湖南大学学报（自然科学版）
Journal of Hunan University (Natural Sciences)
Available online at http://jonuns.org

第 53 卷 第 1 期
2025 年

Vol. 53 No. 01
2025

Open Access Article

10.5281/zenodo.17079436

Computational Trade-offs Between Traditional Algorithms and Machine Learning Models: A Time Complexity Perspective

[1] S. Sivagurunathan, [2*]Sudhaman Parthasarathy

[1]Associate Professor,
[1]Dept of Computer Science and Applications, The Gandhigram Rural Institute (Deemed to be University), India
[1]email:svgrnth@gmail.com

[2]Professor of Data Science
[2]Dept of Applied Mathematics and Computational Science, Thiagarajar College of Engineering, India
[2] Corresponding Author:spcse@tce.edu

Abstract: A fundamental distinction exists between the time complexity of traditional algorithms and machine learning (ML) algorithms. Traditional algorithms are used to solve specific problems by following a set of instructions. Machine learning algorithms are created to extract knowledge from data and apply what they have learned to new, unobserved data. In this research work, we find that there are dissimilar time complexity features between traditional algorithms and machine learning algorithms and therefore we suggest that their evaluation criteria should differ from each other while we perform an efficiency analysis of traditional algorithms and machine learning algorithms. We distinguish this research work from prior related research works by examining the relative performance of machine learning models and traditional algorithms in terms of training and inference time.

Keywords: Time complexity, machine learning, algorithms, efficiency analysis.

Origin of Algorithms

The term "algorithm" originates from the name of the ninth-century Persian mathematician and astronomer, Al-Khwarizmi, who wrote a book on the subject called "Kitab al-Jabr wa-l-Muqabala" (The Book of Restoration and Reduction) (Baki, 1992). An algorithm is a set of rules or instructions used to solve a problem. An algorithm is a specific type of control structure that is finite, abstract, and efficient and is designed to achieve a particular objective based on a predetermined set of rules. Artificial intelligence (AI) refers to algorithms, models, and systems that can learn from data and make predictions or decisions (Mehrabi et al., 2021). Machine learning (ML) is a subfield of AI that employs algorithms and models to enable computers to learn and improve autonomously, without explicit programming (Nguyen et al., 2020, Zhou, 2021, Zhang et al., 2023, Ali et al., 2023). Knowledge about data patterns allow algorithms to learn as well as predict/decide based on this. Machine learning algorithms include data preparation, model training as well as model evaluation. Different techniques of machine learning are supervised, unsupervised and reinforcement learning (Berry et al., 2019, Zhang et al., 2023, Ali et al., 2023).

The supervised machine learning methods include logistic regression, linear regression, decision trees and support vector machines (Zheng et al., 2021; Zeguendey et al., 2023; Parthasarathy & Padmapriya, 2023). However, in unsupervised learning, the algorithms operate on data without specific and pre-defined targets. Prevalent unsupervised learning techniques include k-means clustering and principal component analysis. Reinforcement learning algorithm learns by interacting with its environment and getting feedback—rewards for doing well and penalties for making mistakes. Over time, it gets better at making decisions. Techniques like Q-learning and SARSA are often used in this approach. Despite the fact that such machine learning algorithms and their applications are widely known, no prior studies did a systematic comparison between their computational properties, specifically, time complexity and those of traditional algorithmic paradigms. This research work addresses this gap by examining the relative performance of machine learning models and traditional algorithms in terms of their training and inference time.

In this research, by the term ”time complexity”, we refer to the time taken by an algorithm to execute, which usually depends on how much input data it has to process. This process involves quantifying the duration of individual code statements within an algorithm. The comprehensive run-time of the algorithm was not evaluated. Efficient algorithms can differentiate between software running for a year or a second. Hence, assessing the time complexity of algorithms is inevitable for any software product before its deployment. By redefining traditional time complexity analysis in light of modern machine learning models, which incorporate distinctive multi-phase operations like training, validation, and inference, we propose comparative insights that aren't specifically summarized in the literature (Binson et al. 2024; Assis et al. 2025) by comparing the performance of traditional and machine learning algorithms under comparable problem scenarios in terms of time complexity.

