Contents

[Tree-Specific Parameters 1](#_Toc521331095)

[Boosting Parameters 2](#_Toc521331096)

[Miscellaneous Parameters 2](#_Toc521331097)

The overall parameters can be divided into 3 categories:

1. **Tree-Specific Parameters:** These affect each individual tree in the model.
2. **Boosting Parameters:** These affect the boosting operation in the model.
3. **Miscellaneous Parameters:** Other parameters for overall functioning

# Tree-Specific Parameters

The parameters used for defining a tree are further explained below. Note that I’m using scikit-learn (python) specific terminologies here which might be different in other software packages like R. But the idea remains the same.

1. **min\_samples\_split**
   * Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting.
   * 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.
   * Too high values can lead to under-fitting hence, it should be tuned using CV.
2. **min\_samples\_leaf**
   * Defines the minimum samples (or observations) required in a terminal node or leaf.
   * Used to control over-fitting similar to min\_samples\_split.
   * Generally lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.
3. **min\_weight\_fraction\_leaf**
   * Similar to min\_samples\_leaf but defined as a fraction of the total number of observations instead of an integer.
   * Only one of #2 and #3 should be defined.
4. **max\_depth**
   * The maximum depth of a tree.
   * Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
   * Should be tuned using CV.
5. **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.
6. **max\_features**
   * The number of features to consider while searching for a best split. These will be randomly selected.
   * As a thumb-rule, square root of the total number of features works great but we should check upto 30-40% of the total number of features.
   * Higher values can lead to over-fitting but depends on case to case.

# Boosting Parameters

1. **learning\_rate**
   * This determines the impact of each tree on the final outcome (step 2.4). GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates.
   * Lower values are generally preferred as they make the model robust to the specific characteristics of tree and thus allowing it to generalize well.
   * Lower values would require higher number of trees to model all the relations and will be computationally expensive.
2. **n\_estimators**
   * The number of sequential trees to be modeled (step 2)
   * Though GBM is fairly robust at higher number of trees but it can still overfit at a point. Hence, this should be tuned using CV for a particular learning rate.
3. **subsample**
   * The fraction of observations to be selected for each tree. Selection is done by random sampling.
   * Values slightly less than 1 make the model robust by reducing the variance.
   * Typical values ~0.8 generally work fine but can be fine-tuned further.

# Miscellaneous Parameters

As discussed earlier, there are two types of parameter to be tuned here – tree based and boosting parameters. There are no optimum values for learning rate as low values always work better, given that we train on sufficient number of trees.

Though, GBM is robust enough to not overfit with increasing trees, but a high number for pa particular learning rate can lead to overfitting. But as we reduce the learning rate and increase trees, the computation becomes expensive and would take a long time to run on standard personal computers.

Keeping all this in mind, we can take the following approach:

Keeping all this in mind, we can take the following approach:

1. Choose a relatively **high learning rate**. Generally the default value of 0.1 works but somewhere between 0.05 to 0.2 should work for different problems
2. Determine the **optimum number of trees for this learning rate**. This should range around 40-70. Remember to choose a value on which your system can work fairly fast. This is because it will be used for testing various scenarios and determining the tree parameters.
3. **Tune tree-specific parameters** for decided learning rate and number of trees. Note that we can choose different parameters to define a tree and I’ll take up an example here.
4. **Lower the learning rate** and increase the estimators proportionally to get more robust models.