Sciforce

Journal of Data Science and Information Technology

Journal homepage: www.sciforce.org
ISSN : 2998-3592

Open Access Review Article

Ada Boost Regression-Based Modelling of Heap Behaviour for Predictable Performance in Java Applications

Tirumala Rao Gundala*

Consulting Technical Manager & Performance Architect, Oracle, United States

Abstract

This research addresses the critical challenge in modelling heap behaviour for predictable performance in Java applications using machine learning approaches. Since heap memory management significantly impacts application responsiveness and performance, understanding the complex relationships between heap load, garbage collection pause times, throughput, and memory usage becomes essential for optimization. Through extensive statistical analysis, this study establishes strong correlations between parameters and employs Ada Boost Regression to predict three key performance metrics across 100 observations. This method demonstrates exceptional predictive accuracy, achieving R^2 values of 0.987, 0.955, and 0.977 for GC pause time, throughput, and memory usage respectively on the training data, with robust generalization to the test data (R^2 values of 0.956, 0.880, and 0.937). The results reveal near-perfect positive correlations between heap load, GC pause times, and memory usage, while throughput exhibits a strong negative correlation, illustrating the fundamental JVM memory management trade-offs. This research provides practitioners with a reliable predictive tool for ensuring stable operations in production environments where performance optimization, capacity planning, and efficient memory management are paramount.

Keywords: Heap memory management, AdaBoost regression, Garbage collection process optimization, Java Virtual Machine performance, Predictive performance modelling, Machine learning, Performance prediction

Introduction

Accurate modelling of heap evolution is crucial for optimization and verification tasks, yet previous methods generally fall short in either accuracy or efficiency, limiting their applicability in practice and hindering progress in several research areas. This work presents a general heap analysis approach that aims to provide comprehensive heap insights for real-world programs while remaining computationally feasible. By leveraging targeted design heuristics, it maintains speed and accuracy on typical code patterns, selectively relaxing accuracy in rare cases to improve overall performance. This technique fully supports essential Java language features and reliably generates precise information suitable for various optimization applications [1]. This research proposes a parametric technique for estimating the heap memory usage of imperative programs such as Java. This approach prevents potential memory exhaustion errors by calculating symbolic polynomial approximations of the dynamic memory required for the safe execution of a method.This technique accounts for both object allocations and data transfers occurring within the main function and its nested calls. By employing constrained memory management, it optimizes peak memory usage by organizing objects into regions that align with the function's execution timeline.

The underlying challenge is formulated as a parameter-dependent polynomial optimization problem, which is then solved using the Bernstein basis method. The resulting tool has been evaluated and used in several benchmark programs [2]. Dynamic memory allocation plays a crucial role in modern object-oriented software. However, as processor speeds have grown faster than memory access times, and as applications grow in size and memory requirements, physical memory performance has become increasingly critical. Without this, the slow nature of memory compared to processing would increasingly degrade program performance. This research paper explores the use of profile-driven optimization to predict the reference and lifetime patterns of heap objects. The goal is to improve spatial locality and reduce page faults, thereby enhancing performance, especially in memory-constrained environments [3]. Based on the complexity of the problem, we propose a framework for modelling and predicting the execution time of scientific applications written in Java.

The primary objective of this research is to achieve an efficient workload balance for distributed Java applications running in a grid computing environment. We introduce a distinction between predictable and unpredictable processing points within Java applications and explain our methodology in detail for modelling and predicting program performance [4]. The Real-Time Java Expert Group has developed a specification to extend the benefits of Java, particularly its developer-friendly features, to the field of real-time systems. These benefits are especially crucial for distributed

Received date: November 03, 2025 **Accepted date:** November 25, 2025; **Published date:** December 05, 2025

***Corresponding Author**: *Gundala, Tirumala Rao, Consulting Technical Manager & Performance Architect, Oracle, United States; E- mail: Tirumalagundala7@gmail.com*

