SUS: Machine Learning: Lab4

Eyad Kannout

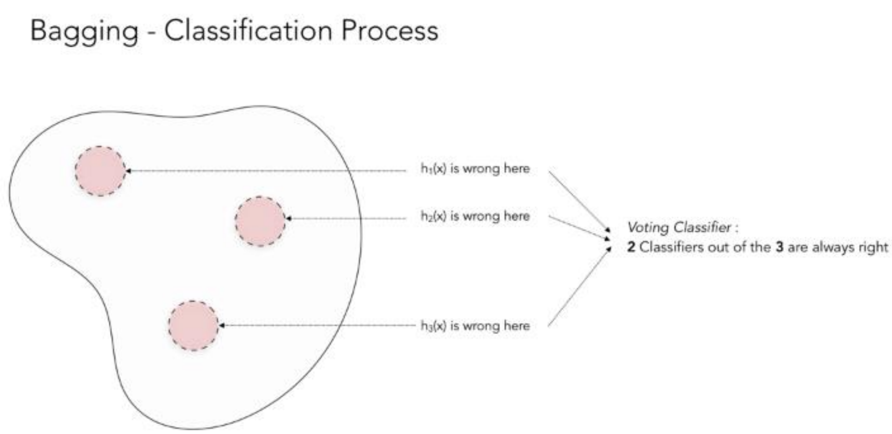
Boosting:

The limits of Bagging "Bootstrap Aggregating":

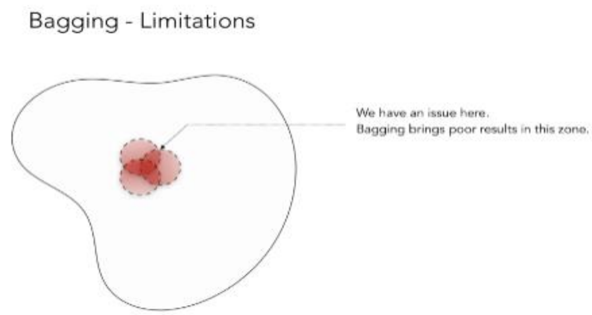
Bagging is a technique that stands for "Bootstrap Aggregating". The essence is to select T bootstrap samples, fit a classifier on each of these samples, and train the models in parallel. Typically, in a Random Forest, decision trees are trained in parallel. The results of all classifiers are then averaged into a bagging classifier:

$$H_T(x) = 1/T \sum_t h_t(x)$$

Let's consider 3 classifiers which produce a classification result and can be either right or wrong. If we plot the results of the 3 classifiers, there are regions in which the classifiers will be wrong. These regions are represented in red.



This example works perfectly, since when one classifier is wrong, the two others are correct. By voting classifier, you achieve a great accuracy! But as you might guess, there's also cases in which Bagging does not work properly, when all classifiers are mistaken in the same region.

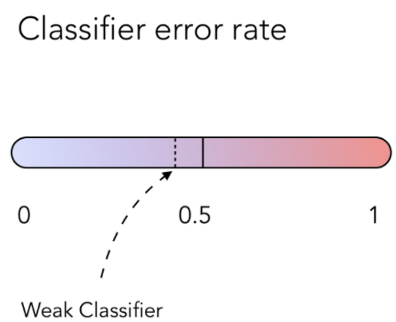


For this reason, the intuition behind the discovery of Boosting was the following:

- **instead of training parallel models, one needs to train models sequentially**
- **each model should focus on where the previous classifier performed poorly**

Introduction to Boosting:

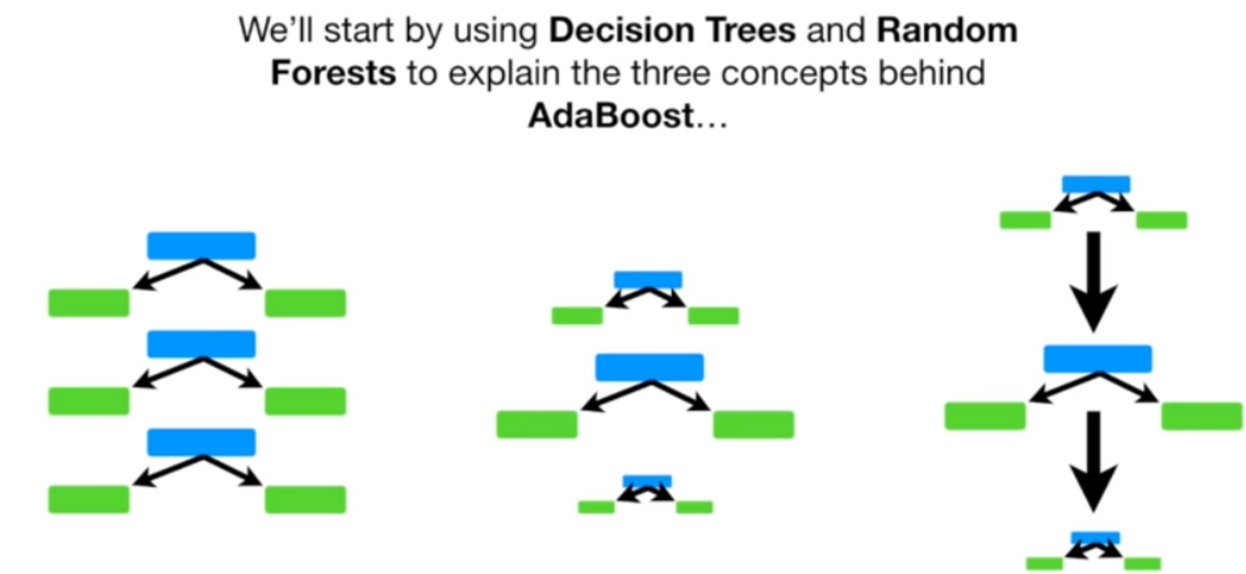
Boosting trains a series of low performing algorithms, called weak learners, by adjusting the error metric over time. Weak learners are algorithms whose error rate is slightly under 50%.

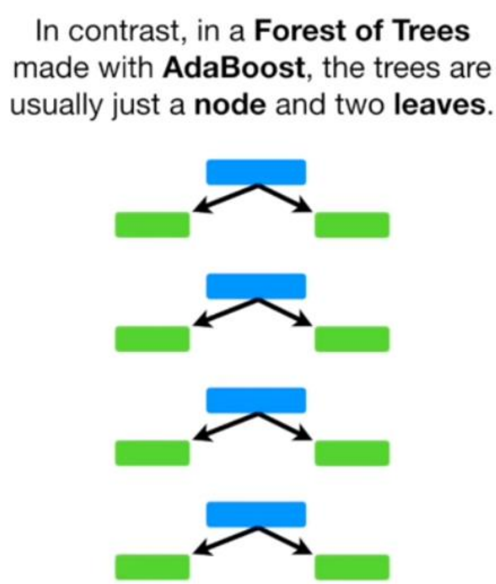


Weak classifiers (or weak learners) are classifiers which perform only slightly better than a random classifier. These are classifiers which have some clue on how to predict the right labels, but not as much as strong classifiers have like, e.g., Naive Bayes, Neurel Networks or SVM.

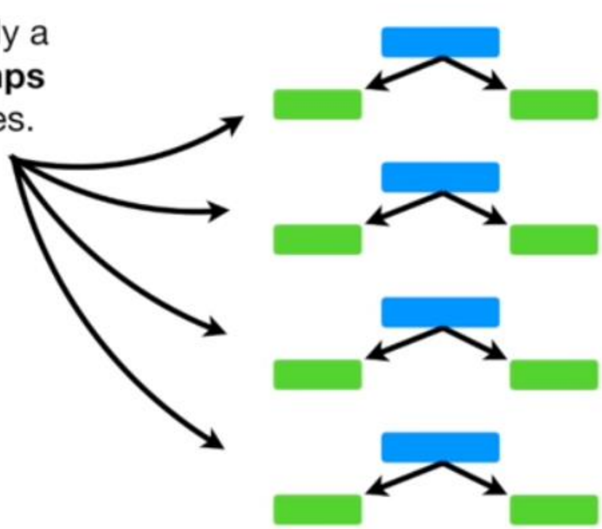
One of the simplest weak classifiers is the Decision Stump, which is a one-level Decision Tree. It selects a threshold for one feature and splits the data on that threshold. AdaBoost will then train an army of these Decision Stumps which each focus on one part of the characteristics of the data.

