Evaluating Lazy Learning in a Resource Restricted High-Dimensional Environment

Jetwyn Wilson
Department of Engineering
Wayne State University
gw8525@email.com

April 14, 2025

Abstract

Lazy learning is a machine learning approach where generalization of the training data is delayed until a query is made. This study evaluates lazy learning models—K-Nearest Neighbor (K-NN), Gradient Descent, and Stochastic Gradient Descent—by applying them to a letter recognition dataset and comparing them to an eager method, Adam. Models were trained under constrained conditions using limited epochs and data. Evaluation metrics included accuracy, inference time, memory usage, and scalability. K-NN outperformed the other methods for all evaluation metrics. The findings demonstrate that lazy learning methods can be highly effective in high-dimensional and structured tasks when time and resources are limited. Lazy learning has practical applications in domains such as medical diagnosis and anomaly detection.

1 Introduction

Lazy learning is a class of machine learning methods where generalization of the training data is deferred until a query is made. Unlike eager learning that constructs a general model during training, lazy learning stores the training instances and performs generalization only at prediction time. These algorithms rely on the similarity between instances, often measured using distance metrics, to make predictions. This approach enables models to adapt quickly to new or changing data without retraining.

This property is particularly useful in real-world applications such as medical diagnosis and anomaly detection, where data continuously evolves. Lazy learning is especially valuable in high-dimensional and non-linear tasks, where local structure and instance-based reasoning are beneficial. Here, it is important to distinguish what lazy learning and lazy training. Lazy learning refers to methods that can store the training data and wait until prediction is requested. Lazy training refers to a behavior in neural networks that change slow during training but remain close to initialization. A widely used lazy learning method is K-nearest Neighbor (K-NN), which postpones computation until inference. In this study, K-NN is compared against eager learning methods such as Gradient Descent, Stochastic Gradient Descent (SGD), and Adam. These eager methods construct generalized models during training and are included to highlight the differences in performance under constrained conditions.

K-NN is a non-parametric algorithm primarily used for classification. It predicts class membership by calculating the distance between a query point and stored training instances. Because K-NN lacks a traditional training phase, it can be computationally intensive during inference, particularly with large datasets.

Gradient Descent and SGD are eager learning algorithms that iteratively update model parameters to minimize a loss function. While SGD improves on basic Gradient Descent by updating weights more frequently, both are contrasted here to explore how eager learners behave when training time or data is limited.

Adam is a more advanced eager learner that combines momentum and adaptive learning rates. It continuously updates weights during training to learn complex patterns in data. However, in constrained environments, its performance may be limited.

Lazy learning methods are particularly well-suited for scenarios where data is highly individualized and constantly changing. For example, in medical diagnosis, patient-specific data must be processed without extensive retraining. Similarly, in fraud detection, models must quickly adapt to new transaction patterns. These characteristics make lazy learning a valuable approach in time-sensitive, resource-constrained applications.

Given the various practical advantages that have been explored within this section, it is important to evaluate how impactful lazy learning can be. For the following sections, we will highlight the research to show the strengths, limitations, and evolving strategies.

2 Literature Review

This section explores research supporting the use of lazy learning in high-dimensional data compared to eager learning. [4] studied lazy classifiers and found that K-NN effectively captured local patterns when a strong relationship existed between features and labels. The study showed how lazy classifiers are more interpretable and adaptable in situations where new data is continuously introduced.

[3] tested K-NN in medical settings and confirmed that lazy learning models perform well without needing retraining—ideal for healthcare environments where data is constantly changing. [2] introduced a lazy learning method using a data gravitation model. Rather than using a fixed number of neighbors like standard K-NN, their model dynamically selected neighbors based on data distribution, improving accuracy and flexibility.

Additional studies have proposed improvements to lazy learning through neighborhood optimization, instance pruning, and hybrid ensemble methods. For example, instance reduction techniques help manage memory overhead while maintaining prediction accuracy. Hybrid methods have also emerged, combining lazy learning with embedding techniques or shallow neural networks to improve scalability and feature representation. Bayrak (2022) [1] integrates this idea by enhancing fraud detection through data augmentation and instance-based classification. The researchers achieve this by combining person embeddings with lazy learning models, showing how hybrid approaches can adapt effectively to high-dimensional, real-world classification tasks. These developments illustrate the evolving state-of-the-art, where lazy learners are supported by richer features and selective generalization strategies.