Citation: Gundala, Tirumala Rao (2025). Ada Boost Regression-Based Modelling of Heap Behaviour for Predictable Performance in Java Applications. Journal of Data Science and Information Technology, 2(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v2i2.275

real-time systems, where software development has traditionally been considered complex, labour-intensive, and prone to errors [5]. Java is increasingly becoming the preferred language for software development in various embedded environments. The use of Java's micro editions for creating embedded software solutions is a prime example of this trend [6]. In recent years, Java has emerged as a robust and well-established platform. Its inherent portability has cemented its popularity for enterprise-level applications and for implementing standard components such as middleware and business logic [7]. Determining the varying parameters proved to be significantly more challenging than initially anticipated. To address this, we developed a dedicated framework within the Java programs for this study to calculate these parameters [8]. Each implementation is designed to meet specific performance requirements, leading to unique design trade-offs. For example, Java's Array List uses less memory than Linked List, but incurs a higher asymptotic cost for inserting or deleting elements at any position in the list [9]. The purpose of this article is to analyse the memory characteristics of key Java benchmarks used for evaluating JVMs and JIT compilers. It aims to provide structured, multi-level insights into these workloads to assist computer architects and JVM component developers in their design processes [10]. Mechanisms such as automatic garbage collection and dynamic class loading are often considered problematic in time-sensitive or performance-critical environments. Additionally, since the introduction of the Java Virtual Machine, a significant number of security vulnerabilities have been discovered in byte code verifiers and just-in-time compilers.

All these issues combined make Java and its related technologies unsuitable for high-reliability systems [11]. It has been proven that the performance characteristics of Java applications differ significantly from the performance of general-purpose tasks. This necessitates an in-depth study of Java-specific functionalities at the microarchitecture level. Conducting performance analyses of Java tasks in real-world environments supports optimization efforts and reveals critical issues that need to be considered in the design and development of future systems designed to run Java [12]. This automation is facilitated by garbage collection, which is a background process that reclaims memory from objects that are no longer in use. Integrating garbage collection into the Java HotSpot Virtual Machine represents a major advancement, greatly simplifying development for large-scale applications and improving runtime stability [13]. The platform examined in this study is Oracle Web Logic Server, a leading Java EE application server in the market. Our method for extracting its model relies on performance monitoring data collected using industry-standard business monitoring tools designed for this platform [15].

Materials and Method

Heap Load: Heap leaching, in the mining industry, refers to the process of loading ore onto a heap leaching pad. Typically, crushed and agglomerated processed ore is systematically stacked in layers using heavy equipment such as conveyors or trucks. This planned stacking creates a permeable, uniform heap, allowing the leaching solution to penetrate evenly for optimal metal recovery. This method is crucial to the efficiency of heap leaching processes used to extract metals such as gold, copper, and uranium.

GC Pause Time (ms): GC pause time is the total duration, in milliseconds, during which all application threads are stopped during a garbage collection (GC) event in managed memory environments such as the JVM or .NET CLR. This "stop-the-world" pause allows the collector to safely reclaim unused memory without interference from the running program threads. This is a critical performance metric, as excessive pause times directly impact application responsiveness, latency, and throughput, making its reduction a key objective in GC tuning.

Throughput ops (per sec): Throughput (operations/second) is a key performance metric that measures the total number of operations a system can successfully complete per second. It quantifies overall work capacity and efficiency. In contexts such as databases, APIs, or servers, an 'operation' is a defined unit of work, such as a transaction, query, or request. Higher throughput indicates robust processing capability under heavy workloads, but to fully assess system performance and user experience, it must be balanced with other metrics such as latency and error rates.

Memory Utilization percent: Memory usage (percentage) measures the ratio of the total physical or virtual RAM currently in use by processes and the operating system on a system. It is a key health metric indicating how much memory is available for new tasks. High usage can indicate efficient resource utilization, but when sustained near 100%, it often leads to performance degradation. This occurs because the system increasingly relies on slower disk-based swapping, causing bottlenecks. Monitoring this percentage is crucial for capacity planning, preventing out-of-memory errors, and ensuring application stability.

Instructions for machine learning

Ada Boost Regression: Boosting is a very effective and widely used technique in supervised learning, which works by successively combining a series of simple, weak models to form a single, highly accurate predictive model. This process operates in a greedy manner, with every new student concentrating on the mistakes made by their predecessors. Although its initial conception was for classification, the method has proven to be remarkably versatile across a variety of problem types. A significant and fascinating area of research in this field involves systematically combining boosting algorithms, such as Ada Boost, with well-defined optimization problems. This line of inquiry seeks to provide a deeper theoretical understanding of their computational behaviour and guarantees. As a result, much of the scholarly effort has focused on two primary optimization objectives: maximizing the classification margin and minimizing the exponential loss function, which are central to the algorithm's performance and theoretical foundation.

Results and Discussions

This dataset captures key performance metrics of a Java application's heap memory under varying loads. It records four interrelated variables: heap load (a measure of memory pressure), GC pause time in milliseconds (the time the application is paused for garbage collection), throughput in operations per second, and memory usage as a percentage. The data reveals clear patterns: as the heap load increases, garbage collection pauses generally lengthen, throughput decreases, and memory usage increases, illustrating the trade-offs inherent in JVM memory management and tuning for optimal application performance.

