XGBoost and AdaBoost

Deependra Singh & Saksham Agarwal

National Institute of Science Education and Research (NISER)
Bhubaneswar

April 21, 2023

Part I: XGBoost

Part II: AdaBoost

Part I

XGBoost

Introduction

XGBoost (eXtreme Gradient Boosting) is a powerful and widely used machine learning algorithm introduced by Tianqi Chen in 2014, that is designed for regression and classification problems. It is an implementation of the Gradient Boosting algorithm that has gained popularity due to its high accuracy and speed, especially in the field of data science and machine learning. In these notes we will dive deep into the working of XGBoost, its advantages, and how it differs from other popular machine learning algorithms.

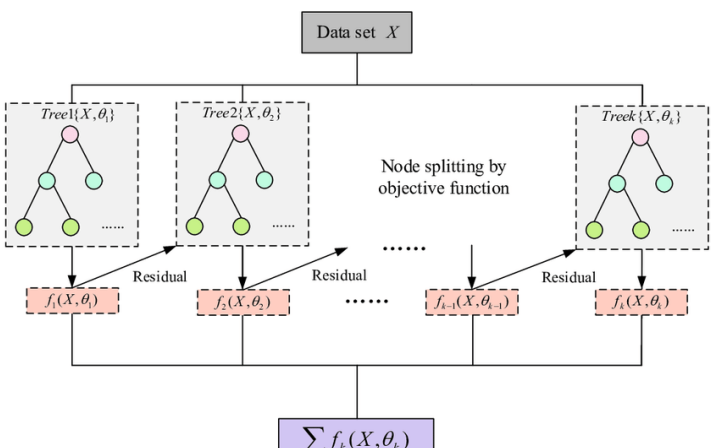

Figure. XGBoost model schematics [2]

Introduction

What is XGBoost?

- XGBoost is a decision tree-based ensemble algorithm that leverages the concept of gradient boosting to create a strong model.
- In essence, XGBoost works by creating a set of decision trees iteratively, each tree attempting to correct the mistakes of the previous tree. The algorithm employs a gradient descent algorithm to minimize a cost function, which is the sum of the errors of each tree in the ensemble.
- The final model is a weighted combination of all the decision trees, with each tree assigned a weight based on its contribution to the cost function.

- **Step 1: Initialize the model**
  The first step in XGBoost is to initialize the model by creating a base prediction. This prediction is usually the mean of the target variable for regression problems or the most common class for classification problems.
- **Step 2: Fit the first tree**
  To train the first tree of a model, features and residuals are used in a greedy manner where informative features are selected first. The residuals are the differences between the model's predictions and the actual values.
- **Step 3: Compute the loss**
  After training the first tree, the loss is computed by summing the errors made by all trees in the ensemble. The sample weights are then updated using the loss, with higher weights assigned to samples with higher errors.
- **Step 4: Fit the next tree**
  After computing the loss, the next step is to fit the next tree. This tree is trained on the updated weights and the residuals from the previous tree. The process is repeated until a predetermined number of trees is reached or the loss stops decreasing.
- **Step 5: Make predictions**
  The final step in XGBoost is to make predictions using the ensemble of trees. The prediction for each sample is the weighted sum of the predictions made by each tree in the ensemble.

XGBoost Algorithm

XGBoost utilizes several techniques to build its algorithm. These include-:

- **Tree Building**
  XGBoost uses a greedy algorithm to build decision trees. It starts with a single node and recursively splits the data into two child nodes based on the feature that maximizes the reduction in the loss function. The loss function is typically the negative log-likelihood for classification problems and the mean squared error for regression problems. The splitting process continues until a stopping criterion is met, such as a maximum depth or a minimum number of samples per leaf node. XGBoost also includes tree pruning to prevent overfitting, which removes nodes that do not contribute to the reduction in the loss function.
- **Gradient Descent**
  Gradient descent is an iterative method to find the minimum of a function by adjusting the parameters towards the negative gradient. In XGBoost, the loss function is the sum of the losses of each data point and a regularization term that penalizes large coefficients. The regularization term, either L1 or L2, makes the model simpler and prevents overfitting. interpretable.

