```python
#!/usr/bin/env python
# coding: utf-8

################################################
# Author:   Sarthak Kumar Maharana
# Email:    maharana@usc.edu
# Date:     02/25/2022
# Course:   EE 559
# Project:  Homework 4
# Instructor: Prof. B Keith Jenkins
################################################

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

np.random.seed(0) # to produce same random numbers

ROOTDIR = './data/'
TRAIN_FILENAME = 'wine_train.csv' ; TEST_FILENAME = 'wine_test.csv'


class PerceptronLearning:
    """
    Implementation of the perceptron learning algorithm.
    """
    def __init__(self):
        self.lr = 0.1
        self.n_iters = 10000

    def _init_weights(self, dims):
        # init the weights with a = 0.1 multiplied by a ones vector\
        # of size dims
        return np.ones(dims) * 0.1

    @staticmethod
    def _shuffle_data(x, y):
        # shuffle the data
        assert len(x) == len(y), "Unequal number of data points."
        p = np.random.permutation(len(x))
        return x[p], y[p]

    @staticmethod
    def _indicator(op):
        # return 1 if g(w) <= 0, else return 0
        return 1 if op <= 0 else 0

    def _predict(self, w, x):
        # compute w.x
        return np.dot(w, x)

    def _fit(self, x, y):
        # init the weights
        weights = self._init_weights(x.shape[1])
        # shuffle the data
        x, y = self._shuffle_data(x, y)
        # obtain z_n for reflecion of the data
```

```
        z = np.array([1 if yi == 1 else -1 for yi in y])

        iters = 0
        J, w = [], []
        decision = False

        while iters <= self.n_iters and not decision:
            misclassified, J_w = 0, 0
            for idx in range(len(x)):
                # compute g(w) = w*z_n*x_n
                op = self._predict(weights, x[idx]) * z[idx]
                # if g(w) <= 0, misclassified
                if self._indicator(op) == 1:
                    weights += self.lr * x[idx] * z[idx]
                    J_w += op
                    misclassified += 1
                else:
                    J_w += 0
                    weights = weights
            if misclassified == 0:
                # if no misclassified data, stop
                print('data is linearly separable')
                decision = True
                return -J_w, weights
            if iters >= 9500:
                J.append(-J_w)
                w.append(weights)
            iters += 1
        # obtain the weights with the lowest J for iters >=9500
        optimal_weights = w[J.index(min(J))]
        return min(J), optimal_weights

    def _classify(self, x, w):
        # classify data points, if w.x < 0, store 2, else store 1
        preds = [2 if self._predict(x[idx], w) <= 0 \
                else 1 for idx in range(len(x))]
        return preds


class Homework4:
    def __init__(self,
                train_path,
                test_path,
                dataset,
                n_iters = 10000,
                learning_rate = 0.1
                ):
        self.lr = learning_rate
        self.train_path = train_path
        self.test_path = test_path
        self.dataset = dataset
        self.n_iters = n_iters

    def _get_features(self, path):
        df = pd.read_csv(path, header = None)
        x, y = df.iloc[:, :-1].values, df.iloc[:, -1].values
        # Consider only the first two classes
        if self.dataset == 'wine':
            x, y = df.iloc[:, :-1].values, df.iloc[:, -1].values
            x, y = np.concatenate((x[y == 1], x[y == 2]), axis = 0), \
```

```
                    np.concatenate((y[y == 1], y[y == 2]), axis = 0)
            return x, y
        return x, y

    def _load(self):
        self.train_x, self.train_y = self._get_features(self.train_path)
        self.test_x, self.test_y = self._get_features(self.test_path)
        return self.train_x, self.train_y, self.test_x, self.test_y

    def _error(self, ground_truth, preds):
        return sum(ground_truth != preds) / len(ground_truth)

    def _plotter(self, x_data, y_data, model, mode = None):

        # Find max and min values of both the features
        max_x, min_x = np.ceil(max(x_data[:, 0])) + 1, np.floor(min(x_data[:, 0])) - 1
        max_y, min_y = np.ceil(max(x_data[:, 1])) + 1, np.floor(min(x_data[:, 1])) - 1

        # Calculate the range of values for x and y
        range_x = np.arange(min_x, max_x, 0.01)
        range_y = np.arange(min_y, max_y, 0.01)

        # Create a mesh grid of values
        xx, yy = np.meshgrid(range_x, range_y)

        # Predict the values on the mesh grid
        grid_preds = np.array(model._classify(np.c_[xx.ravel(), \
                     yy.ravel()], self.weights))
        preds = grid_preds.reshape(xx.shape) # matrix of classifications

        # Obtain data points of both the features
        x_1, x_2 = x_data[:,0], x_data[:,1]

        _, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (8,6), dpi = 200)

        # Plot the filled contours (decision regions)
        ax.contourf(xx, yy, preds, alpha = 0.25)
        # Plot the decision boundary.
        ax.contour(xx, yy, preds, colors = 'k', linewidths = 0.8)
        # Plot the data points (scatter plot)
        ax.scatter(x_1, x_2, c = y_data, edgecolors = 'k')

        ax.grid(False)
        ax.set_xlabel('Feature 1')
        ax.set_ylabel('Feature 2')
        ax.set_title(f'Feature space w/ decision boundary and regions of {self.curr_dataset} : {mode}')
        plt.savefig(f'{self.curr_dataset}_{mode}.png')
        # plt.show()


    def _runner(self):
        self.curr_dataset = self.train_path.split('/')[-1].split('_')[0]
        print(f'Running scripts for dataset: {self.curr_dataset}')

        self.train_x, self.train_y, self.test_x, self.test_y = self._load()

```

```
        # load model params
        model = PerceptronLearning()
        # obtain value of criterion function and the optimal weights.
        J, self.weights = model._fit(self.train_x, self.train_y)

        print(f"The optimal weights and the final criterion function J(w) \
              are {self.weights} and {round(J, 5)} respectively.")

        if self.dataset != 'wine':
            # decision boundary and regions on the training data
            self._plotter(self.train_x, self.train_y, model, mode =
'training')
            # Error on the train set
            preds = model._classify(self.train_x, self.weights)
            e_train = self._error(self.train_y, preds)
            print(f"The error on the train set is {round(e_train, 4)}.")

            # decision boundary and regions on the test data
            self._plotter(self.test_x, self.test_y, model, mode = 'test')
            # Error on the test set
            preds = model._classify(self.test_x, self.weights)
            e_test = self._error(self.test_y, preds)
            print(f"The error on the test set is {round(e_test, 4)}.")

        else:
            # Error on the train set
            preds = model._classify(self.train_x, self.weights)
            e_train = self._error(self.train_y, preds)
            print(f"The error on the train set is {round(e_train, 4)}.")

            # Error on the test set
            preds = model._classify(self.test_x, self.weights)
            e_test = self._error(self.test_y, preds)
            print(f"The error on the test set is {round(e_test, 4)}.")


if __name__ == '__main__':
    # get type of dataset: 'synthetic1', 'synthetic2'. or 'wine'
    dataset = TRAIN_FILENAME.split('_')[0]
    train_filepath = os.path.join(ROOTDIR, TRAIN_FILENAME)
    test_filepath = os.path.join(ROOTDIR, TEST_FILENAME)
    hw = Homework4(train_filepath, test_filepath, dataset)
    hw._runner()

```