EE559 Homework 4 (week 5)

Jingquan Yan

USC ID: 1071912676

Email: jingquan@usc.edu

EE559 repository: Github

Modified plotDecBoundaries.py :

```
################################################
## EE559 HW Wk2, Prof. Jenkins, Spring 2018
## Created by Arindam Jati, TA
## Tested in Python 3.6.3, OSX El Captain
################################################

import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist

def plotDecBoundaries(training, label_train, sample_mean):

    #Plot the decision boundaries and data points for minimum distance to
    #class mean classifier
    #
    # training: traning data
    # label_train: class lables correspond to training data
    # sample_mean: mean vector for each class
    #
    # Total number of classes
    nclass =  max(np.unique(label_train))

    # Set the feature range for ploting
    max_x = np.ceil(max(training[:, 0])) + 1
    min_x = np.floor(min(training[:, 0])) - 1
    max_y = np.ceil(max(training[:, 1])) + 1
    min_y = np.floor(min(training[:, 1])) - 1

    xrange = (min_x, max_x)
    yrange = (min_y, max_y)

    # step size for how finely you want to visualize the decision boundary.
    inc = 0.01

    # generate grid coordinates. this will be the basis of the decision
    # boundary visualization.
    (x, y) = np.meshgrid(np.arange(xrange[0], xrange[1]+inc/100, inc),
np.arange(yrange[0], yrange[1]+inc/100, inc))

    # size of the (x, y) image, which will also be the size of the
    # decision boundary image that is used as the plot background.
```

```
    image_size = x.shape
    xy = np.hstack( (x.reshape(x.shape[0]*x.shape[1], 1, order='F'),
y.reshape(y.shape[0]*y.shape[1], 1, order='F')) ) # make (x,y) pairs as a bunch
of row vectors.
    # distance measure evaluations for each (x,y) pair.
    # pred_label = [1 if -(sample_mean[0]/sample_mean[1])*m[0] < m[1] else 0 for
m in xy]
    pred_label = [1 if m[0]*sample_mean[0] + m[1] * sample_mean[1] +
sample_mean[2] > 0 else 0 for m in xy]
    pred_label = np.array(pred_label)
    # reshape the idx (which contains the class label) into an image.
    decisionmap = pred_label.reshape(image_size, order='F')

    #show the image, give each coordinate a color according to its class label
    plt.imshow(decisionmap, extent=[xrange[0], xrange[1], yrange[0], yrange[1]],
origin='lower')

    # plot the class training data.
    plt.plot(training[label_train == 1, 0],training[label_train == 1, 1], 'rx')
    plt.plot(training[label_train == 2, 0],training[label_train == 2, 1], 'go')
    if nclass == 3:
        plt.plot(training[label_train == 3, 0],training[label_train == 3, 1],
'b*')

    # include legend for training data
    if nclass == 3:
        l = plt.legend(('Class 1', 'Class 2', 'Class 3'), loc=2)
    else:
        l = plt.legend(('Class 1', 'Class 2'), loc=2)
    plt.gca().add_artist(l)

    plt.show()
```

Problem 1:

```
import numpy as np
from plotDecBoundaries import plotDecBoundaries


def read_data(data_dir, label_dir):
    with open(data_dir, 'r') as dataset:
        reader = dataset.readlines()
        data_set = [rows.split(',') for rows in reader]
    for i in range(len(data_set)):
        data_set[i] = list(map(float, data_set[i]))
        data_set[i].append(1)

    with open(label_dir, 'r') as labelset:
        reader = labelset.readlines()
        label_set = [rows.split(',') for rows in reader]
    label_set = [int(i) for j in label_set for i in j]
    data_set_ref = [
        [-data_set[i][0], -data_set[i][1], -data_set[i][2]] if label_set[i] == 2
else [data_set[i][0], data_set[i][1],

      data_set[i][2]]
        for i in range(len(label_set))]
```

```
    return np.array(data_set), np.array(data_set_ref), np.array(label_set)


x, x_reflected, y = read_data("synthetic3_test.csv",
"synthetic3_test_label.csv")
index = np.random.permutation(len(y))
x, x_reflected, y = x[index], x_reflected[index], y[index]  # shuffle data and
label simultaneously
w_temp = np.asarray([0.1, 0.1, 0.1])
loop_counter = 0
stop_sign = 0
w = w_temp.copy()
while not stop_sign:
    no_change_count = np.zeros(len(y))
    for i in range(len(y)):
        if np.sum(w_temp * x_reflected[i]) <= 0:
            w = np.vstack((w, w_temp + x_reflected[i]))
            no_change_count[i] = 1
        else:
            w = np.vstack((w, w_temp))
        w_temp = w[-1].copy()
    loop_counter += 1
    if loop_counter == 1000 or np.count_nonzero(no_change_count) == 0: stop_sign
= 1

w_test = w[-1]
wrong = np.zeros(len(y))
for i in range(len(y)):
    if x[i][0] * w_test[0] + x[i][1] * w_test[1] + w[-1][2] < 0 and y[i] == 1:
        wrong[i] = 1
    elif x[i][0] * w_test[0] + x[i][1] * w_test[1] + w[-1][2] > 0 and y[i] == 2:
        wrong[i] = 1

err_rate = np.count_nonzero(wrong) / len(y)
print("The final weight vector is:", w_test)
print("The error rate is :", err_rate)
plotDecBoundaries(x, y, w_test)
```