AdaBoost

Types of Boosting Algorithms

AdaBoost (Adaptive Boosting)

Gradient Tree Boosting

XGBoost

AdaBoost (Adaptive Boosting)

AdaBoost (Adaptive Boosting) is a very popular **boosting technique** that aims at **combining multiple weak classifiers** to build **one strong classifier**.

Original Paper: https://www.semanticscholar.org/paper/Experiments-with-a-New-Boosting-Algorithm-Freund-Schapire/68c1bfe375dde46777fe1ac8f3636fb651e3f0f8

A **single classifier** may **not be able** to **accurately predict the class of an object**, but when **we group multiple weak classifiers** with each one progressively learning from the others' wrongly classified objects, we can build one such strong model.

The **classifier** mentioned here could be any of your basic classifiers, from Decision Trees (often the default) to Logistic Regression, etc.

Rather than **being a model** in itself, **AdaBoost** can be **applied on top of any classifier** to learn from its shortcomings and propose a more accurate model.

It is usually called the **“best out-of-the-box classifier”** for this reason.

"weak" classifier?

A **weak classifier** is one **that performs better than random guessing**, but **still performs poorly** at **designating classes to objects**.

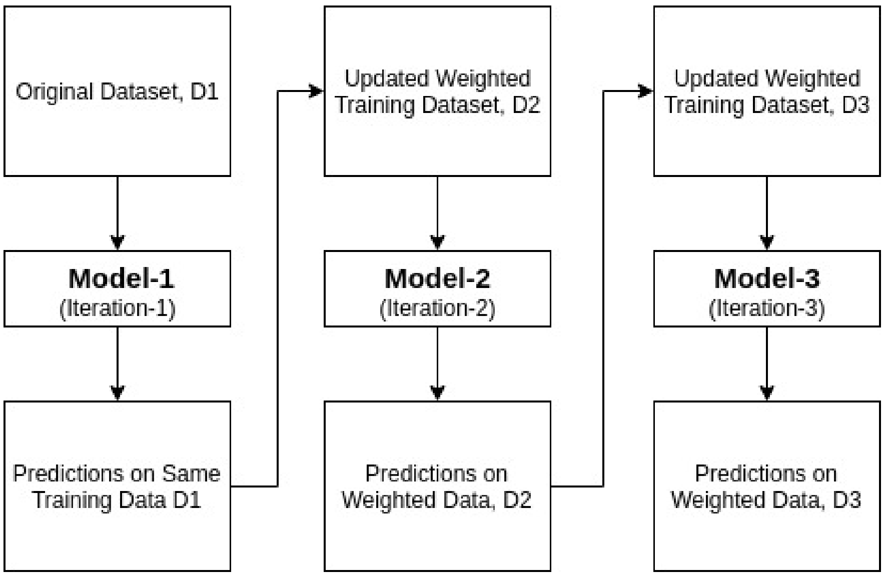
How do ADA Boost classifier works?

The **basic concept** behind **Adaboost** is to **set the weights of classifiers and training the data sample** in each **iteration** such that it **ensures** the **accurate predictions** of **unusual observations**.

Any machine learning algorithm can be used as base classifier if it accepts weights on the training set.

1.The **classifier** should be **trained interactively** on various **weighed training examples**.

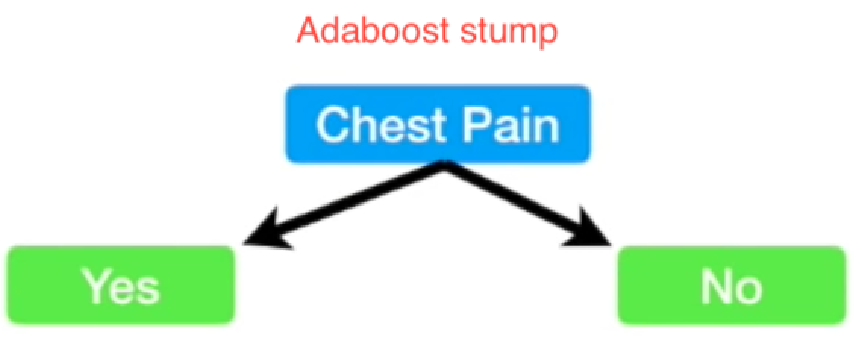
2.In **each iteration**, it tries to **provide** an **excellent fit** for these examples by **minimizing training error**.

