Evaluating the Impact of Randomized Sampling on Training of Diverse Prediction Models

Anonymous Authors[1]

Abstract

In the modern world, data generation and collection have become ubiquitous, and machine learning algorithms are constantly employed to implement intelligent models for a diverse range of practical use-cases. Large corporations leverage customer data to predict consumption patterns. Health practitioners utilize medical datasets to diagnose sickness. Social media companies parse through *likes* and *comments* to analyze human behavior and maximize user engagement. The reliance on big-data has increasingly become the norm. However, as the available data grows in size and dimensionality, the computational power required to train such models increase proportionally. Therefore, optimizing compute efficiency when leveraging large-scale datasets to train machine learning prediction models becomes an obvious necessity as society shifts toward data-driven business models and paradigms. This study focuses on exploring **data sampling** as a possible solution for the problem of training prediction models on large data. The goal, given a dataset, is to derive a smaller subset that retains the main characteristics of the whole, allowing for the training of prediction models that provide comparable accuracy to those trained on the full dataset, however at a lower training cost.

1. Introduction

Logistic regression is a supervised learning technique commonly used to model the probability of an event occurring based on predictor variables. In the field of Machine Learning, logistic regression is often applied in the context of disease classification, anomaly detection, among others.

Despite being a well established and effective method for binary classification use-cases, handling large-scale datasets often present unique computational and algorithmic challenges, particularly in terms of time and space complexity, that demand specialized solutions. For example, logistic regression models typically employ optimization methods such as gradient descent to find the best-fitting parameters. The computational cost of each iteration is proportional to the number of observations and the number of features, often times making the optimization procedure computationally expensive.

With large datasets, the time complexity can become prohibitive, as the number of operations required for each iteration increases. Additionally, many logistic regression implementations require storing the data-points and intermediate computation results in memory. When working with big-data, this approach can lead to hardware bottlenecks. In particular, storing and manipulating large feature matrices can demand significant amounts of random-access memory (RAM), especially when the number of features is substantial. When the dataset exceeds memory limits, performance can degrade significantly, requiring the use of additional tooling, such as distributed computing or data sampling, as a means to overcome such challenges.

In this work, we explore the impacts of randomized sampling as an optimization strategy focused on addressing some of the challenges posed by high-dimensional datasets, specifically when the database size (e.g., number of records) is excessively large. The goal is to provide a comprehensive analysis rooted in extensive empirical observations made with a diverse collection of datasets and prediction methods, and evaluate whether the algorithms outlined by this study are effective in providing benefits when training machine learning prediction models.

2. Related Work

2.1. Randomized Sampling

The application of sampling algorithms to optimize training performance and manage dataset reduction has been a persistent area of research in machine learning, particularly in logistic regression and other high-dimensional classification problems. In recent studies, various methodologies have been proposed, including sophisticated leverage-based sam-

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under pper review. Do not distribute.

pling techniques. For example, Chowdhury & Ramuhalli introduce a randomized sampling algorithm specifically designed for logistic regression scenarios. Their approach leverages row-wise leverage scores to strategically select a subset of the data that maintains key dataset characteristics. The method provides theoretical guarantees for achieving high-quality approximations of the logistic regression model parameters, significantly reducing computational overhead while preserving predictive accuracy. This leverage-based approach contrasts with simpler methods, such as uniform sampling, by prioritizing influential data points that contribute most significantly to model accuracy.

2.2. Signal Strength Leave-One-Out Estimator (SLOE)

Building on methods aiming to enhance efficiency in high-dimensional logistic regression, Yadlowsky et al. proposed the Signal Strength Leave-One-Out Estimator (SLOE). This method addresses challenges inherent in high-dimensional scenarios where traditional logistic regression estimators, like maximum likelihood estimation (MLE), typically perform poorly. The SLOE algorithm efficiently estimates a crucial signal strength parameter using a computationally efficient leave-one-out strategy, significantly outperforming prior heuristics in terms of both speed and accuracy. The authors validate their method through extensive simulations, highlighting its improved computational feasibility and its reliability in correcting biases and variance. The practical significance of SLOE is evident in high-dimensional logistic regression applications such as genomics and clinical diagnostics, where reliable statistical inference and rapid computation are both critical.

2.3. Sample Reweighted Decorrelation Operator (SRDO)

Addressing related stability concerns in training scenarios with mismatched data distributions, Shen et al. introduced the Sample Reweighted Decorrelation Operator (SRDO). SRDO targets model instability caused by collinearity, a common issue in high-dimensional linear and logistic regression models. By intelligently reweighting samples, SRDO effectively reduces correlations among predictor variables, resulting in a design matrix that approximates orthogonality. This adjustment significantly mitigates instability and improves predictive accuracy when training and test datasets differ substantially in their underlying distributions. SRDO thus represents a complementary approach to dataset reduction techniques by ensuring robustness and stability in model predictions, particularly in environments prone to distribution shifts.

2.4. Main Takeaways

While these methods focus primarily on logistic regression and linear models, broader applicability across various predictive algorithms remains an open area for exploration. The leverage scores sampling method proposed by Chowdhury & Ramuhalli, for instance, has not yet been thoroughly tested on non-linear models such as support vector machines, decision trees, or deep neural networks, where the composition of training data can profoundly impact model outcomes. Similarly, although SLOE has demonstrated significant efficiency gains in high-dimensional logistic regression, further research could explore its performance under non-Gaussian distributions and compare its effectiveness against traditional dimensionality reduction techniques, like principal component analysis or feature selection. The literature demonstrates a clear trend towards optimizing computational efficiency and robustness in high-dimensional regression problems through innovative sampling and reweighting strategies.

3. Problem Definition

The rapid growth in data volume has significantly impacted the computational demands placed on machine-learning-based systems, especially those requiring iterative parameter estimation methods, such as logistic regression. Large-scale datasets pose considerable challenges, including prolonged training times, increased resource consumption, and potential reductions in model interpretability and manageability. Consequently, there is a critical need to develop effective dataset reduction strategies that maintain the predictive accuracy of machine learning models while drastically improving computational efficiency.

This research explores solutions to the challenge of reducing the dataset size required for logistic regression and other classification models without incurring substantial losses in accuracy or stability. Traditional methods such as uniform random sampling, though computationally efficient, typically do not account for the inherent structure and importance of individual data points, potentially compromising model accuracy. Conversely, sophisticated sampling approaches leveraging data-driven metrics (e.g., leverage scores) offer theoretical guarantees for accuracy preservation but introduce complexity and additional computational overhead. Therefore, the primary research problem explored in this study is to systematically evaluate and quantify the trade-offs between computational complexity, training efficiency, and predictive performance when employing sampling algorithms with the specific goal of dataset reduction. This study investigates the efficacy of data-sampling strategies, determining their relative advantages and limitations across multiple classification algorithms, primarily logistic regression, but also support vector machines, K-nearest

neighbors, and random forest classifiers.

Through comprehensive experimental analysis, this work seeks to answer a key question: To what extent can dataset reduction be achieved without significant accuracy degradation? Addressing this central question will enable a deeper understanding of dataset reduction techniques, providing valuable guidance for practitioners and researchers aiming to optimize the training processes of various classification models, particularly within the logistic regression framework.

4. Methodology

4.1. Goals

This study aims to explore potential solutions to the computational challenges posed by the use of high-dimensional datasets in the context of prediction model training. Objectively, we evaluate the impacts, implementations, and viable utilizations for randomized sampling algorithms originally designed for logistic regression models. Given a dataset, the goal is to derive a smaller subset that retains the most representative characteristics of the data, allowing the subset to be used for training prediction models capable of achieving accuracy levels comparable to models trained on the entire dataset. We aim to investigate, both theoretically and experimentally, whether the algorithms outlined in this study are successful at reducing the size of training datasets while preserving representativeness of the original data. If successful, this method can present a viable strategy for addressing many barriers faced when handling large databases in machine learning applications.

4.2. Key Metrics

To evaluate the impacts of the sampling algorithms in the training of prediction methods, a thorough benchmark is necessary to evaluate performance of models trained with sampled datasets relative to those trained on full datasets. To keep comparisons objective, we will focus on select metrics such as **accuracy** and **training time**. Formally:

$$Accuracy(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} 1(\hat{y}_i = y_i)$$

$$Training\ Time = s + t$$

Where:

- y is a vector representing the true values.
- $\hat{y}$ is a vector representing the predictions.
- n is the number of predictions.
- s is the time consumed by the sampling algorithm. This is $zero$ for non-sampled models.
- t is the time consumed by the training process.

Moreover, each algorithm-dataset combination is run an adequate number of times for statistical relevance, and the key metrics are to be captured each time. Performance comparisons will be made with average values. This practice will allow for more consistent performance analysis across implementations. Specifically, the sampling algorithms will be tested with well-known open-source datasets, such as the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993), the Iris Dataset (Fisher, 1936) among others. A comprehensive and detailed review of each experiment conducted is included in **Appendix A**.

Lastly, we will assess whether the sampling algorithms contribute positively or negatively when used to train different prediction models beyond binary classification.

4.3. Baseline Setup: Slow Logistic Regression

In this study, we evaluate the impacts of a simplified version of the sampling algorithm proposed by Chowdhury & Ramuhalli (2024), as well as an algorithm for true random sampling with uniform probability distribution, used for comparative analysis. These sampling algorithms are evaluated across a diverse variety of prediction models and databases, however, we here are primarily interested in assessing their applicability for predictions performed by **logistic regression** models.

To set a performance baseline for our analysis, we start with our own implementation of the logistic regression algorithm, which is provided in Python code by class `SlowLogisticRegression`. It provides an abstraction that allows us to create a logistic regression prediction model.

By design, `SlowLogisticRegression` employs a simple implementation strategy, which is intentionally not optimized. The rationale behind this decision is that a slower model would allow for better visualization of the impacts of the sampling algorithms explored here, which are the focus of this work. In other words, we are interested in measuring the extent to which the sampling algorithms improve (or degrade) a logistic regression model that does not implement any other optimizations. This approach will allow us to isolate the impacts caused specifically by the sampling algorithms.

4.3.1. Training Data

The model expects an input of training data in the following format:

- **Features (X):** A matrix of input features where each row represents a data point and each column represents a feature (e.g., a matrix of shape $n \times m$, where n is the number of data points and m is the number of features).
- **Labels (y):** A vector of length n with the target binary labels (0 or 1), corresponding to the input features.

4.3.2. Initialization

Model parameters are initialized as follows:

- **Weights (θ):** The weights (coefficients) of the model, here initialized as zeros. These weights are of size m (one for each feature).
- **Bias (b):** A scalar value added to the output of the linear combination of the features, here initialized to zero.

4.3.3. Model Hypothesis

A linear combination (linear model) is implemented for the inputs and weights.

- **Linear Combination:** For each data point, we compute the weighted sum of the features plus the bias term.

$$z = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_m x_m + b$$

- **Sigmoid Function (Logistic Function):** This function maps the output of the linear combination to a probability between 0 and 1, which represents the probability of the positive class (prediction being $true$). We apply the sigmoid function to the linear combination z to obtain the predicted probability:

$$\hat{y}(z) = \frac{1}{1 + e^{-z}}$$

4.3.4. Loss Function

The loss function used in our logistic regression implementation is the *binary cross-entropy*, which measures the difference between the predicted probabilities versus the actual (expected) values. For a dataset of n examples, the cost function $J(\theta)$ is given by:

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

Where $y^{(i)}$ is the true label and $\hat{y}^{(i)}$ is the predicted probability for each data point.

