APPLICATIONS OF
MODELLING AND SIMULATION
http://arqiipubl.com/ams eISSN 2600-8084 VOL 7, 2023, 190-200

Classification and Ensemble Machine Learning Algorithms to Predict Memory Requirements for Compute Farm Jobs

Esraa Faisal Malik[1], Khai Wah Khaw[1], XinYing Chew[2*], Alhamzah Alnoor[3] and Mariam Al Akasheh[4]

[1]School of Management, Universiti Sains Malaysia, Penang, Malaysia
[2]School of Computer Sciences, Universiti Sains Malaysia, Penang, Malaysia
[3]Management Technical College, Southern Technical University, Basrah, Iraq
[4]Department of Statistics and Business Analytics, College of Business and Economics, United Arab Emirates University, United Arab Emirates

[*]Corresponding author: xinying@usm.my

Submitted 09 September 2023, Revised 31 October 2023, Accepted 18 November 2023, Available online 03 December 2023.

Abstract: Tasks ranging from synthesis to regression are executed within a computational farm environment during chip design. This process is managed by a compute farm scheduler, which handles job scheduling based on the availability of such computational resources as central processing units, memory, and storage. The increasing complexity of chip design over the years, combined with a growing number of cores per chip, has resulted in memory-intensive applications often being executed as compute jobs. Jobs submitted with inaccurate resource-related requests, especially those concerning memory, can overload a compute farm and lead to wasted resources. This study addresses this issue by using a data science-driven, machine learning-based approach to predict the memory required for a compute job at the time of its submission. Improving the accuracy of such predictions can significantly reduce the overall wait times of jobs and enable efficient use of the compute farm to reduce the overall cost because fewer machines are required to complete a set of jobs. We explored the use of the K-nearest neighbor, random forest, and ensemble methods to this end. The proposed approach yielded an accuracy of 80% in experiments, where this demonstrates the success of predicting the memory-related requirements of compute jobs across a diverse suite of applications used in the process of chip design.

Keywords: Chip design; Compute farm scheduler; Machine learning algorithms; Memory prediction; Resource management.

1. INTRODUCTION

Research on field-programmable gate arrays (FPGAs) represents the highly competitive landscape of the semiconductor industry at present. The imperative for manufacturers to retain a competitive edge has underscored the importance of business acceleration and reducing the time-to-market (TTM) of products. Primsa's assertion in [1] shows that a short TTM empowers a business to cultivate momentum, assume the role of an industry pioneer, and proactively engage customers to secure a large market share. It is common for businesses to concurrently undertake multiple projects to sustain an advantage over rivals, but this is sometimes constrained by their limited resources. Such projects are divided into discrete devices that are subsequently further deconstructed into planned milestones. Each project typically encompasses one or more hardware devices, and the associated milestones must be reached before proceeding further. Moreover, the timelines and requirements for each project may vary based on considerations related to pathfinding and release. The concurrent development of components by engineers from a variety of divisions demands a substantial compute farm infrastructure. Effective resource management is essential for guaranteeing the prompt execution and completion of tasks, thereby facilitating the progression of the project to the critical "tapeout" phase, which is shown in Figure 1 [2]. This phase signifies the conclusive step in which the designed FPGA is shipped for manufacturing.

As third-party job scheduler, the IBM Load Sharing Facility (LSF®) is used to manage job submissions. The LSF® can distribute tasks across diverse IT resources to sustain a shared, scalable, and fault-tolerant infrastructure [3]. Before executing a remote Unix batch job within the LSF compute farm, the scheduler is tasked with identifying a suitable machine that is capable of fulfilling its hardware requirements, where this generally encompasses the CPU (slots), memory, and storage. Jobs might over-request or under-request any of the requisite resources. This study focuses on correcting invalid memory allocations. For instance, if a job requests 16 GB of memory and is assigned to a server with 64 GB, but then attempts to use up to 100 GB of memory, the server may crash owing to insufficient memory. Thus, ensuring an appropriate alignment between

requests for resources and their actual usage is paramount, and refining requests for memory resources is particularly crucial in this regard.

This study aims to identify the optimal machine learning algorithm to predict the memory requirements for compute jobs upon their submission. Previous studies in the area have primarily used models of regression with continuous values. By contrast, we focus on the implementation of techniques of classification to more efficiently predict the memory required by a submitted job. The remainder of this paper is organized as follows: Section 2 provides a review of the relevant literature, Section 3 outlines the methodology applied in our proposed solution, including feature selection and a list of machine learning methods. Section 4 details the experiments used to test the proposed method and the corresponding results, and the conclusion of this study and directions for future research are discussed in Section 5.

2. RELATED WORK

High-performance computing (HPC) systems have emerged as an important tool for tackling complex computational tasks across various domains in recent years, including scientific simulations, data analytics, and Artificial Intelligence. A critical challenge in managing HPC clusters is to optimize the memory used by the submitted job. Efficient memory allocation is essential to prevent the waste of resources, enhance job throughput, and ensure the optimal use of computational resources. Machine learning models have emerged as valuable tools for predicting and improving memory allocation in HPC environments, and help improve the overall performance and the efficiency of resource usage. The work in [4] introduced an offline, fully automated, and open-source machine learning-based solution. This tool helps users forecast the requirements of memory and time for the jobs that they have submitted within the given cluster. The framework of the tool includes the use of seven regression machine learning-based discriminative models: lasso least-angle regression, linear regression, ridge regression, elastic net regression, classification and regression trees, random forest regression, and the light gradient-boosting machine (LightGBM). LightGBM achieved an accuracy R2 of 0.72 in terms of predicting the memory requirements for jobs while RF attained an accuracy of 0.74 in predicting the time required for executing them. The work in [5] explored statistical and machine learning methods to accurately forecast the memory requirements of compute jobs submitted to the LSF®. This led to the creation of a memory recommender system for compute jobs tailored to the design of Qualcomm chips. It recorded an impressive accuracy of 90% in predicting the memory required for jobs. These results also highlight the potential of the system to substantially reduce the pending times of jobs. Moreover, the work in [6] developed a supervised machine learning model and integrated it into the Slurm resource management simulator. The primary objective of the model was to forecast the requisite memory resources and the estimated runtime for computational tasks. The authors explored a number of machine learning algorithms, including linear regression, LassoLarsIC regression, ElasticNetCV regression, ridge regression, and decision tree regression.

