Gradient boosting algorithms - their differences and common features

When studying such a field of knowledge as machine learning, you can't do without diving into such a concept as boosting algorithms. They are designed to enhance machine learning (ML). Their very essence can be understood intuitively, without complicated mathematical calculations. The next model should reduce prediction errors when they are mixed with previous models. And in order to reduce errors to a minimum, it is necessary to set targets as accurately as possible. Each prediction makes changes that affect its cumulative error rate. Each successive model must become a step in the right direction. Boosting is an adaptive algorithm, with each step reducing the size of the error from the previous step. In order to know how the error is reduced, we need to understand the concept of a gradient.

A gradient is a measure of how steeply something goes up or down. For example, a gradient might be at the side of a mountain or a road that goes down a hill or up a hill. The notion of a gradient in this concept means that the target results are based on the error of the gradient relative to the forecast. The gradient descent must be negative because we want to reduce the error, not increase it. Gradient descent refers to finding the local minimum or maximum of a function as it moves along the gradient. Many machine learning models, including artificial neural networks, are based on gradient descent methods.

Usually a prediction model is built in the form of a decision tree. It is a tree-like structure, where decisions are used to grow their likely consequences. For example, they may be the outcomes of events or the cost of resources. By taking successive connections of decision trees, we get the basis for the boosting algorithm. Decision trees are added one at a time, leaving existing elements unchanged. More fine-tuning of the algorithm is done by changing the parameters of the decision trees. You can change the following parameters:

- Number of trees.
- The depth of the tree.
- The number of nodes.
- Number of leaves.
- Number of observations before splitting.
- Minimum loss improvement.

Where did the boosting algorithms come from and how did they evolve?

Based on the work of Micheal Kearns, who in 1988 published his work on the possibility of converting weak to strong prediction results, the Adaboost algorithm was created. It became the predecessor of all boosting algorithms. However, Adaboost is based on adjusting the weights of trained instances. Boosting algorithms rely on the same weight of each trained instance and optimize gradient descent.

Imagine a classroom with both strong and weak students. Weak students include those whose predictions are at least slightly better than a random result. Leaving the strong

students alone, we group the weakest students to improve their scores. Each of the weaker students tries to improve the error of the other weaker student, and they work sequentially. This will be the algorithm of gradient boosting.

If the teacher gives his time and attention to the weakest students first, that would be (very simply put) the Adaboost algorithm. This algorithm reduces the weight of strong predictions and increases the weight of weak predictions. The final result is determined by the mean (regression) or majority vote (classification).

So what is gradient descent optimization? It is very simple - an error-reducing tree is added to the model, so that the model begins to follow the gradient. This can be done by parameterizing the tree, and changing the parameters of the tree, leading to a reduction in residual loss.
Learning stops when the loss is reduced to an acceptable level or the data stops improving. The decision tree parameters that can be changed were specified in this article above.

Now you know why boosting algorithms have gained such wide popularity and powerful development. Their varieties include:

- GLMboost.
- GAMboost.
- CoxBoost.
- RankBoost.
- LambdaMART.
- Stochastic GBM.
- GBDT.
- GBRT.
- MART.
- GBM.

Further, these algorithms began to be used to predict both continuous and categorical target variables. Almost since the advent of search engines for the Internet, these algorithms have been used to rank search results. This is how they gained practical application and brought tangible benefits to people.

What does the boosting algorithm look like for a mathematician

From a mathematical point of view, the boosting algorithm looks like this. Suppose we have a function of the following form

$y = f(x)$

It has a set of feature pairs

x

The target variables look like

$\{(x_i, y_i)\}_{i=1,\ldots,n}$

We need to reconstruct the function

$\hat{f}(x)$

With target approximation

$L(y, f)$

To understand which approximation is better, we minimize the loss function, which has the following form

$$y \approx \hat{f}(x),$$
$$\hat{f}(x) = \arg\min_{f(x)} L(y, f(x))$$

We will minimize the function this way

$$y \approx \hat{f}(x),$$
$$\hat{f}(x) = \arg\min_{f(x)} L(y, f(x))$$

In order to do that, the function

$L(y, f)$

must be differentiable.

