《机器学习实战》源码解析

机器学习源码解析 II

kNN 算法

jackycaf

2014/5/3 Saturday

目录

第一章 数据导入和预处理数据

在讲解算法之前，需要简单介绍一下数据导入和预处理函数。

数据导入函数

使用 UltraEdit 新建一个 file2matrix01.py 文件

代码区：file2matrix01.py

```
# -*- coding: utf-8 -*-
# Filename : file2matrix01.py

from numpy import *
import operator
import numpy as np
import matplotlib
import matplotlib.pyplot as plot

filename = 'datingTestSet2.txt'                    # 导入的数据文件路径

fr = file(filename)                                # 打开文件对象

numberOfLines = len(fr.readlines())                # 获取数据集文件长度，值为 1000

print numberOfLines                                # 输出文件的行数:1000

returnMat  =  zeros((numberOfLines,3))             # 初始化返回的数据矩阵，为矩阵行数为数据集文件长度 1000，3 列
# print returnMat    # 1000*3 全零矩阵

classLabelVector = []                              # 初始化类别标签向量

fr.close()                                         # 关闭文件对象

# 以上的步骤得到数据文件的行数，用以初始化数据矩阵的行数

fr = open(filename)                                # 再次打开文件对象

index = 0                                          # 初始化行索引值

for line in fr.readlines():                        # 按行读取数据文件
```

```
        line = line.strip()                                    # 删除左右两侧空格
        listFromLine = line.split('\t')                        # 源文件每行是以 TAB 空格来分隔字符，每个值为矩阵的一个元素
        returnMat[index,:] = listFromLine[0:3]                 # 把分隔好的矩阵前三个元素赋值给 returnMat 中
        # print int(listFromLine[-1])                          # 输出行向量的最后一个元素
        classLabelVector.append(int(listFromLine[-1]))         # 在类别向量中加入该类元素
        index += 1
    fr.close()                                                 # 关闭文件对象
    print returnMat                                            # 输出转换后的数据矩阵
    print classLabelVector                                     # 输出类别标签
```

输出：略

数据归一化

归一化是一种简化计算的方式，即将有量纲的表达式，经过变换，化为无量纲的表达式，成为纯量。 在多种计算中都经常用到这种方法。

定义

归一化是一种无量纲处理手段，使物理系统数值的绝对值变成某种相对值关系。简化计算，缩小量值的有效办法。例如，滤波器中各个频率值以截止频率作归一化后，频率都是截止频率的相对值，没有了量纲。阻抗以电源内阻作归一化后，各个阻抗都成了一种相对阻抗值，“欧姆”这个量纲也没有了。等各种运算都结束后，反归一化一切都复原了。信号处理工具箱中经常使用的是 nyquist 频率，它被定义为采样频率的一半，在滤波器的阶数选择和设计中的截止频率均使用 nyquist 频率进行归一化处理。例如对于一个采样频率为 1000hz 的系统，400hz 的归一化频率就为 400/500=0．8。归一化频率范围在[0,1]之间。如果将归一化频率转换为角频率，则将归一化频率乘以 2*pi；如果将归一化频率转换为 hz，则将归一化频率乘以采样频率的一半。

归一条件

在量子力学里，表达粒子的量子态的波函数必须满足归一条件，也就是说，在空间内，找到粒子的概率必须等于 1。这性质称为归一性。用数学公式表达，

其中，是粒子的位置，是波函数。

归一化导引

一般而言，波函数是一个复函数。可是，概率密度是一个实函数，空间内积分和为 1，称为概率密度函数。所以，在区域内，找到粒子的概率是 1。

既然粒子存在于空间，因此在空间内找到粒子概率是 1。所以，积分于整个空间将得到 1。

假若，从解析薛定谔方程而得到的波函数，其概率是有限的，但不等于，则可以将波函数乘以一个常数，使概率等于 1。或者，假若波函数内，已经有一个任意常数，可以设定这任意常数的值，使概率等于 1。

举例

比如，复数阻抗可以归一化写为：Z = R + jωL = R(1 + jωL/R)

