https://doi.org/10.31891/csit-2025-1-16
UDC 004.85

Mykola ZLOBIN, Volodymyr BAZYLEVYCH
Chernihiv Polytechnic National University

BAYESIAN OPTIMIZATION FOR TUNING HYPERPARAMETRS OF MACHINE LEARNING MODELS: A PERFORMANCE ANALYSIS IN XGBOOST

This paper analyses the optimization of XGBoost hyperparameters using Bayesian optimization with the tree-structured Parzen estimator. The objective is to improve model performance by choosing the optimal hyperparameter values rather than depending on manual or traditional search methods. The performance of machine learning models depends on the selection and tuning of hyperparameters. As a widely used gradient boosting method, XGBoost relies on optimal hyperparameter configurations to balance model complexity, prevent overfitting, and improve generalization. Especially in high-dimensional hyperparameter spaces, traditional approaches including grid search and random search are computationally costly and ineffective. Recent findings in automated hyperparameter tuning, specifically Bayesian optimization with the tree-structured parzen estimator have shown promise in raising the accuracy and efficiency of model optimization. The aim of this paper is to analyze how effective Bayesian optimization is in tuning XGBoost hyperparameters for a real classification issue. Comparing Bayesian optimization with traditional search methods can help to assess its effects on model accuracy, convergence speed, and computing economy. As a case study in this research, a dataset of consumer spending behaviors was used. The classification task aimed to differentiate between two transaction categories: hotels, restaurants, and cafés against the retail sector. The performance of the model was evaluated using loss function minimization, convergence stability, and classification accuracy. This paper shows that Bayesian optimization improves XGBoost hyperparameter tuning, hence improving classification performance while lowering computational costs. The results offer empirical proof that Bayesian optimization outperforms traditional techniques in terms of accuracy, stability, and scalability.

Keywords: XGBoost, Bayesian optimization, hyperparameter tuning, machine learning, tree-structured parzen estimator.

Микола ЗЛОБІН, Володимир БАЗИЛЕВИЧ
Національний університет «Чернігівська Політехніка»

БАЄСОВА ОПТИМІЗАЦІЯ ДЛЯ НАЛАШТУВАННЯ ГІПЕРПАРАМЕТРІВ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ: АНАЛІЗ ПРОДУКТИВНОСТІ В XGBOOST

У цій статті проаналізовано оптимізацію гіперпараметрів XGBoost за допомогою байєсівської оптимізації з використанням деревовидної оцінки Парзена. Мета полягає в тому, щоб покращити продуктивність моделі шляхом вибору оптимальних значень гіперпараметрів замість того, щоб залежати від ручних або традиційних методів пошуку. Ефективність моделей машинного навчання залежить від вибору та налаштування гіперпараметрів. Як широко використовуваний метод градієнтного бустингу, XGBoost покладається на оптимальні конфігурації гіперпараметрів, щоб збалансувати складність моделі, запобігти надмірному пристосуванню та покращити узагальнення. Особливо у високорозмірних просторах гіперпараметрів традиційні підходи, включаючи пошук по сітці та випадковий пошук, є обчислювально дорогими та неефективними. Нещодавні досягнення в автоматизованому налаштуванні гіперпараметрів, зокрема, байєсівська оптимізація за допомогою деревовидної парзен-оцінки, продемонстрували перспективність підвищення точності та ефективності оптимізації моделей. Мета цієї статті - проаналізувати, наскільки ефективною є байєсівська оптимізація при налаштуванні гіперпараметрів XGBoost для реальної задачі класифікації. Порівняння байєсівської оптимізації з традиційними методами пошуку може допомогти оцінити її вплив на точність моделі, швидкість конвергенції (збіжності) та економію обчислень. В якості прикладу в цьому дослідженні було використано набір даних про поведінку споживачів щодо витрат. Завдання класифікації полягало в тому, щоб розрізнити дві категорії транзакцій: готелі, ресторани та кафе від роздрібної торгівлі. Ефективність моделі оцінювалася за допомогою мінімізації функції втрат, стабільності збіжності та точності класифікації. Ця стаття показує, що байєсівська оптимізація покращує налаштування гіперпараметрів XGBoost, а отже, підвищує ефективність класифікації, знижуючи при цьому обчислювальні витрати. Результати є емпіричним доказом того, що байєсовська оптимізація перевершує традиційні методи з точки зору точності, стабільності та масштабованості.

