Machine Learning Report

The Ohio State University

Xuanzhi Zhang(zhang.10419)

April 13, 2022

Definition

Project Overview

These days, it is common to import handwritten digital files into a computer for storage. This project aims to label a single 28x28 pixels handwritten number. In this project, I used 1000 images of MNIST digit 0-9 number data sets as the training set, and I want to label all 1000 images of the test data set. With the KNN algorithm, the computer can identify and match a single test image with each image in the training set and then assign the image's label in the training set that it considers most similar to the test image.

Data sets

- MNIST-JPG
  - Download address: https://github.com/teavanist/MNIST-JPG/blob/master/MNIST%20Dataset%20JPG%20format.zip

I chose 100 images for each 0-9 number and renamed the file to a specific format.

Algorithm and Techniques

The classifier is a K-Nearest Neighbor, which is an algorithm rely on a distance metric.

- Assumption: Similar inputs(feature vectors) have similar outputs(labels)
- Classification rule: for the test data input x, assign the most common label among its K most similar training data instances in $\boldsymbol{D_{tr}}$
- Equation: $\|x - x_n\|_2 = (\sum_{d=1}^{D} |x[d] - x_n[d]|^2)^{1/2}$

Methodology

Data Preprocessing

The preprocessing done in the “knn.py” file consists of the following steps:

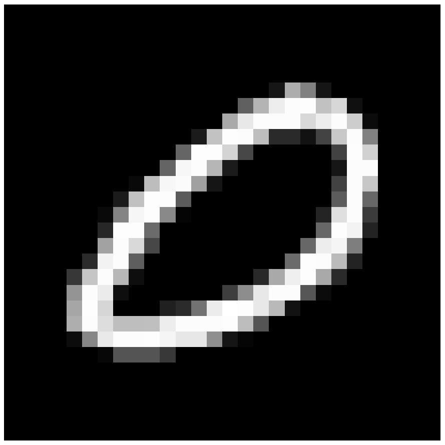
1. Read every image in the “imgs_test” folder.
2. Extract the features of each 28px * 28px image into a 1*784 matrix.
3. Calculate accuracy of testing procedures.
4. Calculating Euclidean distance from the test image to the data set.
5. Make list for the possible output and get the best solution.

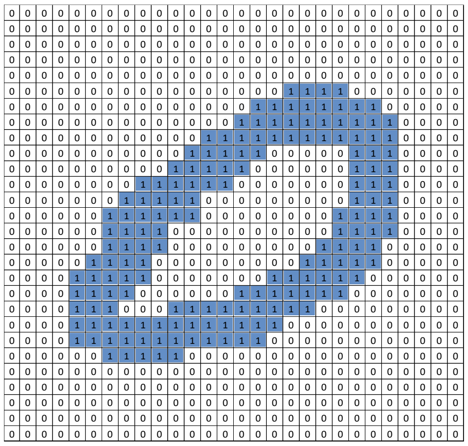
Program Analysis

Img2vector(img_path): Extract image features and return a ***1* 784****(28*28)* matrix

```
def img2vector(img_path):
    """
    Format the input data image and convert it to a vector
    return: 1*784 number vector
    """
    # Using 0 to read image in grayscale mode
    image = cv2.imread(img_path, 0)
    dataVect = np.zeros((1, 784))
    # extract data from 28*28 image with corresponding img path
    for i in range(28):  # y direction
        for j in range(28):  # x direction
            if int(image[i][j]) != 0:  # if the pixel is not white
                dataVect[0, 28 * i + j] = int(1)  # put 1 into the 1 * 784
matrix
            else:  # if the pixel has color
                # Offsets are used to distribute the entire
                # handwritten numeric data across an array
                dataVect[0, 28 * i + j] = int(image[i][j])  # put 0 into the
1 * 784 matrix
    return dataVect
```

Fig. 1 The left picture is a 28px*28px number "0" The right picture is the matrix from Img2vector(img_path)





0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

KNN_classify0((inX, dataSet, labels, k): Calculating Euclidean distance

```
def kNN_classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
```

```python
    # Calculating Euclidean distance
    diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet
    sqDiffMat = diffMat ** 2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances ** 0.5
    sortedDistIndicies = distances.argsort()
    classCount = {}

    # Select k points with the smallest distance
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1

    # Sorting
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1),
reverse=True)
    # print(sortedClassCount)
    return sortedClassCount[0][0]
```

