 CatBoost is the first Russian [machine learning algorithm](https://dataaspirant.com/category/machine-learning-2/) developed to be open source. The algorithm was developed in the year 2017 by machine learning researchers and engineers at Yandex (a technology company).

The term CatBoost is an acronym that stands for "Category” and “[Boosting](https://dataaspirant.com/gradient-boosting-algorithm/).” Does this mean the “Category’ in CatBoost means it only works for categorical features?

The answer is, “No.”

CatBoost has two main features, it works with categorical data (the Cat) and it uses gradient boosting (the Boost). Gradient boosting is a process in which many decision trees are constructed iteratively. Each subsequent tree improves the result of the previous tree, leading to better results. CatBoost improves on the original gradient boost method for a faster implementation.

According to the CatBoost documentation, CatBoost supports numerical, categorical, and text features but has a good handling technique for categorical data.

The CatBoost algorithm has quite a number of [parameters to tune the features](https://dataaspirant.com/hyperparameter-tuning-with-keras-tuner/) in the processing stage.

"Boosting" in CatBoost refers to the [gradient boosting machine learning](https://dataaspirant.com/gradient-boosting-algorithm/). Gradient boosting is a machine learning technique for [regression and classification](https://dataaspirant.com/classification-and-prediction/) problems.

Which produces a prediction model in an ensemble of weak prediction models, typically [decision trees](https://dataaspirant.com/decision-tree-algorithm-python-with-scikit-learn/).

Here we would look at the various features the CatBoost algorithm offers and why it stands out:

**Robust**

CatBoost can improve the performance of the model while [reducing overfitting](https://dataaspirant.com/handle-overfitting-deep-learning-models/) and the time spent on tuning.

CatBoost has several parameters to tune. Still, it reduces the need for extensive [hyper-parameter tuning](https://dataaspirant.com/hyperparameter-tuning-with-keras-tuner/) because the default parameters produce a great result.

Overfitting is a common problem in gradient boosting, especially when the dataset is small or noisy. CatBoost has several features that help reduce overfitting.

One of them is a novel gradient-based regularization technique called ordered boosting, which penalizes complex models that overfit the data. Another feature is the use of per-iteration learning rate, which allows the model to adapt to the complexity of the problem at each iteration.

### Automatic Handling of Missing Values

Missing values are a common problem in real-world datasets. Traditional gradient boosting frameworks require imputing missing values before training the model. CatBoost, however, can handle **missing values automatically.**

During training, it learns the optimal direction to move along the gradient for each missing value, based on the patterns in the data.

### Accuracy

The CatBoost algorithm is a high performance and **greedy novel** gradient boosting implementation.

### Categorical Features Support

The key features of CatBoost is one of the significant reasons why it was selected by many boosting algorithms such as LightGBM,  [XGBoost algorithm](https://dataaspirant.com/xgboost-algorithm/" \t "_blank),etc.

With other machine learning algorithms. After preprocessing and cleaning your data, the data has to be converted into **numerical features** so that the machine can understand and make predictions.

This is same like, for any text related models we convert the text data into to numerical data it is know as [**word embedding techniques**](https://dataaspirant.com/word-embedding-techniques-nlp/).

CatBoost overcomes a limitation of other decision tree-based methods in which, typically, the data must be pre-processed to convert categorical string variables to numerical values, one-hot-encodings, and so on. **This method can directly consume a combination of categorical and non-categorical explanatory variables without preprocessing. It preprocesses as part of the algorithm. CatBoost uses a method called ordered encoding to encode categorical features.** Ordered encoding considers the target statistics from all the rows prior to a data point to calculate a value to replace the categorical feature.

This process of encoding or conversion is time-consuming. **CatBoost supports working with non-numeric factors, and this saves some time plus improves your training results.**

### Faster Training & Predictions

Before the improvement of servers, the maximum number of GPUs per server is 8 GPUs. Some data sets are more extensive than that, but CatBoost uses distributed GPUs.

This feature enables CatBoost to learn faster and make predictions 13-16 times faster than other algorithms.

### Interpretability

CatBoost provides some level of interpretability. It can output feature importance scores, which can help understand which features are most relevant for the prediction.

It also supports visualization of decision trees, which can help understand the structure of the model.

## Is tuning required in CatBoost?

The answer is **not straightforward** because of the type and features of the dataset. The **default** settings of the parameters in CatBoost would do a good job.

CatBoost produces good results without extensive [**hyper-parameter tuning**](https://dataaspirant.com/hyperparameter-tuning-with-keras-tuner/). However, some important parameters can be tuned in CatBoost to get a better result.

