FAQ - Boosting

#### **1. What is the difference between AdaBoost and Gradient Boosting?**

AdaBoost vs. Gradient Boosting:

| **AdaBoost** | **Gradient Boosting** |
| --- | --- |
| Adaboost is more about ‘voting weights’ | Gradient boosting is more about “adding gradient optimization”. |
| Adaboost increases the accuracy by giving more weightage to the target which is misclassified by the model. | Gradient boosting increases the accuracy by minimizing the Loss Function (an error which is the difference of actual and predicted value) and having this loss as a target for the next iteration. |
| At each iteration, the Adaptive boosting algorithm changes the sample distribution by modifying the weights attached to each of the instances. | - Gradient boosting calculates the gradient (derivative) of the Loss Function with respect to the prediction (instead of the features).  - The Gradient boosting algorithm builds the first weak learner and calculates the Loss Function.  - It then builds a second learner to predict the loss after the first step. The step continues for the third learner and then for the fourth learner and so on until a certain threshold is reached. |

#### **2. When to apply AdaBoost (Adaptive Boosting Algorithm)?**

Here is the list of all the **key points** below for an **understanding of Adaboost:**

* **AdaBoost can be applied to any classification algorithm**, so it’s really a technique that builds on top of other classifiers as opposed to being a classifier itself.
* You could just train a bunch of weak classifiers on your own and combine the results.
* There’s really two things it figures out for you:
  + It helps you choose the training set for each new classifier that you train based on the results of the previous classifier.
  + It determines how much weight should be given to each classifier’s proposed answer when combining the results.

#### 

#### **3. How does AdaBoost (Adaptive Boosting Algorithm) work?**

**Important Points regarding working of Adaboost:**

* **AdaBoost can be applied to any classification algorithm**, so it’s really a technique that builds on top of other classifiers as opposed to being a classifier itself.
* Each weak classifier should be trained on a random subset of the total training set.
* AdaBoost assigns a **“weight”** to each training example, which **determines the probability that each example should appear in the training set.**
  + g. (The initial weights generally add up to 1. for example: if there are 8 training examples, the weight assigned to each will be 1/8 initially. So all training examples are having equal weights.)
* Now, if a training example is misclassified, then that training example is assigned a higher weight so that the probability of appearing that particular misclassified training example is higher in the next training set for training the classifier.
* So, after performing the previous step, hopefully, the trained classifier will perform better on the misclassified examples next time.
* So **the weights are based on increasing the probability of being chosen in the next sequential training sub-set.**

#### 

#### **4. How to install the XGBoost library in Windows or Mac operations systems?**

For Windows, run the following command in your **Jupyter notebook:**

!pip install xgboost

For Mac, run the following command in your **terminal:**

conda install -c conda-forge xgboost

Note: Open a new terminal before using the above command in the Mac terminal. To open the terminal, please open the Launchpad and then click on the terminal icon.

#### **5. I am getting the below warning while fitting XGBoost Classifier**

WARNING: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval\_metric if you'd like to restore the old behavior.

#### **How to solve it?**

To remove the warning kindly try setting the eval\_metric hyperparameter as 'logloss', as shown below:

#### **xgb = XGBClassifier(eval\_metric='logloss')**

#### 

#### **6. In the video, while fitting and tuning a model I can see all the hyperparameters listed in the output but when I run the code I can only see a few hyperparameters. Why is it so?**

This is because of version difference. In the earlier versions, Python printed all the hyperparameters in the output while fitting or tuning a model. But, after an update, only those hyperparameters are printed for which the value has been changed from the default value. Those hyperparameters where the default value is the best value after tuning, are not printed.

#### **7. Why does Adaboost gives more weightage to the weak learners?**

In AdaBoost, more weightage is given to the wrong predictions so that the subsequent model can learn from the weak learners as well. This will help the model learn better even from the weak classifiers and hence, it will improve the overall performance of the model.

Let's say x1 is wrongly predicted by the model, we adjust the weight i.e we give more importance to x1 for the subsequent training so that model will be able to identify the pattern for the weak learners.

#### **8. How are the final weighted votes calculated in the Adaboost model video?**

On top of the training data, a weak classifier is built using weighted samples. Here, the weights of each sample indicate how important it is to be correctly classified. Initially, for the first stump, we give all the samples equal weights. For each variable, we create a decision stump and test how well it classifies data into their target classes. We'd look at how many samples were labeled correctly or wrongly. The wrongly classified samples are given more weight so that they can be identified correctly in the next decision stump. Similarly, we iterate the above process for say n classifiers.

Now, for the final prediction as explained in the video, we divide the regions and based on the voting we classify a region as positive or negative. For eg, in the top left region, all the models M1, M2, and M3 predicted positive and hence the final vote is positive (+). Similarly, for the top (first row) middle region, M1 predicts negative (-), M2 and M3 predict positive (+), a majority vote is positive (+) here and hence, the final prediction is positive.