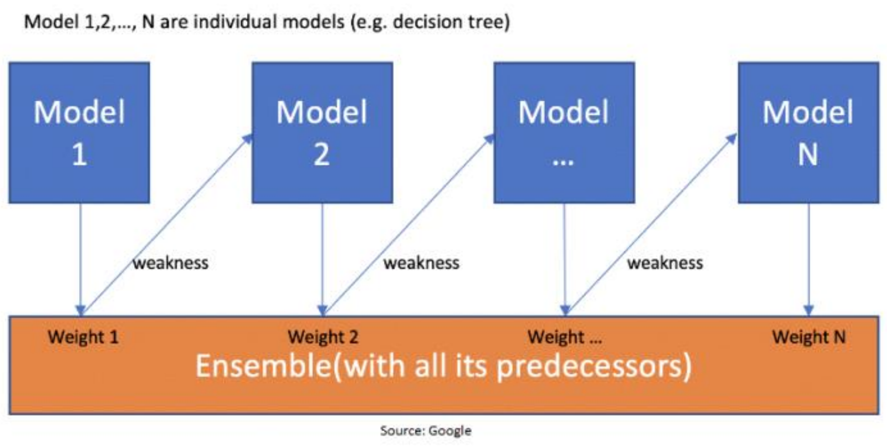
AdaBoost

26 May 2021 16:53

AdaBoost algorithm, short for Adaptive Boosting, is a Boosting technique that is used as an Ensemble Method in Machine Learning. It is called Adaptive Boosting as the weights are re-assigned to each instance, with higher weights to incorrectly classified instances. Boosting is used to reduce bias as well as the variance for supervised learning. It works on the principle where learners are grown sequentially. Except for the first, each subsequent learner is grown from previously grown learners. In simple words, weak learners are converted into strong ones. Adaboost algorithm also works on the same principle as boosting, but there is a slight difference in working. Let's discuss the difference in detail.

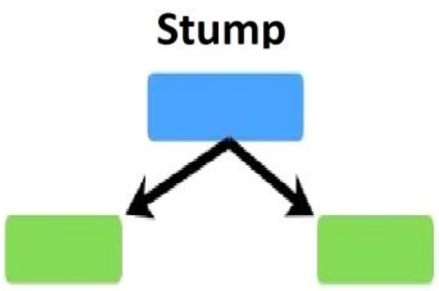
How AdaBoost Works?

First, let us discuss the working of boosting. It makes *n* number of decision trees during the training period of data. As the first decision tree/model is made, the record which is incorrectly classified during the first model is given more priority. Only these records are sent as input for the second model. The process will go on until we specify a number of base learners we want to create. Remember, the repetition of records is allowed with all boosting techniques.



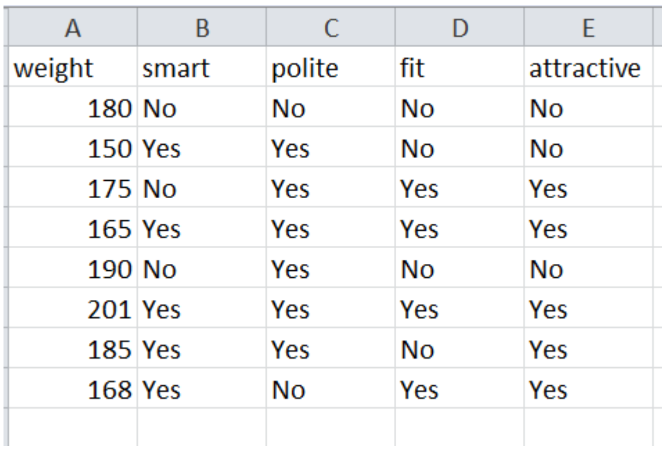
This figure shows that when the first model is made and the errors from the first model are noted by the algorithm, the record which is incorrectly classified is given as the input for the next model. This process is repeated until the specified condition is met. As you can see in the figure, there are *n* number of models made by taking the errors from the previous model. This is how boosting works. The models 1,2, 3,…, N are individual models that can be known as decision trees. All types of boosting models work on the same principle.

Since we know the boosting principle, it will be easy to understand the AdaBoost algorithm. Let's deep dive into the working of Adaboost. When the random forest is used, the algorithm makes *n* number of trees. It makes proper trees that consist of a start node with several leaves nodes. Some trees might be bigger than others, but there is no fixed depth in a random forest. But with Adaboost, that's not the case. In AdaBoost, the algorithm only makes a node with two leaves, and this is known as Stump.



