Extreme Gradient Boosting

Gradient Boosting

The general idea behind machine learning models is to construct one model that is able to make predictions. However, boosting algorithms aims to train a sequence of weak learners that as a whole is able to make predictions, also called ensemble learning. Each model is trained to compensate for the weaknesses of the model before. In the relevance of gradient boosting, the weak learners consists of small decision trees, which are sequentially improved upon based on the residual errors of the previous decision tree. The first tree in the sequence is usually leaf which predicts the average of the target values, for regression problems, and after that the boosting algorithm will gradiently reduce the error by adding more trees. Decision trees consists of a certain amount of binary statements, also called splits, these splits is used to either make a prediction or there will be a new split until they reach a prediction, Figure 8. Gradient boosting algorithms utilizes these trees as the weak learners they train sequentially. The sequential connection of these trees means for each tree, that tree is fitted to the residual errors of the previous tree, which it then tries to minimize. In the case of regressional implementations of gradient boosting algorithms, the residual error is calculated by a loss function. Since gradient boosting is a supervised learning algorithm, it would be easy for these trees to fit itself to the training data and thus very quickly reduce the residual errors to zero. In order for this not to happen, each trees contribution is scaled by the learning rate. The learning rate ensures that each tree contributes the same amount. Friedman who was the one to first implement gradient boosting states that lower values for the learning rate typically yields better results [8]. There is however, a trade-off in terms of implementations. Having a low learning rate require more trees in the model and that increases the computational requirements for the model. Each tree then contributes to the overall prediction with a weight that is added to the rest of the trees weights. This means the model aggregates the weights of all the small trees, this value is then the deviance from the average value that the prediction is. When implementing gradient boosting algorithms the depth of the trees will have to specified as it is one of the hyperparameters for this form of boosting algorithm. The trees calculated by gradient boosting algorithms have a fixed depth, and allows for the individual trees to be quite deep. However increasing the depth of the trees, also increases the computational recourses required by the model quite significantly.

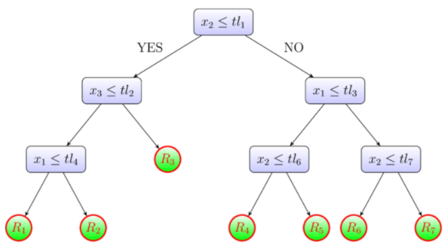


Figure 8: Example of decision tree

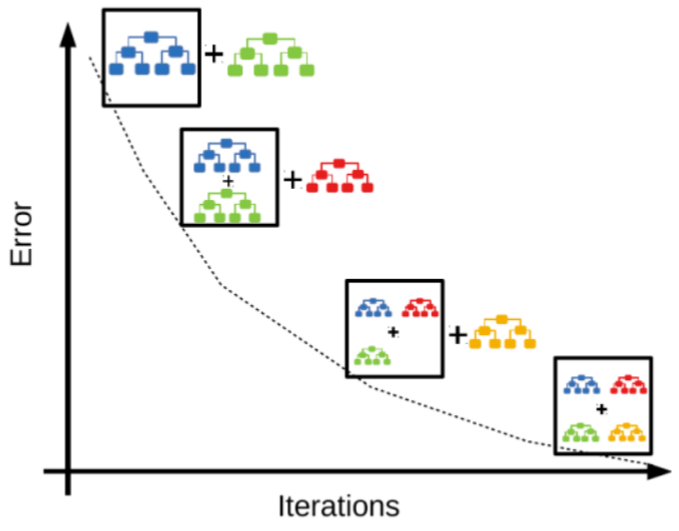


Figure 9: Illustration of how the gradient boosting algorithm sequentially adds trees to reduce error

XGBoosting

The algorithm XGBoost is based upon the main principles of gradient boosting, but there are some very important differences. One of the main differences are the way the decision trees are calculated within the algorithm. The initial leaf in the first tree is always 0.5 instead of the average that the normal gradient boosting algorithm used. The splits in decision trees are based upon a value called information gain, and in regular decision trees this information gain is based upon the reduction in entropy. However, in XGBoosting the splits for the decision trees are calculated based on a value called structure score, or similarity score. This score is a representation of the similarities in the leaf of the split, where a small value means a strong similarity and it is calculated like this:

$$\textit{Similarity score} = \frac{\textit{Sum of Residuals}^2}{\textit{Number of Residuals} + \lambda} \tag{2}$$

Lambda is a regularization parameter and will have to be specified during implementations. The purpose of lambda is to reduce the predictions sensitivity to individual observations, thus helping the model from overfitting. The information gain of a split is then calculated by:

$$Gain = left_{Similarity} + Right_{Similarity} - Root_{Similarity} \tag{3}$$

The method of updating the gain of the split in order to find the split are not the same for all models. When the algorithm has found the best split, the loss function used to calculate the residual errors are squared error algorithm:

$$\sum_{i=1}^{n} L(y_i, p_i) = \frac{1}{2}(y_n - p_n)^2 \tag{4}$$

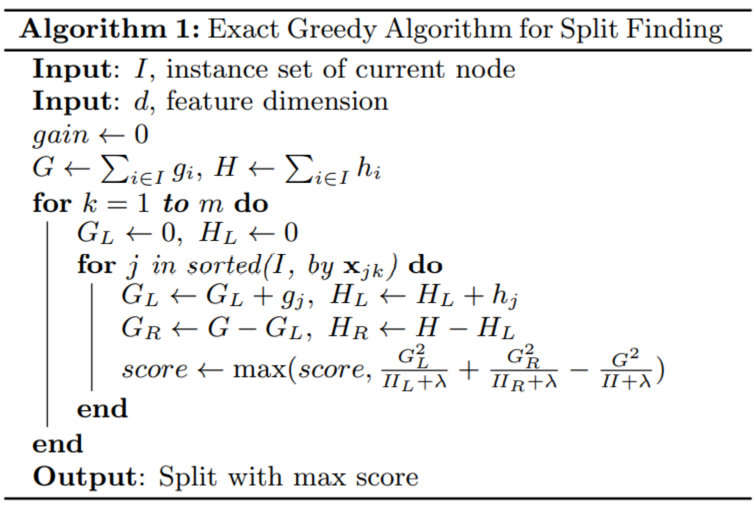
However in order to reducing overfitting the following regularization has been added to the model:

