2.SVM

2.1. Introduction to SVM

Support Vector Machine (SVM) is a generalized linear classifier that performs binary classification of data in a supervised learning manner. Its basic model is a linear classifier with the largest margin defined in the feature space. The largest margin distinguishes it from the perceptron. SVM also includes the kernel technique, which makes it a nonlinear classifier in essence. The learning strategy of SVM is to maximize the margin, which can be formalized as a problem of solving convex quadratic programming, which is also equivalent to the problem of minimizing the regularized hinge loss function. The learning algorithm of SVM is the optimization algorithm for solving convex quadratic programming.

Advantages: Low generalization error rate, low computational overhead, and easy to interpret results

Disadvantages: Sensitive to parameter adjustment and kernel function selection, the original classifier is only suitable for processing two-category problems without modification.

Applicable data types: numerical and nominal data

2.2. General process of SVM:

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

2.3 Comparison between KNN and SVM

Similarities: Both are relatively classic machine learning classification algorithms, both belong to supervised learning algorithms, and both have important theoretical basis for the selection of machine learning algorithms.

Difference:

1. KNN considers every sample. SVM is to find a function to make the samples separable.
2. Naive KNN will not self-learn feature weights, and the essence of SVM is to find weights.
3. KNN cannot handle things with too high sample dimensions, and SVM is better at handling high-dimensional data.
4. KNN has high computational complexity, but the parameters that need to be adjusted are relatively small. SVM requires a training process and has high prediction efficiency.

How to choose between the two?

1. Scenarios for choosing KNN:

a. Accuracy does not need to be refined.

b. There are not many samples.

c. Samples cannot be obtained at one time. Intelligence is obtained one by one over time.

2. Scenarios for choosing SVM:

a. The accuracy needs to be improved.

b. There are many samples.

c. The samples are fixed and will not change over time.

Note: The following code has been run in the factory image and the corresponding files have been generated. If you need to run it again, you can delete the two directories MSIST_data/test and train in this directory and execute the following code.

Preprocess MNIST data and convert it into png images

Source code path:

/home/pi/project_demo/06.Open_source_cv_fundamentals_course/E.Machine_Learning/03_SVM_Digit_Recog/03_1.MNIST_Preproc.ipynb

```
from PIL import Image
import os
import sys
import numpy as np
import time
from sklearn import svm
import joblib

# 获取指定路径下的所有 .png 文件 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")]

# 解析出 .png 图片文件的名称 Parse the name of the .png image file
def get_img_name_str(imgPath):
    return imgPath.split(os.path.sep)[-1]


# 将 28px * 28px 的图像数据转换成 1*784 的 numpy 向量 Convert 28px * 28px image data
into 1*784 numpy vector
# 参数: imgFile--图像名  如: 1.png Parameter: imgFile--image name, such as: 1.png
# 返回: 1*784 的 numpy 向量 Returns: 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 灰度图像 28px * 28px grayscale
image
    img_normalization = np.round(img_arr / 255)  # 对灰度值进行归一化 Normalize the
grayscale value
```

```
    img_arr2 = np.reshape(img_normalization, (1, -1))  # 1 * 784 矩阵 1 * 784
matrix
    return img_arr2


# 读取一个类别的所有数据并转换成矩阵 Read all the data of a category and convert it
into a matrix
# 参数: parameter:
#    basePath: 图像数据所在的基本路径 basePath: The base path where the image data
is located
#       MNIST-data/train/
#       MNIST-data/test/
#    cla: 类别名称 cla: Category name
#       0,1,2,...,9
# 返回: 某一类别的所有数据----[样本数量*(图像宽x图像高)] 矩阵 Returns: All data of a
certain category----[sample number*(image width x image height)] matrix
def read_and_convert(imgFileList):
    dataLabel = []  # 存放类标签 Storage label
    dataNum = len(imgFileList)
    dataMat = np.zeros((dataNum, 784))  # dataNum * 784 的矩阵 Matrix of dataNum *
784
    for i in range(dataNum):
        imgNameStr = imgFileList[i]
        imgName = get_img_name_str(imgNameStr)  # 得到 当前数字的数字编号.png Get
the digital number of the current number.png
        # print("imgName: {}".format(imgName))
        classTag = imgNameStr.split(os.path.sep)[-2]
        # classTag = imgName.split(".")[0].split("_")[0]  # 得到 类标签(数字) Get
class label (number)
        #print(classTag)
        #print(imgNameStr)
        dataLabel.append(classTag)
        dataMat[i, :] = img2vector(imgNameStr)
    return dataMat, dataLabel


# 读取训练数据 Reading 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参数
svm.SVC(C=1.0,kernel='rbf',degree=3,gamma='auto',coef0=0.0,shrinking=True,probab
ility=False,
tol=0.001,cache_size=200,class_weight=None,verbose=False,max_iter=-1,decision_fu
nction_shape='ovr',random_state=None)

C: C-SVC的惩罚参数C?默认值是1.0
C越大，相当于惩罚松弛变量，希望松弛变量接近0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，
这样对训练集测试时
准确率很高，但泛化能力弱。C值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。

kernel ：核函数，默认是rbf，可以是‘linear’，‘poly’，‘rbf’，‘sigmoid’，‘precomputed’
       0 – 线性：u'v
     1 – 多项式：(gamma*u'*v + coef0)^degree
     2 – RBF函数：exp(-gamma|u-v|^2)
     3 –sigmoid：tanh(gamma*u'*v + coef0)

degree ：多项式poly函数的维度，默认是3，选择其他核函数时会被忽略。（没用）

gamma ： ‘rbf’,‘poly’ 和‘sigmoid’的核函数参数。默认是’auto’，则会选择1/n_features

coef0 ：核函数的常数项。对于‘poly’和 ‘sigmoid’有用。（没用）

probability ：是否采用概率估计？.默认为False

shrinking ：是否采用shrinking heuristic方法，默认为true

tol ：停止训练的误差值大小，默认为1e-3

cache_size ：核函数cache缓存大小，默认为200

class_weight ：类别的权重，字典形式传递。设置第几类的参数C为weight*C(C-SVC中的C)

verbose ：允许冗余输出？

max_iter ：最大迭代次数。-1为无限制。

decision_function_shape ：‘ovo’，‘ovr’ or None，default=None3（选用ovr，一对多）

random_state ：数据洗牌时的种子值，int值

主要调节的参数有：C、kernel、degree、gamma、coef0

SVC parameters
svm.SVC(C=1.0,kernel='rbf',degree=3,gamma='auto',coef0=0.0,shrinking=True,probab
ility=False,
tol=0.001,cache_size=200,class_weight=None,verbose=False,max_iter=-1,decision_fu
nction_shape='ovr',random_state=None)
```

```
C: C-SVC penalty parameter C? The default value is 1.0

The larger the C value, the more it is equivalent to penalizing the slack
variable. We hope that the slack variable is close to 0, that is, the penalty for
misclassification increases, tending to the situation where all training sets are
classified correctly. In this way, when testing the training set, the accuracy is
very high, but the generalization ability is weak. The smaller the C value, the
smaller the penalty for misclassification, allowing fault tolerance, treating
them as noise points, and the generalization ability is stronger.

kernel ：kernel function, default is rbf, can be ‘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 ：dimension of polynomial poly function, default is 3, ignored when other
kernel functions are selected. (Not used)

gamma ： kernel function parameter for ‘rbf’, ‘poly’ and ‘sigmoid’. Default is
‘auto’, which selects 1/n_features

coef0 ：constant term of kernel function. Useful for ‘poly’ and ‘sigmoid’. (Not
used)

probability ：Whether to use probability estimation? .Default is False

shrinking: Whether to use the shrinking heuristic method, default is true

tol: The error value for stopping training, default is 1e-3

cache_size: The kernel function cache size, default is 200

class_weight: The weight of the category, passed in dictionary form. Set the
parameter C of the category to weight*C (C in C-SVC)

verbose: Allow redundant output?

max_iter: Maximum number of iterations. -1 means unlimited.

decision_function_shape: ‘ovo’, ‘ovr’ or None, default=None3 (select ovr, one-to-
many)

random_state: Seed value when shuffling data, int value

The main parameters to adjust are: C, kernel, degree, gamma, coef0
'''
# 创建模型 Creating the 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.process_time()
    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.process_time()
    print("Training spent {:.4f}s.".format((et - st)))
```

Training

This training process takes a long time, about four hours. Please wait patiently until the time spent is printed. During this period, the time scale is printed as a negative value due to exceeding the maximum value of the variable. No need to worry about the warning

Source code path:

/home/pi/project_demo/06.Open_source_cv_fundamentals_course/E.Machine_Learning/03_SVM_Digit_Recog/03_2.SVM_Digits.ipynb

```
from PIL import Image
import os
import sys
import numpy as np
import time
from sklearn import svm
import joblib

# 获取指定路径下的所有 .png 文件 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")]

# 解析出 .png 图片文件的名称 Parse the name of the .png image file
def get_img_name_str(imgPath):
    return imgPath.split(os.path.sep)[-1]


# 将 28px * 28px 的图像数据转换成 1*784 的 numpy 向量 Convert 28px * 28px image data
into 1*784 numpy vector
# 参数: imgFile--图像名  如: 1.png Parameter: imgFile--image name, such as: 1.png
# 返回: 1*784 的 numpy 向量 Returns: 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 灰度图像 28px * 28px grayscale
image
    img_normalization = np.round(img_arr / 255)  # 对灰度值进行归一化 Normalize the
grayscale value
    img_arr2 = np.reshape(img_normalization, (1, -1))  # 1 * 784 矩阵 1 * 784
matrix
```

```
    return img_arr2


# 读取一个类别的所有数据并转换成矩阵 Read all the data of a category and convert it into a matrix
# 参数: parameter:
#    basePath: 图像数据所在的基本路径 basePath: The base path where the image data is located
#       MNIST-data/train/
#       MNIST-data/test/
#    cla: 类别名称 cla: Category name
#       0,1,2,...,9
# 返回: 某一类别的所有数据----[样本数量*(图像宽x图像高)] 矩阵 Returns: All data of a certain category----[sample number*(image width x image height)] matrix
def read_and_convert(imgFileList):
    dataLabel = []  # 存放类标签 Storage label
    dataNum = len(imgFileList)
    dataMat = np.zeros((dataNum, 784))  # dataNum * 784 的矩阵 Matrix of dataNum * 784
    for i in range(dataNum):
        imgNameStr = imgFileList[i]
        imgName = get_img_name_str(imgNameStr)  # 得到 当前数字的数字编号.png Get the digital number of the current number.png
        # print("imgName: {}".format(imgName))
        classTag = imgNameStr.split(os.path.sep)[-2]
        # classTag = imgName.split(".")[0].split("_")[0]  # 得到 类标签(数字) Get class label (number)
        #print(classTag)
        #print(imgNameStr)
        dataLabel.append(classTag)
        dataMat[i, :] = img2vector(imgNameStr)
    return dataMat, dataLabel


# 读取训练数据 Reading 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参数
svm.SVC(C=1.0,kernel='rbf',degree=3,gamma='auto',coef0=0.0,shrinking=True,probab
ility=False,
tol=0.001,cache_size=200,class_weight=None,verbose=False,max_iter=-1,decision_fu
nction_shape='ovr',random_state=None)

C: C-SVC的惩罚参数C?默认值是1.0
C越大，相当于惩罚松弛变量，希望松弛变量接近0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，
这样对训练集测试时
准确率很高，但泛化能力弱。C值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。

kernel ：核函数，默认是rbf，可以是‘linear’，‘poly’，‘rbf’，‘sigmoid’，‘precomputed’
       0 – 线性：u'v
     1 – 多项式：(gamma*u'*v + coef0)^degree
     2 – RBF函数：exp(-gamma|u-v|^2)
     3 –sigmoid：tanh(gamma*u'*v + coef0)

degree ：多项式poly函数的维度，默认是3，选择其他核函数时会被忽略。（没用）

gamma ：  ‘rbf’,‘poly’ 和‘sigmoid’的核函数参数。默认是’auto’，则会选择1/n_features

coef0 ：核函数的常数项。对于‘poly’和 ‘sigmoid’有用。（没用）

probability ：是否采用概率估计？.默认为False

shrinking ：是否采用shrinking heuristic方法，默认为true

tol ：停止训练的误差值大小，默认为1e-3

cache_size ：核函数cache缓存大小，默认为200

class_weight ：类别的权重，字典形式传递。设置第几类的参数C为weight*C(C-SVC中的C)

verbose ：允许冗余输出?

max_iter ：最大迭代次数。-1为无限制。

decision_function_shape ：‘ovo’，‘ovr’ or None，default=None3（选用ovr，一对多）

random_state ：数据洗牌时的种子值，int值

主要调节的参数有：C、kernel、degree、gamma、coef0

SVC parameters
svm.SVC(C=1.0,kernel='rbf',degree=3,gamma='auto',coef0=0.0,shrinking=True,probab
ility=False,
tol=0.001,cache_size=200,class_weight=None,verbose=False,max_iter=-1,decision_fu
nction_shape='ovr',random_state=None)

C: C-SVC penalty parameter C? The default value is 1.0
```

```
The larger the C value, the more it is equivalent to penalizing the slack variable. We hope that the slack variable is close to 0, that is, the penalty for misclassification increases, tending to the situation where all training sets are classified correctly. In this way, when testing the training set, the accuracy is very high, but the generalization ability is weak. The smaller the C value, the smaller the penalty for misclassification, allowing fault tolerance, treating them as noise points, and the generalization ability is stronger.

kernel : kernel function, default is rbf, can be '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 : dimension of polynomial poly function, default is 3, ignored when other kernel functions are selected. (Not used)

gamma :  kernel function parameter for 'rbf', 'poly' and 'sigmoid'. Default is 'auto', which selects 1/n_features

coef0 : constant term of kernel function. Useful for 'poly' and 'sigmoid'. (Not used)

probability : Whether to use probability estimation? .Default is False

shrinking: Whether to use the shrinking heuristic method, default is true

tol: The error value for stopping training, default is 1e-3

cache_size: The kernel function cache size, default is 200

class_weight: The weight of the category, passed in dictionary form. Set the parameter C of the category to weight*C (C in C-SVC)

verbose: Allow redundant output?

max_iter: Maximum number of iterations. -1 means unlimited.

decision_function_shape: 'ovo', 'ovr' or None, default=None3 (select ovr, one-to-many)

random_state: Seed value when shuffling data, int value

The main parameters to adjust are: C, kernel, degree, gamma, coef0
'''
# 创建模型 Creating the 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.process_time()
    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.process_time()
    print("Training spent {:.4f}s.".format((et - st)))
```

