EE559 Homework 1 (week 2)

Jingquan Yan

USC ID: 1071912676

Email: jingquan@usc.edu

EE559 repository: Github

Global variables and functions:

```
import math
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 = 11  # list index which starts from 0
label_column = 13  # list index which starts from 0


def read_and_mean(dir, i, j, k):
    with open(dir, '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[i])
            y_data.append(rows[j])
            label.append(rows[k])
        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_index = [[] for i in range(class_no)]   # the index of classes with
size k * n
    class_mean = []
    for i in range(class_no):
        temp = [j for j in range(len(label)) if label[j] == (i + 1)]  # for
label 1, temp is from 0 to 49
        # class_index[i] = temp
        mean_coordinate = [sum(x_data[min(temp):(max(temp) + 1)]) / len(temp),
                           sum(y_data[min(temp):(max(temp) + 1)]) / len(temp)]
        class_mean.append(mean_coordinate)
    return x_data, y_data, label, class_mean


def judge(test, mean):
    temp = []
```

```
    out_put = []
    for i in range(len(mean)):
        temp.append([math.sqrt((test1[0] - mean[i][0]) ** 2 + ((test1[1] -
mean[i][1]) ** 2)) for test1 in test])
    temp = list(map(list, zip(*temp)))
    # Transpose. Little tricky here, first unzip temp and then zip them in
another dimension.
    out_put = [ax.index(min(ax)) + 1 for ax in temp]  # find the minimum
distance's index
    return out_put


def error_rate(output_label, test_label):
    boolean_out = [1 if i == j else 0 for i, j in
                   zip(output_label, test_label)]  # input 1 if the output is
correct, else 0
    correct = 0
    for i in range(len(boolean_out)):
        if boolean_out[i] == 1: correct += 1
    return correct / len(output_label)


def read_test_data(test_dir, i, j, k):
    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[i], data[j]] for data in test]
        test_label = [data[k] for data in test]
        test_label = list(map(int, test_label))
    for i in range(len(test_data)):
        test_data[i] = list(map(float, test_data[i]))
    return test_data, test_label
```

For synthetic dataset 1 and 2:

```
test_data, test_label = read_test_data(test_data_dir, first_column,
second_column, label_column)
x_data, y_data, train_label, mean_data = read_and_mean(train_data_dir,
first_column, second_column, label_column)
train_dataset = np.array([x_data, y_data]).T  # merge x_data and y_data and
convert to np.array
mean_data = np.array(mean_data)
train_label = np.array(train_label)
test_data = np.array(test_data)
test_label = np.array(test_label)

output = judge(test_data.tolist(), mean_data)  # or replace test_data with
train_dataset.tolist()
err_rate = 1 - error_rate(output, test_label.tolist())  # or replace test_label
with train_label.tolist()
print("The error rate is:", err_rate)
plotDecBoundaries(test_data, test_label, mean_data)  # or replace test_data with
train_dataset.tolist()
```

For wine dataset:

```
opt_i, opt_j, opt_err_rate = 0, 0, 100  # opt_err_rate is arbitrarily set that
is larger than 1
err_list = []
for i in range(label_column):  # we assume the label is the last column
    for j in range(i + 1, label_column):
        test_data, test_label = read_test_data(test_data_dir, i, j,
label_column)
        x_data, y_data, train_label, mean_data = read_and_mean(train_data_dir,
i, j, label_column)
        train_dataset = np.array([x_data, y_data]).T  # merge x_data and y_data
and convert to np.array
        train_label = np.array(train_label)

        output = judge(train_dataset.tolist(), mean_data)  # or replace
test_data with train_dataset.tolist()
        err_rate = 1 - error_rate(output, train_label.tolist())  # or replace
test_label with train_label.tolist()
        err_list.append(err_rate)
        if err_rate < opt_err_rate: opt_i, opt_j, opt_err_rate = i, j, err_rate

print("The minimum error rate comes with columns {} and {} with an error-rate of
{}".format(opt_i + 1, opt_j + 1,

           opt_err_rate))
print("The mean of error_rate is :", np.mean(np.array(err_list)))
print("The standard deviation of error_rate is :", np.std(np.array(err_list),
ddof=1))
```