Ключові слова: XGBoost, баєсовса оптимізація, налаштування гіперпараметрів, машинне навчання, деревовидна оцінка парзена.

Introduction

In machine learning, hyperparameters are defined before the start of the training process. They regulate the learning process, affecting how a model generalizes to unseen data, unlike model parameters, which are learned from data during training. The choice of hyperparameters can influence a model's performance, generalization capacity, and computational efficiency. In deep learning, for example, the selection of hyperparameters including learning rate, batch size, and network architecture can define whether a model converges successfully or fails to learn significant patterns. While an optimal rate promotes effective learning, an improper rate could cause slow convergence or even divergence. With smaller batches producing noisy but useful gradient estimates and larger batches yielding more consistent updates, the batch size similarly influences the stability and speed of the training process. Like dropout rates, regularization values are also important in preventing overfitting by modulating the model's memorizing capability of the training data. Therefore, balancing bias and variance depends on adjusting these hyperparameters, guaranteeing the model's performance on both training and unseen data [1,2]. Studying hyperparameter tuning in supervised learning models is driven by the necessity to maximize generalization and

model performance. Studies show that models with correctly configured hyperparameters can outperform models with default values. For instance, a research of machine learning publications revealed a gap in model optimization procedures with only 20.31% of respondents stating their hyperparameter choices and tuning approaches. This control can cause problems duplicating results and less than-ideal model performance. Thus, not only does systematic hyperparameter adjustment improve model accuracy but also guarantees transparency and repeatability in machine learning research [3].

This paper provides results in the field of machine learning, showing an efficient and scalable approach for optimizing XGBoost in real classification problems. The findings contribute to the broader field of automated hyperparameter optimization and offer guidance on selecting hyperparameters that improve both model accuracy and computational efficiency.

Related work

Hyperparameters are used in shaping the efficiency and performance of machine learning models. Their impact differs among several algorithms: decision trees, support vector machines, artificial neural networks, and ensemble methods. Decision trees rely on hyperparameters such as maximum depth, minimum samples per leaf, and the criterion for splitting nodes. The complexity of the tree is controlled by maximum depth; deeper trees may overfit the data but can also capture more complicated patterns. The minimum samples per leaf determine the smallest number of samples required to form a leaf, affecting the granularity of the splits. Gini impurity or entropy, the splitting criterion shapes the tree's data partitioning at every node. Changing these hyperparameters changes the decision boundaries, balancing the trade-off between model generalization and complexity [4,5]. Support vector machines use hyperparameters including the regularizing parameter, kernel type, and kernel-specific parameters. By controlling the trade-off between low error on the training data and minimalizing the model's complexity, the regularization parameter C helps to prevent overfitting. The Support vector machines can manage non-linear interactions using the choice of the kernel (linear, Poisson, radial basis function), hence transforming input data into higher-dimensional spaces. Further defining the flexibility and shape of the decision boundary are kernel-specific parameters such as the degree in polynomial kernels or gamma in radial basis function kernels. The Support vector machines cannot build optimal separating hyperplanes that generalize effectively to unknown data without proper tuning of these hyperparameters [6,7]. Artificial neural networks have hyperparameters including the number of hidden layers, number of neurons per layer, learning rate, and activation functions. Defined by hidden layers and neurons, the architecture controls the capacity of the network to model patterns. The speed and stability of the training process depend on the learning rate; meaning that a rate too high may lead the model to converge too early; a rate too low may produce too slow learning. The capacity of the network to capture non-linearities in the data depends on activation functions including sigmoid, tanh, or ReLU. Effective training and performance of artificial neural networks depend on proper selecting values for these hyperparameters [8,9]. Furthermore, dependent on important hyperparameters are ensemble learning methods, which aggregate predictions from several models to raise general performance. Hyperparameters in methods such as Random Forests include the number of trees taken into account for splitting at every node as well as the ensemble size of trees. More trees can improve performance but raise computational costs. The variety and correlation among the trees depend on the number of features taken into account at every split, therefore affecting the robustness of the ensemble. In boosting algorithms, such as AdaBoost or Gradient Boosting, hyperparameters like the learning rate and number of boosting stages are considered important. The learning rate regulates the contribution of every model to the ensemble; the number of stages decides the combination of the models. Careful tuning of these hyperparameters is used to balance bias and variance, leading to improved predictive performance [10-12].