Test MNIST dataset

Source code path:

/home/pi/project_demo/06.Open_source_cv_fundamentals_course/E.Machine_Learning/03_SVM_Digit_Recog/03_3.SVM_MNIST.ipynb

```
import sys
import time
from PIL import Image
import os
import joblib
import numpy as np
import matplotlib.pyplot as plt
#忽略警告 Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# 获取指定路径下的所有 .png 文件 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")]

# 解析出 .png 图片文件的名称 Parse the name of the .png image file
def get_img_name_str(imgPath):
    return imgPath.split(os.path.sep)[-1]

# 将 28px * 28px 的图像数据转换成 1*784 的 numpy 向量 Convert 28px * 28px image data
into 1*784 numpy vector
# 参数: imgFile--图像名  如: 0_1.png Parameter: imgFile--image name, such as:
0_1.png
# 返回: 1*784 的 numpy 向量 Returns: 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 灰度图像 28px * 28px grayscale
image
    img_normalization = np.round(img_arr / 255)  # 对灰度值进行归一化 Normalize the
grayscale value
    img_arr2 = np.reshape(img_normalization, (1, -1))  # 1 * 400 矩阵 1 * 400
matrix
    return img_arr2
```

```
# 读取一个类别的所有数据并转换成矩阵 Read all the data of a category and convert it into a matrix
# 参数: parameter:
#    basePath: 图像数据所在的基本路径 basePath: The base path where the image data is located
#       MNIST-data/train/
#       MNIST-data/test/
#    cla: 类别名称 cla: Category name
#       0,1,2,...,9
# 返回: 某一类别的所有数据----[样本数量*(图像宽x图像高)] 矩阵 Returns: All data of a certain category----[sample number*(image width x image height)] matrix
def read_and_convert(imgFileList):
    dataLabel = []  # 存放类标签 Storage label
    dataNum = len(imgFileList)
    dataMat = np.zeros((dataNum, 784))  # dataNum * 784 的矩阵 Matrix of dataNum * 784
    for i in range(dataNum):
        imgNameStr = imgFileList[i]
        imgName = get_img_name_str(imgNameStr)  # 得到 当前数字的数字编号.png Get the digital number of the current number.png
        # print("imgName: {}".format(imgName))
        classTag = imgNameStr.split(os.path.sep)[-2]
        # classTag = imgName.split(".")[0].split("_")[0]  # 得到 类标签(数字) 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.perf_counter()
    allErrCount = 0
    allErrorRate = 0.0
    allScore = 0.0
    ErrCount=np.zeros(10,int)
    TrueCount=np.zeros(10,int)
    #加载模型 Loading 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.perf_counter()
        preResult = clf.predict(tdataMat)
        pre_et = time.perf_counter()
```

```
        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.perf_counter()
        score = clf.score(tdataMat, tdataLabel)
        score_et = time.perf_counter()
        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 = perf_counter()
    tet = time.process_time()
    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")  # 将正确的数量设置为绿
色 Set the correct amount to green
    plt.plot(x,ErrCount, color='red', label="ErrCount")      # 将错误的数量为红色
The wrong number is colored red
    plt.legend(loc='best')  # 显示图例的位置，这里为右下方 Displays the location of
the legend, here is the lower right
    plt.title('Projects')
    plt.xlabel('number')    # x轴标签 x-axis label
    plt.ylabel('count')     # y轴标签 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)
```

