Boosting Algorithms Supervised Learning Classification

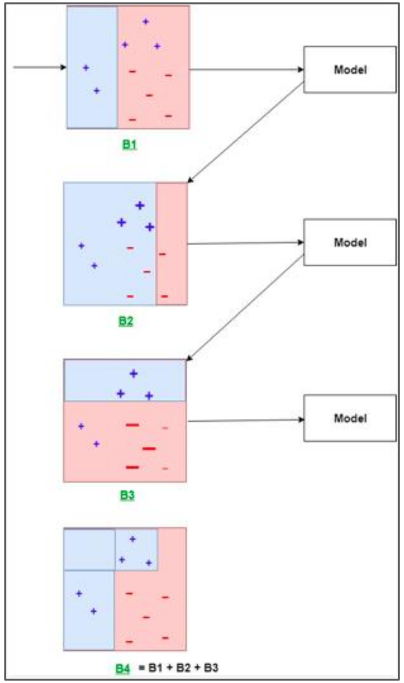
Agenda

- Concept of Boosting
- Boosting Algorithms
  - AdaBoost
  - Gradient Boosting
  - XGBoost

Boosting

What is Boosting?

- The ensemble of weak learners that learn sequentially
- In each iteration weights of the samples are adjusted, such that the misclassified samples have a higher weight, therefore higher chance of getting selected to train the next classifier
- Boosting reduces bias and variance
- Here B1, B2 and B3 are base learners, and B4 is the boosting ensemble of weak learners



Boosting: advantages

- Enhances the efficiency of weak classifiers

- Both precision and recall can be enhanced through boosting algorithms

Boosting: disadvantages

- Loss of simplicity and explanatory power
- Increased computational complexity

How boosting differs from bagging?

BAGGING	BOOSTING
Base learners learn is parallel	Base learners learn sequentially
Random sampling	Non-random sampling
Reduces variance	Reduces bias and variance

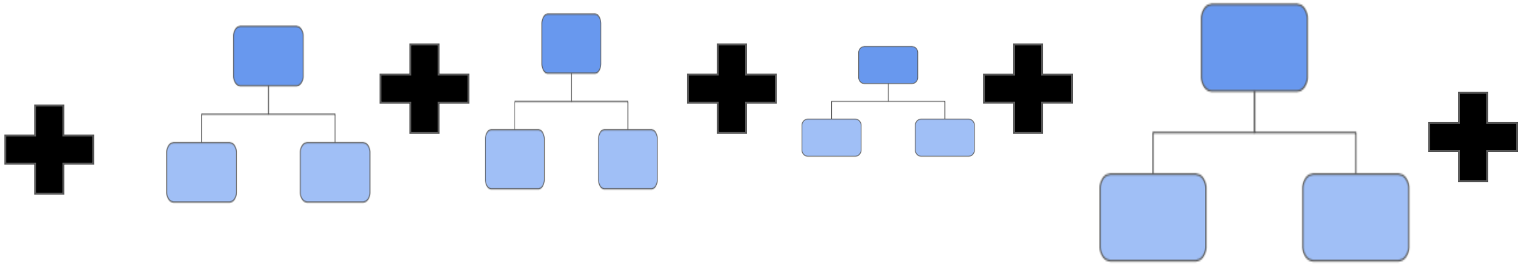
Boosting Algorithms

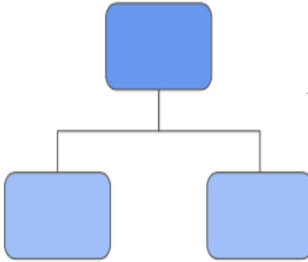
Boosting Algorithms

- AdaBoost
- Gradient Boosting
- XGBoost

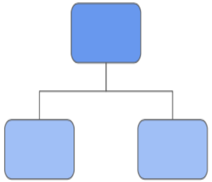
Ada Boost

Adaboost

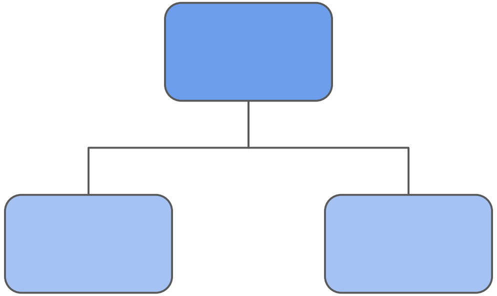




Adaboost is basically a forest of stumps.

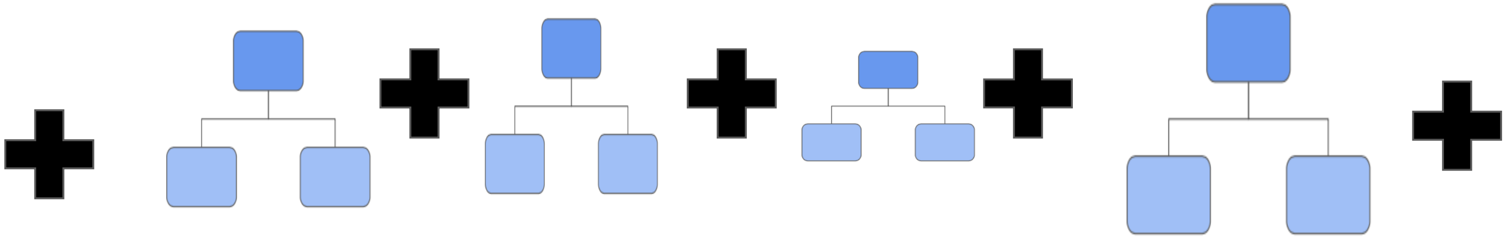


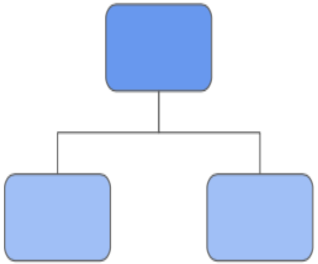
Stump



Stump is a tree with just one parent node and two leaves

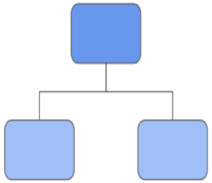
Adaboost: forest of stumps...





- A single algorithm or a stump may classify the objects poorly

- But if we combine multiple classifiers with a selection of training set at every iteration and assigning the right amount of weight in the final voting, we can have good accuracy score for overall classifier



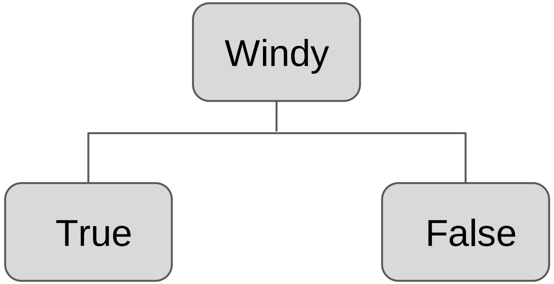
AdaBoost

- AdaBoost is short for Adaptive Boosting
- It was formulated by Yoav Freund and Robert Schapire
- It is sensitive to noisy data and outliers
- Usually, decision trees are used for modelling
- Multiple sequential models are created, each correcting the errors from the last model
- AdaBoost works by weighing the observations, putting more weight on difficult to classify instances and less on those already handled well

AdaBoost: key concept 1

- Adaboost combines a lot of weak learners to make classifications. These weak learners are most often stumps

Observation	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	0
2	sunny	hot	high	true	0
3	overcast	hot	high	false	1
4	rainy	mild	high	false	1
5	rainy	cool	normal	false	1
6	rainy	cool	normal	true	0
7	overcast	cool	normal	true	1
8	sunny	mild	high	false	0
9	sunny	cool	normal	false	1
10	rainy	mild	normal	false	1
11	sunny	mild	normal	true	1
12	overcast	mild	high	true	1
13	overcast	hot	normal	false	1
14	rainy	mild	high	true	0

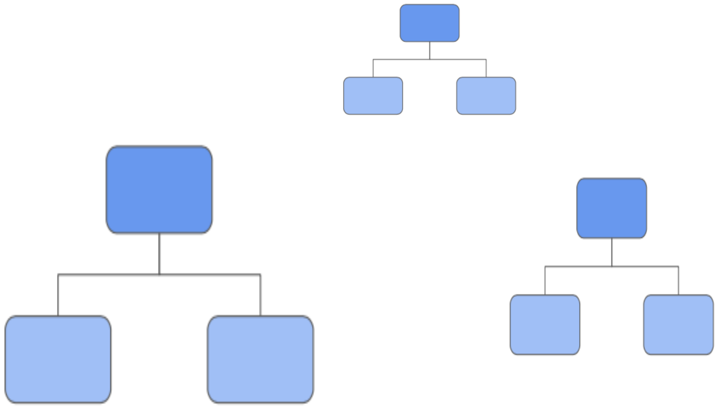


- Stump can use only one variable to make a decision

AdaBoost: key concept 2

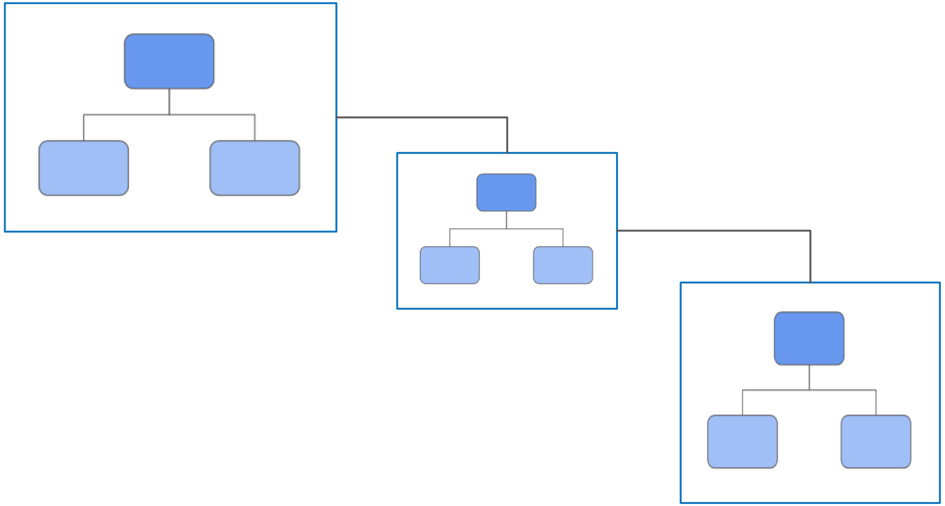
- Some stumps have more weightage in the final prediction than the others

- In this example the larger stumps have more dominance than the smaller stumps



AdaBoost: key concept 3

- Each stump is created by considering the mistakes of the previous stumps. Hence the order in which the stump is created is important
- The error that the 1st stump makes influences how the 2nd stump is made



AdaBoost: key concepts… reiterating

1. Adaboost combines a lot of weak learners to make classifications, these weak learners are most often stumps

2. Some stumps have more say in the final classification than the others

3. Each stump is made by taking into account, the previous stumps mistakes

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	0
2	sunny	hot	high	true	0
3	overcast	hot	high	false	1
4	rainy	mild	high	false	1
5	rainy	cool	normal	false	1
6	rainy	cool	normal	true	0
7	overcast	cool	normal	true	1
8	sunny	mild	high	false	0
9	sunny	cool	normal	false	1
10	rainy	mild	normal	false	1
11	sunny	mild	normal	true	1
12	overcast	mild	high	true	1
13	overcast	hot	normal	false	1
14	rainy	mild	high	true	0

