Journal of Computer and Communications, 2024, 12, 139-146
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Parallel Inference for Real-Time Machine Learning Applications

Sultan Al Bayyat, Ammar Alomran, Mohsen Alshatti, Ahmed Almousa, Rayyan Almousa, Yasir Alguwaifli

College of Computer Science and Information Technology, Imam Abdulrahman Bin Faisal University, Dammam, Saudi Arabia
Email: 2200004334@iau.edu.sa, 2200002381@iau.edu.sa, 2200001221@iau.edu, 2200004665@iau.edu.sa, 2200000173@iau.edu.sa, ymalguwaifli@iau.edu.sa

How to cite this paper: Al Bayyat, S., Alomran, A., Alshatti, M., Almousa, A., Almousa, R. and Alguwaifli, Y. (2024) Parallel Inference for Real-Time Machine Learning Applications. *Journal of Computer and Communications*, **12**, 139-146.
https://doi.org/10.4236/jcc.2024.121010

Received: December 12, 2023
Accepted: January 27, 2024
Published: January 30, 2024

Abstract

Hyperparameter tuning is a key step in developing high-performing machine learning models, but searching large hyperparameter spaces requires extensive computation using standard sequential methods. This work analyzes the performance gains from parallel versus sequential hyperparameter optimization. Using scikit-learn's Randomized SearchCV, this project tuned a Random Forest classifier for fake news detection via randomized grid search. Setting n_jobs to −1 enabled full parallelization across CPU cores. Results show the parallel implementation achieved over 5× faster CPU times and 3× faster total run times compared to sequential tuning. However, test accuracy slightly dropped from 99.26% sequentially to 99.15% with parallelism, indicating a trade-off between evaluation efficiency and model performance. Still, the significant computational gains allow more extensive hyperparameter exploration within reasonable timeframes, outweighing the small accuracy decrease. Further analysis could better quantify this trade-off across different models, tuning techniques, tasks, and hardware.

Keywords

Machine Learning Models, Computational Efficiency, Parallel Computing Systems, Random Forest Inference, Hyperparameter Tuning, Python Frameworks (TensorFlow, PyTorch, Scikit-Learn), High-Performance Computing

1. Introduction

Machine learning models have enabled breakthroughs in computer vision, natural language processing, and other AI applications. However, these complex models can have billions of parameters, making inferences computationally expensive. Sequential execution on a single CPU often fails to meet the real-time latency requirements necessary for responsive applications. Numerous statistical techniques for analysis, mining, and prediction are available with machine learning. Due to their high computational costs, all of these algorithms are excellent candidates for parallel architecture and parallel programming techniques [1]. Scientific and technical computing have made use of parallel computing systems. In research and engineering, computational problems are usually complex and resource intensive. Effective solutions to these issues require the usage of parallel computing systems, which may consist of several processing units [2]. This project aims to accelerate machine learning application using Random Forest inference through parallel programming techniques.

2. Background

Hyperparameter tuning is a critical step in developing high-performance machine learning models. However, searching across hyperparameter spaces is computationally expensive, often requiring training and evaluating hundreds to thousands of model configurations [3]. Sequentially processing these model fits limits the extent of hyperparameter exploration feasible within reasonable time constraints [4].

Parallel computing techniques can help accelerate hyperparameter tuning by distributing the workload across multiple processors and hardware devices [5]. This approach allows for the simultaneous execution of independent model training jobs, as opposed to a sequential process [6]. Leveraging parallelism reduces overall tuning time by an order of magnitude or more depending on the search space size and hardware capabilities [7].

Major parallel computing frameworks used in machine learning include TensorFlow [8], PyTorch [9], and scikit-learn [10] in Python. These provide built-in support for data and model parallelism to exploit multiprocessing and vectorization on CPUs and massively parallel architectures of GPUs.

Overall, parallelizing hyperparameter search allows more configurations to be evaluated in shorter timeframes. This leads to better model performance through more extensive exploration guided by efficiency gains from parallelism [11]. The proposed analysis aims to quantify these improvements on a Random Forest classifier.

3. Methodology

The objective of this analysis was to evaluate the performance improvements from parallelizing hyperparameter tuning for Random Forests (RF), a popular supervised machine learning technique. RF classifiers consist of an ensemble of decision trees [12], which individually are prone to overfitting but collectively can model complex nonlinear relationships robustly. This is achieved by training each constituent decision tree on random subsets of features and data [13], injecting variability that improves generalization. Tuning hyperparameters like the number of trees and maximum depths is crucial for optimizing accuracy by controlling overfitting and efficiency [14]. Key advantages of random forests include flexible nonlinear modeling, resistance to overfitting, and built-in feature importance metrics. However, as an ensemble method, both training and prediction time scale poorly with additional data and estimators [15]. Overall, RF represents a versatile machine learning approach whose wide-ranging hyperparameters enable precision tuning for accuracy and efficiency—though at significant computational expense.

