3.SVM recognition of handwritten numbers

The process of SVM:
① Collect data: provide text files
② Prepare data: construct vector based on binary image
③ Analyze the data: visually inspect the image
④Training algorithm: adopt two different kernel functions, and use different settings for the radial basis kernel function to run the SMO algorithm
⑤Test algorithm: write a function to test different kernel functions and calculate the error rate
⑥ Using algorithm: SVM can be used in almost all classification problems. SVM itself is a second-class classifier. Applying SVM to multi-class problems requires some modifications to the code.

KNN vs SVM
Similarities:
Both are more classic machine learning classification algorithms, both belong to supervised learning algorithms.
Difference:
①KNN must consider each sample. SVM is to find a function to divide the sample to reach.
②The essence of SVM is to find weights.
③KNN cannot handle things with high-dimensions, and SVM can be used to handles high-dimension data.
④KNN possess complicated calculation, and SVM needs training process.

How to choose to use both?
1. Choose KNN scene:
- A. Low accuracy
- B. Fewer samples.
- C. Samples cannot be obtained at once.

2. Select the SVM scene:
- A. Need to improve the accuracy rate.
- B. There are many samples.
- c. The sample is fixed and does not change with time.

---------- Dividing line ----------
This project requires the installation of a Python library. The factory image provided by us has been installed, and users do not need to install it again.
sudo pip install pypng
sudo pip install sklearn
sudo pip install sklearn
sudo apt-get install python-matplotlib
---------- Dividing line ----------

First, we need to pre-process the MNIST data and convert it to a png image.
Code path:
/home/pi/Yahboom_Project/Raspbot/1.OpenCV_course/05machine_learning/03SVM/03_1.MNIST.ipynb

Note:
In the image provided by us, the following code has been run, and the corresponding file has been generated.
If you need to run this code again, you need to delete the two files MSIST_data/test and train in this directory. Then, run the following code.

```
import struct
from array import array
import os
# Input pip install pypng command install this library
import png

trainimg = './MNIST_data/train-images.idx3-ubyte'
trainlabel = './MNIST_data/train-labels.idx1-ubyte'
testimg = './MNIST_data/t10k-images.idx3-ubyte'
testlabel = './MNIST_data/t10k-labels.idx1-ubyte'
trainfolder = './MNIST_data/train'
testfolder = './MNIST_data/test'
if not os.path.exists(trainfolder): os.makedirs(trainfolder)
if not os.path.exists(testfolder): os.makedirs(testfolder)

# open(File path, read-write format), used to open a file and return a file object
# rb means open the file with binary read mode
trimg = open(trainimg, 'rb')
teimg = open(testimg, 'rb')
trlab = open(trainlabel, 'rb')
telab = open(testlabel, 'rb')

# struct
struct.unpack(">IIII", trimg.read(16))
struct.unpack(">IIII", teimg.read(16))
struct.unpack(">II", trlab.read(8))
struct.unpack(">II", telab.read(8))
# The array module is an efficient array storage type implemented in Python
# All array members must be of the same type
# B unsigned byte type, b signed byte type
trimage = array("B", trimg.read())
teimage = array("B", teimg.read())
trlabel = array("b", trlab.read())
```

```
telabel = array("b", telab.read())
# The close method is used to close an opened file. After closing, the file cannot be read or written.
trimg.close()
teimg.close()
trlab.close()
telab.close()
# Define 10 subfolders for the training set and the test set, used to store all the numbers from 0 to 9, the folder names are 0-9
trainfolders = [os.path.join(trainfolder, str(i)) for i in range(10)]
testfolders = [os.path.join(testfolder, str(i)) for i in range(10)]
for dir in trainfolders:
    if not os.path.exists(dir):
        os.makedirs(dir)
for dir in testfolders:
    if not os.path.exists(dir):
        os.makedirs(dir)
# Start saving training image data
for (i, label) in enumerate(trlabel):
    filename = os.path.join(trainfolders[label], str(i) + ".png")
    #print("writing " + filename)
    with open(filename, "wb") as img:
        image = png.Writer(28, 28, greyscale=True)
        data = [trimage[(i*28*28 + j*28) : (i*28*28 + (j+1)*28)] for j in range(28)]
        image.write(img, data)
print("end write train image")

# Start saving testing image data
for (i, label) in enumerate(telabel):
    filename = os.path.join(testfolders[label], str(i) + ".png")
    #print("writing " + filename)
    with open(filename, "wb") as img:
        image = png.Writer(28, 28, greyscale=True)
        data = [teimage[(i*28*28 + j*28) : (i*28*28 + (j+1)*28)] for j in range(28)]
        image.write(img, data)
print("end write test image")
```