2 Various Time Complexities

In the process of analyzing the efficiency of algorithms, a mathematical notation namely “Big Oh notation” is used to analyze the performance and complexity of algorithms. This pertains to the evaluation of an algorithm's maximum performance level, specifically in terms of its behavior in the most unfavorable scenario. The analysis also considers the asymptotic behavior of the algorithm, which pertains to its performance as the input size approaches infinity. The time complexities of the algorithms describe how their execution time scales with input size. Big Oh Notations include constant (O (1)), linear (O (n)), quadratic (O (n2)), and logarithmic (O (log n)) complexities, each representing a different rate of growth, as shown in Figure 1.

Figure 1. Depiction of Various Time Complexities

2.1 Determinants of efficiency in Traditional Algorithms and ML algorithms

A fundamental distinction exists between the time complexity of traditional and machine learning (ML) algorithms. The computational operations of a traditional algorithm are usually expressed as a

function of the magnitude of the input for its temporal efficiency. Quick sort and merge sort are usually followed by sorting algorithms that have average time complexity of O(n log n). This implies that they are roughly going to utilize the same amount of computational time in proportion to the logarithm of their input size. Temporal efficiency of a machine learning algorithm is often measured as the number of training instances needed to build the proficient model, and the number of attributes that each of that training instance has (Shrestha & Mahmood, 2019; Huang et al., 2020; Zeguendry et al., 2023, Dahiya, 2023). The complexity of the model, the size of the dataset, and the optimization technique affords training a machine learning algorithm to take a great deal of time and undergo a fair amount of variability by virtue of this. We include classical complexity classes to highlight the difference between traditional algorithms and machine learning models. Unlike traditional algorithms, ML models don't follow a single, fixed time complexity. Their computation depends on different stages like training and inference which means we need to look at time complexity in a more flexible way when analyzing them.

The analytical approximation of time complexity of a machine learning algorithm traditionally involves a visual examination of what the computational expenditure is as a mathematical expression of the size of the input data, which is usually expressed as the number of training instances and the number of attributes per instance. More specifically, it can be described in terms of the computational expenditure in terms of the number of floating point operations (FLOPs) needed to train the model or in terms of the number of iterative required by the optimization algorithm. There are some very simple machine learning algorithms that in principle are determined by how long they take to run: linear regression. In particular, this algorithm has been estimated to run in time O (nd^2) with 'n' being the number of training instances and 'd' being the number of features. Time complexity in deep learning model can be estimated by analyzing number of layers, size of each layer and count of parameters in the model. One possible estimation of time complexity for a convolutional neural network (CNN) is O (k^2nd^2) where k is convolutional kernel size, 'n' is the number of training instances and 'd' is the number of features. It is more advantageous empirically to evaluate the run time of a machine learning algorithm on a particular dataset using a profiler or a benchmarking tool rather than based on the algorithm's theoretical running time. The reason for that is that the theoretical time complexity might not match the actual algorithm's performance in real-time setting.

3. Computational Complexity and Algorithm's Efficiency

Consider the variations in running time in a loop for n=100 and n=1000, where n is the input size. The computational time of a loop is typically proportional to its number of cycles. Therefore, when the loop executes n times for n = 100, it takes less time than when it executes n times for n = 1000, which entails ten times more iterations. For n = 100, the total time will be 0.0001 seconds, while for n = 1000, it is 0.0010 seconds. This time variation is insignificant. This example provides a basic comparison to help understand how linear scaling appears in both traditional algorithms and machine learning algorithms. For the purpose of iterating over training examples or performing gradient updates during training, machine learning algorithms frequently employ loops. The temporal complexity of said operations is commonly in direct proportion to the number of training instances. As a result, generally, duration of execution for the algorithm will increase if we speculate that we run the algorithm with sample size of n = 1000 training examples rather than n = 100. This emphasizes the need to differentiate machine learning algorithms from traditional ones because in this case, size of input dataset impacts not only execution time but also model performance and generalization, which are factors that are rarely relevant to computational procedures. This explains our emphasis on the need for fundamentally different evaluation criteria for ML algorithms.

The computational time is subject to change by algorithm complexity, dimensionality and magnitude of input data, available resources of hardware and software at disposal and execution of the algorithm. It should be noted that the effectiveness of ML models also relies on underlying infrastructure (computational resources) in contrast to traditional algorithms, which are typically evaluated without taking hardware resources into account. Parallelization techniques is one way that can be applied to certain machine learning algorithms to improve its efficiency on multi-core or distributed systems. This will be useful to get better understanding of the link between priori (theoretical) and posteriori algorithm analysis. It indicates how different system configurations can affect an algorithm's performance, which is not usually the case with traditional algorithms.

