Improving predictive performance of GBoost and XGBoost Algorithms on Time Series Data Forecasting by working on learning rate

Aasneh Prasad 210101001 * **Pranav Jain 210101078** * **Sahil Danayak 210101092** * **Sanjana Kolisetty 210101093** *

Abstract

Regression trees have been proved to be an apt choice for time series forecasting. In this project, we implemented two successful boosting algorithms i.e., Gradient Boosting and XGBoost. Boosting involves sequentially training multiple base models, where each subsequent model corrects the errors made by the previous ones. This approach allows it to adapt it's focus to the misclassified instances, leading to better performance. Thus, we trained models on the chosen dataset using both the boosting algorithms.

1. Introduction

We have chosen a time series dataset on which we have trained a model using Gradient Boosting and XGBoost Algorithms. We need to understand why regression (or decision) trees are appropriate for working on time series dataset.

Regression trees offer several advantages for analyzing time series data, including their ability to capture non-linear relationships between variables, handle seasonality and trends in time series dataset, robustness to outliers, and compatibility with ensemble methods. Ensemble methods such as Random Forests or Gradient Boosting with regression trees as base learners can further improve forecasting accuracy by aggregating predictions from multiple trees.

Bagging and boosting are both ensemble learning techniques used in machine learning. Bagging involves training multiple base models independently on different subsets of the training data. Examples of bagging algorithms include Random Forest. Boosting involves sequentially training multiple base models, where each subsequent model corrects the errors made by the previous ones. This sequential learning approach allows boosting to adapt its focus to the misclassified instances, potentially leading to better performance. Examples of boosting algorithms include AdaBoost, Gradient Boosting Machines (GBM), XGBoost, LightGBM and CatBoost.

Among the various tree boosting algorithms, XGBoost is one technique that shines in many applications. It is scalable, runs faster than many existing solutions and has several algorithmic optimizations. Since XGBoost is the improvised version of Gradient Boosting Algorithm, we provide a comparative analysis of both algorithms and demonstrate how XGBoost and GBoost perform.

2. Methods

2.1. Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in understanding and deriving insights from any dataset. Time series data often exhibit trends and seasonal patterns that must be identified to capture important features of the dataset. Anomalies or outliers in dataset can arise due to errors in data collection, system malfunctions, or rare events. Time series data may exhibit complex dependencies and correlations between different variables or within the same variable across different time points.

Trigonometric encoding, or Fourier encoding, is particularly useful when dealing with periodic or seasonal patterns that occur over fixed intervals, such as daily, weekly, or yearly cycles. In this method we must identify the cyclical features present in the time series data and replace the original values of the cyclical features with their corresponding sine and cosine transformations to effectively encode the cyclical nature of the data into two continuous variables that capture both the amplitude and phase of the cyclical pattern.

2.2. Gradient Boosting Algorithm

Gradient Boosting is a powerful machine learning algorithm which sequentially combining weak learners, often decision trees, to create an ensemble model that predicts the target variable.

The Gradient Boosting algorithm follows a stage-wise approach to build an ensemble of weak learners. At each stage, a new weak learner, typically a decision tree, is trained to predict the negative gradient of the loss function with respect to the current ensemble's predictions. The algorithm optimizes this objective function by iteratively adding weak

learners to correct the errors made by the previous model.

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) \tag{1}$$

is the objective function where $\hat{y}_i$ is the predicted value, y_i is the actual target value, and l is a differentiable loss function. By iteratively updating predictions based on the negative gradient of the loss function, the algorithm converges towards the optimal solution.

$$\hat{y}^{(t)} = \hat{y}^{(t-1)} - \lambda \nabla_{\hat{y}^{(t-1)}} \mathcal{L}(\hat{y}^{(t-1)}) \tag{2}$$

where $\hat{y}^{(t)}$ is the updated predictions at iteration t, λ is the learning rate, and $\nabla_{\hat{y}^{(t-1)}} \mathcal{L}(\hat{y}^{(t-1)})$ is the negative gradient of the loss function with respect to the predictions at iteration $t-1$.