4.3.5. Optimization

The goal of logistic regression is to find the optimal values for the weights and bias that minimize the loss function. In this work, this is done using *gradient descent*, which goes as follows:

1. Compute the gradients (partial derivatives) of the loss function with respect to each parameter (weights and bias):

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{y}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

$$\frac{\partial J(\theta)}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{y}^{(i)} - y^{(i)} \right)$$

2. Update the weights and bias using the gradients, (α is the learning rate, which controls the step size of each update):

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

$$b := b - \alpha \frac{\partial J(\theta)}{\partial b}$$

3. Repeat until the loss converges (i.e., the change in the cost function between iterations is small enough) or a predefined number of iterations is reached.

4.3.6. Model Evaluation

We evaluate the model's performance by comparing the predicted labels against the actual values. Common metrics include accuracy, precision, recall, F1 score, among others.

4.3.7. Output

After training, the learned weights and bias can be used to make predictions on new data. The output is a probability, but for classification, a **decision boundary** (commonly 0.5) is applied to convert the probability to a binary class label:

$$\hat{y} = \begin{cases} 1 & \text{if } \hat{y} \geq 0.5 \\ 0 & \text{if } \hat{y} < 0.5 \end{cases}$$

4.4. Leverage Scores Random Sampler

4.4.1. Overview

At a high level, the goal of the sampling algorithm proposed by Chowdhury & Ramuhalli is to derive a sampled subset that efficiently approximate the full dataset. This is achieved by performing Singular Value Decomposition (SVD) on the data matrix X to calculate leverage scores, then using it to sample a subset of the data. Finally, the sampled data is

used to train a logistic regression model, which is expected to closely replicate the performance of a model trained on the entire dataset.

To better analyze the algorithm and assess its transportability to different prediction models, we encapsulate our first implementation attempt, which is provided in Python code, in class `BasicLeverageScoresSampler`. This class represents the first iteration of the randomized sampling algorithm on which this study is focused. However, this version **simplifies** the paper's approach. Such simplification will be useful later when benchmarking different implementations and analyzing trade-offs.

4.4.2. Sampling Method

To estimate the relevance of each data-point in the dataset, the Singular Value Decomposition (SVD) method is applied. The matrix X is decomposed into its singular vectors and values, and the leverage score for each dataset entry (row in the matrix) is calculated as the sum of the squares of the elements in the corresponding row of the left singular vector matrix.

This step directly follows the approach proposed in the paper. Specifically, the paper states that leverage scores can be computed using SVD to capture the importance of each data point in the dataset with respect to the model. Even though the paper does not specify exactly how the leverage scores are used to calculate the final sample, it does mention that leverage scores help to sample data points that contribute the most to the model's accuracy.

Algorithm 1 is a simplified version of the sampling algorithm proposed by Chowdhury & Ramuhalli (2024). The computed leverage scores are first normalized to create a probability distribution that sums to 1. This normalized distribution is then used to sample rows from the dataset X. The number of rows sampled is determined by a customizeable parameter. This sampling process closely follows the paper's method. The authors are specific on the fact that the leverage scores are used to form a probability distribution from which data points are selected. Rows with higher leverage scores are more likely to be chosen for the sample, ensuring that important data points (those that affect the model the most) are more likely to be included in the subsampled dataset.

The simplified implementation provided by `BasicLeverageScoresSampler` simplifies the original strategy taken by the authors, offering a simplified version of the approach described in the paper. While it is arguably **more computationally efficient**, it does not fully capture the optimizations and error guarantees of the paper's more complex sampling model. The approach proposed by the authors would be more appropriate for high-dimensional datasets or applications where precise error bounds and model accuracy are critical. However, for many practical purposes, the simpler implementation can provide a reasonable trade-off between speed and approximation accuracy.

Algorithm 1 Basic Leverage Scores Sampling

Input: feature-matrix X, target-labels y, sample rate r

$scores \leftarrow$ `leverage_scores(`X`)`
$normalized\_p \leftarrow scores$ / `sum(`$scores$`)`
$sampled\_indices \leftarrow$ `random_select(`X`, `r`, `$normalized\_p$`)`
$X\_sampled \leftarrow X[sampled\_indices]$
$y\_sampled \leftarrow y[sampled\_indices]$

Return: $X\_sampled, y\_sampled$

More specifically, the paper describes a more detailed approach where not only the leverage scores are used to sample the data, but also a *sketching matrix*, which allows the logistic regression model to be approximated more efficiently by reducing the dimensionality of the problem. The full matrix is used to modify how the data is projected into a lower-dimensional space, enabling faster training. In contrast, our initial implementation skips the step of constructing the sketching matrix. This simplifies the implementation, but results in a slightly less optimized solution as a trade-off.

Moreover, the authors design theoretical guarantees for how well the sampled data approximates the full dataset. The process involves complex mathematical analysis to bound the error in terms of approximation. Our implementation does not include these guarantees. While it is expected to work well in practice, we do not have formal error bounds for the approximation quality in this initial implementation.

In contemplation of the approach proposed by our initial implementation, we identify key advantages to simplifying the original algorithm:

- **Simplicity:** The implementation here is simpler and easier to understand. It directly samples rows based on leverage scores, which is computationally efficient and suitable for smaller datasets or less complex problems.
- **Performance:** By using only leverage score sampling without constructing a sketching matrix, the algorithm arguably runs faster, since it requires fewer computations. This makes it suitable for applications where performance is more critical than exact accuracy.

We also acknowledge the trade-offs between simplicity and accuracy:

- **Accuracy:** Because the simplified implementation

skips the step of constructing the sketching matrix, the approximation may not be as accurate as the one described in the paper. Without the full matrix, the approximation quality could be compromised, especially for large, high-dimensional datasets.

- **Theoretical guarantees:** Unlike the paper, this implementation does not provide formal guarantees on the error bounds, which are important for accuracy-critical applications.

4.5. Uniform Random Sampler

4.5.1. Overview

To effectively benchmark advanced data reduction algorithms, it is essential to understand whether there are advantages offered by the added selection complexity of such algorithms relative to a baseline method that employs a purely random selection strategy. For this purpose, a random sampling algorithm with uniform selection probability distribution is constructed. It provides this baseline by uniformly sampling dataset rows without considering any data-specific metrics or characteristics. This sampling technique is designed explicitly to evaluate the inherent performance implications of dataset size reduction independently of data-driven heuristics, thus offering a neutral point of comparison against more complex sampling methods, such as those utilizing leverage scores or other importance metrics.

Our implementation of the Uniform Random Sampler is encapsulated within a concise Python class, `RandomSampler`, intended to replicate simple, unbiased selection behavior. This simplicity allows us to quantify the improvements gained through more sophisticated sampling strategies relative to a truly random baseline. Consequently, performance variations observed between models trained on uniformly sampled data and those using leverage-based or feature-driven sampling methods directly reflect the added benefit of informed data-point selection.

4.5.2. Sampling Method

The fundamental principle underlying the Uniform Random Sampler algorithm is the equal likelihood of selection for each row of the dataset, ensuring unbiased representational distribution. Given a feature matrix X of dimensions $(n_{samples} \times n_{features})$ and an associated target vector y, the Uniform Random Sampler selects a subset containing a predefined fraction of the original dataset, denoted as the sample percentage. Crucially, this sampling procedure employs uniform probabilities, assigning identical selection likelihood to every data entry, irrespective of feature values, variance, or distribution.

Formally, the selection process calculates the number of samples to extract, determined by the sample percentage multiplied by the total number of rows in the dataset. Subsequently, a set of unique indices is drawn randomly without replacement from the range of available indices. The resulting sampled subset, represented by the matrices $X_{sampled}$ and $y_{sampled}$, is then utilized to train the corresponding machine learning model. The simplicity of this method ensures its computational efficiency, making it highly suitable for large-scale datasets as a baseline sampling procedure.

Algorithm 2 Uniform Random Sampling

Input: feature-matrix X, target-labels y, sample rate r

$sampled\_indices \leftarrow$ `random_select(`X`,` r`)`
$X\_sampled \leftarrow X[sampled\_indices]$
$y\_sampled \leftarrow y[sampled\_indices]$

Return: $X\_sampled, y\_sampled$

The implementation of the uniform random sampling algorithm is detailed in **Algorithm 2**. Initially, it calculates the exact number of samples required based on the user-specified sample rate. Next, a uniform random selection without replacement is performed on the available indices, yielding an unbiased subset of dataset rows. Finally, the sampled indices are employed to extract the corresponding rows from the feature and target matrices.

The explicit introduction of uniform random sampling as a baseline significantly enriches comparative studies focused on dataset reduction. It enables precise quantification of the value added by complex selection strategies, such as those leveraging SVD-derived scores, feature importance, or prediction uncertainty. Consequently, due to the experimental nature of this study, we can more conclusively demonstrate the impacts attributable to the various sampling methods by explicitly contrasting their results against a standardized, unbiased baseline.

5. Experimental Results

5.1. Experiments

To assess the efficacy of the sampling algorithms explored by this study, comprehensive experiments have been conducted as detailed in Appendix A. This section provides a short summary of observations for each set of experiments.

Table 1 shows the series of experiments conducted with `SlowLogisticRegression` over the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993). We observe significant reduction in training time, at the negligible cost of 1 percent in prediction accuracy for both `BasicLeverageScoresSampler` and `RandomSampler`. This experiment does not identify statistically relevant differences between the two sampling

Table 1. Experiments conducted for logistic regression using `SlowLogisticRegression` and corresponding average accuracy Acc and training time $t_{training}$ for leverage scores random sampling and uniform random sampling.

EXP. #	SAMPLING	*Accuracy*	*Training Time*
1	NONE	0.97 ± 0.00	0.26 ± 0.04 S
2	LEV. SCORES	0.96 ± 0.04	0.12 ± 0.01 S
5	UNIFORM	0.96 ± 0.04	0.13 ± 0.01 S

Table 2. Experiments conducted for logistic regression using SciKit-Learn's `LogisticRegression` and corresponding average accuracy Acc and training time $t_{training}$ for leverage scores random sampling and uniform random sampling.

EXP. #	SAMPLING	*Accuracy*	*Training Time*
3	NONE	0.97 ± 0.00	1.88 ± 0.83 MS
4	LEV. SCORES	0.94 ± 0.05	0.82 ± 0.38 MS

Table 3. Experiments conducted for K-Nearest Neighbors using SciKit-Learn's `KNeighborsClassifier` and corresponding average accuracy Acc and training time $t_{training}$ for leverage scores random sampling and uniform random sampling.

EXP. #	SAMPLING	*Accuracy*	*Training Time*
7	NONE	1.00 ± 0.00	1.03 ± 0.64 MS
8	LEV. SCORES	0.94 ± 0.11	0.89 ± 0.57 MS
9	UNIFORM	0.93 ± 0.11	0.81 ± 0.40 MS

Table 4. Experiments conducted for Support Vector Machine using SciKit-Learn's `SVC` and corresponding average accuracy Acc and training time $t_{training}$ for leverage scores random sampling and uniform random sampling.

EXP. #	SAMPLING	*Accuracy*	*Training Time*
10	NONE	1.00 ± 0.00	1.48 ± 0.69 MS
11	LEV. SCORES	0.86 ± 0.16	1.27 ± 0.69 MS
12	UNIFORM	0.86 ± 0.16	1.56 ± 1.02 MS

algorithms.

Table 2 shows the series of experiments conducted with SciKit-Learn's `LogisticRegression` model over the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993). We observe significant reduction in training time, at the cost of 3 percent in prediction accuracy.

Next, we explore the impact of the randomized sampling algorithms in three typical classification algorithms, namely K-Nearest Neighbors, Support Vector Machine, and Random Forest Classification.