When working with big datasets or complex models, the computational complexity and scalability of the machine learning algorithm to be used have to be considered. This forms the basis for the present research work where we state that the machine learning models must be evaluated in terms of how they scale under increasing computational demands rather than just their algorithmic structure (e.g., CNN or KNN).

Table 1.0 Efficiency Analysis of Traditional Algorithms and ML Algorithms

	Traditional Algorithm	**Machine Learning Algorithm**
Problem Definition	Sorting an Array with Quick Sort	Classifying Data with a Decision Tree
Efficiency Analysis Methodology	We were given an array of integers which we wanted to sort. A traditional sorting algorithm namely "Quick Sort" can be used. Its time complexity is always measured in terms of the number of comparisons and swaps required to sort an array which ultimately determines the Quick Sorts efficiency. Using Quick Sort we can never achieve an optimal time complexity but the worst-case time complexity of the Quick Sort is $O(n^2)$ where 'n' would be the number of elements in the array. The analysis here is of the straightforward type, mostly based on the input size and the needed operations (comparing and swapping) that are involved in the sorting process.	In case of us being given a dataset of labelled instances, we want to classify new unseen data points. This can be used as a machine learning algorithm called 'decision tree classifier.' This means the efficiency of this algorithm has to do with the time complexity of constructing your decision tree but also just how well the model generalises to new, unseen data. The efficiency analysis is made somewhat different. The complexity of building the decision tree is influenced by the depth of it and the dataset size because it can be very different and depends on the characteristics of a dataset. As for prediction time, once the decision tree is established, classifying a new data point is relatively quick, typically requiring only a traversal of the tree to reach a classification. With regard to model evaluation, unlike traditional algorithms, machine learning models require evaluation based on their accuracy, precision, recall, and other metrics. This aspect of efficiency analysis considers the model's performance on unseen data, rather than just its execution time.
Inference	We can easily calculate and predict the time complexity for a given input size. The efficiency analysis of traditional algorithms focuses on a clear-cut calculation of operations based on the input size.	The efficiency analysis of machine learning algorithms involves a more complex evaluation of both model training and performance metrics.

This would not be addressed in traditional time-complexity analysis of algorithms. In these cases, the optimization of algorithm implementation, the use of parallelization or distributed computing, e.g. involving specialized hardware such as Graphical Processing Unit (GPU), Tensor Processing Unit (TPU) etc. may be required. Graphical Processing Units can simultaneously execute tasks on intricate problems partitioned into discrete tasks, while Tensor Processing Units were designed specifically for neural network workloads and can process each task at faster speed than GPUs and require lesser resources. TPUs at Google are created to speed up machine learning tasks. With the advantage of Google's extensive experience and guidance in the field of machine learning, Tensor Processing Units were developed.

3.1 Comparing Efficiency Analysis of Traditional Algorithms and ML Algorithms

Figure 1 shows the difference between traditional algorithms and machine learning algorithms. The machine learning algorithm is responsible for recognizing patterns in data and then making predictions about new data in a manner that is analogous to the previous one. It is helpful in enabling computers to understand without being obviously programmed with rules that have been established in the past.

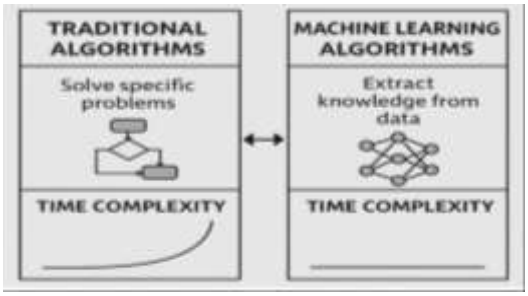

Figure 1 Traditional Algorithms Vs. Machine Learning Algorithms

To illustrate the differences in efficiency analysis between traditional algorithms and machine learning algorithms, we provided an illustration in Table 1.0. In fact, we have considered two examples: sorting an array using a standard algorithm and classifying data using a machine learning algorithms.

We also mathematically represent the efficiency analysis of a traditional algorithm (Quick Sort) and a machine learning algorithm (Decision Tree Classifier).