A practical method to predict the resources required by jobs demanding a large amount of memory was proposed in [7]. The method initially attempts to predict whether a given job is likely to use a substantial amount of memory, and subsequently uses a model trained exclusively on historical data related to jobs requiring a sizeable memory to forecast the final memory usage. The results of assessments showed that this method could reduce the average errors in predictions up to 40% for almost 90% of memory-intensive jobs. At the same time, the training cost for the model can be reduced by over 30%, as evidenced by the evaluated job traces. The anticipation of the memory bandwidth within the framework of non-uniform memory access architecture was undertaken in [8]. The authors used the *k*-nearest neighbor (KNN) regression technique due to its adaptability to a diversity of types of data, ability to accommodate modest volumes of data, harmonious interaction with non-uniform feature vectors, and ease of use. They used this approach to predict the memory bandwidths required for jobs of unknown sizes and thread counts. It is important to clarify that the primary aim of this study is not to achieve pinpoint precision in predicting the components of the memory bandwidth. Instead, we seek to use these components to achieve a satisfactorily accurate prediction of the memory bandwidth. The work in [9] used the support vector machine (SVM) to anticipate the reliability of Unix batch applications within the computational environment. They reported a significantly lower error rate compared with those incurred by straightforward algorithmic executions. Moreover, IBM devised an asynchronous model in [10] by blending the KNN, RF, and SVM. This amalgamated model was subsequently integrated into a voting structure to facilitate accurate predictions of memory-related prerequisites for computational tasks.

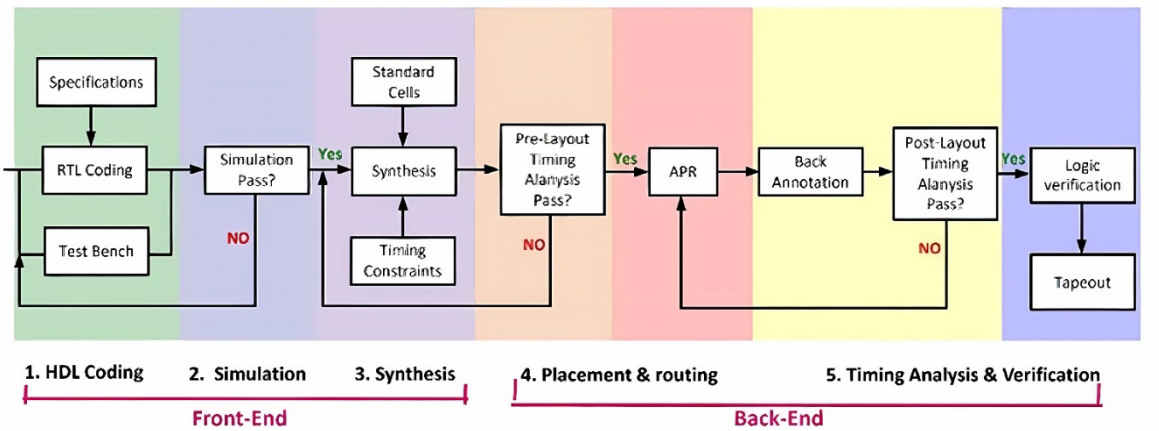

Figure 1. Typical lifecycle of the design of an FPGA [2]

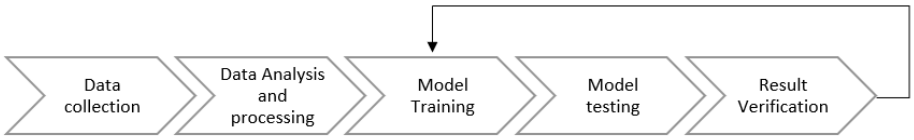

Figure 2. Generic lifecycle of machine learning methods

Table 1. Features and their descriptions

Features	Description
User	User executing the job
Queue	Virtual job scheduler queue where job is executed
Qslot	Virtual job scheduler Q slot (one level down from queue) where job is executed
Command	Full command that is run for the job
tool_name	Name of the tool (e.g., Python, Apache Redhawk, Synopsys PrimeTime)
tool_source	Developer/Owner of the tool (e.g., in-house, EDA vendor etc.)
project_name_mapped	Project name
Milestone	Project milestone
Branch	Project branch
topip	Project top-level IP
Department	User's department
core_used_bin	Average CPU used by job
mem_used_bin-class	Max. memory used by job

3. PROPOSED METHODOLOGY

This section outlines the research methodology used in this paper. Given the growing interest in data science and machine learning, numerous methodologies with subtle variations in their procedures have been proposed. We aim to identify the classification model that delivers the best performance, and thus emphasize the final three phases illustrated in Figure 2.

3.1 Data Collection

The dataset used in this study was obtained from a multi-national company specializing in semiconductors. The engineering tasks for the company were executed on a compute farm within a Unix environment. The compiled data originated from a batch job scheduler used by the organization. The features (labels) encompassed a subset of the default parameters combined with additional values from the execution wrapper, ARC. Approximately 567,816 records were extracted as from this dataset over a one-month window (February 2020). Table 1 provides a description of each feature. It is important to note that the values have been anonymized to preserve privacy.

3.2 Data Analysis and Processing

The data from a compute farm scheduler provide details related to the submitted jobs, including usernames, commands, core utilization, memory utilization, project names, and queues. Supplementary data were generated to enrich the dataset to provide a deeper understanding of an FPGA project. Such additional features as employee department, project name, branch, and top IP were incorporated. Employees within the same department are typically assigned to different aspects of a given project, while the department itself is generally dedicated to specific domains. The branch and top IP contribute weights to specific sections of the given domain. For instance, a particular IP (say, IP A) may be larger (in terms of significance, complexity, or resource) than another IP (IP C) such that it requires more memory for compilation and execution. Moreover, the "command" feature undergoes transformation, given that each command can easily become distinct. For instance, running "firefox page1.html" and "firefox page2.html" essentially serves the same purpose: using Mozilla Firefox to open HTML pages. However, transforming the data for use by machine learning techniques requires treating these instances as equivalent. Thus, the process involves cleaning up the data and applying regular expressions to the "command" feature. In the example of the "firefox" command, the process named "firefox" is captured as the value for the "command" feature."