Citation: Gundala, Tirumala Rao (2025). Ada Boost Regression-Based Modelling of Heap Behaviour for Predictable Performance in Java Applications. Journal of Data Science and Information Technology, 2(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v2i2.275

Table 1. Descriptive Statistics				
	Heap Load	GC Pause Time (ms)	Throughput ops (per sec)	Memory Utilization percent
count	100	100	100	100
mean	0.576145	119.132	827.8941	69.147113
std	0.237992	28.23808	74.63821	12.245263
min	0.204418	73.7447	696.4571	48.065552
25%	0.354561	95.04684	766.0708	57.042893
50%	0.571314	116.8335	833.4043	68.599603
75%	0.784162	144.6639	889.3874	79.905797
max	0.98951	172.6503	973.9838	92.257718

The descriptive statistics provided in Table 1 for the four key performance parameters establish a baseline context for model interaction. For a predictive model interaction, such as Ada Boost Regression, to be effective that is, to perform reliably across different data scenarios it must handle the inherent variability within these parameters. The statistics reveal considerable ranges, with GC pause time varying from 73.7 to 172.7 ms and throughput spanning from 696.5 to 974.0 ops/sec. The mean and median values are generally aligned, suggesting reasonably symmetrical distributions. A model trained on this dataset should generalize across these distributions, ensuring that its predictions are considered robust and valid for data points across the entire observed spectrum.

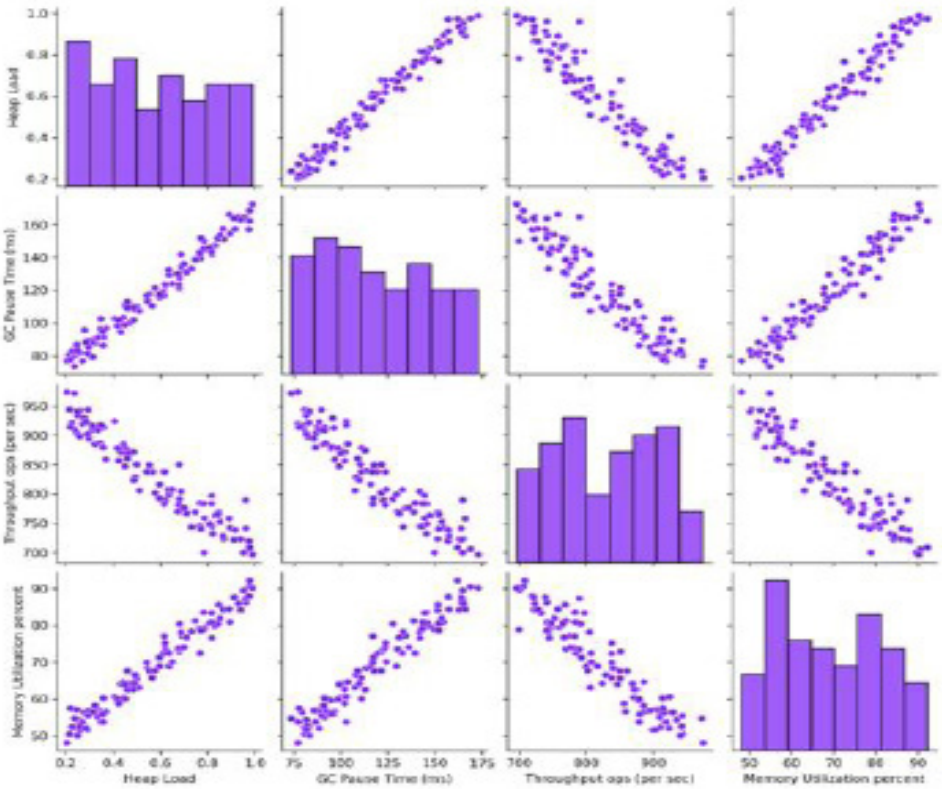

Figure 1: Scatter plot of different machine learning-based Modelling Heap Behaviour for Predictable Performance in Java Applications

Figure 1 illustrates the relationships between heap load, GC pause time, throughput, and memory usage in machine learning-based heap behaviour modelling. Strong positive correlations appear between heap loads, GC pauses, and memory usage, while throughput shows a clear inverse trend, highlighting crucial trade-offs for predictable Java application performance under varying workloads.