$$\left(\sum_{d=1}^{D} |x[d] - x_n[d]|^2\right)^{1/2}$$

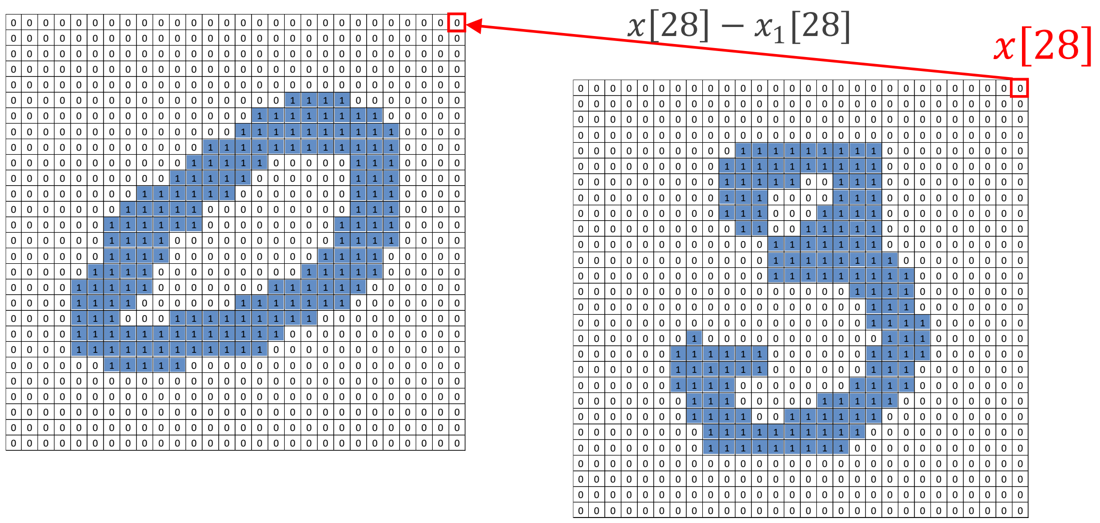


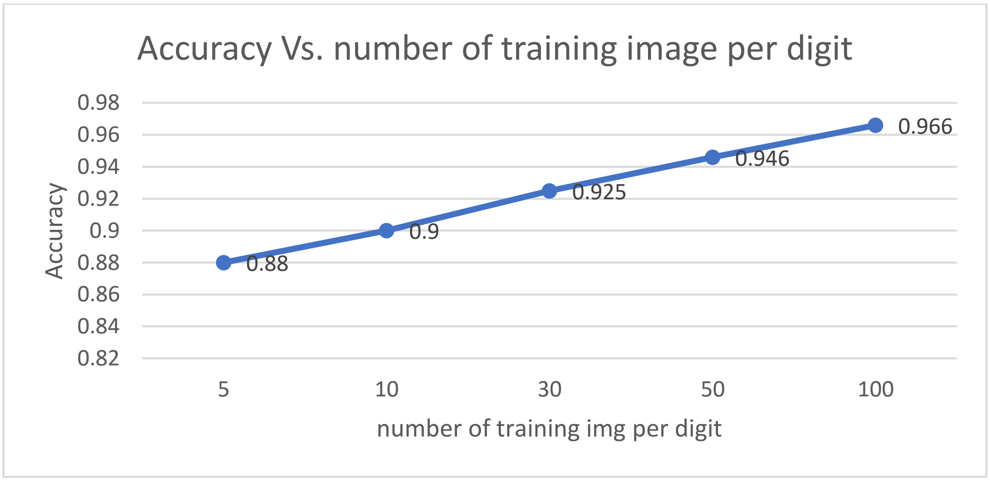
Fig. 2 The Euclidean distance from one pixel to another pixel

def handwritingClassficationTest(): Calculate the accuracy of algorithm.

```python
def handwritingClassficationTest():
    # loading the image data from the file path----start
    hwLabels = []
    path = './imgs_test'
    file_list = os.listdir(path)
    number = 0
    for i in file_list:
        img_path = os.path.join(path, i)
        img_list = os.listdir(img_path)
        number += len(img_list)
    m = number
    num = 0
    trainingMat = np.zeros((m, 784))
    # loading the image data from the file path----end
    for i in file_list:
        img_path = os.path.join(path, i)
        img_list = os.listdir(img_path)
        for j in img_list:
            img_path_one = os.path.join(img_path, j)
            # print(img_path_one)
            trainingMat[num, :] = img2vector(img_path_one)
            num += 1
            hwLabels.append(int(i))

    test_path = './test'
    test_file_list = os.listdir(test_path)
    number_test = 0
    errorCount = 0
    for i in test_file_list:
        img_path = os.path.join(path, i)
        img_list = os.listdir(img_path)
        number_test += len(img_list)
        for j in img_list:
            vectorUnderTest = img2vector(os.path.join(img_path, j))
            classNumStr = int(i)
            classifierResult = kNN_classify0(vectorUnderTest, trainingMat,
hwLabels, 3)
            if classifierResult != classNumStr:
                errorCount += 1.0

    print("\nthe total number of errors is: %d" % errorCount)
    print("\nthe total accuracy is: %f" % (1 - errorCount /
float(number_test)))
```

Result:

```
the total number of errors is: 34

the total accuracy is: 0.966000
```



From the above results, we can find that the accuracy rate of KNN is unexpectedly as high as 96.6% after 1000 test images. There is such a high possibility because the image complexity displayed by the 28x28px image is not high, and the shape overlap rate of each number is not high, so the probability of errors is low.

Problems faced along the way

- Load pictures from a folder into the program
  - The initial program is to use multiple for loops to import pictures through name retrieval, and then modified to use "os.listdir(path)"
- Select the K points with the smallest distance
  - At first, the label that appeared first was used as the best choice, but the accuracy rate was not high. Now it is changed to the best choice with the highest number of matched labels.
- extract data from 28*28 image to a 1*784 feature vector
  - At first, I used the opencv library to extract the information into a 1*784 feature vector and found that it would be very complicated to calculate the Euclidean distance, so I changed it to a binary value vector, that is, 0 as no color and 1 as color.

Future works

1. Try other algorithms. Although the KNN algorithm has a high success rate, it currently only has 1000 image training sets for each single image retrieval time process. If the images are too complex or you need to obtain results in real time, the KNN algorithm is not the best choice.

2. Currently, the Euclidean distance formula is used, but for binary value vector, the L1 formula can be used to calculate the distance, which may reduce the operation time. Or continue to use the Euclidean distance, divide the pixel color value (0-225) into multiple intervals (dimension), and then calculate the result by calculating the distance in different dimensions.

3. Try putting letters or Chinese text or other images in for testing

4. The current image is 28*28px. You can try to expand the image to 56*56px first, and analyze the runtime and accuracy rate to consider the range that the KNN algorithm can use.