**Machine Learning Algorithms(12) — Ensemble techniques (Boosting — Xgboost Classification)**

[[Kasun Dissanayake](https://kasunprageethdissanayake.medium.com/?source=post_page-----885c06b221e5--------------------------------)](https://kasunprageethdissanayake.medium.com/?source=post_page-----885c06b221e5--------------------------------)

[[Towards Dev](https://towardsdev.com/?source=post_page-----885c06b221e5--------------------------------)](https://towardsdev.com/?source=post_page-----885c06b221e5--------------------------------)

[Kasun Dissanayake](https://kasunprageethdissanayake.medium.com/?source=post_page-----885c06b221e5--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F4a3ca0851dfe&operation=register&redirect=https%3A%2F%2Ftowardsdev.com%2Fmachine-learning-algorithms-12-ensemble-techniques-boosting-xgboost-classification-885c06b221e5&user=Kasun+Dissanayake&userId=4a3ca0851dfe&source=post_page-4a3ca0851dfe----885c06b221e5---------------------post_header-----------)

Published in

[Towards Dev](https://towardsdev.com/?source=post_page-----885c06b221e5--------------------------------)

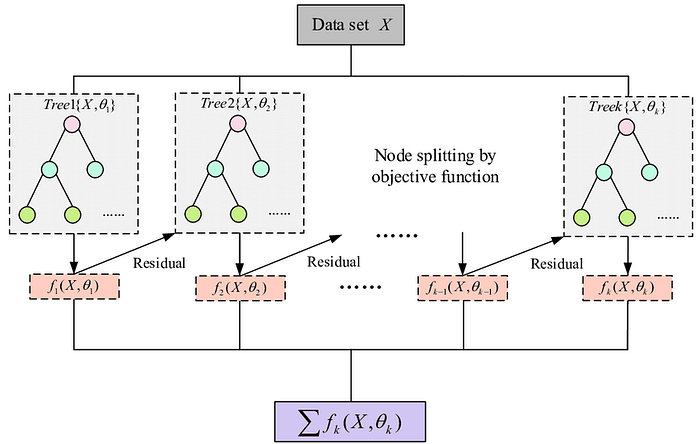
·

10 min read

·

Dec 11

222



This is the 4th article under Ensemble Techniques and if you want to learn more about **Ensemble Techniques** you can refer to my 1st Ensemble Technique article.

**[Machine Learning Algorithms(9) — Ensemble techniques (Bagging —Random Forest Classifier and…](https://towardsdev.com/machine-learning-algorithms-9-ensemble-techniques-bagging-random-forest-classifier-and-5d3747c7a953?source=post_page-----885c06b221e5--------------------------------" \t "_blank)**

[In this article, I am going to explain to you Ensemble techniques and one of the famous Ensemble techniques which…](https://towardsdev.com/machine-learning-algorithms-9-ensemble-techniques-bagging-random-forest-classifier-and-5d3747c7a953?source=post_page-----885c06b221e5--------------------------------" \t "_blank)

[towardsdev.com](https://towardsdev.com/machine-learning-algorithms-9-ensemble-techniques-bagging-random-forest-classifier-and-5d3747c7a953?source=post_page-----885c06b221e5--------------------------------" \t "_blank)

In this article, I am going to explain about **XgBoost Classification Algorithm**. **XgBoost**stands for **Extreme Gradient Boosting**which is a boosting technique that has been designed to optimize distributed gradient boosting. It is an efficient and scalable way to train machine learning models. This learning method combines weak models to produce a stronger prediction, extreme gradient. It is widely used due to its ability to handle large data sets and achieve state-of-the-art performance in machine learning tasks such as classification and regression.

**XGBoost is a more regularized form of Gradient Boosting**. XGBoost uses advanced regularization (L1 & L2), which improves model generalization capabilities. XGBoost delivers high performance as compared to Gradient Boosting. Its training is very fast and can be parallelized across clusters.

XGBoost is usually used with a **tree as the base learner**, that decision tree is composed of a series of binary questions and the final predictions happen at the leaf. XGBoost is itself an ensemble method. The trees are constructed iteratively until a stopping criterion is met.

XGBoost uses **CART(Classification and Regression Trees) Decision trees**. CART is the trees that contain real-valued scores in each leaf, regardless of whether they are used for classification or regression. Real-valued scores can then be converted to categories for classification, if necessary.