注意复数部分变成了纯数了，没有任何量纲。

另外，微波之中也就是电路分析、信号系统、电磁波传输等，有很多运算都可以如此处理，既保证了运算的便捷，又能凸现出物理量的本质含义。

在统计学中，归一化的具体作用是归纳统一样本的统计分布性。归一化在 0-1 之间是统计的概率分布，归一化在-1--+1 之间是统计的坐标分布。

即该函数在(-∞,+∞)的积分为 1

例如概率中的密度函数就满足归一化条件

归一化函数举例：

线性函数转换如下

y=(x-MinValue)/(MaxValue-MinValue)

说明：x、y 分别为转换前、后的值，MaxValue、MinValue 分别为样本的最大值和最小值。

数函数转换如下

y=log10(x)

说明：以 10 为底的对数函数转换。

反正切函数转换如下

y=atan(x)*2/PI

数据的归一化函数：线性函数

# -*- coding: utf-8 -*- 	
# Filename : autoNorm02.py 	
 	
from numpy import *
import operator
import numpy as np
import matplotlib
import matplotlib.pyplot as plot
 	
# 数据集归一化 	
dataSet = array([[4,0.10],[3,0.15],[2,0.13],[1,0.2]])
	左侧的输出：
	dataSet: [[4. 0.1]

```
print "dataSet:",dataSet
print
# 返回数据集的最小值矩阵，按列区分，返回每一列的最小值
minVals = dataSet.min(0)
print "minVals:",minVals
print
# 返回数据集的最大值矩阵，按列区分，返回每一列的最大值
maxVals = dataSet.max(0)
print "maxVals:", maxVals
print
# 返回最大值矩阵-最小值矩阵的范围差
ranges = maxVals - minVals
print "ranges:", ranges
print
# shape(dataSet):返回一个向量: 矩阵的行与列的数量 4,2
# zeros(shape(dataSet)): 初始化 normDataSet 为与 dataSet 相同的全零矩阵
normDataSet = zeros(shape(dataSet))

# 初始化 normDataSet 为与 dataSet 相同的零矩阵
m = dataSet.shape[0]    # dataSet 的行数

# 以最小值填充矩阵 tile(minVals, (m,1))
print "tile(minVals, (m,1)):",tile(minVals, (m,1))
print



# 原始数据集(dataSet)-最小值构成的矩阵
normDataSet = dataSet - tile(minVals, (m,1))
print ("dataSet - tile(minVals, (m,1)):\n%s" %normDataSet)
print


# 线性归一化公式：(x-MinValue)/(MaxValue-MinValue)
# 矩阵各元素除以范围值的矩阵
normDataSet = normDataSet/tile(ranges, (m,1))    #两矩阵逐个元素相除
```

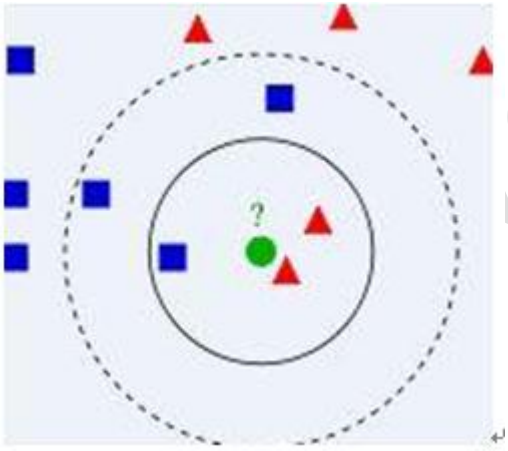
```
 [ 3.    0.15]
 [ 2.    0.13]
 [ 1.    0.2 ]]

左侧的输出：
minVals: [ 1.    0.1]

左侧的输出：
maxVals: [ 4.    0.2]

左侧的输出：
ranges: [ 3.    0.1]

左侧的输出：
tile(minVals, (m,1)): [[ 1.    0.1]
 [ 1.    0.1]
 [ 1.    0.1]
 [ 1.    0.1]]

左侧的输出：
dataSet - tile(minVals, (m,1)):
[[ 3.    0.  ]
 [ 2.    0.05]
 [ 1.    0.03]
 [ 0.    0.1 ]]

左侧的输出：
```

print ("normDataSet/tile(ranges, (m,1)):\n%s" %normDataSet) print # return normDataSet, ranges, minVals	normDataSet/tile(ranges, (m,1)): [[1. 0.] [0.66666667 0.5] [0.33333333 0.3] [0. 1.]]