Citation: Gundala, Tirumala Rao (2025). Ada Boost Regression-Based Modelling of Heap Behaviour for Predictable Performance in Java Applications. Journal of Data Science and Information Technology, 2(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v2i2.275

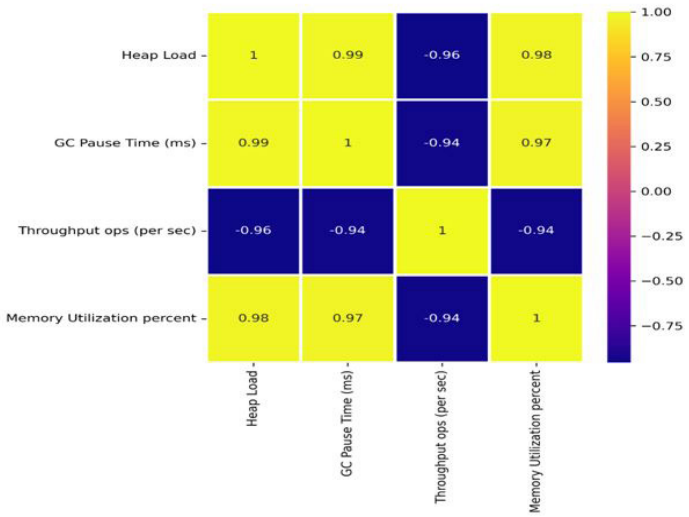

Figure 2: Heat map of the relationship between process parameters and outcomes

Figure 2 provides a heat map illustrating the strong correlations between process parameters and outcomes. Heap load and GC pause time show a nearly perfect positive correlation with memory usage, while throughput is strongly negatively correlated, indicating performance degradation due to increased memory pressure and garbage collection overhead.

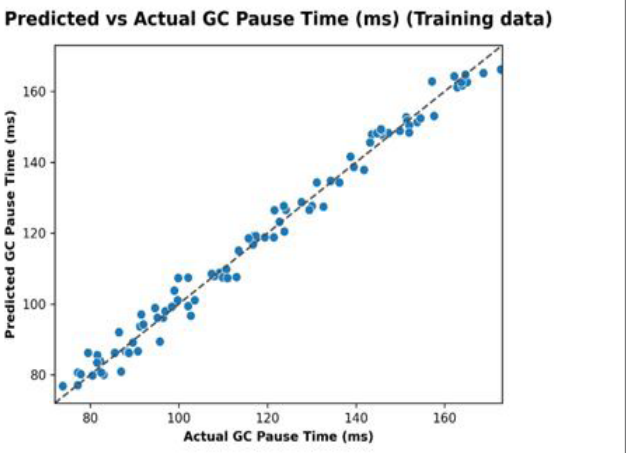

Figure 3: Ada Boost Regression on GC Pause Time (ms): training data

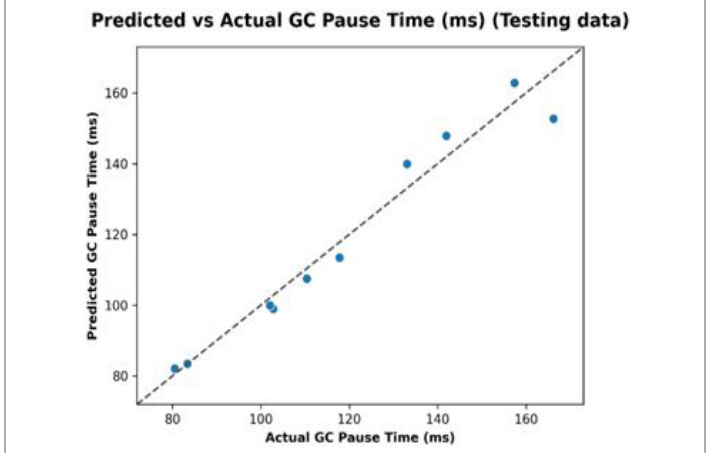

Figure 4: Ada Boost Regression on GC Pause Time (ms): testing data

Figure 3 demonstrates the Ada Boost regression performance on the training data, showing that the predicted GC pause times closely match the actual values. The tight clustering around the diagonal indicates high model accuracy, effective learning of underlying patterns, and a strong suitability for modelling garbage collection behaviour in Java applications.

Figure 4 illustrates the Ada Boost regression performance on the test data, where the predicted GC pause times closely follow the actual measurements. The near-diagonal distribution indicates good generalization beyond the training data, minimal over fitting, and a reliable predictive capability for estimating garbage collection pauses under unseen Java application workloads.