How does AdaBoost (adaptive boosting) work?





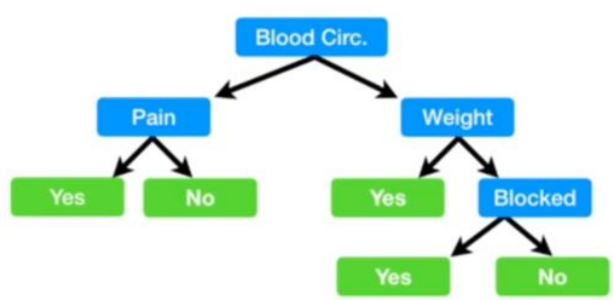
…so this is really a **Forest of Stumps** rather than trees.



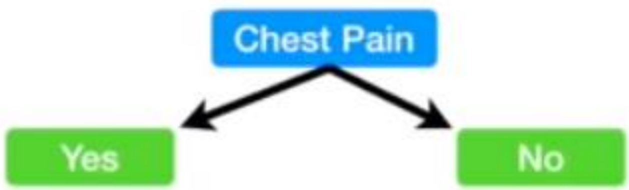
Stumps are not great at making accurate classifications.

…then a full sized **Decision Tree** would take advantage of all **4** variables that we measured (**Chest Pain**, **Blood Circulation**, **Blocked Arteries** and **Weight**) to make a decision…

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes



...but a **Stump** can only use one variable to make a decision.



Thus, **Stumps** are technically "weak learners".

However, that's the way **AdaBoost** likes it, and it's one of the reasons why they are so commonly combined.

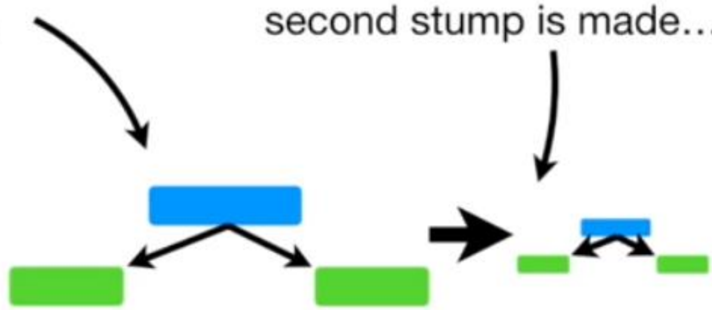
In contrast, in a **Forest of Stumps** made with **AdaBoost**, some stumps get more say in the final classification than others.

Lastly, in a **Random Forest**, each decision tree is made independently of the others.

In contrast, in a **Forest of Stumps** made with **AdaBoost**, order is important.

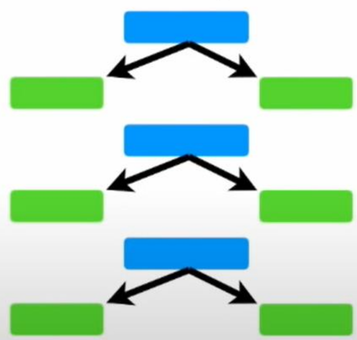
The errors that the first stump makes…

…influence how the second stump is made…

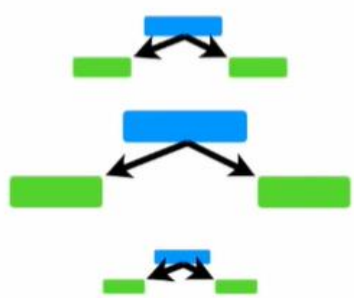


To review, the three ideas behind AdaBoost are:

1) **AdaBoost** combines a lot of "weak learners" to make classifications. The weak learners are almost aways **stumps**.



2) Some **stumps** get more say in the classification than others.



3) Each **stump** is made by taking the previous **stump's** mistakes into account.

Now let's dive into the nitty gritty detail of how to create a **Forest of Stumps** using **AdaBoost**.

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

First, we'll start with some data.

Sample Weight

The first thing we do is give each sample a weight that indicates how important it is to be correctly classified.

