svm.py

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import sys
import os
import time
import random
import scipy
import cv2

from PIL import Image
from sklearn import datasets, linear_model, preprocessing
from sklearn import svm
from sklearn.externals import joblib
from scipy import signal

X_LENGTH = 1280
SVCs = {}
TRAINED = 0
CLASS_NUM = {'letter':24, 'province':6, 'char':34}
POSITION = ['province','letter','char','char','char','char','char']

#----------------------------------------------Scan
Folders-------------------------------------------------
def getFilesName(nclass, TYPE, s):

    input_count = 0
    filenames = []
    types = []
    for i in range(nclass):
        dir = './%s/%s/%s/' % (s,TYPE,i)
        for rt, dirs, files in os.walk(dir):
            for filename in files:
                input_count += 1
                filenames.append(dir+filename)
                types.append(i)
    return input_count,filenames, types

#----------------------------------------------Read Single
Image----------------------------------------------
def getImage(fn):
    global X_LENGTH
    img = Image.open(fn)
    v = np.zeros((1, X_LENGTH))
    width = img.size[0]
    height = img.size[1]
    for h in range(0, height):
        for w in range(0, width):
            #Threshold into binary value
            if img.getpixel((w, h)) < 127:
                v[0,w+h*width] = 0
            else:
                v[0,w+h*width] = 1
    return v

#----------------------------------------------Training Single
Word----------------------------------------------
def training(s):

    if os.path.exists("./%s.m"%s):
        svc = joblib.load("%s.m"%s)
        print("Loaded previous model.")
        return svc

    global X_LENGTH, CLASS_NUM
    input_count, filenames, types= getFilesName(CLASS_NUM[s],s,"train")
    X = np.zeros((input_count, X_LENGTH))
    y = np.zeros(input_count)
```

```
    for i in range(input_count):
        filename = filenames[i]
        X[i,:] = getImage(filename)
        y[i] = types[i]

    svc = svm.SVC(probability=True,  kernel="rbf", C=2.8, gamma=.0073,verbose=10)
    svc.fit(X, y)
    y_hat = svc.predict(X)
    acc = np.mean(y_hat == y)
    print("\n\nTraining Accuracy for %s: %.2f\n"%(s, acc))
    joblib.dump(svc, "%s.m"%s)
    return svc

#----------------------------------------------Testing Single
Word-----------------------------------------------
def testing(s):

    global X_LENGTH, CLASS_NUM
    input_count, filenames, types = getFilesName(CLASS_NUM[s],s,"test")
    Xtest = np.zeros((input_count, X_LENGTH))
    ytest = np.zeros(input_count)

    for i in range(input_count):
        filename = filenames[i]
        Xtest[i,:] = getImage(filename)
        ytest[i] = types[i]

    ytest_hat = SVCs[s].predict(Xtest)
    print("\nTesting Accuracy for %s: %.2f\n"%(s, np.mean(ytest_hat == ytest)))

#----------------------------------------------ClassNum to
Char-----------------------------------------------
def getChar(x, s):
    if s == "province":
        return chr(x+97)
    elif s == "letter":
        if x <= 7:
            return chr(x+65)
        elif x <= 12:
            return chr(x+66)
        else:
            return chr(x+67)
    else:
        if (x >= 10):
            return getChar(x-10, "letter")
        else:
            return chr(x+48)


#----------------------------------------------Predicting Single
Word-----------------------------------------------
def predict(X, s): #X is 1x1280
    global SVCs
    y = SVCs[s].predict(X)
    return getChar(int(y[0]), s)

#----------------------------------------------Predicting A
Plate-----------------------------------------------
def detect(s):

    global POSITION

    #Binary-valuing the plate:
    img_gray = cv2.imread("./%s"%s,0)
    #Reduce the noise:
    img_gray = cv2.GaussianBlur(img_gray,(3,3),0.1)

    img_thre = img_gray
    mid = (img_gray.min()+img_gray.max())/2
    cv2.threshold(img_gray, mid, 255, cv2.THRESH_BINARY_INV, img_thre)
    img = img_thre
```

