Gradient Boosting:

**Class Imbalance:**

* We should use up sampling or down sampling If there is a class imbalance, impacts the entropy and mse calculations

**Metrics:**

**Hyper Parameters:**

Tree Tuning:

**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.

**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.

**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.

**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:**

**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.

**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.

**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.

**Categorical Features with many values:**

* We should handle by applying some techniques

**Interpretability:**

* No interpreatability

**Train and Run time complexities:**

**After Training Runtime and space complexities:**

**Low latency:**

* Yes. Because of Runtime complexities

**Large Data:**

* Good for large data

**Less Data:**

**Large Dimensions:**

* If number of dimension increases, time taken to train DT will increase
* Dimensionality should be less

**One hot encoding:**

* Should avoid one hot encoding if we have a very large levels (Zip Code)

**Column Standardization/Normalization:**

* Not required (As this is a distance based problem)

**Null Values:**

* It treat as a new level. Hence we have to handle the null values

**Co linearity:**

* Decision trees are by nature immune to multi-collinearity. For example, if you have 2 features which are 99% correlated, when deciding upon a split the tree will choose only one of them. Other models such as Logistic regression would use both the features.

**Multi Class classification:**

* It will be used for multi class classification

**Outliers:**

* Outliers will not impact, tree will become unstable

**Advantages:**

* It works fairly well when we have less data
* Can be easily paralized In Python/R
* It will prevent overfitting, It does this by creating random subsets of the features and building smaller (shallow) trees using the subsets and then it combines the subtrees.

**Disadvantages:**

* It might fail when there are rare predictors as this is a boot strap sampling
* It is a black box model

**Assumptions:**

**Classification and Regression:**