Sample Weight
1/8
1/8
1/8
1/8
1/8
1/8
1/8
1/8

At the start, all samples get the same weight...

$$\frac{1}{\text{total number of samples}} = \frac{1}{8}$$

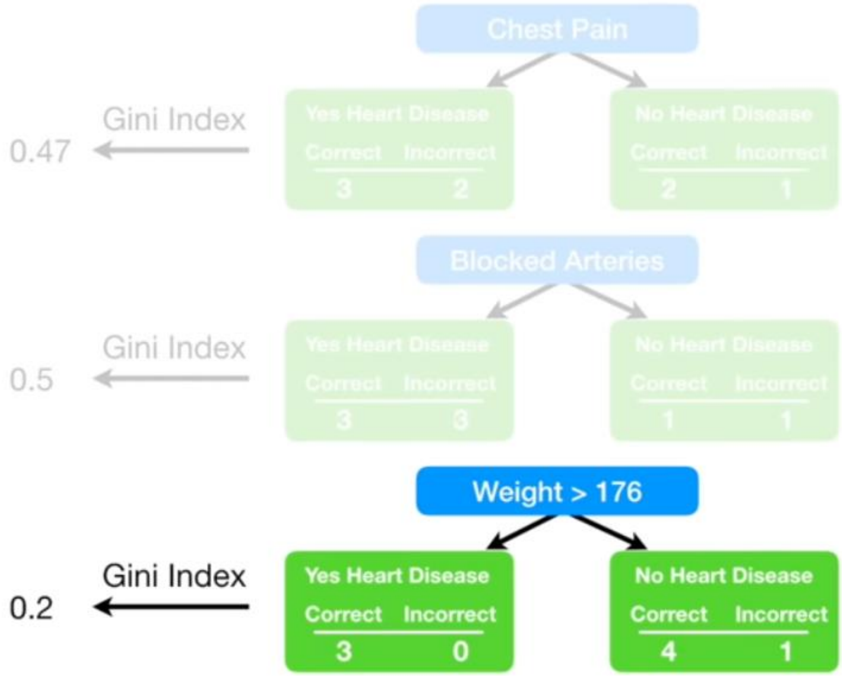
...and that makes the samples all equally important.

However, after we make the first stump, these weights will change in order to guide how the next stump is created.

In other words, we'll talk more about the **Sample Weights** later!

This is done finding the variable, **Chest Pain**, **Blocked Arteries** or **Patient Weight**, that does the best job classifying the samples.

Now we calculate the **Gini Index** for the three stumps.



Now we need to determine how much say this stump will have in the final classification.

Remember, some stumps get more say in the final classification than others.

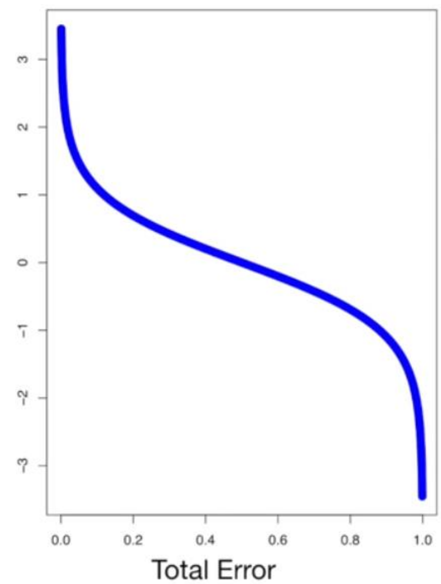
We determine how much say a stump has in the final classification based on how well it classified the samples.

The **Total Error** for a stump is the sum of the weights associated with the *incorrectly* classified samples.

NOTE: Because all of the **Sample Weights** add up to **1**, **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:

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



We can draw a graph of the **Amount of Say** by plugging in a bunch of numbers between **0** and **1** for **Total Error**.

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

Amount of Say = $\frac{1}{2} log \frac{1 - \frac{1}{8}}{\frac{1}{8}}$ = 0.42

Now we need to learn how to modify the weights so that the next stump will take the errors that the current stump made into account.

Sample Weight
1/8
1/8
1/8
1/8

...we will emphasize the need for the next stump to correctly classify it by increasing its **Sample Weight**...