- Target Variable: Play
- Independent Variables: Outlook, Temperature, Humidity, Windy

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Weight
1	sunny	mild	high	false	0	1/14
2	rainy	mild	high	false	1	1/14
3	rainy	mild	high	false	1	1/14
4	sunny	mild	normal	true	1	1/14
5	rainy	mild	high	false	1	1/14
6	overcast	cool	normal	true	1	1/14
7	sunny	hot	high	true	0	1/14
8	rainy	mild	high	true	0	1/14
9	sunny	cool	normal	false	1	1/14
10	rainy	mild	high	false	1	1/14
11	sunny	hot	high	false	0	1/14
12	rainy	mild	normal	false	1	1/14
13	rainy	cool	normal	true	0	1/14
14	rainy	mild	high	false	1	1/14

Step 1:

- Assign weights to each of the sample
- In the beginning each sample will have the same weight
- Sample Weight = 1/No. Of Samples =1/14

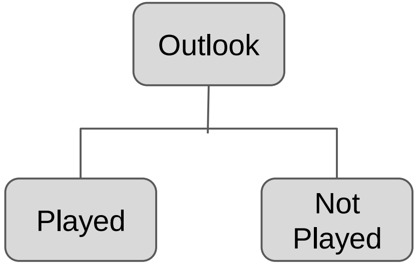
Please note: After we make the 1st stump, these sample weights will change in order to guide how the next stump gets created

AdaBoost: how it works

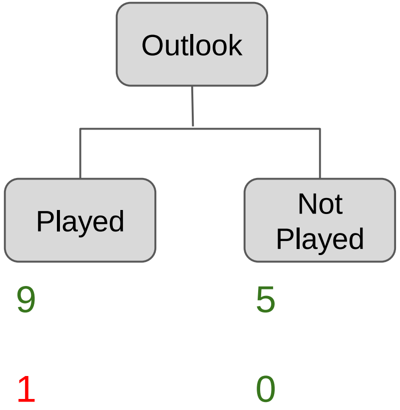
Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Weight
1	sunny	mild	high	false	0	1/14
2	rainy	mild	high	false	1	1/14
3	rainy	mild	high	false	1	1/14
4	sunny	mild	normal	true	1	1/14
5	rainy	mild	high	false	1	1/14
6	overcast	cool	normal	true	1	1/14
7	sunny	hot	high	true	0	1/14
8	rainy	mild	high	true	0	1/14
9	sunny	cool	normal	false	1	1/14
10	rainy	mild	high	false	1	1/14
11	sunny	hot	high	false	0	1/14
12	rainy	mild	normal	false	1	1/14
13	rainy	cool	normal	true	0	1/14
14	rainy	mild	high	false	1	1/14

Step 2:

- Build stumps with each variable. Calculate the Gini Index or Entropy for each variable
- AdaBoost picks the variable with the smallest Gini Index for building the stump
- This becomes the base learning model



AdaBoost: how it works



Step 3:

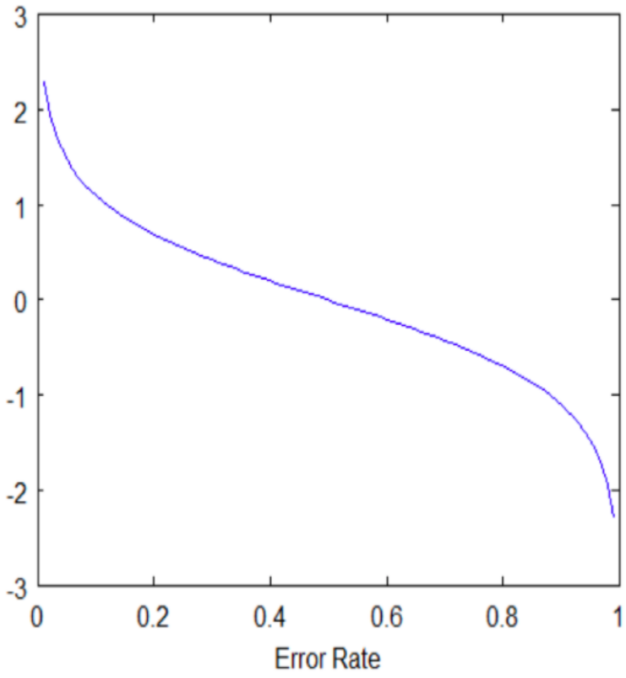
- Determine the ‘Amount of Say’ the stump will have in the final prediction

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

- Total Error = Sum of weights of misclassified samples
- Assuming there are a few correct & incorrect predictions, for eg. if a stump misclassified 1 sample incorrectly out of 14 samples, total error=1/14
- Plug in the values in the above formula:
‘Amount of Say’ =½ $\log_e(13)$=1.2824

AdaBoost: key concept

‘Total Error’ and ‘Amount of Say’



- When stump does a good job then Total Error is small. Amount of Say will be a large positive value
- When stump classifies only half of the samples correctly and half incorrectly, then Total Error is 0.5. Amount of Say will be Zero
- When stump does a very poor job i.e. the stump gives an opposite classification, then the total error would be close to -1 and the Amount of say will be a large negative value

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Weight
1	sunny	mild	high	false	0	1/14
2	rainy	mild	high	false	1	1/14
3	rainy	mild	high	false	1	1/14
4	sunny	mild	normal	true	1	1/14
5	rainy	mild	high	false	1	1/14
6	overcast	cool	normal	true	1	1/14
7	sunny	hot	high	true	0	1/14
8	rainy	mild	high	true	0	1/14
9	sunny	cool	normal	false	1	1/14
10	rainy	mild	high	false	1	1/14
11	sunny	hot	high	false	0	1/14
12	rainy	mild	normal	false	1	1/14
13	rainy	cool	normal	true	0	1/14
14	rainy	mild	high	false	1	1/14

- In AdaBoost, incorrectly classified records from the 1st stump have a greater chance of being passed to the next stump
- AdaBoost does this by, increasing the weights of the wrongly classified samples and decreasing the weights of the correctly classified samples
- Sample weights of the stump are updated basis the ‘Total Error’ & ‘Amount of Say’

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Weight	Updated Weight
1	sunny	hot	high	false	0	1/14	0.0198
2	sunny	hot	high	true	0	1/14	0.0198
3	overcast	hot	high	false	1	1/14	0.0198
4	rainy	mild	high	false	1	1/14	0.2575
5	rainy	cool	normal	false	1	1/14	0.0198
6	rainy	cool	normal	true	0	1/14	0.0198
7	overcast	cool	normal	true	1	1/14	0.0198
8	sunny	mild	high	false	0	1/14	0.0198
9	sunny	cool	normal	false	1	1/14	0.0198
10	rainy	mild	normal	false	1	1/14	0.0198
11	sunny	mild	normal	true	1	1/14	0.0198
12	overcast	mild	high	true	1	1/14	0.0198
13	overcast	hot	normal	false	1	1/14	0.0198
14	rainy	mild	high	true	0	1/14	0.0198

Step 4:

- Increase the sample weights of incorrectly classified samples, by the formula,

$$\text{New Sample Weight} = \text{Sample Weight} \cdot e^{\text{Amount Of Say}}$$

- Plug in the value of, Amount of Say =1.28 (from slide 21)
- New Sample Weight = 1/14 * e^1.28 = 0.2575

Notice the new sample weight is greater than the previous sample weight, 1/14 =0.0714

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Weight	Updated Weight
1	sunny	hot	high	false	0	1/14	0.0198
2	sunny	hot	high	true	0	1/14	0.0198
3	overcast	hot	high	false	1	1/14	0.0198
4	rainy	mild	high	false	1	1/14	0.2575
5	rainy	cool	normal	false	1	1/14	0.0198
6	rainy	cool	normal	true	0	1/14	0.0198
7	overcast	cool	normal	true	1	1/14	0.0198
8	sunny	mild	high	false	0	1/14	0.0198
9	sunny	cool	normal	false	1	1/14	0.0198
10	rainy	mild	normal	false	1	1/14	0.0198
11	sunny	mild	normal	true	1	1/14	0.0198
12	overcast	mild	high	true	1	1/14	0.0198
13	overcast	hot	normal	false	1	1/14	0.0198
14	rainy	mild	high	true	0	1/14	0.0198

Step 5:

- Decrease the sample weights of correctly classified samples, by the formula,

$$\text{New Sample Weight} = \text{Sample Weight} \cdot e^{-\text{Amount Of Say}}$$

- Plug in the value of,
- Amount of Say =1.28 (from slide 21)
- New Sample Weight = 1/14 * e^-1.28 = 0.0198

Notice the new sample weight is less than the previous sample weight, 1/14 =0.0714

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Weight	Updated Weight	Normalised Weight
1	sunny	hot	high	false	0	1/14	0.0198	0.0385
2	sunny	hot	high	true	0	1/14	0.0198	0.0385
3	overcast	hot	high	false	1	1/14	0.0198	0.0385
4	rainy	mild	high	false	1	1/14	0.2575	0.5000
5	rainy	cool	normal	false	1	1/14	0.0198	0.0385
6	rainy	cool	normal	true	0	1/14	0.0198	0.0385
7	overcast	cool	normal	true	1	1/14	0.0198	0.0385
8	sunny	mild	high	false	0	1/14	0.0198	0.0385
9	sunny	cool	normal	false	1	1/14	0.0198	0.0385
10	rainy	mild	normal	false	1	1/14	0.0198	0.0385
11	sunny	mild	normal	true	1	1/14	0.0198	0.0385
12	overcast	mild	high	true	1	1/14	0.0198	0.0385
13	overcast	hot	normal	false	1	1/14	0.0198	0.0385
14	rainy	mild	high	true	0	1/14	0.0198	0.0385
							∑ 0.515	**∑ 1**

Step 6:

- Normalise the weights so that all the new sample weights add up to 1. This can be done by dividing each of the new sample weight by the sum of new sample weights
- In this example all weights add up to 0.515. So divide all sample weights with 0.515 to get the new normalised weights

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Weight	Updated Weight	Normalised Weight	Cumulative Sum of Norm. Weight	Buckets
1	sunny	hot	high	false	0	1/14	0.0198	0.0385	0.0385	0 to 0.0385
2	sunny	hot	high	true	0	1/14	0.0198	0.0385	0.0769	0.0385 to 0.0769
3	overcast	hot	high	false	1	1/14	0.0198	0.0385	0.1154	0.0769 to 0.1154
4	rainy	mild	high	false	1	1/14	0.2575	0.5000	0.6154	0.1154 to 0.6154
5	rainy	cool	normal	false	1	1/14	0.0198	0.0385	0.6538	0.6154 to 0.6538
6	rainy	cool	normal	true	0	1/14	0.0198	0.0385	0.6923	0.6538 to 0.6923
7	overcast	cool	normal	true	1	1/14	0.0198	0.0385	0.7307	0.6923 to 0.7307
8	sunny	mild	high	false	0	1/14	0.0198	0.0385	0.7692	0.7307 to 0.7692
9	sunny	cool	normal	false	1	1/14	0.0198	0.0385	0.8077	0.7692 to 0.8077
10	rainy	mild	normal	false	1	1/14	0.0198	0.0385	0.8461	0.8077 to 0.8461
11	sunny	mild	normal	true	1	1/14	0.0198	0.0385	0.8846	0.8461 to 0.8846
12	overcast	mild	high	true	1	1/14	0.0198	0.0385	0.9231	0.8846 to 0.9231
13	overcast	hot	normal	false	1	1/14	0.0198	0.0385	0.9615	0.9231 to 0.9615
14	rainy	mild	high	true	0	1/14	0.0198	0.0385	1.0000	0.9615 to 1

