EE559 Homework 1 (week 2)

Jingquan Yan

USC ID: 1071912676

Email: jingquan@usc.edu

EE559 repository: Github

Modified plotDecBoundaries.py :

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


def plotDecBoundaries(training, label_train, sample_mean, *argv):  # Additional
Input
    if len(argv) == 0:
        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.
        dist_mat = cdist(xy, sample_mean)
        pred_label_1 = np.argmax(dist_mat[:, 0:2],
                                 axis=1).tolist()  # It is obvious that here has
to be argmax instead of min.
        pred_label_2 = np.argmax(dist_mat[:, 2:4], axis=1).tolist()
```

```
        pred_label_3 = np.argmax(dist_mat[:, 4:6], axis=1).tolist()  # I am not
familiar with numpy so I use list instead.

        pred_label = list(zip(*[pred_label_1, pred_label_2, pred_label_3]))  #
unzip and list
        pred_label = [m.index(1) + 1 if m.count(1) == 1 else 0 for m in
pred_label]
        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)

        # plot the class mean vector.
        m1, = plt.plot(sample_mean[0, 0], sample_mean[0, 1], 'rd',
markersize=12, markerfacecolor='r', markeredgecolor='w')
        m2, = plt.plot(sample_mean[2, 0], sample_mean[2, 1], 'gd',
markersize=12, markerfacecolor='g', markeredgecolor='w')
        if nclass == 3:
            m3, = plt.plot(sample_mean[4, 0], sample_mean[4, 1], 'bd',
markersize=12, markerfacecolor='b',
                           markeredgecolor='w')

        # include legend for class mean vector
        if nclass == 3:
            l1 = plt.legend([m1, m2, m3], ['Class 1 Mean', 'Class 2 Mean',
'Class 3 Mean'], loc=4)
        else:
            l1 = plt.legend([m1, m2], ['Class 1 Mean', 'Class 2 Mean'], loc=4)

        plt.gca().add_artist(l1)

    else:
        nclass = len(np.unique(label_train))
        xrange = (-5, 5)
        yrange = (-5, 5)

        # step size for how finely you want to visualize the decision boundary.
```

```
        inc = 0.01

        (x, y) = np.meshgrid(np.arange(xrange[0], xrange[1] + inc / 100, inc),
                             np.arange(yrange[0], yrange[1] + inc / 100, inc))

        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.
        #  meshgrid 一开始x和y是分开的，这里将x和y先reshape展平，然后压扁（每个x和y一组）
        # distance measure evaluations for each (x,y) pair.
        dist_mat = cdist(xy, sample_mean)  # 生成n * 3的距离矩阵，3列是因为有3个class
        pred_label = np.argmin(dist_mat, axis=1)  # 找到每个meshgrid坐标点对三个
mean的距离最小的index，即分类。axis=1是每一行找

        # 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')  # 将[0,0]放在左上角还是左下角

        # 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)

        # plot the class mean vector.
        m1, = plt.plot(sample_mean[0, 0], sample_mean[0, 1], 'rd',
markersize=12, markerfacecolor='r',
                       markeredgecolor='w')
        m2, = plt.plot(sample_mean[1, 0], sample_mean[1, 1], 'gd',
markersize=12, markerfacecolor='g',
                       markeredgecolor='w')
        if nclass == 3:
            m3, = plt.plot(sample_mean[2, 0], sample_mean[2, 1], 'bd',
markersize=12, markerfacecolor='b',
                           markeredgecolor='w')