Traditional Algorithm: Quick Sort

One of the best-known sorting algorithms is Quick Sort and its average time complexity is usually given to be O(n*log n), where 'n' is the number of elements in the array. In the case of the worst case (where the pivot selection is always bad, allowing you to always select the smallest or largest element), the time is $O(n^2)$.

Machine Learning Algorithm: Decision Tree Classifier

The time complexity of constructing the decision tree in the training depends on the number of splits or nodes that we need; total number of data points present in the data set; and the depth of the tree. This can be mathematically expressed as:$T_{Training}$ =O (d x m x log n) where: 'd' is the maximum depth of the tree, 'm' is the number of features in the dataset and 'n' is the data points in the dataset.

Once the decision tree is built during the prediction phase, the time complexity of classifying a new data point is relatively low denoted as: $T_{Prediction}$= O(d). Thus, the time complexity of the prediction is due to the depth of the decision tree, because each data point has to be classified by navigating from a root to leaf node by the algorithm. In addition to assessing a machine learning model based on time complexity, it is additionally evaluated by its performance on unseen data during the model evaluation. The way to quantify the model's ability to generalize is with metrics such as accuracy, precision, recall and F1 score. This is clear from above mathematical representations that there is huge difference in the efficiency analysis between the traditional algorithms and machine learning algorithms. While a machine learning algorithm can base solution on the training time and solution quality, a traditional algorithm considers only execution time.

3.2 Proof of Concept for Varying Time Complexity for Machine Learning Algorithm

Proof of concept (PoC) is a preliminary demonstration to verify the feasibility of an idea; which can also translate to a medical research course stage. It is used to test core principles in a controlled environment to demonstrate that in fact, the concept should have worked effectively in most cases before full-scale implementation for further investigation (Zhang et al., 2023). As part of this research work, we describe the Proof of Concept for our research objectives as stated in prior sections. The modelling and the PoC are designed to empirically validate the theoretical impacts of input size (N) and feature count (F) in time complexity of certain machine learning algorithms. This allows us to establish the general practice on how these factors affect processing time and then use these considerations to evaluate how scalable are different algorithms for the computational need.

We set up a controlled experimental environment using benchmark dataset with different sizes and different numbers of features in order to

demonstrate these effects. Both synthetic and real-world examples are provided for the same on these datasets. To show the effect of time complexity, we chose an algorithm with different theoretical complexity such as k nearest neighbours (k-NN) algorithm $O(N^2)$, Decision trees O(N log N) and neural networks O(N * F). In order to help generalize the results and provide solid proof of the influence of input size and feature dimensionality across a wide range of use cases, algorithms with different theoretical complexities were chosen. Thus, this diversity enables exploring how different patterns of computation evolve with size of input and number of dimensions of the features. Therefore we test each algorithm on datasets with more and more elements (N) to ascertain its empirical time complexity relative to the theoretical prediction. Separately, we vary the feature count (F) by artificially generating datasets of controlled feature count keeping the input size fixed. It allows us to separate and observe the exclusive effect of feature scaling on algorithm performance. Processing time is the first and only metric recorded for each experiment. Graphs of results are presented, illustrating how time complexity expands as more terms are added to search, these are compared to empirical curves as against theoretical expectations. We see that k-NN's processing time grows exponentially with input size (N), in line with its quadratic time complexity, based on preliminary findings. On the other hand, Decision Trees scale more moderately and possess O(N log N) as its time complexity.

When we focus on feature count (F), it is evident that, for instance, neural network algorithms which are highly complex in terms of F, exhibit significant time increases as F increases. At the same time, tree-based methods are more robust to F increases in high dimensional spaces. Results show that both input size (N) and feature count (F) substantially affect algorithm runtimes, consistent with theoretical time complexity predictions. For high dimensional data, applications of dimensionality reduction techniques may be needed to ensure that higher sensitivities to F algorithms are indeed more appropriate for high dimensional data. On the other hand, algorithms that are highly sensitive to N might need sampling of data if they are to be applied to large data sets.

These findings highlight the importance of N and F, viewing them together, and informing the selection of a particular algorithm within the machine learning application in light of a particular data set and performance needs. Practitioners can enhance prediction performance and optimize computational resources by combining both parameters, which is particularly important in settings with limited resources or time constraints. Our detailed observations on the Proof of Concept are presented in Table 2 and in Figure 2. In the following sections (3.2.1 and 3.2.2), we will demonstrate our claim that the time complexity will keep varying for a given machine learning algorithm depending on its input size and feature selection. We met this objective by executing the time complexity analysis of the KNN-ERP algorithm, which was developed in our previous research work (Parthasarathy & Padmapriya, 2023) to predict degree of ERP customization.