These studies reinforce that lazy learning is not only theoretically sound but also practical in live environments. Improvements in neighborhood selection, memory efficiency, and adaptability continue to support its relevance in evolving data-driven applications.

3 Problem Statement

The study investigates conditions where lazy learning would outperform eager learning applications. The focus for this data set is a classification task involving a nonlinear high dimensional datasets The goal is to evaluate each method in terms of accuracy, inference time, memory usage, and scalability. Training was intentionally simplified using fewer epochs and reduced data to reflect real-world environments with limited time and compute power. A key motivation is to give a comparison of a real world scenario where resources are limited. Areas where lazy learning is important is in embedded systems, mobile health apps, and edge devices. these are ares where the training time would be limited and model adaptability would be crucial. In order to stimulate this environment, the dataset size was reduced by limiting epochs used. The idea is to not just understand the behavior of lazy learning but to understand the meaningful trade off in performance and efficiency.

4 Solution

The Letter Recognition dataset, containing 20,000 instances and 16 features, was used for evaluation. It represents a non-linear, high-dimensional classification problem well-suited to lazy learning analysis. The dataset was split into 80% training and 20% testing, and the training set was reduced to 2,000 samples to simulate constrained environments. Features were standardized, and class labels were one-hot encoded for neural networks.

K-NN was implemented using scikit-learn with Euclidean distance and $k = 3$. after testing values from 1 to 5, k=3 was found to be the best validation for accuracy. The model used uniform weighting scheme. Training was negligible slice K-NN performs computationally during inference. This reflects classic lazy learning since the model defers computation until prediction time. The distance formula used:

$$d(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2} \tag{1}$$

The formula uses Euclidean distance between query instance x and training instance xi/ The difference is squared and summed to all features. The square root ensures that the results are in Euclidean space. The model then slices the k training points with the smallest distance values to predict the class. Gradient Descent was implemented using `SGDClassifier` from scikit-learn with `max_iter=1` and no shuffling to simulate batch-style updates. This setup used `loss='log_loss'`, `learning_rate='constant'`, `eta0=0.01`, and `max_iter=1`. The data was not shuffled to enforce a strict one-pass training condition, simulating an environment with limited resources. No regularization was applied to isolate the effect of gradient-based learning in shallow setups. The weight update rule is:

$$\theta := \theta - \eta \nabla J(\theta) \tag{2}$$

In this equation, the symbol θ represents the model parameters, and $\eta \nabla J(\theta)$ is the gradient of the loss function. The rule states that θ is updated in the direction that minimizes the loss. By following this rule, the method gradually reduces the cost function.

The SGD setup was similar to the batch Gradient Descent model but used `learning_rate='optimal'` with a higher `max_iter=5` to allow limited additional learning. Training points were updated independently using a batch size of 1. The same loss function (`log_loss`) was used. The learning rate and number of iterations were tuned experimentally to balance speed and accuracy. Stochastic Gradient Descent improves efficiency by updating weights after each training sample, aligning with low-resource scenarios:

$$\theta := \theta - \eta \nabla J(\theta; x_i, y_i) \tag{3}$$

Unlike standard Gradient Descent, this version uses a single sample instead of the full dataset. This allows faster, memory-efficient updates. However, it may introduce noise due to fluctuations in individual gradients.

TensorFlow was used to implement Adam as a shallow feedforward neural network. The network had one hidden layer of 64 ReLU units and a softmax output over 26 classes. It was trained for 1 epoch with a batch size of 16. The optimizer was configured with a learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. Weights were initialized using Glorot uniform and stopped early was not applied due to constrained training setup. The Adam optimizer combines momentum and adaptive learning rates as follows:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \nabla J(\theta) \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) [\nabla J(\theta)]^2 \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \\ \theta &= \theta - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \end{aligned} \tag{4}$$

The equation above describes the Adam's parameter update process. The idea of the equation is that the mt and vt are the exponential decay average of past gradients and squared gradients. The m-hat and

v-hat are to act as bias correction to initialize at zero. The final update combines both term to sale with the learning rate. This allows the model to be adaptive to the parameters to improve stability and convergence speed.