Table 3 shows the series of experiments conducted with SciKit-Learn's `KNeighborsClassifier` model over the Iris Dataset (Fisher, 1936). We observe moderate reduction in training time, at the cost of 6 percent in prediction accuracy for `BasicLeverageScoresSampler`, and 7 percent for `RandomSampler`. This experiment indicates uniform random sampling provides faster training time compared to leverage scores sampling.

Table 4 shows the series of experiments conducted with SciKit-Learn's `SVC` model over the Iris Dataset (Fisher, 1936). We observe no reduction in training time, at the significant cost of 14 percent in prediction accuracy for both sampling algorithms.

Table 5 shows the series of experiments conducted with SciKit-Learn's `RandomForestClassifier` model over the Wine Dataset (Aeberhard & Forina, 1992). We observe marginal reduction in training time, at the substantial cost of 7 percent in prediction accuracy for `BasicLeverageScoresSampler` and 6 percent for `RandomSampler`. No statistically significant differences have been identified between the two sampling algorithms in this experiment.

5.2. Results Analysis

The initial set of experiments involved using the `SlowLogisticRegression` model applied to the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993). Results indicated that both leverage scores sampling and uniform random sampling achieved comparable outcomes, reducing training time approximately by half (from 0.26 ± 0.04 s to roughly 0.12 ± 0.01 s). This substantial reduction in computational effort was accompanied by a minimal accuracy loss of merely one percent. The negligible differences in accuracy observed between leverage-based and uniformly random sampling methods suggest that, for logistic regression scenarios similar to the one studied here, the advantage of informed leverage scores sampling might be limited, especially considering its computational overhead.

Further experiments assessed the impact of leverage scores sampling using the SciKit-Learn implementation of `LogisticRegression`, again employing the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993). Here, a considerable reduction in training time was observed—from 1.88 ± 0.83 ms to 0.82 ± 0.38 ms when leveraging scores-based sampling was applied—amounting to approximately a 56% reduction. Nonetheless, this computational improvement was accompanied by a notable accuracy decline of around three percent, from 0.97 to 0.94.

Table 5. Experiments conducted for Random Forest Classifier using SciKit-Learn's `RandomForestClassifier` and corresponding average accuracy Acc and training time $t_{training}$ for leverage scores random sampling and uniform random sampling.

EXP. #	SAMPLING	*Accuracy*	*Training Time*
13	NONE	1.00 ± 0.00	0.15 ± 0.02 MS
14	LEV. SCORES	0.93 ± 0.10	0.13 ± 0.01 MS
15	UNIFORM	0.94 ± 0.10	0.12 ± 0.01 MS

Although this trade-off might be acceptable in computationally intensive practical scenarios, it underscores a potential limitation of these sampling approaches when predictive accuracy remains the primary concern, particularly in critical decision-making contexts.

Additional experiments involved applying sampling methods to the K-Nearest Neighbors classifier using SciKit-Learn's `KNeighborsClassifier` model and the Iris dataset (Fisher, 1936). These outcomes reinforced the trade-off between computational efficiency and predictive accuracy. Training time was moderately reduced with leverage-based sampling (0.89 ± 0.57 ms) and further reduced with uniform sampling (0.81 ± 0.40 ms), compared to the original 1.03 ± 0.64 ms. However, these reductions came with notable accuracy penalties—six percent for leverage-based sampling and seven percent for uniform sampling. Interestingly, uniform sampling marginally outperformed leverage sampling in terms of speed, indicating that simpler random selection algorithms might occasionally be more computationally efficient for lightweight classification models, though accuracy degradation remains a significant factor.

Further analysis addressed the use of sampling methods with the Support Vector Machine with SciKit-Learn's (`SVC`) model on the Iris dataset (Fisher, 1936). Here, both sampling algorithms failed to deliver meaningful computational savings, with training time remaining roughly unchanged compared to using the entire dataset. More critically, a substantial drop in accuracy (14 percent) was observed for both leverage scores and uniform random sampling, declining from a perfect accuracy of 1.00 to an average of 0.86. This outcome clearly illustrates a limitation when applying dataset-reduction sampling methods to kernel-based models such as SVM, whose accuracy is highly sensitive to the specific training instances included, thus highlighting the need for caution in similar contexts.

Lastly, experiments focused on applying sampling methods to a Random Forest Classifier trained with SciKit-Learn's `RandomForestClassifier` model and the Wine dataset (Aeberhard & Forina, 1992). Both sampling methods yielded minor training time improvements, with uniform sampling slightly outperforming leverage-based sampling (0.12 ± 0.01 ms versus 0.13 ± 0.01 ms), compared to the original training time of 0.15 ± 0.02 ms. However, the accuracy reductions of seven percent for leverage-based sampling and six percent for uniform random sampling represented significant declines. No statistically significant difference between the two sampling methods was identified, suggesting that uniform random sampling might be equally effective—or ineffective—in this context, without the added computational complexity of leverage-based sampling.

Overall, these experiments suggest that random sampling algorithms, whether informed by leverage scores or employing uniform selection probability, effectively reduce computational training effort, though often at a variable cost to model accuracy, strongly dependent upon the specific learning algorithm and dataset characteristics. Logistic regression models demonstrated the most promising results, exhibiting minimal accuracy degradation, whereas methods sensitive to the training set's composition, such as SVM and Random Forest, exhibited more substantial accuracy losses. Consequently, while these sampling approaches are valuable tools for dataset reduction in computationally intensive scenarios, careful evaluation and analysis of trade-offs specific to each application domain remain essential for their effective and practical deployment.

6. Conclusion and Future Direction

This research work systematically explored the application of random sampling algorithms as a strategic attempt to effectively address the persistent and challenging problem associated with training machine learning models using datasets of exceptionally high magnitude. Through the rigorous experiments and analyses conducted here, we have gained valuable insights into the potential advantages and tangible benefits of employing random sampling methods, particularly highlighted by the findings observed in the context of logistic regression models. Specifically, these random selection algorithms have consistently demonstrated their capability to serve as a feasible and practical solution, effectively reducing the volume of the training dataset without causing critically significant loss in predictive quality or accuracy.

It is essential and particularly noteworthy to mention, however, that careful consideration is due when determining whether the inevitable loss in predictive accuracy introduced by dataset reduction remains acceptable, given the specific circumstances and demands of the use-case at hand. For example, scenarios involving academic or theoretical research might prioritize predictive accuracy as an indispensable component for a study, whereas practical, industry-focused applications might alternatively prioritize computational speed and reduced training time as key performance met-

rics, thus tolerating a slightly lower accuracy for the sake of faster model training and deployment. Nevertheless, the results presented herein clearly demonstrate the viability and practicality of random sampling methods as effective techniques in machine learning dataset reduction.

Future research efforts can build upon these findings by further strengthening technical guarantees that increase representativeness of the reduced dataset by optimizing leverage scores selection while considering numerical relevance of the original records. Moreover, there could be another pathway in evaluating other strategies in dimensionality reduction that go beyond random sampling methods, and examining the various trade-offs and validating these algorithms across diverse real-world scenarios to consolidate their practical applicability can provide substantial contribution to the field of Machine Learning.

Software and Data

The source code for this work can be downloaded at:

https://github.com/luizparente/logistic-regression

References

Aeberhard, S. and Forina, M. Wine. UCI Machine Learning Repository, 1992. URL https://archive.ics.uci.edu/dataset/109/wine. DOI: https://doi.org/10.24432/C5PC7J.

Chowdhury, A. and Ramuhalli, P. A provably accurate randomized sampling algorithm for logistic regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 11597–11605, 2024. URL https://ojs.aaai.org/index.php/AAAI/article/view/29042.

Fisher, R. A. Iris. UCI Machine Learning Repository, 1936. URL https://archive.ics.uci.edu/dataset/53/iris. DOI: https://doi.org/10.24432/C56C76.

Liu, Y., Gu, J., Wang, K., Zhu, Z., Jiang, W., and You, Y. Dream: Efficient dataset distillation by representative matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17314–17324, October 2023. URL https://openaccess.thecvf.com/content/ICCV2023/html/Liu_DREAM_Efficient_Dataset_Distillation_by_Representative_Matching_ICCV_2023_paper.html.

Munteanu, A., Schwiegelshohn, C., Sohler, C., and Woodruff, D. On coresets for logistic regression. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/63bfd6e8f26d1d3537f4c5038264ef36-Paper.pdf.

Murphy, K. P. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL http://probml.github.io/book1.

Ordozgoiti, B., Matakos, A., and Gionis, A. Generalized leverage scores: Geometric interpretation and applications. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17056–17070. PMLR, Jul 2022. URL https://proceedings.mlr.press/v162/ordozgoiti22a.html.

Shen, Z., Cui, P., Zhang, T., and Kunag, K. Stable learning via sample reweighting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5692–5699, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6024.

Sur, P. and Candès, E. J. A modern maximum-likelihood theory for high-dimensional logistic regression. *Proceedings of the National Academy of Sciences*, 116(29):14516–14525, 2019. URL https://www.pnas.org/doi/abs/10.1073/pnas.1810420116.

Wolberg, W., Mangasarian, O., Street, N., and Street, W. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1993. URL https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic. DOI: https://doi.org/10.24432/C5DW2B.

Yadlowsky, S., Yun, T., McLean, C. Y., and D' Amour, A. Sloe: A faster method for statistical inference in high-dimensional logistic regression. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 29517–29528. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/f6c2a0c4b566bc99d596e58638e342b0-Paper.pdf.

A. Experiments

A.0.1. Objective Metrics

The goal of the experiments described here is to record key metrics for each model-dataset combination, namely **accuracy** and **training time**. Formally, we define:

$$Accuracy(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} 1(\hat{y}_i = y_i)$$

$$Training\ Time = s + t$$

Where:

- y is a vector representing the true values.
- $\hat{y}$ is a vector representing the predictions.
- n is the number of predictions.
- s is the time consumed by the sampling algorithm. This is $zero$ for non-sampled models.
- t is the time consumed by the training process.

Additionally, each experiment defines its own set of parameters and leverage different algorithm implementations.

A.1. Experiment 1: `SlowLogisticRegression`'s Performance, No Sampling

A.1.1. Objective

The experiment aims to assess the performance of a custom implementation of logistic regression, named `SlowLogisticRegression`, in terms of classification accuracy and training time across multiple runs on an open-source dataset (Wolberg et al., 1993). The experiment seeks to quantify the model's generalization ability and computational efficiency under standardized conditions.

A.1.2. Materials and Methods

The experiment utilizes a dataset represented by feature matrix $X \in \mathbb{R}^{n \times m}$ and target vector $y \in \{0, 1\}^n$, where n is the number of samples and m is the number of features. The dataset is assumed to be preprocessed and suitable for binary classification tasks.

For each experimental run, the dataset is split into training and test sets using an 80-20 partition, with 80% of the data allocated to training ($X_{\text{train}}, y_{\text{train}}$) and 20% to testing ($X_{\text{test}}, y_{\text{test}}$). The split is performed using a fixed random seed of 42 to ensure reproducibility across runs. Feature scaling is applied using Python's SciKit-Learn `MinMaxScaler`, which normalizes each feature to the range $[0, 1]$ based on the training set's minimum and maximum values. The scaling transformation is computed as:

$$X'_{i,j} = \frac{X_{i,j} - \min(X_{:,j})}{\max(X_{:,j}) - \min(X_{:,j})}$$

where $X_{i,j}$ denotes the j-th feature of the i-th sample, and the transformation is fitted on X_{train} and applied to both X_{train} and X_{test}.

A.1.3. Model and Training

The classification model employed is a custom logistic regression implementation (`SlowLogisticRegression`), parameterized by a learning rate of 0.1 and a fixed number of 5000 training epochs. The implementation of this model is detailed in section 2 of this paper.

