Ensemble Methods : Boosting

What is boosting

- Boosting is an ensemble learning method that combines a set of weak learners into strong learners to minimize training errors.
- In boosting, a random sample of data is selected, fitted with a model, and then trained sequentially. That is, each model tries to compensate for the weaknesses of its predecessor.
- Each classifier's weak rules are combined with each iteration to form one strict prediction rule.
- Boosting is an efficient algorithm that converts a weak learner into a strong learner.
- They use the concept of the weak learner and strong learner conversation through the weighted average values and higher votes values for prediction.
- These algorithms use decision stump and margin maximizing classification for processing.

Why we use boosting?

To solve more complex problems, we can not rely on base ML models and we require more advanced techniques.
Suppose that, given a data set of about characteristics of houses, we have to to build a model that can predict if we should purchase the house or not. We will start determining those features which can lead to decision making of purchase.

- Total Carpet area: **Purchase**
- Number of bedroom: **Purchase**
- Distance from airport/railway station: **Don't Purchase**
- Availability of essential facilities (Schools/Hospitals): **Don't purchase**
- Expected return on investment (ROI): **Purchase**

These rules help us whether to purchase the house or not.

However, the prediction would be flawed if we were make decision based on an individual (single) rule.

These rules are called weak learners because these rules are not strong enough to make the decision.

Therefore, to ensure our prediction is more accurate, we can combine the prediction from these weak learners by using the majority rule or weighted average. This makes a strong learner model.

In the above example, we have defined 5 weak learners, and the majority of these give us the prediction that we should purchase the house. Therefore, our final output is to purchase the house.

Boosting vs Bagging

Ensemble learning can be performed in two ways:

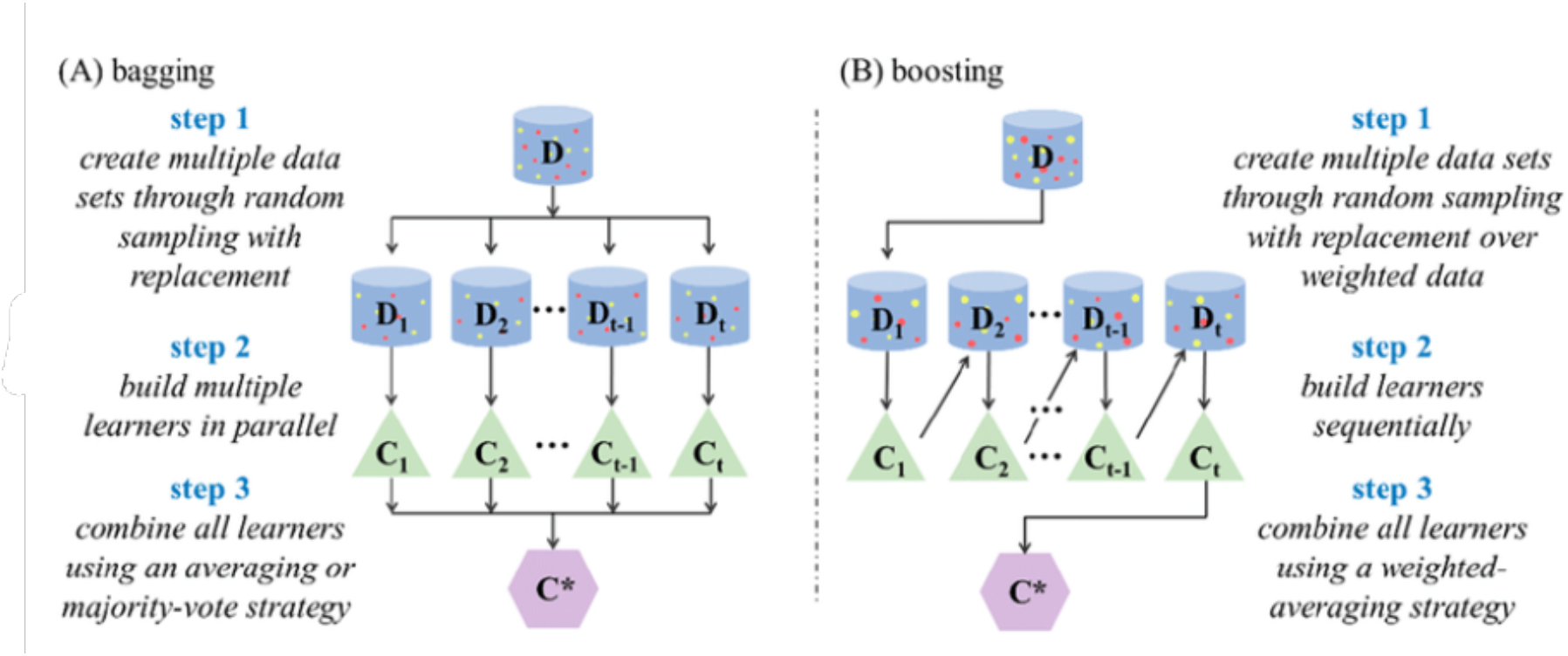
Parallel ensemble

- This is also known as bagging
- Here the weak learners are produced parallelly during the training phase.
- The performance of the model can be increased by parallelly training a number of weak learners on bootstrapped data sets.
- An example of bagging is the Random Forest algorithm.

Sequential ensemble

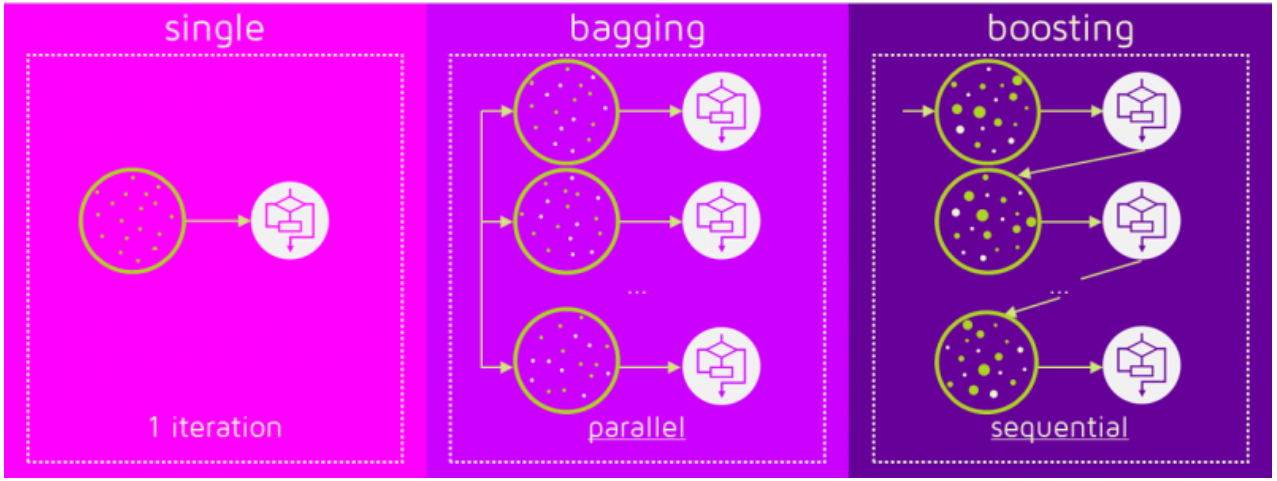
- This is popularly known as boosting,
- Here the weak learners are sequentially produced during the training phase.
- The performance of the model is improved by assigning a higher weightage to the previous, incorrectly classified samples.
- An example of boosting is the AdaBoost algorithm.

Boosting vs Bagging



How boosting Algorithm works

- The basic principle behind the working of the boosting algorithm is to generate multiple weak learners and combine their predictions to form one strong rule.
- These weak rules are generated by applying base Machine Learning algorithms on different distributions of the data set.
- These algorithms generate weak rules for each iteration.
- After multiple iterations, the weak learners are combined to form a strong learner that will predict a more accurate outcome.



How boosting works

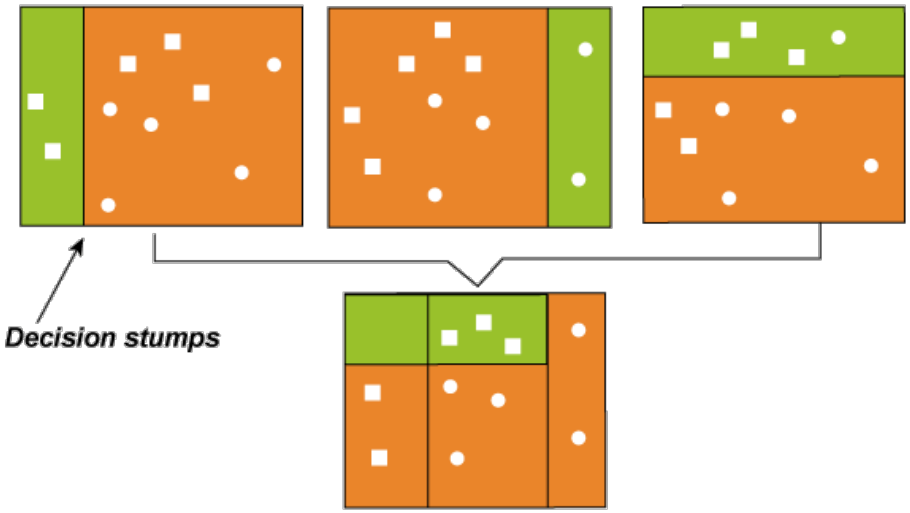
Here's how the algorithm works:

Step 1: The base algorithm reads the data and assigns equal weight to each sample observation.

Step 2: False predictions made by the base learner are identified. In the next iteration, these false predictions are assigned to the next base learner with a higher weightage on these incorrect predictions.

Step 3: Repeat step 2 until the algorithm can correctly classify the output.

How boosting works



Finally, it combines the outputs from weak learner and creates a strong learner which eventually improves the prediction power of the model.

Boosting pays higher focus on examples which are mis-classified or have higher errors by preceding weak rules.

Types of Boosting

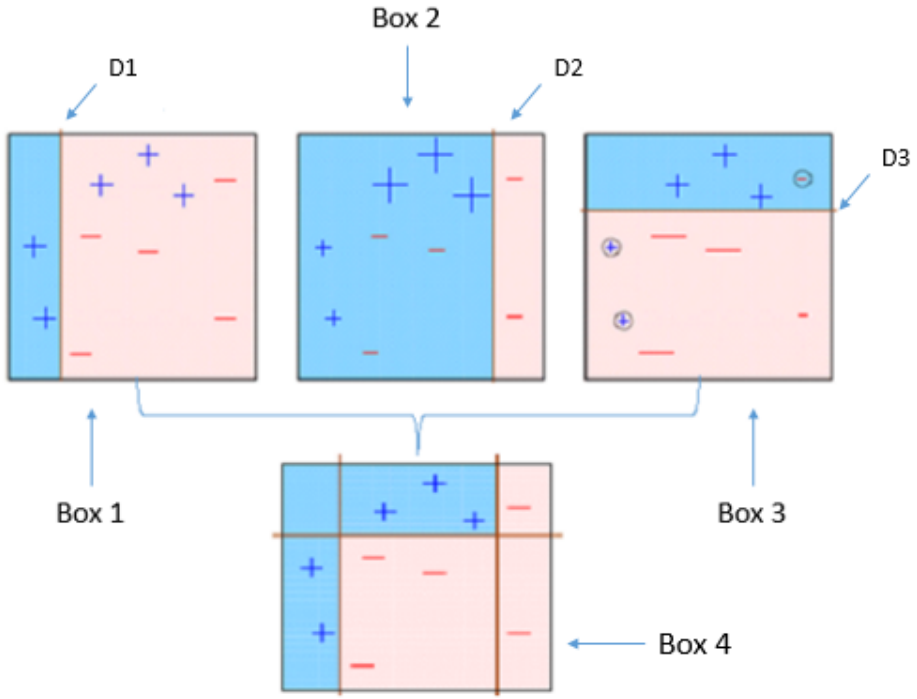
Boosting methods are focused on iteratively combining weak learners to build a strong learner that can predict more accurate outcomes. As a reminder, a weak learner classifies data slightly better than random guessing.

Boosting algorithms can differ in how they create and aggregate weak learners during the sequential process.

Three popular types of boosting methods include:

- **AdaBoost (Adaptive Boosting)**
- **Gradient Tree Boosting**
- **XGBoost**

AdaBoost



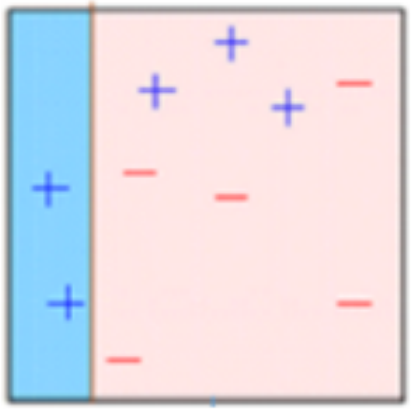
This diagram above explains Ada-boost at very high level.

AdaBoost

Box 1: You can see that we have assigned equal weights to each data point and applied a decision stump to classify them as + (plus) or – (minus).

The decision stump (D1) has generated vertical line at left side to classify the data points. We see that, this vertical line has incorrectly predicted three + (plus) as – (minus).