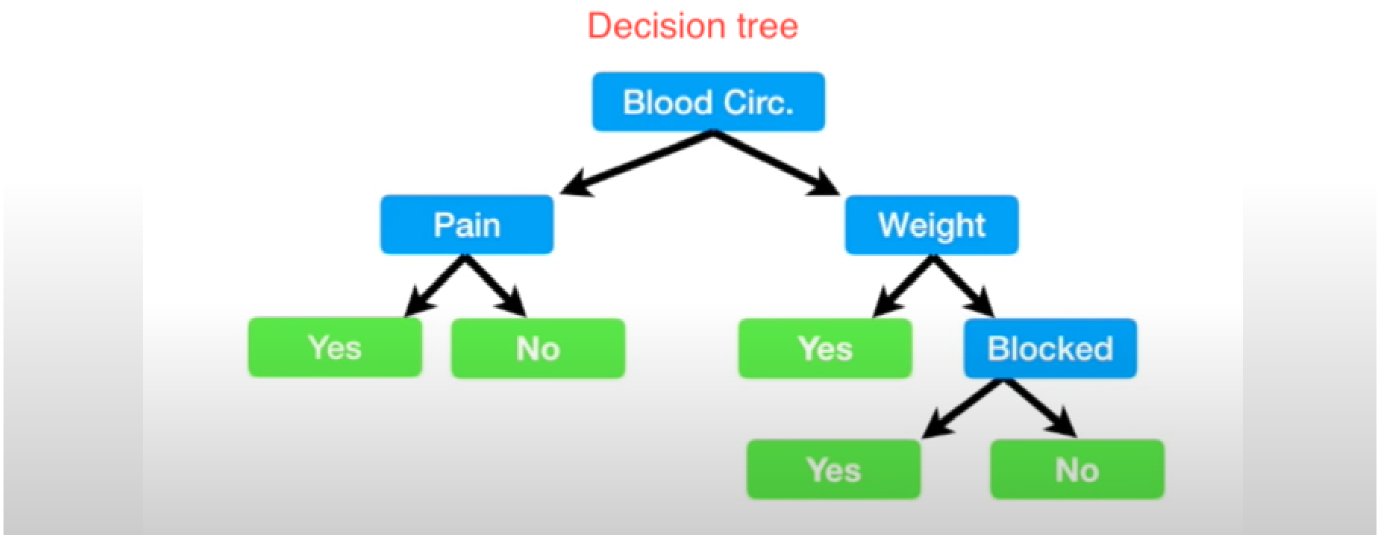

Original Dataset, D1
Updated Weighted Training Dataset, D2
Updated Weighted Training Dataset, D3
Model-1 (Iteration-1)
Model-2 (Iteration-2)
Model-3 (Iteration-3)
Predictions on Same Training Data D1
Predictions on Weighted Data, D2
Predictions on Weighted Data, D3

credit:https://www.datacamp.com/community/tutorials/adaboost-classifier-python

Decision Stumps

Decision Stumps are **like trees in a Random Forest**, **but not "fully grown**." They have one node and two leaves. AdaBoost uses a forest of such stumps rather than trees.





Stumps alone are **not a good way** to make **decisions**.

A **full-grown tree combines** the **decisions** from **all variables** to **predict** the **target** value.

A **stump**, on the other hand, **can only use one variable** to make a **decision**.

Step-wise Analysis

Step 1:

A **weak classifier** (e.g. a decision stump) is **made on top** of the **training data** based on the **weighted samples**.

Here, the **weights of each sample** indicate **how important it is to be correctly classified**. **Initially**, for the **first stump**, we give a**ll the samples equal weights.**

Step 2:

We create a decision stump for **each variable** and see how well each **stump classifies** samples to their **target classes**.

Step 3:

More weight is assigned to the **incorrectly classified** samples so that **they're classified correctly** in the next decision stump.

Weight is also assigned to each classifier based on the accuracy of the classifier, which means high accuracy = high weight!

Step 4:

Reiterate from Step 2 until **all the data points** have been **correctly classified, or the maximum iteration level has been reached**.