- Next, AdaBoost will create a new collection of samples based on the normalized updated weights
- In the new dataset, the misclassified samples will have a higher chance of being selected or added multiple times for retraining purpose
- The next stump will be created on this new dataset

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Weight	Updated Weight	Normalised Weight	Buckets
1	sunny	hot	high	false	0	1/14	0.0198	0.0385	0 to 0.0385
2	sunny	hot	high	true	0	1/14	0.0198	0.0385	0.0385 to 0.0769
3	overcast	hot	high	false	1	1/14	0.0198	0.0385	0.0769 to 0.1154
4	rainy	mild	high	false	1	1/14	0.2575	0.5000	0.1154 to 0.6154
5	rainy	cool	normal	false	1	1/14	0.0198	0.0385	0.6154 to 0.6538
6	rainy	cool	normal	true	0	1/14	0.0198	0.0385	0.6538 to 0.6923
7	overcast	cool	normal	true	1	1/14	0.0198	0.0385	0.6923 to 0.7307
8	sunny	mild	high	false	0	1/14	0.0198	0.0385	0.7307 to 0.7692
9	sunny	cool	normal	false	1	1/14	0.0198	0.0385	0.7692 to 0.8077
10	rainy	mild	normal	false	1	1/14	0.0198	0.0385	0.8077 to 0.8461
11	sunny	mild	normal	true	1	1/14	0.0198	0.0385	0.8461 to 0.8846
12	overcast	mild	high	true	1	1/14	0.0198	0.0385	0.8846 to 0.9231
13	overcast	hot	normal	false	1	1/14	0.0198	0.0385	0.9231 to 0.9615
14	rainy	mild	high	true	0	1/14	0.0198	0.0385	0.9615 to 1

Step 7:

- Start with an empty dataset. Randomly pick a number between 0 and 1
- If the number falls between 0 and 0.0385 then Sample 1 is added to the new dataset
- If the random number falls between 0.0385 and 0.0769 then Sample 2 is added to the new dataset
- If random number falls between 0.0769 and 0.1154 then Sample 3 is added to the new dataset, so on and so forth

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Weight	Updated Weight	Normalised Weight	Buckets
1	sunny	hot	high	false	0	1/14	0.0198	0.0385	0 to 0.0385
2	sunny	hot	high	true	0	1/14	0.0198	0.0385	0.0385 to 0.0769
3	overcast	hot	high	false	1	1/14	0.0198	0.0385	0.0769 to 0.1154
4	rainy	mild	high	false	1	1/14	0.2575	0.5000	0.1154 to 0.6154
5	rainy	cool	normal	false	1	1/14	0.0198	0.0385	0.6154 to 0.6538
6	rainy	cool	normal	true	0	1/14	0.0198	0.0385	0.6538 to 0.6923
7	overcast	cool	normal	true	1	1/14	0.0198	0.0385	0.6923 to 0.7307
8	sunny	mild	high	false	0	1/14	0.0198	0.0385	0.7307 to 0.7692
9	sunny	cool	normal	false	1	1/14	0.0198	0.0385	0.7692 to 0.8077
10	rainy	mild	normal	false	1	1/14	0.0198	0.0385	0.8077 to 0.8461
11	sunny	mild	normal	true	1	1/14	0.0198	0.0385	0.8461 to 0.8846
12	overcast	mild	high	true	1	1/14	0.0198	0.0385	0.8846 to 0.9231
13	overcast	hot	normal	false	1	1/14	0.0198	0.0385	0.9231 to 0.9615
14	rainy	mild	high	true	0	1/14	0.0198	0.0385	0.9615 to 1

Step 7: Continued

- For eg., in 1st iteration it selects a random weight of 0.74
- The algorithm will check this random number falls into which bucket and populate the corresponding record in the new dataset

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play
8	sunny	mild	high	false	0
4	rainy	mild	high	false	1
4	rainy	mild	high	false	1
11	sunny	mild	normal	true	1
4	rainy	mild	high	false	1

Step 7: Continued

Iteration 2: 0.12
Iteration 3: 0.54
Iteration 4: 0.85
Iteration 5: 0.32…

- AdaBoost will run 14 iterations to select different records from older dataset
- Probability is that the same record is selected multiple times

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Weight	Updated Weight	Normalised Weight	Buckets
1	sunny	hot	high	false	0	1/14	0.0198	0.0385	0 to 0.0385
2	sunny	hot	high	true	0	1/14	0.0198	0.0385	0.0385 to 0.0769
3	overcast	hot	high	false	1	1/14	0.0198	0.0385	0.0769 to 0.1154
4	rainy	mild	high	false	1	1/14	0.2575	0.5000	$\boxed{\text{0.1154 to 0.6154}}$
5	rainy	cool	normal	false	1	1/14	0.0198	0.0385	0.6154 to 0.6538
6	rainy	cool	normal	true	0	1/14	0.0198	0.0385	0.6538 to 0.6923
7	overcast	cool	normal	true	1	1/14	0.0198	0.0385	0.6923 to 0.7307
8	sunny	mild	high	false	0	1/14	0.0198	0.0385	0.7307 to 0.7692
9	sunny	cool	normal	false	1	1/14	0.0198	0.0385	0.7692 to 0.8077
10	rainy	mild	normal	false	1	1/14	0.0198	0.0385	0.8077 to 0.8461
11	sunny	mild	normal	true	1	1/14	0.0198	0.0385	0.8461 to 0.8846
12	overcast	mild	high	true	1	1/14	0.0198	0.0385	0.8846 to 0.9231
13	overcast	hot	normal	false	1	1/14	0.0198	0.0385	0.9231 to 0.9615
14	rainy	mild	high	true	0	1/14	0.0198	0.0385	0.9615 to 1

- The interval for the incorrect sample is bigger from 0.1154 to 0.6154, when compared to the correctly classified samples

- **Hence** the misclassified sample will be added multiple times, given its larger sample weight

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play
8	sunny	mild	high	false	0
4	rainy	mild	high	false	1
4	rainy	mild	high	false	1
11	sunny	mild	normal	true	1
4	rainy	mild	high	false	1
7	overcast	cool	normal	true	1
2	sunny	hot	high	true	0
14	rainy	mild	high	true	0
9	sunny	cool	normal	false	1
4	rainy	mild	high	false	1
1	sunny	hot	high	false	0
10	rainy	mild	normal	false	1
6	rainy	cool	normal	true	0
4	rainy	mild	high	false	1

14
Samples

Step 8:

- Add samples to the new dataset by choosing random numbers as explained in step 7, till the new dataset is the same size as the original dataset

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Weight
8	sunny	mild	high	false	0	1/14
4	rainy	mild	high	false	1	1/14
4	rainy	mild	high	false	1	1/14
11	sunny	mild	normal	true	1	1/14
4	rainy	mild	high	false	1	1/14
7	overcast	cool	normal	true	1	1/14
2	sunny	hot	high	true	0	1/14
14	rainy	mild	high	true	0	1/14
9	sunny	cool	normal	false	1	1/14
4	rainy	mild	high	false	1	1/14
1	sunny	hot	high	false	0	1/14
10	rainy	mild	normal	false	1	1/14
6	rainy	cool	normal	true	0	1/14
4	rainy	mild	high	false	1	1/14

New Dataset

Step 9:

- Get rid of the original samples and use the new collection of samples, for the next stump
- Give equal weights to the new samples, just like before

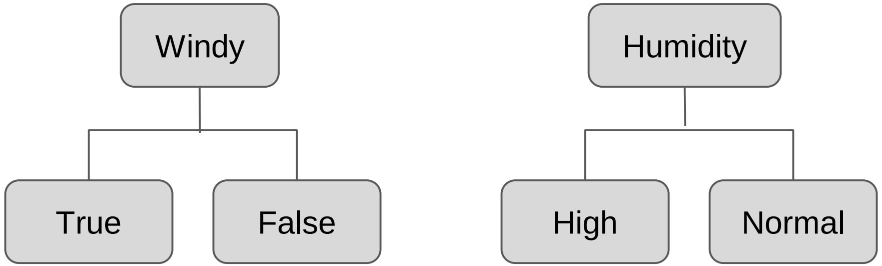
Based on this new dataset, a new stump is created

AdaBoost: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Weight
8	sunny	mild	high	false	0	1/14
4	rainy	mild	high	false	1	1/14
4	rainy	mild	high	false	1	1/14
11	sunny	mild	normal	true	1	1/14
4	rainy	mild	high	false	1	1/14
7	overcast	cool	normal	true	1	1/14
2	sunny	hot	high	true	0	1/14
14	rainy	mild	high	true	0	1/14
9	sunny	cool	normal	false	1	1/14
4	rainy	mild	high	false	1	1/14
1	sunny	hot	high	false	0	1/14
10	rainy	mild	normal	false	1	1/14
6	rainy	cool	normal	true	0	1/14
4	rainy	mild	high	false	1	1/14

As these samples are all the same, they would be treated as a block, creating a large penalty for being misclassified.

AdaBoost: how it works



- Go back to the beginning and find the variable to make the next stump that does the best job at classifying the new collection of sample

This is how AdaBoost works and this is how the errors of the first stump influences how the next stump is made!

