**XGBoost** stands for **"Extreme Gradient Boosting"**. XGBoost is being used as a black box model in many machine learning competitions. As such, the purpose of this article is to explain the Gradient Boosting in plain english.

***What is boosting ?.***

In Bagging, trees are built parallelly, but in boosting, trees are built sequentially such that each tree is minimising the error produced in its previous tree. Each tree tries to learn from its predecessor and it updates the residual errors. In this way, the tree that is generated in the next sequences will be learning from the updated residuals.

Let's start with a simple example , we will predict a person age based on whether they ***LIKE/PLAYS CRICKET , LIKES TO WAKE UP EARLY IN MORNING*** and whether they ***LIKE CHINESE FOOD*** or not**.** Our objective here will be to minimise the squared error. For this simple blog , we will have are having below sample data.

| PersonID | Age | PlaysCricketGAME | LikesToWakeUpEarly | LikesChineseFood |
|---|---|---|---|---|
| 101 | 14 | TRUE | FALSE | TRUE |
| 102 | 15 | TRUE | FALSE | FALSE |
| 103 | 16 | TRUE | FALSE | FALSE |
| 104 | 25 | TRUE | TRUE | TRUE |
| 105 | 35 | TRUE | FALSE | TRUE |
| 106 | 50 | FALSE | TRUE | FALSE |
| 107 | 70 | TRUE | TRUE | TRUE |
| 108 | 73 | FALSE | TRUE | FALSE |
| 109 | 75 | FALSE | TRUE | TRUE |

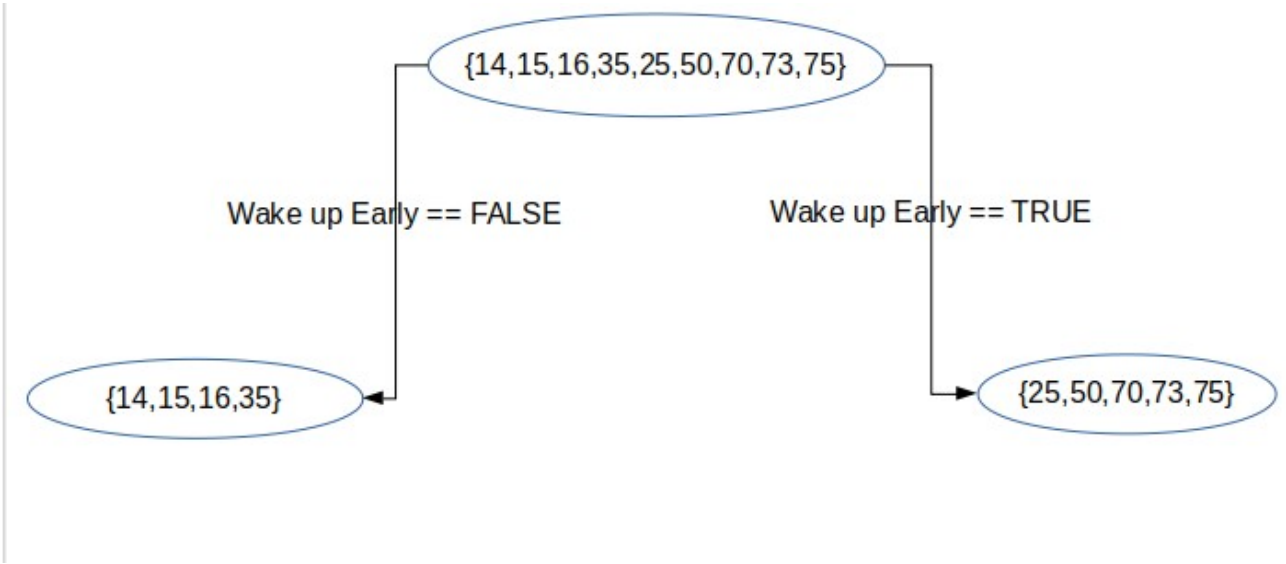
By looking at the data and all the features of the data we can say that

- people who ***LIKE TO WAKE UP EARLY*** in the morning are probably older .
- people who ***LIKE TO PLAY CRICKET*** are most probably younger .
- people who ***LIKE CHINESE FOOD*** is just random noise .

