The Intuition and working behind AdaBoost

Sidharth Sekhar Follow
Sep 4 · 5 min read

There are multiple machine learning algorithms that classify data. We have seen the working of Support Vector Machines and Logistic Regression in the previous blog posts, now it's time to explore another algorithm called AdaBoost or Adaptive Boosting. This is a variant of Decision Tree algorithm.

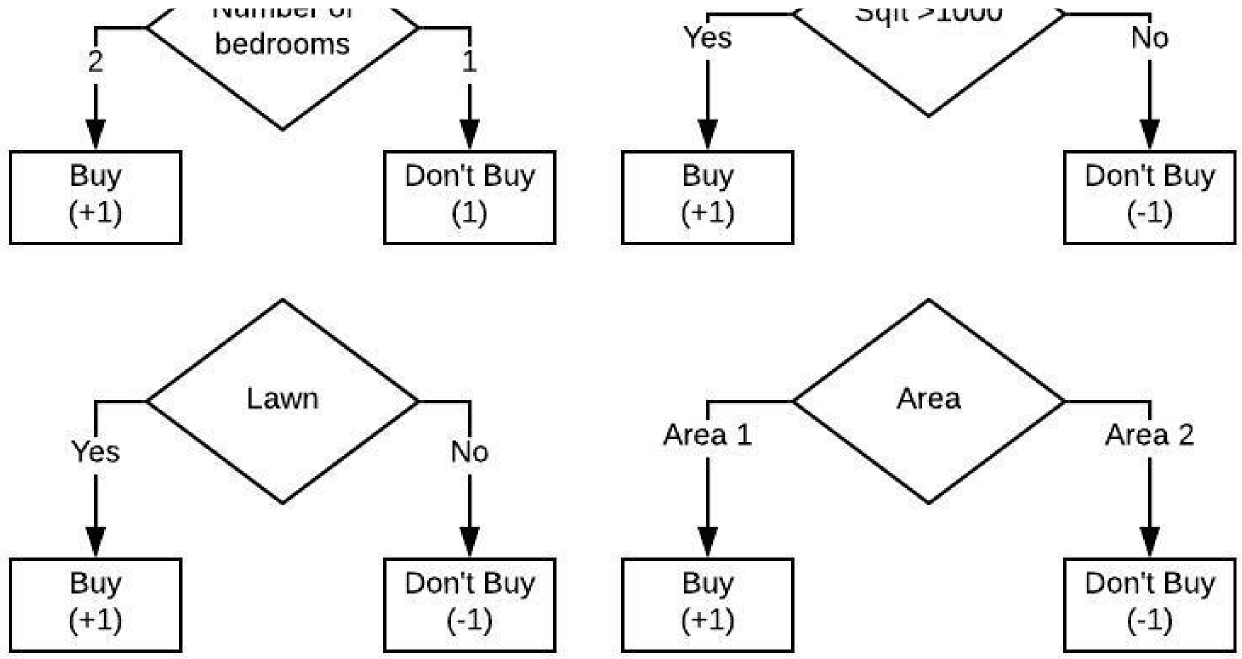
AdaBoost is a part of bigger set of algorithms that belong to methods called Ensemble learning. The entire idea behind Ensemble learning is to create multiple learning models instead of one model and predict the values.

AdaBoost does this process of predicting the value by creating multiple weak learners, associating a weight to each data point based on the predictions of that point in each learner and finally giving the output.

Weak Learners and Boosting :

They are basically multiple models(in this case Decision Tree) that are run on subset of data features. These weak learners have one particularly interesting property that we will take advantage of while creating boosting algorithms which is underfitting or low variance.

To understand better look at the diagram below



Multiple Decision stumps

Let us assume we have to predict whether someone is going to buy a house or not using Machine Learning algorithms. Instead of creating one decision tree or logistic regression model to predict, what we do is create multiple weak learner i.e. Decision Stumps or poor logistic regression model or tree with depth 1 or 2, one after the other to predict the output. In AdaBoost we rig the system in such a way that features that are predicted poorly in previous learner are predicted with better results in the subsequent model.

Let us give an example input to the above Decision stumps.

$$X(Numbedroom = 2, sqft = 1002, Lawn = No, Area = Area1)$$

input to the stumps

The *Numbedroom* in the above input is '*2*', this will go to the primary stump and yield prediction '*+1*'. The *sqft* will go to the second stump and will yield a prediction of '*+1*', *Lawn* will get a prediction of '*-1*' and similarly *Area* will get a prediction of '*+1*' . So the final question that we need to ask is , how do we combine all these predictions from various weak learners and give one final prediction to the above data point?. This is done with the formula below

$$sign(w_1 f_1(x_1) + w_2 f_2(x_1) + w_3 f_3(x_1) + w_n f_n(x_1))$$

weighted voting scheme for 1st data point

$$f_1(x_1), f_2(x_2), f_3(x_3), f_n(x_n)$$

Predictions from each scheme i.e. +1 or -1

We basically take the final sign after we multiply a weight associated with each stump is multiplied with the prediction and added with the rest of the values from each stump.

$$\hat{y} = sign \sum_{i=1}^{i=L} \hat{w}_i f_i(x)$$

Equation of final prediction. L number of stumps.

