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. Literature Review

1.1. Background

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.

[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.

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.

1.2. Problem Definition

This work aims to explore potential solutions to the computational challenges posed by the use of high-dimensional datasets in the context of prediction models. Objectively, we evaluate the impacts, implementations, and viable utilizations for a randomized sampling algorithm originally designed for logistic regression models, as outlined by Chowdhury & Ramuhalli (2024). Given a dataset, the proposed method derives a smaller subset that retains the most representative data-points, using it to train a prediction model capable of achieving accuracy levels comparable to models trained on the entire dataset. As demonstrated both theoretically and experimentally, the algorithm is successful at substantially reducing the size of training datasets, addressing many barriers found when handling large databases in machine learning applications.

Given the predominance of digital systems driven by big-

data in modern society, investigating implementation variations and applicability potential is a justified effort, and may help to further understand possible trade-offs between algorithmic simplicity and prediction accuracy, as well as effectiveness in diverse data configurations.

1.3. Summary and Critique of the Selected Papers

1.3.1. Main Paper: A Provably Accurate Randomized Sampling Algorithm for Logistic Regression (Chowdhury & Ramuhalli, 2024)

Summary: Logistic regression is a widely used method for binary classification tasks, where the goal is to predict one of two possible outcomes based on input features (Murphy, 2022). However, when the number of observations greatly exceeds the number of predictor variables, solving logistic regression becomes computationally expensive. In this work, Chowdhury & Ramuhalli propose a novel approach to address this issue by leveraging a randomized sampling technique that guarantees high-quality approximations to the estimated probabilities of logistic regression, with a sample size significantly smaller than the total number of observations. The primary goal is to make logistic regression more computationally efficient for large-scale datasets while maintaining prediction accuracy.

The paper focuses on the use of leverage scores, which is a method that, given a dataset, allows for the extraction of a subset where the derived data points retain the main characteristics of the overall data (Ordozgoiti et al., 2022). The authors' proposed algorithm constructs a sampling matrix using these scores, and the resulting subsampled log-likelihood function is optimized to estimate the model parameters. This randomized approach allows for substantial reductions in computational cost while preserving the accuracy of the regression model. The paper provides a theoretical derivation of an approximation bound that guarantees the accuracy of the estimated probabilities obtained from the subsampled data. This result is significant because it shows that only a small subset of the data may be needed to achieve a high-quality approximation of the original data, making the algorithm more scalable and efficient.

To validate their approach, the authors conduct extensive empirical evaluations on real-world datasets, including a cardiovascular disease prediction dataset, a customer churn prediction dataset, and a credit card default prediction dataset. The experiments compare the performance of the proposed leverage score-based sampling method with other sampling schemes, including uniform sampling and the L2S method proposed by Munteanu et al.. The results demonstrate that the leverage score-based method performs comparably to other methods in terms of the accuracy of estimated probabilities, and shows potential for outperformance, particularly as the sample size increases. Moreover, the method achieves misclassification rates close to those obtained by the full-data model, further confirming its effectiveness. These empirical results validate the theoretical claims and show that the proposed algorithm can deliver accurate logistic regression estimates with a reduced sample size.

The paper concludes with a discussion about the implications of their findings and suggesting future research directions. While the proposed method performs well in practice, the authors highlight that there is still room for improvement in terms of algorithmic efficiency and scalability, particularly in high-dimensional settings. They also suggest exploring alternative sketching techniques, such as random projection-based methods, to further enhance the sampling process. Additionally, the paper points out that the approach could be extended to handle other forms of regression and machine learning models, making it a valuable tool for a variety of applications. The paper makes a significant contribution by providing an efficient and scalable solution to logistic regression for large datasets, with strong theoretical guarantees and solid empirical results.

Critique: While the paper provides an excellent approach for logistic regression, it does not explore the application of the same sampling algorithm to other types of prediction models. The authors focus exclusively on logistic regression, which is understandable given the complexity and computational challenges associated with large datasets in this context. However, the question of whether the method could potentially be adapted for use with other models, such as linear regression, support vector machines, or even more complex deep learning models, remain unclear. Given that dataset distillation has been an area of interest in machine learning (Liu et al., 2023), a discussion of how their technique could be generalized or modified for broader applications range would have added valuable insight to the work. Exploring the adaptability of the sampling strategy to a range of prediction models could increase the impact and versatility of the proposed method, making it applicable to a wider array of real-world problems.

