Comparison between XGBoost and LightGBM Using Abnormal Data in a Nuclear Power Plant

Jeong-Mu Eun, Moon-Ghu Park, Jae-Yong Lee*
Dept. of Quantum & Nuclear Engineering, Sejong Univ., 209 Neungdong-rok Gwangjin-gu, Seoul, Republic of Korea
**Corresponding author: jylee002@sejong.ac.kr*

***Keywords :** Machine Learning, Abnormal Data, XGBoost, LightGBM

1. Introduction

Many devices and systems within nuclear power plants provide operators with data on the current state. However, assessing the current data of a nuclear power plant based on numerous variables can be burdensome. Moreover, misjudging the plant's data can lead to significant economic losses. This paper aims to address this issue by employing machine learning. Specifically, we utilize eXtreme Gradient Boosting (XGBoost) and Light Gradient Boosting Machine (LightGBM), known for their robust performance.

XGBoost and LightGBM are machine learning models based on the gradient boosting decision tree method with excellent performance. XGBoost is a powerful machine learning technique that won first place in the machine learning competition platform called Kaggle [1], and LightGBM is a machine learning technique that can quickly learn numerous data.

This paper develops a determination model of abnormal status in nuclear power plants using XGBoost and LightGBM, optimizes it, and compares the two models.

2. Methodology

2.1. Machine Learning Model

The gradient boosting decision tree (GBDT) method is an algorithm method that applies boosting, one of the methods of ensemble learning, to combine multiple weak classifiers based on a tree into a strong classifier with high prediction performance. Boosting is characterized by weighting the data according to the prediction result of the first classifier, and the weighted data affects the learning of the next classifier. It also aims to reduce the residual between the actual and predicted values.

XGBoost is an update of the existing GBDT method, efficiently predicting values using minimal resources even for large amounts of data. Unlike the traditional GBDT method, XGBoost sets weights based on the complexity of the model, resulting in faster and more accurate learning than the existing method [1].

The existing GBDT method has a problem in that it takes a long time to learn when there is a large amount of data for high-dimensional variables. LightGBM was developed to address this challenge [2]. Unlike the level-wise tree growth strategy of the existing GBDT, LightGBM uses a leaf-wise tree growth strategy. The level-wise tree growth strategy prioritizes nodes close to the root node to make the tree balanced, while the leaf-wise tree growth strategy splits at the node with the largest loss change and grows unbalanced. The leaf-wise method is faster to learn than the level-wise method; however, it is prone to overfitting when the data is small [3].

2.2. Hypterparameter

XGBoost and LightGBM have hyperparameters that allow the user to influence the model's performance. Table 1 and Table 2 show the hyperparameters we use for both models [4,5].

Table 1. XGBoost hyperparameters and description

Hyperparameters	Description
eta	Boosting learning rate
n_estimators	Number of gradient boosted trees
max_depth	Maximum tree depth for base learners
min_child_weight	Minimum loss reduction required to make a further partition on a leaf node of the tree
subsample	Subsample ratio of the training instance
colsample_bytree	Subsample ratio of columns when constructing each tree
objective	Specify the learning task and the corresponding learning objective or a custom objective function to be used

Table 2. LightGBM hyperparameters and description

Hyperparameters	Description
learning_rate	Boosting learning rate
num_leaves	Maximum tree leaves for base learners
n_estimator	Number of boosted trees to fit
max_depth	Maximum tree depth for base learners
min_child_weight	Minimum sum of instance weight
subsample	Subsample ratio of the training instance

colsample_bytree	Subsample ratio of columns when constructing each tree
objective	Specify the learning task and the corresponding learning objective or a custom objective function to be used

2.3. Optimization Method [6]

To optimize the hyperparameters of the two models, we use the simulated annealing (SA) method. The process of slowly cooling a heated alloy is called annealing, during which the atoms gradually settle into lower energy levels as they cool. SA is an optimization technique that simulates this process to find the global minima.

The main process of the SA method in this paper is as follows: First, we set the initial temperature and randomly train the model ($current$) using the initial hyperparameters. Second, we train a new model (new) by randomly selecting values near the initial hyperparameters. Third, we compare the performance of the two models using the cost function J. Fourth, by default, we save a model with good performance as the current one; however, we save a model with poor performance at a specific probability as the current. Equation (1) shows this probability, where T represents the current temperature. Fifth, the current temperature is multiplied by the cooling rate. Sixth, we compare the current temperature to the end temperature. If the current temperature is higher, we repeat steps 2-6; otherwise, the process ends.

$$(1)\ P = e^{\frac{J(current)-J(new)}{T}}$$

2.4. Cost Function

We use log loss function to evaluate the models. Log loss function is a typical cost function used to evaluate the performance of models in multiclassification problems. Equation (2) is the log loss function.

$$(2)\ logloss = -\frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M} y_{n,m}\log(p_{n,m})$$

N is the total number of test data, M is the total number of labels, $y_{n,m}$ is the answer probability, and $p_{n,m}$ is the prediction value. The closer the log loss value is to 0, the better the model predicts [7].