$$\sum_{i=1}^{n} L(y_i, p_i) + \gamma T + \frac{1}{2}\lambda ||w||^2 \tag{5}$$

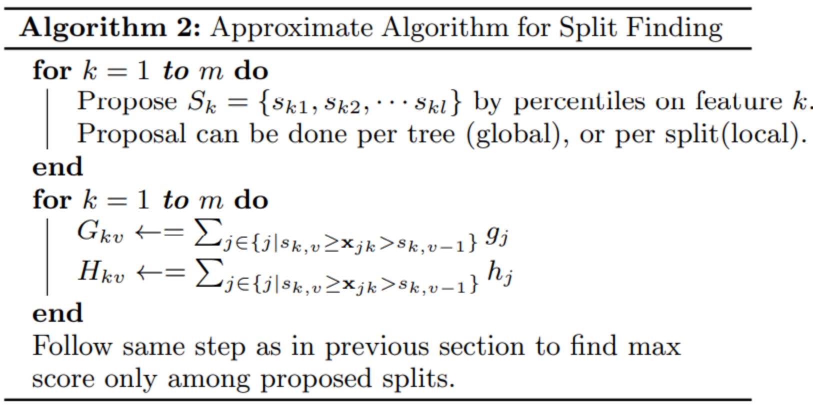
Where the w stands for the output value. That is calculated with the following formula:

$$\textit{Similarity score} = \frac{\textit{Sum of Residuals}}{\textit{Number of Residuals} + \lambda} \tag{6}$$

In theory this would have to be done on all values in the data set. But doing so for very large data sets will be too computationally heavy. Luckily the XGBoost algorithm have multiple ways of updating the gain during training. An exact greedy algorithm and a approximate algorithm for split finding:



```
Algorithm 1: Exact Greedy Algorithm for Split Finding
Input: $I$, instance set of current node
Input: $d$, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i,\ H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ to $m$ do
    $G_L \leftarrow 0,\ H_L \leftarrow 0$
    for $j$ in sorted($I$, by $\mathbf{x}_{jk}$) do
        $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
        $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
        $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
    end
end
Output: Split with max score
```



```
Algorithm 2: Approximate Algorithm for Split Finding
for $k = 1$ to $m$ do
    Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature $k$.
    Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ to $m$ do
    $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
    $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max
score only among proposed splits.
```

Algorithm 1, the exact greedy algorithm iterates over all possible splits and thus comes to the best possible split. But from the XGBoost documentation it is advised to only use this algorithm for smaller dataset since it becomes quite computationally heavy. Therefore the approximate algorithm was developed to increase XGBoost scaling capabilities for bigger data sets. In the implementation that is done by XGBoost the model is able to decide which algorithm to use in the specific situation.

The Approximate algorithm splits up the data set in quantiles, so instead of calculation the split on each input value, these quantiles function as thresholds for splits instead. The quantiles for regression are regular quantiles that contain the same amount of observations in each quantile, but for classification the quantiles are based on the weight of the observations. This wont be explained further since the problem of this project is regressional. Even though this helps reducing the computational requirements for finding the best split. XGBoost goes to an even greater extend by implementing column blocks for parallel learning. This lets the algorithm split up the dataset so multiple threads can store and work on calculating statistical properties. In the paper describing XGBoost, [9], they state that at larger data sets the cache-aware algorithm is able to run twice as fast as the naive.

Long Short Term Memory

Neural Networks

Recurrent Neural Networks

Long Short Term Memory

Implementation

Results

Discussion

Conclusion

Bibliography

[1] Florian Ziel. *Modeling public holidays in load forecasting: a German case study.* 2018. DOI: https://doi.org/10.1007/s40565-018-0385-5.

[2] *Energihåndbogen.* 2019. URL: https://evu.dk/wp-content/uploads/2019/06/Graddage.pdf.

[3] Su-In Lee Scott M. Lundberg. *A Unified Approach to Interpreting Model Predictions.* 2017. URL: https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

[4] Gary R Weckman Iman Ghalehkhondabi Ehsan Ardjmand. *An overview of energy demand forecasting methods published in 2005–2015.* 2016. DOI: https://doi.org/10.1007/s12667-016-0203-y.

[5] José Juan Pazos-Arias Antón Román-Portabales Martín López-Nores. *Systematic Review of Electricity Demand Forecast Using ANN-Based Machine Learning Algorithms.* 2021. DOI: https://doi.org/10.3390/s21134544.

[6] Radu Zmeureanu Jason Runge. *A Review of Deep Learning Techniques for Forecasting Energy use in Buildings.* 2021. DOI: https://doi.org/10.3390/en14030608.

[7] Georgia Papacharalampous Hristos Tyralis. *Boosting algorithms in energy research: a systematic review.* 2021. DOI: https://doi.org/10.1007/s00521-021-05995-8.

[8] Jerome H. Friedman. "GREEDY FUNCTION APPROXIMATION: A GRADIENT BOOSTING MACHINE". In: 1999, pp. 1189–1232.

[9] Carlos Guestrin Tianqi Chen. *XGBoost: A Scalable Tree Boosting System.* 2016. DOI: https://doi.org/10.48550/arXiv.1603.02754.