In machine learning, hyperparameter tuning is needed since it directly affects generalization and model performance. From traditional techniques to new automated approaches, several methods have been developed to find ideal hyperparameter settings. Traditional techniques include manual tuning, grid search, and random search. Manual tuning uses the knowledge to change hyperparameters depending on experience and intuition. Although simple, especially for complex models with many hyperparameters, it takes time and might not produce the best results. Grid search is methodically looking over a predefined set of hyperparameter values. Every combination is assessed and the one with the best performance is chosen. Grid search becomes computationally expensive as the number of hyperparameters rises, despite its simplicity, producing the "curse of dimensionality" [13,14]. Random search chooses randomly among given ranges of hyperparameter combinations. It is used in cases when just a few hyperparameters significantly affect model performance, studies have shown that random search can be more efficient than grid search [13,15]. Automated methods of hyperparameter optimization, including Bayesian optimization, genetic algorithms, and reinforcement learning-based tuning have been developed to go beyond the constraints of conventional methods. Often employing Gaussian processes, Bayesian optimization treats the goal function as a black box and generates a surrogate model to approximate it. Bayesian optimization effectively converges to optimal settings by repeatedly choosing hyperparameters that balance exploration and exploitation. It has proved to find better hyperparameters with fewer evaluations than grid and random search [16]. Inspired by natural evolution, genetic algorithms develop a population of hyperparameter sets over generations by means of operations including selection, crossover, and mutation. This method has been used in many machine-learning

applications for optimizing complex, high-dimensional spaces. Reinforcement Learning based tuning represents the approach, where an agent learns to change hyperparameters by interacting with the learning algorithm and getting performance-based feedback. The agent discovers the best hyperparameter policies over time. This method has been used in neural architecture search as well as other fields needing dynamic hyperparameter customization [17].

Despite these findings, there remains a research gap in applying Bayesian optimization to optimize hyperparameters for gradient-boosting models such as XGBoost. While previously presented research has shown how effective Bayesian optimization in improving model accuracy is, there are limited studies that have evaluated its impact on real segmentation tasks where class imbalance, feature selection, and computational constraints is considered. This paper aims to address this gap by investigating how Bayesian optimization can be used to tune XGBoost hyperparameters for classification tasks. The main contribution of this paper is to provide empirical evidence that Bayesian optimization outperforms traditional tuning methods when applied to XGBoost. In the process of evaluating different hyperparameter configurations, this research shows that optimized hyperparameter selection leads to improved model accuracy, reduced overfitting, and better generalization performance. It also highlights the importance of hyperparameters such as max_depth, gamma, colsample_bytree, and min_child_weight in boosting model performance. The results indicate that Bayesian optimization achieves optimal performance with fewer evaluations compared to other search methods.

XGBoost - Mathematical model, definitions, and formulation

XGBoost or extreme gradient boosting represents an optimized machine learning algorithm based on gradient boosting decision trees. It is designed for efficiency, scalability, and high predictive accuracy. The algorithm builds an ensemble of weak learners (decision trees) sequentially, where each new tree corrects the errors of the previous trees by minimizing an objective function. The success of XGBoost is seen through its regularization mechanisms, parallelized execution, and effective tree-pruning techniques [18,19]. XGBoost uses an additive learning process, where new models are added iteratively to improve the overall prediction. Unlike bagging or random forests, which aggregate independent trees, boosting methods such as XGBoost sequentially train trees to correct errors made by prior models. XGBoost optimizes a loss function that consists of two components:

1. Loss function L - that measures how well the model fits the training data.
2. Regularization term $\Omega(f)$ – that controls model complexity to prevent overfitting.