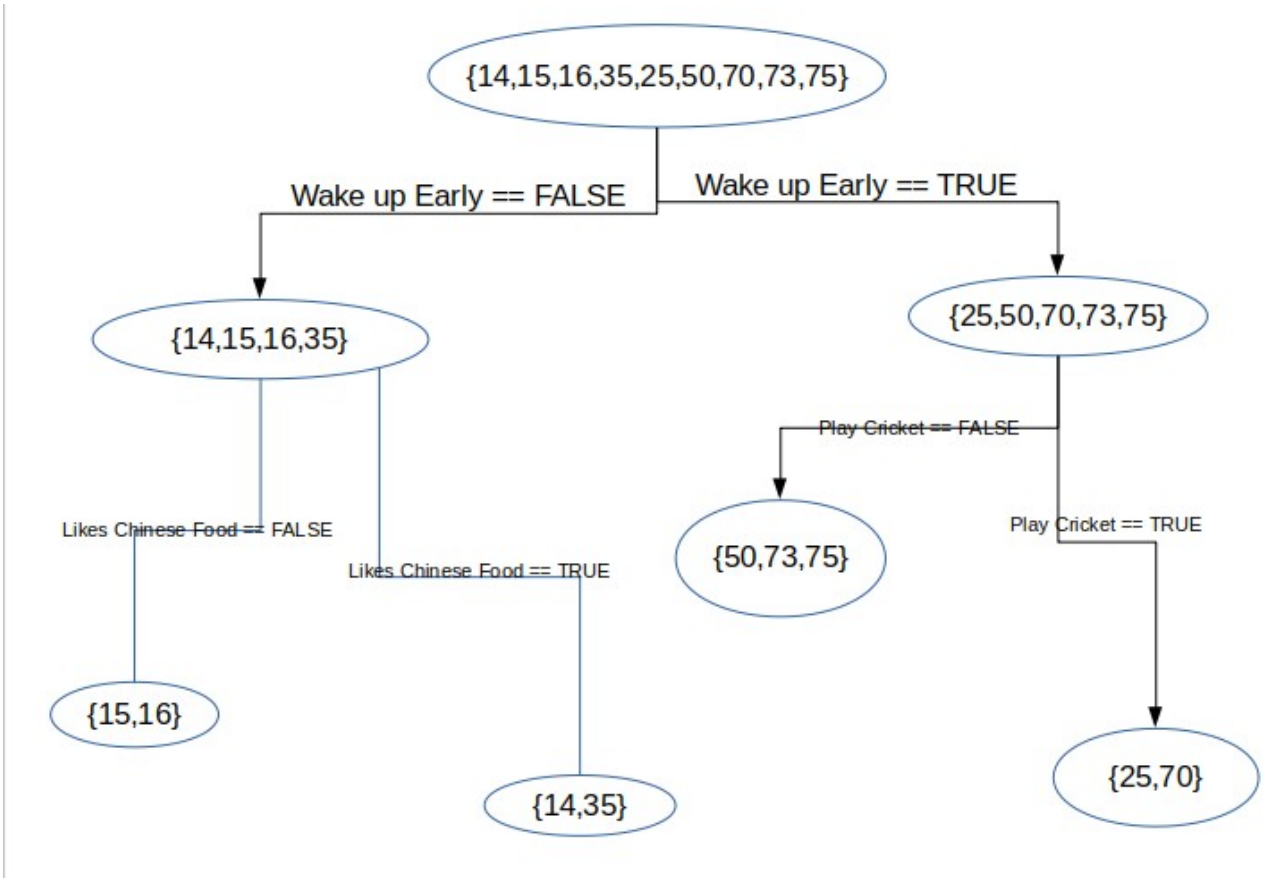
we are going to do some inspection on the data

| FEATURE | FALSE | TRUE |
|---|---|---|
| LIKES TO WAKE UP EARLY | 14-15-16-35 | 25-50-70-73-75 |
| LIKES CRICKET GAMES | 50-73-75 | 14-15-16-25-35-70 |
| LIKES CHINESE FOOD | 15-16-50-73 | 14-25-35-70-75 |

Now first of all we will try to use a regression tree . We will use the nodes that will have atleast three samples . So the tree will make first split on ***LIKE TO WAKE UP EARLY*** .