In such case, we'll assign higher weights to these three + (plus) and apply another decision stump.



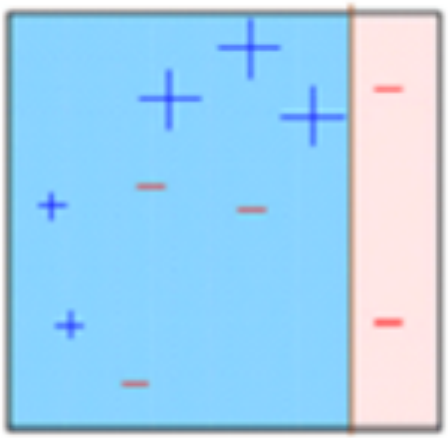
AdaBoost

Box 2: Here, you can see that the size of three incorrectly predicted + (plus) is bigger as compared to rest of the data points.

In this case, the second decision stump (D2) will try to predict them correctly.

Now, a vertical line (D2) at right side of this box has classified three mis-classified + (plus) correctly. But again, it has caused mis-classification errors.

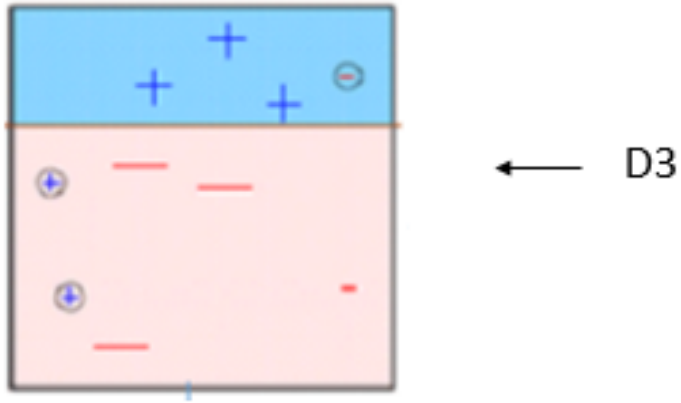
This time with three -(minus). Again, we will assign higher weight to three – (minus) and apply another decision stump.



AdaBoost

Box 3: Here, three – (minus) are given higher weights. A decision stump (D3) is applied to predict these mis-classified observation correctly.

This time a horizontal line is generated to classify + (plus) and – (minus) based on higher weight of mis-classified observation.



AdaBoost

Box 4: Here, we have combined D1, D2 and D3 to form a strong prediction having complex rule as compared to individual weak learner.

You can see that this algorithm has classified these observation quite well as compared to any of individual weak learner.

AdaBoost

- AdaBoost fits a sequence of weak learners on different weighted training data.
- It starts by predicting original data set and gives equal weight to each observation. If prediction is incorrect using the first learner, then it gives higher weight to observation which have been predicted incorrectly.
- Being an iterative process, it continues to add learner(s) until a limit is reached in the number of models or accuracy.
- Mostly, we use decision stumps with AdaBoost. But, we can use any machine learning algorithms as base learner if it accepts weight on training data set. We can use AdaBoost algorithms for both classification and regression problem.

Gradient Boost

- Gradient Boosting is also based on sequential ensemble learning. Here the base learners are generated sequentially so that the present base learner is always more effective than the previous one, i.e., and the overall model improves sequentially with each iteration.
- The difference in this boosting type is that the weights for misclassified outcomes are not incremented. Instead, the Gradient Boosting method tries to optimize the loss function of the previous learner by adding a new model that adds weak learners to reduce the loss function.

Gradient Boost

The main idea here is to overcome the errors in the previous learner's predictions. Gradient boosting has three main components:

Loss function: The use of the loss function depends on the type of problem. The advantage of gradient boosting is that there is no need for a new boosting algorithm for each loss function.

Weak learner: In gradient boosting, decision trees are used as a weak learners. A regression tree is used to give true values, which can combine to create correct predictions. Like in the AdaBoost algorithm, small trees with a single split are used, i.e., decision stump. Larger trees are used for large levels

Additive Model: Trees are added one at a time in this model. Existing trees remain the same. During the addition of trees, gradient descent is used to minimize the loss function.

