1、KNN recognizes handwritten digits

The Raspberry Pi motherboard series does not currently support this tutorial.

1.1、KNN (K nearest neighbor algorithm) recognizes handwritten digits

1.1.1、Introducing KNN

1）、 KNN (K-Nearest Neighbor) is a supervised learning method. Its working mechanism is very simple and does not require training a training set. It is one of the simpler classic machine learning algorithms. Can handle regression problems and classification problems.

2）、Method ideas

In feature space, if most of the k nearest (that is, the nearest neighbor in feature space) samples near a sample belong to a certain category, then the sample also belongs to this category.

In official terms, the so-called K nearest neighbor algorithm means that given a training data set, for a new input instance, find the K instances closest to the instance in the training data set (that is, the K neighbors mentioned above), the majority of these K instances belong to a certain class, and the input instance is classified into this class.

3）、working principle

There is a sample data set, also called a training sample set, and each data in the sample set has a label, that is, we know the relationship between each data in the sample set and its category. After inputting data without labels, compare each feature in the new data with the features corresponding to the data in the sample set, and extract the classification label of the data (nearest neighbor) with the most similar features in the sample set. Generally speaking, we only select the top K most similar data in the sample data set. This is where K in the K nearest neighbor algorithm comes from. Usually K is an integer not greater than 20. Finally, the category that appears the most among the K most similar data is selected as the category of the new data.

4）、KNN advantages and disadvantages

- advantage

  Flexible usage, convenient for small sample prediction, high accuracy, insensitive to outliers, no data input assumptions

- shortcoming

  Lack of training stage, inability to handle multiple samples, high computational complexity and high space complexity

5）、KNN implementation steps：

- Calculate distance

  Euclidean distance：

$$L_2(x_i, x_j) = (\sum_{l=1}^{n} |x_i{}^{(l)} - x_j{}^{(l)}|^2)^{\frac{1}{2}}$$

- Sort by increasing distance
- Select the K points with the smallest distance (generally not more than 20)
- Determine the frequency of occurrence of the category where the first K points belong, frequency of occurrence = a certain category/k
- Return the category with the highest frequency among the first K points as the predicted classification of the test data

1.1.2、Taking recognition of handwritten digits as an example to introduce the implementation of KNN algorithm

1）、data set

- Training set

  The training set is data that has been classified. You can refer to the directory ~/KNN/knn-digits/trainingDigits .

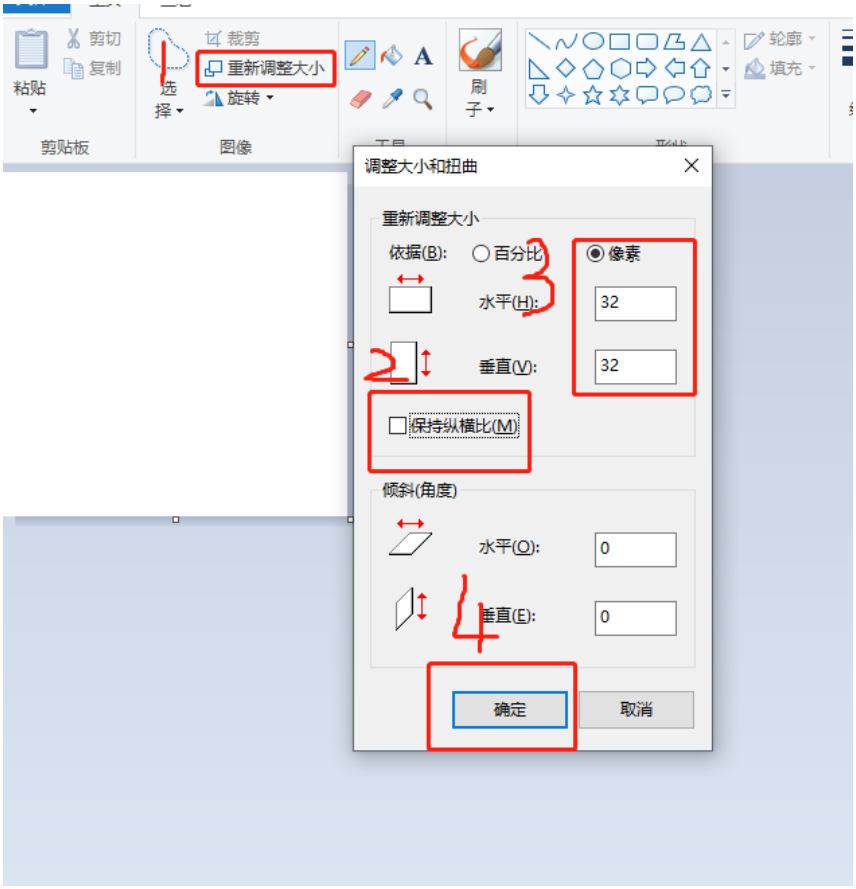
  Note: The training set has been trained. If users want to train by themselves, they need to back up the original training set first. The number of training sets is large.

