water

MDPI

Article

Data-Driven Flood Alert System (FAS) Using Extreme Gradient Boosting (XGBoost) to Forecast Flood Stages

Will Sanders, Dongfeng Li, Wenzhao Li, and Zheng N. Fang *

Supplementary Materials S1: Introduction of XGBoost Algorithm

1. Introduction to XGBoost Algorithm

The XGBoost is an advanced application of the multiple model ensemble methodology, which includes two major schools: Bagging and Boosting. The Bagging algorithm gets the result by using Bootstrap sampling method to train the base classifier according to the equal weight voting of all base classifiers. The Boosting algorithm iteratively trains the weak classifiers which are weighted and added them to a final strong classifier. After the weak classifiers are added, their weights are readjusted (i.e. re-weighting) to grant a higher weight on the misclassified inputs, to let the following weak learners focus on the misclassified examples and improve the overall performance of the final strong classifier. The XGBoost is a representative Boosting method.

1.1. Determination of Leaf Node Values

Based on the idea of Boosting methodology, XGBoost generates a new tree in the direction of decreasing the residual (negative gradient) from the last training. The sum of all tree scores is taken as the final prediction value. From the mathematics point of the view, the algorithm is to solve a functional optimization problem, which is to achieve a certain generalization ability while controlling the overall error of the predicted values. Firstly, to determine the score of each node, the objective function is set as:

$$L(\phi) = \sum_i L\left(\hat{y}_i, y_i\right) + \sum_k \Omega\left(f_k\right) \tag{1}$$

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \parallel w \parallel^2 \tag{2}$$

Here $L(\phi)$ is the objective function with two components: 1) $\sum_i L\left(\hat{y}_i, y_i\right)$ as the loss function component to measure how well model fit on training data, in which $\hat{y}_i$ is the predicted value; y_i is the label value; f_k is the k-th tree model; 2) $\Omega(f)$ is the regularization component to measure complexity of trees, in which γ is a regularizing term for pruning trees; T is the total number of leaf nodes and $\lambda \parallel w \parallel^2$ is an L2 norm of leaf scores, where λ is a regularizing term to avoid over-fitting and w is the weight of leaf nodes. As a boosting method, the new tree is created to fit the residual of the previous tree. When the t-th tree is created, the predicted value can be presented as:

$$\hat{y}_i^t = \hat{y}_i^{(t-1)} + f_t(x) \tag{3}$$

The objective function with total of n samples can be written as:

$$L^{(t)} = \sum_{i=1}^{n} L\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) \tag{4}$$

Citation: Sanders, W.; Li, D.; Li, W.; and Fang, Z.N. Data-Driven Flood Alert System (FAS) Using Extreme Gradient Boosting (XGBoost) to Forecast Flood Stages. *Water* **2022**, *14*, 747. https://doi.org/10.3390/w14050747

Academic Editor: Zhijia Li and Cheng Yao

Received: 13 January 2022
Accepted: 16 February 2022
Published: 26 February 2022

Hence, the direction of optimization is to find the f_t that can minimize the $L^{(t)}$. The second-order Taylor expansion is presented as:

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2 \tag{5}$$

XGBoost uses the second-order Taylor expansion to approximate $L^{(t)}$ to $\mathcal{L}^{(t)}$:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n}\left[L\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_i(x_i) + \frac{1}{2}h_i f_i^2(x_i)\right] + \Omega(f_t) \tag{6}$$

where g_i and h_i are the first and second derivatives of $L\left(y_i, \hat{y}_i^{(t-1)}\right)$, respectively:

$$g_i = \partial_{\hat{y}_i^{(t-1)}} L\left(y_i, \hat{y}_i^{(t-1)}\right) \quad , \quad h_i = \partial^2_{\hat{y}_i^{(t-1)}} L\left(y_i, \hat{y}_i^{(t-1)}\right) \tag{7}$$