Adaboost implementation:

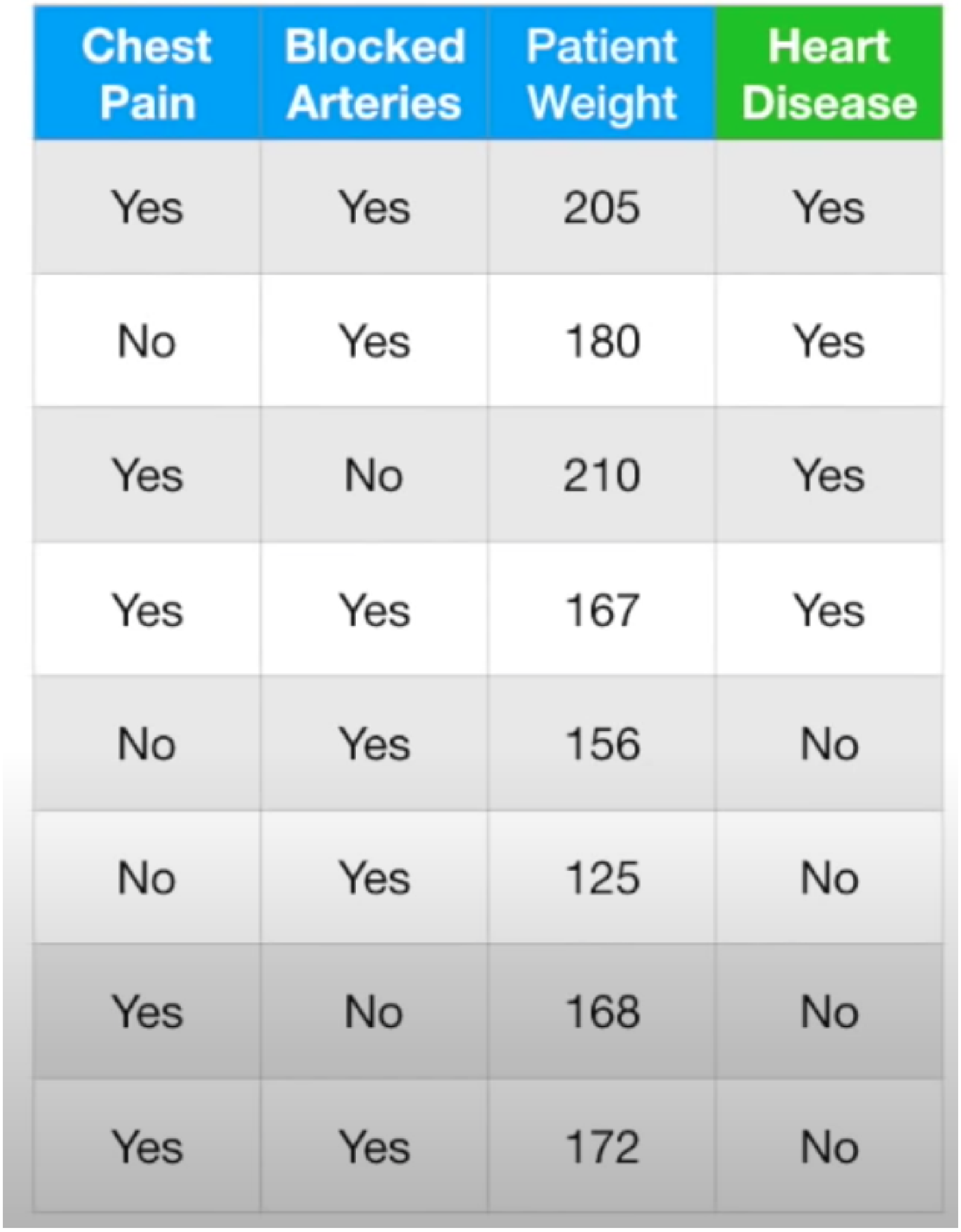
Credit to StatQuest with Josh Starmer

https://www.youtube.com/watch?v=LsK-xG1cLYA

Double Bam!!!!

I have directly taken an example from Josh Starmer video.

Our main aim is to build a model that classifies whether a patient has heart disease are not when these independent variables (chest pain, blocked arteries, weight)are given.