Additionally, even though the paper does provide a robust theoretical foundation for the proposed method, it could benefit from a more detailed exploration of the trade-offs between simplicity and accuracy, particularly when it comes to the complexity of the algorithm. While the proposed algorithm is presented as computationally efficient, the authors do not thoroughly investigate alternative variations of the algorithm that might provide different levels of complexity or accuracy. For example, they focus heavily on the use of row leverage scores to select data points, but it would be valuable to see whether simpler strategies could yield comparable performance. Moreover, a discussion of whether implementing theoretical guarantees—such as the approximation

bounds—justifies the added complexity would provide a clearer understanding of the practical trade-offs involved. The paper does an excellent job of presenting an efficient solution, but more emphasis on evaluating the balance between theoretical rigor and real-world applicability could enrich the findings.

Another area where the paper could expand is in providing a more nuanced analysis of the error bounds and their impact on the overall model performance. While the theoretical guarantees are a key strength of the paper, there is limited exploration of how these bounds manifest in practice when applied to different types of datasets. It would have been helpful for the authors to conduct experiments that show how varying the sample size and error tolerance affects both the accuracy of the model and the computational efficiency. In addition, the relationship between the size of the dataset, the model's complexity, and the effectiveness of the approximation would have benefited from further elaboration. For example, is there a point at which the error bounds no longer provide substantial improvements, or do they continue to yield significant benefits with increasing complexity? Addressing these questions could further clarify when and why this sampling-based approach is most beneficial, providing a more comprehensive evaluation of its utility in real-world applications.

1.3.2. Supplementary Paper: SLOE: A Faster Method for Statistical Inference in High-Dimensional Logistic Regression (Yadlowsky et al., 2021)

Summary: In high-dimensional problems, where the number of features is comparable to or exceeds the number of samples, traditional logistic regression methods, specifically maximum likelihood estimation (MLE), commonly present poor performance. The existing large-sample asymptotic theory for logistic regression approximations fails in these scenarios, leading to inadequate parameter estimates and unreliable statistical inference.

Yadlowsky et al. propose a novel solution, the Signal Strength Leave-One-Out Estimator (SLOE), to efficiently estimate the signal strength parameter, which is crucial for performing dimensionality corrections. This method significantly improves the practical application of the corrections developed in previous works, making it computationally feasible and faster to implement in real-world applications.

At the core of this paper is the challenge of estimating the signal strength, which is used in dimensionality corrections to adjust the bias and variance in high-dimensional **logistic regression**. Previous methods, such as the ProbeFrontier heuristic (Sur & Candès, 2019), attempted to estimate this signal strength but were computationally expensive and conceptually complex. The proposed SLOE method reparameterizes the problem in terms of a more easily estimated "corrupted" signal strength, which accounts for the noise in the parameter estimates due to finite sample sizes. The paper demonstrates that using this reparameterization, the bias and variance adjustments for MLE can be performed accurately, even in finite samples, which is a significant improvement over traditional methods that rely on large-sample approximations.

One of the key advantages of SLOE is its computational efficiency. While previous heuristics like ProbeFrontier required multiple subsampling iterations and linear programming to estimate the signal strength, SLOE uses a simple and fast approach by leveraging leave-one-out (LOO) techniques. The method estimates the corrupted signal strength using a rank-one update formula that avoids refitting the model multiple times. The paper shows that SLOE achieves a substantial reduction in computation time compared to ProbeFrontier, making it a practical tool for routine use in high-dimensional logistic regression. This efficiency gain is particularly important when working with large datasets, where computational speed is crucial.

The authors provide a detailed theoretical analysis of the SLOE method, showing that it consistently estimates the corrupted signal strength, which is asymptotically equivalent to the true signal strength. The theoretical guarantees underpin the practical reliability of SLOE in high-dimensional settings. Through extensive simulations, the paper illustrates how SLOE improves the accuracy of confidence intervals (CIs) and p-values in logistic regression models. In particular, the corrected CIs generated by SLOE are shown to provide reliable coverage, even in high-dimensional settings where traditional large-sample approximations fail. This is especially important for making sound statistical inferences in fields like genomics and clinical applications, where reliable uncertainty quantification is critical.