Since the predictions and residuals from previous *t*-1 trees do not have impact on the optimization of the objective function, $\mathcal{L}^{(t)}$ can be simplified as $\hat{\mathcal{L}}^{(t)}$ without $L\left(y_i, \hat{y}_i^{(t-1)}\right)$. In addition, every sample will be allocated into a leap node, thus the samples can be grouped according to their leap nodes positions. Therefore, $\hat{\mathcal{L}}^{(t)}$ can be transformed as:

$$\begin{aligned}
\hat{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n}\left[g_i f_t(x_i) + \frac{1}{2}h_t f_t^2(x_i)\right] + \Omega(f_t) \\
&= \sum_{i=1}^{n}\left[g_i f_t(x_i) + \frac{1}{2}h_t f_t^2(x_i)\right] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \\
&= \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right)w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right)w_j^2\right] + \gamma T \\
&= \frac{1}{2}\sum_{j=1}^{T}(H_j + \lambda)\left(w_j + \frac{G_j}{H_j + \lambda}\right)^2 + \gamma T - \frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda}
\end{aligned} \tag{8}$$

where $G_j = \sum_{i\in I_j} g_i$ represents the sum of the first-order gradients from the samples placed at leaf node i; $H_j = \sum_{i\in I_j} h_i$ represents the sum of the second-order gradients from the samples placed at leaf node i. Let $\partial_{w_j}\hat{\mathcal{L}}^{(t)} = 0$, the optimal weight for *t*-th tree, w_j^* is:

$$w_j^* = -\frac{G_j}{H_j + \lambda} \tag{9}$$

Hence, the final optimal solution of the object function, $\hat{\mathcal{L}}^*$, is:

$$\hat{\mathcal{L}}^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T \tag{10}$$

1.2. Tree Building

The most critical step when building a tree is to choose a criterion for splitting and evaluate the quality of splitting. For instance, GBDT choose the mean squared error (MSE), or mean absolute error (MAE) to evaluate the quality of the splitting, yet such heuristic splitting criterion does not always relate to the loss function. However, in XGBoost, the splitting criterion is directly linked to the loss function and it is one of the major differences between XGBoost and GBDT. Specifically, XGBoost traverses all the nodes and decides whether to split the tree by evaluating the value of the Gain:

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma$$

(11)

where $G_L = \sum_{i \in L} g_i$ or $H_L = \sum_{i \in L} h_i$ represents the sum of the first-order or second-order gradients from the samples placed at left leaf node L if it is split; $G_R = \sum_{i \in R} g_i$ or $H_R = \sum_{i \in R} h_i$ represents the sum of the first-order or second-order gradients from the samples placed at right leaf node R if it is split. $\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}$ represents the loss value if not splitting the node, while $\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda}$ represents the sum of loss values from both left and right leaf nodes after splitting. Therefore, the choice of the splitting node is to find the one that can maximize the Gain value. γ is used to control the complexity of the tree as a threshold value for splitting. Additionally, if there are missing values in the training data, XGBoost will put the missing values into L and R to calculate the scores, and the missing values will be placed into whichever with higher scores. To sum up, the advantages of the algorithm can be summarized as follows:

1. XGBoost algorithm is an integrated algorithm based on decision tree (e.g. CART), which can solve classification and regression problems with fast training speed and high accuracy.
2. XGBoost algorithm makes the second-order Taylor expansion of the objective function and uses the information of the first derivative and the second derivative of the objective function, which makes the estimation of loss values more accurate.
3. XGBoost algorithm optimizes the objective function and prunes it simultaneously, which reduces the chances of over-fitting. The terms γ and T, as well as L2 norm can limit the tree growth to reduce over-fitting.
4. When the training data is sparse, XGBoost algorithm can select the direction for missing values, which greatly improves the efficiency of the algorithm.
5. XGBoost can use parallel computing and distributed training, which improves the training speed of the algorithm. Its parallelism is not tree-granularity but feature granularity parallelism. Since one of the most time-consuming steps of decision tree learning is to sort the values of the feature to decide the best split point. Before training, XGBoost sorts the data in advance, and then saves it as a block structure. This structure is repeatedly used in iterations, which greatly reduces the amount of calculation, also makes parallel possible. When splitting nodes, it is necessary to calculate the gain of each feature to select the largest gain for splitting, then the gain calculation of each feature can also be performed in multiple threads.
6. XGBoost algorithm also is strongly compatible and can run on Windows, MacOS, Linux and other platforms, and supports Python, R, C++, Java, and other languages.