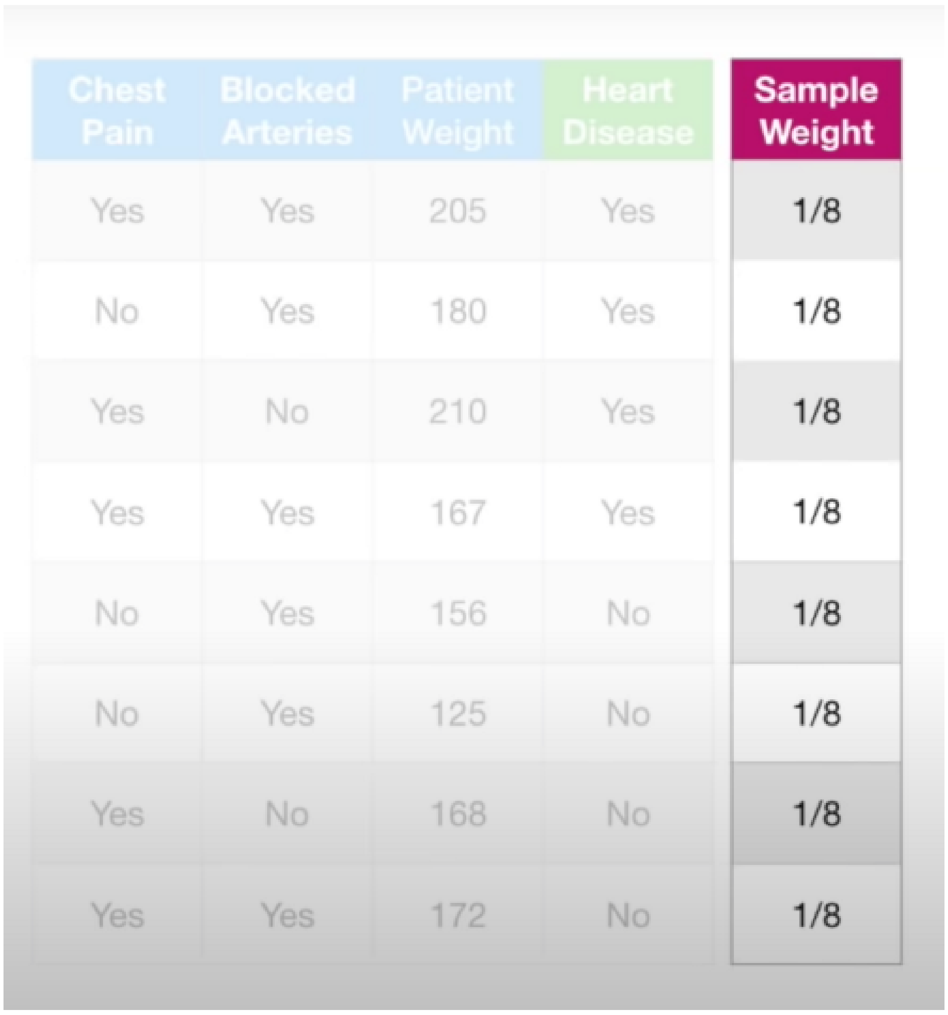


Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
Yes	Yes	205	Yes
No	Yes	180	Yes
Yes	No	210	Yes
Yes	Yes	167	Yes
No	Yes	156	No
No	Yes	125	No
Yes	No	168	No
Yes	Yes	172	No

For the above example, first what we will do is we will give **some sample weights(these values which will tell the importance of the sample value based on their error while predicting)** that **indicate how important it is to be correctly classified. **

At the beginning (for the first stump) all samples get the same weight value. that is,

Sample weight=1% of No.of samples in the dataset



Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

Then we will **find out the first stump**, **after** the **first stump**, we will **update the sample weight** in **order to make the next stump** to **predict that incorrectly classified sample correctly**!!

Now we will find out the **Gini index** of all the **independent variables in order** to find out the first stump, **lower the Gini index value is a good predictor of this class.**

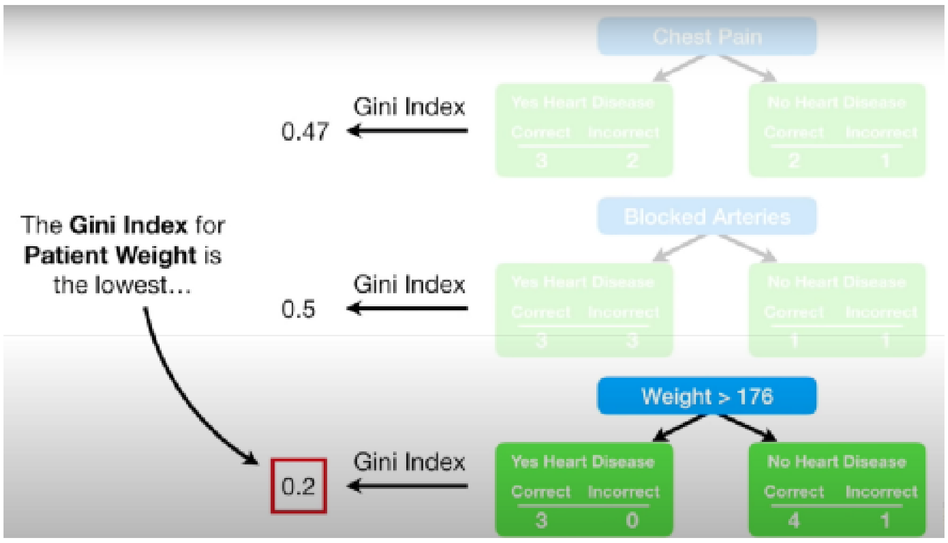
The **formula** for **Gini Index**

$$\text{Gini} = 1 - \sum_{i=1}^{n} (p_i)^2$$

where **pi** is the **probability** of an **object being classified into a particular class.**

