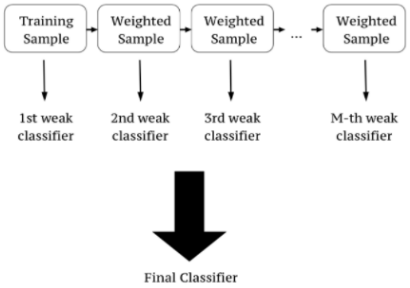
Boosting

Some story -

Assume you want to travel to Goa, but you don't know how to plan because you are not a good decision maker. You check with first friend and he gives some input (he may not be good in making complete plan), You take first friend input and explain to 2nd friend and he gives some more input and you take both inputs and give to 3rd friend, he also add some more inputs.. This will continue until you see you have concrete plan.

Boosting is an ensemble approach(meaning it involves several trees) that starts from a weaker decision and keeps on building the models such that the final prediction is the weighted sum of all the weaker decision-makers. The weights are assigned based on the performance of an individual tree.



Ensemble parameters are calculated in **stagewise way** which means that while calculating the subsequent weight, the learning from the previous tree is considered as well.

Weak classifier - why tree?

First what is a weak classifier? **Weak classifier** - *slightly better* than random guessing.

Any algorithm could have been used as a base for the boosting technique, but the reason for choosing trees are:

Pro's

- computational scalability,
- handles missing values,
- robust to outliers,
- does not require feature scaling,
- can deal with irrelevant inputs,
- interpretable (if small),
- handles mixed predictors as well (quantitive and qualitative)

Con's

- inability to extract a linear combination of features
- high variance leading to a small computational power

And that's where boosting comes into the picture. It minimises the variance by taking into consideration the results from various trees.

Ada Boost (Adaptive Boosting)

Example

For understanding this algorithm, we'll use the following simple dataset for heart patient prediction.

]:
```
import pandas as pd
heart_data= pd.read_csv('heart_disease-test.csv')
heart_data
```

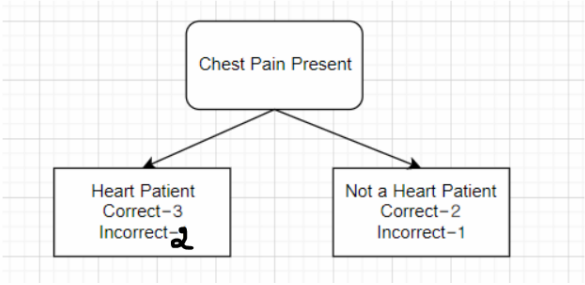
]:

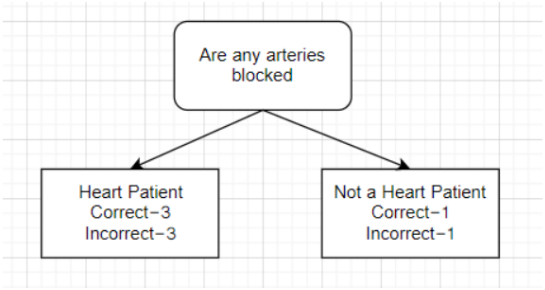
	Is Chest Pain Present	Are any arteries blocked	Weight of the person	Is Heart Patient
0	YES	YES	205	YES
1	NO	YES	180	YES
2	YES	NO	210	YES
3	YES	YES	167	YES
4	NO	YES	156	NO
5	NO	YES	125	NO
6	YES	NO	168	NO
7	YES	YES	172	NO

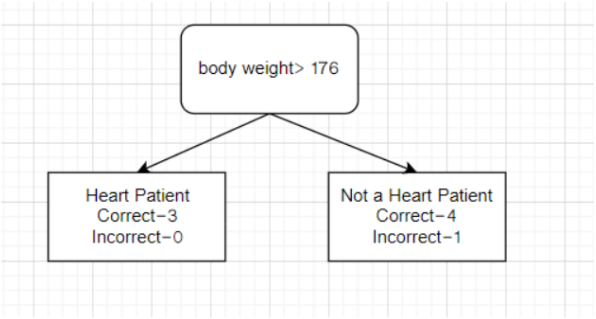
- There are a total of 8 rows in our dataset. Hence, we'll initialize the sample weights($w = \frac{1}{N}$) as 1/8 in the beginning. And, at the beginning, all the samples are equally important.