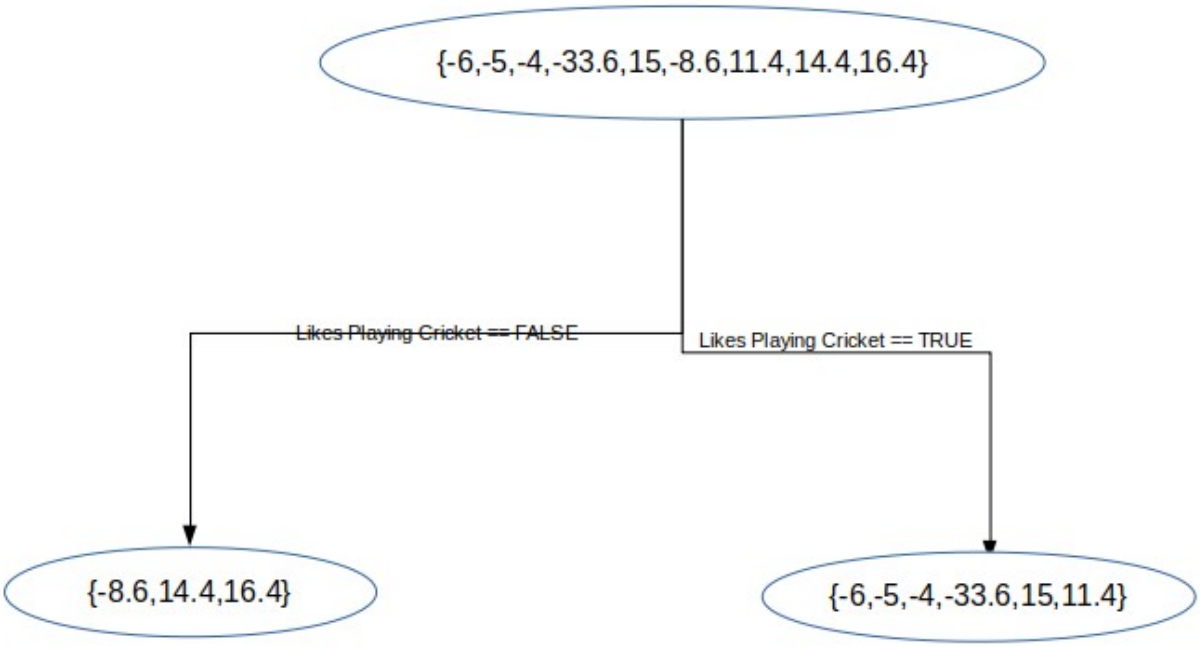

This tree is nice but it's also missing some valuable information like ***LIKES TO PLAY CRICKET*** Now let's try to use the data that have two terminal nodes so we will use the other features too.




From above tree we can see that we picked up some information from ***LIKE CRIKCET*** or not and we also pick up some information from whether ***LIKE CHINESE FOOD*** or not and it is indicating that we are overfitting and our tree is splitting noise. This shows the drawback of the single decision regression tree . It fails to include predictive power from

multiple, overlapping regions of the feature space. Now suppose that if we measure the training errors from our first tree .
Here training errors = sum of all ages of a feature / total person in that feature

| PersonID | Age | Tree 1 Prediction | Tree 1 Residuals |
|---|---|---|---|
| 101 | 14 | 20 | -6 |
| 102 | 15 | 20 | -5 |
| 103 | 16 | 20 | -4 |
| 104 | 25 | 58.6 | -33.6 |
| 105 | 35 | 20 | 15 |
| 106 | 50 | 58.6 | -8.6 |
| 107 | 70 | 58.6 | 11.4 |
| 108 | 73 | 58.6 | 14.4 |
| 109 | 75 | 58.6 | 16.4 |




