XGBoost: An intro

When it comes to a superfast machine learning algorithm that works on tree-based models and tries to reach the best in class accuracy by optimally using computational resources, XGBoost or Extreme Gradient Boosting becomes the most natural choice. Created by Tianqi Chen,

XGBoost can be used with multiple languages and multiple paltforms.

It supports languages python, R, Java, C++

It helps to improve speed and performance, because of regularization factors(lambda,eta,gamma), use of GPU, and Cache memory.

The XGBoost algorithm has recently got so much popularity owing to its massive usage in most of the hackathons and Kaggle competitions.

In simple terms, XGBoost may be formally defined as a decision tree-based ensemble learning framework that uses Gradient Descent as the underlying objective function and comes with a lot of flexibility while delivering the desired results by optimally using computational power.

XGBoost falls under the boosting suite of algorithms which in turn is part of the ensemble learning method. Ensemble learning means combining multiple models to make predictions work better rather than the outcome from an individual model.

XGBoost and its unique features

XGBoost is a scalable and highly accurate implementation of gradient boosting that pushes the limits of computing power for boosted tree algorithms.

Some unique features of XGBoost:

- **Regularization:** XGBoost models are extremely complex and use different types of regularization like Lasso and Ridge etc to penalize the highly complex models
- **Capability to handle sparse data:** XGboost is capable of handling sparse data and hence missing value treatments are not necessary

- **Block structures and parallel processing:** Unlike many other machine learning algorithms, XGBoost can concurrently use multiple cores of the CPU at the same time owing to its block structure in the system design. Because of this capability, XGBoost can work exceptionally faster and can converge well.
- **Cache awareness and out-of-core computing:** XGBoost has been designed keeping in mind the optimal use of hardware as well. Owing to this property, the algorithm works by allocating internal buffer memories at each step and hence uses the cache in the most efficient way. To add to this, the algorithm, while handling very large datasets in typical big data problems can compress the large data into small versions thus optimizing the disk space and computational speed. This property is termed as '*out of core*' computation.
- **Built-in cross-validation:** XGBoots algorithm by its design has the ability to cross-validate models while developing. This reduces the chance of overfitting to a great extent and thus helps in maintaining the bias-variance trade-off.
- **Tree pruning:** XGBoost makes splits up to the ***max_depth*** specified and then starts pruning the tree backward and removing splits beyond which there is no positive gain. This process of backward tree pruning stops XGBoost from being a greedy algorithm and doesn't result in overfit model.

XGBoost Parameters

A complex machine learning algorithm like XGBoost comes along with a lot of parameters and so the scopes of parameter tuning are also high.

There are broadly three different kinds of parameters.

- **General Parameters:** For overall functioning like the type of model (classification/regression), displaying of the error message, and so on.
- **Booster parameters:** These are the main sets of parameters that guide individual trees at every step. Some of these booster parameters are listed below:
    - Eta: Learning rate
    - Max_depth: The maximum depth of the component decision tree
    - Max_leaf_nodes: The maximum number of terminal nodes in the decision tree
    - Subsample: fraction of observation which is to be selected for random sampling of each tree.
    - colsample_bytree: Kind of the maximum number of features. Denotes the fraction of columns to be random samples for each tree.

- Lamda-→ helps for regularizing similarity score
- Gamma-→ puts limit on depth of tree based on gain

- **Learning task parameters:** As the name indicates, these parameters define the optimization objective and the metric (RMSE, MAE, LogLoss, Error, AUC, etc) to be calculated at every step.

XGBoost

XGBoost Is An Optimized Enhancement Of Gradient Boosting. In Boosting, Weights Are Added To The Model Based On The Residuals. However, In Gradient Boosted The Loss Function Is Optimized To Correct Errors Made By Previous Models.

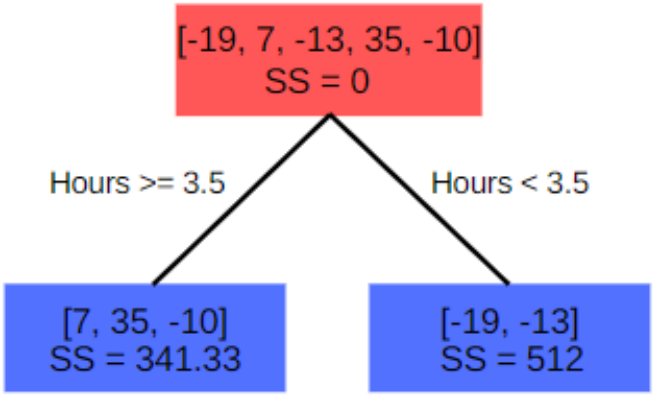
XGBoost Introduces New Features To Gradient Boosting Like Regularization, Tree Pruning, And Parallel Processing. To Clearly Understand The XGboost Algorithm Let's Take A Look At XGboost In Regression As Well As Classification, With Examples.

1. REGRESSION

For Regression, Let's Try To Predict The Marks Of Students Based On The Number Of Hours Studied. Initially, Let Us Consider The Average Marks To Find The Residuals. The Average Of The Five Marks Is 40, Which Will Be The Prediction Of The Base Model. Consequently, The Residuals For The First Five Readings Are -19, 7, -13, 35 And -10, Respectively.

Hours Studied	Marks Obtained
2.5	21
5.1	47
3.2	27
8.5	75
3.5	30

We Use The Residuals To Construct The Decision Tree With Splitting Criteria Say Hours Studied Greater Than 3.3, Then Calculate The Similarity Scores For The Root And Leaf Nodes. Λ In The Equation Is The Regularization Parameter.



$$\text{Similarity Score} = \frac{\left(\sum \text{Residuals}\right)^2}{N + \lambda}$$

N = No. of Residuals

λ = Regularization Parameter

To Begin With, Let Λ=0. The Similarity Score For The Root Is Calculate To Be 0. For The Two Nodes, The Similarity Scores Are 341.33 And 512.