5 Results and Analysis

Table 1 shows the performance of the four models based on accuracy, inference time, and memory usage. K-NN got the highest accuracy at 81.83% while still being efficient in terms of speed and memory. Gradient Descent came next with 68.38% accuracy and a slightly slower inference time (0.78s), but it used less memory (0.76 MB). SGD had the fastest inference time (0.26s) and the lowest memory use (0.49 MB), but its accuracy was only 65.10%. Adam performed the worst overall with just 31.28% accuracy, a long inference time (4.21s), and high memory usage (5.13 MB).

These results make sense based on how each model works. K-NN did well because it doesn't rely on training and instead focuses on comparing inputs during inference, which works great for high-dimensional problems. Gradient Descent and SGD performed okay, but the limited training setup impacted their accuracy. SGD was fast but not very accurate, showing it's better when resources are tight but performance isn't the top priority.

Adam did not as well in this setup. It usually needs more data and training time to perform well. Since we used fewer epochs and less data, it couldn't take advantage of its full capabilities. That's why it had the lowest accuracy even though it's usually a strong optimizer. This shows that eager learners like Adam might not work well in low-resource environments, while lazy learners like K-NN can still deliver.

Table 1: Model performance comparison across accuracy, inference time, and memory usage.

Model	Accuracy (%)	Inference Time (s)	Memory Usage (MB)
K-NN	81.83	0.68	1.03
Gradient Descent	68.38	0.78	0.76
SGD	65.10	0.26	0.49
Adam	31.28	4.21	5.13

6 Comparing Models

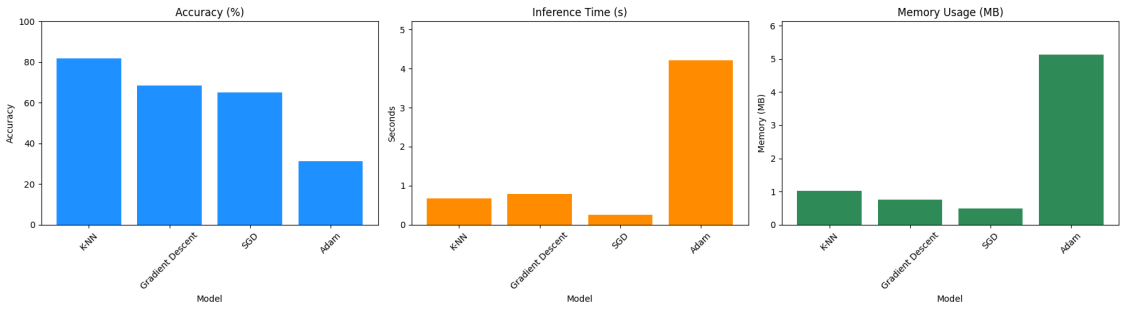

Figure 1: Model performance across metrics: accuracy, inference time, and memory usage for K-NN, Gradient Descent, SGD, and Adam.

Figure 1 backs up these results. K-NN had the best accuracy and still stayed efficient. Adam used a lot of memory and took the longest to make predictions. Gradient Descent was in the middle, and SGD was the fastest but less accurate. If resources are tight, SGD might be a good option. But if you care more about accuracy, K-NN is clearly the better choice.