Recognize your own handwritten numbers

Source code path:

/home/pi/project_demo/06.Open_source_cv_fundamentals_course/E.Machine_Learning/03_SVM_Digit_Recog/03_4.SVM_Personal.ipynb

```
from PIL import Image
import sys
import time
import os
import joblib
import numpy as np
import matplotlib.pyplot as plt

# 获取指定路径下的所有 .png 文件 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")]

# 解析出 .png 图片文件的名称 Parse the name of the .png image file
def get_img_name_str(imgPath):
    return imgPath.split(os.path.sep)[-1]


# 将 28px * 28px 的图像数据转换成 1*784 的 numpy 向量 Convert 28px * 28px image data
into 1*784 numpy vector
# 参数: imgFile--图像名  如: 0_1.png Parameter: imgFile--image name, such as:
0_1.png
# 返回: 1*784 的 numpy 向量 Returns: 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 灰度图像 28px * 28px grayscale
image
    img_normalization = np.round(img_arr / 255)  # 对灰度值进行归一化 Normalize the
grayscale value
    img_arr2 = np.reshape(img_normalization, (1, -1))  # 1 * 400 矩阵 1 * 400
matrix
    return img_arr2


# 读取一个类别的所有数据并转换成矩阵 Read all the data of a category and convert it
into a matrix
# 参数:  parameter:
#    basePath: 图像数据所在的基本路径 basePath: The base path where the image data
is located
#       MNIST-data/train/
#       MNIST-data/test/
#    cla: 类别名称 cla: Category name
#       0,1,2,...,9
# 返回: 某一类别的所有数据----[样本数量*(图像宽x图像高)] 矩阵 Returns: All data of a
certain category----[sample number*(image width x image height)] matrix
def read_and_convert(imgFileList):
    dataLabel = []  # 存放类标签 Storage label
    dataNum = len(imgFileList)
    dataMat = np.zeros((dataNum, 784))  # dataNum * 784 的矩阵 Matrix of dataNum *
784
    for i in range(dataNum):
        imgNameStr = imgFileList[i]
        imgName = get_img_name_str(imgNameStr)  # 得到 当前数字的数字编号.png Get
the digital number of the current number.png
```

```
        # print("imgName: {}".format(imgName))
        classTag = imgNameStr.split(os.path.sep)[-2]
        # classTag = imgName.split(".")[0].split("_")[0]  # 得到 类标签(数字) Get
class label (number)
        #print(classTag)
        #print(imgNameStr)
        dataLabel.append(classTag)
        dataMat[i, :] = img2vector(imgNameStr)
    return dataMat, dataLabel

def svmtest(model_path):

    #图片路径 Image Path
    tbasePath = "image/"

    #加载模型 Loading the model
    clf = joblib.load(model_path)
    #获取文件列表 Get a list of files
    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.perf_counter()
    #预测结果 forecast result
    preResult = clf.predict(tdataMat)
    pre_et = time.perf_counter()
    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 image:

Recognition results:

03_1.MNIST_Preproc.ipynb | 03_2.SVM_Digits.ipynb | 03_3.SVM_MNIST.ipynb | 03_4.SVM_Personal.ipynb

Code

```
    #获取文件列表 Get a list of files
    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.perf_counter()
    #预测结果 forecast result
    preResult = clf.predict(tdataMat)
    pre_et = time.perf_counter()
    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)
```

```
test dataMat shape: (1, 784), test dataLabel len: 1
Recognition  1 spent 0.0157s.
predict result: 1
```