        # include legend for class mean vector
        if nclass == 3:
            l1 = plt.legend([m1, m2, m3], ['Class 1 Mean', 'Class 2 Mean',
'Class 3 Mean'], loc=4)
        else:
            l1 = plt.legend([m1, m2], ['Class 1 Mean', 'Class 2 Mean'], loc=4)

        plt.gca().add_artist(l1)
    plt.show()
```

Problem 1:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
coeff_mat = [[-1, -1, 5], [-1, 0, 3], [-1, 1, -1]]
coeff_mat_T = [[-(x[1] / x[0]), -(x[2] / x[0])] for x in coeff_mat]
dot_set = [[4, 1], [1, 5], [0, 0], [2.5, 3]]  # [2.5, 3] is an example for
indeterminate point
dot_x, dot_y = list(map(list, zip(*dot_set)))

plt.figure(figsize=(8, 8))
y_axis = np.linspace(-4, 15, 100)
plt.xlim(-5, 9)
plt.ylim(-4, 10)

for i in range(len(coeff_mat_T)):
    x_axis = np.array(coeff_mat_T[i][0]) * y_axis + np.array(coeff_mat_T[i][1])
    plt.plot(x_axis, y_axis, linewidth=4)

x1 = np.linspace(-5, 10, 150)
y1, y2 = - x1 + 5, x1 + 1

plt.fill_between(x1, y1, -5, where=x1 <= 3.1, alpha=0.5, label='Class 1')
plt.fill_between(x1, y1, np.max(y1), where=x1 <= 2.1, color='orchid', alpha=0.5,
label='Class 2')
plt.fill_between(x1, y2, np.max(y2), where=x1 >= 2, color='orchid', alpha=0.5)
plt.fill_between(x1, y2, -5, where=x1 >= 2.9, alpha=0.5, label='Class 3')

x_bar = plt.gca()
x_bar.spines['right'].set_color('none')
x_bar.spines['top'].set_color('none')
x_bar.spines['bottom'].set_position(('data', 0))
x_bar.spines['left'].set_position(('data', 0))

plt.scatter(dot_x, dot_y, color='r', zorder=100, linewidths=3)
plt.grid(True)

classify = [[] for i in range(len(dot_set))]
classes = ['Indeterminate Point' for i in range(4)]
for i in range(len(dot_set)):
    classify[i].append(-1 * (dot_set[i][0] + dot_set[i][1]) + 5)
    classify[i].append(-1 * dot_set[i][0] + 3)
    classify[i].append(-1 * dot_set[i][0] + dot_set[i][1] - 1)
    if classify[i][0] > 0 and classify[i][1] > 0:
        classes[i] = 1
    elif classify[i][0] > 0 and classify[i][2] > 0:
        classes[i] = 3
    elif classify[i][1] > 0 and classify[i][2] > 0:
        classes[i] = 2

list(map(lambda x, y: print("Dot", x, "is in class:", y), dot_set, classes))

plt.annotate(s='Class 1', xy=(0.2, 0.1), xytext=(1, 0.5),
arrowprops=dict(facecolor='red'))
plt.annotate(s='Class 2', xy=(1.2, 5.1), xytext=(2, 5.5),
arrowprops=dict(facecolor='red'))
plt.annotate(s='Indeterminate Point', xy=(4.2, 1.1), xytext=(5, 1.5),
arrowprops=dict(facecolor='blue'))
plt.annotate(s='Indeterminate Regions', xy=(2.7, 3.2), xytext=(3.2, 3.7),
arrowprops=dict(facecolor='blue'))
plt.legend()
```

```
plt.show()
```

