EE559_Code_HW2

February 9, 2022

1 Problem 2 (a, b, c)

One-vs-one approach of nearest means classifier.

To run this, please export to a local IDE and change "ROOTDIR" to the directory path where wine_train.csv and wine_test.csv are.

[ ]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

##################################################
# Author:   Sarthak Kumar Maharana
# Email:    maharana@usc.edu
# Date:     02/06/2022
# Course:   EE 559
# Project:  Homework 2
# Instructor: Prof. B Keith Jenkins
##################################################

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from itertools import combinations

from plotDecBoundaries_OvO import *

ROOTDIR = '~/Desktop/spring_22/EE_559/hw1/codes_2/data/'
train_file = 'wine_train.csv'
test_file = 'wine_test.csv'

class OnevsOne:
    """
    One-vs-One Nearest Means Classifier.
    """

    def __init__(self, train_file, test_file):
```

```
        self.train_file = train_file
        self.test_file = test_file

    def _read_csv_get_features(self, filename):
        """ Read csv files and return features, labels, and dataframe. """
        df = pd.read_csv(os.path.join(ROOTDIR, filename),
                         header = None
                         )
        x, y = df.iloc[:, : -1].values, df.iloc[:, -1].values
        return x, y, df

    def _load(self):
        """ Utility function to load data. """
        self.train_x, self.train_y, _ = self._read_csv_get_features(self.
↪train_file)
        self.test_x, self.test_y, _ = self._read_csv_get_features(self.
↪test_file)
        return self.train_x, self.train_y, self.test_x, self.test_y

    def _plot_data(self):
        """ Plot for visualization. """
        plt.scatter(self.train_x[:, 0],
                    self.train_x[:, 1],
                    c = self.train_y,
                    s = 50,
                    cmap = 'viridis'
                    )
        plt.show()

    @staticmethod
    def L2distance(x, y):
        """ Compute L2 (Euclidean) distance between two vectors. """
        return np.sqrt(np.sum((x - y)**2))

    def _flag_assign(self, x, mean, label_1, label_2, flag_12, flag_13,␣
↪flag_23):
        """ Assign flags by classifying. """
        """ Perform OvO here. """
        for row in range(len(x)):
            # Compute distance between each pair of points and each mean.
            dist_a = self.L2distance(x[row], mean[0])
            dist_b = self.L2distance(x[row], mean[1])
            if label_1 == 1 and label_2 == 2:
                # discriminant functions
                if dist_a > dist_b:
                    flag_12[row, 1] = 1
                else:
```

```
                    flag_12[row, 0] = 1
            elif label_1 == 1 and label_2 == 3:
                if dist_a > dist_b:
                    flag_13[row, 1] = 1
                else:
                    flag_13[row, 0] = 1
            elif label_1 == 2 and label_2 == 3:
                if dist_a > dist_b:
                    flag_23[row, 1] = 1
                else:
                    flag_23[row, 0] = 1
        return flag_12, flag_13, flag_23

    def _accuracy(self, y, flag12, flag13, flag23):
        """ Compute accuracy. """
        correct = 0
        for row in range(len(y)):
            if y[row] == 1:
                if flag12[row, 0] == 1 and flag13[row, 0] == 1: correct += 1
            elif y[row] == 2:
                if flag12[row, 1] == 1 and flag23[row, 0] == 1: correct += 1
            elif y[row] == 3:
                if flag13[row, 1] == 1 and flag23[row, 1] == 1: correct += 1
        return float(correct / len(y))

    def _fit(self, x, y):
        """ Fit ("Train") the model. """
        final_means = np.array([])
        # ['1,2', '1,3', '2,3'] -> combinations of classes.
        classifier_combos = [",".join(map(str, comb)) for comb in␣
↪combinations(np.unique(y), 2)]
        for idx in range(len(classifier_combos)):
            #obtain current labels
            label_1, label_2 = int(classifier_combos[idx][0]),␣
↪int(classifier_combos[idx][2])
            #obtain current means
            classifier_mean_1 = np.mean(x[y == label_1], axis = 0).reshape(1,␣
↪-1)
            classifier_mean_2 = np.mean(x[y == label_2], axis = 0).reshape(1,␣
↪-1)
            total_classifier = np.vstack([classifier_mean_1, classifier_mean_2])
            final_means = np.vstack([final_means, total_classifier]) if␣
↪final_means.size else total_classifier
        return final_means, classifier_combos

    def _predict(self, x_train, y_train, x_test, y_test, final_means,␣
↪classifier_combos):
```