Code path:
/home/pi/Yahboom_Project/1.OpenCV_course/05machine_learning/03SVM/03_2.SVM_training model.ipynb

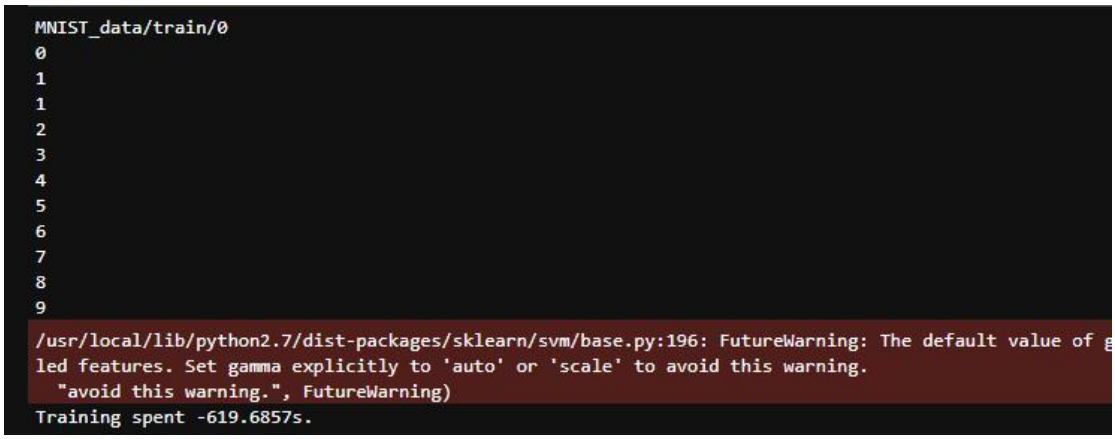
```
from PIL import Image
import os
import sys
```

```
import numpy as np
import time
from sklearn import svm
from sklearn.externals import joblib

# Get all .png files in the specified path
def get_file_list(path):
    return [os.path.join(path, f) for f in os.listdir(path) if f.endswith(".png")]

# Parse the name of the .png image file
def get_img_name_str(imgPath):
    return imgPath.split(os.path.sep)[-1]

# Convert 28px*28px image data to 1*784 numpy vector
# Parameters: imgFile--image name such as: 1.png
# Return: 1*784 numpy vector
def img2vector(imgFile):
    # print("in img2vector func--para:{}".format(imgFile))
    img = Image.open(imgFile).convert('L')
    img_arr = np.array(img, 'i')    # 28px*28px grayscale image
    img_normalization = np.round(img_arr / 255)    #Normalize the gray value
    img_arr2 = np.reshape(img_normalization, (1, -1))    #1*784 matrix
    return img_arr2

# Read all data in a category and convert it into a matrix
# Parameters:
# basePath: The basic path where the image data is located
# MNIST-data/train/
# MNIST-data/test/
# cla: classification name
# 0,1,2,...,9
# Returns: all data of a certain category-[number of samples * (image width x image
height)] matrix
def read_and_convert(imgFileList):
    dataLabel = []    # Storage class label
    dataNum = len(imgFileList)
    dataMat = np.zeros((dataNum, 784))    # dataNum*784 matrix
    for i in range(dataNum):
        imgNameStr = imgFileList[i]
        imgName = get_img_name_str(imgNameStr)    # Get the number of the
current number.png
        # print("imgName: {}".format(imgName))
        classTag = imgNameStr.split(os.path.sep)[-2]
        # classTag = imgName.split(".")[0].split("_")[0]    # Get class label (number)
```

```
            #print(classTag)
            #print(imgNameStr)
            dataLabel.append(classTag)
            dataMat[i, :] = img2vector(imgNameStr)
        return dataMat, dataLabel


# Read training data
def read_all_data():
        cName = ['1', '2', '3', '4', '5', '6', '7', '8', '9']
        #path = sys.path[1]
        train_data_path = 'MNIST_data/train/0' # os.path.join(path,
'./MNIST_data/train/0')
        print(train_data_path)
        #train_data_path = "./MNIST_data/train/0"
        print('0')
        flist = get_file_list(train_data_path)
        #print(flist)
        dataMat, dataLabel = read_and_convert(flist)
        for c in cName:
            print(c)
            #train_data_path = os.path.join(path, './MNIST_data/train/') + c
            train_data_path = 'MNIST_data/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

'''
SVC parameter
svm.SVC(C=1.0,kernel='rbf',degree=3,gamma='auto',coef0=0.0,shrinking=True,proba
bility=False,
tol=0.001,cache_size=200,class_weight=None,verbose=False,max_iter=-1,decision_f
unction_shape='ovr',random_state=None)
C: C-SVC penalty parameter C, the default value is 1.0
kernel: Kernel function, default is rbf, it can be set to ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’,
‘precomputed’
   0-linear: u'v
   1-Polynomial: (gamma * u '* v + coef0) ^ degree
   2-RBF function: exp (-gamma | u-v | ^ 2)
   3-sigmoid: tanh (gamma * u '* v + coef0)
```

```
degree: The dimension of the polynomial poly function, which is 3 by default. It will be ignored when other kernel functions are selected. (Useless)
gamma: Kernel function parameters of 'rbf', 'poly' and 'sigmoid'. The default is 'auto', then 1/n_features will be selected
coef0: constant term of the kernel function. Useful for 'poly' and 'sigmoid'. (Useless)
probability: whether to use probability estimation, the default is False
shrinking: Whether to use shrinking heuristic method, the default is true
tol: the size of the error value for stopping training, the default is 1e-3
cache_size: The cache size of the core function cache, the default is 200
class_weight: The weight of the category, passed in the form of a dictionary. Set the parameter C of the first category to weight *C (C in C-SVC)
verbose: Allow redundant output.
max_iter: Maximum number of iterations. -1 is unlimited.
decision_function_shape: 'ovo', 'ovr' or None, default = None3 (select ovr, one-to-many)
random_state: seed value, int value when the data is shuffled
The main adjustment parameters are: C, kernel, degree, gamma, coef0
'''
# Create a model
def create_svm(dataMat, dataLabel,path,decision='ovr'):
    clf = svm.SVC(C=1.0,kernel='rbf',decision_function_shape=decision)
    rf =clf.fit(dataMat, dataLabel)
    joblib.dump(rf, path)
    return clf

if __name__ == '__main__':
    # clf = svm.SVC(decision_function_shape='ovr')
    st = time.clock()
    dataMat, dataLabel = read_all_data()
    #path = sys.path[1]
    #model_path=os.path.join(path,'model\\svm.model')
    model_path = 'model/svm.model'
    create_svm(dataMat, dataLabel, model_path, decision='ovr')
    et = time.clock()
    print("Training spent {:.4f}s.".format((et - st)))
```