3.2.1 Input Size (N) and its impact on Time Complexity

The input size, represented by N, refers to the total number of ERP projects included in the dataset. The KNN algorithm must compute the distance between the new data point (the new ERP project) and each existing data point in the dataset. The time complexity for calculating the distance between the new point and one existing point is O (F), where F is the number of features. Because the distance calculation needs to be carried out for all N data points, the total time complexity for this step is O (N×F). Once the distances are calculated, the algorithm then sorts them to identify the K nearest neighbours. Sorting N distances generally has a time complexity of O (N log N). However, in practical scenarios where K is much smaller than N, the dominant term remains O (N×F).

For instance, imagine a scenario where the dataset contains 1,000 data points (N) and 10 features (F). The time complexity in this case would be calculated as:

Time Complexity=O (1000×10) = O (10000)

This demonstrates that the time complexity increases linearly as the input size N grows, assuming the number of features remains constant. This linear scaling suggests that unless optimization or approximation techniques are used, KNN may become computationally costly for very large datasets.

3.2.2 Number of Features (F) and its Impact on Time Complexity

The number of features, denoted as F, is another crucial factor influencing the time complexity of the KNN-ERP algorithm. Each feature represents a characteristic of the ERP projects such as the number of ARs, PRs, and DRs. The more features included, the more complex the distance calculations become. When the number of features increases, while keeping the number of data points N fixed, the time complexity still follows the relationship O (N×F). This means that increasing the number of features leads to a linear increase in the time required to calculate distances between data points. For instance, if N is fixed at 1000 moreover F increases to 20, time complexity becomes:

Time Complexity=O (1000×20) = O (20000)

This implies that the computational effort raised by the algorithm increases with the

dimensionality of the data. For illustrative purposes, we will consider the KNN-ERP customization algorithm done by the first author's work (Parthasarathy & Padmapriya, 2023). Table 2 gives illustrative examples

Table 2.0 Illustrative Examples of Time Complexity Analysis of KNN- ERP Algorithm

Scenario	Number of Data Points (N)	Number of Features (F)	Time Complexity Calculation	Resulting Time Complexity
Input Size	1000	10	O (1000×10)	O (10000)
	2000	10	O (2000×10)	O (20000)
Number of Features	1000	20	O (1000×20)	O (20000)
	1000	15	O (1000×15)	O (15000)

of time complexity analysis of KNN-ERP customization algorithm. The variation of the time complexity of the KNN-ERP algorithm with input size and number of features is depicted in Figure 2.

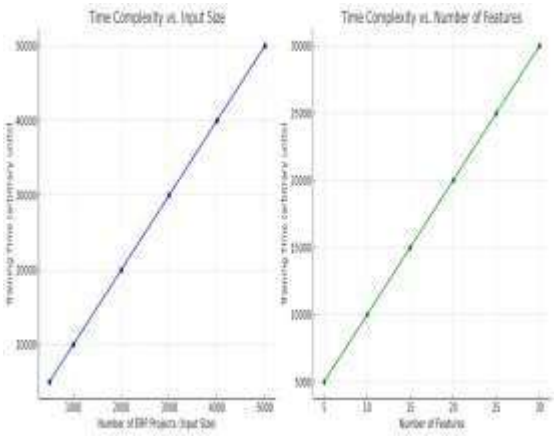

Figure 2. **Depiction of Time Complexity Variation in KNN-ERP Algorithm**

As shown in Figure 2, the left chart shows a high-level detail of the factors that determine the time complexity in the KNN-ERP algorithm, i.e. input size and number of features. In the first figure, we can see that the relationship between the time complexity and the input size is highly linear. The characteristic of the KNN algorithm is that computation time increases linearly with the number of data points, and therefore KNN is less efficient as the number of data points increases. Figure 2's right chart indicates the features that impacts time complexity. It is observed that efficiency of the KNN-ERP customization algorithm depends on both the size of the dataset and the dimensionality of the features (Ray, 2021). KNN is a simple and effective algorithm for low-dimensional datasets that are small but is non-scalable.