```
h = img.shape[0]
w = img.shape[1]

#find blank columns:
white_num = []
white_max = 0
for i in range(w):
    white = 0
    for j in range(h):
        if img[j,i] == 255:
            white += 1
    white_num.append(white)
    white_max = max(white_max, white)
blank = []
for i in range(w):
    if (white_num[i]  > 0.95 * white_max):
        blank.append(True)
    else:
        blank.append(False)

#split index:
i = 0
num = 0
l = 0
x,y,d = [],[],[]
while (i < w):
    if blank[i]:
        i += 1
    else:
        j = i
        while (j<w)and(  (not blank[j])or
                         (j-i<3)  ):
            j += 1
        x.append(i)
        y.append(j)
        d.append(j-i)
        l += 1
        i = j
d = np.array(d)
while (l > 7):
    i = np.argmin(d)
    l1 = d[i-1] if i>0 else 100
    l2 = d[i+1] if i<l-1 else 100
    if l1 > l2:
        x[i+1] = x[i]
    else:
        if (i-1>=0)and(i<l):
            y[i-1] = y[i]
    if (i>=0)and(i<l):
        x.pop(i)
    if (i>=0)and(i<l):
        y.pop(i)
    np.delete(d,[i])
    l -= 1

#predict plate:
stri = ""
Xtest = np.zeros((1,X_LENGTH))
img = Image.fromarray(255-img)
width = img.size[0]
height = img.size[1]
for i in range(l):
    sub_img = img.crop((x[i],0,y[i]-1,40))
    w0 = sub_img.size[0]
    h0 = sub_img.size[1]
    if w0 < 32:
        bg = Image.new("L",(32,40),0)
        for h in range(h0):
            for w in range(w0):
                bg.putpixel((int(w+(32-w0)/2),h),sub_img.getpixel((w,h)))
        sub_img = bg
    sub_img = sub_img.resize((32,40),Image.ANTIALIAS)
```

```
        #sub_img.save("%d.png"%i,"png")
        w0 = sub_img.size[0]
        h0 = sub_img.size[1]
        for h in range(0, h0):
            for w in range(0, w0):
                if sub_img.getpixel((w, h)) < mid:
                    Xtest[0,w+h*w0] = 0
                else:
                    Xtest[0,w+h*w0] = 1
        stri += predict(Xtest, POSITION[i])

    return stri

#--------------------------------------------Predicting All
Plates-------------------------------------------
def test_plate():

    input_count = 0
    acc = 0
    filenames = []
    pred = []
    dir = './test_plate/'
    for rt, dirs, files in os.walk(dir):
        for filename in files:
            if filename[-4:] != ".png":
                continue
            input_count += 1
            ans = detect(dir+filename)
            print(filename+" "+ans)
            filenames.append(filename)
            pred.append(ans)
            if ans == filename[9:16]:
                acc += 1

    print("\nTesting Accuracy for plates: %.2f"%(acc/input_count))

#-----------------------------------------------------
Menu-----------------------------------------------------
def train_menu():
    global SVCs
    global TRAINED
    a = input(" _____________________________ \n"+
              "|_1_|_train_char_data_____|\n"+
              "|_2_|_train_letter_data___|\n"+
              "|_3_|_train_province_data_|\n"+
              "|_4_|_train_all_data______|\n"+
              "\nEnter:")
    if not(a in ["1","2","3","4"]):
        print("Illegal Input!")
        return

    a, b = int(a)-1, [1,2,4,7]
    if b[a]&1 != 0:
        SVCs["char"] = training("char")
        TRAINED |= 1
    if b[a]&2 != 0:
        SVCs["letter"] = training("letter")
        TRAINED |= 2
    if b[a]&4 != 0:
        SVCs["province"] = training("province")
        TRAINED |= 4


def testsingle_menu():
    global SVCs
    global TRAINED
    a = input(" _____________________________ \n"+
              "|_1_|_test_char_data_____|\n"+
              "|_2_|_test_letter_data___|\n"+
              "|_3_|_test_province_data_|\n"+
              "|_4_|_test_all_data______|\n"+
              "\nEnter:")
```