AdaBoost: how it classifies

Stumps that Classify as YES	Amount of Say	Stumps that Classify as NO	Amount of Say
STUMP 1	0.26	STUMP 3	0.83
STUMP 2	0.74		
STUMP 4	0.45		
Total Say for YES	1.45	Total say for NO	0.83

- Suppose Adaboost created 4 stumps
- For a sample S1, the classification by each of the 4 stumps, and the ‘Amount of Say’ of each stump is shown in the table
- Based on the classification by each stump and its amount of say, the sample S1 will be classified as YES

Gradient Boosting

Gradient Boosting

- The idea originated by Leo Breiman and the framework further developed by Friedman

- Gradient Boosting can be thought of as an optimization problem where the objective is to minimise the loss of the model by adding weak base learners using a gradient descent method

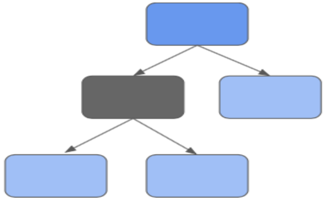
- It is a stage-wise additive algorithm because a new base model is added one at a time and the previously added models remain unchanged

Gradient Boosting

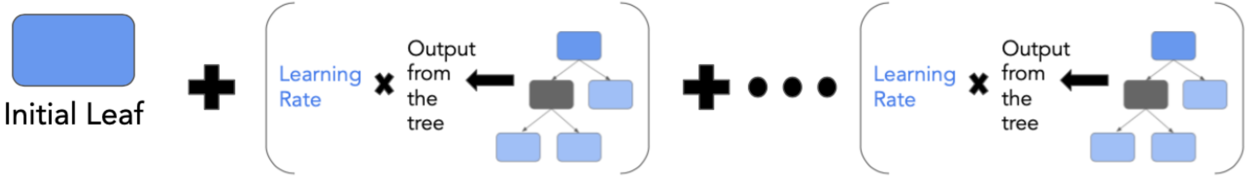
- Gradient Boosting works by adding one weak learner at a time
- The output from each weak tree is added sequentially to get the final prediction
- Thus Gradient boosting works by taking one small step in the right direction at a time!
- Also known as GBM, Gradient Boosting Machine

GBM: key components

- A loss function that needs to be optimized eg. Log Likelihood function
- The base models for making predictions



- An additive model to add these base models such that they reduce the loss function



GBM: key concept 1

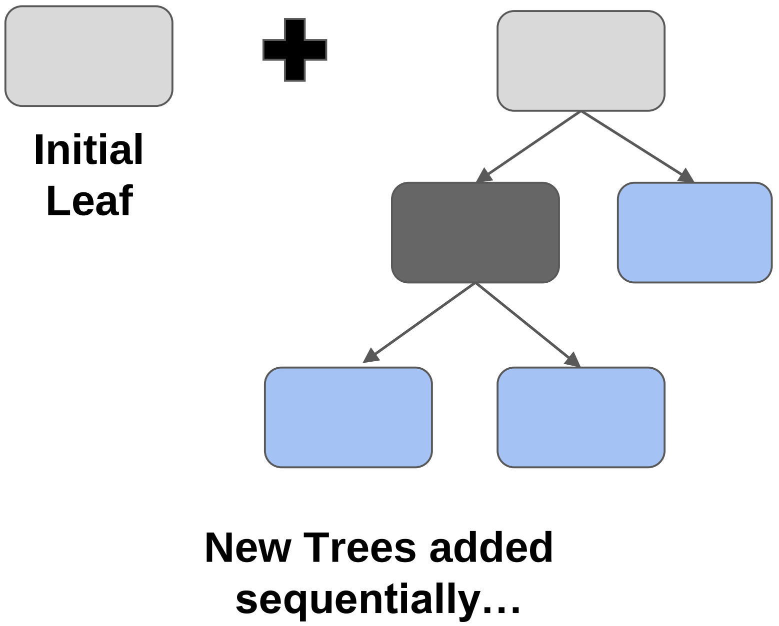
GBM starts with a leaf, which is the initial prediction for all the samples

Initial Leaf

GBM: key concept 2

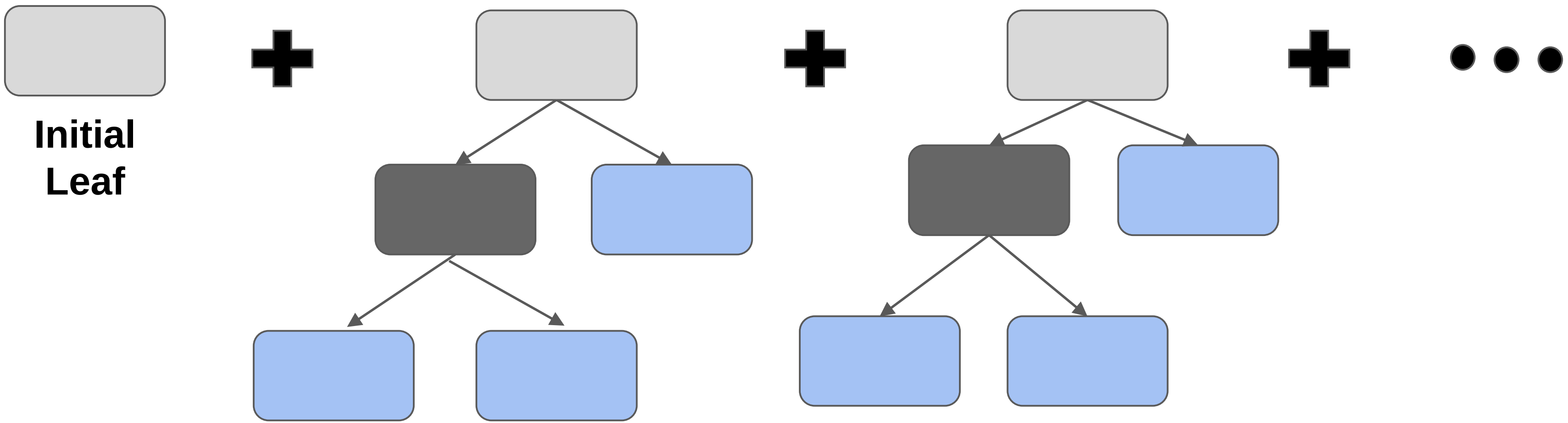
- GBM builds a tree which (like Adaboost) is based on the errors made by the previous weak learner

- In practice, the size of the tree is usually restricted to 4 to 8 levels or between 8 to 32 leaves



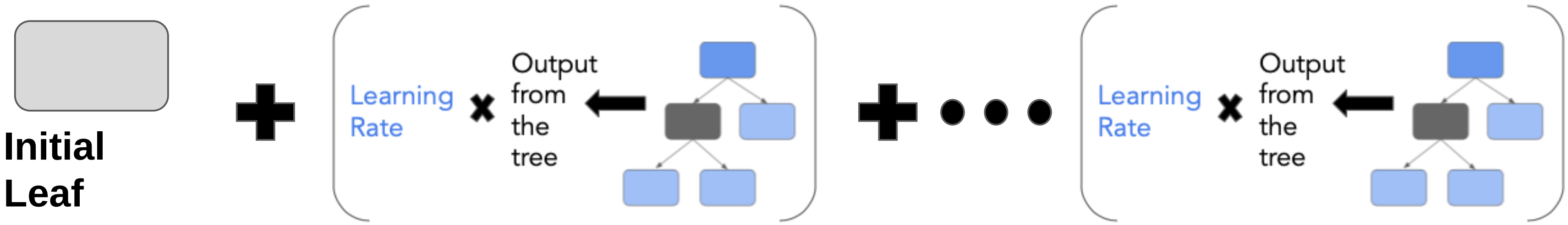
GBM: key concept 3

New trees are added sequentially at each step and the existing trees remain unchanged



GBM: key concept 4

The output from each tree is optimised by a learning rate and added sequentially, to the initial prediction



GBM: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	false	0
2	sunny	hot	high	true	0
3	overcast	hot	high	false	1
4	rainy	mild	high	false	1
5	rainy	cool	normal	false	1
6	rainy	cool	normal	true	0
7	overcast	cool	normal	true	1
8	sunny	mild	high	false	0
9	sunny	cool	normal	false	1
10	rainy	mild	normal	false	1
11	sunny	mild	normal	true	1
12	overcast	mild	high	true	1
13	overcast	hot	normal	false	1
14	rainy	mild	high	true	0

- Target Variable: Play
- Independent Variables: Outlook, Temperature, Humidity, Windy

GBM: how it works

Step 1:

- Start with an initial leaf which is the initial prediction for all samples
- Calculate the odds of play = 9/5 =1.8. **Log(Odds)** = 0.5877
- Convert Log(Odds) to probability by following formula:

$$Probability = \frac{e^{\log(Odds)}}{1+e^{\log(Odds)}}$$

- Plugging values in the above formula, **Probability** = 0.64

0.64

Initial Leaf

GBM: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Probability	Residual
1	sunny	hot	high	false	0	0.64	-0.64
2	sunny	hot	high	true	0	0.64	-0.64
3	overcast	hot	high	false	1	0.64	0.36
4	rainy	mild	high	false	1	0.64	0.36
5	rainy	cool	normal	false	1	0.64	0.36
6	rainy	cool	normal	true	0	0.64	-0.64
7	overcast	cool	normal	true	1	0.64	0.36
8	sunny	mild	high	false	0	0.64	-0.64
9	sunny	cool	normal	false	1	0.64	0.36
10	rainy	mild	normal	false	1	0.64	0.36
11	sunny	mild	normal	true	1	0.64	0.36
12	overcast	mild	high	true	1	0.64	0.36
13	overcast	hot	normal	false	1	0.64	0.36
14	rainy	mild	high	true	0	0.64	-0.64

Step 2:

- Compute the Residuals

$$Residual = Observed - Predicted$$

- For eg, for sample 1, the residual = 0 - 0.64 = -0.64

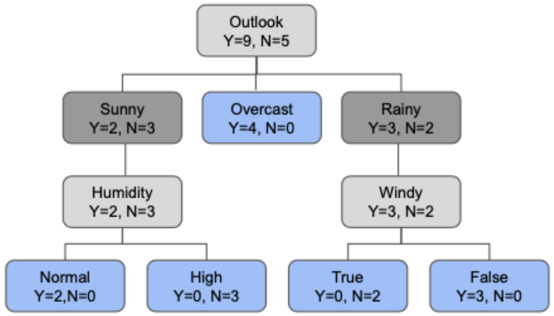
- Similarly, calculate residual for all the samples

GBM: how it works

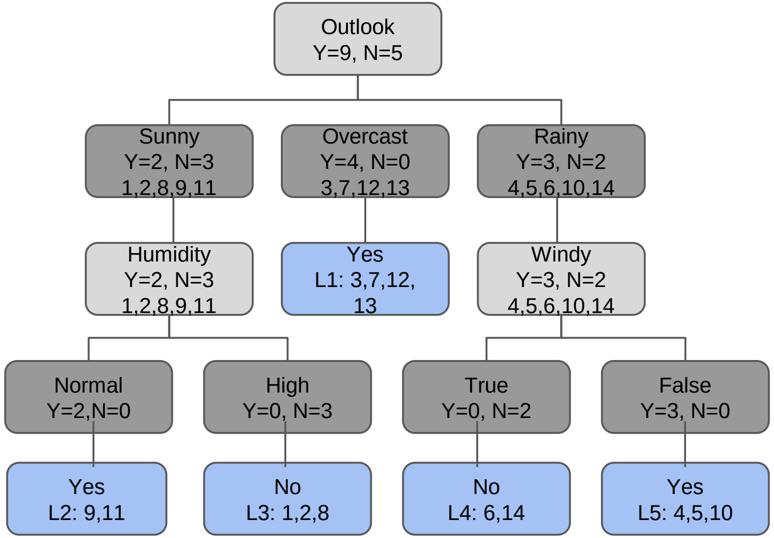
Observation	Outlook	Temperature	Humidity	Windy	Play	Initial Probability	Residual
1	sunny	hot	high	false	0	0.64	-0.64
2	sunny	hot	high	true	0	0.64	-0.64
3	overcast	hot	high	false	1	0.64	0.36
4	rainy	mild	high	false	1	0.64	0.36
5	rainy	cool	normal	false	1	0.64	0.36
6	rainy	cool	normal	true	0	0.64	-0.64
7	overcast	cool	normal	true	1	0.64	0.36
8	sunny	mild	high	false	0	0.64	-0.64
9	sunny	cool	normal	false	1	0.64	0.36
10	rainy	mild	normal	false	1	0.64	0.36
11	sunny	mild	normal	true	1	0.64	0.36
12	overcast	mild	high	true	1	0.64	0.36
13	overcast	hot	normal	false	1	0.64	0.36
14	rainy	mild	high	true	0	0.64	-0.64