Sample Weight
1/8
1/8
1/8
1/8
1/8
1/8
1/8
1/8
...and decreasing all of the other **Sample Weights**.
Weight > 176
Yes Heart Disease
Correct Incorrect
3 0
No Heart Disease
Correct Incorrect
4 1

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

This is the formula we will use to *increase* the **Sample Weight** for the sample that was *incorrectly* classified.

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

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

$$= \frac{1}{8} e^{0.97} = \frac{1}{8} \times 2.64 \boxed{= 0.33}$$

That means the new **Sample Weight** is **0.33**, which is *more* than the old one (**1/8** = **0.125**).

Sample Weight
1/8
1/8
1/8
1/8
1/8
1/8
1/8
1/8

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

This is the formula we will use to *decrease* the **Sample Weights**.

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

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

$$= \frac{1}{8} e^{-0.97} = \frac{1}{8} \times 0.38 \boxed{= 0.05}$$

The new **Sample Weight** is **0.05**, which is *less* than the old one (**1/8** = **0.125**).

New Weight
0.05
0.05
0.05
0.33
0.05
0.05
0.05
0.05

Now we need to normalize the **New Sample Weights** so that they will add up to 1.

New Weight	Norm. Weight
0.05	0.07
0.05	0.07
0.05	0.07
0.33	0.49
0.05	0.07
0.05	0.07
0.05	0.07
0.05	0.07

So we divide each **New Sample Weight** by **0.68** to get the normalized values.

Now, when we add up the **New Sample Weights**, we get **1** (plus or minus a little rounding error).

Now we can use the modified **Sample Weights** to make the second **stump** in the forest.

In theory, we could use the **Sample Weights** to calculate **Weighted Gini Indexes** to determine which variable should split the next stump.

The **Weighted Gini Index** would put more emphasis on correctly classifying this sample (the one that was misclassified by the last stump), since this sample has the largest **Sample Weight**.

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

In previous formula pi represent all samples belong to class c.

In order to involve the weight for each sample, we can multiply pi * wi

Where wi represents the sum of he weighs of all samples in class c

Alternatively, instead of using a **Weighted Gini Index**, we can make a new collection of samples that contains duplicate copies of the samples with the largest **Sample Weights**.

Then we pick a random number between 0 and 1...

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

If the number is between **0** and **0.07**, then we would put this sample into the new collection of samples...

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125		
Yes	No	168		
Yes	Yes	172		

Ultimately, this sample was added to the new collection of samples **4** times, reflecting its larger **Sample Weight**.

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

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

Lastly, we give all the samples equal **Sample Weights**, just like before.

How select the records in the new dataset:

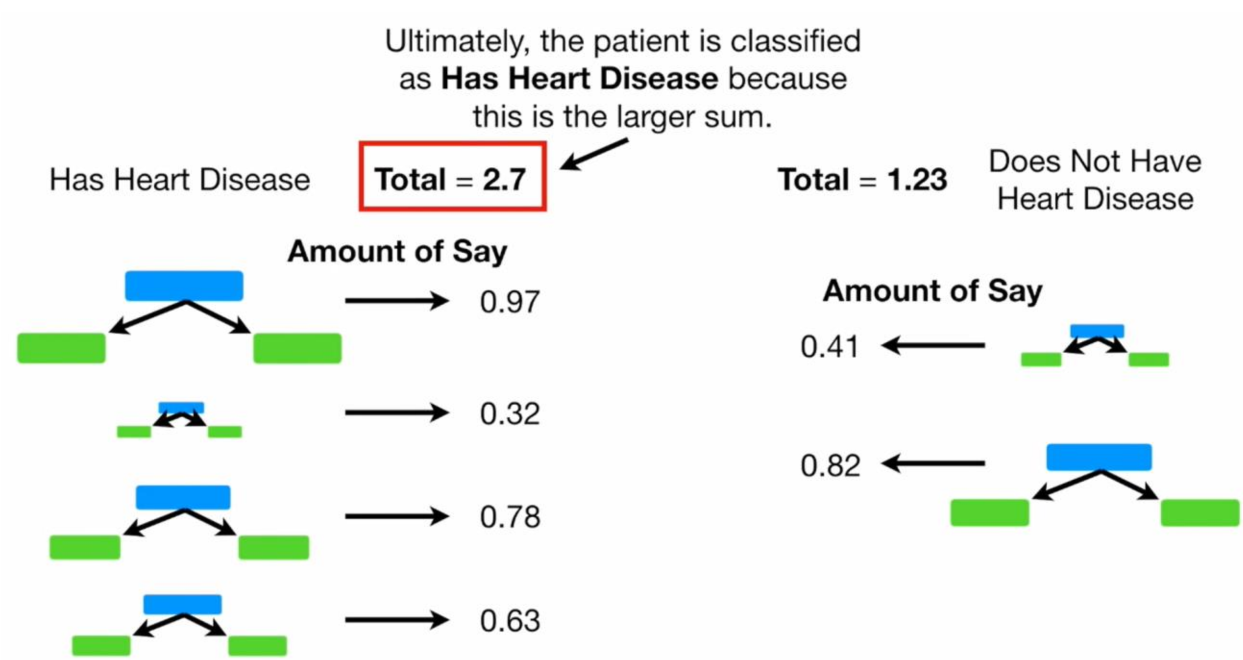
- **the new eights are normalized so the sum of them is equal to one**
- **we generate random number between 0 and 1**
- **if the number is between 0.0 and 0.07 then we pick first sample**
- **if the number is between 0.07 and 0.14 then we pick second sample**
- **if the number is between 0.14 and 0.21 then we pick third sample**
- **if the number is between 0.21 and 0.49 then we pick fourth sample**

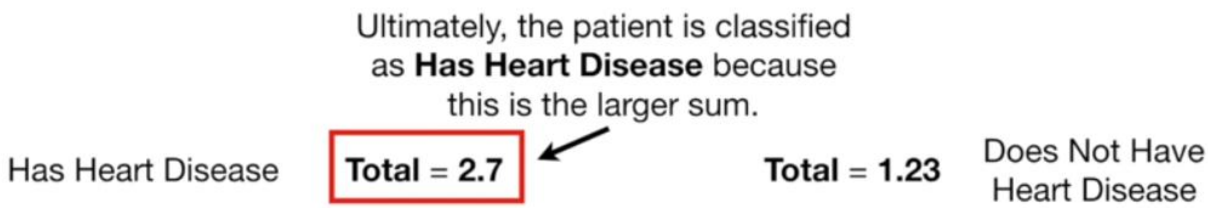
- ….
- **We observe that the probability of picking samples with higher weight is larger than others. This will give a chance to add these records many times in the new dataset**
- **The size of new dataset is same as original one**
- **Finally, we give all new sample same weight.**

Now we need to talk about how a forest of stumps created by **AdaBoost** makes classifications…

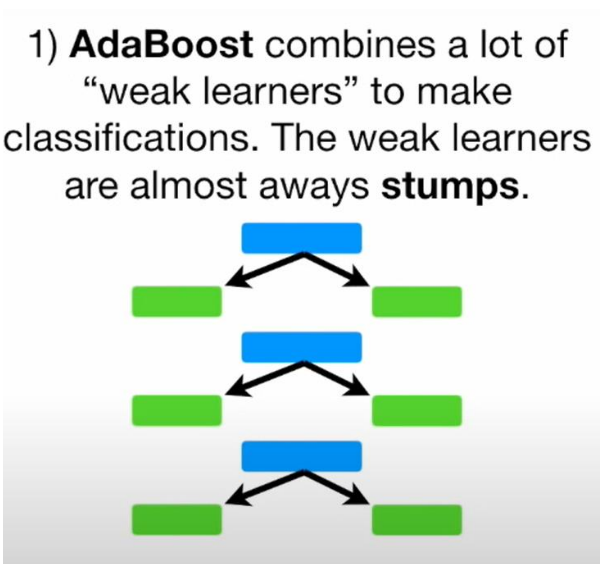
How to make prediction:

- **We find all stumps that predict (YES) or (has heart disease) on left side**
- **We find all stumps that predict (NO) or (Does not have heart disease) on right side**
- **Sum up the amount of say (weight of classifier) for each side**
- **The one has larger value is the one correct prediction**