Friedman's Gradient Boosting Machine (GBM) enhances the standard Gradient Boosting algorithm. It trains sequential regression trees as weak learners to predict gradients rather than updating predictions directly. Let $\hat{y}^{(t-1)}$ represent the predictions from the ensemble at iteration $(t-1)$, and let $g_i^{(t-1)}$ represent the negative gradient of the loss function with respect to the i-th instance's prediction at iteration $(t-1)$. The sequential regression tree is trained to predict these gradients:

$$\hat{g}_i^{(t)} = f_t(\mathbf{x}_i) \tag{3}$$

where $\mathbf{x}_i$ represents the features of the i-th instance and $f_t(\cdot)$ represents the regression tree at iteration t. The step length, denoted by η, is determined through a line search, which iteratively adjusts the step length until an optimal value is found:

$$\eta = \arg\min_{\eta} \sum_i l(y_i, \hat{y}_i^{(t-1)} - \eta \hat{g}_i^{(t)}) \tag{4}$$

where l is the loss function, y_i is the true target value, and $\hat{y}_i^{(t-1)}$ is the prediction from the ensemble at iteration $(t-1)$. By rearranging the loss function in terms of a tree's structure and considering the gradients and number of samples in each leaf node, the algorithm evaluates the quality of different tree structures by:

$$\mathcal{L}_{\text{tree}} = \sum_{j=1}^{T} \left[\left(\frac{\left(\sum_{i \in I_L} g_i\right)^2}{n_L} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{n_R} \right) - \frac{\left(\sum_{i \in I} g_i\right)^2}{n} \right] \tag{5}$$

where I_L and I_R represent instance sets after splits, g_i are the gradients of the loss function, n_L and n_R are the size of I_L and I_R, respectively and n is the sum of n_L and n_R.

2.3. Extreme Gradient Boosting Algorithm

XGBoost is based on the gradient boosting framework, which sequentially combines weak learners (typically decision trees) to create a strong predictive model. In gradient boosting, each subsequent model is trained to correct the errors made by the previous models, thereby minimizing the overall prediction error. To learn the set of functions used in the model, we minimize the following $regularized$ objective.

$$\mathcal{L}(\phi) = \sum_i l\left(\hat{y}_i, y_i\right) + \sum_k \Omega\left(f_k\right) \tag{6}$$
$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Here $\mathcal{L}$ is a differentiable convex loss function that measures the difference between the prediction $\hat{y}_i$ and the target y_i. The second term $\Omega(f)$ penalizes the complexity of the model ($regularization$ and $pruning$). Formally, let ${\hat{y}_i}^{(t)}$ be the prediction of the i-th instance at the t-th iteration, we will need to add f_t to minimize the following objective.

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l\left(y_i, {\hat{y}_i}^{(t-1)} + f_t\left(\mathbf{x}_i\right)\right) + \Omega\left(f_t\right) \tag{7}$$

XGBoost is built makes use of $Newton's$ Method. So instead of just computing the gradient and following it, it uses the second order derivative ($Hessian$) to gather more information to make a better approximation about the direction of the maximum decrease (in the loss function).

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} \left[l\left(y_i, \hat{y}^{(t-1)}\right) + g_i f_t\left(\mathbf{x}_i\right) + \frac{1}{2} h_i f_t^2\left(\mathbf{x}_i\right) \right] + \Omega\left(f_t\right) \tag{8}$$
$$\text{where } g_i = \partial_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right) \& h_i = \partial^2_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right)$$

Define $I_j = \{i \mid q\left(\mathbf{x}_i\right) = j\}$ as the instance set of leaf j. We can rewrite Eq(3) by expanding Ω as follows

$$\mathcal{L}^{(t)} = \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \tag{9}$$

We can compute the optimal weight w_j^* of leaf j by

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \tag{10}$$

and calculate the corresponding optimal value by

$$\mathcal{L}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{11}$$

A greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used instead. Assume that I_L and I_R are the instance sets of left and right nodes after

the split. Lettting $I = I_L \cup I_R$, then the loss reduction after the split is given by

$$\mathcal{L}_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{12}$$

3. Progress and Preliminary Results

3.1. Data Profiling

We intend on using the Hourly Energy Consumption Dataset. This dataset contains the hourly consumption of energy in megawatts (MW) for over past 10 years.
Outlier analysis helps to identify rare events, errors in data collection, or genuine but unusual observations that warrant further investigation. In our dataset, outliers may be removed from the dataset as they are seem to be the result of errors or noise and are unlikely to represent genuine phenomena.