3.2.1. Categorical Encoding

We use an approach to efficiently align memory requests within job allocations on the hardware of the server that is different from regression, in which the target class constitutes a continuous value. Instead, values of the request for memory are categorized into distinct groups or "bins" to formulate the problem of multi-class classification. Each sample in this context is attributed to a singular class, in contrast with multi-label classification in which a given instance can correspond to multiple labels. The following classes are established to accommodate servers with memory capacities ranging from 32 GB to 768 GB by using the class value mem_used_bin-class: CLASS = ["2," "4," "6," "8," "16," "32," "64," "96," "128," "192," "256," "320," "384," "512," "768"].

Like any machine learning-based method of classification, our approach involves translating categorical labels into numerical values before inputting them to the algorithm. The LabelEncoder is exceptionally effective for this purpose. One-hot encoding (OHE) is frequently used in the realm of categorical encoding. The underlying principle is to substitute values with their corresponding numerical representations. For instance, a project denoted by [A, B, C] is transformed into [1, 2, 3]. Conversely, OHE entails the conversion of values into three distinct columns to yield a representation in the form of [[0, 0, 0], [0, 1, 0], [1, 0, 0]].

Both the above methods of encoding have their own merits. However, owing to the substantial number of features, coupled with the presence of numerous unique values within each feature (more than 7,000 distinct top IPs and over 380,000 unique commands), the application of OHE led to memory-related errors in this case. As has been claimed in [11], label encoding is preferred when the number of categories is large because OHE can lead to the consumption of a large amount of memory. It may also introduce the curse of dimensionality in situations involving a multitude of unique values [12].

3.2.2. MinMax Scaling

According to the findings presented in [13], machine learning algorithms deliver superior performance when the features of the data share a consistent scale because this influences their interactions within the dimensional space. To this end, we used the MinMax scaling technique to adjust the values of the features and confine them within the range of (0, 1). Our preference for MinMax scaling primarily stems from its characteristic of not impacting the variance in the data. This is crucial as the technique is sensitive to anomalies that may emerge during the scaling procedure. Figure 3 illustrates an example of the data before and after applying this transformation.

3.3 Feature Selection

Feature selection was conducted in two phases. The features were examined based on their semantics in the first phase. We used a basic elimination strategy to this end. The second phase involved using techniques of feature selection to identify the relevant features. Five feature selection techniques, i.e. LassoCV, RandomForestClassifier (RFC), Chi-2, RandomForestClassifie with StratifiedKFold (RFECV) and ExtraTreesClassifier (ETC) have been selected to perform the feature selection in this phase. LassoCV combines Lasso regression with cross-validation. Lasso regression adds a penalty term to the linear regression that forces some of the model coefficients to be exactly zero; while RFC is an ensemble learning method based on decision trees. RFC can be used by examining the importance of each feature in the context of the random forest model. Features with higher importance scores are considered more relevant for the classification task; meanwhile, Chi-2 is a statistical test used for feature selection, especially in the context of classification tasks. It measures the independence between a categorical feature and the target variable; besides, RFECV starts with all features, fits the model, and eliminates the least important feature based on feature importance scores. It repeats this process while using cross-validation to find the optimal number of features to keep. ETC is another ensemble learning method where decision trees are trained on random subsets of the data and features. It assigns feature importance scores by considering the average impurity decrease each feature causes in the decision trees. Features with higher importance scores are considered more relevant. These feature selection techniques effectively selects a subset of the most important features.

Certain features were discarded in the first phase. For instance, "Jobid" was dropped as it served as a unique identifier. Such features as "workstation" and "Wtime," which were known only once a job had begun running, were also excluded. Moreover, features such as the number of cores and the memory consumed during runtime were retained as they represented actual usage, and this prompted the removal of "requested_core" and "requested_memory." Fields starting with "ARC_*" were non-scheduler features, and were derived from details of the project. These fields were used to derive information about the tools used (tool_name, tool_source) and details of the project (project_name_mapped, topip, milestone, branch), after which they were eliminated. Moving forward, the remaining features, excluding "mem_used_bin-class" as it represented the label, were assessed by using five feature selection algorithms. The top-ranking features output by each algorithm were then selected. Table 2 lists the 12 features chosen by using these algorithms.

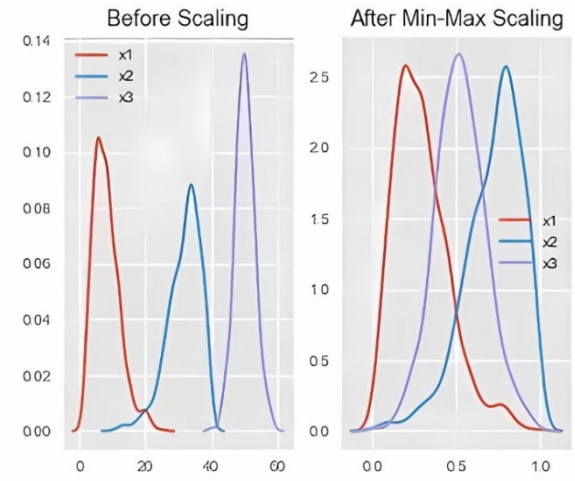

Figure 3. The data before and after the application of MinMax scaling [14]

Table 2. Feature selection result based on highest frequency

Features	LassoCV	RFC	Chi-2	RFECV	ETC	Total
User	Yes	Yes	Yes	Yes	Yes	5
tool_name	Yes	Yes	Yes	Yes	Yes	5
milestone	Yes	Yes	Yes	Yes	Yes	5
topip	Yes	Yes	Yes	Yes	Yes	5
command_regex	No	Yes	Yes	Yes	Yes	4
department	Yes	Yes	Yes	No	Yes	4
core_used_bin	Yes	Yes	Yes	No	Yes	4
Qslot	No	Yes	No	Yes	No	2
tool_source	Yes	No	No	No	Yes	2
Queue	No	No	Yes	No	No	1
project_name_mapped	Yes	No	No	No	No	1
branch	Yes	No	No	No	No	1

To ensure an unbiased approach, only features that appeared in the results of more than two algorithms (i.e., were nominated by at least two algorithms) were retained. Seven features were finally selected: FEATURES = ["User," "tool_name," "milestone," "topip," "command_regex," "department," "core_used_bin"].

