Python徒手实现识别手写数字—图像识别算法(K最近邻)

写在前面

这一段的内容可以说是最难的一部分之一了，因为是识别图像，所以涉及到的算法会相比之前的来说比较困难，所以我尽量会讲得清楚一点。

而且因为在编写的过程中，把前面的一些逻辑也修改了一些，将其变得更完善了，所以一切以本篇的为准。当然，如果想要直接看代码，代码全部放在我的GitHub中，所以这篇文章主要负责讲解，如需代码请自行前往GitHub。

本次大纲

上一次写到了数据库的建立，我们能够实时的将更新的训练图片存入CSV文件中。所以这次继续往下走，该轮到识别图片的内容了。

首先我们需要从文件夹中提取出需要被识别的图片test.png，并且把它经过与训练图片相同的处理得到1x10000大小的向量。因为两者之间存在微小的差异，我也不是很想再往源代码之中增加逻辑了，所以我就直接把增加待识别图片的函数重新写一个命名为GetTestPicture，内容与GetTrainPicture类似，只不过少了"增加图片名称"这一个部分。

之后我们就可以开始进行正式图片识别内容了。

主要是计算待识别图片与所有训练图片的距离。当两个图片距离越近的时候，说明他们越相似，那么他们很有可能写的就是同一个数。所以利用这个原理，我们可以找出距离待识别图像最近的几个训练图片，并输出他们的数字分别是几。比如说我想输出前三个，前三个分别是3，3，9，那就说明这个待识别图片很有可能是3.

之后还可以对每一个位置加个权重，具体的就放在下一次再讲，本节内容已经够多了。

（第一篇文章之中我说过利用图片洞数检测。我尝试了一下，认为有些不妥，具体原因放在本文末。）

MAIN代码

所以直接把主要代码放上来，逻辑相对来说还是比较清晰的

```
import os
import OperatePicture as OP
import OperateDatabase as OD
import PictureAlgorithm as PA
import csv

##Essential vavriable 基础变量
#Standard size 标准大小
N = 100
#Gray threshold 灰度阈值
color = 200/255

n = 10

#读取原CSV文件
reader = list(csv.reader(open('Database.csv', encoding = 'utf-8')))
#清除读取后的第一个空行
del reader[0]
#读取num目录下的所有文件名
fileNames = os.listdir(r"./num/")
#对比fileNames与reader，得到新增的图片newFileNames
newFileNames = OD.NewFiles(fileNames, reader)
print('New pictures are: ', newFileNames)
#得到newFilesNames对应的矩阵
pic = OP.GetTrainPicture(newFileNames)
#将新增图片矩阵存入CSV中
OD.SaveToCSV(pic, newFileNames)
#将原数据库矩阵与新数据库矩阵合并
pic = OD.Combination(reader, pic)

#得到待识别图片
testFiles = os.listdir(r"./test/")
testPic = OP.GetTestPicture(testFiles)

#计算每一个待识别图片的可能分类
result = PA.CalculateResult(testPic, pic)
for item in result:
    for i in range(n):
        print('第'+str(i+1)+'个向量为'+str(item[i+n])+',距离为'+str(item[i]))
```

相比上一篇文章的内容，本篇文章里只增加了下面的的一段代码，即得到待识别图片名称、得到待识别图片向量、计算分类。

下面我们将着重讲解CalculateResult函数的内容，即识别图片的算法。

算法内容

算法大致讲解

我们在大纲之中已经简单介绍过了，所以我就直接把复制过来，并且再添加一些内容。

假设我们在二维平面上有两个点$A = (1, 1)$和$B = (5, 5)$，我现在再放一个点$C = (2, 2)$，那么请问，C点离哪一个更近？

学过初中数学的都会知道肯定是离A点更近。所以我们换一种说法，我们现在有两个类A和B，A类中包括了点$(1, 1)$,B类中包括了点$(5, 5)$，所以对于点$(2, 2)$，它**可能**属于哪一类？

因为这个点离A类的点更近一点，所以它可能属于A类。这就是结论。那么对于3维空间，A类是点$(1, 1, 1)$和B类是$(5, 5, 5)$，那么对于点$(2, 2, 2)$肯定也是属于A类。

可以看出，我们这里是将**两个点的距离**来作为判断属于哪一类的标准。那么对于我们将图片拉成的1xn维向量，他实际上投影到n维空间上就是一个点，所以我们将训练向量分成10类，分别代表十个数字，那么被识别数字靠近哪一个类，那说明它有可能属于这一个类。

那么我们这里可以假设对于被识别向量，列出距离他最近的前十个向量分别属于哪一类别，然后根据名次加上一个权重，并计算出一个值。该值代表了可能是属于哪一个类，因此这就是我们得出的最终的一个结果——被识别手写数字图片的值。

以上是第一篇文章中的内容，下面我着重讲一下数学方面的内容。

考虑到某些地方不能够输入数学公式（或不方便输入），我还是把这一段内容贴成图片出来。

假设存在两个 n 维向量 $\boldsymbol{A_{1\times n}}, \boldsymbol{B_{1\times n}}$，可以写成

$$\boldsymbol{A_{1\times n}} = \begin{bmatrix} A_1 & A_2 & A_3 & \dots & A_n \end{bmatrix}^{\mathrm{T}}$$

与

$$\boldsymbol{B_{1\times n}} = \begin{bmatrix} B_1 & B_2 & B_3 & \dots & B_n \end{bmatrix}^{\mathrm{T}}$$