Our approximation

$\hat{f}(x)$

will lead to an average minimization of the loss function

$\hat{f}(x) = \arg\min_{f(x)} \mathbb{E}_{x,y}[L(y, f(x))]$

We can limit the functions

$f(x)$

which are known to be infinite, we can limit the search space by parametrized functions

$f(x, \theta), \theta \in \mathbb{R}^d$

Thus, we only need to solve analytically the problem of optimizing parameter values for such an equation

$$\hat{f}(x) = f(x, \hat{\theta}),$$
$$\hat{\theta} = \arg\min_{\theta} \mathbb{E}_{x,y}[L(y, f(x, \theta))]$$

By iteratively approaching the parameters of empirical loss function

$L_\theta(\hat{\theta})$

let us denote by the approximation $\hat{\theta}$

for M

iterations in the form of sum

$$\hat{\theta} = \sum_{i=1}^{M} \hat{\theta}_i,$$
$$L_\theta(\hat{\theta}) = \sum_{i=1}^{N} L(y_i, f(x_i, \hat{\theta}))$$

This formula can be called a boosting.

It is just a matter of adding a gradient descent.
The gradient looks like

$\nabla L_\theta(\hat{\theta})$

And it uses interactive estimates

$\hat{\theta}_i$

with a negative sign along the gradient.

Initialize the first approximation

$\hat{\theta}_0$

And select the number of iterations

M

The initial approximation of the parameters will be

$\hat{\theta} = \hat{\theta}_0$

For each iteration

$t = 1, \ldots, M$

Read the gradient of the loss function

$\nabla L_\theta(\hat{\theta})$

At the current approximation

$\hat{\theta}$

Use the obtained gradient to set the current iterative approximation

$\hat{\theta}_t \leftarrow -\nabla L_\theta(\hat{\theta})$

Update approximation of parameters

$\hat{\theta}$

$$\hat{\theta} \leftarrow \hat{\theta} + \hat{\theta}_t = \sum_{i=0}^{t} \hat{\theta}_i$$

Save the final approximation

$$\hat{\theta} = \sum_{i=0}^{M} \hat{\theta}_i$$

Done! Use the resulting function of the form

$\hat{f}(x) = f(x, \hat{\theta})$

At your own risk, you can optimize the derived loss function
by assigning weights.

Nowadays, boosting algorithms are not just algorithms, but an entire methodology for solving ML problems. To make them clearer to you, we recommend this interesting interactive tutorial.

What tools are available for gradient boosting

For gradient boosting algorithms to be applied in practice, software tools are needed. These include:

- **XGboost**. A software library that can be used by developers using common programming languages.
- **LightGBM**. A boosting algorithm that is more focused on the growth and development of decision tree leaves. A subtle tool for experienced developers.
- **Catboosting**. A powerful framework that CERN uses in their work.

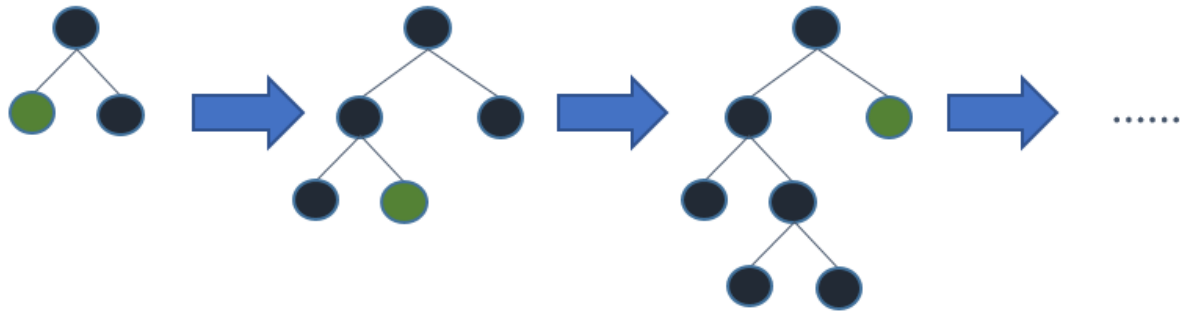
Let's take a brief look at these tools.