A.1.4. Experimental Procedure

The experiment is repeated 100 times for statistical relevance. For each iteration:

1. The dataset is split into training and test sets.
2. Features are scaled using `MinMaxScaler`.
3. The `SlowLogisticRegression` model is initialized and trained on the scaled training data, with training time recorded.
4. Predictions are generated on the scaled test set.
5. Accuracy is calculated by comparing predictions to the true test labels.
6. Accuracy and training time are tracked separately.

This procedure records the accuracy scores and training times across each cycle, which are later used to calculate the average values for each metric.

A.1.5. Analysis

The collected accuracies and training times enable statistical analysis of the model's performance. Key statistics, such as the mean and standard deviation of accuracy, can be computed to evaluate consistency and generalization. Similarly, training time statistics provide insight into the computational cost of the `SlowLogisticRegression` implementation without any sampling.

A.1.6. Assumptions and Limitations

The experiment assumes that the dataset is suitable for binary classification and that the features benefit from min-max scaling. The fixed hyperparameters (learning rate =

0.1, epochs = 5000), obtained empirically, appear to suit this experiment well and provide satisfactory results.

A.1.7. Expected Outcomes

The experiment is expected to yield a distribution of accuracy scores reflecting the model's predictive capability and a distribution of training times indicating its efficiency. These results can be used to benchmark `SlowLogisticRegression` against other implementations or algorithms under identical conditions.

A.1.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 1**.

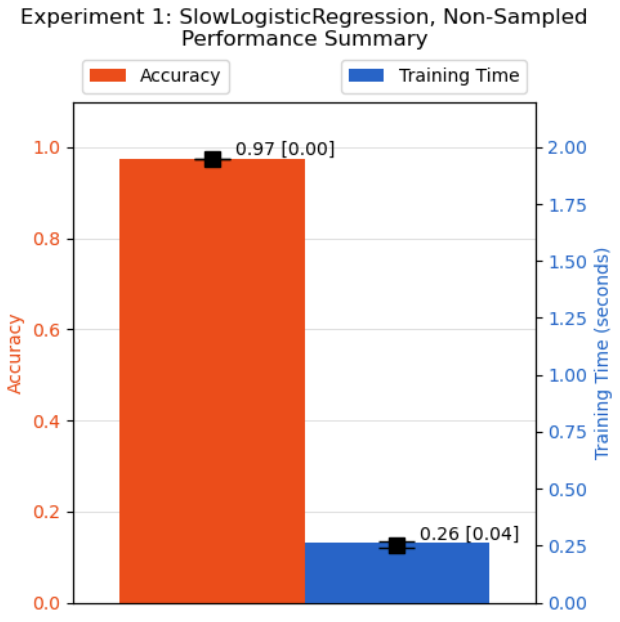

Figure 1. Mean accuracies and training times observed over 100 training cycles of `SlowLogisticRegression`, without any sampling, using the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993).

A.2. Experiment 2: `SlowLogisticRegression`'s Performance with Leverage Score Sampling

A.2.1. Objective

The experiment aims to assess the performance of a custom implementation of logistic regression, named `SlowLogisticRegression`, in terms of classification accuracy and training time across multiple runs on an open-source dataset (Wolberg et al., 1993). The experiment seeks to quantify the model's generalization ability and computational efficiency under standardized conditions, incorporating a data sampling step using the randomized sampling algorithm `BasicLeverageScoresSampler`.

A.2.2. Materials and Methods

The experiment utilizes a dataset represented by feature matrix $X \in \mathbb{R}^{n \times m}$ and target vector $y \in \{0, 1\}^n$, where n is the number of samples and m is the number of features. The dataset is assumed to be preprocessed and suitable for binary classification tasks.

For each experimental run, the dataset is first sampled to 20% of its original size using the `BasicLeverageScoresSampler`, resulting in a reduced dataset $(X_{\text{sampled}}, y_{\text{sampled}})$. This sampled dataset is then split into training and test sets using an 80-20 partition, with 80% of the sampled data allocated to training $(X_{\text{train}}, y_{\text{train}})$ and 20% to testing $(X_{\text{test}}, y_{\text{test}})$. The split is performed using a fixed random seed of 42 to ensure reproducibility across runs. Feature scaling is applied using Python's SciKit-Learn `MinMaxScaler`, which normalizes each feature to the range $[0, 1]$ based on the training set's minimum and maximum values. The scaling transformation is computed as:

$$X'_{i,j} = \frac{X_{i,j} - \min(X_{:,j})}{\max(X_{:,j}) - \min(X_{:,j})}$$

where $X_{i,j}$ denotes the j-th feature of the i-th sample, and the transformation is fitted on X_{train} and applied to both X_{train} and X_{test}.

A.2.3. Model and Training

The classification model employed is a custom logistic regression implementation (`SlowLogisticRegression`), parameterized by a learning rate of 0.1 and a fixed number of 5000 training epochs. The implementation of this model is detailed in section 2 of this paper.

A.2.4. Experimental Procedure

The experiment is repeated 100 times for statistical relevance. For each iteration:

1. The dataset is sampled to 20% of its original size using `BasicLeverageScoresSampler`.
2. The sampled dataset is split into training and test sets.
3. Features are scaled using `MinMaxScaler`.
4. The `SlowLogisticRegression` model is initialized and trained on the scaled training data, with training time recorded.
5. Predictions are generated on the scaled test set.

6. Accuracy is calculated by comparing predictions to the true test labels.

7. Accuracy and training time are tracked separately.

This procedure records the accuracy scores and training times across each cycle, which are later used to calculate the average values for each metric.

A.2.5. Analysis

The collected accuracies and training times enable statistical analysis of the model's performance. Key statistics, such as the mean and standard deviation of accuracy, can be computed to evaluate consistency and generalization. Similarly, training time statistics provide insight into the computational cost of the `SlowLogisticRegression` implementation with leverage score sampling.

A.2.6. Assumptions and Limitations

The experiment assumes that the dataset is suitable for binary classification and that the features benefit from min-max scaling. The fixed hyperparameters (learning rate = 0.1, epochs = 5000), obtained empirically, appear to suit this experiment well and provide satisfactory results. Additionally, the use of leverage score sampling assumes that the selected 20% subset adequately represents the original dataset's structure.

A.2.7. Expected Outcomes

The experiment is expected to yield a distribution of accuracy scores reflecting the model's predictive capability and a distribution of training times indicating its efficiency. These results can be used to benchmark `SlowLogisticRegression` against other implementations or algorithms under identical conditions, and particularly to evaluate the impact of leverage score sampling algorithm `BasicLeverageScoresSampler` impacts on prediction accuracy and model training time.

A.2.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 2**.

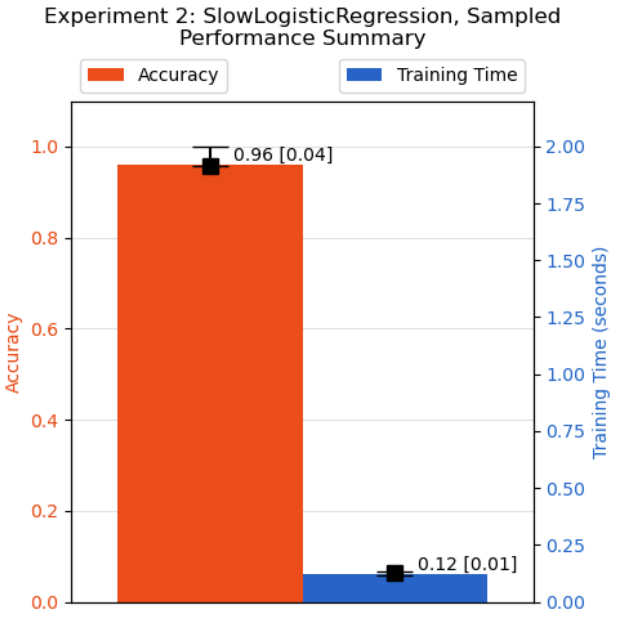

Figure 2. Mean accuracies and training times observed over 100 training cycles of `SlowLogisticRegression`, with `BasicLeverageScoresSampler` sampling, using the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993).

A.3. Experiments 3 and 4: Repeating Experiments 1 and 2 using SciKit-Learn's `LogisticRegression` Model

A.3.1. Objective

Experiments 3 and 4 replicate Experiments 1 and 2, respectively, replacing `SlowLogisticRegression` with SciKit-Learn's optimized `LogisticRegression` class. Experiment 3 uses the full dataset, while Experiment 4 applies 20% leverage score sampling via `BasicLeverageScoresSampler`. The goal is to assess whether sampling impacts the performance of an optimized model, comparing accuracy and training efficiency against Experiments 1 and 2 to evaluate sampling effects across custom and optimized implementations.

A.3.2. Materials and Methods

The dataset, sampling (for Experiment 4), and scaling follow the procedures in Experiments 1 and 2, using $X \in \mathbb{R}^{n \times m}$, $y \in \{0, 1\}^n$, an 80-20 train-test split (random seed 42), and `MinMaxScaler` normalization.

A.3.3. Model and Training

The model is SciKit-Learn's `LogisticRegression` with default parameters, replacing the custom `SlowLogisticRegression` (learning rate 0.1, 5000 epochs) used previously.

A.3.4. Experimental Procedure

The experiment is repeated 100 times for statistical relevance. The same procedure detailed for previous experiments are employed here for Experiment 3 and 4.

A.3.5. Analysis

Mean and standard deviation of accuracy and training times assess consistency, generalization, and computational cost, comparing optimized `LogisticRegression` performance with and without sampling.

A.3.6. Assumptions and Limitations

The experiment assumes binary classification suitability and effective min-max scaling. Default `LogisticRegression` parameters are presumed optimal, and sampling assumes the 20% subset retains dataset structure.

A.3.7. Expected Outcomes

Accuracy and training time distributions will reveal `LogisticRegression`'s performance, highlighting sampling impacts versus Experiments 1 and 2, and benchmarking optimized versus custom models.

A.3.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 3** and **Figure 4**.

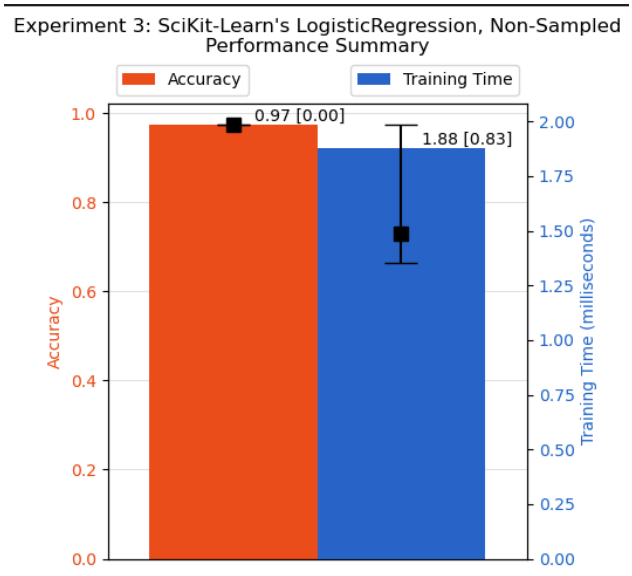

Figure 3. Mean accuracies and training times observed over 100 training cycles of SciKit-Learn's `LogisticRegression` class, without any sampling, using the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993).