Problem 2:

```
import numpy as np
from plotDecBoundaries import plotDecBoundaries  # datasets and script are
supposed to be in same directory

train_data_dir = 'wine_train.csv'  # datasets and script are supposed to be in
same directory
test_data_dir = 'wine_test.csv'
first_column = 0  # list index which starts from 0
second_column = 1  # list index which starts from 0
label_column = 13  # list index which starts from 0


def read_and_mean(directory):
    global first_column, second_column, label_column
    with open(directory, 'r') as train_data:
        reader = train_data.read().splitlines()
        x_data, y_data, label = [], [], []
        for rows in reader:
            rows = rows.split(',')
            x_data.append(rows[first_column])
            y_data.append(rows[second_column])
            label.append(rows[label_column])
        x_data = list(map(float, x_data))
        y_data = list(map(float, y_data))
        label = list(map(int, label))

    class_no = len(set(label))  # number of class labels
    class_mean = []
    for i in range(class_no):
        complement_x = [x_data[m] for m in range(len(label)) if label[m] != i +
1]
        complement_y = [y_data[m] for m in range(len(label)) if label[m] != i +
1]
        data_x = [x_data[m] for m in range(len(label)) if label[m] == i + 1]
        data_y = [y_data[m] for m in range(len(label)) if label[m] == i + 1]
        class_mean.append([sum(data_x) / len(data_x), sum(data_y) / len(data_y),
                           sum(complement_x) / len(complement_x),
sum(complement_y) / len(complement_y)])
    return list(map(list, zip(*[x_data, y_data]))), label, class_mean


def read_test_data(test_dir):
    global first_column, second_column, label_column
    with open(test_dir, 'r') as train_data:
        reader = train_data.readlines()  # if using csv, the reader could only
read once since it's a iterator
        test = [rows.split(',') for rows in reader]
        test_data = [[data[first_column], data[second_column]] for data in test]
        test_label = [data[label_column] for data in test]
        test_label = list(map(int, test_label))
    for first_column in range(len(test_data)):
        test_data[first_column] = list(map(float, test_data[first_column]))
    return test_data, test_label
```

```
# def cal_bound(mean_arr):
#     return [[(m[0]-m[2])/(m[3]-m[1]), (m[2]**2-m[0]**2+m[3]**2-m[1]**2)/(2*
(m[3]-m[1]))] for m in mean_arr]

def judge_and_err(input_data, mean):
    ss = [1 if (m[0] - i[0]) ** 2 + (m[1] - i[1]) ** 2 < (m[2] - i[0]) ** 2 +
(m[3] - i[1]) ** 2 else 0 for i in
          input_data for m in mean]  # direct distance calculation instead of
calling cdist function
    judged = list(zip(*[iter(ss)] * 3))  # using iteration to group each 3 of
the elements in the list
    # judged = [(1, 0, 0), (1, 0, 0), (1, 1, 0)...]
    seq = [m.index(1) + 1 if m.count(1) == 1 else 0 for m in judged]
    # seq = [1, 1, 0, 1, 1, 1, ..., 2, 0 ...]
    return judged, seq


train_data, train_label, train_mean = read_and_mean('wine_train.csv')
test_data, test_label = read_test_data('wine_test.csv')
judged, seq = judge_and_err(train_data, train_mean)  # change train_data to what
you like to input  ###
result = list(map(lambda x, y: 1 if x == y else 0, seq, train_label))  # turn 1,
2 and 3 in seq all into 1  ###
print("The accuracy of training dataset is {}.".format(result.count(1) /
len(result)))
train_mean_input = np.array(
    list(zip(*[iter(sum(train_mean, []))] * 2)))  # Convert mean_array to format
that fit the plot func
plotDecBoundaries(np.array(train_data), np.array(train_label), train_mean_input)
 ###

judged, seq = judge_and_err(test_data, train_mean)  # change train_data to what
you like to input  ###
result = list(map(lambda x, y: 1 if x == y else 0, seq, test_label))  # turn 1,
2 and 3 in seq all into 1  ###
print("The accuracy of testing dataset is {}.".format(result.count(1) /
len(result)))

train_mean_input = np.array(
    list(zip(*[iter(sum(train_mean, []))] * 2)))  # Convert mean_array to format
that fit the plot func
plotDecBoundaries(np.array(test_data), np.array(test_label), train_mean_input)
 ###
```

Problem 3:

```
from plotDecBoundaries import plotDecBoundaries
import numpy as np

data = [[-4,-4],[4,4]]
mean_1 = [[0, -2], [0, 1]]
label_1 = [5,6]
plotDecBoundaries(np.array(data), np.array(label_1), np.array(mean_1), 1)
mean_2 = [[0, -2], [0, 1], [2, 0]]
label_2 = [5,6,7]
plotDecBoundaries(np.array(data), np.array(label_2), np.array(mean_2), 1)
```