**XGBoost**

XGBoost (extreme Gradient Boosting) is an advanced implementation of the gradient boosting algorithm. XGBoost has proved to be a highly effective ML algorithm, extensively used in machine learning competitions and hackathons. XGBoost has high predictive power and is almost 10 times faster than the other gradient boosting techniques. It also includes a variety of regularization which reduces overfitting and improves overall performance. Hence it is also known as ‘**regularized boosting**‘ technique.

Let us see how XGBoost is comparatively better than other techniques:

1. **Regularization:**
   * Standard GBM implementation has no regularisation like XGBoost.
   * Thus XGBoost also helps to reduce overfitting.
2. **Parallel Processing:**
   * XGBoost implements parallel processing and is faster than GBM .
   * XGBoost also supports implementation on Hadoop.
3. **High Flexibility:**
   * XGBoost allows users to define custom optimization objectives and evaluation criteria adding a whole new dimension to the model.
4. **Handling Missing Values:**
   * XGBoost has an in-built routine to handle missing values.
5. **Tree Pruning:**
   * XGBoost makes splits up to the max\_depth specified and then starts pruning the tree backwards and removes splits beyond which there is no positive gain.
6. **Built-in Cross-Validation:**
   * XGBoost allows a user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.

**Parameters**

* **nthread** 
  + This is used for parallel processing and the number of cores in the system should be entered..
  + If you wish to run on all cores, do not input this value. The algorithm will detect it automatically.
* **eta** 
  + Analogous to learning rate in GBM.
  + Makes the model more robust by shrinking the weights on each step.
* **min\_child\_weight** 
  + Defines the minimum sum of weights of all observations required in a child.
  + Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
* **max\_depth**
  + It is used to define the maximum depth.
  + Higher depth will allow the model to learn relations very specific to a particular sample.
* **max\_leaf\_nodes**
  + The maximum number of terminal nodes or leaves in a tree.
  + Can be defined in place of max\_depth. Since binary trees are created, a depth of ‘n’ would produce a maximum of 2^n leaves.
  + If this is defined, GBM will ignore max\_depth.
* **gamma** 
  + A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.
  + Makes the algorithm conservative. The values can vary depending on the loss function and should be tuned.
* **subsample**
  + Same as the subsample of GBM. Denotes the fraction of observations to be randomly sampled for each tree.
  + Lower values make the algorithm more conservative and prevent overfitting but values that are too small might lead to under-fitting.
* **colsample\_bytree** 
  + It is similar to max\_features in GBM.
  + Denotes the fraction of columns to be randomly sampled for each tree