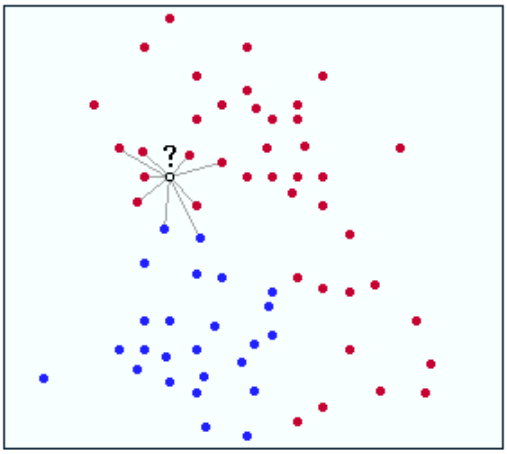
第二章 kNN 算法解析

kNN 算法

K 最近邻（k-Nearest Neighbor，KNN）分类算法可以说是最简单的机器学习算法了。它采用测量不同特征值之间的距离方法进行分类。它的思想很简单：如果一个样本在特征空间中的 k 个最相似（即特征空间中最邻近）的样本中的大多数属于某一个类别，则该样本也属于这个类别。



比如上面这个图，我们有两类数据，分别是蓝色方块和红色三角形，他们分布在一个上图的二维中间中。那么假如我们有一个绿色圆圈这个数据，需要判断这个数据是属于蓝色方块这一类，还是与红色三角形同类。怎么做呢？我们先把离这个绿色圆圈最近的几个点找到，因为我们觉得离绿色圆圈最近的才对它的类别有判断的帮助。那到底要用多少个来判断呢？这个个数就是 k 了。如果 k=3，就表示我们选择离绿色圆圈最近的 3 个点来判断，由于红色三角形所占比例为 2/3，所以我们认为绿色圆是和红色三角形同类。如果 k=5，由于蓝色四方形比例为 3/5，因此绿色圆被赋予蓝色四方形类。从这里可以看到，k 的值还是很重要的。



KNN 算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者

几个样本的类别来决定待分样本所属的类别。由于 KNN 方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，KNN 方法较其他方法更为适合。

该算法在分类时有个主要的不足是，当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的 K 个邻居中大容量类的样本占多数。因此可以采用权值的方法（和该样本距离小的邻居权值大）来改进。该方法的另一个不足之处是计算量较大，因为对每一个待分类的文本都要计算它到全体已知样本的距离，才能求得它的 K 个最近邻点。目前常用的解决方法是事先对已知样本点进行剪辑，事先去除对分类作用不大的样本。该算法比较适用于样本容量比较大的类域的自动分类，而那些样本容量较小的类域采用这种算法比较容易产生误分[参考机器学习十大算法]。

总的来说就是我们已经存在了一个带标签的数据库，然后输入没有标签的新数据后，将新数据的每个特征与样本集中数据对应的特征进行比较，然后算法提取样本集中特征最相似（最近邻）的分类标签。一般来说，只选择样本数据库中前 k 个最相似的数据。最后，选择 k 个最相似数据中出现次数最多的分类。其算法描述如下：