For a given dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$ with n instances and m features, the XGBoost model constructs an ensemble of K regression trees. Each tree $f_k(x)$ contributes to the prediction:

$$\hat{y}_i = \sum_{k=1}^{K} \quad f_k(x_i) \tag{1}$$

where $\hat{y}_i$ is the predicted output, and each $f_k(x)$ is a tree-based function mapping an input x_i to a score. The objective function optimized by XGBoost is:

$$L = \sum_{i=1}^{n} \quad l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \quad \Omega(f_k) \tag{2}$$

where: $l(y_i, \hat{y}_i)$ - is a differentiable loss function, such as squared error for regression or logistic loss for classification; $\Omega(f)$ - is the regularization term, which penalizes model complexity.

Regularization is used in controlling the complexity of the trees. The function $\Omega(f)$ is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \quad \omega_j^2 \tag{3}$$

where: T - is the number of leaf nodes in the tree; ω_j - represents the weight associated with each leaf node; γ - is a parameter controlling the penalty for adding a new leaf node; λ - is the L2 regularization coefficient applied to leaf weights. This regularization term prevents complex trees by penalizing models that add too many nodes.

At each iteration, XGBoost improves the model by adding a new tree that minimizes the residual error from previous predictions. This is done using the second-order Taylor approximation of the loss function:

$$L^{(t)} \approx \sum_{i=1}^{n} \quad \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) \tag{4}$$

where: $g_i = \frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}_i}$ is the first derivative (gradient) of the loss function and $h_i = \frac{\partial^2 l(y_i, \hat{y}_i)}{\partial \hat{y}_i{}^2}$ is the second derivative (Hessian) of the loss function.

Minimizing this function enables XGBoost to optimize the tree split criteria. This second-order information enables more precise adjustments to the model at each boosting step [20].

XGBoost constructs trees by selecting the best-split points that maximize information gain while considering regularization. The split criterion is based on the gain function:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma \tag{5}$$

where: G_L, G_R are the sum of gradients for the left and right child nodes; H_L, H_R are the sum of Hessians for the left and right child nodes; λ is the regularization parameter controlling the complexity of leaf weights; γ penalizes adding new splits to prevent unnecessary complexity.

A split is accepted if $Gain > 0$, meaning that the new tree structure provides an improvement in predictive performance.

The effectiveness of XGBoost is dependent on the selection of hyperparameters. Instead of relying on manual tuning or exhaustive search methods, this paper implements Bayesian optimization using the tree-structured Parzen estimator to efficiently find optimal hyperparameter values. Bayesian optimization builds a probabilistic model to estimate the loss function and selects hyperparameter configurations that improve the model iteratively. Tree-structured Parzen estimator models the probability distribution of promising hyperparameter regions and selects candidates based on expected improvement. This method reduces the number of required evaluations compared to traditional tuning methods like grid search.

Tuning hyperparameters of the XGBOOST model

The dataset considered in this research [21] includes 440 instances across 7 features. More precisely, the data describes annual spending behaviours of customers across different product categories: Fresh, Milk, Grocery, Frozen, Detergents/Paper, and Delicatessen. These features are quantified by the annual spending in monetary units. In addition, the dataset includes two categorical variables: Channel and Region. The Channel variable indicates whether a customer makes transactions within Hotels, Restaurants, or Cafés, or within the Retail sector. The second categorical variable, Region, specifies the location of a customer. The dataset is based on 77 different customers, with the majority of information coming from hotels, restaurants, or cafés (298 out of 440 unique instances). Because of its structure, the dataset is suitable for segmentation tasks, optimization of business strategies, and demand forecasting.

To propose and implement the XGBoost classification model, the Python libraries Pandas and NumPy are used for data manipulation, while Scikit-Learn is integrated to evaluate model performance. Additionally, the XGBoost and Hyperopt libraries are employed for model initialization and hyperparameter optimization, respectively.

The dataset is then split into training and test sets using an 80:20 ratio in favor of training data. All features, except the Channel variable, serve as inputs to the model, while the Channel variable is used as the target output. The main objective of this setup is to develop a model capable of classifying customers into two categories: those making transactions within Hotels, Restaurants, or Cafés, and those within the Retail sector. To optimize the classification process, the class labels for the “Channel” variable are converted into binary values.