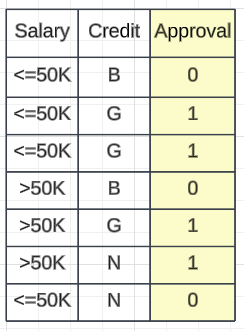
**When to use XGBoost?**

* When there is a larger number of training samples. Ideally, greater than 1000 training samples and less than 100 features or we can say when the number of features < number of training samples.
* When there is a mixture of categorical and numeric features or just numeric features.

**When not to use XGBoost?**

* Image Recognition
* Computer Vision
* When the number of training samples is significantly smaller than the number of features.

Let’s take a simple example to understand it better. The bank gives you credit card **approval** for a loanbased on your **salary**and **credit score**, which are the first, second, and third features. There are 3 categories of credit scores **Bad, Normal, and Good**. Since we are solving a classification problem output either 0 or 1. XgBoost can be used in solving multiclass classification problems also.



Salary-Loan Approval Dataset

**Step 1 —**

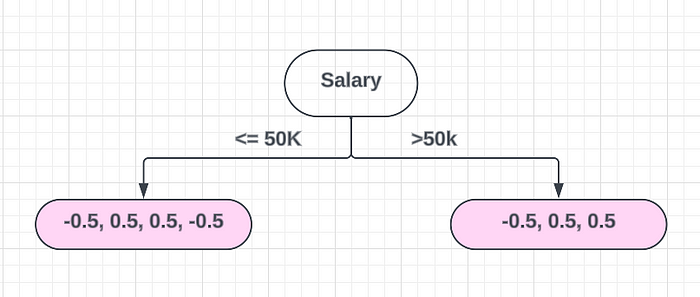
When starting the **XgBoost**classifier, the first step is to **create a specific base model**. This model will always output a probability of 0.5 in the case of a classification problem(output will be either **zero or one**). To calculate the residual, subtract the actual value from the output value of 0.5. For example, if the approval is 0.5, the residual would be **0–0.5 = -0.5**. This base model serves as the foundation for all subsequent decision trees, which must be constructed sequentially. The base model itself is also a type of decision tree, as it takes inputs and provides a default probability of 0.5. Once the base model has been created, the next step is to move on to the first sequential-based decision tree.



Residual Values

**Step 2 —**

Then, we create a binary decision tree using the relevant features. Then I have chosen **Salary**as the first feature and you have 2 categories which are**>50K and ≤50K.** In XGBoost, whenever you create a tree, you need to do a **binary classifier.**This applies even when you have more than two categories.



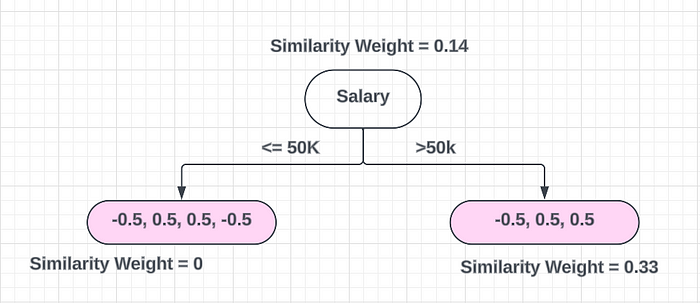
To do this, you need to create a binary classifier and divide it. The leaf nodes will always be two. Then, we calculate the **similarity weight** based on the similarity of the data and find out the Gain.

To come up with these points, we use the values,  
  
≤50K  
-0.5, 0.5, 0.5 and -0.5   
  
>50K  
-0.5, 0.5 and 0.5

**Step 3 —**

Next, we calculate the**similarity weight**, which involves using a formula,

Similarity Weight = Σ(Residuals) ^2 / Σ(probability \* (1 - probability) + λ)  
  
λ= Hyperparameter that prevents overfitting = 0 (For now consider λ value as 0)  
  
probability = This taken from the base model  
  
  
Similarity Weight of the left leaf node   
------------------------------------  
   
Similarity Weight = -0.5 + 0.5 + 0.5 + -0.5 ^ 2/ [0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5)]  
Similarity Weight = 0 / 0.25 + 0.25 + 0.25 + 0.25 + 0.25  
 = 0 / 1.25 = 0  
  
Similarity Weight of the right leaf node  
------------------------------------  
Similarity Weight = -0.5 + 0.5 + 0.5 ^ 2/ [0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5)]  
Similarity Weight = 0.5 ^ 2 / 0.75 = 0.25 / 0.75 = 1/3 = 0.33  
  
  
Similarity Weight of the root node  
------------------------------------  
Similarity Weight = 0.5 ^ 2/ [0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5) ++ 0.5 \* (1 - 0.5) ++ 0.5 \* (1 - 0.5) ++ 0.5 \* (1 - 0.5)]  
Similarity Weight = 0.25 / 1.75 = 1/7 = 0.142