Table 2. Performance Metrics of Ada Boost Regression on GC Pause Time (ms) (Training Data and Testing Data)

Parameter	Data	Symbol	Model	R2	EVS	MSE	RMSE	MAE	MaxError	MSLE	MedAE
GC Pause Time (ms)	Train	ABR	AdaBoost Regression	0.98709	0.98715	10.20513	3.19455	2.65478	7.51396	0.00087	2.42008
	Test	ABR	AdaBoost Regression	0.95623	0.95678	34.10774	5.84018	4.64398	13.43514	0.00166	4.04612

Citation: Gundala, Tirumala Rao (2025). Ada Boost Regression-Based Modelling of Heap Behaviour for Predictable Performance in Java Applications. Journal of Data Science and Information Technology, 2(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v2i2.275

The Ada Boost regression model in Table 2 demonstrates superior performance in predicting GC pause time and effectively generalizes from the training data to the test data. On the training set, it achieves near-perfect performance with an R^2 value of 0.98709 and an RMSE of 3.19 milliseconds. While the test performance is lower as expected, it remains robust with an R^2 value of 0.95623 and an RMSE of 5.84 milliseconds. This minimal performance drop confirms the model's effectiveness and its ability to maintain accuracy and reliability when applied to new data. This stability is crucial for reliable predictions in various real-world computing environments.

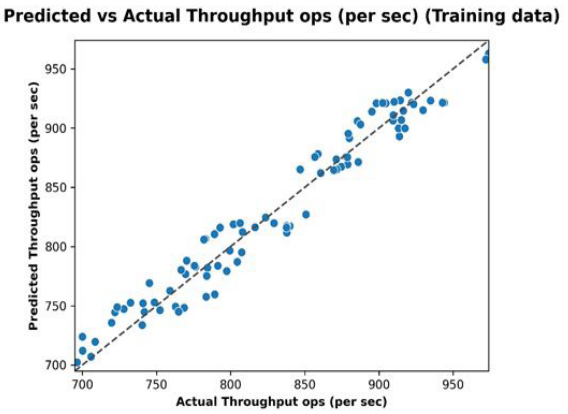

Figure 5: Ada Boost Regression on Throughput ops (per sec): training data

Figure 5 shows the Ada Boost regression results for the training data performance, where the predicted values closely match the actual measurements. The strong alignment along the diagonal reflects high prediction accuracy, effective capture of system performance trends, and the suitability of the model for learning performance behaviour in Java applications.

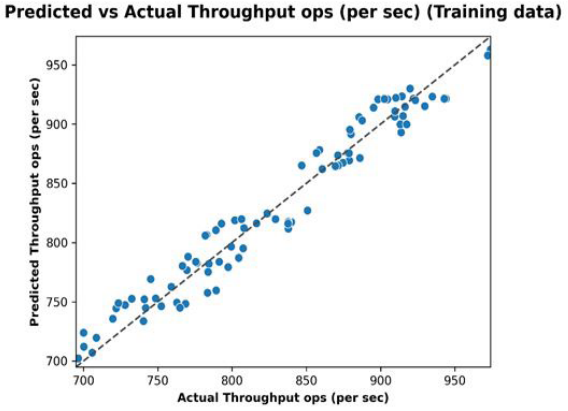

Figure 6: Ada Boost Regression on Throughput ops (per sec): testing data data

Figure 6 illustrates the Ada Boost regression performance in testing the data for performance prediction. The predicted values generally follow the ideal diagonal line, indicating good generalization. Small deviations at higher performance levels indicate limited prediction error, although the overall results confirm the reliability of the model in estimating system performance under unseen workload conditions.

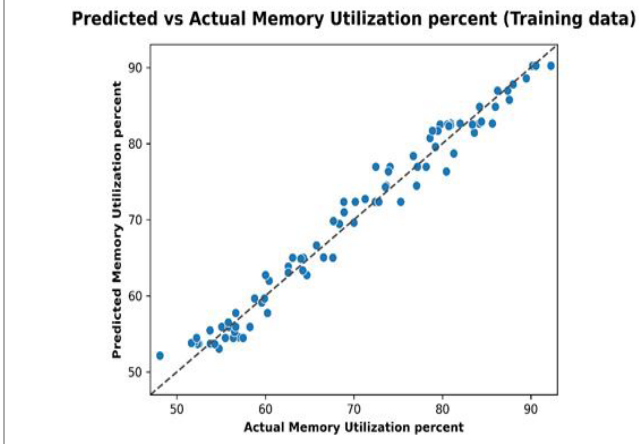

Figure 7: Ada Boost Regression on Memory Utilization percent: training data

Figure 7 shows the Ada Boost regression results for memory usage on the training data. The predicted values closely match the actual measurements along the diagonal line, indicating the high accuracy of the model, robust learning of memory usage patterns, and the effective suitability of this approach for modelling memory behaviour in Java applications.