2.5. Data [7]

This paper uses the data presented in ref. [7]. The training data consists of actual abnormal operation and simulation data based on operating nuclear power plant. The simulation data is not identical to the actual data, but the trends are similar, making it suitable for training data.

As for the data features, 21 out of the 82 abnormal statuses are simulated, and then the 21 abnormal statuses are divided into 198 cases. Each data contains 5,121 variables, with a total of 827 data available. These variables were obtained for 10 minutes at 1-second intervals. Data are assumed to be normal until a specific time but in an abnormal state afterward. This paper assumes a specific time of 10 seconds.

3. Experiment

For training the two models, we separate the 827 data in ref. [7] into 582 training data, 145 validation data, and 100 test data. Then we train the models and optimise the models with the SA method. When using the SA method, we set the initial temperature to 1,000, the limit temperature to 1, and the cooling rate to 0.9. In addition, cost is the cost function value of the model obtained using test data.

Fig. 1 shows the entire training process of the models.

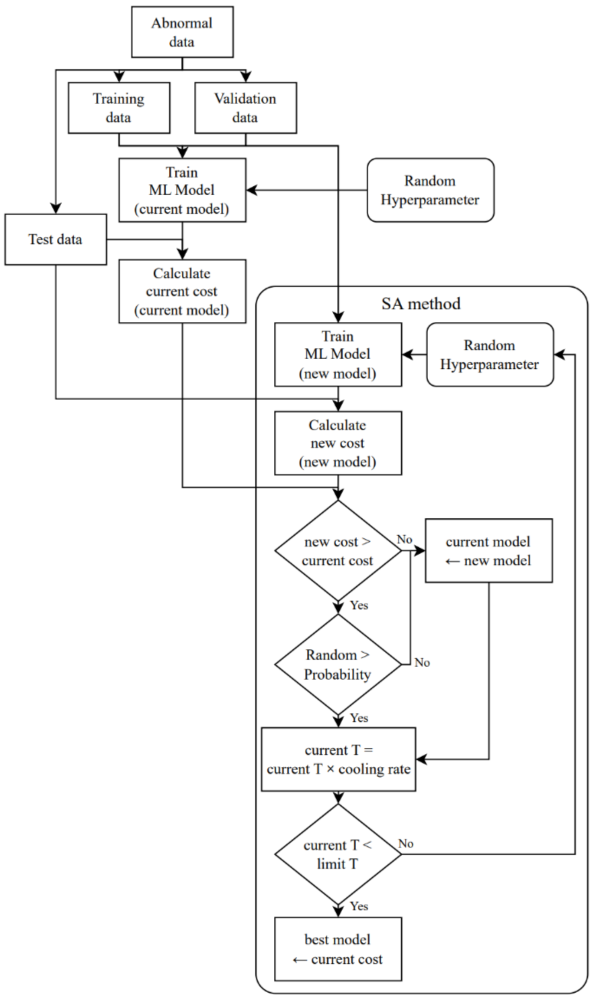
Abnormal data
Training data
Validation data
Train ML Model (current model)
Random Hyperparameter
Test data
Calculate current cost (current model)
SA method
Train ML Model (new model)
Random Hyperparameter
Calculate new cost (new model)
new cost > current cost
No
current model ← new model
Yes
Random > Probability
No
Yes
current T = current T × cooling rate
current T < limit T
No
Yes
best model ← current cost

Fig. 1. Entire training process

When training both models, we used the early stopping. Early stopping is a feature that stops training if the performance of the model does not improve. It prevents the model from overfitting. In this training, we set it to stop if the performance does not improve for 50 epochs.

4. Result and Discussion

Fig. 2 and Fig. 3 are the learning curves of the two models. XGBoost has a training score of 0.0421 and a validation score of 0.6845. LightGBM has a training score of 0.2335 and a validation score of 0.6133.