1）计算已知类别数据集中的点与当前点之间的距离；

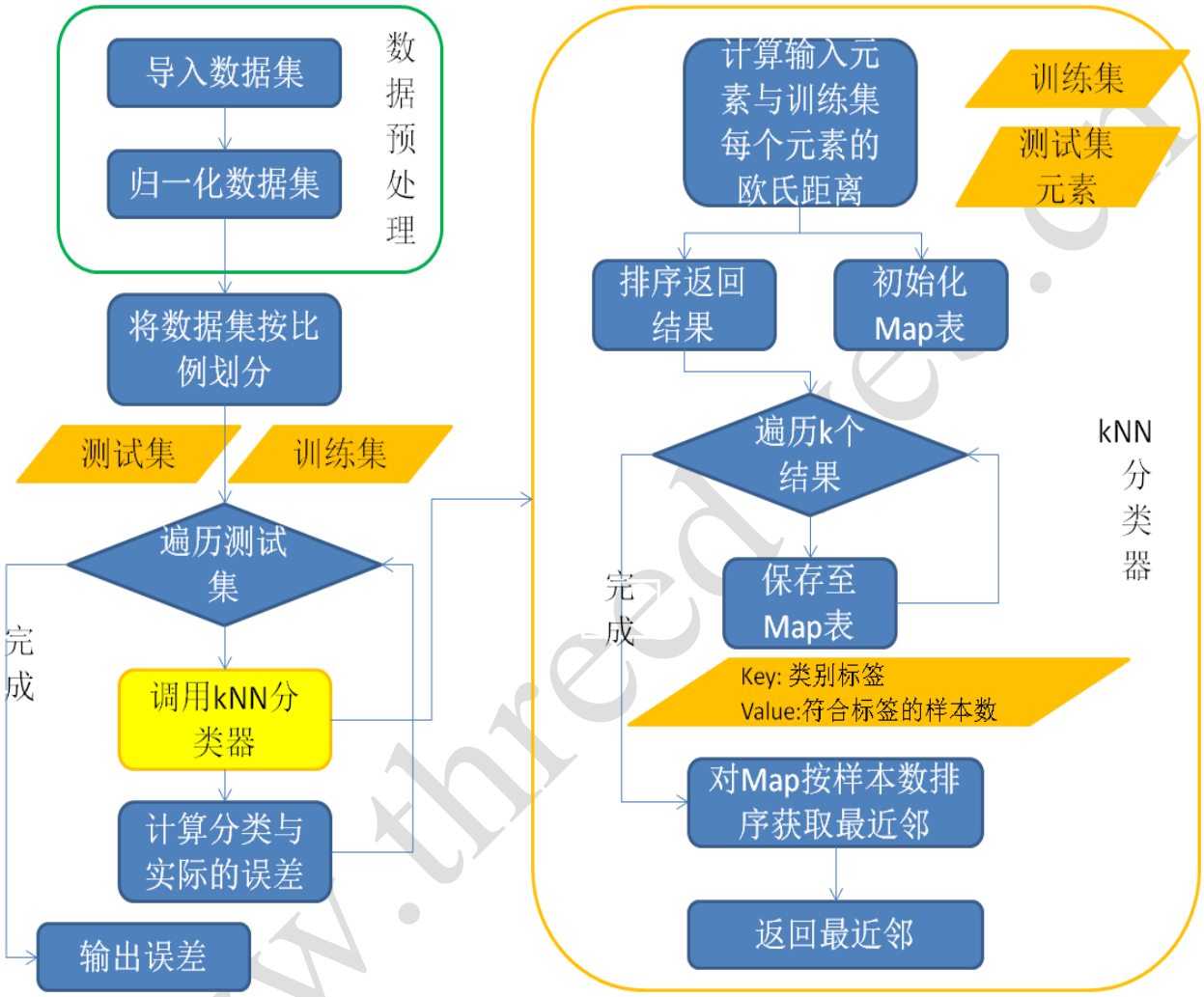
2）按照距离递增次序排序；

3）选取与当前点距离最小的 k 个点；

4）确定前 k 个点所在类别的出现频率；

5）返回前 k 个点出现频率最高的类别作为当前点的预测分类。

kNN 的算法逻辑



kNN 的 Python 实现

代码：ClassTest03.py

```
# -*- coding: utf-8 -*-
# Filename : ClassTest03.py

from numpy import *
import kNN2
import numpy as np
import operator
from os import listdir

hoRatio = 0.001       #测试集与训练集的比例—可以通过改变这个值观察算法的变化
# 加载数据文件
# datingDataMat: 数据集
# datingLabels: 数据分类标签
datingDataMat,datingLabels = kNN2.file2matrix('datingTestSet2.txt')

# 归一化数据集
normMat, ranges, minVals = kNN2.autoNorm(datingDataMat)
m = normMat.shape[0]              # 数据集行数 1000
numTestVecs = int(m*hoRatio)      # 测试集个数为整个数据集的前1000*0.001 个。int 转换为整数。
errorCount = 0.0                  # 误差率
# print normMat[numTestVecs:m,:]     # 训练集，从 numTestVecs-->m
# print datingLabels[numTestVecs:m]    # 训练集的分类标签，从 numTestVecs-->m

for i in range(numTestVecs):
        # kNN2.classify：kNN 分类器
        # normMat[i,:]：测试集
        # normMat[numTestVecs:m,:]：训练集，从 numTestVecs-->m
        #  datingLabels[numTestVecs:m] ： 分类标签集，从 numTestVecs-->m
        # 3 是 k 值
    classifierResult = kNN2.classify(normMat[i,:],normMat[numTestVecs:m,:],datingLabels[numTestVecs:m],3)
    # 按行返回每行的数据:normMat[i,:]
    print ("classifierResult vs datingLabels[i]:%s:%s" %(classifierResult,
```

左侧的输出：

```
classifierResult vs datingLabels[i]:3:3
```

```
datingLabels[i]))
    # print "the classifier came back with: %d, the real answer is: %d" %
(classifierResult, datingLabels[i])
    # 比较 knn 分类结果与实际结果的误差
    if (classifierResult != datingLabels[i]): errorCount += 1.0
print "the total error rate is: %f" % (errorCount/float(numTestVecs))
