```
# -*- coding: utf-8 -*-
"""
MA 326
Code for homework 3: SVM
"""
from matplotlib.pyplot import imread
import matplotlib.pyplot as plt
import numpy as np
from skimage.transform import resize
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.utils import shuffle
```

```
def boundaries(binarized,axis):
    # variables named assuming axis = 0; algorithm valid for axis=1
    # [1,0][axis] effectively swaps axes for summing
    rows = np.sum(binarized,axis = [1,0][axis]) > 0
    rows[1:] = np.logical_xor(rows[1:], rows[:-1])
    change = np.nonzero(rows)[0]
    ymin = change[::2]
    ymax = change[1::2]
    height = ymax-ymin
    too_small = 10 # real letters will be bigger than 10px by 10px
    ymin = ymin[height>too_small]
    ymax = ymax[height>too_small]
    return tuple(zip(ymin,ymax))
```

```
def separate(img):
    orig_img = img.copy()
    pure_white = 255.
    white = np.max(img)
    black = np.min(img)
    thresh = (white+black)/2.0
    binarized = img<thresh
    row_bounds = boundaries(binarized, axis = 0)
    cropped = []
    for r1,r2 in row_bounds:
        img = binarized[r1:r2,:]
        col_bounds = boundaries(img, axis=1)
        print(col_bounds)
        rects = [r1,r2,col_bounds[0][0],col_bounds[0][1]]
        cropped.append(np.array(orig_img[rects[0]:rects[1],rects[2]:rects[3]]/pure_white))
    return cropped

```

```
# Example usage
big_img = imread("d.png")
grey_big_img = big_img[:,:,0]*0.21+big_img[:,:,1]*0.72+big_img[:,:,2]*0.07 # convert to gray-scale image
grey_big_img = grey_big_img*255
grey_big_img = grey_big_img.astype("int")
print(np.min(grey_big_img))
print(np.max(grey_big_img))
plt.imshow(grey_big_img,cmap='gray')
plt.show()

imgs = separate(grey_big_img) # separates big_img (pure white = 255) into array of little images (pure white = 1.0)
for img in imgs:
    img = resize(img, (10,10))
    plt.imshow(img, cmap='gray')
    plt.show()
```

```
def partition(data, target, p):
  # EDIT: Realized last minute that we need to shuffle data,
  # luckily sklearn has built in shuffle function
  data, target = shuffle(data, target, random_state=42)

  split_point = int(len(data) * p) # Finds index to split array at

  # partitioning:
  train_data = data[:split_point]
  test_data = data[split_point:]
  train_target = target[:split_point]
  test_target = target[split_point:]

  return train_data, train_target, test_data, test_target
```

```
# Load in column pictures
# Note: each column contains 8 samples
d_col_img = imread('d.png')
e_col_img = imread('e.png')
f_col_img = imread('f.png')

# Convert to greyscale
grey_d_col = d_col_img[:,:,0]*0.21+d_col_img[:,:,1]*0.72+d_col_img[:,:,2]*0.07
grey_d_col = (grey_d_col*255).astype('int')

grey_e_col = e_col_img[:,:,0]*0.21+e_col_img[:,:,1]*0.72+e_col_img[:,:,2]*0.07
grey_e_col = (grey_e_col*255).astype('int')

grey_f_col = f_col_img[:,:,0]*0.21+f_col_img[:,:,1]*0.72+f_col_img[:,:,2]*0.07
grey_f_col = (grey_f_col*255).astype('int')

# Create image lists using separate function
d_imgs = separate(grey_d_col)
e_imgs = separate(grey_e_col)
f_imgs = separate(grey_f_col)

# Create truth labels
d_labels = np.zeros(8) # Note: could change 8 out for 'len(d_imgs)' for alterations
e_labels = np.ones(8)
f_labels = np.full(8, 2)
```

```
# Error concatenating unless elements of the letter_imgs are the same, so resizing:
resize_d_imgs = [resize(img, (10, 10)) for img in d_imgs]
resize_e_imgs = [resize(img, (10, 10)) for img in e_imgs]
resize_f_imgs = [resize(img, (10, 10)) for img in f_imgs]

# Combine data and labels
data = np.concatenate((resize_d_imgs, resize_e_imgs, resize_f_imgs), axis=0)
labels = np.concatenate((d_labels, e_labels, f_labels), axis=0)
labels
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 2.,
       2., 2., 2., 2., 2., 2., 2.])
```

```
train_data, train_labels, test_data, test_labels = partition(data, labels, .75)
train_data = np.array([img.flatten() for img in train_data])
test_data = np.array([img.flatten() for img in test_data])
```

```
svc = SVC(kernel='linear')
svc.fit(train_data, train_labels)
preds = svc.predict(test_data)
acc = accuracy_score(test_labels, preds) * 100
```

```
print("Prediction:", preds.astype(int))
print("Truth:", test_labels.astype(int))
print("Accuracy: ", acc, '%')
```

```
Prediction: [2 0 1 1 2 0]
Truth: [2 0 1 1 2 0]
Accuracy:  100.0 %
```

Q7:

Not sure if my handwriting or picture quality was just really good or consistent, or if the fact that I picked such different looking characters (d, e, and f) had a positive impact, but unless I drop to just 1 training sample, my accuracy is 100%. With just one sample it drops to 63.7%.

d

d

d

d

d

d

d

d

e
e
e
e
e
e
e
e

f
f
f
f
f
f
f
f