3.4 Model Training

The dataset has been separated into training and testing set. 80% of the dataset was used to perform the model training while 20% of the dataset has been used to perform the testing. We tested several machine learning algorithms to identify the most effective one. Our strategy was guided by Occam's Razor [15], and we considered both parametric and non-parametric models, such as the KNN, naïve Bayes (NB), and SVM. This list was expanded to incorporate tree-based methods like extra trees and the random forest, along with ensemble techniques such as boosting and voting. Furthermore, apart from using well-established machine learning algorithms, we assessed the performance of more intricate algorithmic models. Instead of relying solely on a singular algorithm, we examined ensemble learning, a strategy that seeks to enhance the performance of predictive models by harnessing multiple weak learner models [16]. Ensemble learning, bagging, and boosting have found applications in diverse domains [17]–[19]. We examined three ensemble techniques to identify the one that yielded the highest accuracy. We provide a concise overview of each technique as well as the differences among them below.

Bagging, an acronym for bootstrap aggregation, represents a sampling technique designed to diminish variance in the data by averaging numerous estimates [20]. This is accomplished by training the model on distinct random subsets of the data. We used three classifiers to this end: bagged decision trees, the RF, and extra trees. Boosting constitutes a sequential learning technique [21]. The underlying concept is to train an initial model and then construct subsequent models that consider errors in the previous models (by incorporating weighted versions of the data). We used two classifiers in this study: AdaBoost and gradient boosting. We also used the voting ensemble, which is among the simplest forms of ensemble learning. The fundamental idea is to create multiple, independent models and subsequently arrive at a decision through voting (based either on majority or weight). We used a voting classifier based on the majority vote here. This entailed aggregating the predictions of each model and selecting the class with the largest number of votes. Stratified K-fold validation was used to assess the effectiveness of the machine learning algorithms. This approach seeks to maintain a roughly balanced distribution of classes within each fold [22].

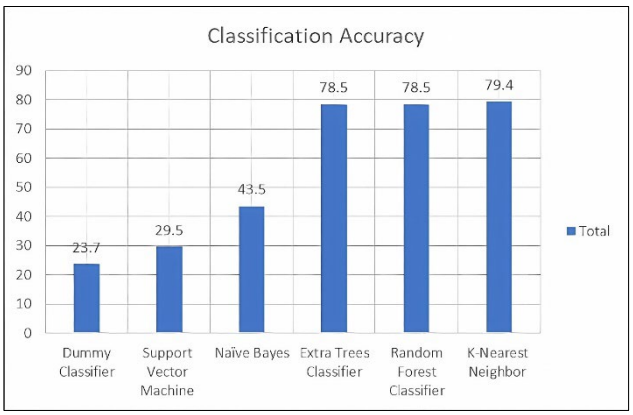

Figure 4. Comparison of the accuracy of the classifiers

Table 3. Results of the classifiers

Model	Accuracy	Precision	Recall	F1-score	Time (s)
Dummy classifier	23.7	23.8	23.7	23.7	1
SVM	29.5	48.4	29.5	30	3172
NB	43.5	27.2	43.5	28.8	1
KNN	**79.4**	**78.7**	**79.4**	**79**	102
RF	78.5	78.4	78.5	78.4	55
Extra trees	78.5	78.3	78.5	78.3	31

Table 4. Testing the hyperparameters of the SVM

Model	Accuracy	Precision	Recall	F1-score	Time (s)
SVM (RBF) - C=0.1	10.3	27.7	10.3	7.6	2879
SVM (RBF) - C=1.0	29.5	48.4	29.5	30	3172
SVM (RBF) - C=10	49.4	**63.3**	49.4	48.1	3082
SVM (RBF) - C=10, class_weight="balanced"	17.9	49.7	17.9	14.6	13532
SVC (linear) - C=1.0	10.6	26.2	10.6	7.8	1545
LinearSVC - C=1.0	54	41.6	54	43.1	29
LinearSVC - C=10	54	41.6	54	43.1	29
LinearSVC - C=10, class_weight="balanced"	**55.5**	49.5	**55.5**	**49.6**	**27**

4. RESULTS AND DISCUSSION

As outlined in the previous section, we applied a variety of machine learning models to the dataset. They covered a wide spectrum, ranging from the dummy classifier to standard models of classification like the NB, SVM, KNN, tree-based models (RF and extra trees), and ensemble methods such as boosting and voting. The objective was to ascertain whether the standard models could achieve satisfactory results or if ensemble methods were needed. The dummy classifier established a baseline metric. Such models as the NB were chosen from the pools of parametric algorithms to assess feature independence. The KNN and tree-based models were selected from among the non-parametric methods to explore feature relevance, and were expected to perform well owing to their non-parametric nature and high algorithmic complexity. The results presented in Figure 4 are based on the modules of scikit-learn with the default parameters.

Table 3 shows that the dummy classifier yielded an accuracy below 25%. It used a stratified strategy by default to generate predictions based on the distribution of classes in the training set. This served as an initial baseline of performance, and any result surpassing this value indicated an enhancement, and was considered. The default SVM algorithm in scikit yielded an accuracy of 29.5%, which represents an improvement over the baseline dummy classifier even though it is a low value. Table 4 provides details of the tuning process and the modules of scikit used: namely, the SVM and LinearSVC. Changes in the value of C influenced the accuracy of the methods because C served as a regularization parameter in the SVM. A low value of C led to a larger margin from the hyperplane, thus reducing the accuracy of training depending on the data. On the contrary, a high value of C corresponded to a narrower margin, leading to a tighter distinction between classes and consequently yielding better results. This phenomenon is illustrated in the table below, where a value of C of 10 led to the highest precision of the SVC.