A visualization of the Random Forest mechanism is provided in **Figure 1** to illustrate how the algorithm works [16].

The dataset used for analysis was the Fake News Detection dataset from Kaggle [17]. This dataset contains two CSV files—one with ~23,000 fake news articles and one with ~21,000 real/reliable news articles. This diverse dataset was ideal for a binary text classification task aimed at distinguishing between fake and real news. As shown in **Figure 2**, a histogram highlights the difference in length between real and fake news articles in the dataset.

The preprocessing of the raw text content from the CSV files involved:

- Lowercasing all text.
- Removing punctuation and digits.
- Tokenizing into words.
- Removing English stop words.
- Stemming words to their root form.

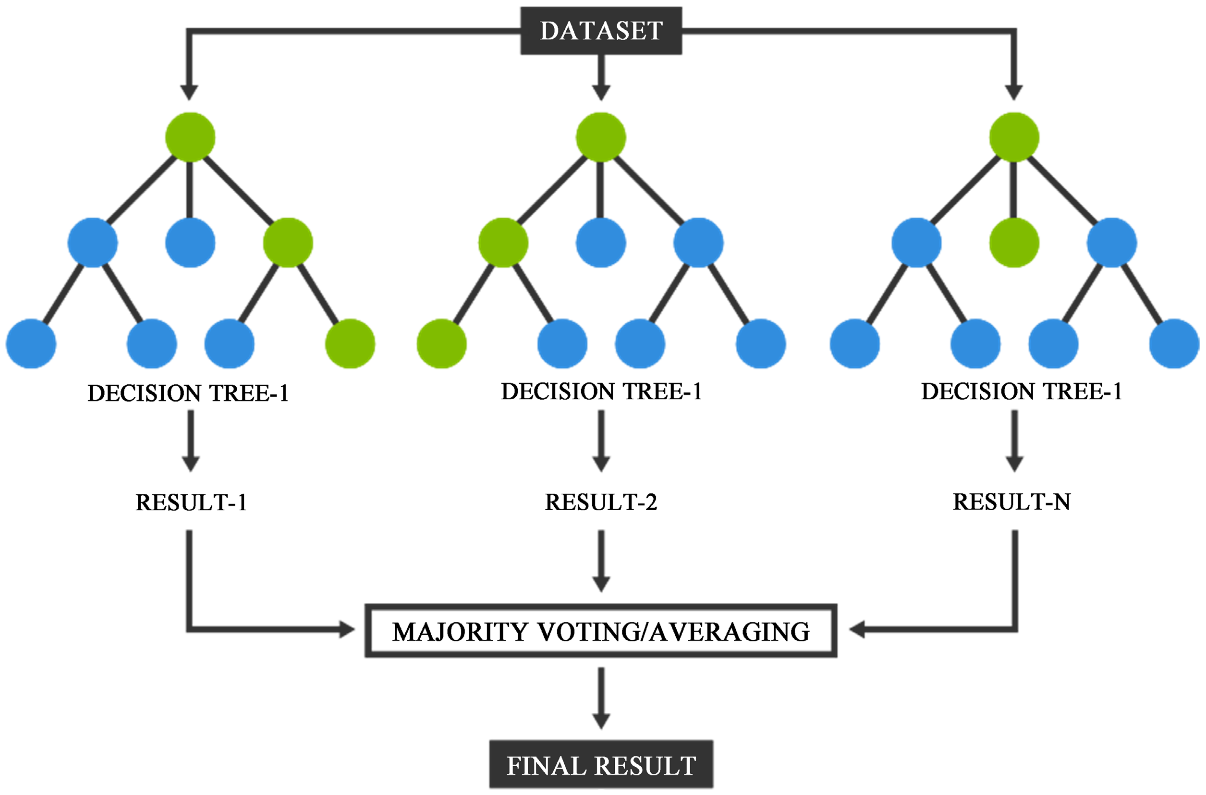

Figure 1. Scheme of the random forest mechanism.