3.3 Mathematical Analysis of Time Complexity Variation in k-Nearest Neighbors (KNN) Algorithm

To generalize the above illustration showcasing the varying time complexity for machine learning algorithms namely KNN ERP customization algorithm, we now present below the mathematical representation and proof of concept for the same.

Time Complexity Analysis:

Step 1: Compute Distance

Distance Calculation: For each query point, the distance to each training point needs to be calculated. If we denote the number of training points by 'n' and the number of features by 'd', the distance calculation involves 'd' operations (typically O (d) for Euclidean distance).

Total Time for Distance Calculation: For each of the 'm' query points, the time to compute distances to all 'n' training points is O (n X d). Thus, for 'm' query points, this becomes O (m X n X d).

Step 2: Sort Distances

Sorting: Sorting 'n' distances for each query point has a time complexity of O (n logn).

Total Time for Sorting: For 'm' query points, this becomes O (m X n logn).

Step 3: Vote/Regression:

Voting or Computing Statistics: Finding the 'k' Nearest Neighbour involves selecting the first 'k' points from the sorted list, which takes O (k) time. Then, voting among the 'k' neighbours or computing a statistic typically takes O (k) time.

Total Time for Voting/Regression: For 'm' query points, this becomes O (m X k).

Combined Time Complexity: Combining the time complexities from all steps, we get: O (m X n X d) + O (m X n logn) + O (m X k)

Influence of Input Size and Feature Selection:

Input Size (n and m): The total training points are denoted as 'n' and the total query points 'm' directly affect the time complexity. Larger values of 'n' and 'm' increase the overall time complexity significantly.

Feature Selection (d): The distance computation step has an influence on the number of features 'd'. Increasing the number of features reduces the speed of each distance calculation.

Simplified Analysis: In this case, the term O (m x n x d) will usually predominate since 'd' is generally smaller than nlogn.

The time complexity of the KNN algorithm is dependent on the input size (training and query points and number of features) and hence, this leads to the time complexity of KNN algorithm.

Mathematical Representation: Thus depending on the number of training points, number of query points and number of dimensions in the feature space, time complexity of the KNN algorithm is as follows:

T (n,m,d) = O (m*n*d) + O (m*nlogn) + O (m*k)

From this formula, it is clear that the time complexity depends on the input size and the features to be selected.

3.3.1 Description

Below we provide a detailed explanation of the mathematical proof of time complexity variation in the K-Nearest Neighbors (KNN) algorithm.

Distance Calculation and Input Size: KNN algorithm is based on distance implementation between the query register point and all of the training points. Supposing there is a finite set of training points possessed by 'n' and a complementary set of query points of 'm'. Thus, for every query point, it was required to calculate distance to every training point which should take O(d) operation for every distance calculation where 'd' is the features. Therefore, the time taken for processing distance of all the 'm' query points is O (m*n*d). It is clear that more time is required for each distance computation as the number of training points (n) and query points (m) get larger and by the number of features (d).

Sorting Distances: The next step is to first compute the distances and then sort these distances to get the K nearest neighbours of each query point. The time taken to sort the 'n' distances, is O (nlogn). The total complexity of sorting for this is O (m x n x logn) for 'm' query points. This step is important in order to identify the Nearest Neighbour, which becomes costly with the increase in size of the training points. Another complexity level of logn is introduced by the logarithmic factor, though this is less influential for large 'n' than the linear terms, it greatly affects the total time complexity for large datasets.

Voting/Regression and Combined Complexity: The last process has to do with either voting amongst the k-nearest Neighbour for classification or computing the regression statistic. The complexity of each step is O(k) when it is done for one query point and hence becomes O(m X k) for 'm' number of the query points. When all these mentioned steps are summed up, the total time complexity of the KNN algorithm becomes O (m X n X d) + O (m X n logn) + m X k). This reinforces the fact that time taken to process k-NN increases with the growth in the number of training and query points and features. The primary term, usually O (m X n X d), indicates the great influence of feature selection on the execution time of the algorithm because the number of features boosts the cost of computations. Hence, the time complexity of KNN depends on the input size and features from nearest neighbours making its practical use in large and high-dimensional data difficult.