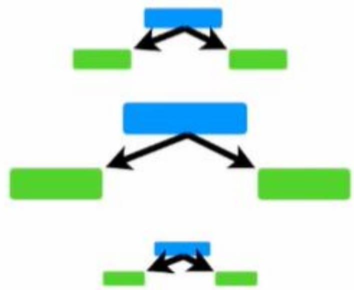




Review the three ideas behind AdaBoost:



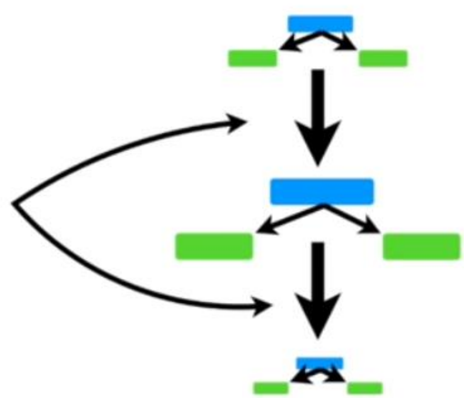
2) Some **stumps** get more say in the classification than others.



3) Each **stump** is made by taking the previous **stump's** mistakes into account.

3) Each **stump** is made by taking the previous **stump's** mistakes into account.

If we have a **Weighted Gini Function**, then we use it with the **Sample Weights**, otherwise we use the **Sample Weights** to make a new dataset that reflects those weights.



Step 1 : Let $w_t(i) = \frac{1}{N}$ where N denotes the number of training samples, and let T be the chosen number of iterations.

Step 2 : For t in T :

a. Pick h^t the weak classifier that minimizes ϵ_t

$$\epsilon_t = \sum_{i=1}^{m} w_t(i)[y_i \neq h(x_i)] \tag{2}$$

b. Compute the weight of the classifier chosen :

$$\alpha_t = \frac{1}{2} ln \frac{1 - \epsilon_t}{\epsilon_t} \tag{3}$$

c. Update the weights of the training examples w_{t+1}^i and go back to step a).

Gini Index:

- If a data set D contains examples from n classes, gini index, $gini(D)$ is defined as:

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class j in D

- If a data set D is split on A into two subsets D_1 and D_2, the *gini* index $gini(D)$ is defined as

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- Reduction in Impurity: $\Delta gini(A) = gini(D) - gini_A(D)$

- If a data set D contains examples from n classes, gini index, $gini(D)$ is defined as:

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class j in D

- If a data set D is split on A into two subsets D_1 and D_2, the *gini* index $gini(D)$ is defined as

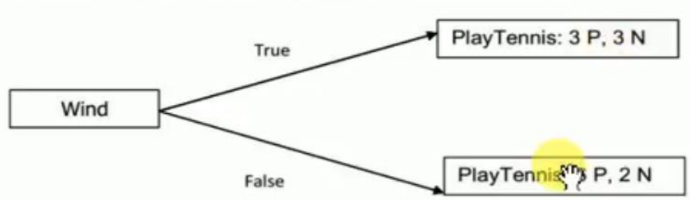
$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- Reduction in Impurity: $\Delta gini(A) = gini(D) - gini_A(D)$

Gini index calculation:

There are 5 Ns and 9 Ps, so the

- Calculate the information gain after the Wind test is applied:



Gini (PlayTennis|Wind=True) = 1- $(3/6)^2 - (3/6)^2$ = 0.5
Gini (PlayTennis|Wind=False) = 1- $(6/8)^2 - (2/8)^2$ = 0.375

Therefore, the Gini index after the Wind test is applied is

6/14 × 0.5 + 8/14 × 0.375 = 0.4286

Task:

- Read the documentation about sklearn.ensemble.AdaBoostClassifier in scikit-learn
- Use Iris dataset from sklearn
- Explore ethe dataset
- Split the data set using train_test_split from sklearn
- Fit the mode using training set and AdaBoostClassifier from sklearn
- Find the best values of input parameters (n_estimators, learning_rate)
- Make predication using testing set
- Evaluate the model
- Repeat previous steps using cross validation from sklearn
- Repeat previous steps using different base_estimator