

# 华中科技大学计算机科学与技术学院


## 机器学习报告



专    业： 计算机科学与技术

班    级：

学    号：

姓    名： 

成    绩：

指导教师：

完成日期： 2020 年    月    日

## 实验一

### 一、实验题目：KNN（K 近邻）

### 二、实验要求

用 KNN 算法实现 MNIST（手写数字识别），实现要求如下：

1. 给定训练集，确定参数  $K$ ，输入单张测试图片，输出这张图片和它的  $K$  近邻的图片，输出它的真实标签和测试标签。
2. 给定训练集和测试集，确定参数  $K$ ，输出在这个训练集上测试这个测试集的正确率。
3. 给定训练集和测试集，确定参数  $K$  的区间和步长，输出不同的  $K$  得到的不同的正确率，绘制 KNN 算法的训练 misclassification rate 曲线。

### 三、算法设计

#### 1. 数据集读取算法设计

根据 .idx3-ubyte 文件的规范，在文件头部读取这个文件的相关信息（照片数、行数、列数），然后按照（行、列）二维矩阵的方式读取每一张照片。用同样方法读取以 .idx1-ubyte 文件存储的标签。

#### 2. KNN 算法设计

用 for 循环计算被测试图片与训练集中每一张图片的距离（欧式距离），然后选出距离最小的  $K$  张图片，统计这  $K$  张图片的各标签出现次数，其中出现次数最多的标签就是预测结果。

### 四、实验环境与平台

本实验使用的语言为 python，版本为 python3.7，实现的开发工具是，系统

为 Windows 10, CPU 为 i7-8550U, 内存 8G, 其后的测试也均在此硬件环境下。

## 五、程序实现

```
import numpy as np
import struct
import matplotlib.pyplot as plt

def idx1_ubyte_parser(idx1_ubyte_filename):
    #read the idx1_ubyte_file in binary format
    data_b = open(idx1_ubyte_filename, 'rb').read()

    #principal information about the labels
    offset = 0
    format_header = '>ii'
    magic_number, num_labels = struct.unpack_from(format_header, data_b, offset)

    #read the labels
    offset += struct.calcsize(format_header)
    format_label = '>B'
    labels = np.empty(num_labels)
    for i in range(num_labels):
        labels[i] = struct.unpack_from(format_label, data_b, offset)[0]
        offset += struct.calcsize(format_label)

    labels = labels.reshape(num_labels)

    return labels

def idx3_ubyte_parser(idx3_ubyte_filename):
    #read the idx3_ubyte file in binary format
    data_b = open(idx3_ubyte_filename, 'rb').read()

    #principal information about the dataset
    offset = 0
    format_header = '>iiii'
    magic_number, num_images, num_rows, num_columns = struct.unpack_from(format_header,
data_b, offset)

    #read the images
    image_size = num_rows * num_columns
    offset += struct.calcsize(format_header)
    format_image = '>' + str(image_size) + 'B'
    images = np.empty((num_images, num_rows, num_columns))
    for i in range(num_images):
        images[i] = np.array(struct.unpack_from(format_image, data_b, offset)).reshape(num_rows,
num_columns)
        offset += struct.calcsize((format_image))

    #plot figure
    #plt.imshow(images[i], 'gray')
    #plt.pause(0.0001)
    #plt.show()

    return images

def KNN(K, training_images, training_labels, test_image):
```

```

#calculate distances
(training_range, y, z) = np.shape(training_images)
distances = []
for i in range(training_range):
    difference = (test_image - training_images[i]) **2
    distance = np.sum(np.reshape(difference,(difference.size,)))
    distances.append(distance)

#sort the distances and get the corresponding labels
index = np.argsort(distances)
count = np.zeros((10))
plt.figure()
for i in range(K):
    label = int(training_labels[index[i]])
    count[label] = count[label] + 1
    #plot figure
    plt.subplot(1,K,i+1)
    plt.imshow(training_images[index[i]], 'gray')
    plt.xticks([])
    plt.yticks([])
plt.show()

#find the most class in its K Nearest Neighbours
classes = np.argsort(count)

return classes[9]

def test_accuracy(K, training_images, training_labels, testing_images, testing_labels):
    right = 0
    (testing_range, y, z) = np.shape(testing_images)
    for test_position in range(testing_range):
        test_image = testing_images[test_position]
        prediction = KNN(K, training_images, training_labels, test_image)
        if int(testing_labels[test_position]) == int(prediction):
            right += 1

    accuracy = right / testing_range

    return accuracy

if __name__ == '__main__':
    train_data_filename = 'train-images.idx3-ubyte'
    train_label_filename = 'train-labels.idx1-ubyte'

    test_data_filename = 't10k-images.idx3-ubyte'
    test_label_filename = 't10k-labels.idx1-ubyte'

    #test_label_filename = 'train-labels.idx1-ubyte'
    #test_data_filename = 'train-images.idx3-ubyte'

    train_images = idx3_ubyte_parser(train_data_filename)
    test_images = idx3_ubyte_parser(test_data_filename)
    train_labels = idx1_ubyte_parser(train_label_filename)
    test_labels = idx1_ubyte_parser(test_label_filename)

    mode = int(input('1.test single images\n2.test the accuracy\n3.draw misclassification rate\nchoose
a mode:'))

    if mode == 1:
        while True:

```



```

train_start = int(input('Enter the Start image of the traing datasets (0-59999) :'))
train_end = int(input('Enter the End image of the training datasets (0-60000) :'))
test_position = int(input('Enter the Test image of the testing datasets (0- 9999) :'))

#get the designated testing image and label & slice the training set
test_image = test_images[test_position]
training_images = train_images[train_start : train_end]
training_labels = train_labels[train_start : train_end]

#plot the figure
plt.figure()
plt.imshow(test_images[test_position], 'gray')
plt.pause(0.001)
plt.show()

K = int(input('Parameter K = '))

prediction = KNN(K, training_images, training_labels, test_image)

print('The true label of the tested image :',int(test_labels[test_position]))
print('The prediction of KNN with K equal to ', K, ' : ', prediction)

stop = input('To Exit? y/n :')
if stop == 'y':
    break

elif mode == 2:
    train_start = int(input('Enter the Start image of the traing datasets (0-59999) :'))
    train_end = int(input('Enter the End image of the training datasets (0-60000) :'))
    test_start = int(input('Enter the Start image of the testing process (0- 9999) :'))
    test_end = int(input('Endter the End image of the testing process (0-10000) :'))
    K = int(input('Parameter K = '))

    training_images = train_images[train_start : train_end]
    training_labels = train_labels[train_start : train_end]
    testing_images = train_images[test_start : test_end]
    testing_labels = train_labels[test_start : test_end]

    accuracy = test_accuracy(K, training_images, training_labels, testing_images, testing_labels)

    print('accuracy = ', accuracy)

elif mode == 3:
    train_start = int(input('Enter the Start image of the traing datasets (0-59999) :'))
    train_end = int(input('Enter the End image of the training datasets (0-60000) :'))
    test_start = int(input('Enter the Start image of the testing process (0- 9999) :'))
    test_end = int(input('Endter the End image of the testing process (0-10000) :'))
    K_start = int(input('Enter the Smallest K of the testing process :'))
    K_end = int(input('Enter the Biggest K of the testing process :'))
    K_step = int(input('Enter the step of K of the testing process :'))

    training_images = train_images[train_start : train_end]
    training_labels = train_labels[train_start : train_end]
    testing_images = train_images[test_start : test_end]
    testing_labels = train_labels[test_start : test_end]

    accuracy = []
    mis_rate = []

    for K in range(K_start, K_end, K_step):

```

```

acc = test_accuracy(K, training_images, training_labels, testing_images, testing_labels)
accuracy.append(acc)
mis_rate.append(1-acc)
print('K = ', K, 'accuracy = ', acc)

T = np.array(range(K_start, K_end, K_step))
power = np.array(mis_rate)
plt.plot(T, power)
plt.scatter(T, power)
plt.xlabel('K')
plt.ylabel('misclassification rate')
plt.show()

else:
    print('Invalid Input!')