Step 3:

Build a tree for the residuals



GBM: how it works



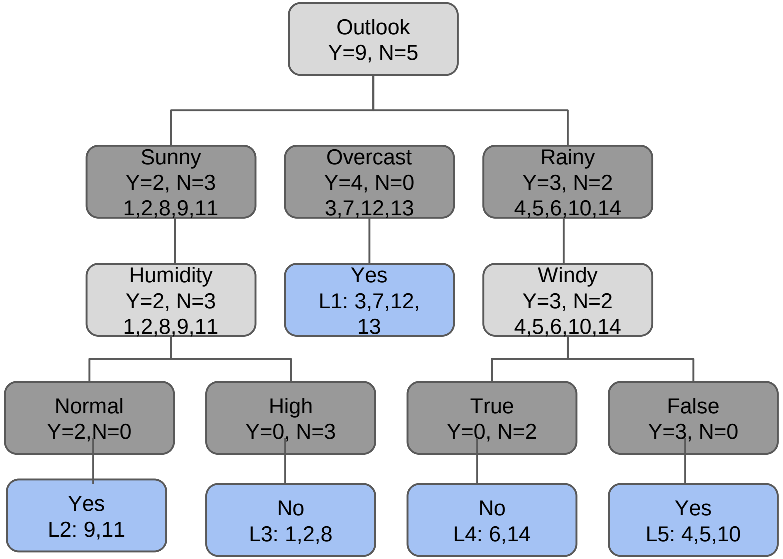
Step 4:

- Calculate the output(in terms of log(Odds)) for each leaf of the tree as follows:

$$\gamma = \frac{\sum(residuals)}{\sum(p(1-p))}$$

- **For e.g. Output of Leaf L1**
  = (0.36*4)/ ((0.64*0.36) * 4)
  = 1.56

GBM: how it works

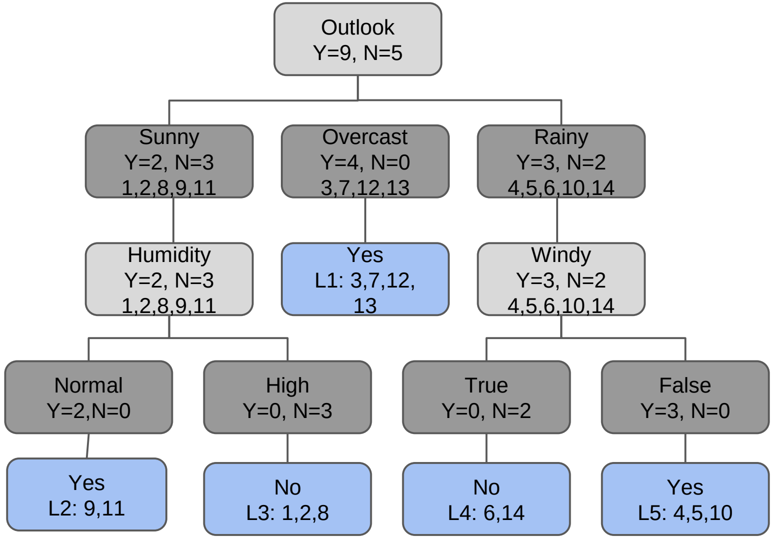


Step 4: continued...

Output of all the Leaf node is tabulated below:

Samples in Leaf Nodes	Output
L1: 3,7,12, 13	1.56
L2: 9, 11	1.56
L3: 1, 2, 8	-2.78
L4: 6, 14	-2.78
L5: 4, 5, 10	1.56

GBM: how it works



Step 5:

- Scale the output by learning rate
- It is a common practice to use learning rate = 0.1
- In this example learning rate is taken as 0.6

Samples in Leaf Nodes	Output	Learning Rate	O/P * LR
L1: 3,7,12, 13	1.56	0.6	0.94
L2: 9, 11	1.56	0.6	0.94
L3: 1, 2, 8	-2.78	0.6	-1.67
L4: 6, 14	-2.78	0.6	-1.67
L5: 4, 5, 10	1.56	0.6	0.94

GBM: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initail Log(Odds)	Initial Probability	Residual	New Log(Odds)
1	sunny	hot	high	false	0	0.59	0.64	-0.64	-1.08
2	sunny	hot	high	true	0	0.59	0.64	-0.64	-1.08
3	overcast	hot	high	false	1	0.59	0.64	0.36	1.53
4	rainy	mild	high	false	1	0.59	0.64	0.36	1.53
5	rainy	cool	normal	false	1	0.59	0.64	0.36	1.53
6	rainy	cool	normal	true	0	0.59	0.64	-0.64	-1.08
7	overcast	cool	normal	true	1	0.59	0.64	0.36	1.53
8	sunny	mild	high	false	0	0.59	0.64	-0.64	-1.08
9	sunny	cool	normal	false	1	0.59	0.64	0.36	1.53
10	rainy	mild	normal	false	1	0.59	0.64	0.36	1.53
11	sunny	mild	normal	true	1	0.59	0.64	0.36	1.53
12	overcast	mild	high	true	1	0.59	0.64	0.36	1.53
13	overcast	hot	normal	false	1	0.59	0.64	0.36	1.53
14	rainy	mild	high	true	0	0.59	0.64	-0.64	-1.08

Step 6:

- Update the prediction,as
- New Log(Odds) = Log(Odds) prediction by Initial leaf + (O/P value of 1st Tree * Learning Rate)

GBM: how it works

Observation	Outlook	Temperature	Humidity	Windy	Play	Initail Log(Odds)	Initial Probability	Residual	New Log(Odds)	New Probability
1	sunny	hot	high	false	0	0.59	0.64	-0.64	-1.08	0.25
2	sunny	hot	high	true	0	0.59	0.64	-0.64	-1.08	0.25
3	overcast	hot	high	false	1	0.59	0.64	0.36	1.53	0.82
4	rainy	mild	high	false	1	0.59	0.64	0.36	1.53	0.82
5	rainy	cool	normal	false	1	0.59	0.64	0.36	1.53	0.82
6	rainy	cool	normal	true	0	0.59	0.64	-0.64	-1.08	0.25
7	overcast	cool	normal	true	1	0.59	0.64	0.36	1.53	0.82
8	sunny	mild	high	false	0	0.59	0.64	-0.64	-1.08	0.25
9	sunny	cool	normal	false	1	0.59	0.64	0.36	1.53	0.82
10	rainy	mild	normal	false	1	0.59	0.64	0.36	1.53	0.82
11	sunny	mild	normal	true	1	0.59	0.64	0.36	1.53	0.82
12	overcast	mild	high	true	1	0.59	0.64	0.36	1.53	0.82
13	overcast	hot	normal	false	1	0.59	0.64	0.36	1.53	0.82
14	rainy	mild	high	true	0	0.59	0.64	-0.64	-1.08	0.25

Step 7:

Convert Log(Odds) to probability by following formula:

$$Probability = \frac{e^{\log(Odds)}}{1+e^{\log(Odds)}}$$

GBM: how it works

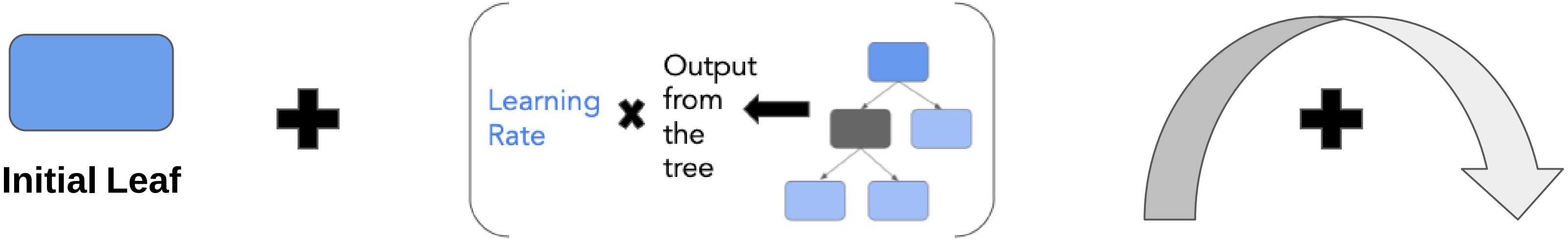
Observation	Outlook	Temperature	Humidity	Windy	Play	Initail Log(Odds)	Initial Probability	Residual	New Log(Odds)	New Probability	New Residuals
1	sunny	hot	high	false	0	0.59	0.64	-0.64	-1.08	0.25	-0.25
2	sunny	hot	high	true	0	0.59	0.64	-0.64	-1.08	0.25	-0.25
3	overcast	hot	high	false	1	0.59	0.64	0.36	1.53	0.82	0.18
4	rainy	mild	high	false	1	0.59	0.64	0.36	1.53	0.82	0.18
5	rainy	cool	normal	false	1	0.59	0.64	0.36	1.53	0.82	0.18
6	rainy	cool	normal	true	0	0.59	0.64	-0.64	-1.08	0.25	-0.25
7	overcast	cool	normal	true	1	0.59	0.64	0.36	1.53	0.82	0.18
8	sunny	mild	high	false	0	0.59	0.64	-0.64	-1.08	0.25	-0.25
9	sunny	cool	normal	false	1	0.59	0.64	0.36	1.53	0.82	0.18
10	rainy	mild	normal	false	1	0.59	0.64	0.36	1.53	0.82	0.18
11	sunny	mild	normal	true	1	0.59	0.64	0.36	1.53	0.82	0.18
12	overcast	mild	high	true	1	0.59	0.64	0.36	1.53	0.82	0.18
13	overcast	hot	normal	false	1	0.59	0.64	0.36	1.53	0.82	0.18
14	rainy	mild	high	true	0	0.59	0.64	-0.64	-1.08	0.25	-0.25

Step 8:

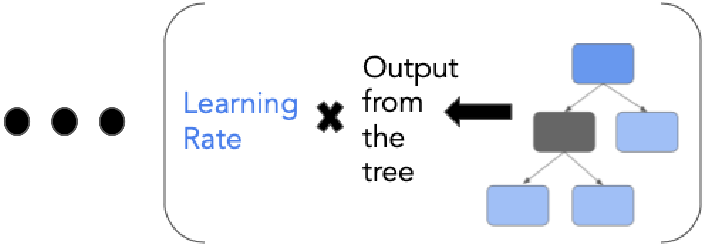
- Calculate the Residuals again
- Build the second tree
- GBM repeats these steps until it has built the specified number of trees or the residuals are very small or reach a threshold
- i.e., adding new trees do not improve the fit

Thus GBM works, by taking small steps in the right direction!

