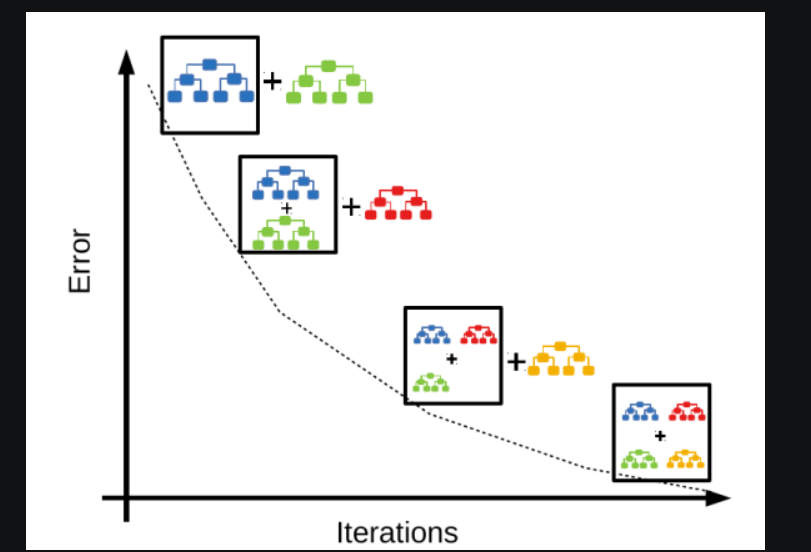
## **Light GBM**

Import concepts: https://lightgbm.readthedocs.io/en/latest/

Light GBM is one of the more novel types of GBDT. It was developed by a team of researchers at Microsoft in 2016.

Gradient Boosting refers to a methodology in machine learning where an ensemble of weak learners is used to improve the model performance in terms of efficiency, accuracy, and interpretability. These learners are defined as having better performance than random chance. Such models are typically decision trees and their outputs are combined for better overall results.



The hypothesis is to filter out instances that are difficult to accurately predict and develop new weak learners to handle them.

The initial model is trained and predictions are run on the whole dataset. The error between the actual value and prediction is calculated and more weightage is given to the incorrect predictions. Subsequently, a new model that attempts to fix the error of the previous model and in a similar way several models are thus created. We arrive at the final model by weighting the mean of all models.

Gradient Boosting can be applied in the following scenarios:

1. Regression – taking the average of the outputs by the weak learners
2. Classification – finding the class prediction occurring the maximum number of times

Some of the most popular boosting algorithms widely used in enterprises and data science competitions are XGBoost, LightGBM, and CatBoost. Several data science practitioners spend a lot of time pondering over which algorithm to go for and often end up doing a lot of hits and trials.

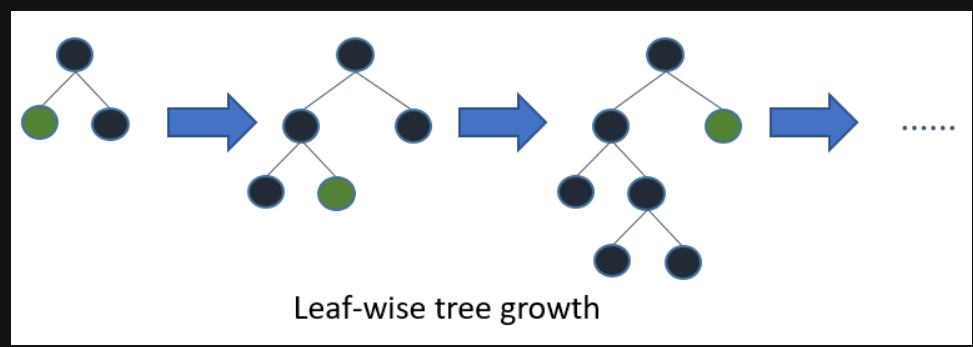
Owing to the ever-rising popularity of XGBoost and LightGBM, we are going to explore both of these from a practical standpoint as well as a little bit of theory to understand their pros and cons a little better.

Similar to XGBoost, [LightGBM](https://lightgbm.readthedocs.io/en/latest/" \t "_blank) (by Microsoft) is a distributed high-performance framework that uses decision trees for ranking, classification, and regression tasks.