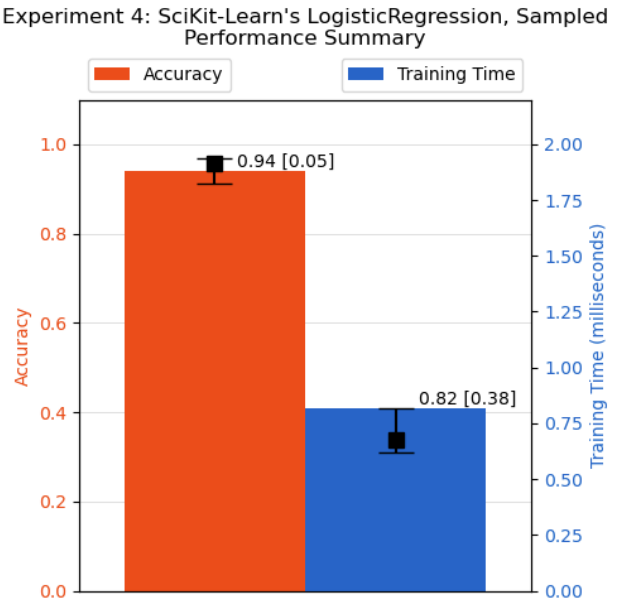

Figure 4. Mean accuracies and training times observed over 100 training cycles of SciKit-Learn's `LogisticRegression` class, with `BasicLeverageScoresSampler` sampling, using the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993).

A.4. Experiment 5: `SlowLogisticRegression`'s Performance with Random Sampling

A.4.1. Objective

The experiment aims to assess the performance of a custom implementation of logistic regression, named `SlowLogisticRegression`, in terms of classification accuracy and training time across multiple runs on an open-source dataset (Wolberg et al., 1993). The experiment seeks to quantify the model's generalization ability and computational efficiency under standardized conditions, incorporating a data sampling step using the randomized sampling algorithm `RandomSampler`.

A.4.2. Materials and Methods

The experiment utilizes a dataset represented by feature matrix $X \in \mathbb{R}^{n \times m}$ and target vector $y \in \{0, 1\}^n$, where n is the number of samples and m is the number of features. The dataset is assumed to be preprocessed and suitable for binary classification tasks.

For each experimental run, the dataset is first sampled to 20% of its original size using the `RandomSampler`, resulting in a reduced dataset ($X_{\text{sampled}}, y_{\text{sampled}}$). This sampled dataset is then split into training and test sets using an 80-20 partition, with 80% of the sampled data allocated to training

($X_{\text{train}}, y_{\text{train}}$) and 20% to testing ($X_{\text{test}}, y_{\text{test}}$). The split is performed using a fixed random seed of 42 to ensure reproducibility across runs. Feature scaling is applied using Python's SciKit-Learn `MinMaxScaler`, which normalizes each feature to the range $[0, 1]$ based on the training set's minimum and maximum values. The scaling transformation is computed as:

$$X'_{i,j} = \frac{X_{i,j} - \min(X_{:,j})}{\max(X_{:,j}) - \min(X_{:,j})}$$

where $X_{i,j}$ denotes the j-th feature of the i-th sample, and the transformation is fitted on X_{train} and applied to both X_{train} and X_{test}.

A.4.3. Model and Training

The classification model employed is a custom logistic regression implementation (`SlowLogisticRegression`), parameterized by a learning rate of 0.1 and a fixed number of 5000 training epochs. The implementation of this model is detailed in section 2 of this paper.

A.4.4. Experimental Procedure

The experiment is repeated 100 times for statistical relevance. For each iteration:

1. The dataset is sampled to 20% of its original size using `RandomSampler`.
2. The sampled dataset is split into training and test sets.
3. Features are scaled using `MinMaxScaler`.
4. The `SlowLogisticRegression` model is initialized and trained on the scaled training data, with training time recorded.
5. Predictions are generated on the scaled test set.
6. Accuracy is calculated by comparing predictions to the true test labels.
7. Accuracy and training time are tracked separately.

This procedure records the accuracy scores and training times across each cycle, which are later used to calculate the average values for each metric.

A.4.5. Analysis

The collected accuracies and training times enable statistical analysis of the model's performance. Key statistics, such as the mean and standard deviation of accuracy, can be computed to evaluate consistency and generalization. Similarly, training time statistics provide insight into the computational cost of the `SlowLogisticRegression` implementation with leverage score sampling.

A.4.6. Assumptions and Limitations

The experiment assumes that the dataset is suitable for binary classification and that the features benefit from min-max scaling. The fixed hyperparameters (learning rate = 0.1, epochs = 5000), obtained empirically, appear to suit this experiment well and provide satisfactory results. Additionally, the use of random sampling assumes that the selected 20% subset adequately represents the original dataset's structure.

A.4.7. Expected Outcomes

The experiment is expected to yield a distribution of accuracy scores reflecting the model's predictive capability and a distribution of training times indicating its efficiency. These results can be used to benchmark `SlowLogisticRegression` against other implementations or algorithms under identical conditions, and particularly to evaluate the impact of random sampling algorithm `RandomSampler` on prediction accuracy and model training time.

A.4.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 5**.

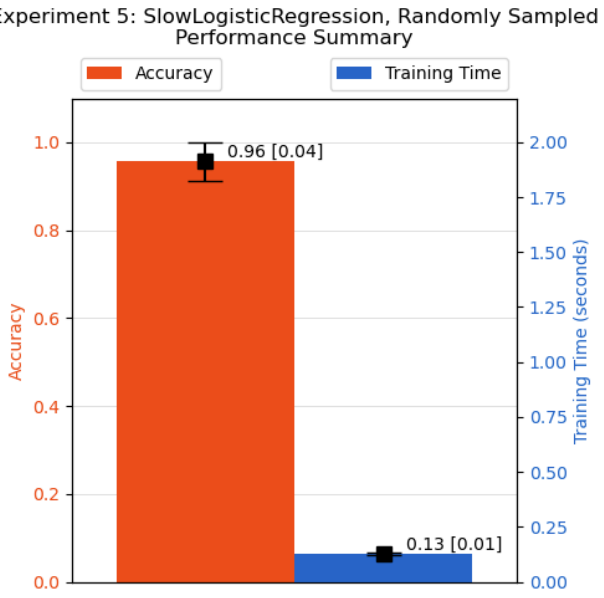

Figure 5. Mean accuracies and training times observed over 100 training cycles of `SlowLogisticRegression`, with `RandomSampler` sampling, using the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993).

A.5. Experiment 6: Statistical Validation of Experiments 1–5 with Increased Runs

A.5.1. Objective

Experiment 5 re-executes Experiments 1, 2, 3, 4, and 5, increasing the number of cycles from 100 to 1000 per condition, to enhance the statistical relevance of prior observations. It assesses `SlowLogisticRegression` and `LogisticRegression` performance under full and 20% sampled dataset scenarios, aiming to confirm the consistency and reliability of accuracy and training time findings from earlier experiments.

A.5.2. Materials and Methods

The dataset, sampling (for Experiments 2 and 4), and scaling remain as in Experiments 1–5, using $X \in \mathbb{R}^{n \times m}$, $y \in \{0, 1\}^n$, an 80-20 train-test split (random seed 42), and `MinMaxScaler` normalization.

A.5.3. Model and Training

Models are `SlowLogisticRegression` (learning rate 0.1, 5000 epochs) for Experiments 1, 2, and 5, and SciKit-Learn's `LogisticRegression` (default parameters) for Experiments 3 and 4, as previously defined.

A.5.4. Experimental Procedure

The experiment is repeated 1000 times per condition (Experiments 1–5). The same procedures described earlier for each experiment still apply here.

A.5.5. Analysis

Mean and standard deviation of accuracy and training times, based on 1000 runs, provide robust statistical validation of consistency, generalization, and computational cost across all conditions.

A.5.6. Assumptions and Limitations

Assumptions align with Experiments 1–5: binary classification suitability, effective scaling, and representative sampling. Fixed parameters for `SlowLogisticRegression` and defaults for `LogisticRegression` are presumed suitable.

A.5.7. Expected Outcomes

Distributions of accuracy and training times across 1000 runs will confirm the statistical relevance of Experiments 1–5, reinforcing comparisons between models and sampling effects with greater confidence.

A.5.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 6**.

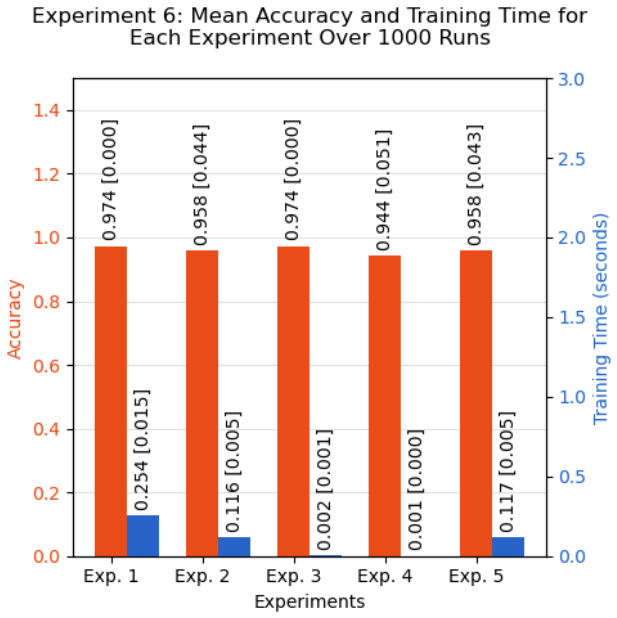

Figure 6. Mean accuracies and training times observed when repeating experiments 1, 2, 3, 4, and 5, this time with 1000 training cycles each.

A.6. Experiment 7: SciKit-Learn's `KNeighborsClassifier` Model's Performance, No Sampling

A.6.1. Objective

The experiment aims to assess the performance of SciKit-Learn's K-Nearest Neighbors model, named `KNeighborsClassifier`, in terms of classification accuracy and training time across multiple runs on an open-source dataset (Fisher, 1936). The experiment seeks to quantify the model's generalization ability and computational efficiency under standardized conditions.

A.6.2. Materials and Methods

The experiment utilizes a dataset represented by feature matrix $X \in \mathbb{R}^{n \times m}$ and target vector $y \in \{0, 1\}^n$, where n is the number of samples and m is the number of features. The dataset is assumed to be preprocessed and suitable for classification tasks.

For each experimental run, the dataset is split into training and test sets using an 80-20 partition, with 80% of the data allocated to training ($X_{\text{train}}, y_{\text{train}}$) and 20% to testing

$(X_{\text{test}}, y_{\text{test}})$. The split is performed using a fixed random seed of 42 to ensure reproducibility across runs. Feature scaling is not used in this experiment.

A.6.3. Model and Training

The classification model employed is provided by SciKit-Learn's `KNeighborsClassifier` class.

A.6.4. Experimental Procedure

The experiment is repeated 1000 times for statistical relevance. For each iteration:

1. The dataset is split into training and test sets.
2. The `KNeighborsClassifier` model is initialized and trained, with training time recorded.
3. Predictions are generated on the test set.
4. Accuracy is calculated by comparing predictions to the true test labels.
5. Accuracy and training time are tracked separately.

This procedure records the accuracy scores and training times across each cycle, which are later used to calculate the average values for each metric.

A.6.5. Analysis

The collected accuracies and training times enable statistical analysis of the model's performance. Key statistics, such as the mean and standard deviation of accuracy, can be computed to evaluate consistency and generalization. Similarly, training time statistics provide insight into the computational cost of the `KNeighborsClassifier` implementation without any sampling.

A.6.6. Assumptions and Limitations

The experiment assumes that the dataset is suitable for classification and that the features.

A.6.7. Expected Outcomes

The experiment is expected to yield a distribution of accuracy scores reflecting the model's predictive capability and a distribution of training times indicating its efficiency. These results can be used to benchmark `KNeighborsClassifier` against other implementations or algorithms under identical conditions.