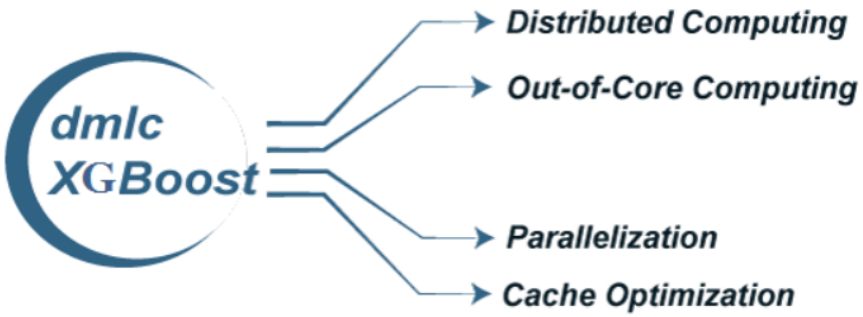
Like AdaBoost, Gradient Boosting can also be used for classification and regression problems.

XGBoost

XGBoost is an advanced gradient boosting method. XGBoost, falls under the Distributed Machine Learning Community (DMLC) category.

The main aim of this algorithm is to increase the speed and efficiency of computation. The Gradient Descent Boosting algorithm computes the output slower since they sequentially analyze the data set. Therefore XGBoost is used to boost or extremely boost the model's performance.

XGBoost is designed to focus on computational speed and model efficiency. The main features provided by XGBoost are:



XGBoost

Parallel Processing: XG Boost provides Parallel Processing for tree construction which uses CPU cores while training.

Cross-Validation: XG Boost enables users to run cross-validation of the boosting process at each iteration, making it easy to get the exact optimum number of boosting iterations in one run.

Cache Optimization: It provides Cache Optimization of the algorithms for higher execution speed.

Distributed Computing: For training large models, XG Boost allows Distributed Computing.

Benefits of Boosting

The boosting method presents many advantages and challenges for classification or regression problems. The benefits of boosting include:

Ease of Implementation: Boosting can be used with several hyper-parameter tuning options to improve fitting. No data preprocessing is required, and boosting algorithms have built-in routines to handle missing data. In Python, the sci-kit-learn library of ensemble methods makes it easy to implement the popular boosting methods, including AdaBoost, XGBoost, etc.

Reduction of bias: Boosting algorithms combine multiple weak learners in a sequential method, iteratively improving upon observations. This approach can help to reduce high bias, commonly seen in shallow decision trees and logistic regression models.

Computational Efficiency: Since boosting algorithms have special features that increase their predictive power during training, it can help reduce dimensionality and increase computational efficiency.

Challenges of Boosting

Overfitting: There's some dispute in the research around whether or not boosting can help reduce overfitting or make it worse. We include it under challenges because in the instances that it does occur, predictions cannot be generalized to new datasets.

Intense computation: Sequential training in boosting is hard to scale up. Since each estimator is built on its predecessors, boosting models can be computationally expensive, although XGBoost seeks to address scalability issues in other boosting methods. Boosting algorithms can be slower to train when compared to bagging, as a large number of parameters can also influence the model's behavior.

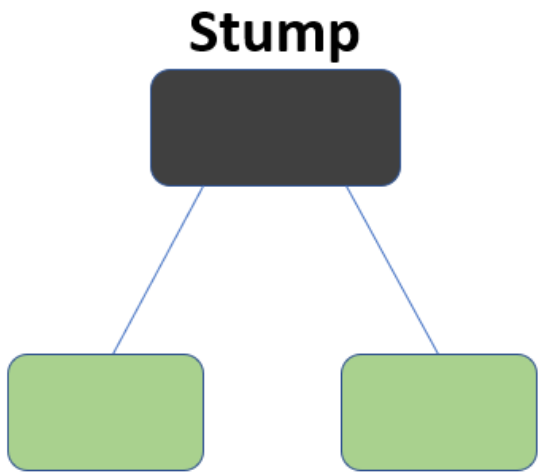
Vulnerability to outlier data: Boosting models are vulnerable to outliers or data values that are different from the rest of the dataset. Because each model attempts to correct the faults of its predecessor, outliers can skew results significantly.

Real-time implementation: You might find it challenging to use boosting for real-time implementation because the algorithm is more complex than other processes. Boosting methods have high adaptability, so you can use various model parameters that immediately affect the model's performance.