  test set

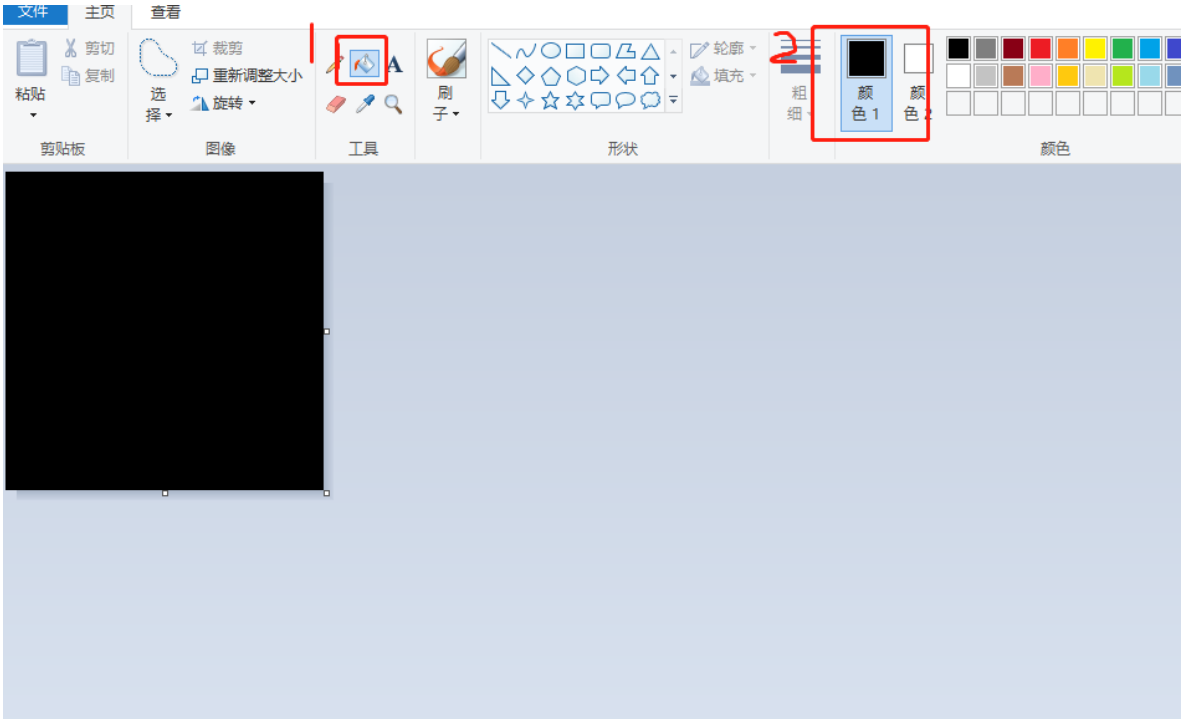
The test set is used to test the algorithm. You can refer to the directory ~/KNN/knn-digits/testDigits

