SVM 算法实现数字识别建模

一、问题描述

以课堂上讲的手写数字数据集 MNIST 为例，利用 SVM 去实现数字识别建模。

要求

1、在给定的数据集上添加至少 10 个自己用手机拍的 0-9 数字，补充到原数据集中，并显示你添加的数据；

2、提交 python 的 project 打包

3、提交实验报告；

二、数据集说明

1.MNIST 数据集

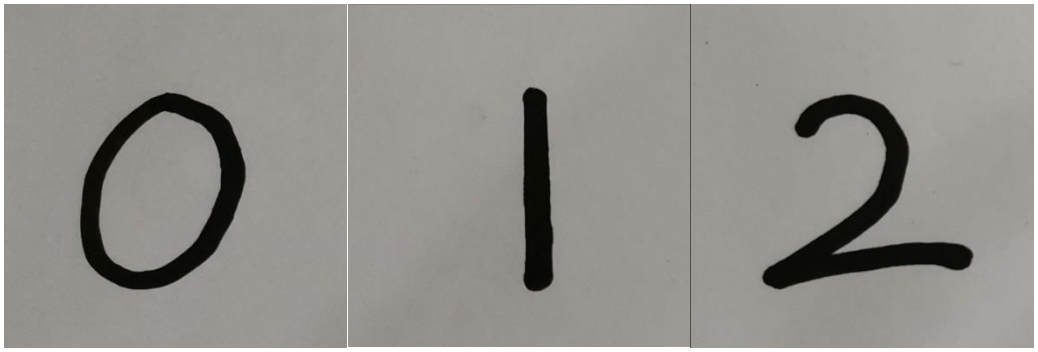
MNIST 数据集是机器学习领域中非常经典的一个数据集，由 60000 个训练样本和 10000 个测试样本组成，每个样本都是一张 28 * 28 像素的灰度手写数字图片。

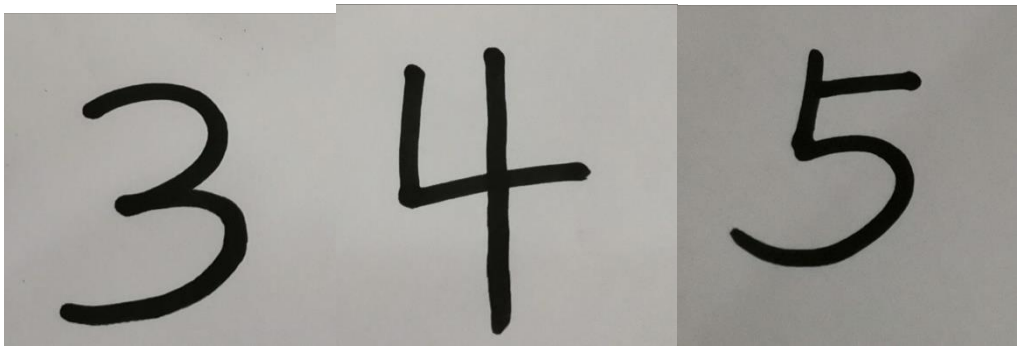
它来自美国国家标准与技术研究所，训练集(training set)由来自 250 个不同人手写的数字构成，其中 50%是高中学生，50%来自人口普查局的工作人员。测试集(test set)也是同样比例的手写数字数据。

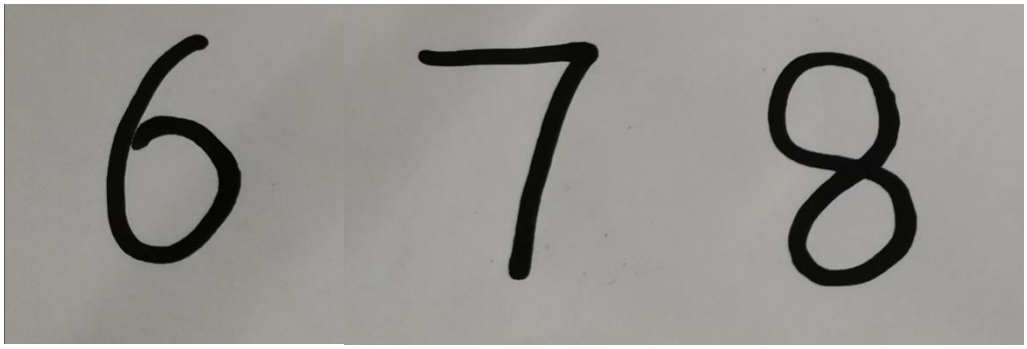
使用 TensorFlow 时，可以使用 tf.keras.datasets.mnist.load_data()来读取数据及标签,使用这种方式时,可以不用事先下载好数据集,它会自动下载并存放到你指定的位置。