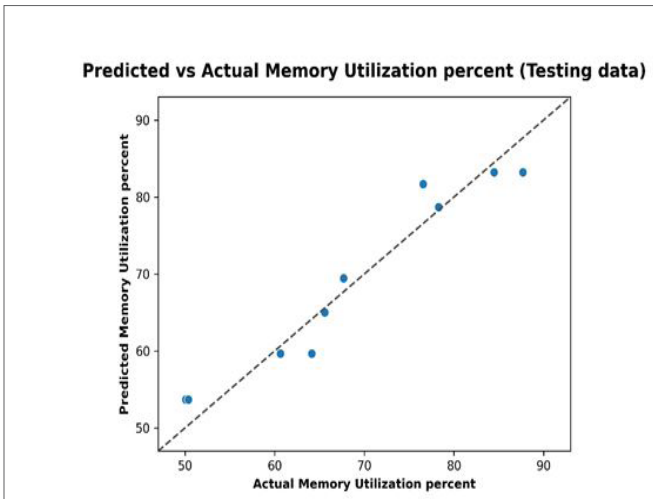

Figure 8: Ada Boost Regression on Memory Utilization percent: testing data

Figure 8 illustrates the Ada Boost regression performance on the test data for memory usage prediction. The predicted values being very close to the ideal diagonal line indicates good generalization

Citation: Gundala, Tirumala Rao (2025). Ada Boost Regression-Based Modelling of Heap Behaviour for Predictable Performance in Java Applications. Journal of Data Science and Information Technology, 2(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v2i2.275

and limited error. The small scatter at higher usage levels suggests low variance, and the overall results confirm reliable predictive capability for unseen workloads.

Table 3. Performance Metrics of Ada Boost Regression on Throughput ops (per sec) (Training Data and Testing Data)

Parameter	Data	Symbol	Model	R2	EVS	MSE	RMSE	MAE	MaxError	MSLE	MedAE
Throughput ops (per sec)	Train	ABR	AdaBoost Regression	0.95491	0.95529	243.65400	15.60942	13.47391	29.56912	0.00037	13.68261
	Test	ABR	AdaBoost Regression	0.88046	0.89548	768.79570	27.72717	23.94814	47.81094	0.00110	27.28496

The Ada Boost Regression model in Table 3 demonstrates robust performance in predicting throughput (operations per second). It effectively applies the patterns learned during training to unseen test data. Its performance on the training set is excellent; an R^2 value of 0.95491 and an RMSE of 15.61 ops/sec indicate a precise fit to the data. While an expected drop in performance is observed on the test set, the model maintains strong predictive capabilities with an R^2 value of 0.88046 and an RMSE of 27.73 ops/sec. This consistent, albeit slightly lower, performance confirms the model's robustness that is, its reliable operation and generalization ability across different data subsets. This is crucial for stable deployment in production environments.

Table 4. Performance Metrics of Ada Boost Regression on Memory Utilization percent (Training Data and Testing Data)

Parameter	Data	Symbol	Model	R2	EVS	MSE	RMSE	MAE	MaxError	MSLE	MedAE
Memory Utilization percent	Train	ABR	AdaBoost Regression	0.97715	0.97737	3.37688	1.83763	1.55226	4.49083	0.00075	1.49328
	Test	ABR	AdaBoost Regression	0.93719	0.93762	9.68710	3.11241	2.60147	5.16216	0.00217	2.55767

The Ada Boost Regression model in Table 4 demonstrates robust performance in predicting memory usage. This is evidenced by its strong performance metrics on both the training and testing datasets, indicating reliable generalization. On the training data, the model achieves an almost perfect fit with an R^2 value of 0.97715 and a low RMSE of 1.84%, showing its effective learning of the underlying patterns. Crucially, its performance remains high on unseen test data, with an R^2 value of 0.93719 and an RMSE of 3.11%. This minimal drop in performance confirms the model's robustness its ability to maintain predictive accuracy and stability when applied to new, independent data, which is essential for deployment in diverse operational environments.

