**Writing a AdaBoost Algorithm from scratch**

**Overview**

At the end of this article, you will be able to-

* Understand the working and math behind Ensemble ML techniques namely AdaBoost.
* Able to write the Ada boost python code from scratch

**Boosting:**

Boosting is an ensemble technique that attempts to create a strong classifier from a number of weak classifiers. Unlike many ML models which focus on high quality prediction done by a single model, boosting algorithms seek to improve the prediction power by training a sequence of weak models, each compensating the weaknesses of its predecessors. Boosting grants power to machine learning models to improve their accuracy of prediction.

**AdaBoost:**

AdaBoost is a specific Boosting algorithm developed for both classification and regression problem. AdaBoost, short for Adaptive [Boosting](https://en.wikipedia.org/wiki/Boosting_(meta-algorithm)), is a [machine learning](https://en.wikipedia.org/wiki/Machine_learning) [algorithm](https://en.wikipedia.org/wiki/Meta-algorithm) formulated by Yoav Freund and [Robert Schapire](https://en.wikipedia.org/wiki/Robert_Schapire). Ada Boost technique follows a decision tree model with depth equals to one. Ada boost is nothing but the forest of stumps rather than trees. Ada Boost works by putting more weight on difficult to classify instances and less on those already handled well.

Idea behind Ada Boost:

* Stumps are not great in making accurate classification so it is nothing but a week classifier/ weak learner. Combination of many weak classifier makes a strong classifier and this is the principle behind the Ada boost algorithm.
* Some stumps get more performance or saying in classification than others.
* Consecutive stump is made by taking the previous stumps mistakes into account.

**AdaBoost Algorithm from scratch:**

Here I have used Iris dataset as an example in building the algorithm from the scratch.

**Step 1:**

Initially assign same weights to each record in the dataset.

Sample weight = 1/N

Where N = Number of records

**Step 2:**

Draw random sample with replacement from original data with the probabilities equal to the sample weights and fit the ML model to it. Here the ML model (base learners) used in AdaBoost are decision trees. Decision trees are created with one depth which as one node and two leaves and are referred to as stumps. Fit the model to the random sample and predict the values for the original data.

**Step 3:**

Calculate the total error. Where total error is nothing but the sum of weights of misclassified record.

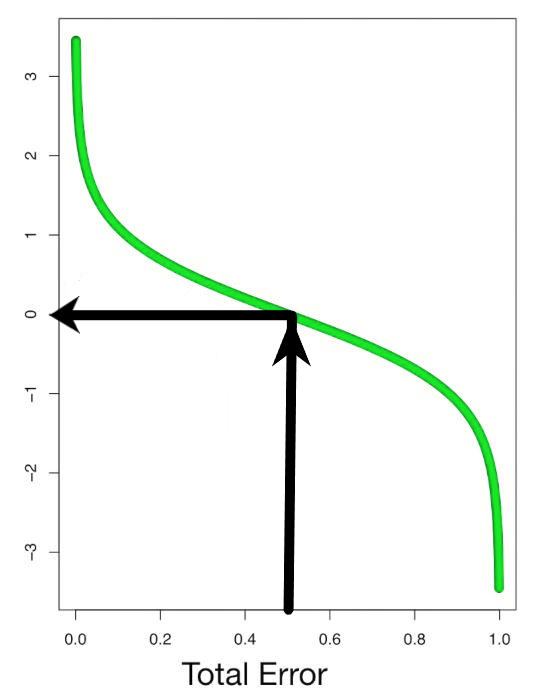
Total error = Weights of misclassified records

Total error will be always between 0 and 1. Zero represents perfect stump and one represent weak stump.

**Step 4:**

Try to find the performance of the stump. Using the Total error, determine the performance of the base learner with the following formula

Performance of the stump(α) = ½ ln (1 – Total error/Total error)



“Image by author”

Cases:

* If the total error is 0.5 then the performance of the stump will be zero.
* If the total error is zero, then the performance will become infinity.

When the total errors are equal to one or zero the above equation will behave in a weird manner. So in practice a small error term is added to prevent this from happening.

When the performance is relatively large the stump did a good job in classifying the records. When the performance is relatively low the stump did not do a good job in classifying the records. Using the performance parameter, we can increase the weights of the wrongly classified records and decrease the weights of the correctly classified records

**Step 5:**

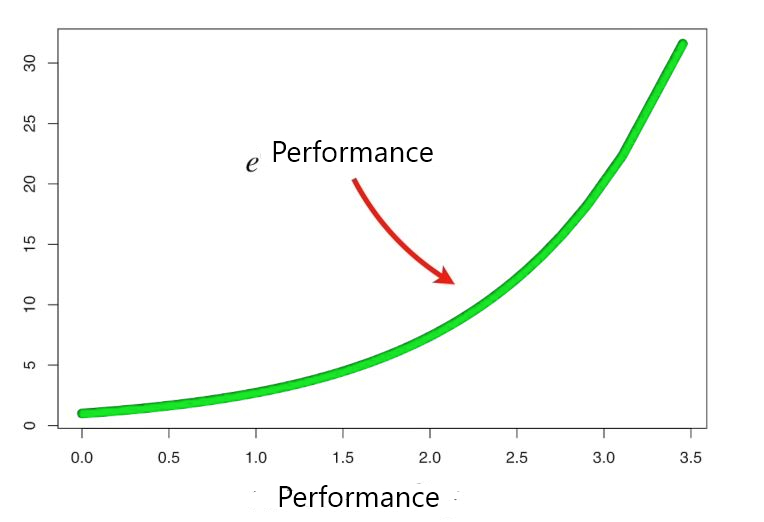
Based on the performance of the stump(α) update the weights. We need the next stump to correctly classify the misclassified record by increasing the corresponding sample weight and decreasing the sample weights of the correctly classified records.

