说明文档

def get\_threshold(image): # 获取图片中像素点数量最多的像素

def get\_bin\_table(threshold): # 按照阈值进行二值化处理，threshold: 像素阈值

def cut\_noise(image): # 去掉二值化处理后的图片中的噪声点

def picture\_processing(img\_path): # 传入参数为图片路径，返回结果为处理后的图片

def get\_crop\_imgs(img): # 图片字符切割

def mkdir(path): # 创建文件夹，以便里面存放图像的数字文件

def num\_name(ndir,nei): # 统计文件夹里面关于各个字符的文件个数，以便数字文件排序

def digital\_file(dirct,file): # 在指定的文件夹中生成各个字符的数字文件（训练集）

def digital\_file2(dirct,file): # 在指定的文件夹中生成各个字符的数字文件（测试集）

def identify(directory,dirct): # 识别指定文件目录下的图片（训练集）

def identify2(directory,dirct): # 识别指定文件目录下的图片（测试集）

def loadData(dir): # 给出文件目录，读取数据，生成训练集

def predict\_verification(files\_dict, mktestdata, label\_dict): # 对验证集进行验证

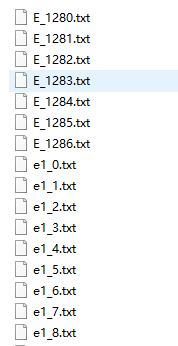
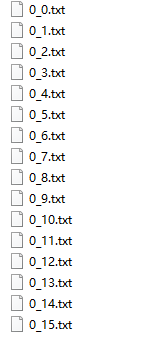
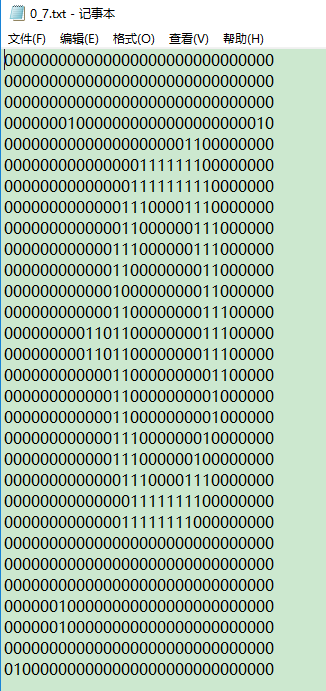
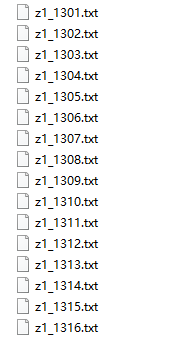
def predict\_test(files\_dict, mktestdata, label\_dict): # 对测试集进行预测

def csv\_save(answer) # 将测试集的结果保存到csv文件里

**代码思想：**

把彩色验证码图片转化成灰度图，再经过二值化以及去噪等图片预处理，然后分割验证码，验证码包括26个大小写字母和10个数字共62个类别，这就转化成了62分类的问题。关键步骤是把图像文件转换成数字文件，二值化后的图像像素只有1和0, 0表示黑色，1表示白色，具体实现是把0和1互相转化，写进txt文件，格式如下图所示。一张验证码对应5个txt文件，训练集一共19994张验证码，一共生成99970个txt文件。验证码分割后单个验证码的大小是30\*30，即生成的txt文件有30行30列，相当于900个属性，62个类别。再用分类方法进行分类，最后把5个单个字符的分类结果合起来，这就组成了一张验证码的预测结果。

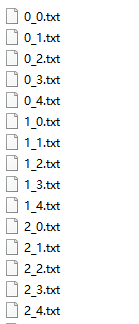
txt文件命名格式：①0\_61：下划线前的数字0表示文件存的字符是0，下划线后面的数字61表示这是字符0的第62个txt文件（字符0的第一个文件为0\_0.txt）。②文件名在被程序读入的时候不分大小写，所以我在字母后面加个1表示这是字母对应的大/小写形式，例如Z1表示z, a1表示A。

