CE4041 Project2: AdaBoost Classifier using weak linear classifiers

Andrei Cristian Codrea
Student ID:22236341

November 28, 2022

Contents

1 Introduction

The goal is to build a program that uses the AdaBoost algorithm in order to acheive high classification accuracy from a bunch of low accuracy, weak learners. The data used for this project consists of a training set with 400 samples and a testing set of 1200 samples. The goal is to plot the classification accuracy in both the training set and and the testing set, and find how many weak learners we need to achieve high accuracy. The weak learner will be a linear classifier. The linear classifier is described in Peter Flach's book[1] and the classifier presented in this paper is based on Flach's idea.

A weak linear classifier, as described by Flach, is based on the fact that we can create a decision boundary that separates the data points based on their label. This is done by finding the positive and negative centres of mass of this points.

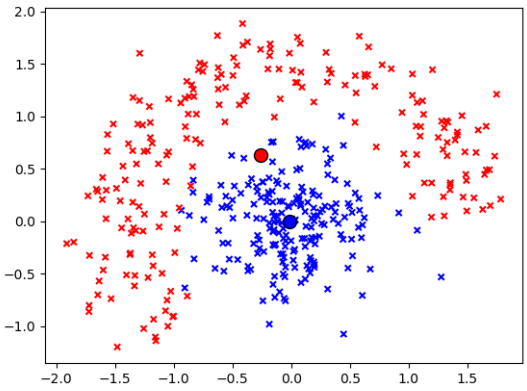

Figure 1: Weighted positive mean and negative positive mean

Once we got this two points we can create a line refereed to as the orientation line. After this, all data points will be projected on this line. Any two adjacent points can create a separation line which will be used as our threshold when making predictions in the algorithm. We want to find the best separation line, the one which gives minimum error (For this algorithm we will need the minimum weighted error).

The AdaBoost algorithm learns with the help of weights. This weights will change the position of the orientation line and thus, the separation line constantly. The weights are adjusted for each weak learner accordingly so that points which are classified incorrectly, have a bigger impact in order for the algorithm to improve.

In order to create the project, I looked out for a simpler implementation of the AdaBoost algorithm that I could research first. There are two implementations that I found: one using a Gini Index[2] for assessing different features and one using a Decision Stump[3]. The implementation presented here is based on both these articles as they provide a good starting ground for an AdaBoost algorithm.

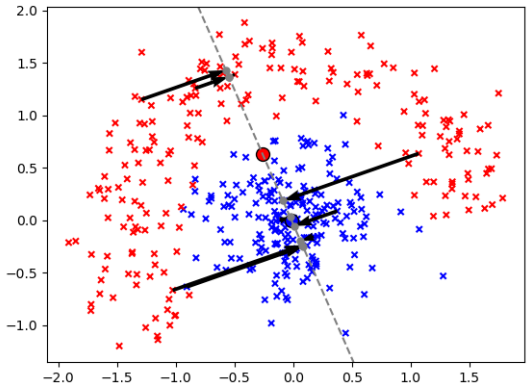

Figure 2: Weighted positive mean and negative positive mean

2 Methods

2.1 Preparing data

2.1.1 Importing Libraries

These following libraries are used for preparing data, manipulating data and creating relevant plots:

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

2.1.2 Loading dataset

```
def load_data():
    # load data
    data = np.loadtxt("adaboost-train-22.txt")
    X_train = data[:, 0:2]
    y_train = data[:, 2]
    data2 = np.loadtxt("adaboost-test-22.txt")
    X_test = data2[:, 0:2]
    y_test = data2[:, 2]
    return X_train, y_train, X_test, y_test
```

Here we load the data in arrays that we are going to use later. The array are split into the ones used for training and for testing

2.2 AdaBoost algorithm

The algorithm is implemented using classes. In this case we are going to have to classes: one for our linear classifier, and one for the actual algorithm. In the linear classifier class we will have two methods. The first method has to find the best separation line. A separation line is the midway point between to adjacent data points that have been projected onto the orientation line.

2.2.1 Linear Classifier Class

```
        for i in range(len(canditate_points) - 1):

         prediction1 = np.ones(len(y)) * (-1)
         prediction2 = np.ones(len(y))
         # calculate separation point
         sep_line = (canditate_points[i + 1][0] + canditate_points[i][0]) / 2
         sep_arr = np.append(sep_arr, sep_line)
```

Afterwards, the minimum error and polarity are calculated based on a bunch of candidate separation lines. The polarity tells us which label had a better classification.