2）、Create handwritten digital pictures using the drawing function under Windows

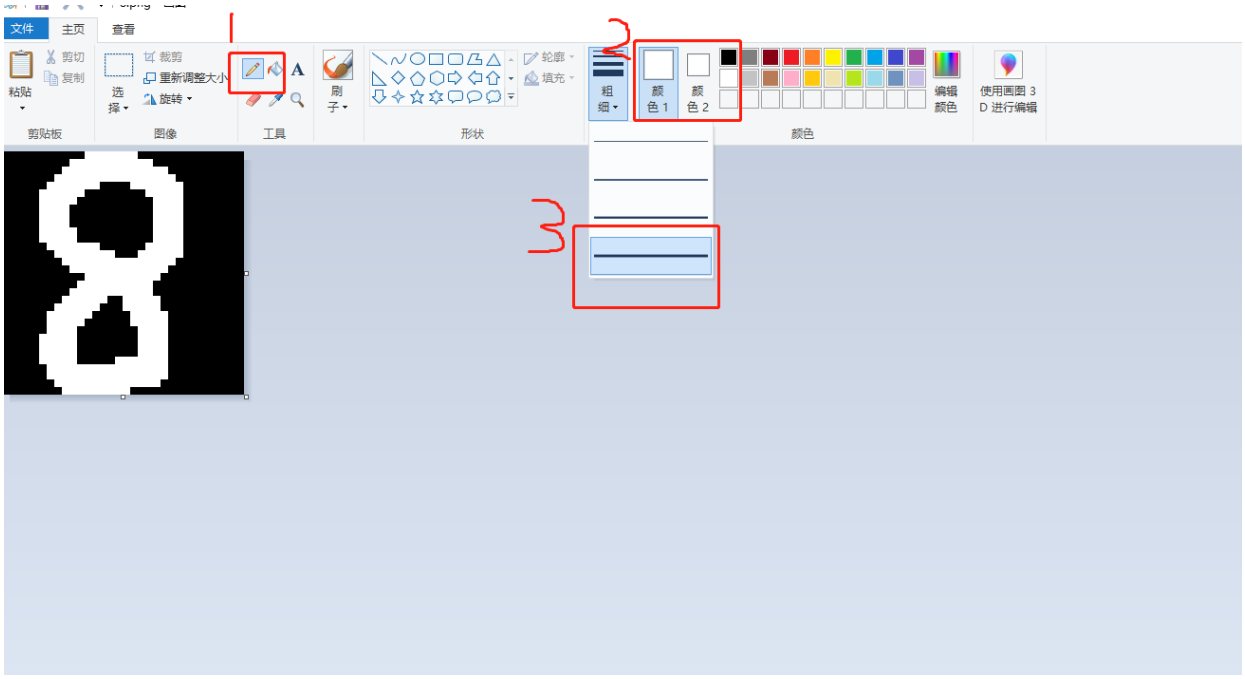
- Open the drawing software and adjust the resolution to 32*32. Other resolutions can also be used, but it is best to draw the picture to fill the entire interval, otherwise the error rate will be high.



- Hold down Ctrl and slide the mouse wheel up to enlarge the image to its maximum size. Select the Paint Bucket Tool and choose a black color to fill the entire image.



- Select the pencil tool, select white for color, and select maximum width for line thickness. As shown below:



After drawing, save it as a png picture (this example uses 8.png as an example), and copy it to the project directory through the WinSCP tool.

3）、Convert image (.img) to text (.txt)

- code

code location：~/KNN/img2file.py

```
from PIL import Image

import numpy as np


def img2txt(img_path, txt_name):

    im = Image.open(img_path).convert('1').resize((32, 32))  # type:Image.Image

    data = np.asarray(im)

    np.savetxt(txt_name, data, fmt='%d', delimiter='')
img2txt("8.png","./knn-digits/testDigits/8_1.txt")
```