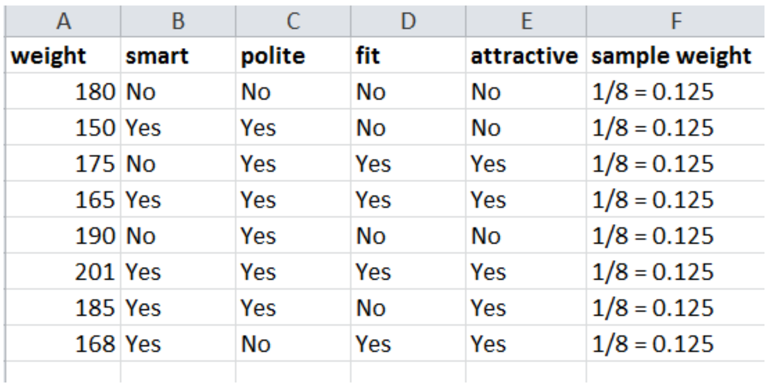
Source: Google

The figure here represents the stump. It can be seen clearly that it has only one node with only two leaves. These stumps are weak learners, and boosting techniques prefer this. The order of stumps is very important in AdaBoost. The error of the first stump influences how the other stump is made. Let’s understand this with an example.



A	B	C	D	E
weight	smart	polite	fit	attractive
180	No	No	No	No
150	Yes	Yes	No	No
175	No	Yes	Yes	Yes
165	Yes	Yes	Yes	Yes
190	No	Yes	No	No
201	Yes	Yes	Yes	Yes
185	Yes	Yes	No	Yes
168	Yes	No	Yes	Yes

Here I have created a sample dataset that consists of only **four** features, and the output is in categorical form. The image shows the actual representation of the dataset. As the output is in binary/categorical form, it becomes a classification problem. In real life, the dataset can have any number of records and features in it. Let us consider 8 samples for explanation purposes. The output is in categorical form and, here it’s **Yes or No**. All these records will get a sample weight. To assign some sample weight, the formula used is, W=1/N where N is the number of records. In this dataset, there are only 8 records, so the sample weight becomes 1/8 initially. Every record gets the same weight. In this case, it’s 1/8.

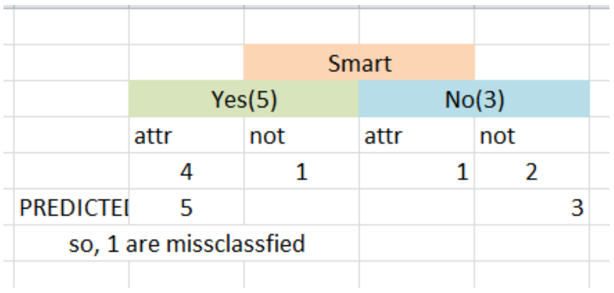


A	B	C	D	E	F
weight	**smart**	**polite**	**fit**	**attractive**	**sample weight**
180	No	No	No	No	1/8 = 0.125
150	Yes	Yes	No	No	1/8 = 0.125
175	No	Yes	Yes	Yes	1/8 = 0.125
165	Yes	Yes	Yes	Yes	1/8 = 0.125
190	No	Yes	No	No	1/8 = 0.125
201	Yes	Yes	Yes	Yes	1/8 = 0.125
185	Yes	Yes	No	Yes	1/8 = 0.125
168	Yes	No	Yes	Yes	1/8 = 0.125