AdaBoost:

The way AdaBoost works is as follows:

1.) Primarily each data point is initialized with a weight '*alpha*' that is equal to (1/number of data points).

2.) Then for each weak learner or model, we iterate to calculate the predicted value, then go onto compute the weight '*w*'(weighted error) and based on that re-adjust the '*alpha*' term for next iteration until the very end.

3.) After all the iterations are done we calculate the prediction based on the above equation.

There are two fundamental questions with respect to AdaBoost that we need to ask here, How are we going to update the weights '*w*' and how are we going to update the '*alpha*' values.?

In this algorithm what we do is add all the weights of data points that have been miss-classified and divide it by the weight of all data points i.e.

$$\sum_{i=1}^{i=n} \alpha_i (\hat{y} \neq y)$$

Summing over weights of misclassified points

$$\frac{\sum_{i=1}^{i=n} \alpha_i(\hat{y} \neq y)}{\sum_{i=1}^{i=n} \alpha_i}$$

Weighted error formula

Following the creation of weighted error, we ensure that we update the weight associated with each stump , whose formula is

$$w = (0.5) * log((1 - weightederror)/(weightederror))$$

updating weight for each stump

The above equation is a very interesting one:

For a value of weighted error = 0.5, it will yield a '0' since the output as (1/2) * log(1) equals 0 while a a good decision stump that classifies the points in a right way will have very low weighted error which implies a high value for ((1-weighted_error)/weighted_error), taking the half of log of this value will yield a positive weight with high value. For stumps with poor classifications the weights will be pretty low.

In the section below on updating '*alpha*' , we will see why decision stumps with good classification have high weights and stumps associated with poor classification have low value for weights.

After the weights are updated we update alpha's in such a way that the points that are classified wrongly are given more importance than the points classified correctly thereby ensuring that our machine learning model caters to miss-classified points more. This is done with the help of the exponent function.When predictions are correct the alphas associated with each data point are updated using the below formula

$$\alpha_{i+1} = \alpha_i * e^{-\hat{w}_j}$$

i -> For each data point, j->for each stump

When the predictions are incorrect, then we update the alpha value associated with each data point using this formula

$$\alpha_{i+1} = \alpha_i * e^{\hat{w}_j}$$

i -> For each data point, j->for each stump

The use of the exponential function with negative and positive sign here is critical as each time a point is classified right we use a negative value on top of the exponent along with the weight associated with the stump. A high value for '*w*' means points that are classified right are given less importance in the next stump. In the event that a stump has done miss-classified on certain set of data points, the alpha value associated with those points will have higher value in the next stump there by giving them more importance.

Finally, once we have all our stumps based on our training data along with associated weights(not the weighted_error but the weights associated with each stump), we calculate the predictions for the data that has to be classified with,

$$\hat{y} = sign \sum_{i=1}^{i=L} \hat{w}_i f_i(x)$$

Equation of final prediction. L number of stumps.

The github gist is given below.

Note: The gist is only for adaboost and not for creating a decision stump

```
def create_adaboost(x_train,y):
    alpha = np.full((x_train.shape[0],1),1/x_train.shape[0])
    w=[]
    stumps=[]
    for j in range(number_of_stumps):
        tree=create_stump(x_train,y_train,alpha,max_depth=1) #creating a decision stump or weak
        predictions=predictstumps(x_train,tree)  #predicting the values
        stumps.append(tree)   #creating a list of all stumps that are bein created
        error=0
        weighted_error=0
        for i in range(100): #number of datapoints 100 taken as example
            if(predictions[i]!=y[i]):
                error+=alpha[i]   #computing the error of each mis-classification
```

```python

        weighted_error = error/np.sum(alpha)

        w.append((1/2)*np.log((1-weighted_error)/weighted_error)) #updating weights for each stu


    for i in range(100):
            if(predictions[i]==y[i]):   #updating alpha values
                alpha[i]=alpha[i]*np.exp(-w[j])
            else:
                alpha[i]=alpha[i]*np.exp(w[j])

    alpha=alpha/np.sum(alpha)  #normalizing alpha's
    return(stumps,w)
def predict(stumps,w,x_test):
    scores=np.full((x_test.shape[0],1),0)
    for i,tree in enumerate(stumps):
        preds=predict(tree,x_test)
        score[i]=w[i]*preds
        if(score[i]>0):
            return(1)
        else:
            return(-1)


```

adaboost.py hosted with ♡ by GitHub view raw

Feel free to point out any errors in comments.

Machine Learning