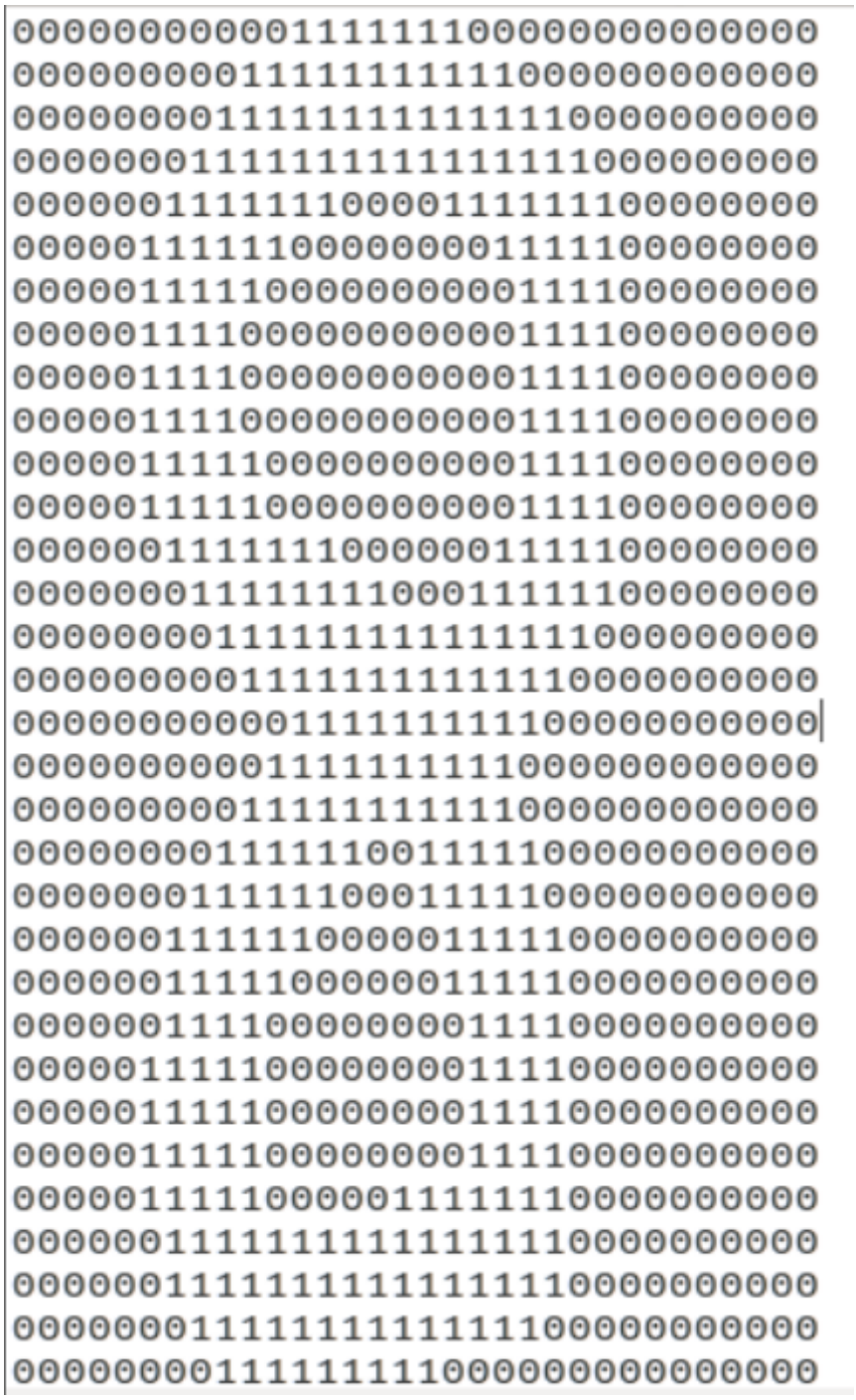
img_path：Image file name

txt_name: After conversion, it is saved in the ~/KNN/knn-digits/testDigits directory with the name 8_1.txt

- Run program

```
cd KNN
python img2file.py
```

After the program is run, a file named 8_1.txt will be generated in the KNN directory. Double-click it and you will see something like this:



You can see that all the parts of the number 1 roughly form a number 8.