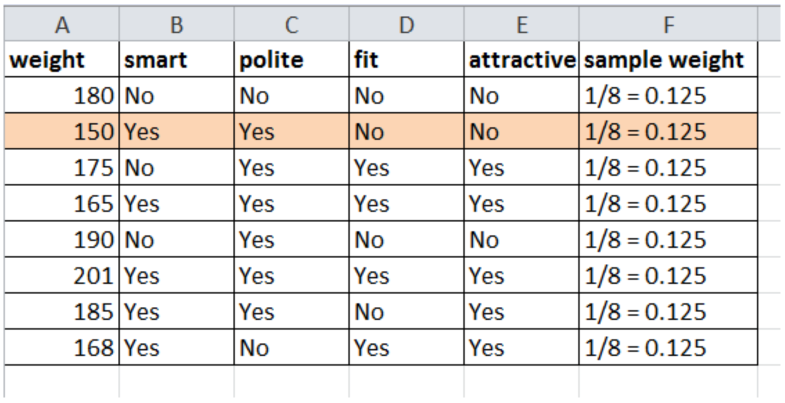
==

Step 1 – Creating First Base Learner

Now it’s time to create the **first base learner**. The algorithm takes the first feature, i.e., feature 1, and creates the first stump f1. It will create the same number of stumps as the number of features. Here, it will create 4 stumps as there are only 4 features in this dataset. From all these stumps it will create 4 decision trees and can be called stumps base learner model. Out of these 4 models, the algorithm selects only one. For selecting a base learner, there are two properties, those are, Gini and Entropy. We must calculate Gini or Entropy the same way it is calculated for decision trees. The stump that has the least value will be the first base learner. The number below the leaves represents the correctly and incorrectly classified records. By using these records, the Gini or entropy index is calculated. The stump that has the least entropy or Gini will be selected for the base learner. Let’s assume that the entropy index is the least for stump 2. So, let’s take stump 2, i.e., feature 2(smart) as our first base learner.



		Smart		
	Yes(5)		No(3)	
	attr	not	attr	not
	4	1	1	2
PREDICTEI	5			3
so, 1 are missclassfied				



A	B	C	D	E	F
weight	smart	polite	fit	attractive	sample weight
180	No	No	No	No	1/8 = 0.125
150	Yes	Yes	No	No	1/8 = 0.125
175	No	Yes	Yes	Yes	1/8 = 0.125
165	Yes	Yes	Yes	Yes	1/8 = 0.125
190	No	Yes	No	No	1/8 = 0.125
201	Yes	Yes	Yes	Yes	1/8 = 0.125
185	Yes	Yes	No	Yes	1/8 = 0.125
168	Yes	No	Yes	Yes	1/8 = 0.125

Here, feature (f2) has classified 7 records correctly and 1 incorrectly. The row in the figure that is marked red is incorrectly classified. For this, we will be calculating the total error.

==

Step 2 – Calculating the Total Error (TE)

The total error is the sum of all the errors in the classified record for sample weights. In our case, there is only 1 error, so **Total Error (TE) = 1/8**. if we have multiple errors then number of sum of all errors divide by total sample.

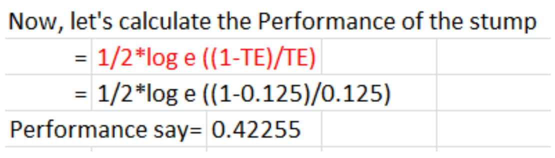
==

Step 3 – Calculating Performance of Stump

Formula for calculating Performance of Stump is: –

$$\text{Performance of Stump} = 1/2 \ln \left[\frac{1-TE}{TE} \right]$$

where, **ln** is natural log and **TE** is Total Error.



Now, let's calculate the Performance of the stump	
=	1/2*log e ((1-TE)/TE)
=	1/2*log e ((1-0.125)/0.125)
Performance say=	0.42255

In our case, TE is **1/8**. By putting the value of total error in the above formula and after solving, we get the value for the performance of Stump as **0.42255**. You must be wondering why it's necessary to calculate the TE and performance of stump? The answer is, we must update the sample weight before proceeding for the next model or stage because if the same weight is applied, we receive the output from the first model. In **boosting**, only the **wrong /incorrectly classified records** got more preference than the correctly classified records. Thus, only the wrong records from the decision tree/stump are passed on to another stump. While in **AdaBoost**, both records were allowed to pass, the **wrong records are repeated more than the correct ones**. We must increase the weight for the wrongly classified records and decrease the weight for the correctly classified records. In the next step, we will be updating the weights based on the performance of the stump.