4. Discussion

The landscape of computational algorithms is vast, encompassing both traditional and machine learning algorithms, each with unique characteristics and performance implications. Understanding the time complexity of these algorithms is crucial for effective implementation, optimization, and achieving the desired outcomes in various applications. Traditional algorithms, often characterized by deterministic processes, have well-defined steps that lead to a solution. Their time complexity is usually assessed based on input size and can be classified into different complexity categories, such as O (1), O (log n), O (n), O (n log n), O (n^2), and others. For example, binary search has a time complexity of O (log n), making it effective for searching within sorted arrays. Similarly, sorting algorithms like quicksort and merge sort typically have an average time complexity of O (n log n), striking balance between performance efficiency and implementation complexity. Traditional algorithms tend to excel in situations where the problem is well-defined and the input size is manageable. However, as input size grows, the limitations of traditional algorithms become apparent, especially for those with polynomial or exponential time complexities.

Machine learning algorithms bring a different aspect to time complexity analysis. These algorithms learn patterns from data and use the learned model to make predictions or decisions. The time complexity of machine learning algorithms can be broken down into two phases: training and inference. In the training phase, the algorithm processes input data to uncover underlying patterns. For instance, the training complexity of a k-Nearest Neighbors (k-NN) algorithm is O (m × n × d) + O (m × n log n), where 'm' is the number of query points, 'n' is the number of training points, and 'd' is the number of features. Similarly, training a Support Vector Machine (SVM) involves solving a quadratic optimization problem, resulting in a time complexity of O (n^2 × d) in its primal form. The training phase is often computationally demanding, especially with large datasets and high-dimensional feature spaces. However, since this phase is typically conducted offline, there is room for using extensive computational resources and time to optimize the model.

In contrast, the inference phase, where the trained model is utilized to make predictions, typically has a lower time complexity. For k-NN, inference involves computing distances to the training points, leading to a complexity of O (n X d) per query point. For a trained neural network, inference is often O (d), as it involves a fixed number of operations determined by the network architecture. The primary distinction between traditional and machine learning algorithms lies in their approach to problem-solving and the resulting time complexities. Traditional algorithms, with their deterministic nature, offer predictable performance but struggle with scalability and adaptability to new data. Machine learning algorithms, on the other hand, excel in adaptability and handling complex, high-dimensional data but come at the cost of high training time complexities. In practical applications, the choice between traditional and machine learning algorithms often hinges on the problem requirements. For problems with well-defined rules and manageable input sizes, traditional algorithms are preferable due to their predictable performance. Conversely, for complex problems involving large datasets and requiring adaptive learning, machine learning algorithms are more suitable despite their higher training complexity. The interplay between input size, feature dimensionality, and time complexity underscores the importance of understanding the computational demands of both traditional and machine learning algorithms. This understanding enables practitioners to make informed decisions, optimizing algorithm selection and implementation for efficient and effective problem-solving.

4.1 Contributions to Research and Practice

In this research work, we introduce a systematic comparison of time complexity between traditional algorithms and machine learning (ML) algorithms, establishing a technical novelty in how algorithmic efficiency is evaluated in data-driven contexts. Unlike conventional studies that assess algorithms in isolation, this research analyzes time complexity across both traditional and ML paradigms, leveraging empirical and theoretical insights. One key contribution is the proof of concept for studying the variation in time complexities of traditional and ML algorithms. Through this, we highlight the scalability limitations and performance bottlenecks specific to each paradigm under varying data conditions.

Another contribution is the mathematical analysis of time complexity variation in ML algorithms, a focused analysis of how ML algorithms' time complexity shifts under different feature and dataset dimensions. By modeling time complexity using mathematical formulas, this section offers a precise, quantitative understanding of how ML models respond to increased data dimensionality and complexity. This analysis is especially novel, as it underscores the role of both feature engineering and data preprocessing in computational efficiency, areas often under-explored in ML research. The insights from this study hold practical value for practitioners who must navigate trade-offs between computational resources and accuracy. For developers and data scientists, the findings serve as a guide to choose algorithms that meet both performance and scalability demands, particularly in environments with high data variability.

This research work encourages the adoption of hybrid approaches, where traditional algorithms could be combined with ML techniques to optimize performance. For instance, preprocessing tasks with traditional algorithms might reduce dimensionality, making ML algorithms more efficient when processing large datasets. These insights support practical applications in areas such as real-time data processing, AI-driven analytics, and high-performance computing, where computational efficiency is crucial. By bridging theoretical insights with practical analysis, our study offers a comprehensive approach for evaluating algorithm efficiency, guiding both academic research and real-world implementations across diverse data-driven fields.