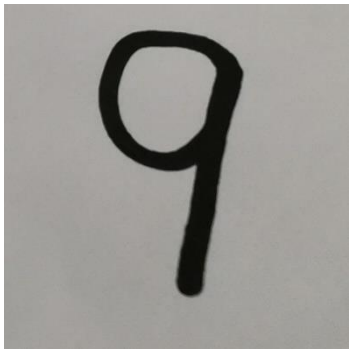
2.自己的数据

通过手机拍摄，得到 0~9 共十张图片如下：









这十张图片作为自己的测试集，用于测试，以检验数字识别建模的成果。

三、算法代码

1.MNIST 数据预处理

在 MNIST 数据预处理中，有以下几步：

1）数据格式转换：将原本的 60000*28*28 的矩阵转换为 60000*784 的矩阵，并且将数据类型转为 float32。

2）规范化：将数据都除 255，使得数据变为 0-1 之间的浮点数，可加速运算。

代码：

```
'''步骤二：数据预处理'''
# 格式转换
x_train =x_train.reshape(60000, 784).astype('float32')
x_test = x_test.reshape(10000, 784).astype('float32')

# 规范化，将像素值缩至0-1之间
x_train /= 255
x_test /= 255
```

2.模型搭建

首先使用 sklearn.model_selection.GridSearchCV 进行自动化调参，因为使用自动化调参速度会很慢，所以采用切片，对于前 1000 个数据进行训练得到最优的参数，得到的最优参数为 C=10.0， kernel='rbf'， gamma=0.01。然后使用将参数填入 svm.SVC 分类器，进行训练。

代码：

```
'''步骤三：模型搭建'''
# # 最优超参数组合列表
# params = [
#     {'kernel': ['linear'], 'C': [1, 10, 100, 100]},
#     {'kernel': ['poly'], 'C': [1], 'degree': [2, 3]},
#     {'kernel': ['rbf'], 'C': [1, 10, 100, 100], 'gamma':[1, 0.1, 0.01, 0.001]}
# ]

# # 自动化调参
# model = ms.GridSearchCV(svm.SVC(probability=True), params, refit=True, return_train_score=True,
cv=5)

model = svm.SVC(C=10.0, kernel='rbf', gamma=0.01)
```