```
    if not(a in ["1","2","3","4"]):
        print("Illegal Input!")
        return

    a, b = int(a)-1, [1,2,4,7]
    if TRAINED&b[a] != b[a]:
        print("Haven't train data!")
        return
    if b[a]&1 != 0:
        testing("char")
    if b[a]&2 != 0:
        testing("letter")
    if b[a]&4 != 0:
        testing("province")


#-------------------------------------------------------
Main-------------------------------------------------------
a = ""
while (a != "5"):
    a = input(" _____________________________ \n"+
              "|_1_|_train_data_____________|\n"+
              "|_2_|_test_single_character_datas_|\n"+
              "|_3_|_test_plate_datas_____________|\n"+
              "|_4_|_show_____________________|\n"+
              "|_5_|_exit_____________________|\n"+
              "\nEnter:")
    if   a == "1":
        train_menu()
    elif a == "2":
        testsingle_menu()
    elif a == "3":
        if TRAINED != 7:
            print("Haven't train data!")
            continue
        test_plate()
    elif a == "4":
        if TRAINED != 7:
            print("Haven't train data!")
            continue
        s = input("Enter picture name:")
        ans = detect(s)
        print(ans)
```

Plate_Generator

```
"""
Generate training and test images.

"""


__all__ = (
    'generate_ims',
)


import itertools
import math
import os
import random
import sys

import cv2
import numpy

from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont

import common

FONT_DIR = "./fonts"
FONT_HEIGHT = 40  # Pixel size to which the chars are resized

#OUTPUT_SHAPE = (64, 128)
OUTPUT_SHAPE = (40, 225)


CHARS = common.CHARS + " "
ALLCHARS = common.CHARS + common.PROVINCES + " "


def make_char_ims(font_path, output_height):
    font_size = output_height * 4

    font = ImageFont.truetype(font_path, font_size)

    height0 = max(font.getsize(c)[1] for c in CHARS)


    for c in ALLCHARS:

        if c in common.PROVINCES:
            im = Image.open("province/%c.bmp"%c)
            width, height = im.size
            scale = float(output_height) / height
            im = im.resize((int(width * scale), output_height), Image.ANTIALIAS)
            yield c, numpy.array(im)[:, :, 0].astype(numpy.float32) / 255.
        else:
            width = font.getsize(c)[0]
            #height = height0
            height = font.getsize(c)[1]
            im = Image.new("RGBA", (width, height), (0, 0, 0))
            draw = ImageDraw.Draw(im)
            draw.text((0, 0), c, (255, 255, 255), font=font)
            scale = float(output_height) / height
            im = im.resize((int(width * scale), output_height), Image.ANTIALIAS)
            yield c, numpy.array(im)[:, :, 0].astype(numpy.float32) / 255.


def euler_to_mat(yaw, pitch, roll):
    # Rotate clockwise about the Y-axis
    c, s = math.cos(yaw), math.sin(yaw)
    M = numpy.matrix([[  c, 0.,  s],
```

```
                    [ 0., 1., 0.],
                    [ -s, 0.,  c]])

    # Rotate clockwise about the X-axis
    c, s = math.cos(pitch), math.sin(pitch)
    M = numpy.matrix([[ 1., 0., 0.],
                    [ 0.,  c, -s],
                    [ 0.,  s,  c]]) * M