These features are easy to tune and are well-explained in the CatBoost documentation. Here are some of the parameters that can be optimized for a better result;

* cat\_ features,
* one\_hot\_max\_size,
* learning\_rate & n\_estimators,
* max\_depth,
* subsample,
* colsample\_bylevel,
* colsample\_bytree,
* colsample\_bynode,
* l2\_leaf\_reg,
* random\_strength.
* Before we dive into the several differences that these algorithms possess, it should be noted that the CatBoost algorithm does not require the conversion of the data set to any specific format. Precisely numerical format, unlike XGBoost and Light GBM.
* The oldest of these three algorithms is the [XGBoost algorithm](https://dataaspirant.com/xgboost-algorithm/" \t "_blank). It was introduced sometime in March 2014 by Tianqi Chen, and the model became famous in 2016.
* Microsoft introduced lightGBM in January 2017. Then Yandex open sources the **CatBoost algorithm later in April 2017.**
* The algorithms differ from one another in implementing the boosted trees algorithm and their technical compatibilities and limitations.
* XGBoost was the first to improve GBM's training time. Followed by LightGBM and CatBoost, each with its techniques mostly related to the splitting mechanism.

### Split

The CatBoost algorithm introduced a unique system called Minimal Variance Sampling (MVS), which is a weighted sampling version of the widely used approach to regularization of boosting models, Stochastic Gradient Boosting.

Also, Minimal Variance Sampling (MVS) is the new default option for subsampling in CatBoost.

With this technique, the number of examples needed for each iteration of boosting decreases, and the quality of the model improves significantly compared to the other gradient boosting models.

The features for each boosting tree are sampled in a way that maximizes the accuracy of split scoring.

### Leaf Growth

Another unique characteristic of CatBoost is that it uses symmetric trees. This means that at every depth level, all the decision nodes use the same split condition.

The CatBoost algorithm grows a balanced tree. In the tree structure, the feature-split pair is performed to choose a leaf.

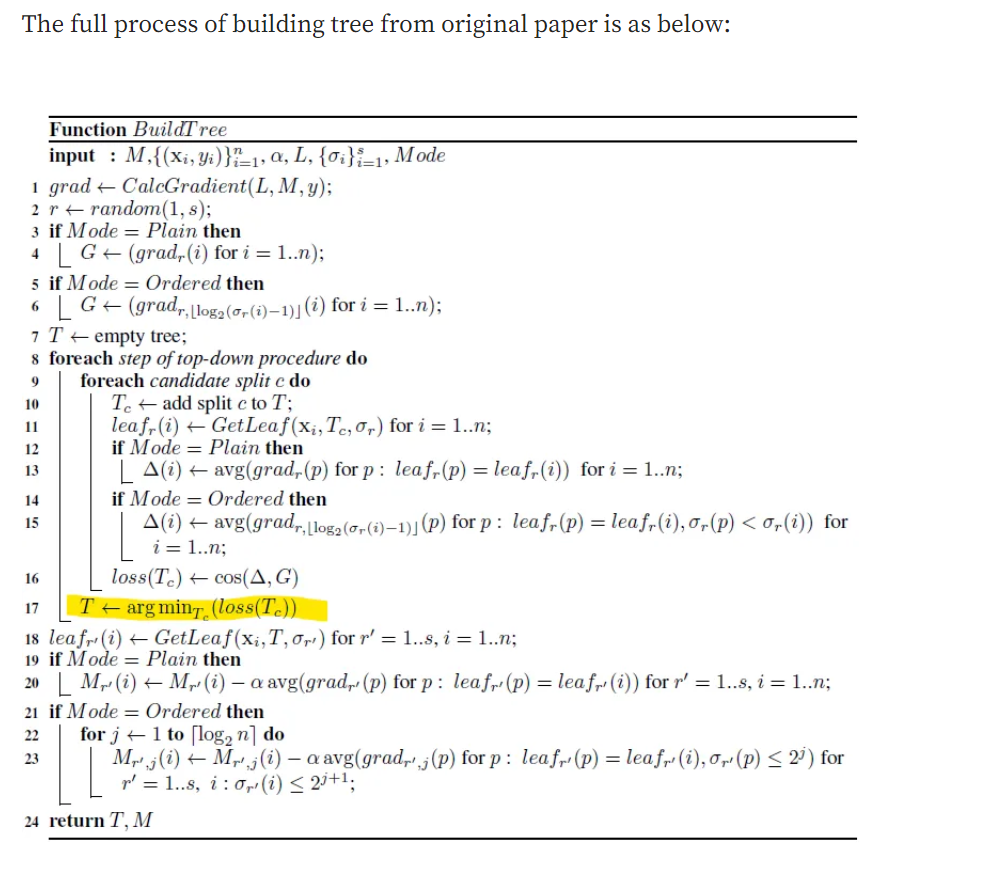
The split with the smallest penalty is selected for all the level's nodes according to the penalty function. This method is repeated level by level until the leaves match the **depth of the tree**.

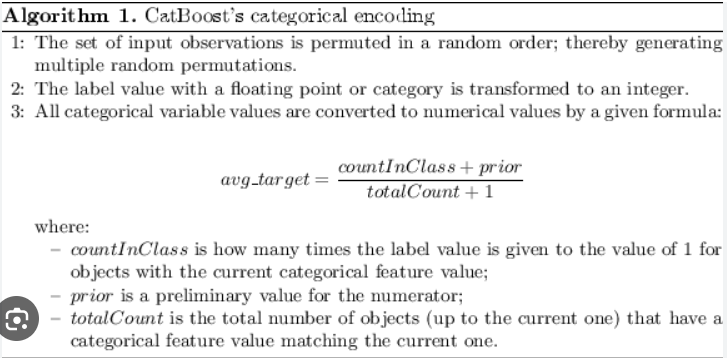
By default, CatBoost uses symmetric trees ten times faster and gives better quality than non-symmetric trees.