print errorCount
```

左侧的输出：

```
the total error rate is: 0.000000
0.0
```

kNN 的分类器

代码：kNN2.py

```
# -*- coding: utf-8 -*-
# Filename : kNN2.py

from numpy import *
import operator
from os import listdir

# kNN 分类器
# 测试集：inX
# 训练集：dataSet
# 类别标签：labels
# k:kNN 中的 k 个邻居数
def classify(inX, dataSet, labels, k):
    # 返回训练集的行数
    dataSetSize = dataSet.shape[0]

    # 计算测试集元素与每个训练集元素之间的距离：欧氏距离
    # 1.计算测试项与训练集各项的差
    diffMat = tile(inX, (dataSetSize,1)) - dataSet
    # 2.计算差的平方和
    sqDiffMat = diffMat**2
    # 3.按列求和
    sqDistances = sqDiffMat.sum(axis=1)
    # 4.生成均方差距离
    distances = sqDistances**0.5
    # 5.根据生成的欧氏距离大小排序,结果为索引号
    sortedDistIndicies = distances.argsort()

    classCount={}

    # 获取欧氏距离的前 k 项作为参考项
```

```
    for i in range(k):                  #    i = 0~(k-1)
        # 按序号顺序返回样本集对应的类别标签
        voteIlabel = labels[sortedDistIndicies[i]]
        print ("i=%s sortedDistIndicies = %s labels[%s]=%s" %
(i,sortedDistIndicies[i],sortedDistIndicies[i],voteIlabel))
        # 为字典 classCount 赋值,相同 key，其 value 加 1
        # key: voteIlabel   分类标签
        # value: 符合分类标签的累计样本数
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1

    print “classCount:”,classCount

    # 对分类字典 classCount 按累计样本数重新排序
    # sorted(data.iteritems(), key=operator.itemgetter(1), reverse=True)
    # 该句是按字典值排序的固定用法
    # classCount.iteritems()：字典迭代器函数
    # key：排序参数；operator.itemgetter(1)：多级排序
    sortedClassCount = sorted(classCount.iteritems(),
key=operator.itemgetter(1), reverse=True)
    print "sortedClassCount:",sortedClassCount
    # 返回序最高的一项
    return sortedClassCount[0][0]
```