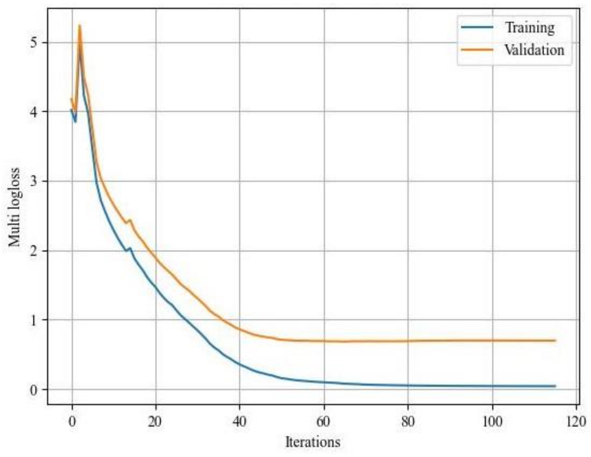

Fig. 2. XGBoost learning curves

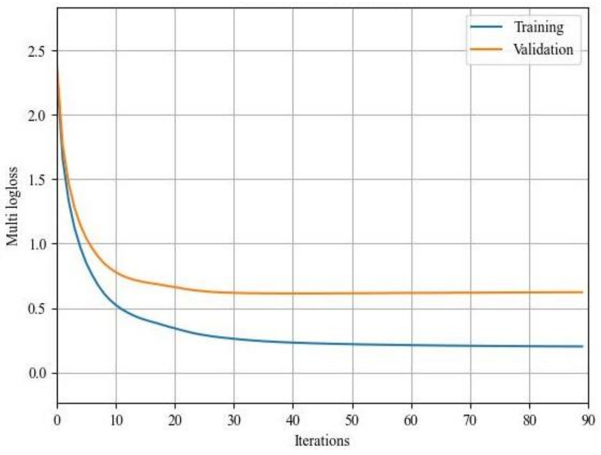

Fig. 3. LightGBM learning curves

Table 3 shows the optimized parameters obtained by SA. Table 4 shows the log loss and training time of the two optimized models. We use test data for log loss.

Table 3. Optimized hyperparameters

Hyperparameters	XGBoost	LightGBM
learning_rate(eta)	0.646495	0.193919
num_leaves	-	400
n_estimator	300	300
max_depth	7	-1
min_child_weight	4	8
subsample	0.462069	0.465144
colsample_bytree	0.709519	0.634746
objective	softprob	multiclass

Table 4. Performance evaluation result

Model	Log loss score	Training time (second)
XGBoost	0.292732	2,336
LightGBM	0.272960	141

Comparing the log loss values in Table 4, we see that LightGBM performs better than XGBoost. Also, comparing the training time, we see that LightGBM is significantly faster than XGBoost. This is related to the size of the training data.

Multiplying the number of driving variables by the measurement time and the number of data files is the size of the training data. According to the calculation, both models were trained with 1,788,253,200 variables. For this reason, we see that LightGBM, which has the advantage of fast training even with a large data size, trained faster than XGBoost.

5. Conclusion

In this paper, we compared XGBoost and LightGBM using abnormal condition data from a nuclear power plant. The results showed that the performance of the two models was similar, but the training time was significantly different due to the size of the training data. This indicates that LightGBM is more efficient than XGBoost in identifying abnormalities in nuclear power plants with many data.

However, there is a limitation in that we could not compare the learning results for all abnormalities because we used only 21 of the 82 abnormalities in this study, and the number of data files is 827. Also, since we used a combination of real and simulated data, there may be differences when evaluating only real data. Also, since we optimized only a limited number of hyper-parameters for both models, there may be differences in performance.

To overcome these limitations, we plan to add more hyperparameters to the two models to see if their performance improves and to compare them with other machine learning models.

REFERENCES

[1] T. Chen, and C. Guestrin, “Xgboost: A scalable tree boosting system”, KDD ‘16, pp. 785-794, 2016.
[2] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y.Liu, “Lightgbm: A highly efficient gradient boosting decision tree”, Advances in neural information processing systems, vol. 30, 2017.
[3] E. Al Daoud, “Comparison between XGBoost, LightGBM and CatBoost using a home credit dataset”, International Journal of Computer and Information Engineering, vol. 13, no. 1, pp. 6-10, 2019.

[4] "XGBoost Python Package Python API Reference", https://xgboost.readthedocs.io/en/stable/python/python_api.html.
[5] "lightgbm.LGBMClassifier," https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html.
[6] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing", science, vol. 220, no. 4598, pp. 671-680, 1983.
[7] K. N. Y. Ho Sun Ryu, Yun Goo Kim, "Development to Diagnose Model of Abnormal Status in Nuclear Power Plant Operation using Machine Learning Algorithms", Transactions of the Korean Nuclear Society Autumn Meeting, 2020.