```python
"""
Synthetic1 Dataset (Question 2(a))
Name: Haolun Cheng
EE559 HW4
"""
from __future__ import print_function
import copy
import csv
import random as rm
import numpy as np
import matplotlib.pyplot as plt
from plotDecBoundaries import plotDecBoundaries


class VectorOp(object):
    @staticmethod
    def element_multiply(x, y):
        return list(map(lambda x_y: x_y[0] * x_y[1], zip(x, y)))

    @staticmethod
    def element_add(x, y):
        return list(map(lambda x_y: x_y[0] + x_y[1], zip(x, y)))

    @staticmethod
    def scala_multiply(v, s):
        return map(lambda e: e * s, v)


class Perceptron(object):
    def __init__(self, input_num):
        self.weights = [0.1] * (input_num + 1)
        self.bias = 0.0

    def __str__(self):
        return 'weights\t:%s\nbias\t:%f\n' % (self.weights, self.bias)

    def train(self, shuffled_set, iteration):
        JnX = []
        weight_vectors_list = []
        weight = np.array(self.weights)
        wrongpts = 0
        Jw = 0
        ZnX = 0
        for m in range(iteration):
            for n in range(len(shuffled_set)):
                i = (m - 1) * len(shuffled_set) + n
                x = np.array(shuffled_set[n][:-1]).astype(float)
                if(float(shuffled_set[n][-1]) == 1):
                    ZnX = 1
                else:
                    ZnX = -1
                if(np.dot(weight, ZnX*x) <= 0):
                    wrongpts += 1
                    weight += ZnX * x
                    Jw += (-1) * np.dot(weight, ZnX * x)
                if i >= 9500:
                    JnX.append(Jw)
                    weight_vectors_list.append(weight)
            # Two halting conditions
            if wrongpts == 0:
                JnX.append(Jw)
                weight_vectors_list.append(weight)
                print("i.1 reached")
                break

            if i > 10000:
                print("i.2 reached")
                break

            wrongpts  = 0

        return np.array(JnX).astype(float), np.array(weight_vectors_list).astype(float)

    def calculate_error_rate(self, dataset, minwvalue):
        dataset_copy = copy.deepcopy(dataset)
        incorrect_count = 0
        for line in dataset_copy:
            input_vec = np.array(line[:-1]).astype(float)
            if np.dot(input_vec, minwvalue) > 0:
                line.append('1')
            else:
                line.append('2')

        for i in dataset_copy:
            if i[-1] != i[-2]:
                incorrect_count += 1
        return float(incorrect_count) / len(dataset_copy)


def f(x):
    return 1 if x > 0 else 0


def read_dataset(data_path):
    set_as_list = []
    with open(data_path,"r") as  f:
        dataset = csv.reader(f)
        for eachLine in dataset:
            if len(eachLine) != 0:
                eachLine.insert(0, 1)
                set_as_list += [eachLine]
    return set_as_list

def read_for_plot(data_path):
    input_vecs = []
    with open(data_path,"r") as f:
        dataset = csv.reader(f)
        shuffle_set = np.array(list(dataset))
        for line in shuffle_set:
            x, y = line[0], line[1]
            input_vecs.append((float(x), float(y)))
    return input_vecs


def train_and_perceptron():
    p = Perceptron(2)
    train_list = read_dataset("./synthetic1_train.csv")
    test_list = read_dataset("./synthetic1_test.csv")

    # shuffle data points
    rm.shuffle(train_list)
    shuffled_train_set = np.array(train_list)
    rm.shuffle(test_list)
    shuffled_test_set = np.array(test_list)

    train_datapts = []
    train_labels = []
    for line in shuffled_train_set:
        datapt1, label1 = line[1:3], line[-1]
        train_datapts.append(datapt1)
        train_labels.append(label1)

    test_datapts = []
    test_labels = []
    for line in shuffled_test_set:
        datapt2, label2 = line[1:3], line[-1]
        test_datapts.append(datapt2)
        test_labels.append(label2)

    Jvalue, weight_vectors = p.train(shuffled_train_set, 10000)
    minJvalue = min(Jvalue)
    minPos = 0
    for i in range(Jvalue.shape[0]):
        if Jvalue[i] == minJvalue:
            minPos = i
            break
    minwvalue = weight_vectors[minPos]
    error_rate_train_set = p.calculate_error_rate(train_list, minwvalue)
    error_rate_test_set = p.calculate_error_rate(test_list, minwvalue)
    print(f"The best weight vector omega (w) is {minwvalue}")
    print(f"The criterion function value is {minJvalue}")
    print(f"The error rate of training set is {error_rate_train_set}")
    print(f"The error rate of test set is {error_rate_test_set}")

    train_data_points = read_for_plot("./synthetic1_train.csv")
    plt.scatter(np.array(train_data_points)[:,0],np.array(train_data_points)[:,1])
    plt.xlabel('Feature1')
    plt.ylabel('Feature2')
    plt.title('Feature Plot of all Elements')
    plt.show()

    training = np.array(train_datapts).astype(float)
    label_train = np.array(train_labels).astype(float)
    weight_vector = minwvalue[1:]