Bayesian optimization, a component of this study for fine-tuning hyperparameters, is implemented using the Hyperopt library. This method explores the search space and identifies optimal parameter values that minimize the loss function. The optimization process involves defining the domain space, specifying the objective function, selecting the tree-structured Parzen Estimator as the optimization algorithm, and evaluating the results. By considering various hyperparameters within the defined search space, this research incorporates the XGBoost hyperparameters presented in Table 1.

Table 1

The domain space hyperparameters

Hyperparameter	Description	Range/Value
max_depth	Maximum depth of trees	3 to 18 (integer)
gamma	Minimum loss reduction required for a split	1 to 9 (continuous)
reg_alpha	L1 regularization term on weights	40 to 180 (integer)
reg_lambda	L2 regularization term on weights	0 to 1 (continuous)
colsample_bytree	Fraction of features used per tree	0.5 to 1 (continuous)
min_child_weight	Minimum sum of instance weight in a child	0 to 10 (integer)
n_estimators	Number of boosting ounds	180 (fixed)
seed	Random seed for reproducibility	0 (fixed)

The final part of proposing the coding solution was to create the trials object to store relevant information like loss values and tested combinations of hyperparameters.

Results

The number of trial attempts is set to 100, after which, the best-performing values of hyperparameters are identified (after approximately 9 seconds) and presented in Fig.1.

```
The best hyperparameters are :

{'colsample_bytree': 0.5064796146601775, 'gamma': 2.964193854351327, 'max_depth': 14.0, 'min_child_weight': 9.0, 'reg_alpha': 70.0, 'reg_lambda': 0.1139297603374341 7}
```

Fig.1. The optimal values of tested hyperparameters

It is important to highlight that the distribution of scores for trials converges around 0.3484, indicating that many hyperparameter combinations did not lead to a satisfactory improvement in performance. The best loss value achieved during the optimization process was 0.8939, which determined the selection of the optimal hyperparameter configuration.

The optimization algorithm identified a few particularly influential hyperparameters. First, the colsample_bytree parameter reached a maximum value of 0.872, meaning that 87.2% of features were used per tree. This balance contributed to both diversity and stability in tree formation. Next, the gamma value was found to be 8.349, enforcing strong regularization to prevent overfitting. Additionally, a max_depth of 9 was selected, suggesting a moderately deep tree structure. The min_child_weight was set to 9, ensuring that a minimum sum of instance weights was required in child nodes before splitting, further reducing the risk of overfitting. The model was also optimized using reg_alpha set to 63, applying strong L1 regularization. Finally, reg_lambda was determined to be 0.0427, ensuring minimal L2 regularization and promoting generalization.

A visual summary of these hyperparameters is provided in Fig.2, which presents the hyperparameter importance plot. This plot illustrates the relationship between each hyperparameter and the optimization score. Each bar in the figure represents the impact of a specific hyperparameter on model performance. These insights allow for a clear evaluation of which parameters play a crucial role in enhancing the model's effectiveness.

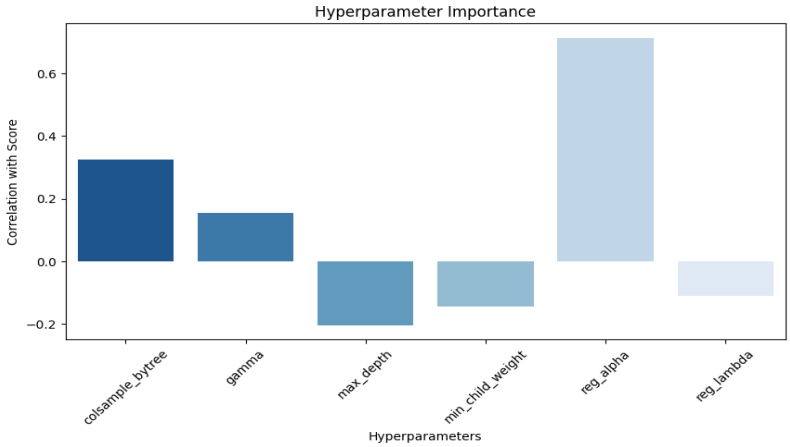

Fig.2. The Hyperparameters' importance plot

Finally, Fig.3 presents the convergence plot for this study. This visualization illustrates the optimization process over time, highlighting the model's performance throughout the search progression. Each point on the plot represents a specific trial, indicating the corresponding achieved loss or score. Ideally, the plot should exhibit a steady decrease in the loss function. In this case, two significant drops are observed, signaling that many trials did not contribute to further improvements. After the 24th iteration, no notable decrease in the loss function is observed, indicating that the optimization process has reached a stable point.