We Then Calculate The Gain Which In This Case Is 853.33.

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

After The Gain Is Calculate, The Auto Tree Pruning Is Complete. For This Purpose, We Use The Gamma Parameter In XGboost Regression. If The Gain Is Less Than The Gamma Value Then The Branch Is Cut And No Further Splitting Takes Place Else Splitting Continues. If The Value Of Gamma Is More, More Pruning Takes Place. The Learning Rate In XGboost Is In Use To Know The Convergence Of The Model. For This Algorithm, It Is Refer To As Eta And The Default Value Is Set To 0.3.

Now We Find The New Prediction For The Data.

$$\text{Output value} = \frac{\sum \text{Residuals}}{N + \lambda}$$

New prediction = Previous Prediction + Learning rate x Output

For First Reading, New Prediction = 40 + (0.3 X -16) = 35.2

The New Residual For This Reading Becomes, 21 – 35.2 = -14

Similarly, We Continue To Do So For All Readings. We Then Use These Values For The Next Model And So On, The Loss Will Keep On Optimizing.

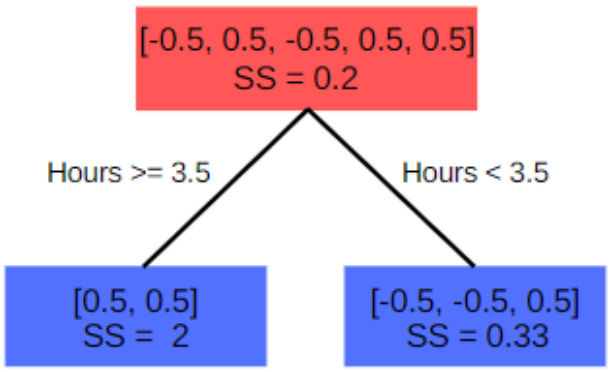
Hours Studied	Marks Obtained	Residuals	New Residuals
2.5	21	-19	-14.0
5.1	47	7	3.8
3.2	27	-13	-8.2
8.5	75	35	31.8
3.5	30	-10	-13.2

2. CLASSIFICATION

For Classification, Let's Try To Predict If Students Pass Or Fail Based On The Number Of Hours Studied. For Binary Classification, The Probability For The Base Model Is 0.5, Meaning There Is A Fifty Percent Chance That A Student Will Either Pass Or Fail. Consequently, The Residuals For The First Five Readings Are -0.5, 0.5, -0.5, 0.5 And 0.5, Respectively.

Hours Studied	Pass/Fail
2.5	0
5.1	1
3.2	0
8.5	1
3.5	1

Similar To Regression, We Construct The Decision Tree For Hours Greater Than 3.5 And Then Calculate The Similarity Scores. The Difference Between The Similarity Score Formula For Regression And Classification Is The Loss Function. λ Remains The Same.



$$\text{Similarity Score} = \frac{\left(\sum \text{Residuals}\right)^2}{\sum^{N}[P\ (1 - P)] + \lambda}$$

P = Probability

With Λ = 0, The Similarity Scores For The Root And Nodes Are 0.2, 0.33 And 2, Respectively. The Gain Is 2.13 Which Is Calculated Similarly As We Calculate For Regression Part.

Pruning In Classification Is Complete Differently Than In Regression. In Classification, We Calculate The Cover Value Of Each Branch. If The Gain Is Greater Than The Cover Value Only Then Further Splitting Is Ready Else The Branch Is Cut. In This Case, Both Branches Can Split. This Is How Pruning Is Done In XGboost Classification.

$$\text{Cover value} = \sum^{N}[P\ (1 - P)]$$

For The New Predictions, We First Find The New Probability Using The Log(Odds) Function. Using The Log(Odds) Formula, The Log(Odds) Value For The Base Model Is 0 (P=0.5).

$$\log(\text{odds}) = \log\left(\frac{P}{1 - P}\right)$$

For The First Reading, We Calculate The Log(Odds) Prediction Value As..

0 + (0.3 X -0.66) = -0.198

Where 0.3 Is The Learning Rate(Eta) Like In XGboost Regression And -0.66 Is The Output Value Is..

$$\text{Output Value} = \frac{\left(\sum \text{Residuals}\right)}{\sum^{N}[P\ (1 - P)] + \lambda}$$

We Then Need To Convert This Log(Odds) Value Into Probability. For This, We Use The Logistic Function.

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

Plugging In Log(Odds) = -0.198 In The Above Formula, We Get A Probability Of ≈0.45.

The New Residual For This Reading Becomes -0.45. Just Like In Regression We Continue The Process To Optimize The Loss Until The Residuals Become Very Small Or Have Reached The Maximum Number.

Hours Studied	Pass/Fail	Residuals	New Residuals
2.5	0	-0.5	-0.45
5.1	1	0.5	0.35
3.2	0	-0.5	-0.45
8.5	1	0.5	0.35
3.5	1	0.5	0.35

For Both XGboost Classification And Regression, As The Value Of Λ Increases, More Pruning Takes Place Due To A Decrease In The Similarity Score And Smaller Output Values. An Increase In The Value Of The Regularization Parameter Reduces The Sensitivity Of The Model Towards Individual Observations. As A Result, The Regularization Parameter Takes Care Of Outliers To An Extent And Reduces Overfitting.