    plotDecBoundaries(training, label_train, weight_vector)
    return p


if __name__ == '__main__':
    and_perception = train_and_perceptron()
```

```python
"""
Synthetic2 Dataset (Question 2(b))
Name: Haolun Cheng
EE559 HW4
"""
from __future__ import print_function
import copy
import csv
import random as rm
import numpy as np
import matplotlib.pyplot as plt
from plotDecBoundaries import plotDecBoundaries


class VectorOp(object):
    @staticmethod
    def element_multiply(x, y):
        return list(map(lambda x_y: x_y[0] * x_y[1], zip(x, y)))

    @staticmethod
    def element_add(x, y):
        return list(map(lambda x_y: x_y[0] + x_y[1], zip(x, y)))

    @staticmethod
    def scala_multiply(v, s):
        return map(lambda e: e * s, v)


class Perceptron(object):
    def __init__(self, input_num):
        self.weights = [0.1] * (input_num + 1)
        self.bias = 0.0

    def __str__(self):
        return 'weights\t:%s\nbias\t:%f\n' % (self.weights, self.bias)

    def train(self, shuffled_set, iteration):
        JnX = []
        weight_vectors_list = []
        weight = np.array(self.weights)
        wrongpts = 0
        Jw = 0
        ZnX = 0
        for m in range(iteration):
            for n in range(len(shuffled_set)):
                i = (m - 1) * len(shuffled_set) + n
                x = np.array(shuffled_set[n][:-1]).astype(float)
                if(float(shuffled_set[n][-1]) == 1):
                    ZnX = 1
                else:
                    ZnX = -1
                if(np.dot(weight, ZnX*x) <= 0):
                    wrongpts += 1
                    weight += ZnX * x
                    Jw += (-1) * np.dot(weight, ZnX * x)
                if i >= 9500:
                    JnX.append(Jw)
                    weight_vectors_list.append(weight)
            # Two halting conditions
            if wrongpts == 0:
                JnX.append(Jw)
                weight_vectors_list.append(weight)
                print("i.1 reached")
                break

            if i > 10000:
                print("i.2 reached")
                break

            wrongpts  = 0

        return np.array(JnX).astype(float), np.array(weight_vectors_list).astype(float)

    def calculate_error_rate(self, dataset, minwvalue):
        dataset_copy = copy.deepcopy(dataset)
        incorrect_count = 0
        for line in dataset_copy:
            input_vec = np.array(line[:-1]).astype(float)
            if np.dot(input_vec, minwvalue) > 0:
                line.append('1')
            else:
                line.append('2')

        for i in dataset_copy:
            if i[-1] != i[-2]:
                incorrect_count += 1
        return float(incorrect_count) / len(dataset_copy)


def f(x):
    return 1 if x > 0 else 0


def read_dataset(data_path):
    set_as_list = []
    with open(data_path,"r") as  f:
        dataset = csv.reader(f)
        for eachLine in dataset:
            if len(eachLine) != 0:
                eachLine.insert(0, 1)
                set_as_list += [eachLine]
    return set_as_list

def read_for_plot(data_path):
    input_vecs = []
    with open(data_path,"r") as f:
        dataset = csv.reader(f)
        shuffle_set = np.array(list(dataset))
        for line in shuffle_set:
            x, y = line[0], line[1]
            input_vecs.append((float(x), float(y)))
    return input_vecs


def train_and_perceptron():
    p = Perceptron(2)
    train_list = read_dataset("./synthetic2_train.csv")
    test_list = read_dataset("./synthetic2_test.csv")
```

```python

    # shuffle data points
    rm.shuffle(train_list)
    shuffled_train_set = np.array(train_list)
    rm.shuffle(test_list)
    shuffled_test_set = np.array(test_list)

    train_datapts = []
    train_labels = []
    for line in shuffled_train_set:
        datapt1, label1 = line[1:3], line[-1]
        train_datapts.append(datapt1)
        train_labels.append(label1)

    test_datapts = []
    test_labels = []
    for line in shuffled_test_set:
        datapt2, label2 = line[1:3], line[-1]
        test_datapts.append(datapt2)
        test_labels.append(label2)

    Jvalue, weight_vectors = p.train(shuffled_train_set, 10000)
    minJvalue = min(Jvalue)
    minPos = 0
    for i in range(Jvalue.shape[0]):
        if Jvalue[i] == minJvalue:
            minPos = i
            break
    minwvalue = weight_vectors[minPos]
    error_rate_train_set = p.calculate_error_rate(train_list, minwvalue)
    error_rate_test_set = p.calculate_error_rate(test_list, minwvalue)
    print(f"The best weight vector omega (w) is {minwvalue}")
    print(f"The criterion function value is {minJvalue}")
    print(f"The error rate of training set is {error_rate_train_set}")
    print(f"The error rate of test set is {error_rate_test_set}")

    train_data_points = read_for_plot("./synthetic2_train.csv")
    plt.scatter(np.array(train_data_points)[:,0],np.array(train_data_points)[:,1])
    plt.xlabel('Feature1')
    plt.ylabel('Feature2')