左侧的输出：

```
i=0 sortedDistIndicies = 485 labels[485]=3
i=1 sortedDistIndicies = 914 labels[914]=3
i=2 sortedDistIndicies = 432 labels[432]=3
```

左侧的输出：

```
classCount: {3: 3}
```

含义是标签为 3 的样本有 3 个

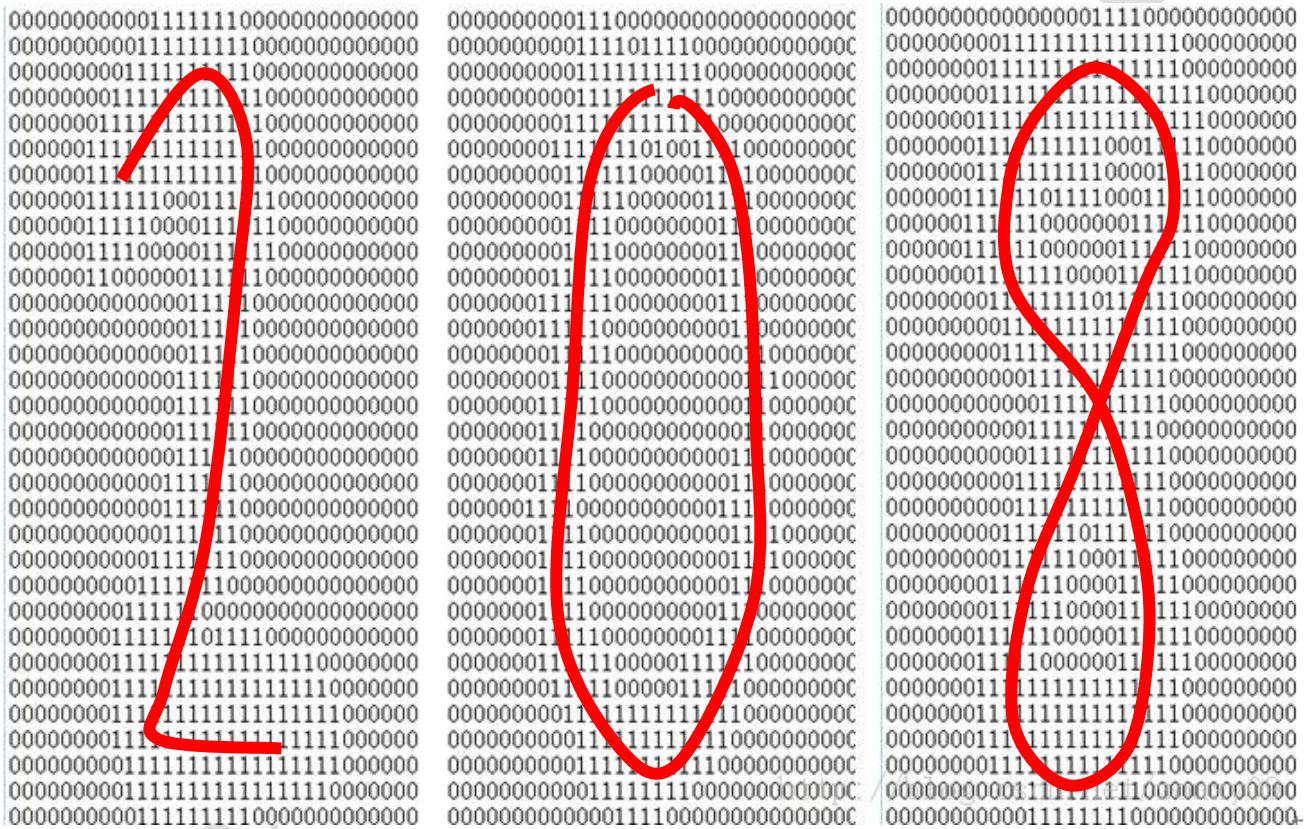
左侧的输出：

```
sortedClassCount: [(3, 3)]
```

