Using the XGBoost algorithm

Group K :

- Nelly AGOSSOU
- Jean-Baptiste GOMEZ

Student in Master 2 EBDS and MAG3

supervised by: Pierre Michel - Predictive Methods

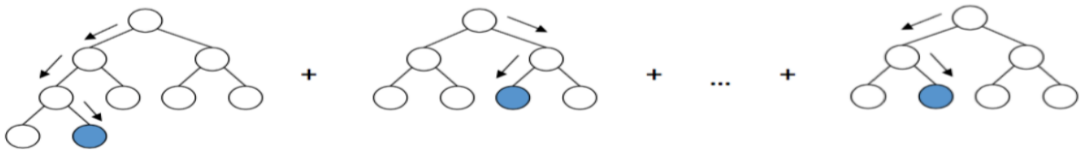
I. Introduction

XGBoost (as in **eXtreme Gradient Boosting**) is an optimized open source implementation of the gradient boosting tree algorithm. This algorithm aims to combine the results of a simpler and weaker set of models to provide a better prediction. First Gradient Boosting is a special case of boosting where errors are minimized by the gradient descent algorithm. To better understand what this is all about, let's take an example. We want to build a boost on decision trees: we will therefore choose a shallow depth so that they are not too complex (a depth of 3 for example). Since the decision tree isn't very deep, two things will happen:

- On a few observations, the predictions $\hat{y}$ are close to the responses y, and thus the residuals for these weak observations.
- Conversely, for others, predictions $\hat{y}$ are far from the responses y, and therefore the residuals for these observations are high.

In practice, therefore, we would like to correct predictions for which residuals are high, and not to touch those for which residuals are low. In other words, we only want to correct for observations that have been poorly predicted. And that's exactly what the next decision tree is going to do: it's going to compensate for the mistakes made previously without deteriorating the predictions that were right. So rather than predicting y, the second tree should predict $y - \hat{y}$, because by summing the previous prediction plus the second tree, we should get the answer y. On the other hand, if the prediction was already correct, the second tree will predict a very small value, which will not change the value already predicted. By repeating the operation several times, n+1st tree will correct the errors of the n-th tree, which explains this recurrence relationship between the models.

Fig: Representation of how the algorithm works



Now let's get back to the XGBoost. The idea is simple: instead of using a single model, the algorithm will use several that will then be combined to obtain a single result. Above all, it is a pragmatic approach that makes it possible to manage regression and classification problems. It allows you to have the same types of results as with the sklearn library but much faster. Consider an initial weak classifier f_0. After optimizing it, the boosting method seeks to build a new weak classifier f_1 from f_0 In the introduction, a residue term h such that: $f_1(x) = f_0(x) + h(x)$, so that $f_1$is more efficient than f_0. Repeating the operation a certain number of times, say p, we construct a complex final classifier *F* which is a linear combination of the f_i where each of the f_i is associated with a weight α_i : $F(x) = \sum_{i=0}^{p} \alpha_i f_i(x)$. So why is this algorithm so popular? In practice, XGBoost has advantages that:

- **Parallelization:** XGBoost relies on parallelism to speed up training, which can reduce training time by a factor of 10 or more.
- **Regularization:** XGBoost uses a technique called regularization to avoid overfitting, which helps improve the generalization of the model.
- **Non-linearity:** XGBoost is based on decision trees, which are able to capture nonlinear relationships between data.
- **Cross-validation:** XGBoost integrates cross-validation, which allows the hyperparameters of the model to be chosen optimally.
- **Missing data:** XGBoost supports missing data, allowing it to be processed without having to eliminate it from the dataset.
- **Flexibility:** XGBoost can be used for regression, classification, and ranking. It is also possible to define a custom objective function for model training.

- **Availability and scalability:** XGBoost is available on many platforms and programming languages. It can also be used to process massive data on clusters.
- **Save & Load:** XGBoost allows you to save and load a template, making it easy to reuse.
- **Tree pruning:** XGBoost allows you to prune decision trees, which can improve the performance of the model.

In summary, this is a powerful and flexible machine learning algorithm, capable of producing high-performance models on complex, nonlinear data. It is available for free and compatible with many languages and platforms.

XGBoost is the algorithm of choice in many situations. It can be used for a wide variety of needs. When it comes to small to medium-sized **structured or tabular data**, the XGBoost algorithm is currently considered the best in its class. Thanks to its sequential self-improvement principle, it can be used to solve **regression, classification, and even ranking problems**. This model should not be chosen to solve problems involving **computer vision, NLP, data extrapolation, or where the number of categories is much greater than the number of observations.** As with all algorithms based on decision trees, care must be taken to avoid overfitting. This is why XGBoost is not the best for very large datasets. This can be corrected with methods to avoid overlearning, such as limiting tree size (max_depth) or building models on samples of the base dataset (called **Stochastic Gradient Boosting**). Although **boosting can increase accuracy**, and although it is the most powerful model in machine learning today, this model sacrifices **intelligibility**. It behaves like a black box, mainly due to its composition of several decision trees.