A.6.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 7**.

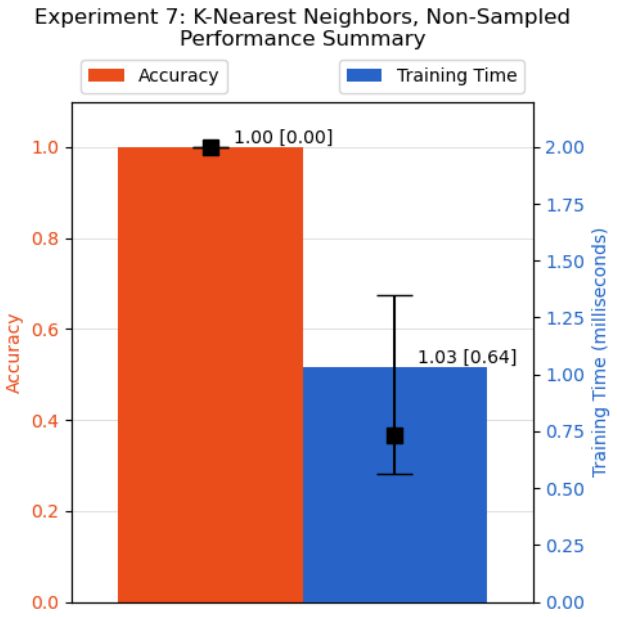

Figure 7. Mean accuracies and training times observed over 1000 training cycles of `KNeighborsClassifier`, without any sampling, using the Iris Dataset (Fisher, 1936).

A.7. Experiment 8: SciKit-Learn's `KNeighborsClassifier` Model's Performance with Leverage Score Sampling

A.7.1. Objective

The experiment aims to assess the performance of SciKit-Learn's `KNeighborsClassifier` model, in terms of classification accuracy and training time across multiple runs on an open-source dataset (Fisher, 1936). The experiment seeks to quantify the model's generalization ability and computational efficiency under standardized conditions, incorporating a data sampling step using the randomized sampling algorithm `BasicLeverageScoresSampler`.

A.7.2. Materials and Methods

The experiment utilizes a dataset represented by feature matrix $X \in \mathbb{R}^{n \times m}$ and target vector $y \in \{0, 1\}^n$, where n is the number of samples and m is the number of features. The dataset is assumed to be preprocessed and suitable for classification tasks.

For each experimental run, the dataset is first sampled to 20% of its original size using the `BasicLeverageScoresSampler`, resulting in a reduced dataset $(X_{\text{sampled}}, y_{\text{sampled}})$. This sampled dataset is then split into training and test sets using an 80-20 partition, with 80% of the sampled data allocated to training

($X_{\text{train}}, y_{\text{train}}$) and 20% to testing ($X_{\text{test}}, y_{\text{test}}$). The split is performed using a fixed random seed of 42 to ensure reproducibility across runs. Feature scaling is not used in this experiment.

A.7.3. Model and Training

The classification model employed is provided by SciKit-Learn's `KNeighborsClassifier` class.

A.7.4. Experimental Procedure

The experiment is repeated 1000 times for statistical relevance. For each iteration:

1. The dataset is sampled to 20% of its original size using `BasicLeverageScoresSampler`.
2. The sampled dataset is split into training and test sets.
3. The `KNeighborsClassifier` model is initialized and trained on the training data, with training time recorded.
4. Predictions are generated on the test set.
5. Accuracy is calculated by comparing predictions to the true test labels.
6. Accuracy and training time are tracked separately.

This procedure records the accuracy scores and training times across each cycle, which are later used to calculate the average values for each metric.

A.7.5. Analysis

The collected accuracies and training times enable statistical analysis of the model's performance. Key statistics, such as the mean and standard deviation of accuracy, can be computed to evaluate consistency and generalization. Similarly, training time statistics provide insight into the computational cost of the `KNeighborsClassifier` implementation with leverage score sampling.

A.7.6. Assumptions and Limitations

The experiment assumes that the dataset is suitable for classification. Additionally, the use of leverage score sampling assumes that the selected 20% subset adequately represents the original dataset's structure.

A.7.7. Expected Outcomes

The experiment is expected to yield a distribution of accuracy scores reflecting the model's predictive capability and a distribution of training times indicating its efficiency. These results can be used to benchmark `KNeighborsClassifier` against other implementations or algorithms under identical conditions, and particularly to evaluate the impact of leverage score sampling algorithm `BasicLeverageScoresSampler` impacts on prediction accuracy and model training time.

A.7.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 8**.

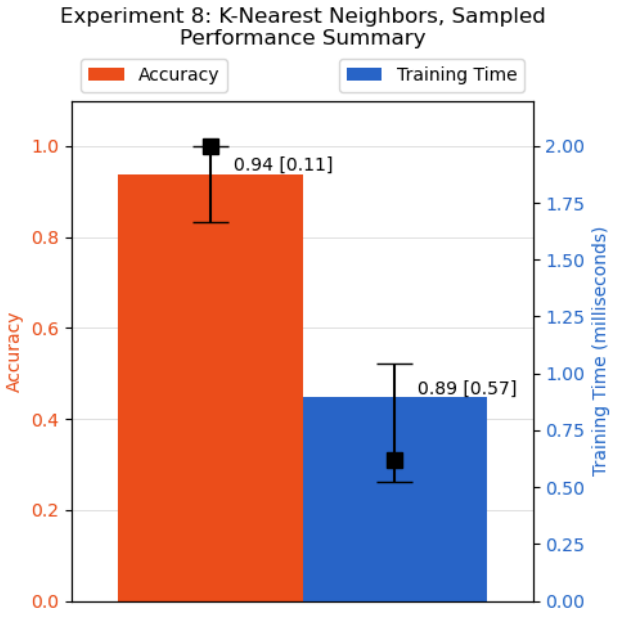

Figure 8. Mean accuracies and training times observed over 1000 training cycles of `KNeighborsClassifier`, with `BasicLeverageScoresSampler` sampling, using the Iris Dataset (Fisher, 1936).

A.8. Experiment 9: SciKit-Learn's `KNeighborsClassifier` Model's Performance with Random Sampling

A.8.1. Objective

The experiment aims to assess the performance of SciKit-Learn's `KNeighborsClassifier` model, in terms of classification accuracy and training time across multiple runs on an open-source dataset (Fisher, 1936). The experiment seeks to quantify the model's generalization ability and computational efficiency under standardized conditions, incorporating a data sampling step using the randomized sampling algorithm `RandomSampler`.

A.8.2. Materials and Methods

The experiment utilizes a dataset represented by feature matrix $X \in \mathbb{R}^{n \times m}$ and target vector $y \in \{0, 1\}^n$, where n is the number of samples and m is the number of features. The dataset is assumed to be preprocessed and suitable for classification tasks.

For each experimental run, the dataset is first sampled to 20% of its original size using the `RandomSampler`, resulting in a reduced dataset ($X_{\text{sampled}}, y_{\text{sampled}}$). This sampled dataset is then split into training and test sets using an 80-20 partition, with 80% of the sampled data allocated to training ($X_{\text{train}}, y_{\text{train}}$) and 20% to testing ($X_{\text{test}}, y_{\text{test}}$). The split is performed using a fixed random seed of 42 to ensure reproducibility across runs. Feature scaling is not used in this experiment.

A.8.3. Model and Training

The classification model employed is provided by SciKit-Learn's `KNeighborsClassifier` class.

A.8.4. Experimental Procedure

The experiment is repeated 1000 times for statistical relevance. For each iteration:

1. The dataset is sampled to 20% of its original size using `RandomSampler`.
2. The sampled dataset is split into training and test sets.
3. The `KNeighborsClassifier` model is initialized and trained on the training data, with training time recorded.
4. Predictions are generated on the test set.
5. Accuracy is calculated by comparing predictions to the true test labels.
6. Accuracy and training time are tracked separately.

This procedure records the accuracy scores and training times across each cycle, which are later used to calculate the average values for each metric.

A.8.5. Analysis

The collected accuracies and training times enable statistical analysis of the model's performance. Key statistics, such as the mean and standard deviation of accuracy, can be computed to evaluate consistency and generalization. Similarly, training time statistics provide insight into the computational cost of the `KNeighborsClassifier` implementation with random sampling.

A.8.6. Assumptions and Limitations

The experiment assumes that the dataset is suitable for classification. Additionally, the use of random sampling assumes that the selected 20% subset adequately represents the original dataset's structure.

A.8.7. Expected Outcomes

The experiment is expected to yield a distribution of accuracy scores reflecting the model's predictive capability and a distribution of training times indicating its efficiency. These results can be used to benchmark `KNeighborsClassifier` against other implementations or algorithms under identical conditions, and particularly to evaluate the impact of leverage score sampling algorithm `RandomSampler` impacts on prediction accuracy and model training time.

A.8.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 9**.

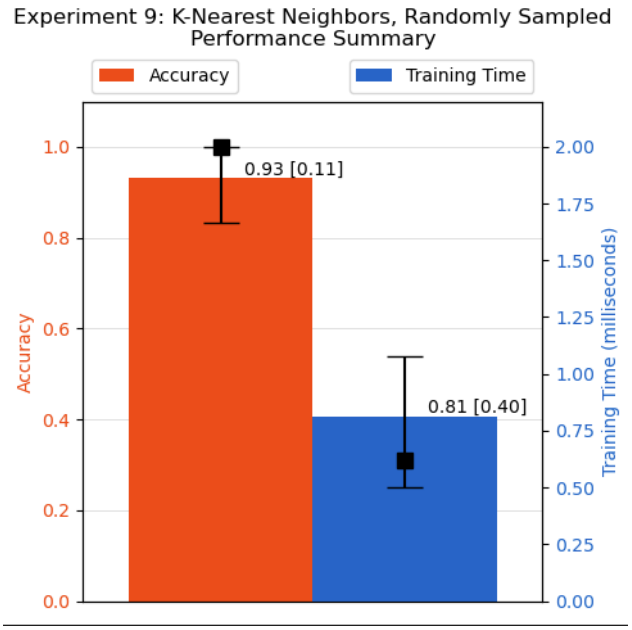

Figure 9. Mean accuracies and training times observed over 1000 training cycles of `KNeighborsClassifier`, with `RandomSampler` sampling, using the Iris Dataset (Fisher, 1936).

A.9. Experiment 10: SciKit-Learn's SVC Model's Performance, No Sampling

A.9.1. Objective

The experiment aims to assess the performance of SciKit-Learn's Support Vector Machine model, named `SVC`, in terms of classification accuracy and training time across multiple runs on an open-source dataset (Fisher, 1936). The experiment seeks to quantify the model's generalization ability and computational efficiency under standardized conditions.

A.9.2. Materials and Methods

The experiment utilizes a dataset represented by feature matrix $X \in \mathbb{R}^{n \times m}$ and target vector $y \in \{0, 1\}^n$, where n is the number of samples and m is the number of features. The dataset is assumed to be preprocessed and suitable for classification tasks.

For each experimental run, the dataset is split into training and test sets using an 80-20 partition, with 80% of the data allocated to training ($X_{\text{train}}, y_{\text{train}}$) and 20% to testing ($X_{\text{test}}, y_{\text{test}}$). The split is performed using a fixed random seed of 42 to ensure reproducibility across runs. Feature scaling is not used in this experiment.

A.9.3. Model and Training

The classification model employed is provided by SciKit-Learn's `SVC` class.