5. Conclusions

The main conclusions are as follows. The underlying principle of efficiency analysis of the machine learning algorithms must be reset. The time complexity of such algorithms depends on feature selection, training dataset volume, and input size. The time complexity of solving the same problem may vary depending on the machine learning algorithm used, such as linear regression or decision trees. In addition,

the training dataset may be added in real time, making a priori algorithm analysis unhelpful. In such cases, an empirical (posteriori) algorithm analysis must always be used. Given these facts, representing the time complexity of such algorithms as a function of input size makes little sense.

5.1 Future research

Future research shall explore a broader range of real-world applications to validate and refine the observed trade-offs in diverse computational environments. Investigations could focus on hybrid models that combine traditional algorithmic logic with machine learning components to optimize both accuracy and efficiency. Additionally, extending the analysis to include space complexity, energy consumption, and scalability on edge devices or cloud platforms would provide a more holistic view. Exploring the role of explainability and robustness in selecting between algorithmic and ML-based approaches, especially in safety-critical domains, also presents a valuable direction for further inquiry.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Author Contributions

Conceptualization, methodology, and analysis: **Sudhaman Parthasarathy;** Writing—original draft preparation: **Sudhaman Parthasarathy; S. Sivagurunathan:** Validation and Writing [review and editing]: All authors read and approved the final manuscript.

Conflict of interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

Ali, Y. A., Awwad, E. M., Al-Razgan, M., & Maarouf, A. (2023). Hyperparameter search for machine learning algorithms for optimizing the computational complexity. Processes, 11(2), 349.

Assis, A., Dantas, J., & Andrade, E. (2025). The performance-interpretability trade-off: a comparative study of machine learning models. Journal of Reliable Intelligent Environments, 11(1), 1.

Baki, A. (1992). Al Khwarizmi's Contributions to the Science of Mathematics: Al Kitab Al Jabr Wa'l Muqabalah. Journal of Islamic Academy of Sciences, 5(3), 225-228.

Berry, M. W., Mohamed, A., & Yap, B. W. (Eds.). (2019). Supervised and unsupervised learning for data science. Springer Nature.

Binson, V. A., Thomas, S., Subramoniam, M., Arun, J., Naveen, S., & Madhu, S. (2024). A review of machine learning algorithms for biomedical applications. Annals of Biomedical Engineering, 52(5), 1159-1183.

Dahiya, S. (2023). Scalable Machine Learning Algorithms: Techniques, Challenges, and Future Directions. MZ Computing Journal, 4(1).

Huang, L., Yin, Y., Fu, Z., Zhang, S., Deng, H., & Liu, D. (2020). LoAdaBoost: Loss-based AdaBoost federated machine learning with reduced computational complexity on IID and non-IID intensive care data. Plos one, 15(4), e0230706.

Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2021). A survey on bias and fairness in machine learning. ACM Computing Surveys (CSUR), 54(6), 1-35.

Nguyen, T. T., Nguyen, N. D., & Nahavandi, S. (2020). Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. IEEE transactions on cybernetics, 50(9), 3826-3839.

Parthasarathy, S., & Padmapriya, S. T. (2023). Understanding algorithm bias in artificial intelligence-enabled ERP software customization. Journal of Ethics in Entrepreneurship and Technology, 3(2), 79-93.

Ray, S. (2021). An analysis of computational complexity and accuracy of two supervised machine learning algorithms—K-nearest neighbor and support vector machine. In Data Management, Analytics and Innovation: Proceedings of ICDMAI 2020, Volume 1 (pp. 335-347). Springer Singapore.

Shrestha, A., & Mahmood, A. (2019). Review of deep learning algorithms and architectures. IEEE access, 7, 53040-53065.

Zeguendry, A., Jarir, Z., & Quafafou, M. (2023). Quantum machine learning: A review and case studies. Entropy, 25(2), 287.

Zhang, Z., Chen, J., & Zhao, W. (2023). Multi-AGV route planning in automated warehouse system based on shortest-time Q-learning algorithm. Asian Journal of Control.

Zheng, X., Jia, J., Guo, S., Chen, J., Sun, L., Xiong, Y., & Xu, W. (2021). Full parameter time complexity (FPTC): A method to evaluate the running time of machine learning classifiers for land use/land cover classification. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 14, 2222-2235.

Zhou, Z. H. (2021). Machine learning. Springer Nature.