Conclusion

Based on the comprehensive analysis presented in this study, the Ada Boost regression algorithm demonstrates exceptional capability in modelling heap behaviour for predictable performance in Java applications. This research successfully establishes strong correlations between critical performance parameters, including heap load, garbage collection pause time, throughput, and memory usage, providing valuable insights into the inherent complex trade-offs in JVM memory management. The model's performance across the three target variables confirms its effectiveness as a predictive tool for Java application optimization. The Ada Boost regression model exhibits significant accuracy and generalization capabilities across different performance metrics. For GC pause time prediction, the model achieves excellent results with R^2 values of 0.987 on the training data and 0.956 on the test data, demonstrating minimal over fitting and strong predictive power. Similarly, throughput prediction maintains robust performance with R^2 values of 0.955 and 0.880 for the training and test datasets respectively, while memory usage prediction shows superior reliability with R^2 values of 0.977 and 0.937. These consistent performance metrics across training and testing phases confirm the model's ability to effectively generalize to unseen workloads, making it highly relevant for real-world application in diverse operational environments.

The strong correlations revealed through heat map analysis provide crucial insights for performance tuning and capacity planning in Java applications. The near-perfect positive correlation between heap load and both GC pause time and memory usage, combined with a strong negative correlation with performance, highlights the crucial balance required in memory management strategies. This research significantly contributes to the growing body of knowledge on predictive performance modelling in Java environments, providing practitioners with a reliable tool to anticipate system behaviour under various workloads. The proven strength and accuracy of Ada Boost Regression in simultaneously predicting multiple performance parameters make it an invaluable asset for optimizing Java application performance, minimizing downtime, and ensuring stable operations in production systems where memory management and garbage collection efficiency are critical concerns.

References

1. Kapur, D., &Marron, M. (2008). Modeling the heap: a practical approach. http://www.software.imdea.org/~marron/papers/marron_doctoral.pdf
2. Braberman, V., Fernández, F., Garbervetsky, D., &Yovine, S. (2008). Parametric prediction of heap memory requirements. International Symposium on Memory Management, 141–150. https://doi.org/10.1145/1375634.1375655
3. Seidl, M. L., & Zorn, B. G. (1998). Segregating heap objects by reference behavior and lifetime. 33(11), 12–23. https://doi.org/10.1145/291006.291012
4. Perikala. K (2024). Architecting Retail-Scale Product Knowledge Graph Systems. International Journal of Artificial intelligence and Machine Learning, 2(3), 1-6. doi: https://doi.org/10.55124/jaim.v2i3.292
5. Zhang, Rui, ZoranBudimlic, and Ken Kennedy. "Performance modeling and prediction for scientific Java applications." In 2006 IEEE International Symposium on Performance Analysis of Systems and Software, pp. 199-210. IEEE, 2006.
6. Raman, Krishna, Yue Zhang, Mark Panahi, Juan A. Colmenares, and Raymond Klefstad. "Patterns and tools for achieving predictability and performance with real time Java." In 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05), pp. 247-253. IEEE, 2005.

Citation: Gundala, Tirumala Rao (2025). Ada Boost Regression-Based Modelling of Heap Behaviour for Predictable Performance in Java Applications. Journal of Data Science and Information Technology, 2(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v2i2.275

7. PK Kanumarlapudi. (2023). Multi-Objective Optimization of Healthcare Service Parameters Using GRA-Based Genetic Algorithms. J Comp Sci Appl Inform Technol. 8(1): 1-7.

8. Kiran Kumar Mandula Samuel, Venkata Pavan Kumar Aka. (2023). AI for Special Education Closing Gaps in Inclusive Learning Using MOORA Method. International Journal of Computer Engineering and Technology (IJCET), 14(1), 249-267.

9. Chen, Guilin, M. Kandemir, Narayanan Vijaykrishnan, AnandSivasubramaniam, and Mary Jane Irwin. "Analyzing heap error behavior in embedded JVM environments." In Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pp. 230-235. 2004.

10. Harkema, Marcel, Dick Quartel, B. M. M. Gijsen, and Robert D. van der Mei. "Performance monitoring of Java applications." In Proceedings of the 3rd international workshop on Software and performance, pp. 114-127. 2002.

11. Dufour, Bruno, KarelDriesen, Laurie Hendren, and Clark Verbrugge. "Dynamic metrics for Java." In Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications, pp. 149-168. 2003.

12. Costa, Diego, ArturAndrzejak, Janos Seboek, and David Lo. "Empirical study of usage and performance of java collections." In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, pp. 389-400. 2017.