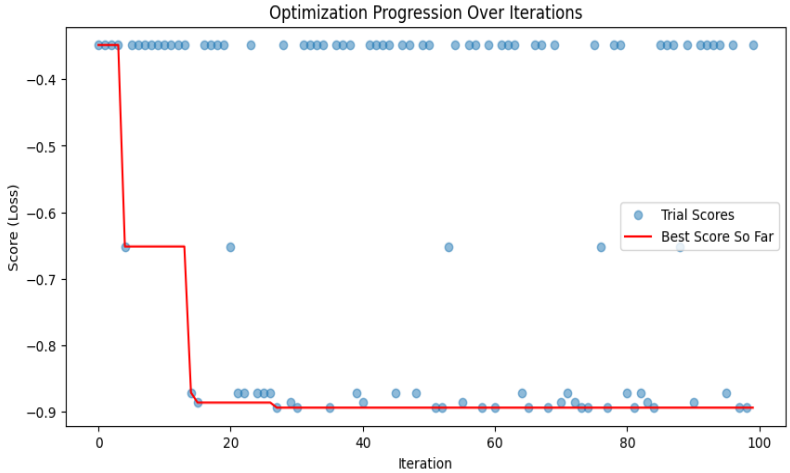

Fig.3. The convergence plot

Conclusion

This paper analyses the optimization of XGBoost hyperparameters using Bayesian optimization with the tree-structured Parzen estimator. The objective is to improve model performance by choosing the optimal hyperparameter values rather than depending on manual or traditional search methods. The results showed that despite lowering computational costs, Bayesian optimization increases classification accuracy. The paper used a real

dataset on customer spending behaviors, aiming to classify transactions into two distinct categories. The optimal hyperparameter configuration was found by using Bayesian optimization, therefore improving stability in model predictions. The results revealed that maximizing XGBoost's performance depends on several hyperparameters including max_depth, gamma, colsample_bytree, and min_child_weight. The insight from the data was that, following 24 rounds, no more loss reduction was seen, suggesting that Bayesian optimization effectively converged to an optimal solution faster than traditional search methods. This emphasizes how well the tree-structured Parzen estimator explores the hyperparameter space, so it is a better option than random or grid search.

The paper highlights the significance of XGBoost's regularizing methods in preventing overfitting while preserving accuracy. The chosen L1 and L2 regularization values helped to produce a well-balanced model capable of efficiently extending over several data sets. Bayesian optimization has certain restrictions even if it is rather successful. The technique still depends on the choice of search space boundaries since inappropriate parameter ranges can influence optimization effectiveness. Furthermore, even if Bayesian optimization lowers computing time, its advantages decrease in low-dimensional hyperparameter spaces when grid search can still be competitive. The analyses showed how well Bayesian optimization fine-tuned XGBoost hyperparameters, produced a scalable and highly performing classification model. Its use in different machine learning algorithms, hybrid optimization methods, and real-time hyperparameter tuning for dynamic datasets could be investigated in the next studies.

References

1. Wang, B., & Gong, N. Z. Stealing hyperparameters in machine learning. *In 2018 IEEE symposium on security and privacy (SP) IEEE*. 2018. C.36-52.

2. Kumar, J. Hyperparameters in Deep Learning: A Comprehensive Review. *International Journal of Intelligent Systems and Applications in Engineering*. 2024. T.12. №4. C.4015-4023.

3. Arnold, C., Biedebach, L., Küpfer, A., Neunhoeffer, M. The role of hyperparameters in machine learning models and how to tune them. *Political Science Research and Methods*. 2024. T.12. №4. C. 841-848.

4. Costa, V. G., Pedreira, C. E. Recent advances in decision trees: An updated survey. *Artificial Intelligence Review*. 2023. T.56. №5. C.4765-4800.

