数据挖掘作业3

姓名：胡宗晖 学号：3220220922

基于支持向量机的手写数字识别

In [1]:

```
# 导入所需的库
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
from PIL import Image
import time
from sklearn import svm
from sklearn import metrics
```

1.读取MNIST数据以及划分训练集与测试集

1.1将MNIST数据集另存为png格式的图片，并按标签分别存放于0-9文件夹中

由于数据集网站已经将数据划分为训练集和测试集，以下将沿用这种划分方式，其中训练集样本60000个，测试集样本10000个

In [2]:

```
# 读入MNIST数据集
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# 预处理数据集

# 将标签转换为字符串类型
y_train = y_train.astype(str)
y_test = y_test.astype(str)

# 创建一个文件夹，用于存储图片
os.mkdir('D:/study/code/data/MNIST_png')
os.mkdir('D:/study/code/data/MNIST_png/train')
os.mkdir('D:/study/code/data/MNIST_png/test')

# 创建10个文件夹，分别用0-9命名
for i in range(10):
    os.mkdir('D:/study/code/data/MNIST_png/train/'+str(i))
    os.mkdir('D:/study/code/data/MNIST_png/test/'+str(i))

# 遍历训练集，将每张图片以标签为文件名，保存为png格式 60000个
for i in range(len(x_train)):
    # 获取图片和标签
    image = x_train[i]
    label = y_train[i]

    # 创建文件名，格式为"label_index.png"
    filename = label + '_' + str(i) + '.png'

    # 使用matplotlib库，将图片保存到指定文件夹
    #plt.imsave(os.path.join('D:/study/code/data/MNIST_png/train', filename), image, cmap='gray')
    plt.imsave('D:/study/code/data/MNIST_png/train/'+str(label) + "/" + str(label) + "_" + str(i

# 遍历测试集，将每张图片以标签为文件名，保存为png格式 10000个
for i in range(len(x_test)):
    # 获取图片和标签
    image = x_train[i]
    label = y_train[i]

    # 创建文件名，格式为"label_index.png"
    filename = label + '_' + str(i) + '.png'

    # 使用matplotlib库，将图片保存到指定文件夹
    plt.imsave('D:/study/code/data/MNIST_png/test/'+str(label) + "/" + str(label) + "_" + str(i)
```

2.读取数据并对图像数据进行归一化处理

In [3]:

```
# 获取指定路径下的所有 .png 文件
def get_file_list(path):
    return [os.path.join(path, f) for f in os.listdir(path) if f.endswith(".png")]


# 解析出 .png 图件文件的名称
def get_img_name_str(imgPath):
    return imgPath.split(os.path.sep)[-1]
# 将 20px * 20px 的图像数据转换成 1*400 的 numpy 向量
# 参数：imgFile--图像名  如：0_1.png
def img2vector(imgFile):
    #print("in img2vector func--para:{}".format(imgFile))
    img = Image.open(imgFile).convert('L')
    img_arr = np.array(img, 'i') # 20px * 20px 灰度图像
    img_normalization = np.round(img_arr/255) # 对灰度值进行归一化
    img_arr2 = np.reshape(img_normalization, (1,-1)) # 1 * 400 矩阵
    return img_arr2

# 读取一个类别的所有数据并转换成矩阵
# 返回：某一类别的所有数据----[样本数量*(图像宽x图像高)] 矩阵
def read_and_convert(imgFileList):
    dataLabel = [] # 存放类标签
    dataNum = len(imgFileList)
    dataMat = np.zeros((dataNum, 784)) # dataNum * 400 的矩阵
    for i in range(dataNum):
        imgNameStr = imgFileList[i]
        imgName = get_img_name_str(imgNameStr)  # 得到 数字_实例编号.png
        #print("imgName: {}".format(imgName))
        classTag = imgName.split(".")[0].split("_")[0] # 得到 类标签(数字)
        #print("classTag: {}".format(classTag))
        dataLabel.append(classTag)
        dataMat[i,:] = img2vector(imgNameStr)
    return dataMat, dataLabel
```

2.1读取训练集数据

In [4]:

```
# 读取训练数据
def read_all_data():
    cName = ['1', '2', '3', '4', '5', '6', '7', '8', '9']
    train_data_path = "D:/study/code/data/MNIST_png/train/0"
    flist = get_file_list(train_data_path)
    dataMat, dataLabel = read_and_convert(flist)
    for c in cName:
        train_data_path_ = "D:/study/code/data/MNIST_png/train/" + c
        flist_ = get_file_list(train_data_path_)
        dataMat_, dataLabel_ = read_and_convert(flist_)
        dataMat = np.concatenate((dataMat, dataMat_), axis=0)
        dataLabel = np.concatenate((dataLabel, dataLabel_), axis=0)
    #print(dataMat.shape)
    #print(len(dataLabel))
    return dataMat, dataLabel
```