| PersonID | Age | Tree 1 Prediction | Tree 1 Residuals | Tree 2 Prediction | Combined Prediction | Final Residuals |
|---|---|---|---|---|---|---|
| 101 | 14 | 20 | -6 | -3.71 | 16.29 | 2.29 |
| 102 | 15 | 20 | -5 | -3.71 | 16.29 | 1.29 |
| 103 | 16 | 20 | -4 | -3.71 | 16.29 | 0.29 |
| 104 | 25 | 58.6 | -33.6 | -3.71 | 54.89 | 29.89 |
| 105 | 35 | 20 | 15 | -3.71 | 16.29 | −18.71 |
| 106 | 50 | 58.6 | -8.6 | 7.4 | 66 | 16 |
| 107 | 70 | 58.6 | 11.4 | -3.71 | 54.89 | -15.11 |
| 108 | 73 | 58.6 | 14.4 | 7.4 | 66 | -7 |
| 109 | 75 | 58.6 | 16.4 | 7.4 | 66 | -9 |

***Generalizing the above discussed model***

What we did is summarised by the below equations

1. Fit a model to the data, F1(X)=Y
2. Fit a model to the residuals, h1(X) = Y - F1(X)
3. Create a new model, F2(X) = F1(X) + h1(X) [Note: F2 is boosted version of F1 because we used information from F1 ]

and goes on… Fm(X) = F(m-1)(X) + h(m-1)(X)

Here **hm** is just a model and we never tried to say that it will be a tree based model . Gradient Boosting is really just a framework for iteratively improving any weak learner

So in theory, a well coded gradient boosting module would allow you to "plug in" various classes of weak learners at your disposal. In practice however, $h_m$ is almost always a tree based learner, so for now it's fine to interpret $h_m$ as a regression tree like the one in our example.

***Moving towards XGBoost:***

*Up until now we've been building a model that minimizes squared error, but what if we wanted to minimize absolute error? We could alter our base model (regression tree) to minimize absolute error, but this has a couple drawbacks..*

1. Depending on the size of the data this could be very computationally expensive. (Each considered split would need to search for a median.)
2. It ruins our "plug-in" system. We'd only be able to plug in weak learns that support the objective function(s) we want to use.

Now suppose if we are able to make the each prediction one unit closer to its target. So a regression tree, which by default minimizes squared error, will focus heavily on reducing the residual of the first training sample. But if we want to minimize absolute error, moving each prediction one unit closer to the target produces an equal reduction in the cost function.

Let's try something different now. Instead of fitting h0(x) on the residuals, we will fit h0(X) on the gradient of the loss function L(Y, F0(X)) with respect to the prediction values produced by F0(X). Essentially, we'll train h0 on the cost reduction for each sample if the predicted value were to become one unit closer to the observed value. In the case of absolute error, hm will simply consider the sign of every Fm residual (as apposed to squared error which would consider the magnitude of every residual). After samples in hm are grouped into leaves, **an average gradient** can be calculated and then scaled by some factor, $\gamma$, so that

$$F_m + \gamma h_m$$

minimizes the loss function for the samples in each leaf. (Note that in practice, a different factor is chosen for each leaf.)

For each node, there is a factor γ with which hm(x) is multiplied. This accounts for the difference in the impact of each branch of the split. Gradient boosting helps in predicting the optimal gradient for the additive model, unlike classical gradient descent techniques which reduce error in the output at each iteration.

***Leveraging Gradient Descent.***

The objective function we want to minimize is L. Our starting point is F0(x). For iteration m=1, we compute the gradient of L with respect to F0(X). Then we fit a weak learner to the gradient components. In the case of a regression tree, leaf nodes produce an **average gradient** among samples with similar features. For each leaf, we step in the direction of the average gradient (using a line search to determine the step magnitude). The result is F1. Then we can repeat the process until we have Fm.

***Summary of XGBoost:***

In XGBoost we fit a model on the gradient of loss generated from the previous iteration or from previous step.

In XGBoost, we fit a model on the gradient of loss generated from the previous step. In XGBoost as long as we have a differential loss function we are good to go.

Sources:


1. XGBoost: A Scalable Tree Boosting System by Tianqi Chen, Carlos Giestrin
2. Blogs: Kaggle
3. Medium Blogs
4. Blogs: Data Science Websites