GBM: how it classifies



- GBM makes prediction by adding predictions from each tree to the initial prediction
- Output from each tree is scaled by Learning Rate
- Convert this log(Odds)prediction to probability
- Based on the decided threshold, classify



AdaBoost Vs Gradient Boosting

AdaBoost Vs GBM

Similarities

In both algorithms:

1. Decision trees are the base learners

2. Trees are built based on the errors made by the previous trees

3. The size of the tree is restricted

4. Trees are scaled

AdaBoost Vs GBM

Differences

AdaBoost	Gradient Boost
Mostly made of stumps (Tree with a root and two leaves)	Starts with a leaf. Then build trees with 8 to 32 leaves. So, it does restrict the size of the trees
Stumps are scaled such that each stump has different amount of say in the final prediction	Output from each tree is scaled by a learning rate, however all trees have an equal amount of say in the final prediction
Shortcoming of the base learner is identified by high-weight samples	Shortcoming of the base learner is identified by gradients

did you know?

Bagging and Boosting both use trees as base learners to build more powerful models!

Question:

Is Boosting limited to classifications problems only?

Solution:

Boosting is a machine learning technique for classification and regression problems.

XGBoost

XGBoost

- It is popularly said in the Machine learning world that, “When in doubt, use XGBoost”!

- The two main reasons for the success and popularity of XGBoost are:

  1. Execution speed - It is way faster when compared to other boosting implementations.

  2. Model performance - It is the top performing algorithm for structured datasets for both classification and regression predictive modelling problems.

XGBoost

- XGBoost is eXtreme Gradient Boosting
- Developed at the University of Washington as a project
- XGBoost is much faster than GBM
- XGBoost is a combination of both, hardware and software optimization techniques, to provide better results using less computing resources in a short amount of time
- XGBoost is designed to be used with very large and complicated datasets

XGBoost: key features

- Similar to Gradient Boost
- Regularization
- A unique decision tree
- Approximate Greedy algorithm
- Weighted Quantile Sketch
- Sparsity-Aware split finding
- Parallel learning
- Cache-Aware Access
- Blocks for Out of Core Computations

XGBoost: how it works

Student	Hours of Study	Pass/ Fail
A	2	No
B	8	Yes
C	12	Yes
D	17	No

Sample dataset:

- Target Variable: Pass
- Independent Variables: Hours of Study

XGBoost: how it works

Student	Hours of Study	Pass/ Fail	Initial Probability
A	2	No	0.5
B	8	Yes	0.5
C	12	Yes	0.5
D	17	No	0.5

0.50

Initial leaf

Step 1:

- Just like GBM, XGBoost starts with a Leaf. XGBoost makes an initial prediction through this leaf
- The initial prediction can be any value (target rate), by default it is 0.5
- In this dataset the target rate is 0.5 i.e., there is a 50% probability for a student to pass
- Initial prediction = 0.5

XGBoost: how it works

Student	Hours of Study	Pass/ Fail	Initial Probability	Residual
A	2	No	0.5	-0.5
B	8	Yes	0.5	0.5
C	12	Yes	0.5	0.5
D	17	No	0.5	-0.5

Step 2:

- Calculate the Residual as:

  $$Residual = Observed - Predicted$$

- Residuals show how good or bad the prediction is!

XGBoost: how it works

Step 3:

- Build a tree for the residuals
- Just like GBM, XGBoost fits a tree to the residuals. However XGBoost builds trees which are different from the trees built in GBM

Step 3.1:

- Start as a single leaf. All residuals go to this single leaf

-0.5, 0.5, 0.5, -0.5

Note: There are many ways to build the tree. The most common way to build the XGBoost trees is discussed here.

XGBoost: how it works

Step 3.2:

- Calculate the Similarity Score for the 1st leaf as per the below formula.
  Note: Since we do not square the residuals before adding them together, most of the large +/- residuals cancel out each other

$$Similarity = \frac{(SumResidual_i)^2}{\sum[PreviousProbability_i(1-PreviousProbability_i)]+\Lambda}$$

- It is a metric used by XGBoost to calculate the Gain. The variable/ threshold which gives the maximum gain is used to split the data

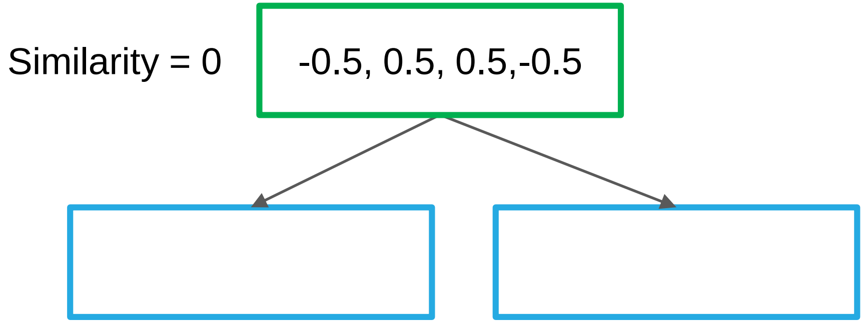
- Here lambda (**λ**) is the Regularisation parameter used to enhance the stability and the accuracy of the model

XGBoost: how it works

Step 3.2:

- Plugging in values, for the first leaf in the formula (on slide 72), Similarity = 0

Student	Hours of Study	Pass/ Fail	Initial Probability	Residual
A	2	No	0.5	-0.5
B	8	Yes	0.5	0.5
C	12	Yes	0.5	0.5
D	17	No	0.5	-0.5



- Next, split the leaf into two groups, clustering similar residuals

XGBoost: how it works

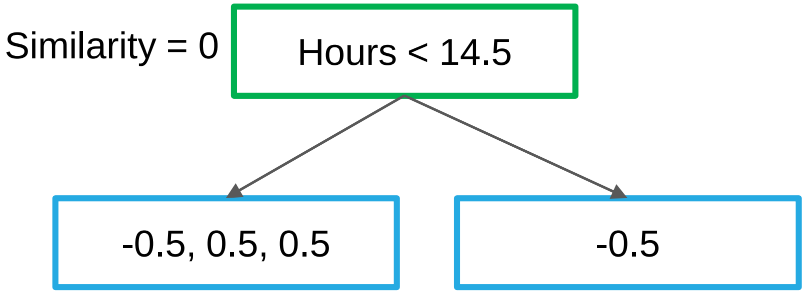
Student	Hours of Study	Pass/ Fail	Initial Probability	Residual
A	2	No	0.5	-0.5
B	8	Yes	0.5	0.5
C	12	Yes	0.5	0.5
D	17	No	0.5	-0.5

Average = 14.5

Note: We can choose any value of `Hours` between 12 and 17 as a threshold, to obtain the maximum gain

Step 3.3:

- Choose a threshold to split the leaf such that the gain will be the highest, for our example consider Hours < 14.5

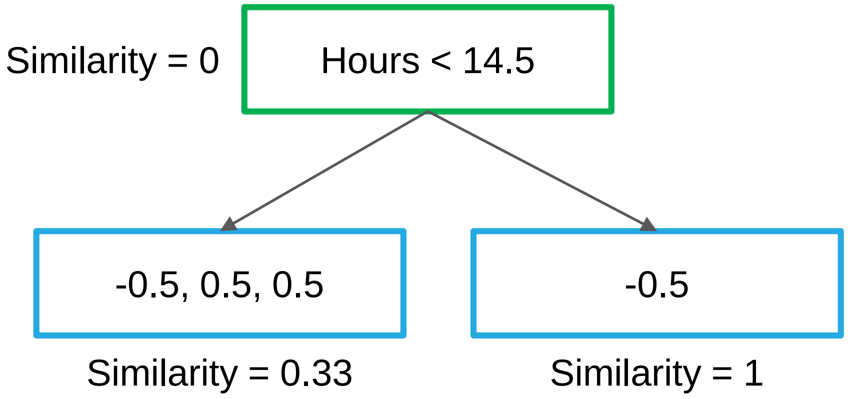


- The residuals for the 3 obs. less than 14.5, end up in the left child node

XGBoost: how it works

Step 3.4:

- Calculate the Similarity for left and right child nodes, using Similarity formula (on slide 72)



Similarity Score:

- Left Child = ((-0.5+0.5+0.5)^2) / 3*(0.5*(1-0.5)

  =0.25/3*0.25

  =0.33

- Right Child = ((-0.5)^2)/(0.5*(1-0.5))

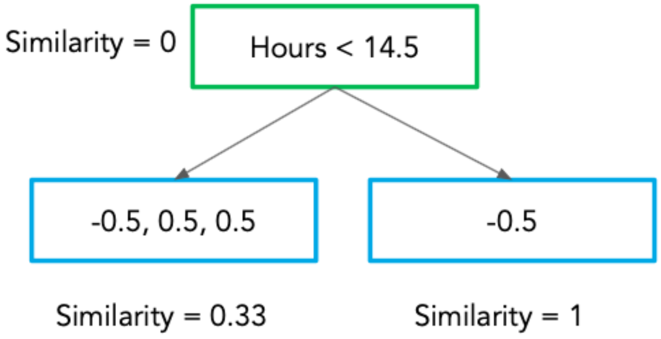
  =1

XGBoost: how it works

Step 3.5:

- Calculate Gain for the branch as:

$$\text{Gain} = \text{Left}_{\text{Similarity}} + \text{Right}_{\text{Similarity}} - \text{Root}_{\text{Similarity}}$$

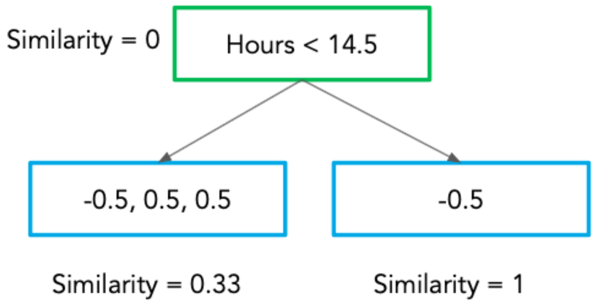


- Gain helps XGBoost determine how to split the data
- Plug in the values, Gain= 0.33+1-0 = 1.33

XGBoost: how it works

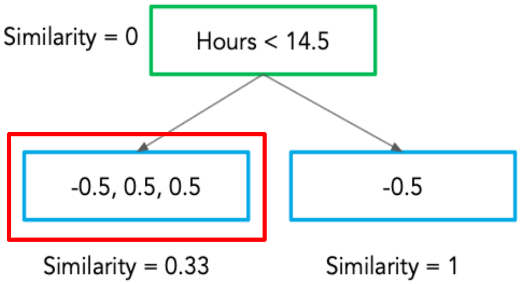
Step 3.5: Continued...

