机器学习作业： KNN_MNIST

开发环境

安装 Anaconda，在 Jupyter notebook 进行开发测试。生成 python3 文件 knn_mnist。进行后续的操作。对比以前使用过的 pycharm，Jupyter 更加便利。

源代码

```
import numpy as np
from os import listdir
from sklearn.neighbors import KNeighborsClassifier as KNN

#将 32*32 的二进制图像转换为 1x1024 向量

def img2vector(filename):
    returnVect = np.zeros((1, 1024))       #创建 1x1024 零向量
    fr = open(filename)       #打开文件
    for i in range(32):       #读一行数据
        lineStr = fr.readline()
        for j in range(32):       #每一行的前 32 个元素依次添加到 returnVect 中
            returnVect[0, 32*i+j] = int(lineStr[j])
    return returnVect       #返回转换后的 1x1024 向量


#手写数字分类测试

def handwritingClassTest():
    #训练集的 Labels
    hwLabels = []
    #返回 trainingDigits 目录下的文件名
    trainingFileList = listdir('E:/trainingDigits')
    #返回文件夹下文件的个数
    m = len(trainingFileList)
    #初始化训练的 Mat 矩阵,训练集
    trainingMat = np.zeros((m, 1024))
    #从文件名中解析出训练集的类别
    for i in range(m):
        #获得文件的名字
        fileNameStr = trainingFileList[i]
        #获得分类的数字
```

```
            classNumber = int(fileNameStr.split('_')[0])
            #将获得的类别添加到 hwLabels 中
            hwLabels.append(classNumber)
            #将每一个文件的 1x1024 数据存储到 trainingMat 矩阵中
            trainingMat[i,:] = img2vector('E:/trainingDigits/%s' % (fileNameStr))
        #构建 kNN 分类器
        neigh =KNN(n_neighbors = 3, algorithm = 'auto')
        #拟合模型, trainingMat 为训练矩阵,hwLabels 为对应的标签
        neigh.fit(trainingMat, hwLabels)
        #返回 testDigits 目录下的文件列表
        testFileList = listdir('E:/testDigits')
        #错误检测计数
        errorCount = 0.0
        #测试数据的数量
        mTest = len(testFileList)
        print("Test Results \t Real results")
        #从文件中解析出测试集的类别并进行 分类测试
        for i in range(mTest):
            #获得文件的名字
            fileNameStr = testFileList[i]
            #获得分类的数字
            classNumber = int(fileNameStr.split('_')[0])
            #获得测试集的 1x1024 向量,用于训练
            vectorUnderTest = img2vector('E:/testDigits/%s' % (fileNameStr))
            #获得预测结果
            classifierResult = neigh.predict(vectorUnderTest)

            print("       %d        \t       %d " % (classifierResult, classNumber))
            if(classifierResult != classNumber):
                errorCount += 1.0
        print("\nError rate    :     %f%%\nCorrect rate:    %f%%" %
                (errorCount/mTest * 100, (1-errorCount/mTest) * 100))


    #main 函数

    if __name__=='__main__':
        handwritingClassTest()
```

训练与测试数据

将白色部分设为 0，黑色部分设为 1，MNIST 集变成 32*32 的二进制图像，以便进行后续操作。测试集 testDigits 训练集 trainDigits（图 1、2）保存在 E 盘。