```

## 六、实验结果

### 1. 单张图片测试

测试功能：输出这张图片和它的 K 近邻的图片，给出预测标签。

测试样例：测试集中第 123 号图片（见图 1-1）。

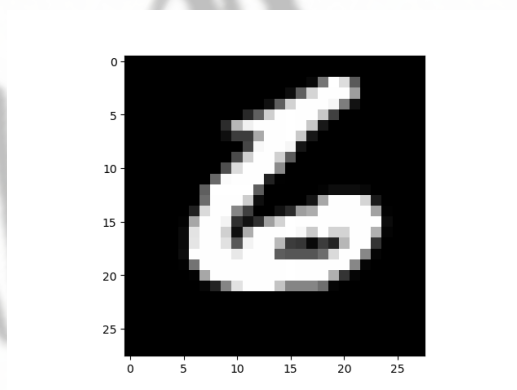


图 1-1 测试集中第 123 号图片

测试结果：预测标签为 6，其 K 近邻（K=13）的图片见图 1-2。



图 1-2 K 近邻的图片

### 2. 正确率测试

（1）取 K=3，训练集为全部 60000 张图片，测试集为测试集全部 10000 张图片，正确率为 98.74%.

（2）取 K=3，训练集为前 60000 张图片，测试集也为训练集全部 60000 张

图片，正确率为 98.705%.

3. 错误率与参数 K 的关系

(1) 测试集范围越来越大

训练集范围为前 10000 张图片，K 的测试范围为 1 到 119，步长为 2，根据错误率画出来的 misclassification rate 曲线见图 1-3。其中图 1-3.(a)的测试范围为测试集前 100 张图片，图 1-3.(b)的测试范围为测试集前 500 张图片，图 1-3.(c)的测试范围为测试集前 1000 张图片，图 1-3.(d)的测试范围为测试集前 10000 张图片。

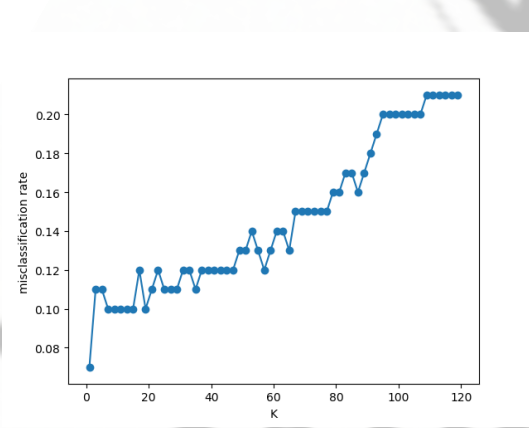


图 1-3.(a)

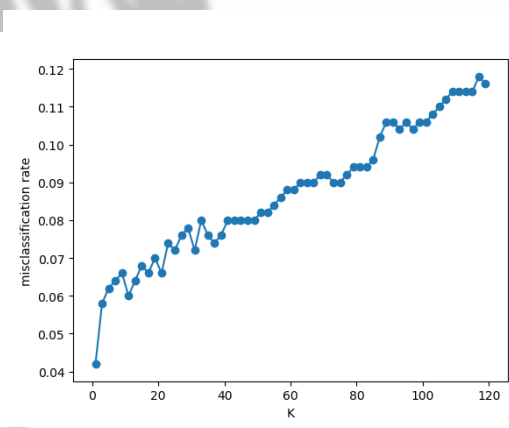


图 1-3.(b)

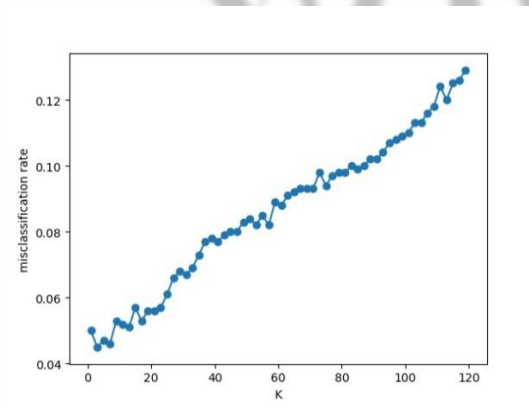


图 1-3.(c)

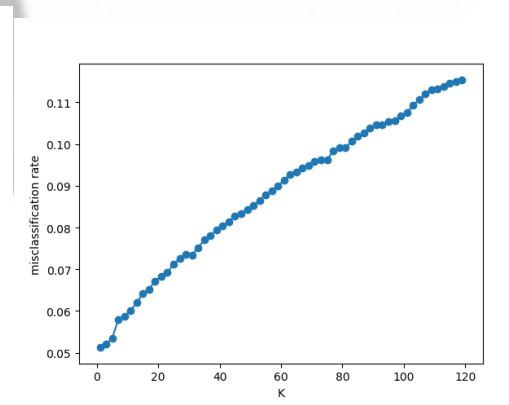


图 1-3.(d)