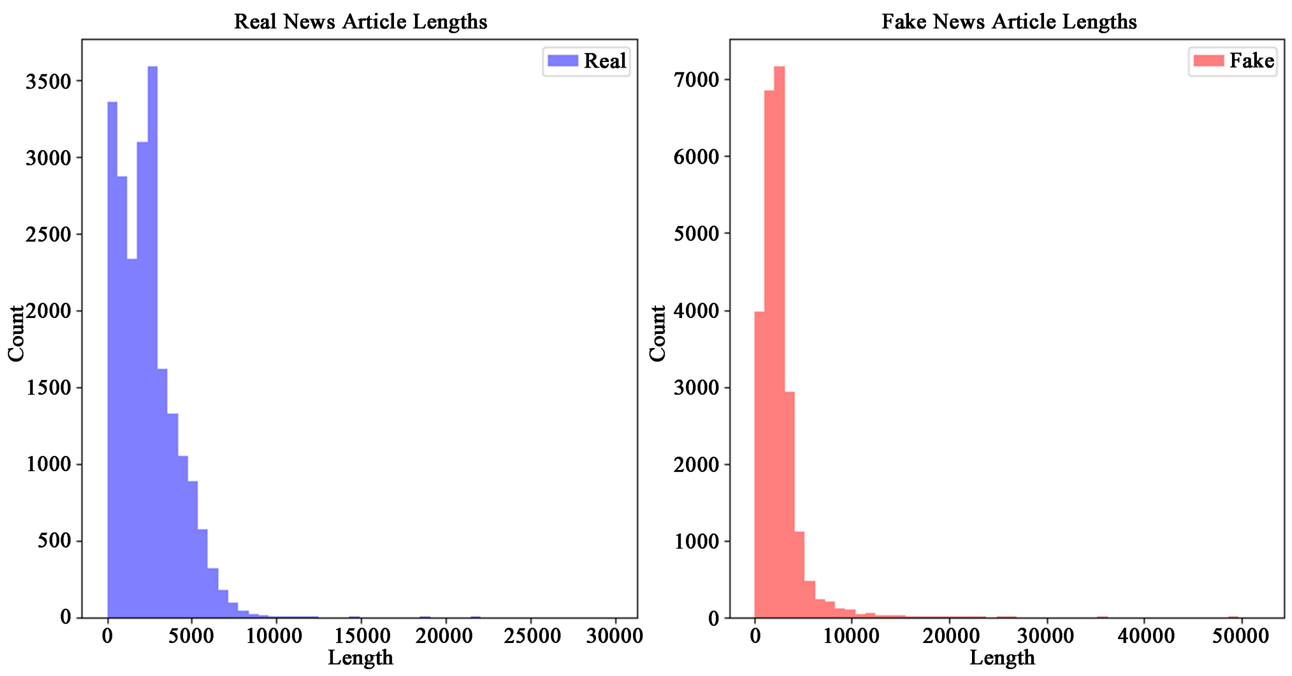

Figure 2. Histogram of the lengths of real and fake news articles.

This preprocessed text was then vectorized using CountVectorizer from scikit-learn, which converts the text into a matrix of token counts. This matrix represents the features (X) to be used for training and evaluation. The labels (y) were set to 1 for real news rows and 0 for fake news rows.

The dataset was carefully divided into features (X_train) and labels (y_train) for training, with a separate test set (X_test, y_test) reserved for final model evaluation. This division was critical to evaluate the model's ability to generalize to new, unseen data. The train and test sets were split 80/20.

Hyperparameter tuning was conducted using RandomizedSearchCV from scikit-learn, a powerful tool that combines a randomized grid search with cross-validation. This method is known for its effectiveness in navigating the vast hyperparameter space of machine learning models. The specific hyperparameters tuned included the number of trees, tree depth, minimum sample parameters, and whether to use bootstrapping, all chosen based on Random Forest best practices. The aim was to find a combination that maximizes the accuracy of the classifier while preventing overfitting.

The sequential implementation performed 10 iterations of 3-fold stratified cross-validation over the defined search spaces. This provides a robust estimate of out-of-sample model performance. Timing and accuracy results on the test set were recorded.

The parallel implementation utilized the n_jobs = −1 parameter to leverage all available CPU cores during the RandomizedSearchCV. This allows each CV split to be processed in parallel instead of sequentially, accelerating the overall tuning procedure. The same 10×3 CV evaluation was performed, along with a timing and accuracy assessment.