XGboost is a software library. It can be used in conjunction with the following programming languages:

- Java.
- Python.
- C++.
- R.
- Julia.

XGboost is open source and can run on Windows, Linux and macOS operating systems. It is a great option for structured or tabular data. It supports many cloud technologies such as AWS and Azure. Hardware optimization is provided to handle large data sizes that don't fit in computer memory. XGboost is often used in competitions by novice machine learning professionals.

LightGBM is an algorithm that primarily focuses on developing leaves on decision trees. Such algorithms converge well, but require fine-tuning.



Leaf-wise tree growth

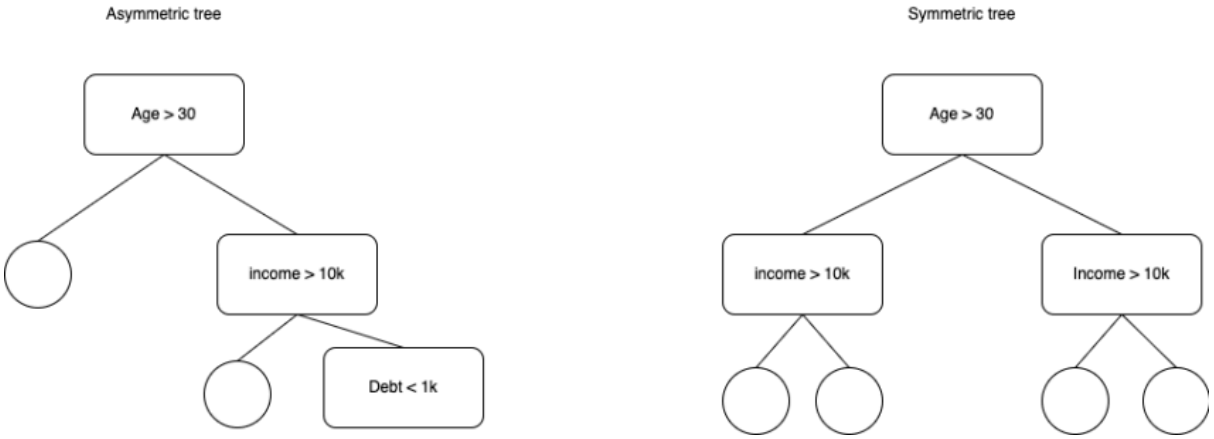
Source:
https://lightgbm.readthedocs.io/en/latest/Features.html?highlight=dart#other-features

LightGBM also refers to open-source software libraries. In practice, it is used in banking, industry, and weather forecasting.

What are the advantages of using the Catboosting framework

Source: http://yandex.com

Catboosting is an algorithm created in 2017 whose name is formed by the words Category and Boosting. In practice, this means that
Catboosting is open source. Catboost uses CERN to interpret results from research at the Large Hadron Collider. The prediction trees here are symmetric, which means that the same condition results in the same leaf splitting everywhere. The algorithm structure is thus made more balanced and requires less computational power.



The difference between symmetric and asymmetric decision trees

Source: https://neptune.ai/blog/when-to-choose-catboost-over-xgboost-or-lightgbm

Also, the Catboost shuffling algorithm can efficiently solve problems using small or heavily noisy datasets. Classical gradient boosting can't do it because the gradient size is too small. Thus Catboost demands minimum time for preliminary data processing. Catboost also works faster than other algorithms both on CPU and GPU due to data paralleling. However all algorithms of gradient boosting have common disadvantages. These are the complexity of data interpretation, frequent overfitting and high demand on computing power.

Conclusion

Each software tool for gradient boosting has its own advantages and limitations in its application.

Catboosting is used by Internet search engines, unmanned cars, virtual assistants and in scientific research.
Xgboost is used by developers who need to incorporate elements of machine learning into their programs. They have a powerful and convenient software library at their disposal.
Without LightGBM you can't do without it in complex computational tasks that require huge computational power.

However, to choose the best program tool it is necessary to know the theoretical basis of such a domain of mathematics as gradient boosting. We hope that this article was useful for you.