KNN 算法试验报告

刘群*

地球系统科学研究中心

2015 年 4 月 11 日

这次作业是让我们编写利用 KNN 算法实现基于 Iris 数据集分类的程序, 与课上我们练习的情形类似，但这是一个多维的情形. 这个实验报告的思路是先对 KNN 算法做一个简单的介绍，然后对程序的整体思路做一个介绍，再对每个函数做详细的介绍，最后进行总结。

1 KNN 算法简介

KNN 算法是通过计算某个样本点与已经分好类的数据中所有点的距离，并对这些距离进行排序，找出其中距离最近的 k 个点，然后根据这 k 个点中所有类别个数的多少判断该样本点所属的类别。

2 程序的整体思路

首先，我将已分类的数据读入，由于数据中既有数据也有类别字符串，因此我先将这两者分离。这主要是由读入文件的函数实现的。然后就是编写 KNN 的分类函数，主要是通过计算新的数据与已有数据中所有样本点的距离来确定。通过计算距离，我们可以确定出所有点中与这个代分类点最近的 k 个点，然后可以统计这 k 个点中每个类别出现的次数, 次数最多的类别即为代分类样本点所属的类别。接下来就是对错误率的统计，主要实现思路为将测试用例中的数据依次进行 KNN 分类，将分类所得的结果与它实际所属的类别进行对比，如果出现错误则进行统计。最后返回错误率. 主要函数如下：

1. loadDataSet 主要是读入数据，对数据进行分割形成规则数据集和类别标签
2. knnClassify 主要进行 KNN 分类
3. KNN 这个函数主要是对测试文件中的数据进行分类，统计错误率

3 主要函数的介绍

3.1 loadDataSet

主要是读入数据，对数据进行分割形成规则数据集和类别标签, 现将函数详细介绍如下。open 是打开要读取的文件，返回文件的句柄。然后用 read 可以读入整个文件，但是返回的结果是一个字符串，包含了文件中所有的内容。然后由于每一行后面都有一个 \n, 因此可以通过字符串的 split 命令将整个文件字符串分解为每一行为一个元素的 list, 即程序中的 wholeData(未防止后面出现空行，所以加上 if 非空的判断)。注意，其中的每个元素此时仍为字符串。然后对这个列表中的每个元素再做一个 split，注意到此时元素中以逗号为分隔符。这样就可以把最后的类别名读入 labels 列表，把前面的四个数据读入 dataSet 矩阵。此时要注意的是要想转换成矩阵的形式，需要调用 numpy 中的 array 函数, 否则的话只是一个单纯的 list。

```
 import numpy as np
def loadDataSet(filename):
    wholeData = open(filename).read().split('\n')
    # 将最后的类别名读入labels列表
    labels = [ line.split(',')[4] for line in wholeData if line != '']
    # 将前面四列数据读入dataSet 数组(矩阵形式)
    dataSet = np.array([[float(x) for x in line.split(',')[0:4]] for line in wholeData if line
     != ''])
    return dataSet, labels
```

*电子邮件: liu-q14@mails.tsinghua.edu.cn，学号: 2014211591

3.2 knnClassify

这个程序主要是进行 KNN 分类，首先读入已分好类的数据，通过 loadDataSet 函数得到这些数据及其相应的标签。在进行距离的计算时，可以充分利用 python 中提供的矩阵运算的特点，利用 numpy 中的 tile 函数将新读入的数据扩展成一个与已排好数据相同大小的矩阵，这样可以直接进行计算。最后利用 numpy 中的 argsort 函数得到从小到大序列的下标 (在原数组中的位置)。然后利用字典，以 label 为 key 值，将出现的次数记为其 value 值，然后统计前 k 个距离最近的点中不同类别的个数。最后找出个数最大的类别并将其名称返回。

```
def knnClassify(newdata, k):
    # 读入训练数据 read train data
    dataSet, labels = loadDataSet('train_iris.data')
    # 训练数据的行数
    nrows = dataSet.shape[0]
    # 计算新的数据点到所有已分好类的数据点的距离，并对其排序，得到排序序列的index
    # 注意利用矩阵来运算
    sortedIndex= np.argsort(np.sqrt(np.sum((np.tile(newdata, (nrows, 1))-dataSet)**2, axis =
    1)))

    # Error check 错误检测，如果k比行数还大或者k小于1，输出提示信息，并抛出错误
    if k > nrows or k <= 0:
        print "k should be no larger than " + str(nrows) + "."
    assert k <= nrows and k >= 1

    # 利用字典，统计每种类别在前k个中的个数
    classCount = {}
    for i in range(k):
        voteLabel = labels[sortedIndex[i]]
        # 如果voteLabel所对应的value不存在，返回0(get 函数)
        # 相应的label加1
        classCount[voteLabel] = classCount.get(voteLabel, 0) + 1

    # 统计个数最多的Label
    maxCount = 0
    for key, value in classCount.items():
        if value > maxCount:
            maxCount = value
            maxIndex = key

    return maxIndex
```

3.3 KNN

这个函数主要是对测试文件进行测试，得到 KNN 分类的错误率。其主要过程如下，通过 loadDataSet 函数读入要测试的文件，然后通过对代测试文件中的每一行数据调用 knnClassify 函数进行分类，将分类结果与其本身的类别做比较，如果不同，则分类错误，利用 float 将 bool 型变量转为浮点型的数据。由于将这些 0 或者 1 放到了一个 list 里，然后利用 sum 函数求和即可得到错误的行数。最后即可算出错误率并返回。

```
def KNN(test_file_name, k):
    # 读入测试文件，得到数据和类别标签
    testDataSet, testLabels = loadDataSet(test_file_name)
    # 测试文件的行数
    totalRows = testDataSet.shape[0]
    # KNN分类错误的行数
    errorRows = sum([ float( knnClassify(testDataSet[i], k) != testLabels[i] ) for i in range(
    totalRows) ])
    # 返回错误率
    return errorRows / float(totalRows)
```

4 总结

KNN 算法虽然比较简单，但是对于我来说，由于对 python 掌握的不是很熟练，尤其是对于 numpy 等模块里的函数更是不熟悉，因此在编写时感到比较吃力。在编写 python 程序时，我们可以充分利用 comprehension 结构，将程序写的简单易读。