换为ndarray数据类型

#print(mat)

for i in range(numTrainVecs,m):#后面numTrainVecs~m-1的数据为测试集，0~numTrainVecs-1的数据为训练集

tot+=1

classifierResult = classify0(mat[i,:],mat[0:numTrainVecs,:],

label[0:numTrainVecs],4)

if (classifierResult != label[i]): errorCount +=1.0

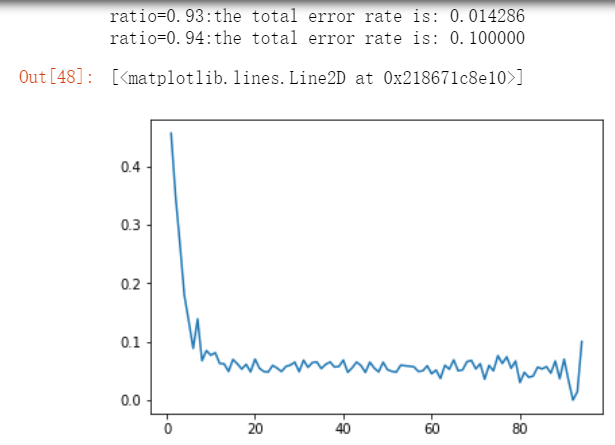
print("ratio=%.2f:the total error rate is: %f" % (hoRatio,errorCount/float(tot)))

kx.append(k)

ky.append(errorCount/float(tot))

plt.plot(kx,ky)

运行结果如下：



横坐标为训练集大小百分比，纵坐标为错误率

原始程序对应训练集大小为90%，对应的错误率为0.03，上图的曲线与这个值相符。可以看出，在训练集比例很小时，错误率较高，可以达到接近50%；随着训练集比例增大，错误率迅速降低；当训练集比例达到8%后，错误率上下波动稳定在0.05附近，之后变化趋势不大；而当训练集比例过大，测试集比例过小时，这时错误率受随机因素影响变化较大。

1. 利用sklearn实现使用kNN改进约会网站的配对效果和使用kNN识别手写体应用。并分别比较应用PCA前后的效果。

导入sklearn库，使用以下代码得到kNN改进约会网站配对效果的结果:

import sklearn

from sklearn.neighbors import KNeighborsClassifier

datingDataMat,datingLabels = file2matrix("D:\\datingTestSet2.txt")

normMat,ranges,minVals = autoNorm(datingDataMat)

skknn2=KNeighborsClassifier(n\_neighbors=4)

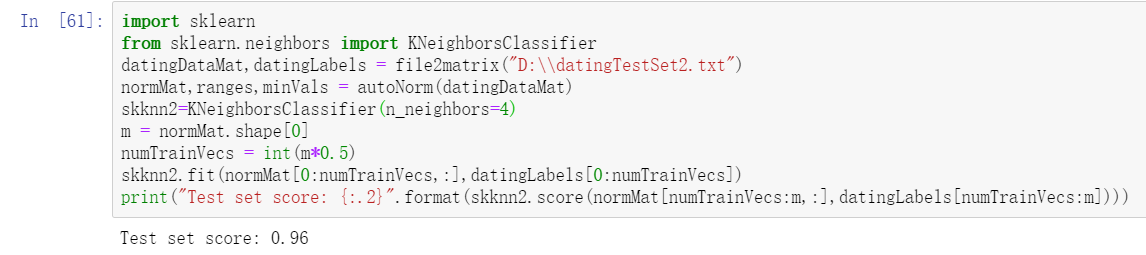
m = normMat.shape[0]

numTrainVecs = int(m\*0.5)

skknn2.fit(normMat[0:numTrainVecs,:],datingLabels[0:numTrainVecs])

print("Test set score: {:.2}".format(skknn2.score(normMat[numTrainVecs:m,:],datingLabels[numTrainVecs:m])))

结果如下：



归一化后的数据正确率较高，达到0.96

使用PCA降维处理的代码：

print(normMat.shape)

from sklearn.decomposition import PCA

pca=PCA()