* **Missing value support**: CatBoost provides three inherent missing values strategies for processing missing values:
  + “Forbidden”: Missing values are interpreted as an error as they are not supported.
  + “Min”: Missing values are processed as the minimum value(less than all other values) for the feature under observation.
  + “Max”: Missing values are processed as the maximum value(greater than all other values) for the feature under observation. CatBoost only has missing values imputation for numerical values only and the default mode in Min.

The following modes for processing missing values are supported:

* Missing values prohibited; their presence is interpreted as an error.
* Missing values are processed as the minimum value (less than all other values) for the feature. It is guaranteed that a split separates missing values from all other values is considered when selecting trees.
* Missing values are processed as the maximum value (greater than all other values) for the feature. It is guaranteed that a split separates missing values from all other values is considered when selecting trees.





Unlike some other [machine learning algorithms](https://dataaspirant.com/category/machine-learning-2/), CatBoost performs well with a small and large dataset. However, it is advisable to be mindful of [overfitting](https://dataaspirant.com/handle-overfitting-deep-learning-models/). A little tweak to the parameters might be needed here.

There are not many disadvantages of using CatBoost for whatever data set.

**So far, the hassle why many do not consider using CatBoost is because of the slight difficulty in tuning the parameters to optimize the model for categorical features.**

**Training Parameters**

 Let’s look at the common parameters in CatBoost:

* loss\_function alias as objective — Metric used for training. These are regression metrics such as root mean squared error for regression and logloss for classification.
* eval\_metric — Metric used for detecting overfitting.
* iterations — The maximum number of trees to be built, defaults to 1000. It aliases are num\_boost\_round, n\_estimators, and num\_trees.
* learning\_rate alias eta — The learning rate that determines how fast or slow the model will learn. The default is usually 0.03.
* random\_seed alias random\_state — The random seed used for training.
* l2\_leaf\_reg alias reg\_lambda — Coefficient at the L2 regularization term of the cost function. The default is 3.0.
* bootstrap\_type — Determines the sampling method for the weights of the objects, e.g Bayesian, Bernoulli, MVS, and Poisson.
* depth —The depth of the tree.
* grow\_policy — Determines how the greedy search algorithm will be applied. It can be either SymmetricTree, Depthwise, or Lossguide. SymmetricTree is the default. In SymmetricTree, the tree is built level-by-level until the depth is attained. In every step, leaves from the previous tree are split with the same condition. When Depthwise is chosen, a tree is built step-by-step until the specified depth is achieved. On each step, all non-terminal leaves from the last tree level are split. The leaves are split using the condition that leads to the best loss improvement. In Lossguide, the tree is built leaf-by-leaf until the specified number of leaves is attained. On each step, the non-terminal leaf with the best loss improvement is split
* min\_data\_in\_leaf alias min\_child\_samples — This is the minimum number of training samples in a leaf. This parameter is only used with the Lossguide and Depthwise growing policies.
* max\_leaves alias num\_leaves — This parameter is used only with the Lossguide policy and determines the number of leaves in the tree.
* ignored\_features — Indicates the features that should be ignored in the training process.
* nan\_mode — The method for dealing with missing values. The options are Forbidden, Min, and Max. The default is Min. When Forbidden is used, the presence of missing values leads to errors. With Min, the missing values are taken as the minimum values for that feature. In Max, the missing values are treated as the maximum value for the feature.
* leaf\_estimation\_method — The method used to calculate values in leaves. In classification, 10 Newton iterations are used. Regression problems using quantile or MAE loss use one Exact iteration. Multi classification uses one Netwon iteration.
* leaf\_estimation\_backtracking — The type of backtracking to be used during gradient descent. The default is AnyImprovement. AnyImprovement decreases the descent step, up to where the loss function value is smaller than it was in the last iteration. Armijo reduces the descent step until the [Armijo condition](https://en.wikipedia.org/wiki/Wolfe_conditions#Armijo_rule_and_curvature) is met.
* boosting\_type — The boosting scheme. It can be plain for the classic gradient boosting scheme, or ordered, which offers better quality on smaller datasets.
* score\_function — The [score type](https://catboost.ai/docs/concepts/algorithm-score-functions.html) used to select the next split during tree construction. Cosine is the default option. The other available options are L2, NewtonL2, and NewtonCosine.
* early\_stopping\_rounds — When True, sets the overfitting detector type to Iter and stops the training when the optimal metric is achieved.
* classes\_count — The number of classes for multi-classification problems.
* task\_type — Whether you are using a CPU or GPU. CPU is the default.
* devices — The IDs of the GPU devices to be used for training.
* cat\_features — The array with the categorical columns.
* text\_features —Used to declare text columns in classification problems.