Here are a few economics problems for which the XGBoost algorithm can be used:

- **Demand forecasting:** XGBoost can be used to predict future demand for products or services based on various economic factors such as revenue, prices, advertising, etc. This is especially useful for inventory management and production planning.
- **Credit Risk Assessment:** In the financial sector, XGBoost can be used to assess the credit risk of borrowers. Using characteristics such as payment history, financial history, and other factors, the model can predict the likelihood of default.
- **Real Estate Pricing:** XGBoost can be applied to predict property prices based on various characteristics such as property size, location, amenities, etc. This can be useful for real estate investors, real estate agents, and
- **Economic forecasting:** XGBoost can be used for economic forecasting by analyzing various indicators such as GDP, inflation, unemployment rate, etc.
- **Marketing Campaign Optimization:** For businesses, XGBoost can be used to optimize marketing campaigns by predicting the likelihood of conversion based on various advertising strategies, past customer behaviors, and more.
- **Fraud detection:** In the financial industry, XGBoost can be applied to detect fraudulent transactions by analyzing behavior patterns and identifying anomalous transactions.
- **Energy Demand Forecasting:** XGBoost can be used to predict energy demand based on various parameters such as weather, economic trends, etc. This can help energy suppliers plan energy production and distribution efficiently.
- **Economic Sentiment Analysis:** XGBoost can be used to analyze economic sentiment from unstructured data such as news articles, social media, etc. This can provide indications of market expectations and reactions.

In our project, we will apply this algorithm to the **Census Income database** from **UCI machine learning website** to predict whether an individual has an annual income greater than $50,000 based on certain social characteristics (gender, age, education, occupation...).

II. Materials and Methods

This section is more presented in our **Jupiter Notebook** with an application of XGBoost Algorithm on the Census Income database. However, we can present here the important parameters and hyperparameters for optimizing the XGBoost algorithm.

Key Parameters of XGBoost:

The authors of XGBoost have categorized the primary parameters into **three (03)** distinct groups:

1- **General Parameters:** Supervise global functionality.
2- **Booster Parameters:** Direct the individual booster (tree/regression) at each step.
3- **Learning Task Parameters:** Govern the optimization process.

General Parameters

Define the supervision of global functionality of XGBoost

1- ***booster [default=gbtree]***

At each iteration, select the type of model to run. It has 2 options:

gbtree: tree-based models

gblinear: linear models

2- ***silent [default=0]***

Silent mode is activated is set to 1, i.e no running messages will be printed. Maintaining it at 0 is usually advisable, as the messages can help in understanding the model.

3- ***nthread [default to maximum number of threads available if not set]***

This parameter is employed for parallel processing, and you should input the number of cores available in the system. If you intend to utilize all available cores, the value should be left unspecified, allowing the algorithm to automatically detect and utilize them.

Booster Parameters

While there are two types of boosters available, we will focus solely on the tree booster as it consistently outperforms the linear booster, making the latter seldom used.

1- ***eta [default=0.3]***

Like the learning rate in Gradient Boosting Machine (GBM), this parameter enhances the model's robustness by reducing the weights at each step. Commonly recommended final values to use are between 0.01 and 0.2.

2- ***min_child_weight [default=1]***

Specifies the minimum sum of weights for all observations needed in a child node. Employed to manage overfitting, higher values restrict the model from capturing relationships that may be excessively specific to the sample chosen for a tree. Excessively high values can result in underfitting, so it's advisable to fine-tune it using cross-validation.

3- ***max_depth [default=6]***

Utilized for controlling overfitting, as a greater depth enables the model to learn relationships specific to a particular sample. This parameter should undergo tuning using cross-validation. Commonly recommended values fall within the range of 3 to 10.

4- ***max_leaf_nodes***

This parameter denotes the maximum number of terminal nodes or leaves in a tree. It can be specified as an alternative to *max_depth.* Given that binary trees are constructed, a depth of 'n' would result in a maximum of $\mathbf{2^n}$ leaves.

5- ***gamma [default=0]***

A node only undergoes splitting if the resulting split yields a positive reduction in the loss function. Gamma defines the minimum loss reduction necessary for a split to occur. This parameter introduces a certain conservatism into the algorithm. The specific values can vary based on the loss function and are subject to tuning.

6- ***max_delta_step [default=0]***

In the maximum delta step, we determine the allowed weight estimation for each tree. A value of 0 implies no constraint, while a positive value promotes a more conservative update step. Typically, this parameter is not required, but in cases of highly imbalanced classes, such as in logistic regression, it may help. While generally not utilized, further exploration is possible if desired.

7- ***subsample [default=1]***

Specifies the fraction of observations to be randomly sampled for each tree. Lower values introduce more conservatism, preventing overfitting, but excessively small values may result in underfitting. Commonly recommended values range between 0.5 and 1.