    # Rotate clockwise about the Z-axis
    c, s = math.cos(roll), math.sin(roll)
    M = numpy.matrix([[  c, -s, 0.],
                    [  s,  c, 0.],
                    [ 0., 0., 1.]]) * M

    return M


def pick_colors():
    first = True
    while first or plate_color - text_color < 0.5:
        text_color = random.random()
        plate_color = random.random()
        #plate_color = 1.0
        if text_color > plate_color:
            text_color, plate_color = plate_color, text_color
        first = False
    return text_color, plate_color


def make_affine_transform(from_shape, to_shape,
                          min_scale, max_scale,
                          scale_variation=1.0,
                          rotation_variation=1.0,
                          translation_variation=1.0):
    out_of_bounds = False

    from_size = numpy.array([[from_shape[1], from_shape[0]]]).T
    to_size = numpy.array([[to_shape[1], to_shape[0]]]).T

    scale = random.uniform((min_scale + max_scale) * 0.5 -
                           (max_scale - min_scale) * 0.5 * scale_variation,
                           (min_scale + max_scale) * 0.5 +
                           (max_scale - min_scale) * 0.5 * scale_variation)
    if scale > max_scale or scale < min_scale:
        out_of_bounds = True
    roll = random.uniform(-0.3, 0.3) * rotation_variation
    pitch = random.uniform(-0.2, 0.2) * rotation_variation
    yaw = random.uniform(-1.2, 1.2) * rotation_variation

    # Compute a bounding box on the skewed input image (`from_shape`).
    M = euler_to_mat(yaw, pitch, roll)[:2, :2]
    h, w = from_shape
    corners = numpy.matrix([[-w, +w, -w, +w],
                            [-h, -h, +h, +h]]) * 0.5
    skewed_size = numpy.array(numpy.max(M * corners, axis=1) -
                              numpy.min(M * corners, axis=1))

    # Set the scale as large as possible such that the skewed and scaled shape
    # is less than or equal to the desired ratio in either dimension.
    scale *= numpy.min(to_size / skewed_size)

    # Set the translation such that the skewed and scaled image falls within
    # the output shape's bounds.
    trans = (numpy.random.random((2,1)) - 0.5) * translation_variation
    trans = ((2.0 * trans) ** 5.0) / 2.0
    if numpy.any(trans < -0.5) or numpy.any(trans > 0.5):
        out_of_bounds = True
    trans = (to_size - skewed_size * scale) * trans

    center_to = to_size / 2.
    center_from = from_size / 2.
```

```
    M = euler_to_mat(yaw, pitch, roll)[:2, :2]
    M *= scale
    M = numpy.hstack([M, trans + center_to - M * center_from])

    return M, out_of_bounds


def generate_code():

    return "{}{} {}{}{}{}{}{}".format(
        random.choice(common.PROVINCES),
        random.choice(common.LETTERS),
        random.choice(common.CHARS),
        random.choice(common.CHARS),
        random.choice(common.CHARS),
        random.choice(common.CHARS),
        random.choice(common.CHARS))


def rounded_rect(shape, radius):
    out = numpy.ones(shape)
    out[:radius, :radius] = 0.0
    out[-radius:, :radius] = 0.0
    out[:radius, -radius:] = 0.0
    out[-radius:, -radius:] = 0.0

    cv2.circle(out, (radius, radius), radius, 1.0, -1)
    cv2.circle(out, (radius, shape[0] - radius), radius, 1.0, -1)
    cv2.circle(out, (shape[1] - radius, radius), radius, 1.0, -1)
    cv2.circle(out, (shape[1] - radius, shape[0] - radius), radius, 1.0, -1)

    return out


def generate_plate(font_height, char_ims):
    #h_padding = random.uniform(0.2, 0.4) * font_height
    #v_padding = random.uniform(0.1, 0.3) * font_height
    h_padding, v_padding = 0, 0
    spacing = font_height * random.uniform(0.04, 0.06)
    radius = 1 + int(font_height * 0.1 * random.random())

    code = generate_code()
    text_width = sum(char_ims[c].shape[1] for c in code)
    text_width += (len(code) - 1) * spacing

    out_shape = (int(font_height + v_padding * 2),
                 int(text_width + h_padding * 2))

    text_color, plate_color = pick_colors()

    text_mask = numpy.zeros(out_shape)

    x = h_padding
    y = v_padding
    for c in code:
        char_im = char_ims[c]
        ix, iy = int(x), int(y)
        text_mask[iy:iy + char_im.shape[0], ix:ix + char_im.shape[1]] = char_im
        x += char_im.shape[1] + spacing

    plate = (numpy.ones(out_shape) * plate_color * (1. - text_mask) +
             numpy.ones(out_shape) * text_color * text_mask)

    return plate, rounded_rect(out_shape, radius), code.replace(" ", "")


def generate_bg(num_bg_images):
    found = False
    while not found:
        fname = "bgs/%d.jpg"%(random.randint(0, num_bg_images - 1))
        bg = cv2.imread(fname, cv2.IMREAD_GRAYSCALE) / 255.
```