It is interesting to note the differences between SVC (linear) and LinearSVC. They aligned with the findings presented in [23]. The SVM employed a one-vs.-one scheme to construct n_classes * (n_classes - 1)/2 classifiers. By contrast, LinearSVC used a one-vs.-rest approach and constructed n_classes classifiers. It applied a squared hinge loss function while the SVM used hinge loss. LinearSVC was implemented in liblinear [24], a linear classifier known for its satisfactory scalability when dealing with large datasets, and this significantly reduced the time needed for training. Using LinearSVC with the default parameters yielded a twofold improvement over the dummy classifier, and it recorded an accuracy of 55.5%.

The NB algorithm yielded an accuracy of 43.5%, surpassing the baseline dummy classifier. However, its performance was significantly inferior to that of the KNN and forest algorithms. This discrepancy can be attributed to its data preprocessing. Because NB is predicated on the assumption of feature independence, using OHE on the dataset can enhance its performance [25]. However, this would have caused the dataset to have a large number of dimensions, and would have potentially led to the curse of dimensionality owing to the large number of unique values in certain fields. An alternative is to downsize the dataset before applying OHE, but this approach entails data loss—an undesirable outcome in an experiment designed to evaluate classifiers by using the same data.

The remaining three models—namely, the KNN, RF, and extra trees—yielded minimal training times coupled with a significantly improved accuracy of over 75%, more than three times higher than that of the baseline dummy classifier. A key feature common to these three models is their non-parametric nature [25]. As such, these algorithms make no assumptions about the data except for the logic that they inherently employ. For instance, the logic of the KNN assumes that patterns of nodes neighboring the given node (as determined by the *k* value) share similar classes, while the tree algorithms construct internal decision trees to derive their results. This innate flexibility contributed to their improved performance. Numerous comparisons between tree algorithms and the KNN have been documented, such as in [26]–[28], and have shown that the performance of these algorithms remains closely aligned. Both algorithms exhibited a similar growth in accuracy as the size of the dataset increased in the results reported in [26] in particular. This corresponds to our findings thus far, whereby these algorithms delivered improved performance owing to the large size of the dataset, which contained over 500,000 records.

4.1 Ensemble Learning

We now focus on evaluating ensemble models, specifically boosting and voting. The parameters of the base classifier were retained at their default values to analyze boosting, while the three best-performing classifiers were used to analyze voting. Of the six initial classifiers, the best performance was recorded by the KNN, with an accuracy of 79.4%, followed by RF at 78.5% and extra trees at 78.5%. We thus applied ensemble voting based on the KNN, RF, and extra trees. Table 5 provides an

overview of the results. The voting approach yielded the highest accuracy of 79.8% among the ensemble models, marking a 0.4% improvement compared with when the models were executed by themselves.

4.1.1 Boosting

We tuned the hyperparameters of AdaBoost to analyze its accuracy. We considered the parameters n_estimators and learning_rate because there is a trade-off between them [29]. n_estimators signifies the number of iterations of AdaBoost while learning_rate denotes the weight reduction factor per cycle. Typically, more iterations may be necessary for weak learners on smaller datasets. Reducing the value of learning_rate prolongs convergence but allows the model to be trained more effectively. Figure 5 illustrates the results of simulations run by varying the values of n_estimators (X-axis) and learning_rate (series) from 0.2 to the default value of 1. The findings align with those of the learning rate vs. estimator theory: A lower learning rate corresponded to superior performance. Using a high learning rate in conjunction with a high value of n_estimators degraded the performance of the model. Specifically, when n_estimators was set to 500 and learning_rate to 1, its accuracy was below 30%. It exhibited similar performance with other configurations within this range. The optimal performance, an accuracy of 58.9%, was obtained when n_estimators was set to 100 and learning_rate to 0.2. This configuration is represented by the green line in Figure 5.

In a similar manner to the above, we tuned the hyperparameters of gradient boosting by exploring the relationship between the estimators and the learning rate. In contrast to AdaBoost, the accuracy of gradient boosting did not vary significantly, and its various configurations recorded accuracies ranging from 70% to 80%. The best result, an accuracy of 79.3%, was obtained when n_estimators was set to 500 and learning_rate to 0.08. This particular configuration is represented by the yellow line in Figure 6.

A significant contrast was noted between the behaviors of AdaBoost and gradient boosting as the number of estimators was increased. While the performance of AdaBoost tended to suffer in this case, that of gradient boosting improved with the number of estimators. This discrepancy arose from the distinct approaches used by these algorithms for boosting. While both algorithms operate on the principle of generating a collection of weak learners—typically, one-level decision trees that are known as decision stumps—and enhancing their effectiveness to form stronger learners, the AdaBoost model seeks to fit weak learners during every iteration. It adjusts the weights of the samples in each iteration by assigning larger weights to samples containing incorrect observations and smaller weights to those containing accurate observations. Its final prediction is determined through a weighted majority vote of the weak learners [30].

By contrast, gradient boosting focuses on minimizing the loss function in each iteration, with every subsequent iteration designed to improve upon the errors identified in the previous round. As has been claimed in Ref. [31], gradient boosting identifies challenging observations based on their substantial residuals computed in prior iterations. The analogy of a golfer refining their swing (as shown in Figure 7) aptly captures the essence of how gradient boosting operates.

Table 6 compares the performance of gradient boosting and AdaBoost between when their default values were used and when their hyperparameters were tuned through ensemble models. The gradient boosting model, when its parameters were tuned by the ensemble model, achieved the highest accuracy of 79.3%.