```
        """ Predict the labels for train/test data. """
        # flags for train and test (all possible combinations)
        flag_12, flag_13, flag_23 = np.zeros((len(x_train), 2)), np.
↪zeros((len(x_train), 2)) \
                                                          , np.zeros((len(x_train),␣
↪2))
        flag_12_test, flag_13_test, flag_23_test = np.zeros((len(x_train), 2)),␣
↪np.zeros((len(x_train), 2)) \
                                                          , np.zeros((len(x_train),␣
↪2))

        for idx, cluster in enumerate(range(0, len(final_means) // 2 + 2, 2)):
            # current classes
            label_1, label_2 = int(classifier_combos[idx][0]),␣
↪int(classifier_combos[idx][2])

            print(f"--- Current classes : {label_1}, {label_2} ---")
            print("--- Computing and plotting for the training data ---")
            # assign labels for train data
            flag12, flag13, flag23 = self._flag_assign(x_train,␣
↪final_means[cluster: cluster + 2, :], label_1, label_2, flag_12, flag_13,␣
↪flag_23)
            DecBoundaries_perclass(x_train, y_train, final_means[cluster:␣
↪cluster + 2, :], label_1, label_2)

            print("--- Computing and plotting for the test data ---")
            # assign labels for test data
            flag12_test, flag13_test, flag23_test = self._flag_assign(x_test,␣
↪final_means[cluster: cluster + 2, :], label_1, label_2, flag_12_test,␣
↪flag_13_test, flag_23_test)
        return flag12, flag13, flag23, flag12_test, flag13_test, flag23_test

    def _runner(self):
        """ Runner method. """
        # Load data.
        x_train, y_train, x_test, y_test = self._load()
        # Consider the first two features.
        x_train, x_test = x_train[:,:2], x_test[:,:2]
        # Obtain cluster means (6x2 array) and classifier combinations.
        final_means, classifier_combos = self._fit(x_train, y_train)
        # Predict.
        flag12, flag13, flag23, flag12_test, flag13_test, flag23_test = self.
↪_predict(x_train, y_train, x_test, y_test, final_means, classifier_combos)
        print("--- Final decision boundaries and regions ---")
        # Plot final decision boundaries and regions.
        # Use hw1 plot decision boundary function.
```

```
        plotDecBoundaries(x_train, y_train, np.vstack([final_means[0],␣
↪final_means[1], final_means[3]]))
        plotDecBoundaries(x_test, y_test, np.vstack([final_means[0],␣
↪final_means[1], final_means[3]]))
        print(f"Classification accuracy on the train set: {round(self.
↪_accuracy(y_train, flag12, flag13, flag23), 3)}")
        print(f"Classification accuracy on the test set: {round(self.
↪_accuracy(y_test, flag12_test, flag13_test, flag23_test), 3)}")


if __name__ == '__main__':
    ovo = OnevsOne(train_file, test_file)
    ovo._runner()
```

Modified function to plot the 2-class decision boundaries and regions. Export the below plotting module as "plotDecBoundaries_OvO.py"

To plot the final decision boundaries and regions, the original plot function from hw1 has been used and hence not added here.

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


def DecBoundaries_perclass(training, label_train, sample_mean, label1, label2):
    # Set the feature range for plotting
    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 = 0.005

    # 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 = np.argmin(dist_mat, axis=1)

    # 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')

    # find the missing label (not current labels)
    remaining_label = label_train[label_train != label1]
    remaining_label = remaining_label[remaining_label != label2][0]

    # plot the class training data.
    plt.plot(training[label_train == label1, 0], training[label_train ==␣
↪label1, 1], 'rx')
    plt.plot(training[label_train == label2, 0], training[label_train ==␣
↪label2, 1], 'go')
    plt.plot(training[label_train == remaining_label, 0], training[label_train␣
↪== remaining_label, 1], 'b*')

    leg = plt.legend((str(label1), str(label2), str(remaining_label)), loc=2)
    plt.gca().add_artist(leg)

    # plot the class mean vector of current pair of classes
    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')

    l1 = plt.legend([m1, m2], ['Class' + ' ' + str(label1) + ' ' + 'Mean',␣
↪'Class' + ' ' + str(label2) + ' ' + 'Mean'], loc=4)
    plt.gca().add_artist(l1)
    plt.savefig('DecB_'+str(label1)+'_'+str(label2)+'.png')
    plt.show()
```