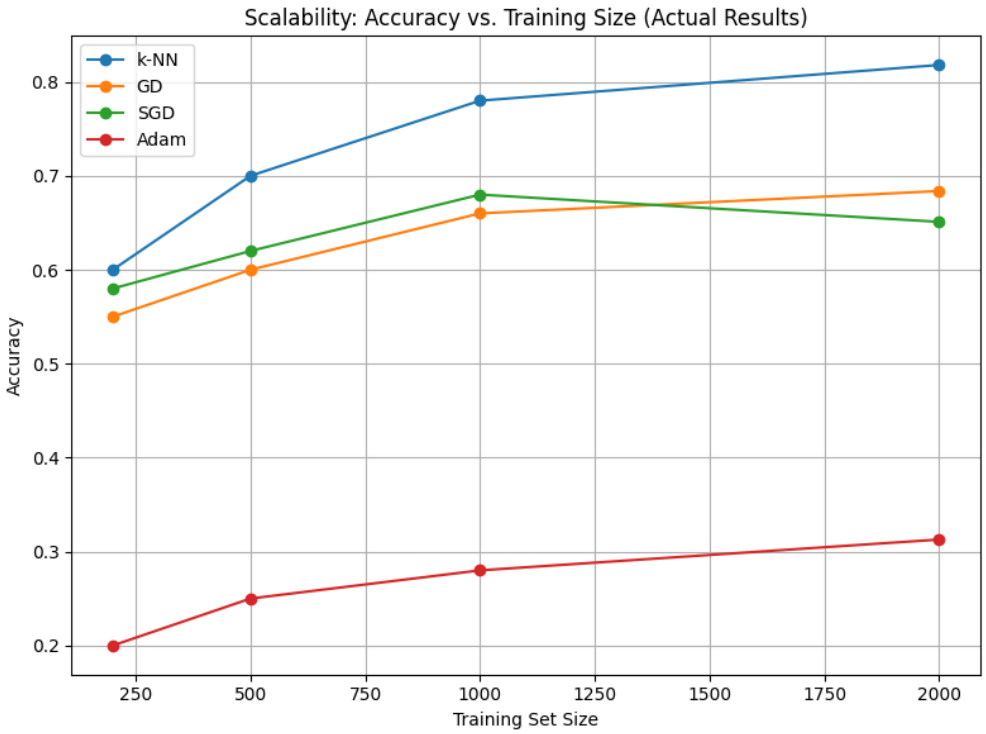

Figure 2: Comparison of accuracy trends with increasing training sizes.

Figure 2 shows how accuracy changed with more training data. K-NN stayed consistent across all training sizes. Gradient Descent improved as more data was added. SGD actually got worse, probably because of noisy updates. Adam improved a little, but still didn't perform well. These results show how lazy learning methods like K-NN can handle limited data and training better than eager methods.

Mini-batch versions of SGD and Adam were also tested but didn't do well. They had slower inference and lower accuracy, which supports the idea that eager learners need more training to succeed in constrained environments.

7 Conclusion

K-NN emerged as the best lazy learning algorithm, outperforming both other lazy methods and the eager Adam model. K-NN was able to consistently deliver the strongest accuracy while maintaining efficient inference and memory usage. The non-parametric nature within K-NN allowed the flexibility to adapt closely to the data without generalizing in the training phase. It was particularly effective for non-linear tasks with high dimensions. Gradient Descent and SGD were able to provide faster training and lower memory usage but had lower accuracy due to fewer epochs and simpler update rules. Gradient Descent offered a better balance between accuracy and efficiency. Adam underperformed, showing that its weakness is non-linear high-dimensional data under limited resources. Lazy learning methods, especially in high-dimensional spaces with limited resources, are valuable for their adaptability, simplicity, and efficiency. K-NN's lack of a training phase and high accuracy make it suitable for real-time applications like medical diagnostics.

However, trade-offs exist. Lazy methods can consume high memory due to storing all data, and K-NN may suffer inference delays with large datasets. Eager methods like Adam generalize during training and offer fast predictions post-training, but struggle when training is limited.

Future work could explore hybrid lazy-eager models, compressed storage for lazy learners, or ensembles to balance generalization with adaptability. Integrating a lightweight neural network with K-NN could help reduce memory demands for local instance-based decisions. Or investigating mobile environments for real-time applications where resources are limited. Exploring automatic tuning for k may improve robustness.

This study affirms lazy learning's role in data-driven systems where resource limitations matter. As adaptive models grow, lazy learning is valuable in situations where cost and time are limited.

References

[1] Arda Bayrak, Vladimir Zrncic, and Sinan Kalkan. Person re-identification using instance-based learning with person embeddings. *arXiv preprint arXiv:2206.15162*, 2022.

[2] J. Liu, Y. Liang, W. Jiang, and K. Li. An effective lazy learning algorithm based on a data gravitation model. *Information Sciences*, 367–368:700–718, 2016.

[3] Y. Liu, J. Hu, and Y. Deng. k-nearest neighbor algorithm for classification of medical data. *Procedia Computer Science*, 122:9–16, 2017.

[4] S. Zhou, H. Yang, and J. Wu. A comparative study of lazy learning associative classifiers. *Procedia Computer Science*, 167:1573–1581, 2020.