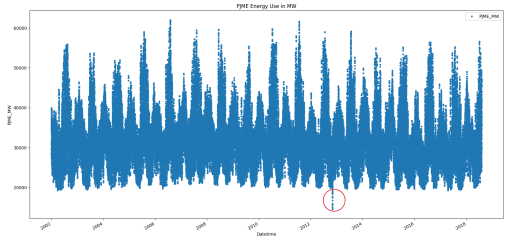


Figure 1. Outlier identification on the dataset

A lag k for a time series variable X_t refers to the value of X at time $t-k$ and is denoted by X_{t-k}. By including lag features as predictors, the model can learn to exploit the temporal patterns and dependencies present in the data to make accurate predictions. In our dataset, values on a particular day are very similar to the values recorded on the same day, in the previous years.

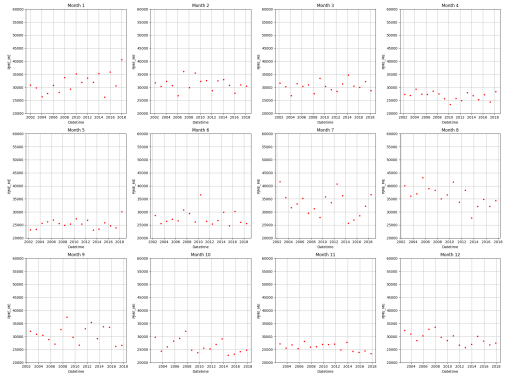
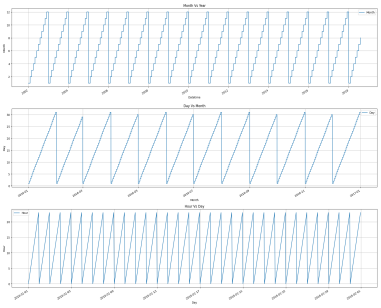
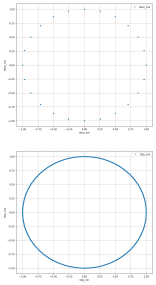


Figure 2. Lag feature analysis

The idea behind trigonometric encoding is to map each cyclical feature (e.g., hour of the day, day of the week) onto a periodic function, typically a sine and cosine function. Energy consumption is cyclic year-wise, i.e, it's values on the same day of the year are similar. Energy consumption values at days, which are not far apart, as well as it's values at different hours of a day, within close proximity, are similar and such relations can easily be captured using trigonometric encoding.



((a)) Cyclicity of hours, days, months



((b)) Trigonometric encoding

Figure 3. Handling Cyclic Features

3.2. GBoost Algorithm

Algorithm 1 Exact Greedy Algorithm for Split Finding in GBoost

Input: I, Instance set of correct node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i\in I} g_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$
  for j *in sorted* $(I,$ *by* $x_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$
    $G_R \leftarrow G - G_L$
    $score \leftarrow \max\left(score, \frac{G_L^2}{n_l} + \frac{G_R^2}{n_r} - \frac{G^2}{n}\right)$
  end
end
Output: Split with max score

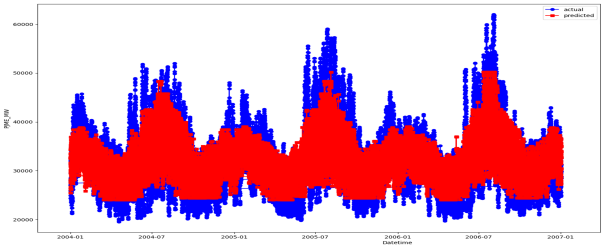


Figure 4. Preliminary results on a fraction of training dataset (GBoost)

3.3. XGBoost Algorithm

Algorithm 2 Exact Greedy Algorithm for Split Finding

Input: I, Instance set of correct 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* $x_{jk})$ **do**
    $G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$
    $score \leftarrow \max\left(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right)$
  end
end
Output: Split with max score