AdaBoost

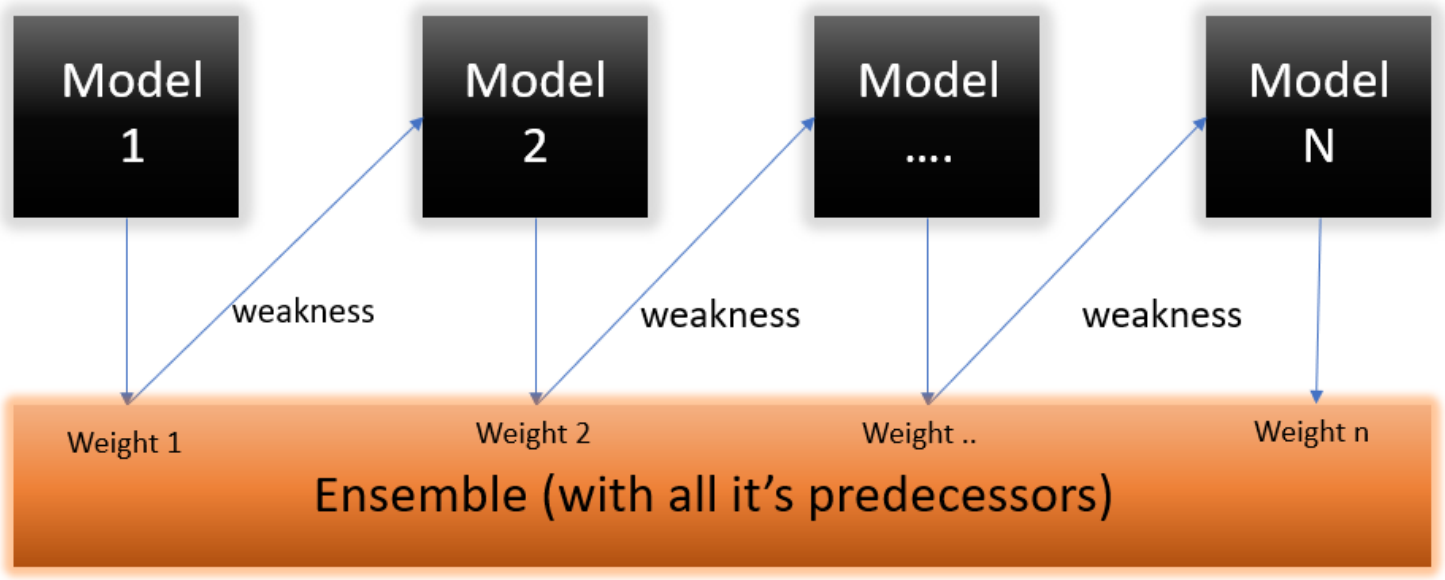
What is AdaBoost

- AdaBoost also called Adaptive Boosting is a technique in Machine Learning used as an Ensemble Method. The most common algorithm used with AdaBoost is decision trees with one level that means with **Decision trees with only 1 split**. These trees are also called Decision **Stumps**.



What is AdaBoost

- AdaBoost builds a model and then gives equal weights to all the data points. It then assigns higher weights to points that are wrongly classified. Now all the points which have higher weights are given more importance in the next model. It will keep training models until and unless a lower error is received.



Working of AdaBoost

Step 1 : Assigning weights to each observations

The dataset is shown below. Since the target column is binary it is a classification problem.

First of all these data points will be assigned some weights. Initially, all the weights will be equal.

Row No.	Gender	Age	Income	Illness	Sample Weights
1	Male	41	40000	Yes	1/5
2	Male	54	30000	No	1/5
3	Female	42	25000	No	1/5
4	Female	40	60000	Yes	1/5
5	Male	46	50000	Yes	1/5

The formula to calculate the sample weights is: $w(x_i, y_i) = \frac{1}{N}, \; i = 1, 2, \ldots . n$

Where N is the total number of observations in the dataset.

Here since we have 5 data points so the sample weights assigned will be 1/5.

Working of AdaBoost

Step 2 – Identification of first stump

We start by seeing how well “Gender” classifies the samples and will see how the variables (Age, Income) classifies the samples.

We’ll create a decision stump for each of the features and then calculate the Gini Index of each tree.

The tree with the lowest Gini Index will be our first stump.

Here in our dataset let’s say Income has the lowest gini index so it will be our first stump.

Gender = Male	
2 Yes 1 No	1 Yes 1 No

Age<=44	
2 Yes 1 No	1 Yes 1 No

Income <= 35000	
2 No	3 Yes

Working of AdaBoost

Step 3 – Calculation of amount of say

We'll now calculate the **Amount of Say** or **Importance** or **Influence** for this classifier in classifying the datapoints using this formula:

$$\frac{1}{2}\log\frac{1-Total\ Error}{Total\ Error}$$

The total error is nothing, but the summation of all the sample weights of misclassified data points.