    plt.title('Feature Plot of all Elements')
    plt.show()

    training = np.array(train_datapts).astype(float)
    label_train = np.array(train_labels).astype(float)
    weight_vector = minwvalue[1:]
    plotDecBoundaries(training, label_train, weight_vector)
    return p


if __name__ == '__main__':
    and_perception = train_and_perceptron()
```

```python
"""
Wine Dataset (Question 2(c))
Name: Haolun Cheng
EE559 HW4
"""
from __future__ import print_function
import copy
import csv
import random as rm
import numpy as np


class VectorOp(object):
    @staticmethod
    def element_multiply(x, y):
        return list(map(lambda x_y: x_y[0] * x_y[1], zip(x, y)))

    @staticmethod
    def element_add(x, y):
        return list(map(lambda x_y: x_y[0] + x_y[1], zip(x, y)))

    @staticmethod
    def scala_multiply(v, s):
        return map(lambda e: e * s, v)


class Perceptron(object):
    def __init__(self, input_num):
        self.weights = [0.1] * (input_num + 1)
        self.bias = 0.0

    def __str__(self):
        return 'weights\t:%s\nbias\t:%f\n' % (self.weights, self.bias)

    def train(self, shuffled_set, iteration):
        JnX = []
        weight_vectors_list = []
        weight = np.array(self.weights)
        wrongpts = 0
        Jw = 0
        ZnX = 0
        for m in range(iteration):
            for n in range(len(shuffled_set)):
                i = (m - 1) * len(shuffled_set) + n
                x = np.array(shuffled_set[n][:-1]).astype(float)
                if(float(shuffled_set[n][-1]) == 1):
                    ZnX = 1
                else:
                    ZnX = -1
                if(np.dot(weight, ZnX*x) <= 0):
                    wrongpts += 1
                    weight += ZnX * x
                    Jw += (-1) * np.dot(weight, ZnX * x)
                if i >= 9500:
                    JnX.append(Jw)
                    weight_vectors_list.append(weight)
            # Two halting conditions
            if wrongpts == 0:
                JnX.append(Jw)
                weight_vectors_list.append(weight)
                print("i.1 reached")
                break

            if i > 10000:
                print("i.2 reached")
                break

            wrongpts  = 0

        return np.array(JnX).astype(float), np.array(weight_vectors_list).astype(float)

    def calculate_error_rate(self, dataset, minwvalue):
        dataset_copy = copy.deepcopy(dataset)
        incorrect_count = 0
        for line in dataset_copy:
            input_vec = np.array(line[:-1]).astype(float)
            if np.dot(input_vec, minwvalue) > 0:
                line.append('1')
            else:
                line.append('2')

        for i in dataset_copy:
            if i[-1] != i[-2]:
                incorrect_count += 1
        return float(incorrect_count) / len(dataset_copy)


def f(x):
    return 1 if x > 0 else 0


def read_dataset(data_path):
    set_as_list = []
    with open(data_path,"r") as  f:
        dataset = csv.reader(f)
        for eachLine in dataset:
            if eachLine[-1] != '3':
                if len(eachLine) != 0:
                    eachLine.insert(0, 1)
                    set_as_list += [eachLine]
    return set_as_list

def read_for_plot(data_path):
    input_vecs = []
    with open(data_path,"r") as f:
        dataset = csv.reader(f)
        shuffle_set = np.array(list(dataset))
        for line in shuffle_set:
            x, y = line[0], line[1]
            input_vecs.append((float(x), float(y)))
    return input_vecs


def train_and_perceptron():
    p = Perceptron(13)
    train_list = read_dataset("./wine_train.csv")
    test_list = read_dataset("./wine_test.csv")

```

```python
    # shuffle data points
    rm.shuffle(train_list)
    shuffled_train_set = np.array(train_list)
    rm.shuffle(test_list)
    shuffled_test_set = np.array(test_list)

    train_datapts = []
    train_labels = []
    for line in shuffled_train_set:
        datapt1, label1 = line[1:3], line[-1]
        train_datapts.append(datapt1)
        train_labels.append(label1)

    test_datapts = []
    test_labels = []
    for line in shuffled_test_set:
        datapt2, label2 = line[1:3], line[-1]
        test_datapts.append(datapt2)
        test_labels.append(label2)

    Jvalue, weight_vectors = p.train(shuffled_train_set, 10000)
    minJvalue = min(Jvalue)
    minPos = 0
    for i in range(Jvalue.shape[0]):
        if Jvalue[i] == minJvalue:
            minPos = i
            break
    minwvalue = weight_vectors[minPos]
    error_rate_train_set = p.calculate_error_rate(train_list, minwvalue)
    error_rate_test_set = p.calculate_error_rate(test_list, minwvalue)
    print(f"The best weight vector omega (w) is {minwvalue}")
    print(f"The criterion function value is {minJvalue}")
    print(f"The error rate of training set is {error_rate_train_set}")
    print(f"The error rate of test set is {error_rate_test_set}")
    return p


if __name__ == '__main__':
    and_perception = train_and_perceptron()
```