**代码流程：**

在if \_\_name\_\_ == '\_\_main\_\_'中，定义训练集图片的数字文件的存放目录mktraindata = "E:\\dasanshang\\机器学习框架\\traindata19994\\"，通过mkdir(mktraindata)函数创建出来这个目录，训练集图片存放的目录为“E:\\dasanshang\\机器学习框架\\train”，通过identify(traindir, mktraindata)函数让训练集验证码都生成对应的数字文件。文件格式如上所示。

通过loadData('traindata19994') 得到训练集数据trainDigits，训练集标签trainLabels，0-61与大小写字母和10个数字的映射关系label\_dict。选用KNN模型进行分类，通过knn.fit(trainDigits,trainLabels)对训练集进行训练。

定义测试集图片的数字文件的存放目录mktestdata = "E:\\dasanshang\\机器学习框架\\ testdata20000\\"，通过mkdir(mktestdata)函数创建出来这个目录，训练集图片存放的目录为“E:\\dasanshang\\机器学习框架\\ test”，通过identify2(testdir,mktestdata) 函数让测试集验证码都生成对应的数字文件，并获取到图片名字与生成的对应数字文件名字的映射关系files\_dict。这次的数字文件命名格式与上文训练集的命名格式不同，因为测试集图片的名字是数字。例如：测试集第一张图片的名字为0.jpg, 则它生成的对应5个数字文件就是0\_0.txt, 0\_1.txt, 0\_2.txt, 0\_3.txt, 0\_4.txt。下划线前面的数字表示这是测试集第几张图片的对应数字文件，下划线后面的数字表示这是这张图片的第几个字符。格式如下：

通过predict\_test(files\_dict,mktestdata,label\_dict)对测试集进行预测，得到结果sorted\_files\_dict。由于预测时结果不是按照0 1 2 3 ……. 10 这种顺序产生的，所以最后要对预测结果再排一下序，最终得出正确的结果顺序，将其用csv\_save(sorted\_result)函数存进csv文件。

**核心代码：**

1. 对图像进行预处理的过程： def get\_threshold(image)---获取图片中像素点数量最多的像素，def get\_bin\_table(threshold)---按照阈值进行二值化处理，def cut\_noise(image)--- 去掉二值化处理后的图片中的噪声点
2. 图像文件生成数字文件的过程：def digital\_file(dirct,file)--- 在指定的文件夹中生成各个字符的数字文件（训练集）

#===============获取图片中像素点数量最多的像素==================#

def get\_threshold(image):

pixel\_dict = defaultdict(int)

# 像素及该像素出现次数的字典

rows, cols = image.size

for i in range(rows):

for j in range(cols):

pixel = image.getpixel((i, j))

pixel\_dict[pixel] += 1

count\_max = max(pixel\_dict.values()) # 获取像素出现出多的次数

pixel\_dict\_reverse = {v:k for k,v in pixel\_dict.items()}

threshold = pixel\_dict\_reverse[count\_max] # 获取出现次数最多的像素点

return threshold

#========按照阈值进行二值化处理，threshold: 像素阈=================#

def get\_bin\_table(threshold):

# 获取灰度转二值的映射table

table = []

for i in range(256):

rate = 0.1 # 在threshold的适当范围内进行处理

if threshold\*(1-rate)<= i <= threshold\*(1+rate):

table.append(1)

else:

table.append(0)

return table

#===============去掉二值化处理后的图片中的噪声点=================#

def cut\_noise(image):

rows, cols = image.size # 图片的宽度和高度

change\_pos = [] # 记录噪声点位置

# 遍历图片中的每个点，除掉边缘

for i in range(1, rows-1):

for j in range(1, cols-1):

# pixel\_set用来记录该点附近的黑色像素的数量

pixel\_set = []

# 取该点的邻域为以该点为中心的九宫格

for m in range(i-1, i+2):

for n in range(j-1, j+2):

if image.getpixel((m, n)) != 1: # 1为白色,0位黑色

pixel\_set.append(image.getpixel((m, n)))