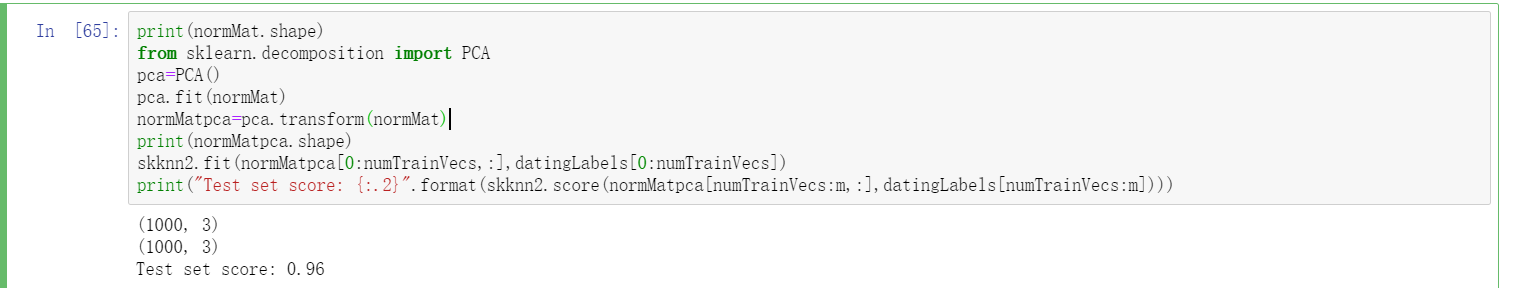
pca.fit(normMat)

normMatpca=pca.transform(normMat)

print(normMatpca.shape)

skknn2.fit(normMatpca[0:numTrainVecs,:],datingLabels[0:numTrainVecs])

print("Test set score: {:.2}".format(skknn2.score(normMatpca[numTrainVecs:m,:],datingLabels[numTrainVecs:m])))



可以看出knn2三个分量相关性不强，使用PCA降维不会自动压缩维度，正确率仍为0.96，保持不变

而如果用下面的代码强行选出2个维度来降维：

print(normMat.shape)

from sklearn.decomposition import PCA

pca=PCA(n\_components=2)

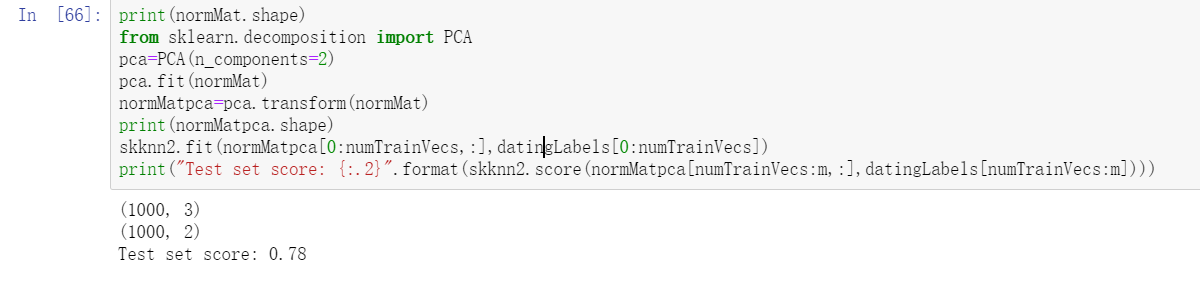
pca.fit(normMat)

normMatpca=pca.transform(normMat)

print(normMatpca.shape)

skknn2.fit(normMatpca[0:numTrainVecs,:],datingLabels[0:numTrainVecs])

print("Test set score: {:.2}".format(skknn2.score(normMatpca[numTrainVecs:m,:],datingLabels[numTrainVecs:m])))



则正确率反而会下降

用sklearn处理knn3手写体识别的代码如下：

def getTestDataSet():

hwLabels = []

# 获取目录traningDigits内容(即数据集文件名)，并储存在一个list中

trainingFileList = listdir('D:\\testDigits')

m = len(trainingFileList) #当前目录文件数

# 初始化m维向量的训练集，每个向量1024维

trainingMat = np.zeros((m,1024))

for i in range(m):

fileNameStr = trainingFileList[i]

# 从文件名中解析分类数字，作为分类标签

hwLabels.append(classnumCut(fileNameStr))

# 将图片矩阵转换为向量并储存在新的矩阵中

trainingMat[i,:] = img2vector('D:\\testDigits/%s' % fileNameStr)

return hwLabels,trainingMat

import sklearn

from sklearn.neighbors import KNeighborsClassifier

hwLabels,trainingMat = trainingDataSet()

testLabels,testMat=getTestDataSet()

skknn3=KNeighborsClassifier(n\_neighbors=5)

skknn3.fit(trainingMat,hwLabels)

print("Test set score: {:.6}".format(skknn3.score(testMat,testLabels)))



使用参数k=5，正确率较高，达到98%以上

再使用PCA降维处理：

由于测试集必须要经过和训练集同样的降维变换处理，才能正确用作训练集。为了保证训练集与测试集内容属性一致性，这里采用训练集作为测试集

hwLabels,trainingMat = trainingDataSet()

skknn3.fit(trainingMatpca,hwLabels)

print("Test set score: {:.6}".format(skknn3.score(trainingMatpca,hwLabels)))

print(trainingMat.shape)

from sklearn.decomposition import PCA

pca=PCA()