This training process takes a long time, about four hours, please wait patiently until the time spent is printed.
The following time range exceeds the maximum value of the variable and will be printed as a negative value, ignoring the warning.



This code for testing the MNIST data set.
Code path:
/home/pi/Yahboom_Project/1.OpenCV_course/05machine_learning/03SVM/03_3.SVM_MNIST.ipynb

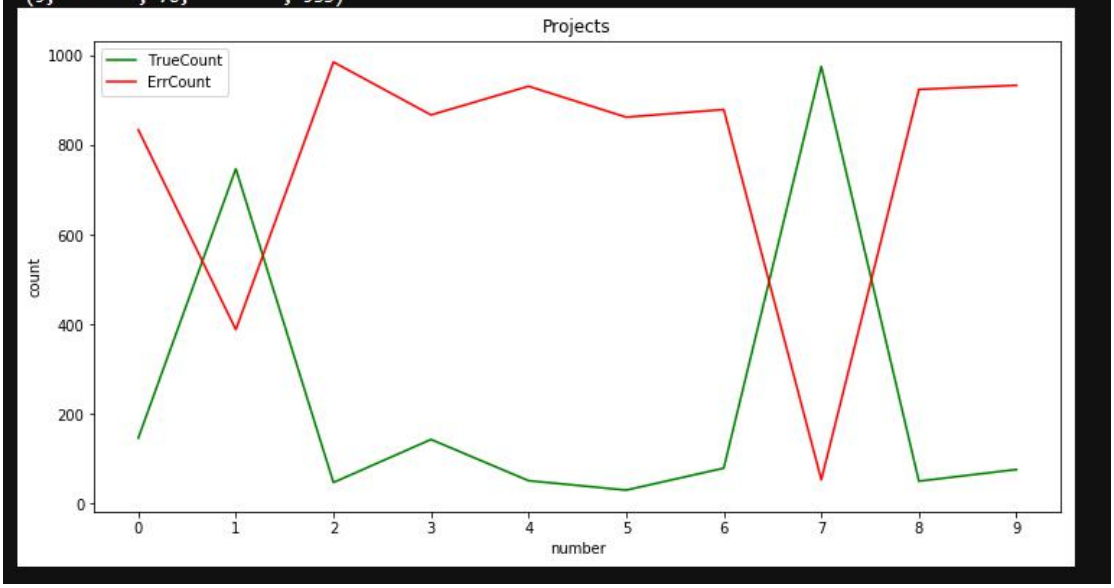
```
# Test handwritten digits of MNIST data
import sys
import time
import svm
import os
from sklearn.externals import joblib
import sys
import time
from PIL import Image
import os
from sklearn.externals import joblib
import numpy as np
import matplotlib.pyplot as plt

# Get all .png files in the specified path
def get_file_list(path):
    return [os.path.join(path, f) for f in os.listdir(path) if f.endswith(".png")]

# Parse the name of the .png image file
def get_img_name_str(imgPath):
    return imgPath.split(os.path.sep)[-1]

# Convert 28px*28px image data to 1*784 numpy vector
# Parameter: imgFile--image name eg: 0_1.png
# Return: 1*784 numpy vector
def img2vector(imgFile):
```

```
    # print("in img2vector func--para:{}".format(imgFile))
    img = Image.open(imgFile).convert('L')
    img_arr = np.array(img, 'i')    # 28px*28px grayscale image
    img_normalization = np.round(img_arr / 255)    # Normalize the gray value
    img_arr2 = np.reshape(img_normalization, (1, -1))    #1*400 matrix
    return img_arr2

# Read all data of a category and convert it into a matrix
# Parameter:
#     basePath: the base path where the image data is located
#     MNIST-data / train /
#     MNIST-data / test /
#     cla: category name
#     0,1,2, ..., 9
# Return: all data of a certain category-[number of samples * (image width x image height)] matrix

def read_and_convert(imgFileList):
    dataLabel = []
    dataNum = len(imgFileList)
    dataMat = np.zeros((dataNum, 784))    # dataNum*784 matrix
    for i in range(dataNum):
        imgNameStr = imgFileList[i]
        imgName = get_img_name_str(imgNameStr)
        # print("imgName: {}".format(imgName))
        classTag = imgNameStr.split(os.path.sep)[-2]
        # classTag = imgName.split(".")[0].split("_")[0]    # Get class label (number)
        #print(classTag)
        #print(imgNameStr)
        dataLabel.append(classTag)
        dataMat[i, :] = img2vector(imgNameStr)
    return dataMat, dataLabel

def svmtest(model_path):

    tbasePath = "MNIST_data/test/"
    tcName = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
    tst = time.clock()
    allErrCount = 0
    allErrorRate = 0.0
    allScore = 0.0
    ErrCount=np.zeros(10,int)
    TrueCount=np.zeros(10,int)