A.9.4. Experimental Procedure

The experiment is repeated 1000 times for statistical relevance. For each iteration:

1. The dataset is split into training and test sets.
2. The `SVC` model is initialized and trained, with training time recorded.
3. Predictions are generated on the test set.
4. Accuracy is calculated by comparing predictions to the true test labels.
5. Accuracy and training time are tracked separately.

This procedure records the accuracy scores and training times across each cycle, which are later used to calculate the average values for each metric.

A.9.5. Analysis

The collected accuracies and training times enable statistical analysis of the model's performance. Key statistics, such as the mean and standard deviation of accuracy, can be computed to evaluate consistency and generalization. Similarly, training time statistics provide insight into the computational cost of the `SVC` implementation without any sampling.

A.9.6. Assumptions and Limitations

The experiment assumes that the dataset is suitable for classification and that the features.

A.9.7. Expected Outcomes

The experiment is expected to yield a distribution of accuracy scores reflecting the model's predictive capability and a distribution of training times indicating its efficiency. These results can be used to benchmark `SVC` against other implementations or algorithms under identical conditions.

A.9.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 10**.

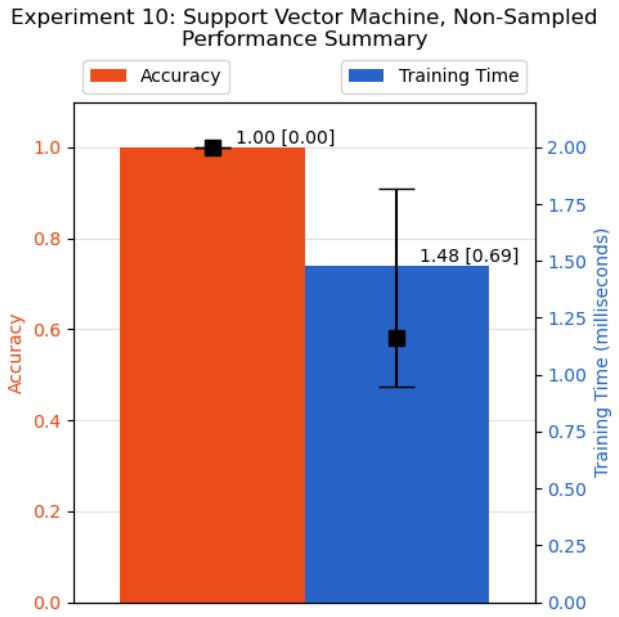

Figure 10. Mean accuracies and training times observed over 1000 training cycles of `SVC`, without any sampling, using the Iris Dataset (Fisher, 1936).

A.10. Experiment 11: SciKit-Learn's SVC Model's Performance with Leverage Score Sampling

A.10.1. Objective

The experiment aims to assess the performance of SciKit-Learn's `SVC` model, in terms of classification accuracy and training time across multiple runs on an open-source dataset (Fisher, 1936). The experiment seeks to quantify the model's generalization ability and computational efficiency under standardized conditions, incorporating a data sampling step using the randomized sampling algorithm `BasicLeverageScoresSampler`.

A.10.2. Materials and Methods

The experiment utilizes a dataset represented by feature matrix $X \in \mathbb{R}^{n \times m}$ and target vector $y \in \{0, 1\}^n$, where n is the number of samples and m is the number of features. The dataset is assumed to be preprocessed and suitable for classification tasks.

For each experimental run, the dataset is first sampled to 20% of its original size using the `BasicLeverageScoresSampler`, resulting in a reduced dataset $(X_{\text{sampled}}, y_{\text{sampled}})$. This sampled dataset is then split into training and test sets using an 80-20 partition, with 80% of the sampled data allocated to training $(X_{\text{train}}, y_{\text{train}})$ and 20% to testing $(X_{\text{test}}, y_{\text{test}})$. The split is performed using a fixed random seed of 42 to ensure reproducibility across runs. Feature scaling is not used in this experiment.

A.10.3. Model and Training

The classification model employed is provided by SciKit-Learn's `SVC` class.

A.10.4. Experimental Procedure

The experiment is repeated 1000 times for statistical relevance. For each iteration:

1. The dataset is sampled to 20% of its original size using `BasicLeverageScoresSampler`.
2. The sampled dataset is split into training and test sets.
3. The `SVC` model is initialized and trained on the training data, with training time recorded.
4. Predictions are generated on the test set.
5. Accuracy is calculated by comparing predictions to the true test labels.
6. Accuracy and training time are tracked separately.

This procedure records the accuracy scores and training times across each cycle, which are later used to calculate the average values for each metric.

A.10.5. Analysis

The collected accuracies and training times enable statistical analysis of the model's performance. Key statistics, such as the mean and standard deviation of accuracy, can be computed to evaluate consistency and generalization. Similarly, training time statistics provide insight into the computational cost of the `SVC` implementation with leverage score sampling.

A.10.6. Assumptions and Limitations

The experiment assumes that the dataset is suitable for classification. Additionally, the use of leverage score sampling assumes that the selected 20% subset adequately represents the original dataset's structure.

A.10.7. Expected Outcomes

The experiment is expected to yield a distribution of accuracy scores reflecting the model's predictive capability and a distribution of training times indicating its efficiency. These results can be used to benchmark `SVC` against other implementations or algorithms under identical conditions, and particularly to evaluate the impact of leverage score sampling algorithm `BasicLeverageScoresSampler` impacts on prediction accuracy and model training time.

A.10.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 11**.

A.11. Experiment 12: SciKit-Learn's SVC Model's Performance with Random Sampling

A.11.1. Objective

The experiment aims to assess the performance of SciKit-Learn's `SVC` model, in terms of classification accuracy and training time across multiple runs on an open-source dataset (Fisher, 1936). The experiment seeks to quantify the model's generalization ability and computational efficiency under standardized conditions, incorporating a data sampling step using the randomized sampling algorithm `RandomSampler`.

A.11.2. Materials and Methods

The experiment utilizes a dataset represented by feature matrix $X \in \mathbb{R}^{n \times m}$ and target vector $y \in \{0, 1\}^n$, where n is the number of samples and m is the number of features. The dataset is assumed to be preprocessed and suitable for

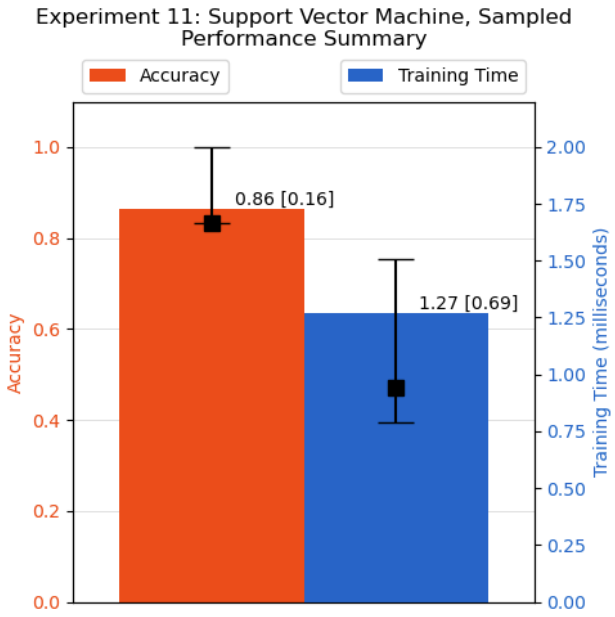

Figure 11. Mean accuracies and training times observed over 1000 training cycles of `SVC`, with `BasicLeverageScoresSampler` sampling, using the Iris Dataset (Fisher, 1936).

classification tasks.

For each experimental run, the dataset is first sampled to 20% of its original size using the `RandomSampler`, resulting in a reduced dataset ($X_{\text{sampled}}, y_{\text{sampled}}$). This sampled dataset is then split into training and test sets using an 80-20 partition, with 80% of the sampled data allocated to training ($X_{\text{train}}, y_{\text{train}}$) and 20% to testing ($X_{\text{test}}, y_{\text{test}}$). The split is performed using a fixed random seed of 42 to ensure reproducibility across runs. Feature scaling is not used in this experiment.

A.11.3. Model and Training

The classification model employed is provided by SciKit-Learn's `SVC` class.

A.11.4. Experimental Procedure

The experiment is repeated 1000 times for statistical relevance. For each iteration:

1. The dataset is sampled to 20% of its original size using `RandomSampler`.
2. The sampled dataset is split into training and test sets.
3. The `SVC` model is initialized and trained on the training data, with training time recorded.
4. Predictions are generated on the test set.
5. Accuracy is calculated by comparing predictions to the true test labels.
6. Accuracy and training time are tracked separately.

This procedure records the accuracy scores and training times across each cycle, which are later used to calculate the average values for each metric.

A.11.5. Analysis

The collected accuracies and training times enable statistical analysis of the model's performance. Key statistics, such as the mean and standard deviation of accuracy, can be computed to evaluate consistency and generalization. Similarly, training time statistics provide insight into the computational cost of the `SVC` implementation with random sampling.

A.11.6. Assumptions and Limitations

The experiment assumes that the dataset is suitable for classification. Additionally, the use of random sampling assumes that the selected 20% subset adequately represents the original dataset's structure.

A.11.7. Expected Outcomes

The experiment is expected to yield a distribution of accuracy scores reflecting the model's predictive capability and a distribution of training times indicating its efficiency. These results can be used to benchmark `SVC` against other implementations or algorithms under identical conditions, and particularly to evaluate the impact of leverage score sampling algorithm `RandomSampler` impacts on prediction accuracy and model training time.

A.11.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 12**.

A.12. Experiment 13: SciKit-Learn's `RandomForestClassifier` Model's Performance, No Sampling

A.12.1. Objective

The experiment aims to assess the performance of SciKit-Learn's Random Forest Classifier model, named `RandomForestClassifier`, in terms of classification accuracy and training time across multiple runs on an open-source dataset (Aeberhard & Forina, 1992). The experiment seeks to quantify the model's generalization ability and computational efficiency under standardized conditions.

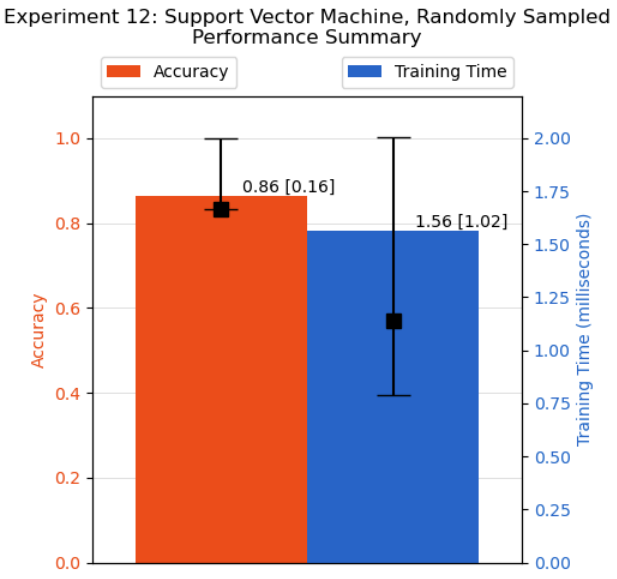

Figure 12. Mean accuracies and training times observed over 1000 training cycles of `SVC`, with `RandomSampler` sampling, using the Iris Dataset (Fisher, 1936).

A.12.2. Materials and Methods

The experiment utilizes a dataset represented by feature matrix $X \in \mathbb{R}^{n \times m}$ and target vector $y \in \{0, 1\}^n$, where n is the number of samples and m is the number of features. The dataset is assumed to be preprocessed and suitable for classification tasks.