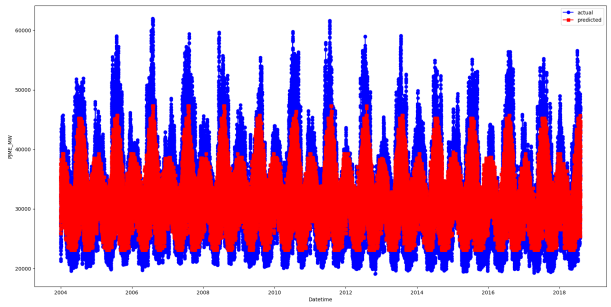


Figure 5. Preliminary results on whole training dataset (XGBoost)

3.4. Overview of Algorithms

Here's a high-level overview of the Exact Greedy Algorithm, which we have implemented, for split finding in GBoost & XGBoost:

- **Initialization:** The algorithm starts with the entire dataset associated with a particular tree node, and initial prediction is the mean of dataset. Each data point consists of features and corresponding labels or gradients/residuals.
- **Feature Selection:** For each feature, the algorithm evaluates potential split points. The algorithm sorts the unique feature values and considers splitting for each value.
- **Split Evaluation, Selection and Splitting:** For each potential split point, the algorithm computes the loss function associated with the resulting partitions. The algorithm selects the split point that maximizes the gain or minimizes the loss function. This split point becomes the optimal decision boundary for the selected feature. The algorithm chooses the feature that provides the maximum gain in impurity reduction for the node. The node is then split into two child nodes based on this feature and split point.
- **Recursion:** The algorithm recursively applies the split finding process to each child node until a stopping criterion is met. Stopping criteria include reaching a maximum tree depth, reaching a minimum number of samples per node, or achieving a minimum gain in loss reduction.

4. Conclusion and Further Intended Experiments

The study explored the application of Gradient Boosting and XGBoost algorithms on time series data forecasting. We observed that regression trees, particularly when used in ensemble methods like Gradient Boosting and XGBoost, offer robust solutions for analyzing time series datasets.

Through exploratory data analysis, we identified key features and patterns in the dataset, including trends, seasonality, and cyclical behaviors. Trigonometric encoding proved to be a valuable technique for capturing cyclical patterns in time series data, enhancing the performance of predictive models.

By implementing both Gradient Boosting and XGBoost algorithms, we gained insights into their respective strengths and performance characteristics. The study demonstrated the effectiveness of these algorithms in modeling complex dependencies and making accurate predictions on time series data.

Moving on to the next phase of our project, we will invest more time in learning rate optimizations, feature engineering and hyperparameter tuning.

1. **Feature engineering** : For seasonal data, features computed using Fourier transform and Wavelet transform can be useful to study periodicity in data and decompose the data spread in its periodic components. We intend to use FFT and Wavelet transform for feature engineering to improve the prediction.
2. **Hyperparameter tuning** : Depth, minimum number of samples in leaf node, number of boosting rounds and minimum gain in loss reduction are hyperparameters in the algorithm. We will use K-fold cross validation technique for better accuracy of our models
3. **Learning rate optimizations** : Currently, we had a fixed learning rate in both models. We will introduce optimization methods namely Simulated annealing and Adam optimizer. Simulated Annealing helps escape local optima and converge towards global optima. Adam optimizer dynamically adjusts the learning rate for each parameter based on the first and second moments of their gradients.

5. References

- XGBoost: Introduction to XGBoost Algorithm in Machine Learning, Analytics Vidhya
- XGBoost: A Scalable Tree Boosting System
- XGBoost Documentation
- Hourly Energy Consumption Dataset, Kaggle
- Handling cyclical features, such as hours in a day, for machine learning pipelines with Python example, Selfidian Academy
- A Comparative Analysis of XGBoost: Candice BentéjacAnna Csörgő, Gonzalo Martínez-Muñoz
- A Proof of Local Convergence for the Adam Optimizer: Sebastian Bock, Martin Weiß
- Understanding gradient boosting as gradient descent.
- Deconstructing time series using Fourier Transform
- Time series features extraction using fourier and wavelet transform on ECG data.