XGBoost Algorithm

- **Gradient Boosting**
  Gradient boosting is a method of combining several weak learners to create a strong learner. In XGBoost, each weak learner is a decision tree that is added to the model iteratively. The decision trees are added one at a time, each correcting the mistakes of the previous trees. During each iteration, XGBoost computes the negative gradient of the loss function with respect to the predicted values. This gradient represents the direction and magnitude of the error, and XGBoost uses it to update the predicted values for the next iteration.
- **Shrinkage**
  Shrinkage is a technique used in XGBoost to prevent overfitting by reducing the contribution of each individual tree to the final model. Instead of adding the full contribution of each tree to the model, XGBoost adds only a fraction of the contribution, called the learning rate or shrinkage rate. The learning rate is a hyperparameter that controls the contribution of each tree. A lower learning rate reduces the contribution of each tree, making the model more robust to overfitting, but increasing the number of iterations required to achieve high acaccuracy.

XGBoost Algorithm

- **Feature Importance**
  XGBoost calculates feature importance by measuring its contribution to the loss reduction. This score is proportional to the number of times it splits the data across all trees. It helps in identifying crucial features and performing feature selection. Additionally, XGBoost has a built-in feature selection method that automatically selects important features and discards less important ones.
- **Early Stopping**
  Early stopping is a technique in XGBoost that prevents overfitting by halting training when the model no longer improves on a validation set. This method prevents the model from memorizing the training data and reduces the required computational resources and time for training.

Advantages of XGBoost

XGBoost has become popular in the field of machine learning due to several advantages it offers over other algorithms. These advantages include:

- **High accuracy:** XGBoost is known for its high accuracy, as it can capture complex relationships between variables in the data. It achieves high accuracy by iteratively creating decision trees, each correcting the mistakes of the previous tree. This results in a model that is able to capture subtle patterns and interactions between features.
- **Scalibility:** XGBoost is highly scalable and can handle large datasets with millions of features and samples. It is designed to work efficiently with multi-core CPUs, making it suitable for distributed computing environments.
- **Flexibility:** XGBoost can handle various types of data, including continuous and categorical variables. It can also handle missing data and can perform feature selection, making it a useful tool for feature engineering.
- **Regularization:** XGBoost includes regularization techniques to prevent overfitting, such as L1 and L2 regularization. These techniques penalize large coefficients, which can lead to a simpler and more interpretable model.
- **Speed:** XGBoost is known for its speed and efficiency, as it can handle large datasets and still train a high-quality model in a relatively short amount of time. It achieves this speed by using a number of optimization techniques, such as parallel processing and histogram-based approximate algorithms

Disadvantages of XGBoost

XGBoost also has some disadvantages that need to be considered-:

- **Computational expensive:** XGBoost can be computationally intensive, particularly when training large models, which can make it unsuitable for resource-constrained systems.
- **Overfitting:** Overfitting is a potential issue with XGBoost, especially when training on small datasets or with too many trees in the model.
- **Hyperparameter Tunings:** Proper hyperparameter tuning is essential for optimizing XGBoost's performance, but it can be time-consuming and requires expertise.
- **Memory requirements:** XGBoost can be memory-intensive, especially when working with large datasets, which may make it unsuitable for systems with limited memory resources.

Part II

AdaBoost

Introduction

What is AdaBoost?

Adaboost is a popular ensemble learning algorithm that was proposed in 1997 by Yoav Freund and Robert Schapire. It combines multiple weak learners to make accurate predictions. Adaboost trains a set of weak learners iteratively on a weighted version of the data. Misclassified data points have their weights increased to focus more on them, helping Adaboost learn a set of weak learners that specialize in different parts of the data and work together for accurate prediction. In these notes, we will discuss Adaboost in detail, including the algorithm's intuition, the training process, and its advantages and disadvantages.

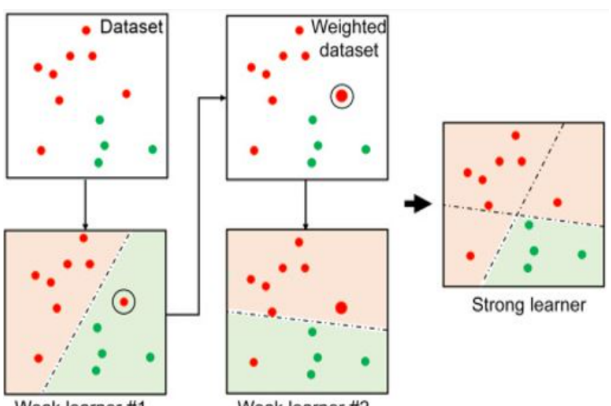

Figure. AdaBoost model schematics [3]

- Adaboost works by combining the predictions of multiple weak learners to make accurate predictions. A weak learner is a model that is only slightly better than random guessing. For example, a weak learner for binary classification might be a decision stump, which is a decision tree with only one split.
- The idea behind Adaboost is to iteratively train a set of weak learners on different parts of the data. In each iteration, the weights of the misclassified data points are increased, which means that the next weak learner will focus more on these points. By doing this, Adaboost is able to iteratively learn a set of weak learners that are specialized in different parts of the data, and that work together to make accurate predictions.
- The final prediction of the Adaboost algorithm is a weighted sum of the predictions of the individual weak learners. The weights of the weak learners are determined by their accuracy on the training data. The more accurate a weak learner is, the more weight it is given in the final prediction.