The advantages are as follows:

* Faster training speed and accuracy resulting from LightGBM being a histogram-based algorithm that performs bucketing of values (also requires lesser memory)
* Also compatible with large and complex datasets but is much faster during training
* Support for both parallel learning and GPU learning

In contrast to the level-wise (horizontal) growth in XGBoost, LightGBM carries out leaf-wise (vertical) growth that results in more loss reduction and in turn higher accuracy while being faster. But this may also result in overfitting on the training data which could be handled using the max-depth parameter that specifies where the splitting would occur. Hence, XGBoost is capable of building more robust models than LightGBM.



We are going to go over a few fundamental differences between the two algorithms – leaf growth, categorical feature handling, missing values handling, and respective feature importance methods:

### Leaf Growth

**LightGBM**has a faster rate of execution along with being able to maintain good accuracy levels primarily due to the utilization of two novel techniques:

**1. Gradient-Based One-Side Sampling (GOSS):**

* In Gradient Boosted Decision Trees, the data instances have no native weight which is leveraged by GOSS.
* Data instances with larger gradients contribute more towards information gain.
* To maintain the accuracy of the information, GOSS retains instances with larger gradients and performs random sampling on instances with smaller gradients.
* We can learn more about this concept in the article – [What makes LightGBM lightning fast?](https://towardsdatascience.com/what-makes-lightgbm-lightning-fast-a27cf0d9785e)
* The YouTube channel Machine Learning University also released a [video](https://www.youtube.com/watch?v=NsCdkzi2soE&ab_channel=MachineLearningUniversity) on LightGBM speaking about GOSS.

**2. Exclusive Feature Bundling (EFB):**

* EFB is a near lossless method to reduce the number of effective features.
* Just like One-Hot encoded features, in the sparse space, many features rarely take non-zero values simultaneously.
* To reduce dimensionality, improve efficiency, and maintain accuracy, EFB bundles these features, and this bundle is called an Exclusive Feature Bundle.
* [This](https://datascience.stackexchange.com/questions/41907/lightgbm-why-exclusive-feature-bundling-efb) thread on EFB and [LightGBM’s paper](https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf" \t "_blank) can be referred to gain better insight.

### Handling Categorical Features

Both LightGBM and XGBoost accept numerical features only. This means that the nominal features in our data need to be transformed into numerical features.

Let’s assume we want to predict if it’s “Hot”, “Cold” or “Unknown” i.e. 2, 1, or 0. Since they are categorical features and are not ordered (which means that 1 greater than 0 actually is not a correct interpretation), the algorithm needs to avoid comparing the greatness of the two values and focus more on creating rules for equality.

**LightGBM**accepts a parameter to check which column is a categorical column and handles this issue with ease by splitting on equality. However, the H2O library provides an implementation of XGBoost that supports the native handling of categorical features.

### Handling Missing Values

Both the algorithms treat missing values by assigning them to the side that reduces loss the most in each split.

## Feature Importance methods

### Gain:

* Every feature in a dataset has some sort of importance/ weightage in helping build an accurate model.
* Gain refers to the relative contribution of a particular feature in the context of a particular tree.
* This can also be understood by the extent of relevant information that the model gains from a feature for making better predictions.
* Available both in XGBoost and LightGBM.

### Split/Frequency/Weight:

* Split for LightGBM and Frequency or Weight for XGBoost method calculates the relative count of times a particular feature occurs in all splits of the model’s trees. One issue with this method is that it is prone to bias when there are a large number of categories in categorical features.
* Available both in XGBoost and LightGBM.

**Light GBM parameters:**

<https://neptune.ai/blog/lightgbm-parameters-guide>

**Parameter Tuning of LightBGM Model**  
A few important parameters and their usage is listed below :

1. **max\_depth :**It sets a limit on the depth of tree. The default value is 20. It is effective in controlling over fitting.
2. **categorical\_feature :**It specifies the categorical feature used for training model.
3. **bagging\_fraction :**It specifies the fraction of data to be considered for each iteration.
4. **num\_iterations :**It specifies the number of iterations to be performed. The default value is 100.
5. **num\_leaves :**It specifies the number of leaves in a tree. It should be smaller than the square of *max\_depth*.
6. **max\_bin :**It specifies the maximum number of bins to bucket the feature values.
7. **min\_data\_in\_bin :** It specifies the minimum amount of data in one bin.
8. **task :**It specifies the task we wish to perform which is either train or prediction. The default entry is *train*. Another possible value for this parameter is *prediction.*
9. **feature\_fraction**: It specifies the fraction of features to be considered in each iteration. The default value is one.