Error calculation:

```
error = np.array([sum(canditate_points[:, 1] * (np.not_equal(canditate_points
    [:, 2], prediction1))),
                      sum(canditate_points[:, 1] * (np.not_equal(
                          canditate_points[:, 2], prediction2)))])
```

Finding minimum error:

```
min_error = float('inf')
    line_i = 0
    # get minimum error and polarity
    for err1, err2 in error_arr:
        if err1 < min_error:
            min_error = err1
            self.polarity = 1
            self.best_sep = sep_arr[line_i]
        if err2 < min_error:
            min_error = err2
            self.polarity = -1
            self.best_sep = sep_arr[line_i]

        line_i +=1
```

Finally, the linear classifier has a method for predicting data using the best threshold it currently has, which will be the separation line found before.

```
    for i in range(len(X)):
        if self.polarity == 1:
            if projections[i] < self.best_sep:
                y_pred[i] = 1
            else:
                y_pred[i] = -1
        else:
            if projections[i] < self.best_sep:
                y_pred[i] = -1
            else:
                y_pred[i] = 1
     # y_pred holds our predictions
```

2.2.2 AdaBoost Class

All that we have to do here is to, initialize the weights, find the best separation line, create the predictions using the weak classifier, calculate alpha and update the weights. In the end all the weak learners are combined in order to give out the final prediction.

```
class Adaboost:

def __init__(self, n_clsfs):
    self.clsfs = []
    self.n_clsfs = n_clsfs
    self.alphas = []

def fit(self, X, y):


    # init weights
    w = np.ones(len(y)) * 1 / len(y)

    for i in range(0, self.n_clsfs):

        linear = LinearClassifier()
        min_error = linear.find_sep_line(X, y, w)
        print(f'min_error: {min_error}')

        # calculate predictions
        predictions = linear.predict(X)


        # calculate alpha
        EPS = 1e-10 # a small non-zero value
        alpha = 0.5 * np.log((1 - min_error) / min_error)
        print(f'alpha: {alpha}')
```

```
            # update weights
            w = w * np.exp(-alpha * y * predictions)
            w = w / sum(w)
            #print(f'w: {w}')

            # save classifier
            self.alphas.append(alpha)
            self.clsfs.append(linear)

    def predict(self, X):
        clf_preds = []
        for i in range(0, self.n_clsfs):
            clf_preds.append(self.alphas[i] * self.clsfs[i].predict(X))

        y_pred = np.sum(clf_preds, axis=0)
        y_pred = np.sign(y_pred).astype(int)

        return y_pred


def load_data():
    # load data
    data = np.loadtxt("adaboost-train-22.txt")
    X_train = data[:, 0:2]
    y_train = data[:, 2]
    data2 = np.loadtxt("adaboost-test-22.txt")
    X_test = data2[:, 0:2]
    y_test = data2[:, 2]

    return X_train, y_train, X_test, y_test
```

2.3 Training

We train the algorithm and we also try to find an n where we achieve 100% accuracy.

```
   # train set
    acc_list = []
    for i in range(1, 40): # iterate through diffrent numbers of weak classifiers

        # initialize adaboost object with i weak classifiers
        adaboost = Adaboost(i)
        # train
        adaboost.fit(X_train, y_train)
        # predict using strong classifier
        predictions = adaboost.predict(X_train)
        # save accuracies so we can plot them after
        acc = accuracy(y_train, predictions)
        acc_list.append(acc)
```

2.4 Evaluating the model

We test the training model against a new set and see what kind of accuracy can be expected.

```
# test set
for i in range(1, 100):
    adaboost = Adaboost(i)
    adaboost.fit(X_train, y_train)
    predictions = adaboost.predict(X_test)
    acc = accuracy(y_test, predictions)
    acc_list2.append(acc)
```

3 Results

From the plots we can see that 100% accuracy on the training set is achieved in 35 rounds. Also, we can observe that high accuracy is achieved on the testing set as well, 100% using 80 classifiers. I also wanted to create a contour plot for the decision boundary but couldn't quite manage to do it in time.

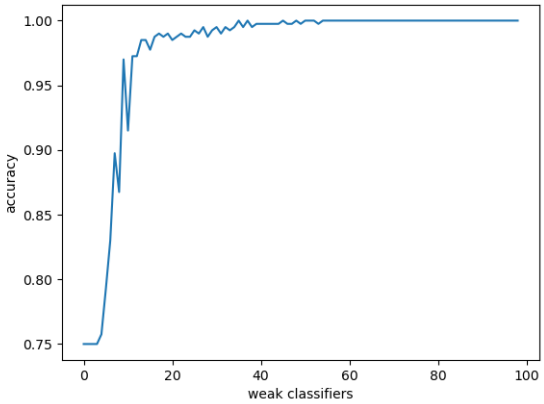

Figure 3: Training set classification accuracy

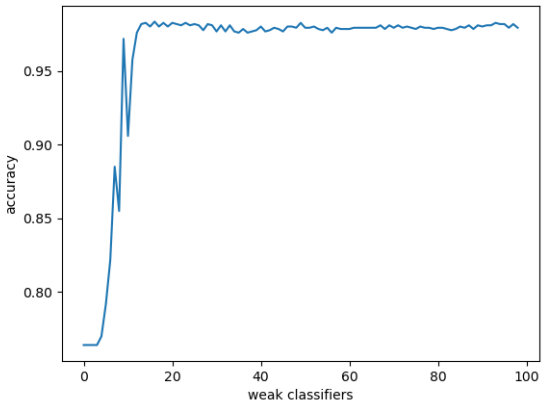

Figure 4: Testing set classification accuracy

References

[1] Peter Flach. Machine learning: The art and science of algorithms that make sense of data.

[2] Dominik Polzer. AdaBoost, Step-by-Step.
`https://towardsdatascience.com/adaboost-in-7-simple-steps-a89dc41ec4`.

[3] Patrick Loeber. AdaBoost in Python - ML From Scratch 13.
`https://www.python-engineer.com/courses/mlfromscratch/13_adaboost/`.