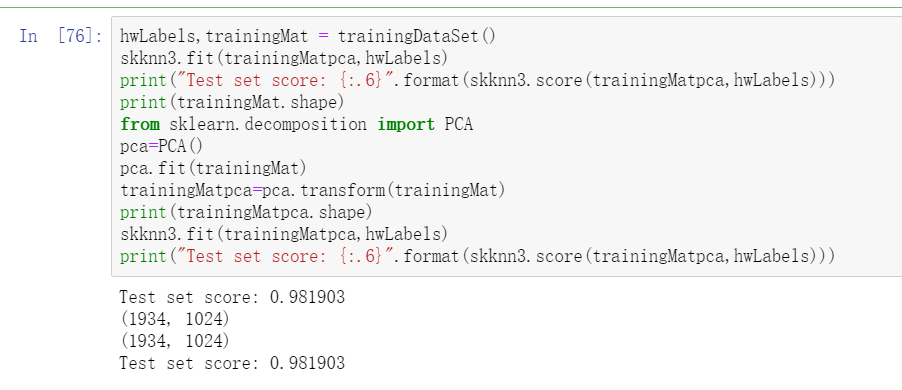
pca.fit(trainingMat)

trainingMatpca=pca.transform(trainingMat)

print(trainingMatpca.shape)

skknn3.fit(trainingMatpca,hwLabels)

print("Test set score: {:.6}".format(skknn3.score(trainingMatpca,hwLabels)))



由于01矩阵内容过于简单，且大多独立，没有相关性，因此降维前后错误率没有发生改变

1. 用Tensorflow实现knn1,knn2,knn3

#tensorflow knn1

def createDataSet1():

group = np.array([[1.0,1.1],[1.0,1.0],[0,0],[0,0.1]])

labels = ['A','A','B','B']

return group,labels

import tensorflow as tf

group,labels=createDataSet1()#group为训练集数据，labels为训练集标签

test=group#test为测试集数据

testlab=labels#testlab为测试集标签

tftrain=tf.placeholder("float",[None,2])

tftest=tf.placeholder("float",[2])

#distance=tf.reduce\_sum(tf.abs(tf.add(tftrain,tf.negative(tftest))),axis=1)

distance=tf.sqrt(tf.reduce\_sum(tf.square(tf.add(tftrain,tf.negative(tftest))),axis=1))#欧几里得距离

pred=tf.argmin(distance,0)

accuracy=0

init = tf.global\_variables\_initializer()

with tf.Session() as sess:

sess.run(init)

m=len(test)

for i in range(m):

index=sess.run([pred,distance],feed\_dict={tftrain:group,tftest:test[i,:]})

pred\_label=labels[index[0]]#index[0]为pred预测值

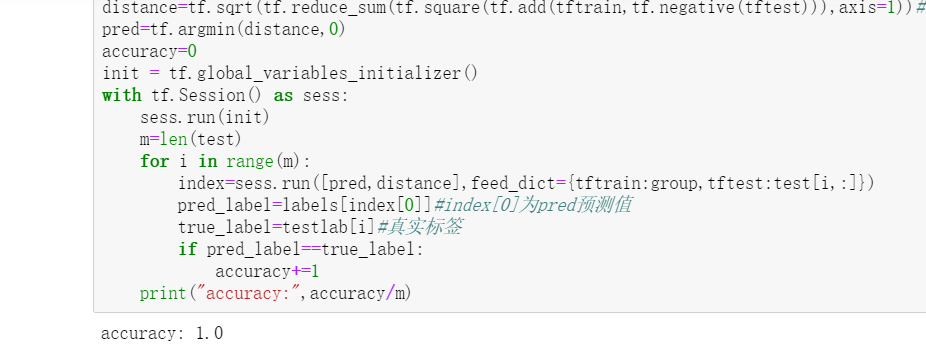
true\_label=testlab[i]#真实标签

if pred\_label==true\_label:

accuracy+=1

print("accuracy:",accuracy/m)

tensorflow实现Knn1中，由于没有大量来源于实际的测试数据，将训练集数据作为测试数据，得到测试结果：



tensorflow实现knn2中，将前ratio(50%)数据作为训练集，后1-ration的比例(50%)数据作为测试集，采取欧几里得距离和最近邻算法，代码和得到的运行结果如下：

#tensorflow knn2

datingDataMat,datingLabels = file2matrix("D:\\datingTestSet2.txt")

normMat,ranges,minVals = autoNorm(datingDataMat)

#print(normMat.shape)

import tensorflow as tf

hoRatio=0.5

m=normMat.shape[0]

numTrainVecs = int(m\*hoRatio) #前50%划分为训练集，后50%划分为测试集

tftrain=tf.placeholder("float",[None,3])

tftest=tf.placeholder("float",[3])