- Similarly, XGBoost calculates Gain for split by other threshold values
- The threshold which gives maximum Gain is chosen for the split
- Here, threshold Hours<14.5, gives maximum Gain, hence it becomes the 1st branch of the tree



XGBoost: how it works

Student	Hours of Study	Pass/ Fail	Initial Probability	Residual
A	2	No	0.5	-0.5
B	8	Yes	0.5	0.5
C	12	Yes	0.5	0.5
D	17	No	0.5	-0.5



Step 3.5:

- Next, split the residuals in the child nodes
- Split residuals in the left node for threshold which gives the max Gain, for e.g.
  - Threshold Hours < 10, or
  - For Threshold <5

XGBoost: how it works

Similarity = 0.33

Hours < 10

-0.5, 0.5

Similarity = 0

(-0.5+0.5)^2/ 0.5*(1-0.5)=0

0.5

Similarity = 1

(0.5)^2/ 0.5*(1-0.5)=1

Gain = 0 + 1 - 0.33 = 0.66

Step 3.5: Continued…

For Threshold Hours < 10,

Gain = 0 +1- 0.33 = 0.66

XGBoost: how it works

Similarity = 0.33

Hours < 5

-0.5

Similarity = 1

(-0.5)^2/ 0.5*(1-0.5)=1

0.5,0.5

Similarity = 2

(0.5+0.5)^2/(2 * 0.5*(1-0.5))=2

Gain = 1 + 2 - 0.33 = 2.66

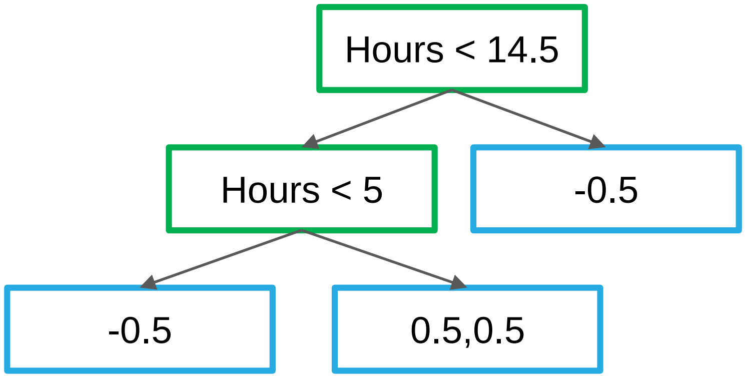
Step 3.5: Continued…

- For Threshold Hours < 5,
  Gain = 1 + 2 - 0.33 = 2.66
- As threshold for, Hours < 5 results in higher Gain, next split is basis this threshold

XGBoost: how it works

Step 3.5: Continued…

- This is the 1st XGBoost tree
- In this example we limit the tree to 2 levels

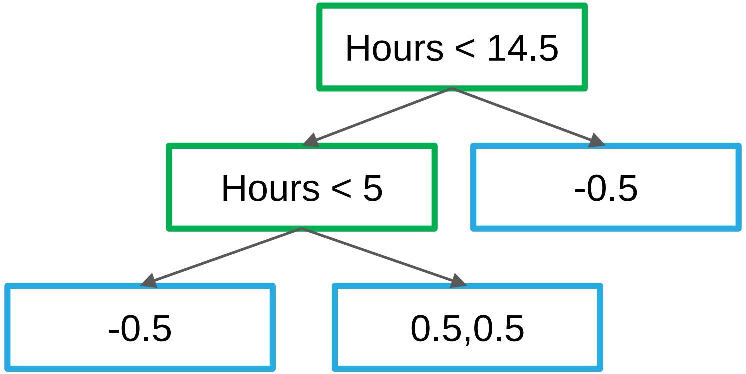


XGBoost: key concept

XGBoost limits the size of the trees by the following parameters:

- XGBoost limits the level of trees, by parameter **max_depth**. It is the maximum number of levels in a tree. Default = 6

- XGBoost continues branching the residuals in each leaf until it reaches the threshold for a minimum number of observations in a leaf, min_child_weight also, called Cover



XGBoost: key concept

- The minimum number of residuals in a leaf are related to a metrics called Cover

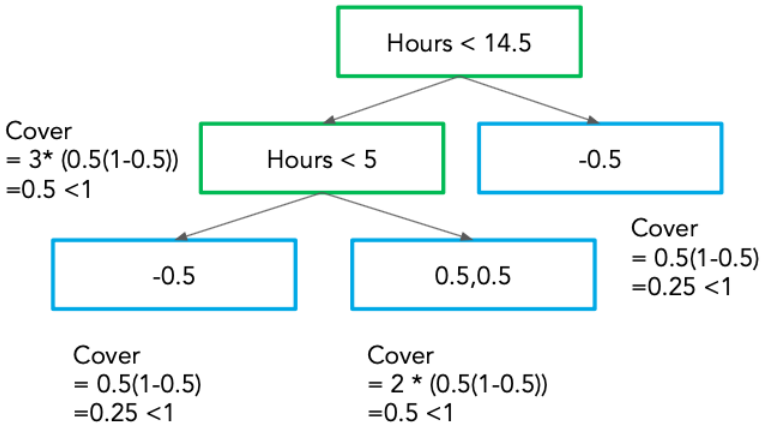
$$Similarity = \frac{(SumResidual_i)^2}{\underbrace{\boxed{\sum[PreviousProbability_i(1-PreviousProbability_i)]}}_{\textbf{Cover}}+\Lambda}$$

- It is the denominator of the similarity score, without $\boldsymbol{\lambda}$
- Cover is dependent on the previously predicted probability for each residual in the leaf
- The default value for cover is 1. Cover is equivalent to parameter -> **min_child_weight**

XGBoost: how it works

Step 3.6:

- Compute cover for all leaf nodes, starting with the lowest branch
- Compare with the default value of Cover =1
- If cover for a leaf < Default Cover then, XGBoost would not allow those leaves
- Cover for the leaves in this e.g. is <1. Hence, XGBoost would not allow these leaves, and we would be left with the Root node only
- Hence for this e.g. we set Cover or **min_child_weight = 0**



XGBoost: key concept

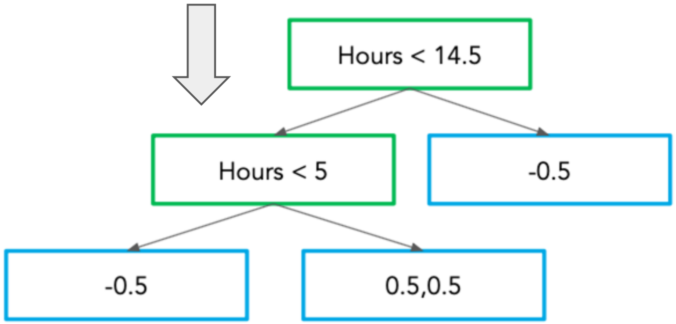
- γ (gamma) is the Tree Complexity Parameter which helps in controlling overfitting by the tree
- It is the minimum reduction in loss required to make a further partition on a leaf node of a tree
- It is a user defined value and can be any value set by the user, e.g. 1, 2 etc
- XGBoost prunes the tree by calculating the difference between Gain of lowest branch and γ
- If (Gain - γ) is a positive number then it does not prune
- If (Gain - γ) is a negative number then it prunes the branch

XGBoost: how it works

Step 3.7:

- Prune the tree by calculating the difference between the Gain of the lowest branch and **γ** (gamma)
- If (Gain - **γ**) is a positive number then do not prune
- If (Gain - **γ**) is a negative number then prune the branch
- If **γ** = 2 then the branch is not pruned as, 2.66 - 2 = 0.66 (Gain - **γ** is a positive value)
- If **γ** = 3 then the branch is pruned as, 2.66 - 3 = -0.34 (Gain - **γ** is a negative value)

Gain for this node = 2.66

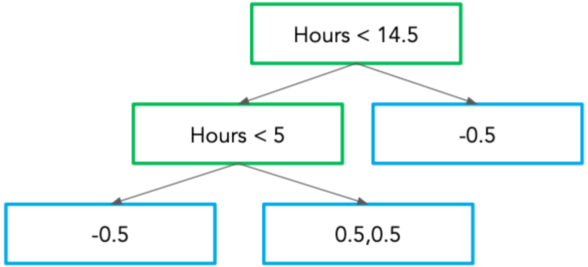


XGBoost: how it works

Step 3.8:

- Calculate the Output value for each leaf of the tree as per the below formula.

$$Output = \frac{(\sum Residual_i)}{\sum[PreviousProbability_i(1 - PreviousProbability_i)] + \Lambda}$$



- This Output formula is the same as used in GBM, minus the Lambda (Regularisation Parameter)

XGBoost: key concept

- λ Lambda is the regularisation parameter

- It reduces the Similarity Scores. Lower similarity scores lead to a lower value of Gain. So even smaller values of γ will result in a negative number and hence makes it easier to prune the tree

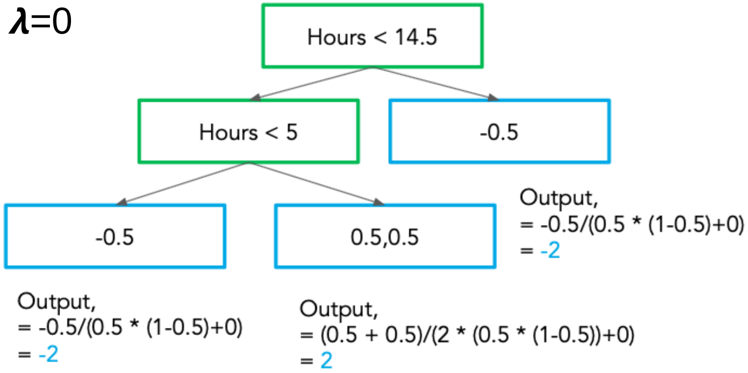
- Values of λ greater than 0 reduce the sensitivity of the tree to individual observations

XGBoost: how it works

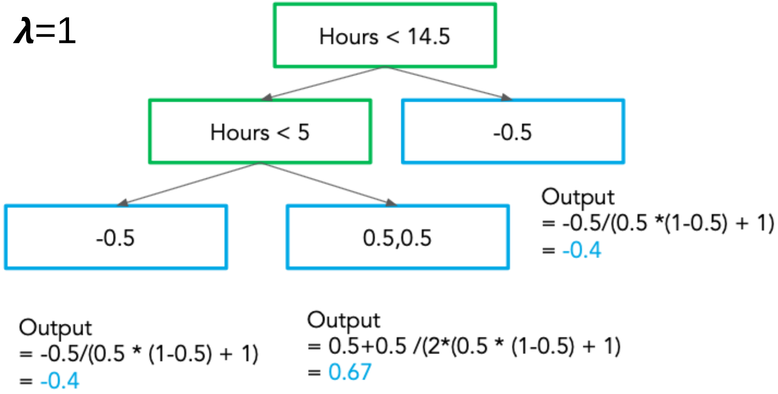
Step 3.8: Continued...