3.模型训练

模型在训练中使用 Mnist 数据集中的所有训练数据。

代码：

```
'''步骤四：训练模型'''
h = model.fit(x_train, y_train)
```

4.自己数据预处理

在处理自己手机拍摄的图片时，有以下算法：

1）图像居中算法：通过获取手写字最左、最右、最上、最下的方位，得到中心点，然后将图像改为正方形，并保留四边各约 20%的背景色边缘，实现手写字居中，且大小适中。

2）先膨胀后腐蚀：消除手写字内部可能出现的背景色小泡。

3）转灰度图并进行高斯滤波：消除可能存在的噪点并使图像平滑。

4）二值化并反色：将图像二值化，并将手写的白底黑字转为黑底白字。

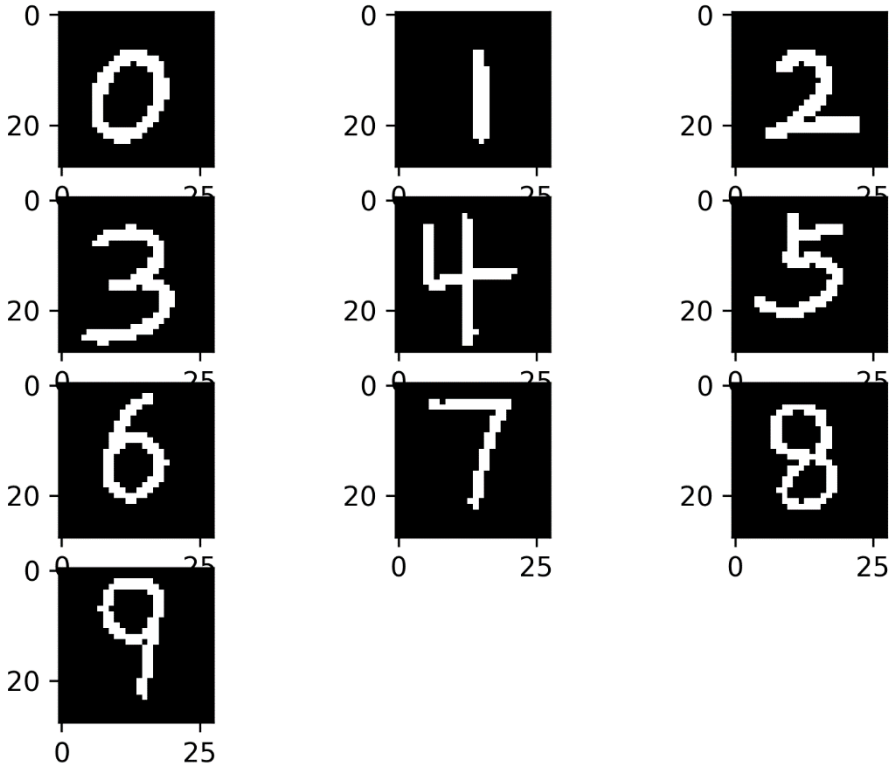
5）骨架提取和膨胀：使手写字粗细相对均匀且适中。

代码：

```
'''获取边界'''
def getbound(img):
    img_binary=img.convert('1')
    img_array = np.array(img_binary)
    shape = img_array.shape
    sumx = shape[0] - np.sum(img_array, axis=0)/255
    sumy = shape[1] - np.sum(img_array, axis=1)/255
    l, r, h, b = 0, 0, 0, 0
    for i in np.arange(shape[1]):
        if sumx[i] >= 3:
            l = i
            break
    for i in range(shape[1]-1, -1, -1):
        if sumx[i] >= 3:
            r = i
            break
    for i in np.arange(shape[0]):
        if sumy[i] >= 3:
            h = i
            break
    for i in range(shape[0]-1, -1, -1):
        if sumy[i] >= 3:
            b = i
            break
    return l, r, h, b
```

```python
'''数据预处理'''
def pretreat(filename):
    img = (Image.open(filename).convert('L'))
    l, r, h, b = getbound(img)
    shape = np.array(img).shape
    dx, dy = (r - l)//2, (b - h)//2

    # 获取中心点
    centerx, centery = l+dx, h+dy

    # 改为正方形
    dx = max(dx, dy)
    dy = dx

    # 得到轮廓
    l, r, h, b = max(0, centery-dy*1.2), min(shape[0], centery+dy*1.2), max(0, centerx-dx*1.2),
min(shape[1], centerx+dx*1.2)

    # 切片
    img_array = np.array(img)
    the_img = img_array[l:r, h:b]

    # 先膨胀后腐蚀，消除手写字内部可能出现的小洞
    the_img=morphology.closing(the_img, selem=None, out=None)

    # 转灰度
    the_img=Image.fromarray(cv2.cvtColor(the_img, cv2.COLOR_BGR2RGB))
    img_gray=cv2.cvtColor(np.array(the_img),cv2.COLOR_RGB2GRAY)

    # 高斯滤波
    img_gauss = cv2.GaussianBlur(img_gray,(5,5),0)

    # 二值化，反色
    ret,img_binary=cv2.threshold(img_gauss, 0, 1, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

    # 骨架提取
    img_binary=morphology.skeletonize(img_binary)

    # 膨胀
    img_binary=morphology.binary_dilation(img_binary, morphology.disk(29))

    # 调整大小为(28, 28)
    img_binary=Image.fromarray(img_binary)
    img_binary=img_binary.resize((28,28),Image.ANTIALIAS)

    # 为1的点转为255
    img_array=np.array(img_binary).astype(np.uint8)
    img_array[img_array==1]=255

    # 数据格式规范
    img_array.reshape((-1,28,28,1))
    img_array=img_array[:,:,np.newaxis]

    return img_array
```