# 如果该位置的九宫内的黑色数量小于等于4，则判断为噪声

if len(pixel\_set) <= 4:

change\_pos.append((i,j))

# 对相应位置进行像素修改，将噪声处的像素置为1（白色）

for pos in change\_pos:

image.putpixel(pos, 1)

return image # 返回修改后的图片

#=========在指定的文件夹中生成各个字符的数字文件（训练集）=============#

def digital\_file(dirct,file):

#对传入的图片进行预处理及切割

out = picture\_processing(file)

child\_img\_list = get\_crop\_imgs(out)

#文件名字

file\_name = file.split('.')[0]

#图片名字

picture\_name = file\_name[-5:]

#创建存放图片名字与生成的5个对应数字文件名字的映射关系的字典

files\_name\_dict = defaultdict(int)

name\_list = []

for i in range(5):

#进入目录

os.chdir(dirct)

#单个字符的名字

name = picture\_name[i]

num,transfername = num\_name(dirct,name)

#单个字符的数字文件的名字

digits\_file\_name = transfername+'\_'+str(num)+'.txt'

name\_list.append(digits\_file\_name)

#生成单个字符的数字文件

fileObject = open(digits\_file\_name, 'w+')

#填充数字文件，把图像像素转换为数字

for m in range(child\_img\_list[i].size[1]):

for n in range(child\_img\_list[i].size[0]):

pos=(n,m)

col=child\_img\_list[i].getpixel(pos)

if col==0:

col=1

else:

col=0

fileObject.write(str(col))

fileObject.write('\n')

fileObject.close()

#将映射关系存入字典

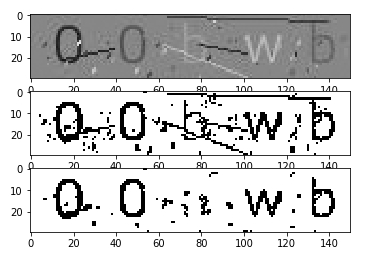
files\_name\_dict[picture\_name] = name\_list

return files\_name\_dict

**问题解决：**

一、图片预处理问题

要想获得精确的数字文件以便之后让分类的效果更好，前面图像的预处理工作得做好，即二值化和去噪。普通的二值化是直接设个阈值，大于这个阈值的设为1，小于的设为0，这样的效果并不好。阈值的设定是个问题，我选择算出彩色验证码中像素出现最多的像素，这个像素便是背景色的像素，把它当做阈值threshold，并在阈值的适当范围内进行处理，属于0.9\*threshold ~ 1.1\*threshold范围的像素值都置为1，其他的像素值置为0，用这样处理后的图片再进行去噪效果稍微好点。处理结果如下图所示。



二、模型选择问题

从训练集中抽出1333张图片作为训练集，抽出667张图片作为验证集，对训练集进行训练，然后预测验证集，得出正确率。用这一波比较小的数据来进行模型选择。

模型选择我首先选用了softmax，最终的结果为验证集667张图片只对了4张。后来我将19994张图片全用做了训练，但是因为数据太多，梯度下降过程中有许多数据都变成了负数，由于太小都变成了nan。后来我在softmax函数中进行了归一化，这样虽然最终数据都没有变成nan, 但最终效果还是不理想。后来发现是迭代次数和学习率没有选好的问题，由于每次测试都花费几个小时的时间，迭代次数和学习率只能一个个试，所以我最终放弃了这种方法。

后来我尝试了决策树、KNN、神经网络等模型。通过比较，得出KNN模型训练结果相对较好，再通过调最大近邻数n\_neighbors的参数，比较正确率，最终 n\_neighbors参数取值为1。由这个模型得到的正确率最终结果为0.05547，即667张验证码对了37张。后来我用了全部的训练集进行训练，对测试集进行预测，上交结果后得到的结果为0.20780，大约是20000张验证码对了4156张。