For each experimental run, the dataset is split into training and test sets using an 80-20 partition, with 80% of the data allocated to training ($X_{\text{train}}, y_{\text{train}}$) and 20% to testing ($X_{\text{test}}, y_{\text{test}}$). The split is performed using a fixed random seed of 42 to ensure reproducibility across runs. Feature scaling is not used in this experiment.

A.12.3. Model and Training

The classification model employed is provided by SciKit-Learn's `RandomForestClassifier` class.

A.12.4. Experimental Procedure

The experiment is repeated 1000 times for statistical relevance. For each iteration:

1. The dataset is split into training and test sets.
2. The `RandomForestClassifier` model is initialized and trained, with training time recorded.
3. Predictions are generated on the test set.
4. Accuracy is calculated by comparing predictions to the true test labels.
5. Accuracy and training time are tracked separately.

This procedure records the accuracy scores and training times across each cycle, which are later used to calculate the average values for each metric.

A.12.5. Analysis

The collected accuracies and training times enable statistical analysis of the model's performance. Key statistics, such as the mean and standard deviation of accuracy, can be computed to evaluate consistency and generalization. Similarly, training time statistics provide insight into the computational cost of the `RandomForestClassifier` implementation without any sampling.

A.12.6. Assumptions and Limitations

The experiment assumes that the dataset is suitable for classification and that the features.

A.12.7. Expected Outcomes

The experiment is expected to yield a distribution of accuracy scores reflecting the model's predictive capability and a distribution of training times indicating its efficiency. These results can be used to benchmark `RandomForestClassifier` against other implementations or algorithms under identical conditions.

A.12.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 13**.

A.13. Experiment 14: SciKit-Learn's `RandomForestClassifier` Model's Performance with Leverage Score Sampling

A.13.1. Objective

The experiment aims to assess the performance of SciKit-Learn's `RandomForestClassifier` model, in terms of classification accuracy and training time across multiple runs on an open-source dataset (Aeberhard & Forina, 1992). The experiment seeks to quantify the model's generalization ability and computational efficiency under standardized conditions, incorporating a data sampling step using the randomized sampling algorithm `BasicLeverageScoresSampler`.

A.13.2. Materials and Methods

The experiment utilizes a dataset represented by feature matrix $X \in \mathbb{R}^{n \times m}$ and target vector $y \in \{0, 1\}^n$, where n

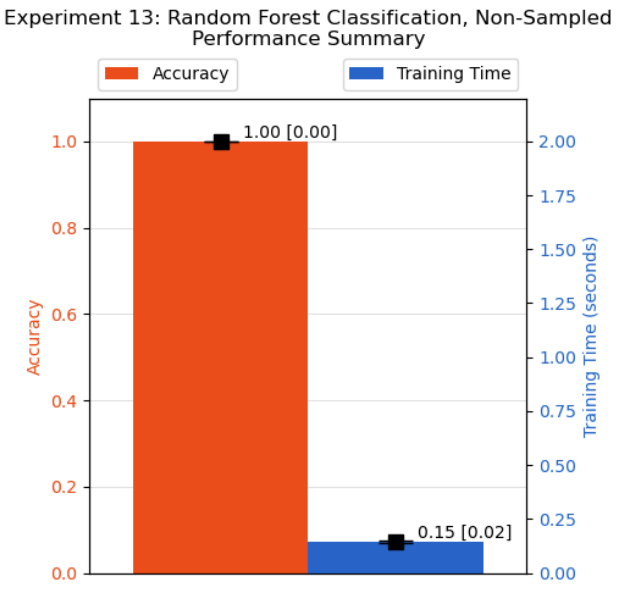

Figure 13. Mean accuracies and training times observed over 1000 training cycles of `RandomForestClassifier`, without any sampling, using the Wine Dataset (Aeberhard & Forina, 1992).

is the number of samples and m is the number of features. The dataset is assumed to be preprocessed and suitable for classification tasks.

For each experimental run, the dataset is first sampled to 20% of its original size using the `BasicLeverageScoresSampler`, resulting in a reduced dataset $(X_{\text{sampled}}, y_{\text{sampled}})$. This sampled dataset is then split into training and test sets using an 80-20 partition, with 80% of the sampled data allocated to training $(X_{\text{train}}, y_{\text{train}})$ and 20% to testing $(X_{\text{test}}, y_{\text{test}})$. The split is performed using a fixed random seed of 42 to ensure reproducibility across runs. Feature scaling is not used in this experiment.

A.13.3. Model and Training

The classification model employed is provided by SciKit-Learn's `RandomForestClassifier` class.

A.13.4. Experimental Procedure

The experiment is repeated 1000 times for statistical relevance. For each iteration:

1. The dataset is sampled to 20% of its original size using `BasicLeverageScoresSampler`.
2. The sampled dataset is split into training and test sets.
3. The `RandomForestClassifier` model is initialized and trained on the training data, with training time recorded.
4. Predictions are generated on the test set.
5. Accuracy is calculated by comparing predictions to the true test labels.
6. Accuracy and training time are tracked separately.

This procedure records the accuracy scores and training times across each cycle, which are later used to calculate the average values for each metric.

A.13.5. Analysis

The collected accuracies and training times enable statistical analysis of the model's performance. Key statistics, such as the mean and standard deviation of accuracy, can be computed to evaluate consistency and generalization. Similarly, training time statistics provide insight into the computational cost of the `RandomForestClassifier` implementation with leverage score sampling.

A.13.6. Assumptions and Limitations

The experiment assumes that the dataset is suitable for classification. Additionally, the use of leverage score sampling assumes that the selected 20% subset adequately represents the original dataset's structure.

A.13.7. Expected Outcomes

The experiment is expected to yield a distribution of accuracy scores reflecting the model's predictive capability and a distribution of training times indicating its efficiency. These results can be used to benchmark `RandomForestClassifier` against other implementations or algorithms under identical conditions, and particularly to evaluate the impact of leverage score sampling algorithm `BasicLeverageScoresSampler` impacts on prediction accuracy and model training time.

A.13.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 14**.

A.14. Experiment 15: SciKit-Learn's `RandomForestClassifier` Model's Performance with Random Sampling

A.14.1. Objective

The experiment aims to assess the performance of SciKit-Learn's `RandomForestClassifier` model, in terms of classification accuracy and training time across multiple runs on an open-source dataset (Aeberhard & Forina, 1992).

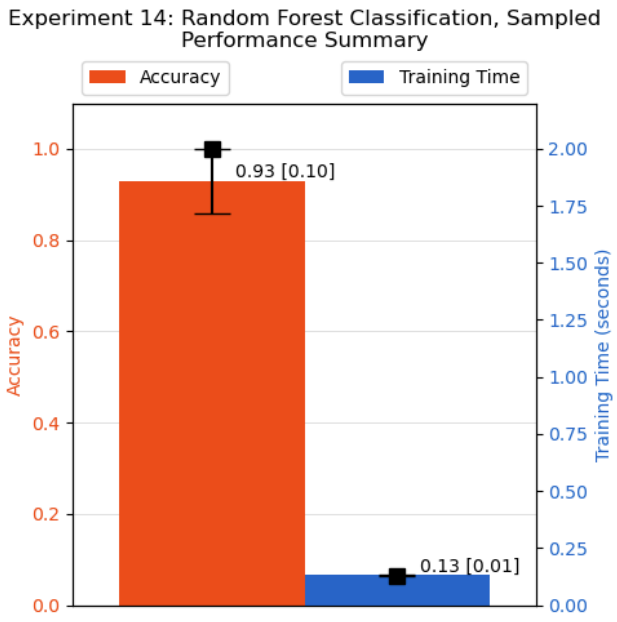

Figure 14. Mean accuracies and training times observed over 1000 training cycles of `RandomForestClassifier`, with `BasicLeverageScoresSampler` sampling, using the Wine Dataset (Aeberhard & Forina, 1992).

The experiment seeks to quantify the model's generalization ability and computational efficiency under standardized conditions, incorporating a data sampling step using the randomized sampling algorithm `RandomSampler`.

A.14.2. Materials and Methods

The experiment utilizes a dataset represented by feature matrix $X \in \mathbb{R}^{n \times m}$ and target vector $y \in \{0, 1\}^n$, where n is the number of samples and m is the number of features. The dataset is assumed to be preprocessed and suitable for classification tasks.

For each experimental run, the dataset is first sampled to 20% of its original size using the `RandomSampler`, resulting in a reduced dataset $(X_{\text{sampled}}, y_{\text{sampled}})$. This sampled dataset is then split into training and test sets using an 80-20 partition, with 80% of the sampled data allocated to training $(X_{\text{train}}, y_{\text{train}})$ and 20% to testing $(X_{\text{test}}, y_{\text{test}})$. The split is performed using a fixed random seed of 42 to ensure reproducibility across runs. Feature scaling is not used in this experiment.

A.14.3. Model and Training

The classification model employed is provided by SciKit-Learn's `RandomForestClassifier` class.

A.14.4. Experimental Procedure

The experiment is repeated 1000 times for statistical relevance. For each iteration:

1. The dataset is sampled to 20% of its original size using `RandomSampler`.
2. The sampled dataset is split into training and test sets.
3. The `RandomForestClassifier` model is initialized and trained on the training data, with training time recorded.
4. Predictions are generated on the test set.
5. Accuracy is calculated by comparing predictions to the true test labels.
6. Accuracy and training time are tracked separately.

This procedure records the accuracy scores and training times across each cycle, which are later used to calculate the average values for each metric.

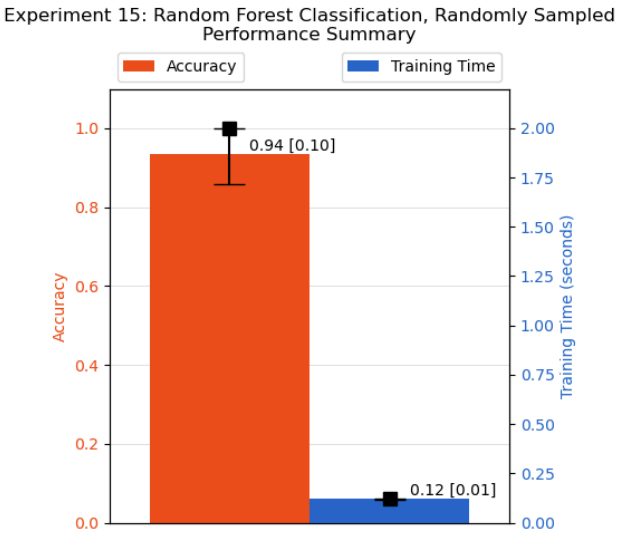

Figure 15. Mean accuracies and training times observed over 1000 training cycles of `RandomForestClassifier`, with `RandomSampler` sampling, using the Wine Dataset (Aeberhard & Forina, 1992).

A.14.5. Analysis

The collected accuracies and training times enable statistical analysis of the model's performance. Key statistics, such as the mean and standard deviation of accuracy, can be computed to evaluate consistency and generalization. Similarly, training time statistics provide insight into the

computational cost of the `SVC` implementation with random sampling.

A.14.6. Assumptions and Limitations

The experiment assumes that the dataset is suitable for classification. Additionally, the use of random sampling assumes that the selected 20% subset adequately represents the original dataset's structure.

A.14.7. Expected Outcomes

The experiment is expected to yield a distribution of accuracy scores reflecting the model's predictive capability and a distribution of training times indicating its efficiency. These results can be used to benchmark `RandomForestClassifier` against other implementations or algorithms under identical conditions, and particularly to evaluate the impact of leverage score sampling algorithm `RandomSampler` impacts on prediction accuracy and model training time.

A.14.8. Measured Outcomes

The expected outcomes for this experiment have been empirically confirmed, as illustrated in **Figure 15**.