预处理结果：



四、实验分析

1.结果分析

实验结果如下：

对于 MNIST 测试集：测试集准确率:0.9833。

混淆矩阵如下:

```
训练集的样本数：60000，测试集的样本数：10000
输入图像的大小：28*28
训练集的图像类别分布： Counter({1: 6742, 7: 6265, 3: 6131, 2: 5958, 9: 5949, 0: 5923, 6: 5918, 8: 5851, 4: 5842, 5: 5421})
Mnist手写数据集混淆矩阵如下：
[[ 973    0    2    0    0    2    0    1    2    0]
 [   0 1130    1    1    0    1    0    1    1    0]
 [   5    1 1014    0    1    0    1    6    4    0]
 [   0    0    2  994    0    2    0    4    5    3]
 [   0    0    4    0  965    0    2    0    0   11]
 [   3    0    0    8    1  872    3    0    3    2]
 [   5    2    0    0    2    3  945    0    1    0]
 [   0    5    9    2    1    0    0 1005    0    6]
 [   3    0    2    3    3    3    1    2  953    4]
 [   2    2    0    6    9    3    0    5    0  982]]
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      1.00      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.98      0.98      0.98      1010
           4       0.98      0.98      0.98       982
           5       0.98      0.98      0.98       892
           6       0.99      0.99      0.99       958
           7       0.98      0.98      0.98      1028
           8       0.98      0.98      0.98       974
           9       0.97      0.97      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000

accuracy= 0.9833
```

对于自己的测试集：测试集准确率:0.8

混淆矩阵如下:

```
我的数据集混淆矩阵如下：
[[1 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0]
 [0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0]
 [0 0 0 1 0 0 0 0 0 0]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       0.50      1.00      0.67         1
           2       1.00      1.00      1.00         1
           3       0.50      1.00      0.67         1
           4       1.00      1.00      1.00         1
           5       1.00      1.00      1.00         1
           6       1.00      1.00      1.00         1
           8       1.00      1.00      1.00         1

   micro avg       0.80      1.00      0.89         8
   macro avg       0.88      1.00      0.92         8
weighted avg       0.88      1.00      0.92         8

accuracy= 0.8
```

MNIST 测试集有约 98%的正确率，自己的测试集由于测试样本较少（10 个样本），有 80%的正确率，可说明整体实验成功。

2.实验过程分析

在本次实验中，有两个难点，一是设计 SVM 模型，二是设计关于自己图像的预处理算法。

在模型设计时，需要考虑任务的难度、模型的搭建难度、以及训练所需花费的时间等；

由于本次我们使用的是经典的 MNIST 数据集，进行经典的手写字识别，所以这次的任务难度不大，然后使用 sklearn.model_selection.GridSearchCV 进行了自动化调参得到了较好的参数，基于以上综合考虑，我们最终选择了 svm.SVC(C=10.0, kernel='rbf', gamma=0.01)，事实也证明我们使用的模型在测试集上效果很好。

在对自己的图像进行预处理时，我们需要考虑原始图像的方方面面，包括手写字的位置是不是居中、大小是否合适、背景是否有大量的噪点、图像是否相对平滑、线条内部是否有大量的噪点、线条粗细是否均匀适中等。基于以上考虑，我们设计了较为复杂的图像预处理算法。在调试过程中，往往会出现预处理之后图像的效果很不好，此时我们通过在之前的每次操作后都输出一次图像，来观察到底是哪里出了问题，从而调整我们的处理。例如对于骨架提取后的膨胀，就多次调整了滤波器的样式和大小。

在调试过程中，我发现如果将所有的步骤写在一个文件里，会导致浪费大量时间训练模型。因为在实验后期，模型往往已经很成熟，主要需要修改的是自己图像的预处理。因此我将实验的不同部分划分到不同的文件中，并把训练好的模型参数保存下来，每次调整了图像预处理后只需要拿去测试即可。