==

Step 4 – Updating Weights

For **incorrectly classified records** the formula is:

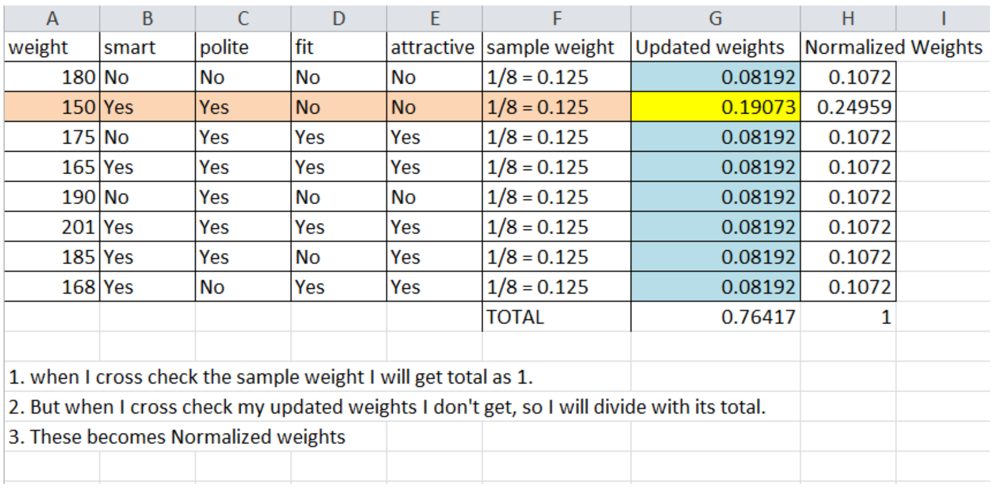
New Sample Weight = Sample Weight * $e^{(\text{Performance})}$

In our case Sample weight = 1/8   so, **1/8 * $e^{(0.42255)}$ = 0.19073**

And for **correctly classified records,** we use the same formula with a **negative sign** with performance, so that the weight for correctly classified records will reduce compared to the incorrect classified ones. The formula is:

New Sample Weight = Sample Weight * $e^{(\text{Performance})}$

Putting the values, **1/8 * $e^{-(0.42255)}$ = 0.08192**



A	B	C	D	E	F	G	H	I
weight	smart	polite	fit	attractive	sample weight	Updated weights	Normalized Weights	
180	No	No	No	No	1/8 = 0.125	0.08192	0.1072	
150	Yes	Yes	No	No	1/8 = 0.125	0.19073	0.24959	
175	No	Yes	Yes	Yes	1/8 = 0.125	0.08192	0.1072	
165	Yes	Yes	Yes	Yes	1/8 = 0.125	0.08192	0.1072	
190	No	Yes	No	No	1/8 = 0.125	0.08192	0.1072	
201	Yes	Yes	Yes	Yes	1/8 = 0.125	0.08192	0.1072	
185	Yes	Yes	No	Yes	1/8 = 0.125	0.08192	0.1072	
168	Yes	No	Yes	Yes	1/8 = 0.125	0.08192	0.1072	
					TOTAL	0.76417	1	

1. when I cross check the sample weight I will get total as 1.
2. But when I cross check my updated weights I don't get, so I will divide with its total.
3. These becomes Normalized weights

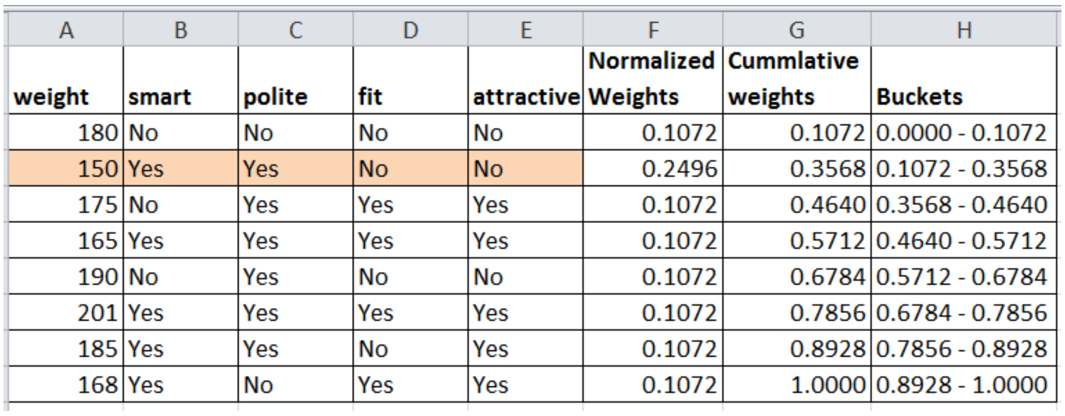
The updated weight for all the records can be seen in the figure. As is known, the total sum of all the weights should be 1. But in this case, one can see that the total updated weight of all the records is not 1, it's 0.76417. To make the total sum 1, one must divide every updated weight by the total sum of updated weight. For example, if our updated weight is 0.19073 and we divide this by 0.76417, i.e. **0.19073/0.76417 = 0.24959 .**