Table 5. Results of ensemble models

Ensemble Models	Accuracy	Precision	Recall	F1-score	Time (s)
Boosting (AdaBoost)	58.3	49.4	58.3	50.1	17
Boosting (Gradient boosting)	77.1	76.5	77.1	75.3	445
Voting (KNN, RF, extra trees)	**79.8**	**78.7**	**79.8**	**78.9**	**161**

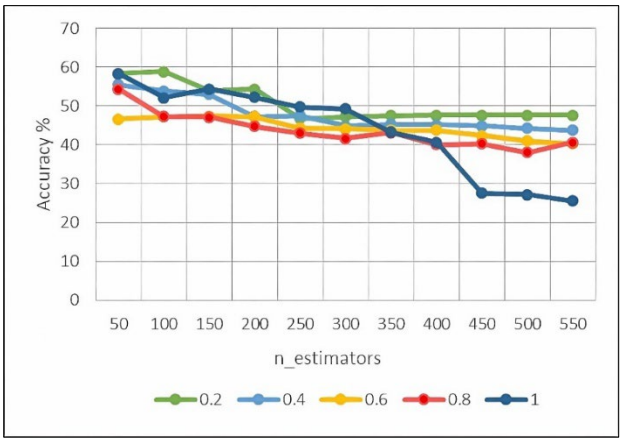

Figure 5. Values of accuracy of AdaBoost with different values of n_estimators and learning_rate

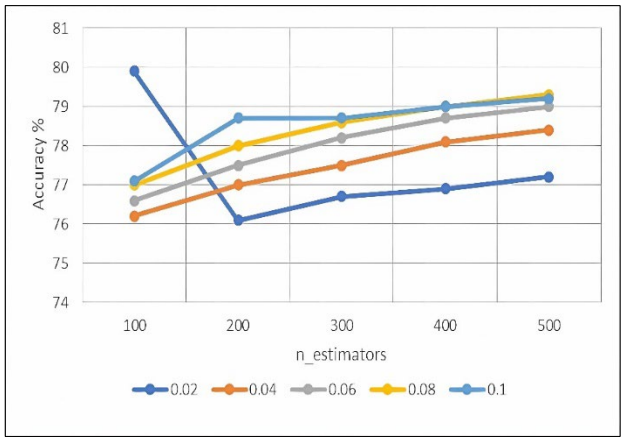

Figure 6. Variations in the accuracy of gradient boosting with different values of n_estimator and learning_rate

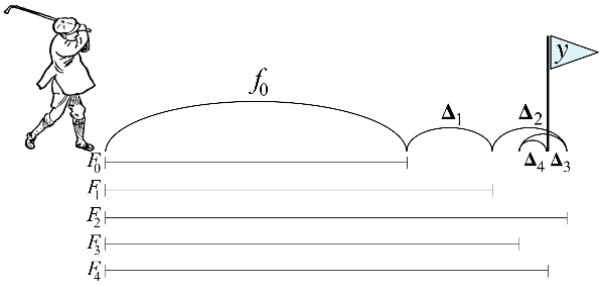

Figure 7. Golfer analogy for gradient boosting [32]

Table 6. Comparison of results of AdaBoost and gradient boosting between when their default values were used and when their hyperparameters were tuned by using ensemble models

Model	n_estimators	learning_rate	Accuracy	Precision	Recall	F1-score	Time (s)
Gradient Boosting–Default	100	0.1	77.1	76.5	77.1	75.3	441
Gradient Boosting–Tuned	500	0.08	**79.3**	**78.5**	**79.3**	**77.8**	2242
AdaBoost–**Default**	50	1	58.3	49.4	58.3	50.1	17
AdaBoost–**Tuned**	100	0.2	58.9	47.1	58.9	48.8	35

Table 7. Results of the KNN with different values of *k*

k	Accuracy	Precision	Recall	F1-score	Time (s)
1	77.1	77.4	77.1	77.2	105
3	79.4	78.7	79.4	79	105
5	80.4	79.6	80.4	79.8	105
7	80.7	80	80.7	80.1	106
9	80.7	79.9	80.7	80	104
11	80.8	80.1	80.8	80.1	104
13	80.8	80.1	80.8	80.1	104
15	80.8	80.1	80.8	80	105
17	80.7	80	80.7	79.9	105
19	80.7	80	80.7	79.9	105
21	80.6	79.9	80.6	79.8	105
23	80.6	79.9	80.6	79.7	106
25	80.5	79.7	80.5	79.6	106
27	80.4	79.7	80.4	79.5	106
29	80.3	79.5	80.3	79.4	106
31	80.2	79.4	80.2	79.3	107

4.1.2 Voting

The voting ensemble was formed by selecting the three best-performing initial classifiers. The hyperparameters of each classifier were tuned to assess the potential for improving its performance in this way. The accuracy of the KNN improved until $k = 7$, beyond which further increases in its value led to an improvement in neither the accuracy nor the F1-score of the model. This indicates a saturation point, such that overfitting can occur if k is set to larger than seven, as shown in Table 7.

The accuracy of AdaBoost was within the range of 40% to 60%, and was contingent on the learning rate. When a higher learning rate (step) was paired with a larger number of estimators, the model had a propensity to overfit on the training set compared with the test set, which degraded its performance. The work in [29] has noted the substantial influence of the number of trees, n_estimators, on the outcomes. Consequently, we examined the performance of the RF and extra trees classifiers over a range of values of n_estimators by increasing it by 50. Intriguingly, this did not yield a significant enhancement in the accuracy of the models. Only a marginal increase of 0.1% was observed, suggesting that the models were sufficiently robust when using the default value of n_estimators shown in Figures 8 and 9.