8- ***lambda [default=1]***

L2 regularization term on weights (analogous to Ridge regression). Employed to manage the regularization aspect of XGBoost. While not widely utilized by many data scientists, it is worth exploring to mitigate overfitting.

9- ***alpha [default=0]***

L1 regularization term on weights, comparable to Lasso regression. It can be advantageous in scenarios with exceptionally high dimensionality, contributing to faster algorithm execution.

10- ***scale_pos_weight [default=1]***

A value greater than 0 should be used in case of high-class imbalance as it helps in faster convergence.

Learning Task Parameters

These parameters are used to define the optimization objective of the metric to be calculated at each step.

1- ***objective [default=reg:linear]***

This defines the loss function to be minimized. Mostly used values are:

binary:logistic – logistic regression for binary classification, returns predicted probability (not class),

multi:softmax –multiclass classification using the softmax objective, returns predicted class (not probabilities),

multi:softprob –same as softmax, but returns predicted probability of each data point belonging to each class.

2- ***eval_metric [default according to objective]***

The metric to be used for validation data.The default values are ***rmse*** for regression and ***error*** for classification. Typical values are:

rmse — root mean square error

mae — mean absolute error

logloss — negative log-likelihood

error — Binary classification error rate (0.5 threshold)

merror — Multiclass classification error rate

mlogloss — Multiclass logloss

auc: Area under the curve.

The algorithmic implementation of XGBoost also requires the specification of several **hyperparameters**, which can be classified into different categories:

- **Hyperparameters associated with numerical computation**, including asynchronous executions.
- **Tree-specific hyperparameters**, such as maximum depth or minimum number of observations in a node.
- **Hyperparameters dedicated to optimization**, such as learning rate or objective function.

It's important to note that not all XGBoost hyperparameters are calibrated simultaneously. This is due not only to the large number of hyperparameters, but also to the fact that learning an XGBoost model takes longer than other Machine Learning models, resulting in considerable computation time. To solve this challenge, Data Scientists have developed a method that is nonetheless effective.

1. Initially, we set a relatively high ***learning_rate***, between **0.1** and **1**. Each new tree is multiplied by this *learning_rate*, and the choice of this value is crucial. A low learning rate favors slow convergence towards the optimum, while too high a rate can hinder convergence, leading to less accurate predictions. In addition, the number of trees is adjusted according to the complexity of the data.

2. The following hyperparameters, which have a significant impact on performance, relate to tree structure: ***max_depth*** (defining the maximum depth of a tree) and *min_child_weight* (establishing a minimum threshold for the number of individuals in a node). A *Gridsearch* on these two hyperparameters is recommended, with a moderate grid size, typically around twenty points.

3. The ***gamma*** coefficient is then introduced to regularize tree depth. A high value of this hyperparameter requires penalization to avoid the construction of trees that are too deep unless this brings significant gains in performance. *Gridsearch* is also applied, generally with values of $\gamma \in [0, +\infty[$.

4. Sampling hyperparameters, such as ***colsample_bytree*** (a random sampling ratio of variables at each tree construction) and subsample (a ratio between 0 and 1 randomly selecting observations at each iteration to train a new tree), can also be optimized by grid search. Note that it is often not necessary to test values below 50%.

5. Finally, for smaller datasets, we recommend reducing the ***learning_rate*** to values such as 0.01 and then 0.001, while increasing the number of trees (*n_estimators*) with the hyperparameters optimized in the previous steps.

The use of an early stopping technique may be relevant for determining the optimal number of trees. However, in the case of large datasets, such as ours with 500,000 rows, this approach may not be practical.

III. Result

In this section we present a statistical description of the data, as well as the results of algorithm in terms of prediction performance. **(cf. Notebook)**

IV. Conclusion

XGBoost is an extremely efficient and versatile machine learning algorithm that has gained considerable popularity over the years. It uses the gradient boosting technique to combine multiple weak decision trees to create a robust and accurate final model. Among its key advantages, XGBoost stands out for its ability to efficiently process large and complex datasets, handle overfitting problems, and adapt to various machine learning tasks, ranging from classification to regression.

However, XGBoost is not without limitations. Its training cost can be high for very large datasets, and it can sometimes be difficult to interpret its predictions due to its non-linear nature. Additionally, optimizing XGBoost's hyperparameters can be a tedious process requiring extensive expertise. Compared to other decision tree-based methods studied in class, XGBoost stands out for its overall performance, but the best method may depend on the specifics of the problem and the characteristics of the dataset. Random Forest, for example, offers a solid alternative, while CART (Classification and Regression Trees) may be preferable in particular contexts.

It is a solid choice for machine learning applications where efficiency, accuracy, and robustness are critical. Its ability to handle big data and its ability to adapt to different types of tasks make it a valuable tool for machine learning specialists. However, it is important to keep in mind its potential limitations in terms of complexity and parameterization, and to use it with caution in scenarios where linearity of predictions is crucial.