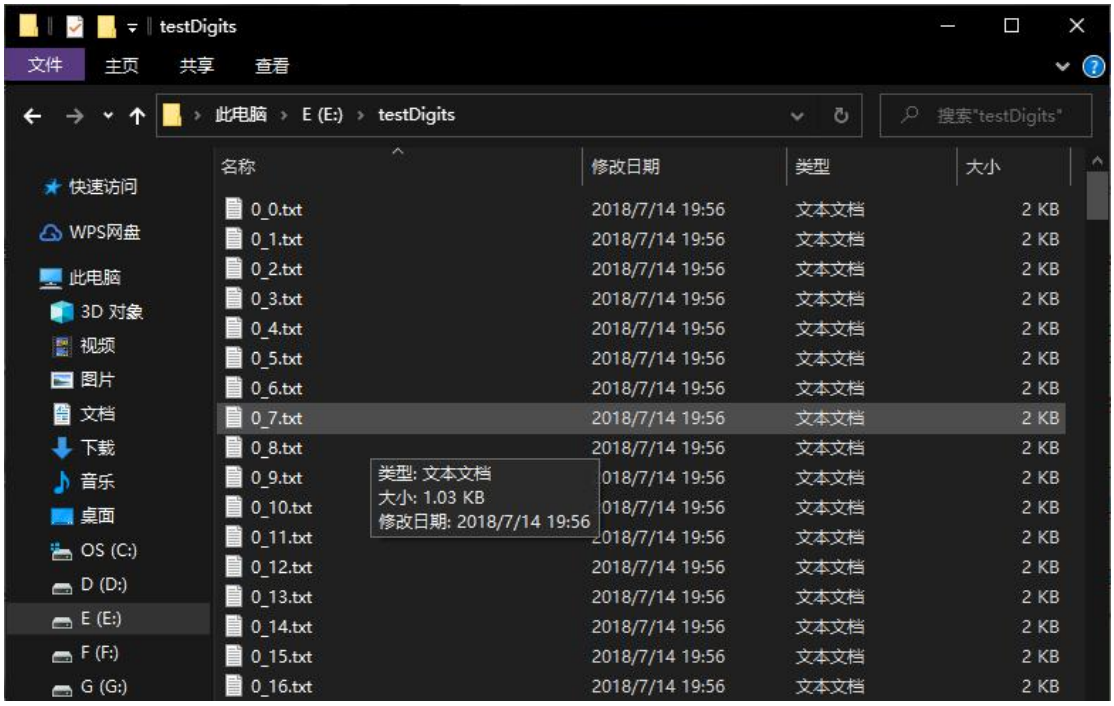

图 1 测试集数据

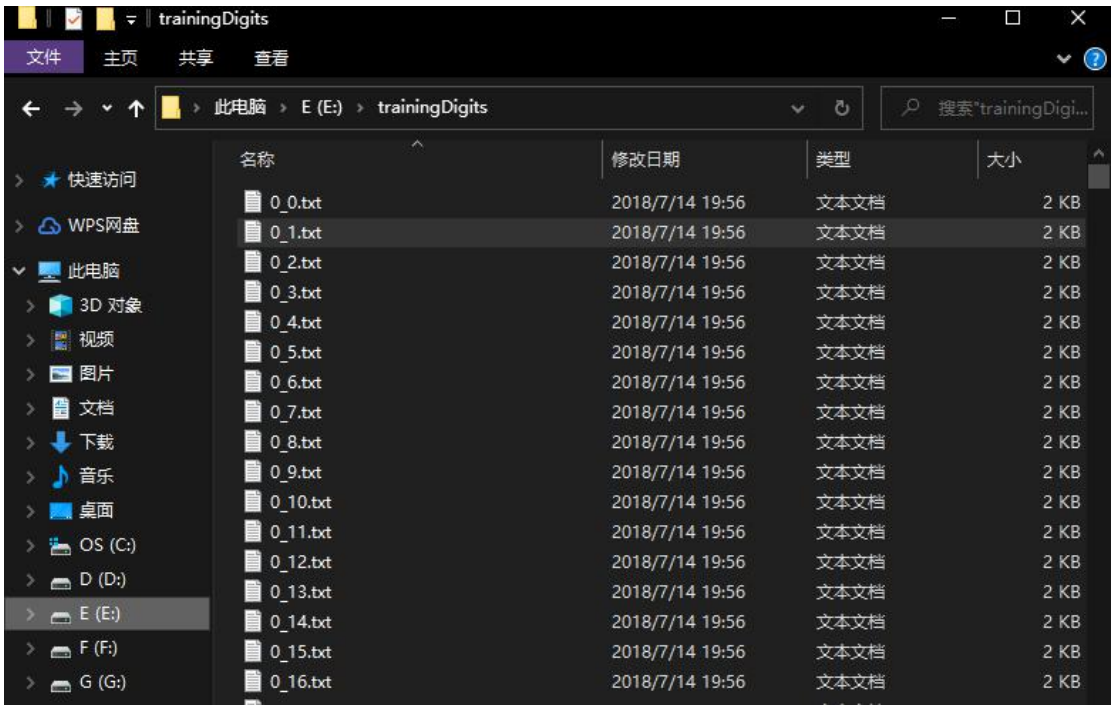

图 2 训练集数据

测试结果

输出形式如下图 3，会出现一些识别错误，进行统计计算。

Test Results	Real results
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0

图 3 输出格式

9	9
9	9
9	9
9	9
9	9

Error rate : 1.268499%
Correct rate: 98.731501%

图 4 结果统计分析

小结感悟

本次实验是用 KNN 算法的思想来解决手写数字的识别问题。进一步加深了我对 KNN 算法的理解。我体会到了老师视频中所讲的“KNN 算法思想简单，但是可以处理很多问题。”这次将图片处理为二进制形式再进行操作的方法在几乎不影响准确度的前提下简单了很多。为此，我专门找到了去年的 CNN_MNIST 作业来对比（见图 5）。当然 CNN 的算法复杂得多，测试数据也更全面。

```
Iter 0, Testing Accuracy= 0.9566
Iter 1, Testing Accuracy= 0.9724
Iter 2, Testing Accuracy= 0.9781
Iter 3, Testing Accuracy= 0.9812
Iter 4, Testing Accuracy= 0.9841
Iter 5, Testing Accuracy= 0.9861
Iter 6, Testing Accuracy= 0.9865
Iter 7, Testing Accuracy= 0.9869
Iter 8, Testing Accuracy= 0.9887
Iter 9, Testing Accuracy= 0.9902
Iter 10, Testing Accuracy= 0.9896
Iter 11, Testing Accuracy= 0.99
Iter 12, Testing Accuracy= 0.9899
Iter 13, Testing Accuracy= 0.9897
Iter 14, Testing Accuracy= 0.991
Iter 15, Testing Accuracy= 0.9907
Iter 16, Testing Accuracy= 0.9897
Iter 17, Testing Accuracy= 0.9913
Iter 18, Testing Accuracy= 0.9913
Iter 19, Testing Accuracy= 0.9918
Iter 20, Testing Accuracy= 0.9923
```

图 5　CNN 算法的准确度