4）、Run the KNN recognition algorithm program

In the ~/KNN directory, open the terminal and run

```
python knn.py
```

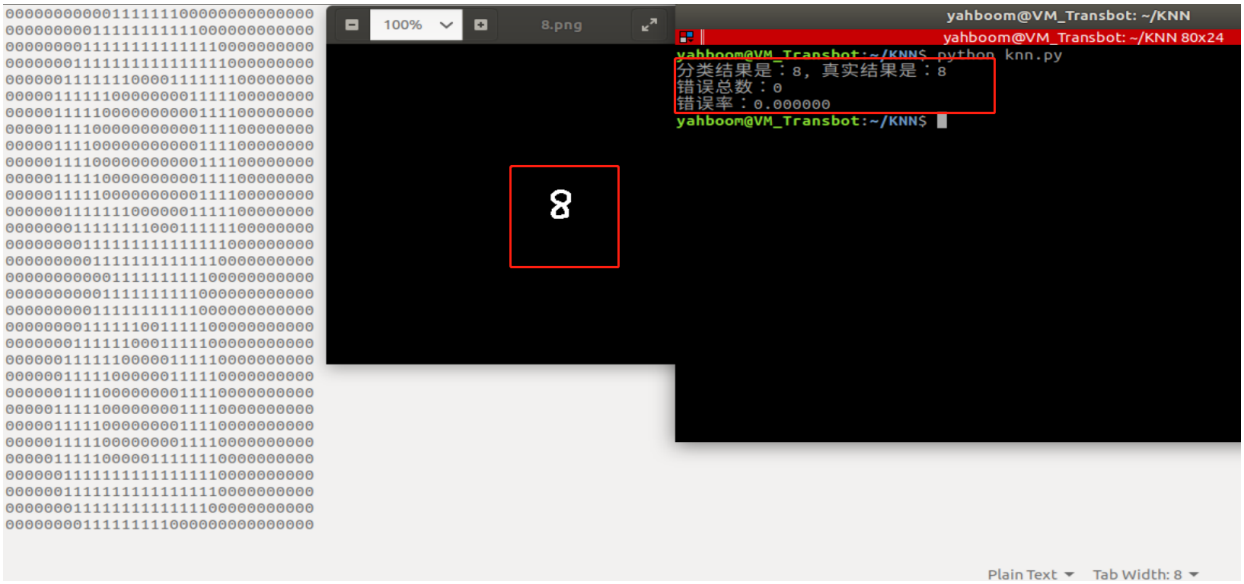
- code

code location：~/KNN/knn.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 16 14:53:46 2017

@author: jiangkang
"""


import numpy
import operator
import os
import matplotlib.pyplot as plt

def img2vector(filename):
    returnVect = numpy.zeros((1, 1024))
    file = open(filename)
    for i in range(32):
        lineStr = file.readline()
        for j in range(32):
            returnVect[0, 32 * i + j] = int(lineStr[j])
    return returnVect

def classifier(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    diffMat = numpy.tile(inX, (dataSetSize, 1)) - dataSet
    sqDiffMat = diffMat ** 2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances ** 0.5
    sortedDistIndicies = distances.argsort()
    classCount = {}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1),
reverse=True)
    return sortedClassCount[0][0]
#Train first, then test recognition. k represents the k value in the algorithm -
select the top K most similar data in the sample data set. The k value is generally
an integer not greater than 20.
def handWritingClassTest(k):
    hwLabels = []
    trainingFileList = os.listdir('knn-digits/trainingDigits') #Read training set
data
    m = len(trainingFileList)
    trainingMat = numpy.zeros((m, 1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        trainingMat[i, :] = img2vector("knn-digits/trainingDigits/%s" % fileNameStr)
    testFileList = os.listdir('knn-digits/testDigits') #Read test set data
```

```
    errorCount = 0.0
    mTest = len(testFileList)
    for i in range(mTest):
        fileNameStr = testFileList[i]
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        vectorTest = img2vector("knn-digits/testDigits/%s" % fileNameStr)
        result = classifier(vectorTest, trainingMat, hwLabels, k)
        print("分类结果是：%d, 真实结果是：%d" % (result, classNumStr))
        if result != classNumStr:
            errorCount += 1.0
    '''fileStr = "2.txt"
    classNumStr = int(fileStr.split('.')[0])
    vectorTest = img2vector("./2.txt")
    result = classifier(vectorTest, trainingMat, hwLabels, 3)'''
    print("错误总数：%d" % errorCount)
    print("错误率：%f" % (errorCount / mTest))
    return errorCount

handWritingClassTest(1)
```

- Run screenshot



As shown in the picture, it is identified as 8, which is correct. If the recognition result is different from the real result, please copy the converted txt document to knn-digits/trainingDigits/ and name it as follows: 8_801.txt, and then retrain the recognition and the recognition will be normal.

- Program Description

  The text file converted from the image is named, taking 8_1 as an example. knn.py will parse the file name in the program, with underscores as the boundaries. The 8 in the front represents the real number, and the following part can be customized. See the code.

  ```
  classNumStr = int(fileStr.split('_')[0])
  ```

Training first, recognition later.