```
            if (bg.shape[1] >= OUTPUT_SHAPE[1] and
                bg.shape[0] >= OUTPUT_SHAPE[0]):
                found = True

        x = random.randint(0, bg.shape[1] - OUTPUT_SHAPE[1])
        y = random.randint(0, bg.shape[0] - OUTPUT_SHAPE[0])
        bg = bg[y:y + OUTPUT_SHAPE[0], x:x + OUTPUT_SHAPE[1]]

        return bg


    def generate_im(char_ims, num_bg_images):
        bg = generate_bg(num_bg_images)

        plate, plate_mask, code = generate_plate(FONT_HEIGHT, char_ims)

        M, out_of_bounds = make_affine_transform(
                                from_shape=plate.shape,
                                to_shape=bg.shape,
                                min_scale=1.0,
                                max_scale=1.0,
                                rotation_variation=0.0,
                                scale_variation=1.0,
                                translation_variation=1.0)
        plate = cv2.warpAffine(plate, M, (bg.shape[1], bg.shape[0]))
        plate_mask = cv2.warpAffine(plate_mask, M, (bg.shape[1], bg.shape[0]))

        out = plate * plate_mask + bg * (1.0 - plate_mask)

        out = cv2.resize(out, (OUTPUT_SHAPE[1], OUTPUT_SHAPE[0]))

        out += numpy.random.normal(scale=0.05, size=out.shape)
        out = numpy.clip(out, 0., 1.)

        return out, code, not out_of_bounds


    def load_fonts(folder_path):
        font_char_ims = {}
        fonts = [f for f in os.listdir(folder_path) if f.endswith('.ttf')]
        for font in fonts:
            font_char_ims[font] = dict(make_char_ims(os.path.join(folder_path,
                                                                  font),
                                                     FONT_HEIGHT))
        return fonts, font_char_ims


    def generate_ims():
        """
        Generate number plate images.

        :return:
            Iterable of number plate images.
        """
        variation = 1.0
        fonts, font_char_ims = load_fonts(FONT_DIR)
        num_bg_images = len(os.listdir("bgs"))
        while True:
            yield generate_im(font_char_ims[random.choice(fonts)], num_bg_images)


    if __name__ == "__main__":
        os.mkdir("test")
        os.mkdir("test/letter")
        os.mkdir("test/digit")
        os.mkdir("test/province")
        for i in range(24):
            os.mkdir("test/letter/%d"%i)
        for i in range(10):
            os.mkdir("test/digit/%d"%i)
        for i in range(6):
            os.mkdir("test/province/%d"%i)
```

```
im_gen = itertools.islice(generate_ims(), int(sys.argv[1]))
for img_idx, (im, c, p) in enumerate(im_gen):
    '''
    if (c>='A')and(c<='H'):
        fname = "test/letter/%d/%d.png"%(ord(c)-65, img_idx)
    elif (c>='J')and(c<='N'):
        fname = "test/letter/%d/%d.png"%(ord(c)-65-1, img_idx)
    elif (c>='P')and(c<='Z'):
        fname = "test/letter/%d/%d.png"%(ord(c)-65-2, img_idx)
    elif (c>='0')and(c<='9'):
        fname = "test/digit/%d/%d.png"%(ord(c)-48, img_idx)
    else:
        fname = "test/province/%d/%d.png"%(ord(c)-97, img_idx)
    '''
    fname = "test/{:08d}_{}.png".format(img_idx, c)
    print(fname)
    cv2.imwrite(fname, (1-im) * 255.)
```