The **degree of the Gini index** varies **between 0 and 1**, where **0 denotes that all elements belong to a certain class** or if there exists only one class, and **1 denotes that the elements are randomly distributed across various classes.**

A Gini index of **0.5** denotes **equally distributed elements into some classes**.



Amount of say

When we apply the Gini index to all the independent variable we get a low Gini index of 0.2 to weight so we take the weight parameter as our first stump. the next least value is taken as the next stump…

Now we will find out how well this variable says to our class(how well this variable predicts) this is called Amount of say.

This stump made only one error, that is the patient whose weight is equal to 167(less than 176) has heart disease but our stump says that this patient does not have heart disease.

We need to find out the total error did by our stump in order to say how well it predicts our data.

So total error of a stump is the sum of weights associated with the incorrectly classified sample.

So in our case the total error of weight stump is 1/8.

NOTE: because all of the sample weight add up 1, the total error will always be between **0, for a perfect stump, and 1 for a horrible stump!!**

We use the **Total Error** to determine **Amount of Say** this stump has in the final classification with the following formula:

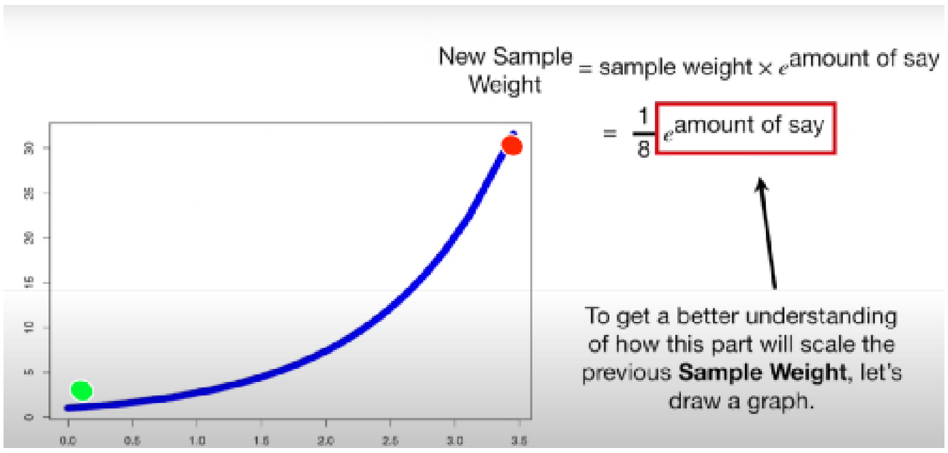
$$\text{Amount of Say} = \frac{1}{2}\log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$

How to update the sample weight

This is important because at the beginning all the Sample weight values are the same but after the first stump, we need to change this value in order to tell the next stump that this sample was not correctly classified, so the next stump will try to give more preference to that sample(so at the end it can predict correctly that sample).

So we will increase that sample weight value(which is been classified wrongly) and we will decrease all the other sample weights(which are classified correctly).

$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$



$$\text{New Sample Weight} = \text{sample weight} \times e^{\text{amount of say}}$$

$$= \frac{1}{8}\boxed{e^{\text{amount of say}}}$$

To get a better understanding of how this part will scale the previous **Sample Weight**, let's draw a graph.

formula to find the **new sample weight for the correctly classified sample** is:

$$\text{New Sample Weight} = \text{sample weight} \times e^{\boxed{-\text{amount of say}}}$$

Hyperparameter tuning with Adaboost

n_estimator

Often by changing the number of base models or weak learners we can adjust the accuracy of the model.

learning_rate:

Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. There is a trade-off between the learning_rate and n_estimators parameters.

Advanatages and Disadvanatages

Coming to the **advantages**, **Adaboost** is **less prone to overfitting as the input parameters are not jointly optimized**.

The **accuracy** of** weak classifiers can be improved by using Adaboost**.

Nowadays, Adaboost is being used to classify text and images rather than binary classification problems.

The main **disadvantage** of **Adaboost is that it needs a quality dataset. Noisy data and outliers have to be avoided before adopting an Adaboost algorithm.**