13. Perikala. K. (2025). Cloud-Native NoSQL Foundations for Large-Scale Generative AI Platforms. International Journal of Cloud Computing and Supply Chain Management, 1(3), 1-6. doi: https://doi.org/10.55124/ijccscm.v1i3.248

14. PK Kanumarlapudi. "Improving Data Market Implementation Using Gray Relational Analysis in Decentralized Environments" Journal of Artificial intelligence and Machine Learning., 2024, vol. 2, no. 1, pp. 1–7. doi: https://dx.doi.org/10.55124/ jaim.v2i1.271

15. Shuf, Yefim, Mauricio J. Serrano, Manish Gupta, and Jaswinder Pal Singh. "Characterizing the memory behavior of Java workloads: A structured view and opportunities for optimizations." In Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pp. 194-205. 2001.

16. Kwon, Jagun, Andy Wellings, and Steve King. "Ravenscar-Java: A high integrity profile for real-time Java." In Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande, pp. 131-140. 2002.

17. Rao, Pradeep, and Kazuaki Murakami. "Empirical performance models for Java workloads." In International Conference on Architecture of Computing Systems, pp. 219-232. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

18. Gurubasannavar. S D, Sunku. R, Dandasi. V V. (2025). Selecting an Extract, Transform, and Load (ETL) Software Solution: A Comprehensive Evaluation and Comparison . International Journal of Cloud Computing and Supply Chain Management, 1(3), 1-7. doi: https://doi.org/10.55124/ijccscm.v1i3.249

19. Devalla, Sireesha. "Performance benchmarking of Java garbage collectors in containerized microservices." Journal of Scientific and Engineering Research 7, no. 6 (2020): 326-334.

20. Venkata Pavan Kumar Aka and Kiran Kumar Mandula Samuel. (2024). Adoption of SAP FSCM – Enhancing Collections and Dispute Processes in Spain, Portugal, and UK Operations. International Journal of Information Technology and Management Information Systems (IJITMIS), 15(2), 148–161.

21. DOI: https://doi.org/10.34218/IJITMIS_15_02_012

22. Brosig, Fabian, Samuel Kounev, and Klaus Krogmann. "Automated extraction of palladio component models from running enterprise java applications." In Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools, pp. 1-10. 2009.

23. Chen, Kandemir, Vijaykrishnan, and Irwin. "PennBench: a benchmark suite for embedded Java." In 2002 IEEE International Workshop on Workload Characterization, pp. 71-80. IEEE, 2002.

24. Perikala. K (2024). Large-Scale Architecture for Retail Platforms Using Cloud-Native Big Data Systems. International Journal of Computer Science and Data Engineering, 1(3), 1–7 doi: https://dx.doi.org/10.55124/csdb.v1i3.268

25. Freund, Robert M., Paul Grigas, and Rahul Mazumder. "Adaboost and forward stagewise regression are first-order convex optimization methods." arXiv preprint arXiv: 1307.1192 (2013).

26. Zhang, Zhihua, James T. Kwok, and Dit-Yan Yeung. "Surrogate maximization/minimization algorithms for adaboost and the logistic regression model." In Proceedings of the twenty-first international conference on Machine learning, p. 117. 2004.

27. PK Kanumarlapudi. (2024) Improving Data Governance with Advanced trade-off-Based Decision Models: A Comparative Analysis of Open Data Platform Implementations in Multiple Domains. SOJ Mater Sci Eng 10(2): 1-6. DOI: 10.15226/2473-3032/10/2/00183

28. Li, Pan, ZhongjieShen, and Xingwu Zhang. "Evaluation method of degradation index based on AdaBoost regression." In Journal of Physics: Conference Series, vol. 2031, no. 1, p. 012059. IOP Publishing, 2021.

29. Wen, Jingran, Xiaoyan Zhang, Ye Xu, Zuofeng Li, and Lei Liu. "Comparison of AdaBoost and logistic regression for detecting colorectal cancer patients with synchronous liver metastasis." In 2009 International Conference on Biomedical and Pharmaceutical Engineering, pp. 1-6. IEEE, 2009.

30. Suresh Deepak Gurubasannavar, Varun Venkatesh Dandasi, Raghavendra Sunku. (2025). Enhancing Smart Home Automation with AI And Topsis-Based Decision Making. International Journal of Information Technology and Management Information Systems (IJITMIS), 16(6), 1-22.

31. Patil, Sangram, AumPatil, and Vikas M. Phalle. "Life prediction of bearing by using adaboostregressor." In Proceedings of TRIBOINDIA-2018 an International Conference on Tribology. 2018.

Citation: Gundala, Tirumala Rao (2025). Ada Boost Regression-Based Modelling of Heap Behaviour for Predictable Performance in Java Applications. Journal of Data Science and Information Technology, 2(2), 1–7 doi: https://dx.doi.org/10.55124/jdit.v2i2.275