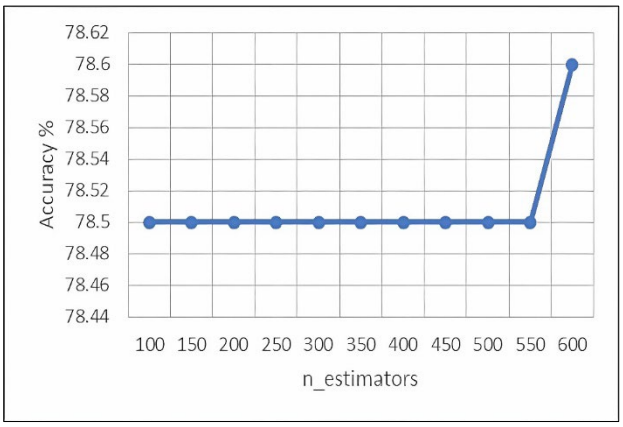

Figure 8. RF with different n_estimators

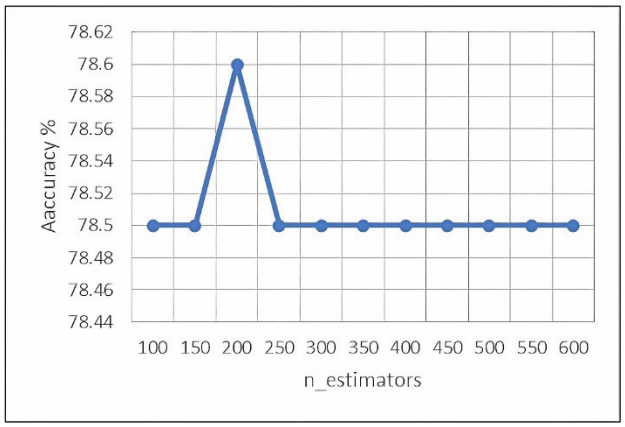

Figure 9. Extra trees with different n_estimators

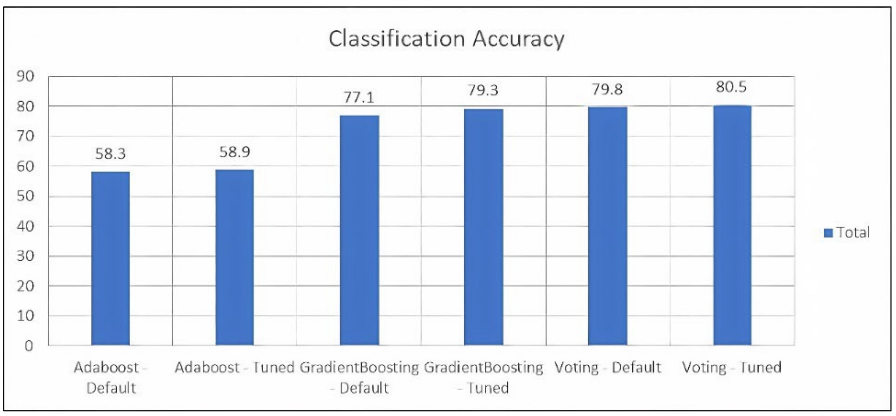

Figure 10. Comparison of the results of various models between when their default values were used and when their hyperparameters were tuned by ensemble models

Table 8. Comparison of the results of various models between when their default values were used and when their hyperparameters were tuned by ensemble models

Model	Accuracy	Precision	Recall	F1-score	Time (s)
Voting–Default	79.8	78.7	79.8	78.9	161
Voting–Tuned	**80.5**	**79.6**	**80.5**	**79.4**	300
Gradient Boosting–Default	77.1	76.5	77.1	75.3	441
Gradient Boosting–Tuned	79.3	78.5	79.3	77.8	2242
AdaBoost **- Default**	58.3	49.4	58.3	50.1	**17**
AdaBoost **- Tuned**	58.9	47.1	58.9	48.8	35

Tuning the hyperparameters yielded an accuracy of 80.5% when using an ensemble voting approach based on the KNN (k = 7), RF (n_estimators = 600), and extra trees (n_estimators = 200). This result surpasses the accuracy of the "Gradient Boosting–Tuned" model by 1.2%, and was the highest accuracy achieved in this experiment thus far. The outcomes of the ensemble model when it used both boosting and voting, and used its default parameters and those tuned by the ensemble model are depicted in Figure 10 and Table 8.

5. CONCLUSION

The authors of this study investigated the applicability of various machine learning methods to predict the memory requested by jobs within a compute farm environment. The results showed that ensemble learning was the best approach when hyperparameter tuning was used to refine the models. By appropriately balancing the performance of the model, its training time, maintainability, and other constraints, the ensemble voting method was the best approach with the KNN, RF, and extra trees classifiers. Their hyperparameters had been tuned in advance.

The experimental results highlighted the versatility of applying different classifiers to the dataset to yield diverse initial

results. The performance of the classifiers was optimized by tuning their parameters. The evaluation of ensemble models underscored the significance of methods of data preprocessing in determining the appropriateness of each model. We plan to extend the scope of the proposed model in future work in the area by applying it to larger and more diverse datasets. In the first experimental phase, we will formulate a comprehensive dataset and assess the capability of the ensemble voting model to predict the memory requirements of jobs at the time of their submission. Continual refinements to the model will involve retraining it on new data within a rolling one-week window. The second experimental phase will aim to ascertain the generalizability of the model across distinct datasets.

Amid the global focus on reducing our carbon footprint and advancing green computing in data centers, the importance of accurate resource utilization is paramount [33]. Applications of the compute farm scheduler can leverage machine learning algorithms to enhance memory-related intelligence, optimize job scheduling, and reduce cost. Such optimization translates into fewer hardware failures, better uptime, less manpower, and lower costs of maintenance while enabling a higher rate of batch job completion by using the available resources of the server.

ACKNOWLEDGEMENT AND FUNDING

This work is supported by the Ministry of Higher Education Malaysia, Fundamental and Research Grant Scheme under Grant No. FRGS/1/2019/STG06/USM/02/6 for the project entitled “A New Hybrid Model for Monitoring the Multivariate Coefficient of Variation in Healthcare Surveillance”.

DECLARATION OF CONFLICTING INTERESTS

The authors declare no potential conflicts of interest with respect to the research and publication of this article.