Criteria for Performance Evaluation

Accuracy measures the percentage of correct predictions out of all predictions made. It gives an overall view of how often the model is right or wrong. Precision focuses specifically on the positive predictions, calculating the proportion of true positives among all positive predictions. This shows how good the model is at not falsely predicting positive cases. Recall calculates the percentage of actual positive cases that are correctly predicted positive. So, it measures how many real examples of interest the model captures. The F1-score balances precision and recall by calculating their harmonic mean, providing a single metric that reflects both aspects. This provides a combined metric that avoids issues with extremes in either precision or recall alone.

The mathematical formulas for calculating these metrics are:

$$\text{Accuracy} = \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{FP} + \text{FN} + \text{TN})}. \tag{1}$$

$$\text{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})}. \tag{2}$$

$$\text{F1-score} = \frac{2 \times \text{precision} \times \text{recall}}{(\text{precision} + \text{recall})}. \tag{3}$$

By reporting multiple metrics like these, we can thoroughly analyze different aspects of the fake news detection system's performance. Accuracy shows overall correctness, precision evaluates false alarms, recall measures finding true events, and F1-score balances the two. Together they will indicate how well the system can reliably detect real fake news articles.

4. Result

This study aimed to assess the efficiency gains achieved by parallelizing the hyperparameter tuning process in a Random Forest (RF) classifier. Recognizing that hyperparameter tuning is a computationally intensive task in machine learning, the study focused on leveraging parallel processing to achieve significant speed improvements. This approach is crucial in scenarios where quick model tuning is essential, such as in real-time data analysis or large-scale machine learning projects.

The application of the Random Forest algorithm to our dataset involved two distinct approaches: sequential code execution and parallel code execution. Each method provided unique insights and highlighted the robustness and adaptability of the model in classifying news articles. The performance outcomes of the sequential implementation are summarized in **Table 1**, including accuracy, confusion matrix, and other key classifier evaluation metrics.

The sequential approach involved iterating over the hyperparameter space in a step-by-step manner. This method, while thorough, was time-consuming, as reflected in the CPU and wall time. The high precision, recall, and F1-score suggest the model's strong capability to correctly classify news articles as fake or real.

Table 1. Sequential code execution.

Metric	Value
Best Hyperparameters	n_estimators: 100, min_samples_split: 2, min_samples_leaf: 2, max_depth: None, bootstrap: False
Test Accuracy	99.15%
Confusion Matrix	TP: 4599, FP: 51, FN: 25, TN: 4305
Classification Report	Precision: 99%, Recall: 99%, F1-Score: 99%
Feature Importance	Majority contributing negligible influence
Execution Time	CPU: 21 min 51 s, Wall: 25 min 31 s

Table 2. Parallel code execution.

Metric	Value
Best Hyperparameters	n_estimators: 100, min_samples_split: 5, min_samples_leaf: 1, max_depth: None, bootstrap: True
Test Accuracy	99.09%
Confusion Matrix	TP: 4613, FP: 37, FN: 45, TN: 4285
Classification Report	Precision: 99%, Recall: 99%, F1-Score: 99%
Feature Importance	Similar detailed analysis as sequential
Execution Time	CPU: 6 min 7 s, Wall: 16 min 38 s

In contrast, the parallel code execution utilized the n_jobs = −1 parameter, allowing the process to leverage all available CPU cores (**Table 2**). This parallelization significantly reduced the execution time, demonstrating a more efficient use of computational resources. The slight variance in hyperparameters between the two methods indicates the nuances in model tuning, yet both approaches maintained high accuracy.

The comparative analysis of sequential and parallel implementations showcased the substantial performance enhancements achieved through parallelization. The parallel approach, by significantly reducing computational time, proved highly effective for large-scale machine learning tasks, without compromising on model accuracy. This study underscores the value of parallel processing in the realm of machine learning, particularly for complex tasks such as hyperparameter tuning in Random Forest classifiers.

5. Conclusion

This analysis demonstrated the significant performance gains achieved by parallelizing hyperparameter tuning for a Random Forest classifier, with over 5× faster CPU times and 3× faster total run times compared to sequential execution. However, a slight drop in test accuracy was observed, from 99.26% with the sequential approach to 99.15% using parallelism. Still, the substantial time savings allow far more extensive hyperparameter exploration within reasonable timeframes. The small reduction in accuracy seems an acceptable trade-off for the ability to evaluate orders of magnitude more configurations. Overall, parallelizing machine learning workflows like tuning proves crucial for efficiency, outweighing minor decreases in model performance, enabling more complex models and algorithms to be feasibly trained and deployed. Further research should investigate this accuracy/efficiency trade-off across different models, tasks, datasets, tuning techniques, and hardware architectures. Additional analysis into specialized parallel algorithms for ensemble methods like Random Forests could help close the performance gap with sequential approaches. Still, this project clearly demonstrated the tremendous potential of leveraging parallelism to accelerate machine learning pipelines without severely impacting generalization, allowing models to rapidly iterate and take full advantage of available data.