5. Gajowniczek, K., Dudziński, M. Influence of Explanatory Variable Distributions on the Behavior of the Impurity Measures Used in Classification Tree Learning. *Entropy*. 2024. T. 26. №12. C.1020.

6. Sipper, M. High per parameter: A large-scale study of hyperparameter tuning for machine learning Algorithms. *Algorithms*. 2022. T.15. №9. C.315.

7. Elgeldawi, E., Sayed, A., Galal, A. R., Zaki, A. M. Hyperparameter tuning for machine learning algorithms used for arabic sentiment analysis. *Informatics*. 2021. T.8. №4. C.79.

8. Kadhim, Z. S., Abdullah, H. S., Ghathwan, K. I. Artificial Neural Network Hyperparameters Optimization: A Survey. *Int. J. Online Biomed. Eng*. 2022. T.18. №15. C.59-87.

9. Probst, P., Boulesteix, A. L., Bischl, B. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*. 2019. T.20. №53. C.1-32.

10. Mienye, I. D., Sun, Y. A survey of ensemble learning: Concepts, algorithms, applications, and prospects. *Ieee Access*. 2022. T.10. C.99129-99149.

11. Haixiang, G., Yijing, L., Yanan, L., Xiao, L., Jinling, L. BPSO-Adaboost-KNN ensemble learning algorithm for multi-class imbalanced data classification. *Engineering Applications of Artificial Intelligence*. 2016. T.49. C.176-193.

12. Kumar, P. S., Swathi, M. N. Rain Fall Prediction using Ada Boost Machine Learning Ensemble Algorithm. *Journal of advanced applied scientific research*. 2023. T.5. №4. C.67-81.

13. Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.L. Deng, D.Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*. 2023. T.13. №2. C.1484.

14. Du, X., Xu, H., Zhu, F. Understanding the effect of hyperparameter optimization on machine learning models for structure design problems. *Computer-Aided Design*. 2021. T.135. C.103013.

15. Ali, Y. A., Awwad, E. M., Al-Razgan, M., Maarouf, A. Hyperparameter search for machine learning algorithms for optimizing the computational complexity. *Processes*. 2023. T.11. №2. C.349.

16. Wu, J., Chen, X. Y., Zhang, H., Xiong, L. D., Lei, H., Deng, S. H. Hyperparameter optimization for machine learning models based on Bayesian optimization. *Journal of Electronic Science and Technology*. 2019. T.17. №1. C.26-40.

17. Rijsdijk, J., Wu, L., Perin, G., Picek, S. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*. 2021. №3. C. 677-707.

18. Ramraj, S., Uzir, N., Sunil, R., Banerjee, S. Experimenting XGBoost algorithm for prediction and classification of different datasets. *International Journal of Control Theory and Applications*. 2016. T.9. №40. C.651-662.

19. Dhaliwal, S. S., Nahid, A. A., Abbas, R. Effective intrusion detection system using XGBoost. *Information*. 2018. T.9. №7. C.149.

20. Yi, X., Sun, J., Wu, X. Novel feature-based difficulty prediction method for mathematics items using XGBoost-based SHAP model. *Mathematics*. 2024. T.12. №10, C.1455.

21. Cardoso, M. Wholesale customers [Dataset]. UCI Machine Learning Repository, 2013. Available at: https://doi.org/10.24432/C5030X

Mykola Zlobin **Микола Злобін**	PhD student in Computer Science, Chernihiv Polytechnic National University, Chernihiv, Ukraine, https://orcid.org/0009-0000-7653-6109 e-mail: mykolay.zlobin@gmail.com, Scopus Author ID: 59337918100	Аспірант у галузі Комп'ютерні науки, Національний університет «Чернігівська політехніка», Чернігів, Україна
Volodymyr Bazylevych **Володимир Базилевич**	PhD in Economics, Associate Professor, Head of ESI EIT, Chernihiv Polytechnic National University, Chernihiv, Ukraine, https://orcid.org/0000-0001-8935-446X e-mail: bazylamar@stu.cn.ua, Scopus Author ID: 57214432127	Кандидат економічних наук, доцент, директор НІІ ЕІТ, Національний університет «Чернігівська політехніка», Чернігів, Україна