- Notice the difference in the output values when λ = 0 and λ = 1
- Output values are closer to Observed values when λ is greater than 0

λ=0

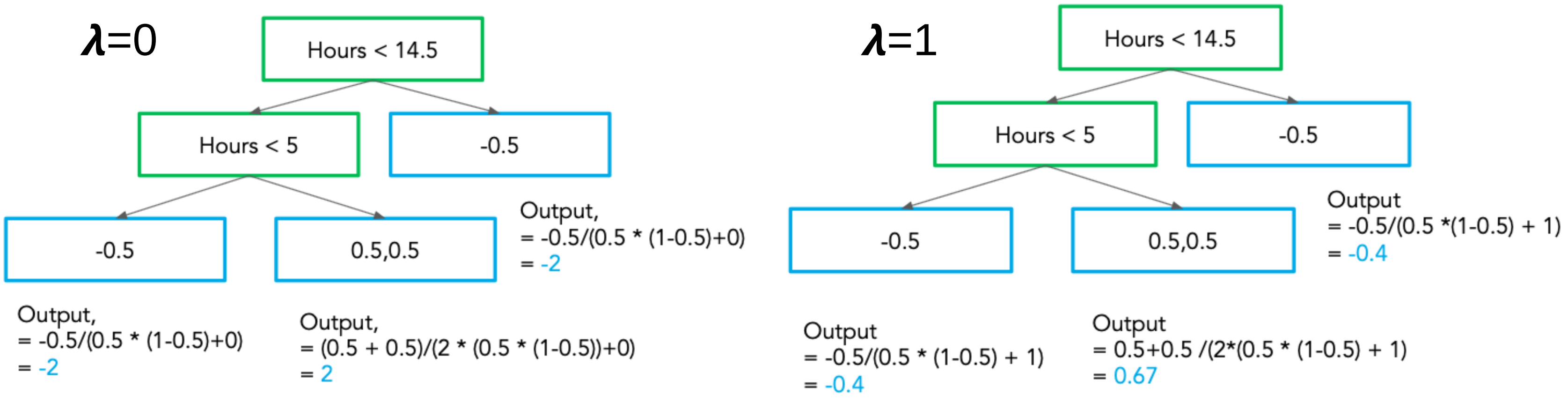


λ=1

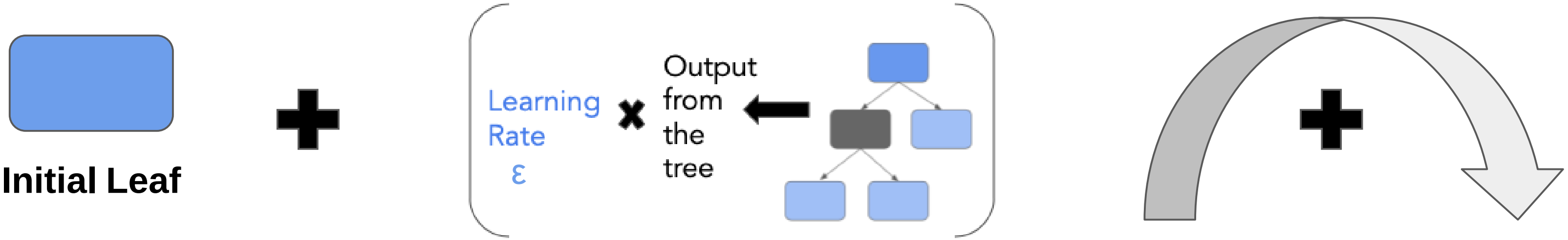


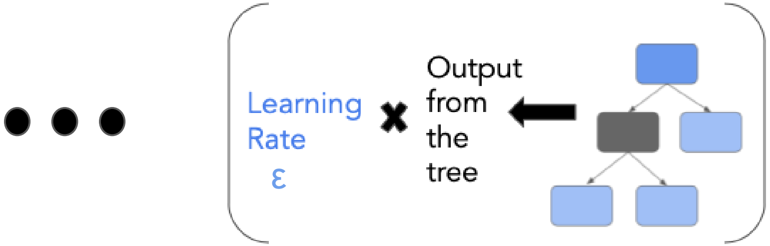
XGBoost: key concept

When λ is greater than 0 it reduces the amount that a single obs adds to the new prediction. Thus it reduces the prediction sensitivity to isolated observation!



XGBoost: how it predicts





- Like GBM, XGBoost makes a new prediction by adding the predictions from each tree to the initial prediction
- The output from each tree is scaled by a Learning Rate ε (Eta). By default it's value is 0.3

XGBoost: how it works

Step 4: Predict

- XGBoost makes a new prediction by starting with the initial leaf
- Convert the initial prediction to log(Odds)
- As we know,

$$Odds = \frac{P}{1-P}$$

$$\therefore \log(Odds) = \log\left(\frac{P}{1-P}\right)$$

- log(Odds) of Initial Prediction = log(0.5/(1-0.5)) = Log(1) = 0

Initial leaf

$\Rightarrow$ $\boxed{0.50}$

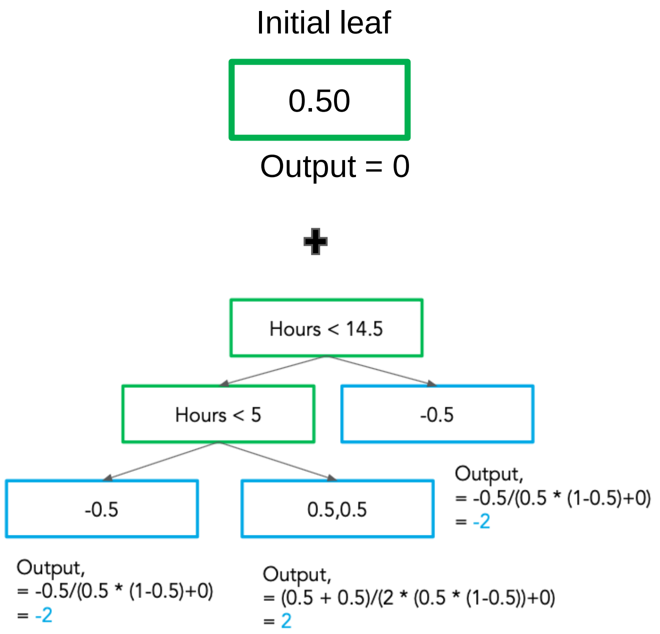
XGBoost: how it works

Step 4: Continued…

- Add the output from 1st tree, scaled by learning rate to the initial prediction

- Convert Log(Odds) prediction to probability, using the below formula:

$$Probability = \frac{e^{\log(Odds)}}{1+e^{\log(Odds)}}$$

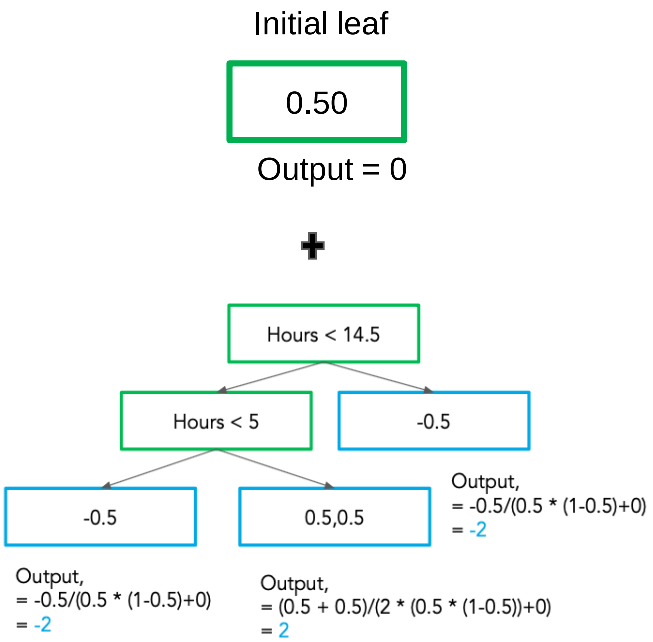


XGBoost: how it works

Step 4: Continued…

- Taking the example of the left-most leaf,
  Prediction = 0 + 0.3 * - 2 = - 0.6
- Convert this log(Odds) prediction to probability, so new predicted probability = 0.35
- Observed Value for this leaf was 0
- Initial Prediction from the leaf was 0.5
- New Prediction after adding the output from the 1st tree is 0.35

Thus, taken a small step in the right direction!



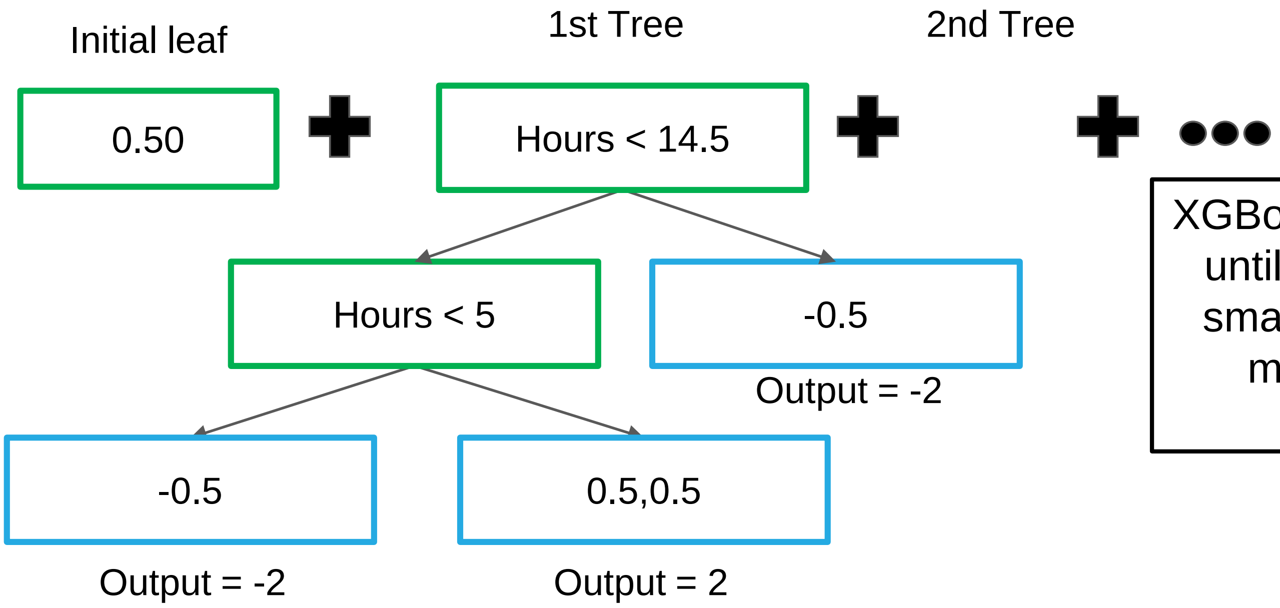
XGBoost: how it works

Student	Hours of Study	Pass/ Fail	Initial Probability	Residual	New Predicted Probability	New Residual
A	2	No	0.5	-0.5	0.35	-0.35
B	8	Yes	0.5	0.5	0.65	0.35
C	12	Yes	0.5	0.5	0.65	0.35
D	17	No	0.5	-0.5	0.35	-0.35

Step 5:

- Similarly, compute the prediction for other leaf nodes
- Notice that the new residuals are smaller than the previous residuals
- XGBoost will build a new tree to fit the new residuals

XGBoost: how it works



XGBoost keeps building trees until the residuals are very small or it has reached the maximum no. of trees specified.

Why eXtreme Gradient Boost?

- According to the authors of ‘xgboost’ it ”automatically does parallel computation on a single machine which could be more than 10 times faster than existing gradient boosting packages”

- XGBoost scales beyond billions of examples using far fewer resources than existing systems

In the words of Tianqi Chen,

- “The name xgboost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms.”

- “A more regularized model formalization to control over-fitting, which gives it better performance”

WANT TO KNOW MORE?

Read about:

- LightGBM framework released by Microsoft Research

- CatBoost developed by Yandex Technology

Thank You