所以在 n 维空间内，我们可以考虑两个向量的端点之间的距离 d 或者两个向量的夹角 θ。

两个向量之间的距离为

$$\begin{aligned} d &= \sqrt{(A_1-B_1)^2+(A_2-B_2)^2+\cdots+(A_n-B_n)^2} \\ &= ||\boldsymbol{A}-\boldsymbol{B}|| \end{aligned}$$

或者用夹角来表示为

$$\theta = \arccos\frac{|\boldsymbol{A}|^2+|\boldsymbol{B}|^2-|\boldsymbol{A}-\boldsymbol{B}|^2}{2|\boldsymbol{A}||\boldsymbol{B}|}$$

所以两个向量之间的比较就是通过 d 与 θ，当这两个值越小的时候说明越靠近（两个图片越像），越大的时候说明越远（两个图片越不像）。

之后直接挑出前几个离被识别图片最近的向量数字，基本上这几个数字就是被识别图片的数字了。但这样做未免有些简单，所以下一篇文章我会再深入一下，这张先讲计算距离的内容。

主代码

下面的代码中文件夹test用来存放待识别图片，并通过函数GetTestPicture来得到图片向量，之后和训练图片pic一起放进计算距离的函数CalculateResult中计算每一个待识别向量和其他所有图片向量的距离。

```
#得到待识别图片
testFiles = os.listdir(r"./test/")
testPic = OP.GetTestPicture(testFiles)

#计算每一个待识别图片的可能分类
result = PA.CalculateResult(testPic, pic)
for item in result:
    for i in range(n):
        print('第'+str(i+1)+'个向量为'+str(item[i+n])+',距离为'+str(item[i]))
```

函数CalculateResult在文件PictureAlgorithm.py中，这个文件里面包含了两个函数为CalculateDistance函数和CalculateResult函数，代表识别图片所用到的算法。

函数CalculateResult

这个函数的逻辑比较简单，也没什么好说的，主要的联系就是这个计算距离的CalculateDistance函数。

```
def CalculateResult(test, train):
    '''计算待识别图片test的可能分类'''
    #得到每个图片的前n相似图片
    testDis = CalculateDistance(test[:,0:N**2], train[:,0:N**2], train[:,N**2], n)
    #将testDis变成列表
    tt = testDis.tolist()
    #输出每一个待识别图片的所有前n个
    for i in tt:
        for j in i:
            print(j)
```

函数CalculateDistance

函数中我导入了四个参数：被识别向量test，训练向量train，与训练向量对应的每个向量对应代表的数字num，想要导出的前n个距离最近的向量。

```
def CalculateDistance(test, train, num, n):
    '''计算每个图片前n相似图片'''
    #前n个放距离，后n个放数字
    dis = np.zeros(2*n*len(test)).reshape(len(test), 2*n)
    for i, item in enumerate(test):
        #计算出每个训练图片与该待识别图片的距离
        itemDis = np.sqrt(np.sum((item-train)**2, axis=1))
        #对距离进行排序，找出前n个
        sortDis = np.sort(itemDis)
        dis[i, 0:n] = sortDis[0:n]
        for j in range(n):
            #找到前几个在原矩阵中的位置
            maxPoint = list(itemDis).index(sortDis[j])
            #找到num对应位置的数字，存入dis中
            dis[i, j+n] = num[maxPoint]
    return dis
```

首先建立一个行数为test内被识别向量数量，列数为2*n的矩阵，每一行前n个放距离，后n个放数字。之后针对每一个被识别向量进行循环。

首先直接计算每个训练图片与该识别图片的距离，直接可以用一行代码表示

```
itemDis = np.sqrt(np.sum((item-train)**2, axis=1))
```

这一行代码就是上文中的算法过程，我个人觉得还是比较复杂的，可以详细的拆开看一下，我这里不细讲了。下面的内容就是开始排序并且找到距离最近的前几个向量。

这里的逻辑是：先排序，找到距离最小的前n个，存入矩阵。找到前n个在原矩阵中的位置，并找到对应位置上num的数字，存入dis的后n个。

这样子就相当于完成了所有内容，返回dis即可。

实际测试

我自己动手写了一些数字，如图所示。所以实际上我们的数据库还是比较小的。

所以我又写了一个数字作为待识别图像，通过程序运行以后，我们的以直接输出前十个最相似的向量：

```
第1个向量为2.0,距离为33.62347223932534
第2个向量为2.0,距离为35.64182105224185
第3个向量为2.0,距离为38.69663119274146
第4个向量为2.0,距离为43.52904133387693
第5个向量为2.0,距离为43.69029199677604
第6个向量为1.0,距离为43.730883339256714
```

```
第7个向量为6.0,距离为44.94800943845918
第8个向量为2.0,距离为45.033283944455924
第9个向量为4.0,距离为45.43926712996951
第10个向量为7.0,距离为45.64893989116544
```

之后我又依次从1-9试了一遍，我自己手写的数字全部识别正确，可以看出准确率还是挺高的。所以做到这一步相当于已经完成度很高了。

所以我就试了一下从网上找的图片，发现几乎没有正确的了。说明我们的数据库还是太小，只认得我的字体。不过话说这样，也可以做一个字体识别的程序。

所以如果要提高准确率，那么扩大图库是必须的。这一次就到这里。

总结

所有源代码我都放在了我的GitHub中，如果有兴趣的话可以去看看。

到这里就相当于算法内容写完了，比较简单，只用了一个类似于K最近邻的算法。

下一篇文章将会讲一个给前n个排名加权的想法，这样来提高准确度。

所以这一次就先到这里为止，谢谢。

如果喜欢的话，麻烦点一个喜欢和关注一下噢，谢谢~