#distance=tf.reduce\_sum(tf.abs(tf.add(tftrain,tf.negative(tftest))),axis=1)

distance=tf.sqrt(tf.reduce\_sum(tf.square(tf.add(tftrain,tf.negative(tftest))),axis=1))#欧几里得距离

pred=tf.argmin(distance,0)

accuracy=0

init = tf.global\_variables\_initializer()

with tf.Session() as sess:

sess.run(init)

tot=0

for i in range(numTrainVecs,m):

index=sess.run([pred,distance],feed\_dict={tftrain:normMat[0:numTrainVecs,:],tftest:normMat[i,:]})

pred\_label=datingLabels[index[0]]#index[0]为pred预测值

true\_label=datingLabels[i]#真实标签

if pred\_label==true\_label:

accuracy+=1

tot+=1

print("accuracy:",accuracy/tot)



分类较为准确，正确率达到95%

tensorflow实现knn3中，采用欧几里得距离和最近邻算法，使用所有读入的测试集和训练集数据，代码和计算结果如下：

#tensorflow knn3

hwLabels,trainingMat = trainingDataSet()

testLabels,testMat=getTestDataSet()

#rint(testMat.shape)

import tensorflow as tf

tftrain=tf.placeholder("float",[None,1024])#1\*1024的01向量

tftest=tf.placeholder("float",[1024])

#distance=tf.reduce\_sum(tf.abs(tf.add(tftrain,tf.negative(tftest))),axis=1)

distance=tf.sqrt(tf.reduce\_sum(tf.square(tf.add(tftrain,tf.negative(tftest))),axis=1))#欧几里得距离

pred=tf.argmin(distance,0)

accuracy=0

init = tf.global\_variables\_initializer()

with tf.Session() as sess:

sess.run(init)

m=testMat.shape[0]

for i in range(m):

index=sess.run([pred,distance],feed\_dict={tftrain:trainingMat[:,:],tftest:testMat[i,:]})

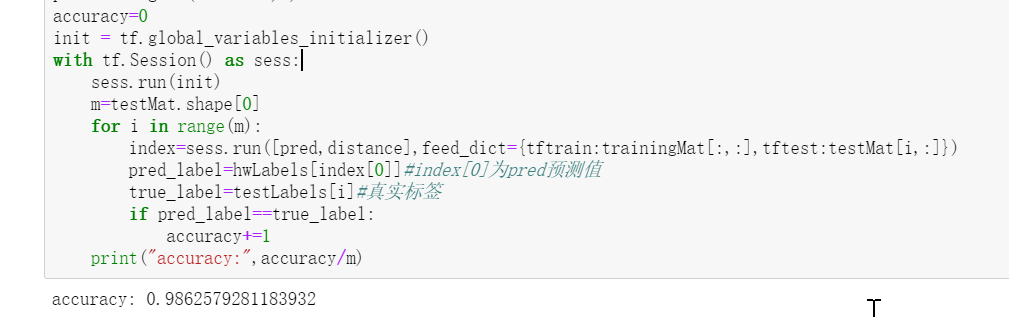
pred\_label=hwLabels[index[0]]#index[0]为pred预测值

true\_label=testLabels[i]#真实标签

if pred\_label==true\_label:

accuracy+=1

print("accuracy:",accuracy/m)



训练结果较为准确，正确率高达98.6%

1. 实验总结：
2. 随机选取训练样本及让训练集和测试集没有交集时遇到了一定困难。后面考虑通过随机生成原始数据的排列，并按照比例将排列置换后的新数据集前一半和后一半分别作为测试集和训练集的做法。由于要保证每个数据和标签一一对应，所以不能用shuffle()随便打乱，必须要先生成一个排列p，然后将数据和标签的每个元素都按照这个p一一对应置换，这样保证排列置换后数据仍与标签一一对应。
3. 类型转换：考虑将normMat按照p的置换生成一个新的同样类型的数据集mat。由于采用mat=[]，mat.append(normMat[p[i],:])的方式构造mat，后面将mat作为新参数传入函数时发生了错误。这是因为mat数据类型与normMat不相同。经过print(mat),print(normMat),print(type(mat)),print(type(normMat))语句后成功确定了他们的数据类型，用numpy.array将mat转换为了与normMat相同的ndarray类型，解决了这个问题。
4. Tensorflow和sklearn都可以只用很短地代码方便地实现knn算法。由于sklearn的knn是封装好的测试过程，因此健壮性较好，适应性应该也较强。而Tensorflow需要自己按照算法流程实现knn的计算过程，近邻个数k不为1和k近邻中有多种数据出现次数相同的情况可能还需要更多的处理。
5. tensorflow结点保存的不是真正的数据的值，保存的是计算过程，可以定义计算的函数。然后将训练集和测试集传入tensorflow的结点得到计算结果
6. listdir命令可以获得路径下的所有文件名，open命令可以读入文件内容
7. sorted命令可以自动实现k近邻元素出现次数的排序