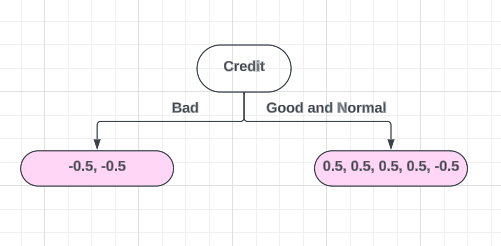


**Step 4—**

Calculate the **Information Gain**. To do this we can add all the leaf node similarity weights together and subtract the root node’s similarity weight from it.

Total Gain with respect to the split = 0 + 0.33 - 0.14 = 0.19

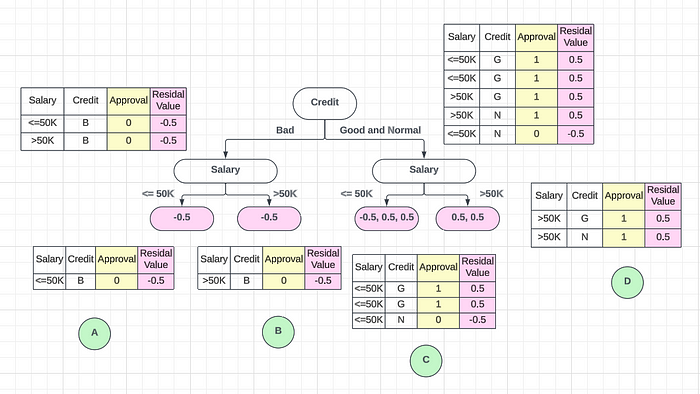
Okay! We have chosen the **Salary** feature for splitting and we got Gain as**0.19**. But we can start doing the splitting from the **Credit**feature also. If we start splitting the Credit feature, you need to do a **binary classifier.**The leaf nodes will always be two. But you have more than two categories(**Bad, Normal, and Good**). To do this, you can do splitting like this,



Step 1  
=======  
Bad  
-0.5, -0.5  
  
Good and Normal  
0.5, 0.5, 0.5, 0.5 and -0.5  
  
Step 2  
==========  
  
Similarity Weight = Σ(Residuals) ^2 / Σ(probability \* (1 - probability))  
  
Similarity Weight of the left leaf node   
------------------------------------  
   
Similarity Weight = -0.5 + -0.5 ^ 2/ [0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5)]  
Similarity Weight = 1 / 0.25 + 0.25   
 = 1 / 0.5 = 2  
  
Similarity Weight of the right leaf node  
------------------------------------  
Similarity Weight = 0.5 + 0.5 + 0.5 + 0.5 - 0.5 ^ 2/ [0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5)]  
Similarity Weight = 2.25 / 1.25 = 1.8  
  
Similarity Weight of the root node  
------------------------------------  
Similarity Weight = 0.142  
  
  
Step 3  
=========  
  
Total Gain with respect to the split = 2 + 1.8 - 0.142 = 3.658

Now you can see if we split from the Credit feature **we can get the highest Gain**. Then you can go with the Credit feature. You can do this for all the combinations and **select the root node feature that gives the highest gain**.

Okay, now I am selecting my first slipt as Credit feature and continue splitting on that feature. I have to do a **binary split** again and I am going to select the Salary feature for the second split and categorize that as ≤50K and >50K. Now you can see how many data points over ≤50K and >50K.



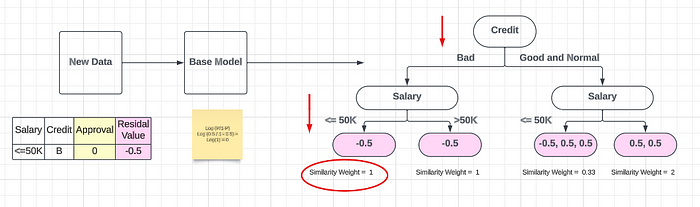
Now we can calculate the similarity weight for the second-level leaf nodes.