    # Load the model
```

```
    clf = joblib.load(model_path)
    for tcn in tcName:
        testPath = tbasePath + tcn
        # print("class " + tcn + " path is: {}.".format(testPath))
        tflist = get_file_list(testPath)
        # tflist
        tdataMat, tdataLabel = read_and_convert(tflist)
        print("test dataMat shape: {0}, test dataLabel len: {1}
".format(tdataMat.shape, len(tdataLabel)))
        # print("test dataLabel: {}".format(len(tdataLabel)))
        pre_st = time.clock()
        preResult = clf.predict(tdataMat)
        pre_et = time.clock()
        print("Recognition    " + tcn + " spent {:.4f}s.".format((pre_et - pre_st)))
        # print("predict result: {}".format(len(preResult)))
        errCount = len([x for x in preResult if x != tcn])
        ErrCount[int(tcn)]=errCount
        TrueCount[int(tcn)]= len(tdataLabel)-errCount
        print("errorCount: {}.".format(errCount))
        allErrCount += errCount
        score_st = time.clock()
        score = clf.score(tdataMat, tdataLabel)
        score_et = time.clock()
        print("computing score spent {:.6f}s.".format(score_et - score_st))
        allScore += score
        print("score: {:.6f}.".format(score))
        print("error rate is {:.6f}.".format((1 - score)))

    tet = time.clock()
    print("Testing All class total spent {:.6f}s.".format(tet - tst))
    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))
    print("number"," TrueCount"," ErrCount")
    for tcn in tcName:
        tcn=int(tcn)
        print(tcn,"         ",TrueCount[tcn],"          ",ErrCount[tcn])
    plt.figure(figsize=(12, 6))
    x=list(range(10))
    plt.plot(x,TrueCount, color='green', label="TrueCount")    # Mark right s as green
    plt.plot(x,ErrCount, color='red', label="ErrCount") #Mark errors as red
plt.legend(loc='best')    #Show the position of the legend, here is the lower right
    plt.title('Projects')
```

```
        plt.xlabel('number')        # x-axis label
        plt.ylabel('count')         # y-axis label
        plt.xticks(np.arange(10), ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'])
        plt.show()


if __name__ == '__main__':
        model_path='model/svm.model'
        svmtest(model_path)
```

```
All error Count is: 7656.
Average accuracy is: 0.223739.
Average error rate is: 0.776261.
('number', ' TrueCount', ' ErrCount')
(0, '     ', 146, '      ', 834)
(1, '     ', 747, '      ', 388)
(2, '     ', 47, '      ', 985)
(3, '     ', 143, '      ', 867)
(4, '     ', 51, '      ', 931)
(5, '     ', 30, '      ', 862)
(6, '     ', 79, '      ', 879)
(7, '     ', 975, '      ', 53)
(8, '     ', 50, '      ', 924)
(9, '     ', 76, '      ', 933)
```