图 1-3 错误率与参数 K 的关系曲线

(2) 训练集范围越来越大

测试集范围为测试集前 2000 张图片，K 的测试范围为 1-119，步长为 2，根据错误率画出来的 misclassification rate 曲线见图 1-4。其中图 1-4.(a)的训练集范围为训练集前 10000 张图片，图 1-4.(b)的训练集范围为训练集前 60000 张图片。

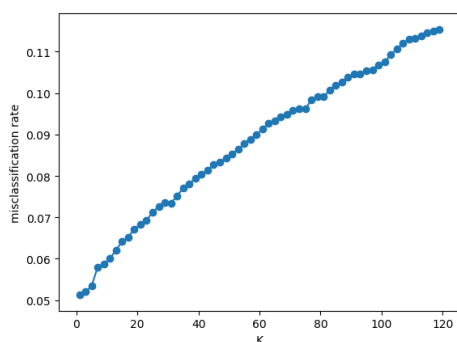


图 1-4.(a)

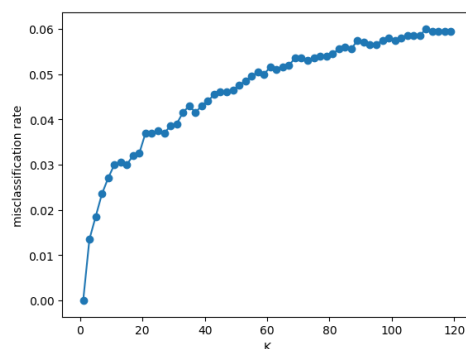


图 1-4.(b)

图 1-4 错误率与参数 K 的关系曲线

(3) 用训练集来测试训练集与用训练集测试测试集对比

训练集范围为训练集前 10000 张图片，K 的测试范围为 1-119，步长为 2，根据错误率画出来的 misclassification rate 曲线见图 1-4。其中图 1-4.(a)的测试集范围为训练集前 1000 张图片，图 1-4.(b)的测试集范围为测试集前 1000 张图片。

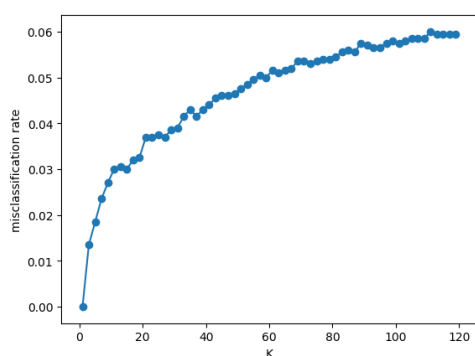


图 1-5.(a)

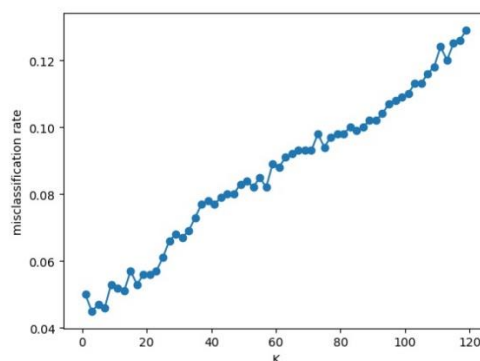


图 1-5.(b)

图 1-5 错误率与参数 K 的关系曲线

## 七、结果分析

1. 当 K 取得较小时（比如不超过 20），KNN 算法对于 MNIST 的正确率一般不会低于 0.95（训练样本和测试集足够多），且当 K 小于 10 时，其正确率一般不会低于 97%（训练样本和测试集足够多），这说明 KNN 算法对于解决以 MNIST 为代表的某些非线性分类问题具有非常好的效果。

2. 由图 1-3 可知，随着测试集样本数量的增大，错误率的波动越来越小，这一点表现为错误率曲线随着测试集样本数量的增加越来越平滑。除此以外，还可以看出，总体上错误率和参数 K 值成正相关。



3. 由图 1-4 可知，随着训练集样本的增加，错误率明显下降，但错误率与参数  $K$  值的正相关关系并没有改变。

4. 由图 1-5 可知，用训练集测试其本身的错误率明显小于用训练集测试测试集，但是错误率与参数  $K$  值得正相关关系也没有改变。

5. 错误率与参数  $K$  值得正相关关系可能与决策边界随着  $K$  值增大越来越平滑有关，因为这会导致某些奇异值在平滑化的过程中被分错类，从而导致错误率得升高。