New weight = Weight \* e^(performance) 🡪 misclassified records

New weight = Weight \* e^-(performance) 🡪 correctly classified records

Short note on **e**^(performance) i.e. for misclassification

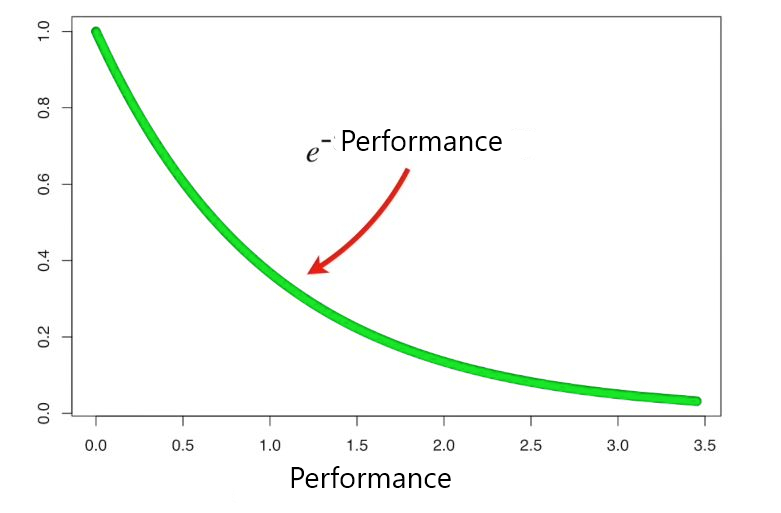
When the performance is relatively large the last stump did a good job in classifying the records now the new sample weight will be much larger than the old one. When the performance is relatively low the last stump did not do a good job in classifying the records now the new sample weight will only be little larger than the old one.



“Image by author”

Short note on e^-(performance) i.e. for no misclassification

When the performance is relatively large the last stump did a good job in classifying the records now the new sample weight will be very small than the old one. When the performance is relatively small the last stump did not do a good job in classifying the records now the new sample weight will only be little smaller than the old one.



“Image by author”

Note: Here the sum of the updated weights is not equal to one. whereas in case of initial sample weight it is equal to one. So, for this we will be dividing it by a number which is nothing but the sum of the updated weights (normalizing constant).

Normalizing constant = ∑ New weight

Normalized weight = New weight/ Normalizing constant

Now the sum of normalized weight is equal to 1.

**Step 6:**

Use the normalized weight and make the second stump in the forest. Create a new dataset of same size of the original dataset with repetition based on the newly updated sample weight. So that the misclassified records get selected much more times than the others. Repeat step 2 to step 5 by updating the weights for a particular number of iterations.

**Step 7:**

Final prediction is done by obtaining the sign of the weighted sum of final predicted value.

Final prediction/sign (weighted sum) = Sign ∑ ( \* (predicted value at each iteration))

For example: 5 weak classifiers may predict the values 1.0, 1.0, -1.0, 1.0, -1.0. From a majority vote, it looks like the model will predict a value of 1.0 or the first class. These same 5 weak classifiers may have the performance (α) values as 0.2, 0.5, 0.8, 0.2 and 0.9 respectively. Calculating the weighted sum of these predictions results in an output of -0.8, which would be an ensemble prediction of -1.0 or the second class.

**Pros and Cons:**

Pros:

* One of the many advantages of the AdaBoost Algorithm is it is fast, simple and easy to program.
* Boosting has been shown to be resistant to overfitting. This is a major plus.
* It has been extended to learning problems beyond binary classification and it is versatile as it can be used with text or numeric data.

Cons:

* AdaBoost can be sensitive to noisy data and outliers.
* Weak classifiers being too weak can lead to low margins and overfitting.

**Conclusion**

AdaBoost helps in choosing the training set for each new classifier that is trained based on the results of the previous classifier. Also, while combining the results; it determines how much weight should be given to each classifier’s proposed answer. It combines the weak learners to create a strong one to correct classification errors which is also the first successful [boosting algorithm](https://www.educba.com/boosting-algorithm/) for binary classification problems.

GitHub Link:

**Bio:** [**James Ajeeth J**](https://www.linkedin.com/in/jamesajeeth/)is a Postgraduate Student at Praxis Business School in Bangalore, India.

References:

1. Hastie, Trevor, Tibshirani, Robert, Friedman, Jerome, *The Elements of Statistical Learning, Data Mining, Inference, and Prediction, Second Edition.*
2. [Statquest](https://statquest.org/adaboost-clearly-explained/)
3. [Wikipedia](https://en.wikipedia.org/wiki/AdaBoost)