REFERENCES

[1] Prisma, Speed to market: Why is it so important? https://www.poweredbyprisma.com/speed-to-market-why-is-it-so-important/, 2022 (accessed 23.12.2022).
[2] Asicnorth, ASIC vs FPGA: What's the difference?, https://www.asicnorth.com/blog/asic-vs-fpga-difference/, 2020 (accessed 01.03.2023).
[3] A. Reuther *et al.*, Scheduler technologies in support of high performance data analysis, *2016 IEEE High Performance Extreme Computing Conference HPEC 2016*, Waltham, USA, 2016, 1-6.
[4] M. Tanash, D. Andresen and W. Hsu, AMPRO-HPCC: A machine-learning tool for predicting resources on slurm HPC clusters, *ADVCOMP International Conference on Advanced Engineering Computing and Applications in Sciences*, Barcelona, Spain, 2021, 20-27.
[5] T. Taghavi, M. Lupetini and Y. Kretchmer, Compute job memory recommender system using machine learning, *Proceedings ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, USA, 2016, 609-616.
[6] M. Tanash, B. Dunn, D. Andresen, W. Hsu, H. Yang and A. Okanlawon, Improving HPC system performance by predicting job resources via supervised machine learning, *ACM International Conference Proceedings Series*, Daejeon, South Korea, 2019, 1-8.
[7] X. Li, N. Qi, Y. He and B. McMillan, Practical resource usage prediction method for large memory jobs in HPC clusters, *Asian Conference on Supercomputing Frontiers*, Singapore, 2019, 1-18.
[8] S. Salehian and L. Lu, Memory bandwidth prediction in NUMA architecture using supervised machine learning, *International Conference on Computational Science and Computational Intelligence (CSCI 2019)*, Las Vegas, USA, 2019, 1517-1522.
[9] D. Oliveira, F. B. Moreira, P. Rech and P. Navaux, Predicting the reliability behavior of HPC applications, *30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2018)*, Lyon, France, 2019, 124-131.
[10] E. R. Rodrigues, R. L. F. Cunha, M. A. S. Netto and M. Spriggs, *2016 Third International Workshop on HPC User Support Tools (HUST)*, Salt Lake City, USA, 45, 2016.
[11] A. Sethi, One-hot encoding vs. label encoding using Scikit-Learn , https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/, 2020 (accessed 21.12.2022).
[12] R. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.
[13] A. Ethem, *Introduction to Machine Learning*, Fourth Edi. The MIT Press, 2014.
[14] B. Keen, Feature scaling with Scikit-Learn, http://benalexkeen.com/feature-scaling-with-scikit-learn/, 2017 (accessed 21.12.2022).
[15] P. Gibbs and H. Sugihara, What is Occam’s Razor? https://math.ucr.edu/home/baez/physics/General/occam.html, 1997 (accessed 20.01.2023).
[16] X. Zhu, J. Li, J. Ren, J. Wang and G. Wang, Dynamic ensemble learning for multi-label classification, *Information Sciences*, 623, 2023, 94-111.
[17] M. H. D. M. Ribeiro, R. G. da Silva, G. T. Ribeiro, V. C. Mariani and L. dos S. Coelho, Cooperative ensemble learning model improves electric short-term load forecasting, *Chaos, Solitons and Fractals*, 166, 2023, 112982.
[18] C. Choudhary, I. Singh and M. Kumar, SARWAS: Deep ensemble learning techniques for sentiment based recommendation system, *Expert System with Applications*, 216, 2023, 119420.
[19] C. Cakiroglu, K. Islam, G. Bekdaş and M. L. Nehdi, Data-driven ensemble learning approach for optimal design of cantilever soldier pile retaining walls, *Structures*, 51, 2023, 1268-1280.

[20] X. Liu, A. Liu, J. L. Chen and G. Li, Impact of decomposition on time series bagging forecasting performance, *Tourism Management*, 97, 2023, 104725.
[21] I. F. Kilincer, F. Ertam and A. Sengur, A comprehensive intrusion detection framework using boosting algorithms, *Computers and Electrical Engineering*, 100, 2022, 107869.
[22] S. Widodo, H. Brawijaya and S. Samudi, Stratified K-fold cross validation optimization on machine learning for prediction, *Sinkron*, 7, 2022, 2407-2414.
[23] H. Zou and Z. Jin, Comparative study of big data classification algorithm based on SVM, *2018 Cross Strait Quad-Regional Radio Science and Wireless Technology Conference (CSQRWC 2018)*, Xuzhou, China, 2, 2018, 1-3.
[24] R. -E. Fan, K. -W. Chang, C. -J. Hsieh, X. -R. Wang and C. -J. Lin, LIBLINEAR: A library for large linear classification, *The Journal of Machine Learning Research*, 9, 2008, 1871-1874.
[25] J. Brownlee, Parametric and Nonparametric Machine Learning Algorithms, https://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/, 2016 (accessed 15.02.2023).
[26] R. Devika, S. V. Avilala and V. Subramaniyaswamy, Comparative study of classifier for chronic kidney disease prediction using naive bayes, KNN and random forest, *3rd International Conference on Computing Methodologies and Communication (ICCMC 2019)*, Erode, India, 2019, 679-684.
[27] R. Nugrahaeni and K. Mutijarsa, Comparative analysis of machine learning KNN, SVM, and random forests algorithm for facial expression classification, *2016 International Seminar on Application for Technology of Information Communication*, Minna, Nigeria, 2016, 163-168.
[28] N. S. Intizhami, A. Y. Husodo and W. Jatmiko, Warfare simulation: Predicting battleship winner using random forest, *2019 IEEE International Conference on Communication, Networks and Satellite (ComNetSat)*, Makassar, Indonesia, 2019, 30-34.
[29] Z. Tan, Z. Yan and G. Zhu, Stock selection with random forest: An exploitation of excess return in the Chinese stock market, *Heliyon*, 5, 2019, e02310.
[30] T. Hastie, T. Robert and J. Friedman, *The Elements of Statistical Learning*, 2nd Edition, Springer New York, 2009.
[31] A. Mayr, H. Binder, O. Gefeller and M. Schmid, The evolution of boosting algorithms: From machine learning to statistical modelling, *Methods of Information in Medicine*, 53, 2014, 419-427.
[32] T. Parr and J. Howard, Gradient boosting: Distance to target, https://explained.ai/gradient-boosting/L2-loss.html, 2018 (accessed 22.02.2023).
[33] M. Soltanshahi, R. Asemi and N. Shafiei, Energy-aware virtual machines allocation by krill herd algorithm in cloud data centers, *Heliyon*, 5, 2019, 3-8.