According to the above figure, we can see that only the number 7 has a high recognition rate.

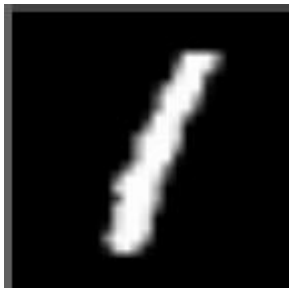
Code path:
/home/pi/Yahboom_Project/1.OpenCV_course/05machine_learning/03SVM/03_4.SVM.ipynb

```
from PIL import Image
import sys
import time
```

```
import os
from sklearn.externals import joblib
import numpy as np
import matplotlib.pyplot as plt

# Get all .png files in the specified path
def get_file_list(path):
    return [os.path.join(path, f) for f in os.listdir(path) if f.endswith(".png")]

# Parse the name of the .png image file
def get_img_name_str(imgPath):
    return imgPath.split(os.path.sep)[-1]

# Convert 28px*28px image data to 1*784 numpy vector
# Parameter: imgFile--image name eg: 0_1.png
# Return: 1*784 numpy vector
def img2vector(imgFile):
    # print("in img2vector func--para:{}".format(imgFile))
    img = Image.open(imgFile).convert('L')
    img_arr = np.array(img, 'i')    #28px*28px grayscale image
    img_normalization = np.round(img_arr / 255)    #Normalize the gray value
    img_arr2 = np.reshape(img_normalization, (1, -1))    # 1*400 matrix
    return img_arr2

# Read all data of a category and convert it into a matrix
# Parameter:
#     basePath: The basic path where the image data is located
#         MNIST-data/train/
#         MNIST-data/test/
#     cla: classification name
#         0,1,2,...,9
# Returns: all data of a certain category-[number of samples * (image width x image
height)] matrix
def read_and_convert(imgFileList):
    dataLabel = []    # Storage class label
    dataNum = len(imgFileList)
    dataMat = np.zeros((dataNum, 784))    # dataNum*784 matrix
    for i in range(dataNum):
        imgNameStr = imgFileList[i]
        imgName = get_img_name_str(imgNameStr)
        # print("imgName: {}".format(imgName))
        classTag = imgNameStr.split(os.path.sep)[-2]
        # classTag = imgName.split(".")[0].split("_")[0]    # Get class label (number)
        #print(classTag)
```

```
            #print(imgNameStr)
            dataLabel.append(classTag)
            dataMat[i, :] = img2vector(imgNameStr)
    return dataMat, dataLabel

def svmtest(model_path):
    # Picture path
    tbasePath = "image/"
    # Load the model
    clf = joblib.load(model_path)
    # Get file list
    tflist = get_file_list(tbasePath)
    # tflist
    tdataMat, tdataLabel = read_and_convert(tflist)
    print("test dataMat shape: {0}, test dataLabel len: {1} ".format(tdataMat.shape,
len(tdataLabel)))
    pre_st = time.clock()
    # forecast result
    preResult = clf.predict(tdataMat)
    pre_et = time.clock()
    print("Recognition  1 spent {:.4f}s.".format((pre_et - pre_st)))
    print("predict result: {}".format(len(preResult)))
    score = clf.score(tdataMat, tdataLabel)

if __name__ == '__main__':
    model_path='model/svm.model'
    svmtest(model_path)
```

Original picture：



Result, as shown below.

```
test dataMat shape: (1, 784), test dataLabel len: 1
Recognition  1 spent 0.1134s.
predict result: 1
```