0.24959 can be known as the normalized weight. In the below figure, we can see all the normalized weight and their sum is approximately 1

==

Step 5 – Creating New Dataset

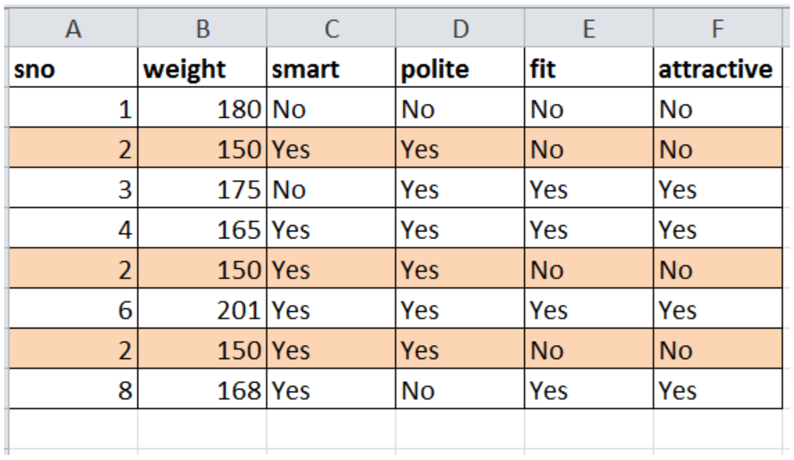
Now, it's time to create a new dataset from our previous one. In the new dataset, the frequency of **incorrectly classified records will be more than the correct ones**. While considering these normalized weights, we have to create a new dataset and that dataset is based on normalized weights. It will probably select the wrong records for training purposes. That will be the second decision tree/stump. To make a new dataset based on normalized weight, the algorithm will divide it into buckets.



A	B	C	D	E	F	G	H
weight	smart	polite	fit	attractive	Normalized Weights	Cummlative weights	Buckets
180	No	No	No	No	0.1072	0.1072	0.0000 - 0.1072
150	Yes	Yes	No	No	0.2496	0.3568	0.1072 - 0.3568
175	No	Yes	Yes	Yes	0.1072	0.4640	0.3568 - 0.4640
165	Yes	Yes	Yes	Yes	0.1072	0.5712	0.4640 - 0.5712
190	No	Yes	No	No	0.1072	0.6784	0.5712 - 0.6784
201	Yes	Yes	Yes	Yes	0.1072	0.7856	0.6784 - 0.7856
185	Yes	Yes	No	Yes	0.1072	0.8928	0.7856 - 0.8928
168	Yes	No	Yes	Yes	0.1072	1.0000	0.8928 - 1.0000

So, our first bucket is from **0 – 0.1072,** second will be from **0.1072 – 0.3568(0.1072+0.2496),** third will be from **0.3568 – 0.4640(0.3568 +0.1.72),** and so on. After this the algorithm will run 8 iterations to select different-different records from the older dataset. Suppose, in 1st iteration, the algorithm will take a random value **0.46,** then it will go and see in which bucket that value falls and selects that records in the new dataset, then again it will select a random value and see in which bucket it is and select that record for the new dataset and the same process is repeated for 8 times.

There is a high probability for the wrong records to get selected several times. This will be the new dataset. It can be seen in the below image that row number 2 has been selected multiple times from the older dataset as that row is incorrectly classified in the previous dataset.



A	B	C	D	E	F
sno	**weight**	**smart**	**polite**	**fit**	**attractive**
1	180	No	No	No	No
2	150	Yes	Yes	No	No
3	175	No	Yes	Yes	Yes
4	165	Yes	Yes	Yes	Yes
2	150	Yes	Yes	No	No
6	201	Yes	Yes	Yes	Yes
2	150	Yes	Yes	No	No
8	168	Yes	No	Yes	Yes

Based on this new dataset, the algorithm will again create a new decision tree/stump and it will repeat the same process from **step 1** till it sequentially passes through all stumps and finds that there is less error when compared with normalized weight that we had in the initial stage.

So these process will be keeps on running until number of trees/n_iterators is completed and you has to verify where you get the least test error.

==