	A	B	C	D	E
1	Is Chest Pain Present	Are any arteries blocked	Weight of the person	Is Heart Patient	Sample Weight
2	YES	YES	205	YES	1/8
3	NO	YES	180	YES	1/8
4	YES	NO	210	YES	1/8
5	YES	YES	167	YES	1/8
6	NO	YES	156	NO	1/8
7	NO	YES	125	NO	1/8
8	YES	NO	168	NO	1/8
9	YES	YES	172	NO	1/8

- We'll consider the individual columns to create weak decision-makers as shown below and then try to figure out what are the correct and incorrect predictions based on that column.







- We'll now calculate the Gini index of the individual stumps using the formula

G.I= $\sum(weight of the decision) * (1 - (p^2 + (1 - p)^2))$

```
G.I for chest pain tree= 0.47
G.I for blocked arteries tree= 0.5
G.I for body-weight tree= 0.2

And, we select the tree with the lowest Gini Index. This will be the first decision-maker for our model.
```

- Now, we'll calculate the contribution of this tree(stump) to our final decision using the formula:

Contribution= $½(log(1 - totalerror)/totalerror)$

```
As this stump classified only one data incorrectly out of the 8, hence the total error is 1/8.

Putting this into the formula we get contribution= 0.97
```

- We'll now calculate the new weights using the formula:
- Increase the sample weight for incorrectly classified datapoints New weight= old weighte^ *contribution= 1/8* e^0.97=0.33
- Decrease the sample weight for correctly classified datapoints New weight= old weighte^- *contribution= 1/8* e^-0.97=0.05
- Populate the new weights as shown below:

	Is Chest Pain Present	Are any arteries blocked	Weight of the person	Is Heart Patient	Sample Weight	New Sample Weight
2	YES	YES	205	YES	1/8	0.05
3	NO	YES	180	YES	1/8	0.05
4	YES	NO	210	YES	1/8	0.05
5	YES	YES	167	YES	1/8	0.33
6	NO	YES	156	NO	1/8	0.05
7	NO	YES	125	NO	1/8	0.05
8	YES	NO	168	NO	1/8	0.05
9	YES	YES	172	NO	1/8	0.05

- Normalize the sample weights: If we add all the new sample weights, we get 0.68. Hence, for normalization we divide all the sample weights by 0.68 and then create normalized sample weights as shown below:

	Is Chest Pain Present	Are any arteries blocked	Weight of the person	Is Heart Patient	Sample Weight	New Sample Weight	Normalized weights
2	YES	YES	205	YES	1/8	0.05	0.07
3	NO	YES	180	YES	1/8	0.05	0.07
4	YES	NO	210	YES	1/8	0.05	0.07
5	YES	YES	167	YES	1/8	0.33	0.49
6	NO	YES	156	NO	1/8	0.05	0.07
7	NO	YES	125	NO	1/8	0.05	0.07
8	YES	NO	168	NO	1/8	0.05	0.07
9	YES	YES	172	NO	1/8	0.05	0.07

```
These new normalized weights will act as the sample weights for the next iteration.
```

- Then we create new trees which consider the dataset which was prepared using the new sample weights.
- Suppose, m trees(stumps) are classifying a person as a heart patient and n trees(stumps) are classifying a person as a healthy one, then the contribution of m and n trees are added separately and whichever has the higher value, the person is classified as that.

For example, if the contribution of m trees is 1.2 and the contribution of n trees is 0.5 then the final result will go in the favour of m trees and the person will be classified as a heart patient.

In [ ]:

Python Implementation