含义是对 classCount 重排序使最多样本的那个分类标签排在最前

第三章 kNN 算法进阶

这里我们用 kNN 来分类一个大点的数据库，包括数据维度比较大和样本数比较多的数据库。这里我们用到一个手写数字的数据库，可以到附件中下载。这个数据库包括数字 0-9 的手写体。每个数字大约有 200 个样本。每个样本保持在一个 txt 文件中。手写体图像本身的大小是 32x32 的二值图，转换到 txt 文件保存后，内容也是 32x32 个 0 或者 1 的数字，如下：



数据库解压后有两个目录：目录 trainingDigits 存放的是大约 2000 个训练数据，testDigits 存放大约 900 个测试数据。

手写识别分类

手写识别主程序

代码：HandWriting04.py

```
# -*- coding: utf-8 -*-
```

```
# Filename : HandWriting04.py

from numpy import *
import kNN2
import numpy as np
import operator
from os import listdir

# 初始化分类标签
hwLabels = []
#加载训练集
#listdir('trainingDigits'): 列出 trainingDigits 里面的所有文件和目录，但不包括子目录中的内容。
trainingFileList = listdir('trainingDigits')
# 返回目录中文件的数量
m = len(trainingFileList)
# 初始化训练集矩阵：每个数字为一个向量： 1*1024
trainingMat = zeros((m,1024))

# 循环生成训练集矩阵
for i in range(m):
        # 对训练集的文件名进行处理
    fileNameStr = trainingFileList[i]
    fileStr = fileNameStr.split('.')[0]        # 去除 .txt
    # 获得下划线分隔文件名的前半部分，并转换为整型
    # 该文件名就是分类标签
    classNumStr = int(fileStr.split('_')[0])
    # 附加为生成分类标签集
    hwLabels.append(classNumStr)
    # 将训练集转换为向量形式,长度为 1024
    trainingMat[i,:] = kNN2.img2vector('trainingDigits/%s' % fileNameStr)

# 列出 testDigits 里面的所有文件和目录，作为测试集。
testFileList = listdir('testDigits')
errorCount = 0.0
# 测试集文件的数
mTest = len(testFileList)
# 循环每个测试集
for i in range(mTest):
        # 对测试集的文件名进行处理
    fileNameStr = testFileList[i]
    fileStr = fileNameStr.split('.')[0]        #去除后缀.txt
    # 文件名生成分类标签--用于测试
    classNumStr = int(fileStr.split('_')[0])
```

```
    # 将测试集转换为 1024 向量形式
    vectorUnderTest = kNN2.img2vector('testDigits/%s' % fileNameStr)
    # 将测试集应用相应的训练集进行 kNN 分类
    # vectorUnderTest 测试集元素
    # trainingMat 训练集
    # hwLabels, 训练集的分类标签集
    # 3， k 近邻
    classifierResult = kNN2.classify(vectorUnderTest, trainingMat, hwLabels, 3)
    print "the classifier came back with: %d, the real answer is: %d" % (classifierResult, classNumStr)
    if (classifierResult != classNumStr): errorCount += 1.0
print "\nthe total number of errors is: %d" % errorCount
print "\nthe total error rate is: %f" % (errorCount/float(mTest))
```

输出：

```
略...

the total number of errors is: 11

the total error rate is: 0.011628
```

图片数据预处理

将 32*32 的图片数据转换为 1*1024 的向量

代码：kNN2.py

```
# 图片转换向量的函数
def img2vector(filename):
        # 初始化长度为 1024 的全零向量
    returnVect = zeros((1,1024))
    # 打开文件
    fr = open(filename)
    # 每 32 个元素为 1 行，循环 32 次将矩阵累加为一个向量 向量长度为 32*32=1024
    for i in range(32):
        lineStr = fr.readline()
        for j in range(32):
            returnVect[0,32*i+j] = int(lineStr[j])
            # print returnVect
    # 返回处理后的向量
    return returnVect
```