Here in our dataset let's assume there is 1 wrong output, so our total error will be 1/5

Row No	Gender	Age	Income	Illness	Sample weights	Prediction
1	Male	41	40000	Yes	1/5	Yes
2	Male	54	30000	No	1/5	No
3	Female	42	25000	No	1/5	No
4	Female	40	60000	Yes	1/5	No
5	Male	46	50000	Yes	1/5	Yes

Working of AdaBoost

Step 3 – Calculation of amount of say

$$Performance\ of\ the\ stump\ =\ \frac{1}{2}\log_e(\frac{1-Total\ Error}{Total\ Error})$$

$$\alpha\ =\ \frac{1}{2}\log_e\left(\frac{1-\frac{1}{5}}{\frac{1}{5}}\right)$$

$$\alpha\ =\ \frac{1}{2}\log_e\left(\frac{0.8}{0.2}\right)$$

$$\alpha\ =\ \frac{1}{2}\log_e(4)\ =\ \frac{1}{2}*(1.38)$$

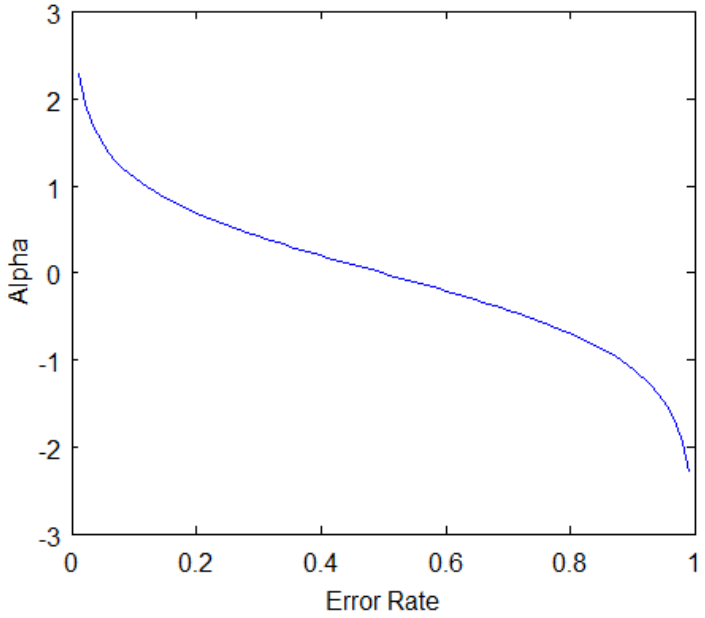
$$\alpha\ =\ 0.69$$

Note: Total error will always be between 0 and 1 where 0 Indicates perfect stump and 1 indicates horrible stump.

Working of AdaBoost

Step 3 – Calculation of amount of say

- From the adjacent graph we can see that **when there is no misclassification** then we have no error (Total Error = 0), so the **amount of say (alpha)** will be a large number.
- When the **classifier predicts half right and half wrong** then the **Total Error = 0.5** and the importance (amount of say) of the classifier will be 0.
- If **all the samples have been incorrectly classified** then the **error will be very high (approx. to 1)** and hence our alpha value will be a negative real number.



Working of AdaBoost

Step 4 – Update the weights

Why should we update the weights?

- We need to update the weights because if the same weights are applied to the next model, then the output received will be the same as what was received in the first model.
- The wrong predictions will be given more weight whereas the correct predictions weights will be decreased.
- Now when we build our next model after updating the weights, more preference will be given to the points with higher weights.
- After finding the importance of the classifier and total error we need to finally update the weights.
- For this, we use the following formula:

$$New\ sample\ weight\ =\ old\ weight\ *\ e^{\pm Amount\ of\ say\ (\alpha)}$$

Working of AdaBoost

Step 4 – Update the weights

- The amount of say (alpha) will be negative when the sample is correctly classified.
- The amount of say (alpha) will be positive when the sample is miss-classified.
- There are four correctly classified samples and 1 wrong, here the sample weight of that datapoint is 1/5 and the amount of say/performance of the stump of Income is 0.69.