Leaf Node A:  
  
Similarity Weight = Σ(Residuals) ^2 / Σ(probability \* (1 - probability))  
 = -0.5^2 / 0.25 = 0.25 / 0.25 = 1  
  
Leaf Node B:  
  
Similarity Weight = Σ(Residuals) ^2 / Σ(probability \* (1 - probability))  
 = -0.5^2 / 0.25 = 0.25 / 0.25 = 1  
  
  
Root Node:  
  
Similarity Weight = -0.5 + -0.5 ^ 2/ [0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5)]  
Similarity Weight = 1 / 0.25 + 0.25   
 = 1 / 0.5 = 2  
  
  
Leaf Node C:  
  
Similarity Weight = Σ(Residuals) ^2 / Σ(probability \* (1 - probability))  
 = (0.5 + 0.5 + -0.5)^2 / (0.25 + 0.25 + 0.25) = 0.25 / 0.25 = 1  
 = 0.25 / 0.75 = 1/3 = 0.33  
  
  
Leaf Node D:  
  
Similarity Weight = Σ(Residuals) ^2 / Σ(probability \* (1 - probability))  
 = (0.5 + 0.5)^2 / (0.25 + 0.25) = 1 / 0.5 = 2  
  
Root Node:   
Similarity Weight = Σ(Residuals) ^2 / Σ(probability \* (1 - probability))  
 = 0.5 + 0.5 + 0.5 + 0.5 - 0.5 ^ 2/ [0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5) + 0.5 \* (1 - 0.5)]  
 = 2.25 / 1.25 = 1.8  
  
  
  
Now we can calculate the Information Gain:  
Information Gain Left Tree = Similarity Weight of A + Similarity Weight of B - Similarity Weight of Root Node   
  
 = 1 + 1 - 2 = 0  
  
Information Gain Left Tree = Similarity Weight of C + Similarity Weight of D - Similarity Weight of Root Node   
  
 = 0.33 + 2 - 1.8 = 0.53

We will be comparing which split is the best based on the **Information Gain**. Now I have created my entire decision tree. Let’s consider the inferencing part. Let’s say a new record is going to go into the model and how we calculate the output. First of all that row will go to the base model. Then the **base model will give a probability of 0.5**. Now how do we calculate the real probability from the base model? For that, we can apply something called **Logs**. We can use a formula to calculate the probability.

P = Base Model Probability  
Log (P/1-P)  
Log (0.5 / 1 - 0.5) = Log(1) = 0

If we take a closer look, this is equal to nothing but zero. This means that the initial value will be zero and go through the binary decision tree.



The resulting value will be added to the branches that fall for Bad Credit limit and Salary which is ≤ 50K. The **similarity weight is 1** and we pass with the learning rate parameter(α).

0 + α (1) α = Learning Rate = 0.001

The learning rate parameter is multiplied by the similarity weight of **1** to obtain the reference value. We use the Alpha value(α) as our learning rate, which can be a minimal value based on the learning parameter we have defined elsewhere. To solve this classification problem, we apply the **activation function called Sigmoid(σ)**. This ensures that the output value falls between zero and one.

σ (0 + α (1))

Similarly, you can create other decision trees as well.

So finally your output for a new record will be like this,

σ (0 + α1 (Dicision Tree Similarty Weight1) + α2 (Dicision Tree Similarty Weight2) + α3 (Dicision Tree Similarty Weight3) + α4 (Dicision Tree Similarty Weight4) + ..... + αn (Dicision Tree Similarty Weightn))

Similarly, the algorithm produces more than one decision tree and combines them additively to generate better estimates.

**Advantages of XGBoost**

1. Performance: XGBoost has a strong track record of producing high-quality results in various machine learning tasks, especially in Kaggle competitions, where it has been a popular choice for winning solutions.
2. Scalability: XGBoost is designed for efficient and scalable training of machine learning models, making it suitable for large datasets.
3. Customizability: XGBoost has a wide range of hyperparameters that can be adjusted to optimize performance, making it highly customizable.
4. Handling of Missing Values: XGBoost has built-in support for handling missing values, making it easy to work with real-world data that often has missing values.
5. Interpretability: Unlike some machine learning algorithms that can be difficult to interpret, XGBoost provides feature importances, allowing for a better understanding of which variables are most important in making predictions.

**Disadvantages of XGBoost**

1. Computational Complexity: XGBoost can be computationally intensive, especially when training large models, making it less suitable for resource-constrained systems.
2. Overfitting: XGBoost can be prone to overfitting, especially when trained on small datasets or when too many trees are used in the model.
3. Hyperparameter Tuning: XGBoost has many hyperparameters that can be adjusted, making it important to properly tune the parameters to optimize performance. However, finding the optimal set of parameters can be time-consuming and requires expertise.
4. Memory Requirements: XGBoost can be memory-intensive, especially when working with large datasets, making it less suitable for systems with limited memory resources.

This is all about XgBoost Classifier. I hope you get a better understanding of this algorithm. See you in another tutorial.

Thank You!