3.建立和训练SVM模型，核函数取rbf径向基核

In [5]:

```
# create model
def create_svm(dataMat, dataLabel, decision='ovr'):
    clf = svm.SVC(decision_function_shape=decision,kernel='rbf')
    clf.fit(dataMat, dataLabel)
    return clf


st = time.perf_counter()
dataMat, dataLabel = read_all_data()
clf = create_svm(dataMat, dataLabel, decision='ovr')
et = time.perf_counter()
print("Training spent {:.4f}s.".format((et-st)))
```

```
Training spent 181.9508s.
```

4.在测试集上进行预测，计算准确率

In [6]:

```
# 对10个数字进行分类测试
tbasePath = "D:/study/code/data/MNIST_png/test/"
tcName = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
tst = time.perf_counter()
allErrCount = 0
allErrorRate = 0.0
allScore = 0.0
allCount = 0
y_pred = []
y_true = []

for tcn  in tcName:
    testPath = "D:/study/code/data/MNIST_png/test/" + tcn
    tflist = get_file_list(testPath)
    tdataMat, tdataLabel = read_and_convert(tflist)
    print("class "+tcn+" has {}".format(len(tdataLabel))+" test cases")
    allCount += len(tdataLabel)
    pre_st = time.perf_counter()
    preResult = clf.predict(tdataMat)
    pre_et = time.perf_counter()
    errCount = len([x for x in preResult if x!=tcn])
    print("errorCount: {}.".format(errCount))
    print("******************************")
    allErrCount += errCount
    score_st = time.perf_counter()
    score = clf.score(tdataMat, tdataLabel)
    score_et = time.perf_counter()
    allScore += score
    #y_pred.append(preResult)
    y_pred = y_pred + list(preResult)

    for i in range(len(tdataLabel)):
        y_true.append(tcn)


tet = time.perf_counter()
print("Testing All class total spent {:.6f}s.".format(tet-tst))
print("sum of test cases: {}".format(allCount))
print("All error Count is: {}.".format(allErrCount))
avgAccuracy = allScore/10.0
print("Average accuracy is: {:.6f}.".format(avgAccuracy))
print("Average error rate is: {:.6f}.".format(1-avgAccuracy))
```

```
class 0 has 1001 test cases
errorCount: 3.
****************************
class 1 has 1127 test cases
errorCount: 5.
****************************
class 2 has 991 test cases
errorCount: 10.
****************************
class 3 has 1032 test cases
errorCount: 22.
****************************
class 4 has 980 test cases
errorCount: 6.
****************************
class 5 has 863 test cases
errorCount: 9.
****************************
class 6 has 1014 test cases
errorCount: 2.
****************************
class 7 has 1070 test cases
errorCount: 15.
****************************
class 8 has 944 test cases
errorCount: 12.
****************************
class 9 has 978 test cases
errorCount: 14.
****************************
Testing All class total spent 161.293890s.
sum of test cases: 10000
All error Count is: 98.
Average accuracy is: 0.990159.
Average error rate is: 0.009841.
```

计算召回率、F1值

In [7]:

```
# 生成混淆矩阵
print("生成混淆矩阵：")
cm = metrics.confusion_matrix(y_true, y_pred)
print(cm)
# 计算每个类别的召回率
print("每个类别的召回率：")
recall_per_class = metrics.recall_score(y_true, y_pred, average=None)
print(recall_per_class)

# 计算整体召回率（Macro-F1）
print("整体召回率：")
recall_macro = metrics.recall_score(y_true, y_pred, average='macro')
print(recall_macro)
print('\n')

# 计算每个类别的f1值
print("每个类别的f1值：")
f1_per_class = metrics.f1_score(y_true, y_pred, average=None)
print(f1_per_class)

# 计算整体f1值（Macro-F1）
print("整体f1值：")
f1_macro = metrics.f1_score(y_true, y_pred, average='macro')
print(f1_macro)
```

```
生成混淆矩阵：
[[ 998    0    0    0    2    0    1    0    0    0]
 [   0 1122    1    0    1    0    0    0    1    2]
 [   0    3  981    1    1    1    0    1    2    1]
 [   0    1    2 1010    0    4    0    8    4    3]
 [   0    1    0    0  974    0    0    0    0    5]
 [   1    0    1    3    2  854    2    0    0    0]
 [   1    0    0    0    0    1 1012    0    0    0]
 [   0    2    3    0    5    0    0 1055    1    4]
 [   1    4    1    0    0    3    2    1  932    0]
 [   3    2    0    3    5    0    0    1    0  964]]
每个类别的召回率：
[0.997003   0.99556344 0.98990918 0.97868217 0.99387755 0.98957126
 0.99802761 0.98598131 0.98728814 0.98568507]
整体召回率：
0.9901588736005473


每个类别的f1值：
[0.99551122 0.99204244 0.99090909 0.98584675 0.98883249 0.98957126
 0.99655342 0.98782772 0.98938429 0.9851814 ]
整体f1值：
0.9901660084197468
```