Working of AdaBoost

Step 4 – Update the weights

- New weights for correctly classified samples are:

$$New\ sample\ weight\ =\ \frac{1}{5}\ *\ \exp(-0.69)$$

$$New\ sample\ weight\ =\ 0.2\ *\ 0.502\ =\ 0.1004$$

- For wrongly classified samples the updated weights will be:

$$New\ sample\ weight\ =\ \frac{1}{5}\ *\ \exp(0.69)$$

$$New\ sample\ weight\ =\ 0.2\ *\ 1.994\ =\ 0.3988$$

Note: See the sign of alpha.
The alpha is negative when the data point is correctly classified, and this decreases the sample weight from 0.2 to 0.1004.
It is positive when there is misclassification, and this will increase the sample weight from 0.2 to 0.3988

Working of AdaBoost

Step 4 – Update the weights

Row No.	Gender	Age	Income	Illness	Sample Weights	New Sample Weights
1	Male	41	40000	Yes	1/5	0.1004
2	Male	54	30000	No	1/5	0.1004
3	Female	42	25000	No	1/5	0.1004
4	Female	40	60000	Yes	1/5	0.3988
5	Male	46	50000	Yes	1/5	0.1004

- The total sum of the sample weights must be equal to 1 but here if we sum up all the new sample weights, we will get 0.8004.
- To bring this sum equal to 1 we will normalize these weights by dividing all the weights by the total sum of updated weights that is 0.8004.
- So, after normalizing the sample weights we get this dataset and now the sum is equal to 1.:

Working of AdaBoost

Row No.	Gender	Age	Income	Illness	Sample Weights	New Sample Weights
1	Male	41	40000	Yes	1/5	0.1004/0.8004 =0.1254
2	Male	54	30000	No	1/5	0.1004/0.8004 =0.1254
3	Female	42	25000	No	1/5	0.1004/0.8004 =0.1254
4	Female	40	60000	Yes	1/5	0.3988/0.8004 =0.4982
5	Male	46	50000	Yes	1/5	0.1004/0.8004 =0.1254

Now in the above image we can see that the weights are adjusted (dividing by sum of weights) so that sum of the weights should be 1

Working of AdaBoost

Step 5 – Creation of Buckets based on weights

- Now we need to make a new dataset to see if the errors decreased or not.
- For this we will remove sample weights column and we will divide our data points into **buckets** based on the new sample weights.

Row No.	Gender	Age	Income	Illness	New Sample Weights	Buckets
1	Male	41	40000	Yes	0.1004/0.8004= 0.1254	0 to 0.1254
2	Male	54	30000	No	0.1004/0.8004= 0.1254	0.1254 to 0.2508
3	Female	42	25000	No	0.1004/0.8004= 0.1254	0.2508 to 0.3762
4	Female	40	60000	Yes	0.3988/0.8004= 0.4982	0.3762 to 0.8744
5	Male	46	50000	Yes	0.1004/0.8004= 0.1254	0.8744 to 0.9998

Working of AdaBoost

Step 6 – Creation of data for next iteration

- In this step the algorithm selects random numbers from 0-1.
- Since incorrectly classified records have higher sample weights, the probability to select those records is very high.
- Suppose the 5 random numbers our algorithm take is 0.38,0.22,0.29,0.76,0.55.
- According to random numbers, we'll create our new dataset shown below.

Row No.	Gender	Age	Income	Illness
1	Female	40	60000	Yes
2	Male	54	30000	No
3	Female	42	25000	No
4	Female	40	60000	Yes
5	Female	40	60000	Yes

- We can see the datapoint which was wrongly classified has been selected 3 times because it has a higher weight.

Working of AdaBoost

Step 7 – Repeat the above steps

Now this act as our new dataset and we need to repeat all the above steps i.e.

1) Assign equal weights to all the datapoints
2) Find the stump that does the best job classifying the new collection of samples by finding their Gini Index and selecting the one with the lowest Gini index
3) Calculate the “Amount of Say” and “Total error” to update the previous sample weights.
4) Normalize the new sample weights.

Iterate through these steps until and unless a low training error is achieved.

Suppose with respect to our dataset we have constructed 3 decision trees (DT1, DT2, DT3) in a sequential manner. If we send our test data now it will pass through all the decision trees and finally, we will see which class has higher sum of amount of say, and based on that we will do predictions for our test dataset.

Sklearn Important Parameters

base_estimator : default=None

The base estimator from which the boosted ensemble is built. If None, then the base estimator is DecisionTreeClassifier initialized with max_depth=1.

n_estimators : default=50

The maximum number of estimators at which boosting is terminated.

learning_rate : default=1.0

Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier.

algorithm : {'SAMME', 'SAMME.R'}, default='SAMME.R'

SAMME. R uses the probability estimates to update the additive model, while SAMME uses the classifications only.

random_state: default=None

Controls the random seed given at each base_estimator at each boosting iteration.

Pass an int for reproducible output across multiple function calls.