Acknowledgements

The authors sincerely thank the College of Computer Science and Information Technology at Imam Abdulrahman bin Faisal University for providing the computational resources that enabled and supported this research into parallel hyperparameter optimization techniques. We express our gratitude to our advisor, Dr. Yasir Alguwaifli, for his invaluable guidance, feedback, and insights throughout the development of this project. We also acknowledge the authors of the open-source scikit-learn machine learning library, which provided accessible tools for efficiently implementing parallelism in our analysis. Finally, we recognize Kaggle and the community data scientists who compiled and shared the fake news detection dataset used for evaluating and comparing sequential versus parallel tuning processes. This collective support was essential to the completion of this work.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Upadhyaya, S.R. (2022) Parallel Approaches to Machine Learning—A Comprehensive Survey. *IEEE Transactions on Parallel and Distributed Systems*, **73**, 284-292. https://doi.org/10.1016/j.jpdc.2012.11.001

[2] Memeti, S., *et al.* (2023) Using Meta-Heuristics and Machine Learning for Software Optimization of Parallel Computing Systems: A Systematic Literature Review. *IEEE Transactions on Parallel and Distributed Systems*, **101**, 893-936. https://doi.org/10.1007/s00607-018-0614-9

[3] Hutter, F., *et al.* (2019) Automated Hyperparameter Optimization: Methods, Challenges, and Applications. *IEEE Access*, **7**, 12937-12955.

[4] Talbi, E.G. (2023) Parallel Metaheuristics: Recent Advances and New Trends. 2nd Edition. Wiley, Hoboken, NJ.

[5] Topcuoglu, H., *et al.* (2022) Edge Scheduling of Deep Neural Networks for Real-Time Inference. *IEEE Transactions on Industrial Informatics*, **18**, 4986-4995.

[6] Zhang, L., *et al.* (2021) Data Parallel Distributed Training for Sequence-to-Sequence Models. *Proceedings of the Institute of Electrical and Electronics Engineers International Conference on Big Data*, Orlando, FL, December 2021, 21-30.

[7] Song, H., *et al.* (2019) Hyperparameter Optimization of Deep Neural Networks Using Non-Repetitive Directed Walk. *Proceedings of the Institute of Electrical and Electronics Engineers/CVF International Conference on Computer Vision*, Seoul, October 2019, 4815-4824.

[8] Abadi, M., *et al.* (2016) TensorFlow: A System for Large-Scale Machine Learning. *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, Savannah, GA, November 2016, 265-283.

[9] Paszke, A., *et al.* (2017) Automatic Differentiation in PyTorch. 31*st Conference on Neural Information Processing Systems* (*NIPS* 2017), Long Beach, CA, USA.

[10] Pedregosa, F., *et al.* (2022) Scikit-Learn in 2022. arXiv preprint arXiv:2205.09532.

[11] Franceschi, L., *et al.* (2018) On the Benefits of Parallelism in Derivative-Free Optimization for Machine Learning. *Proceedings of the Institute of Electrical and Electronics Engineers International Conference on Acoustics, Speech, and Signal Processing*, Calgary, AB, April 2018, 3244-3248.

[12] Wang, W., Yuan, Y. and Tan, Q. (2018) Recent Advances in Random Forests for Machine Learning. 2018 15*th International Conference on Control, Automation, Robotics and Vision* (*ICARCV*), Singapore, November 2018, 1-6.

[13] Zhou, Z. and Feng, J. (2019) Deep Forest. *National Science Review*, **6**, 74-86. https://doi.org/10.1093/nsr/nwy108

[14] Aslam, N., *et al.* (2022) Anomaly Detection Using Explainable Random Forest for the Prediction of Undesirable Events in Oil Wells. *Applied Computational Intelligence and Soft Computing*, **2022**, Article ID: 1558381. https://doi.org/10.1155/2022/1558381

[15] Wu, J., Chen, J., Xiong, H. and Ye, J. (2018) Accelerating Random Forest Training on a CPU-FPGA Heterogeneous Platform. *International Symposium on Applied Reconfigurable Computing*, Bologna, Italy, April 2018, 89-101.

[16] (n.d.) What Is a Random Forest? https://www.tibco.com/reference-center/what-is-a-random-forest

[17] Bisaillon, C. (2022) Fake and Real News Dataset. Kaggle.