AdaBoost Algorithm

The training process of Adaboost can be summarized in the following steps:

- Initialize the weights of the training data. In the first iteration, each data point is given equal weight.
- Train a weak learner on the weighted data. The weak learner should have an error rate that is better than random guessing. The error rate is calculated as the sum of the weights of the
- misclassified data points divided by the sum of all the weights.
- Calculate the weight of the weak learner. The weight of the weak learner is calculated as follows:

$$weight = \ln\left(\frac{1 - error_{rate}}{error_{rate}}\right)$$

AdaBoost Algorithm

- The weight of the weak learner reflects its accuracy on the training data. A weak learner that is more accurate will be given a higher weight.
- Update the weights of the training data. The weights of the misclassified data points are increased, while the weights of the correctly classified data points are decreased. This means that the next weak learner will focus more on the misclassified data points.
- Repeat steps 2-4 for a fixed number of iterations or until the error rate on the training data reaches a certain threshold.
- Combine the weak learners into a final model. The final model is a weighted sum of the predictions of the individual weak learners. The weights of the weak learners are determined by their accuracy on the training data.

AdaBoost Algorithm

Mathematical Formulation

- **Input:** Training set of examples (x_1, y_1), (x_2, y_2), ..., (x_n, y_n), where x_i is the i-th training example and y_i is the corresponding label. The number of weak learners T to use.
- Initialize the weights for each sample among n training examples as:

$$w_i = \frac{1}{2}$$

- For $t = 1$ to T: Train a weak learner $h_t(x)$ on the training data with weights w_i.
- Calculate the error of the weak learner:

$$e_t = sum(w_i[h_t(x_i)! = y_i])$$

for $i = 1$ to n, where $[h_t(x_i)! = y_i]$ is the indicator function that is 1 if $h_t(x_i)$ is not equal to y_i and 0 otherwise.

AdaBoost Algorithm

Mathematical Formulation

- Calculate the weight of the weak learner:

$$\alpha_t = \frac{1}{2} \times \ln\left(\frac{1 - e_t}{e_t}\right)$$

- Update the weights of the training examples, such that for $i = 1$ to n, set:

$$w_i = w_i \times exp(-\alpha_t \times y_i \times h_t(x_i))$$

- Normalize the weights so that they sum to 1:

$$w_i = \frac{w_i}{sum(w_i)}$$

- The final strong classifier is:

$$H(x) = \text{sgn}(sum(\alpha_t \times h_t(x)))$$

- **Output:** A strong classifier $H(x)$ that predicts the label y for a given input x.

Advantages of Adaboost

Adaboost has several advantages over other machine learning algorithms-:

- Adaboost is a powerful ensemble learning algorithm that can achieve high accuracy on a wide range of problems.
- Adaboost is robust to overfitting, as the algorithm focuses on the misclassified data points in each iteration.
- Adaboost is easy to implement and requires only a few hyperparameters.
- Adaboost can handle both categorical and numerical data.
- Adaboost can be used with a variety of weak learners, such as decision trees, support vector machines, and neural networks.
- Adaboost can be used for both classification and regression problems.

Disadvantages of Adaboost

Adaboost also has some disadvantages that need to be considered:-:

- Adaboost is sensitive to noisy data and outliers, as these points can be misclassified and have a large impact on the weights in each iteration.
- Adaboost can be computationally expensive, as it requires training multiple weak learners and updating the weights of the data in each iteration.
- Adaboost can suffer from overfitting if the weak learners are too complex or if the number of iterations is too high.

References

1 XGBoost, AdaBoost, Coursera, Andrew Ng.

2 Guo, Rui, et al. "Degradation state recognition of piston pump based on ICEEMDAN and XGBoost." Applied Sciences 10.18 (2020): 6593.

3 XGBoost, geeksforgeeks, https://www.geeksforgeeks.org/xgboost/.

4 AdaBoost, Analytic Vidhya, https://www.analyticsvidhya.com/blog/2022/01/introduction-to-adaboost-for-absolute-beginners/

5 Krish Naik, Youtube Channel.

6 Codebasics, Youtube Channel.