This paper ultimately makes an important contribution to the field of high-dimensional statistical inference. The SLOE method introduced by the authors provide a simpler, faster, and more accurate approach to estimating signal strength in high-dimensional logistic regression. SLOE offers a practical solution to the problems caused by large-dimensional data, enabling more reliable statistical inference in these settings. The method is tested and validated on several real-world datasets, including applications to heart disease prediction and genomics, where it outperforms traditional methods in terms of both computational efficiency and statistical accuracy. SLOE paves the way for more robust and computationally feasible dimensionality corrections, making it a valuable tool for applied data science and statistical modeling.

Critique: While the paper addresses the high-dimensional logistic regression setting under the assumption of Gaussian

or sub-Gaussian features, it could have further explored how SLOE performs when the data distribution deviates from these assumptions. Many real-world datasets, for example those encountered in genomics, healthcare, or social sciences, often exhibit non-Gaussian characteristics. The authors briefly mention that SLOE works well in the presence of sub-Gaussian features, but a more thorough analysis of how the method generalizes to different data distributions would provide a clearer picture of it performs in diverse settings. A discussion on this would also be helpful in understanding whether additional steps are needed when dealing with non-Gaussian data, making the method even more applicable across different domains.

In addition, since the paper focuses on overcoming challenges in high-dimensional logistic regression, it could have offered a more detailed comparison to other dimensionality reduction techniques commonly used in similar settings, such as principal component analysis (PCA), partial least squares (PLS), or feature selection methods. These methods are often used in high-dimensional regression problems to reduce the number of predictors, making it more manageable to estimate models. A brief discussion on how SLOE compares or complements these dimensionality reduction techniques would provide additional insights on when to use SLOE versus other popular methods, and whether it offers specific advantages or disadvantages depending on the type of dataset or the modeling goal.

Lastly, one area that could have been explored in more depth is the scalability of the SLOE method to extremely large datasets. The computational efficiency of SLOE is emphasized, particularly in comparison to ProbeFrontier, but the paper could have provided more concrete examples or benchmarks when the dataset size approaches millions of samples, which can be of particular interest in modern real-world applications driven by big data. While the authors demonstrate that SLOE is much faster than its competitors, additional discussion around its performance in ultra-large datasets would be valuable, as it would allow the paper to transcend the academic domain and make a connection with common industry challenges.

1.3.3. Supplementary Paper: Stable learning via sample reweighting (Shen et al., 2020)

The paper aims to address the problem of model instability in machine learning, especially when there is a mismatch between training and test data distributions. In real-world scenarios, it is often unrealistic to assume that the training data and the test data come from the same distribution. This discrepancy can lead to unreliable performance, especially when the model relies heavily on collinear input variables. Shen et al. aim to develop a method that ensures stability in predictions, even when the underlying data distribution changes. Their work is focused on linear models, which are commonly used for regression and classification tasks but are sensitive to collinearity, which can significantly distort parameter estimates and lead to unstable predictions.

A central challenge addressed in this paper is the impact of collinearity on model stability. Collinearity occurs when there is a high correlation between input variables, leading to a sub-optimal design matrix. This problem is particularly notable in high-dimensional settings, where variables are highly interdependent. The authors argue that traditional methods, such as ordinary least squares (OLS), are susceptible to this issue, causing large errors in parameter estimation when the training data differs from the test data. In particular, they show that even small model misspecifications can lead to large errors in predictions due to the instability introduced by collinearity. Thus, the paper emphasizes the need for a robust approach that can mitigate the adverse effects of collinearity.

To address this issue, the authors propose a novel method called Sample Reweighted Decorrelation Operator (SRDO). This method works by assigning appropriate weights to samples, effectively reweighting the data to reduce collinearity among input variables. By doing so, the design matrix becomes closer to an orthogonal structure, which is more stable and less prone to errors caused by multicollinearity. The theoretical foundation of SRDO shows that, under ideal conditions, the sample weights can make the design matrix nearly orthogonal, significantly improving the model's stability. The method is presented as a general pretreatment technique, meaning it can be integrated with standard linear regression methods like ordinary least squares (OLS), Lasso, and **logistic regression** to enhance their robustness against distribution shifts.

The paper provides a detailed theoretical analysis of the SRDO method. It demonstrates that, in an idealized setting with infinite sample size, the optimal sample weights can minimize the effects of model misspecification and collinearity. However, the authors also acknowledge the practical challenges when working with finite sample sizes. In such cases, there is a tradeoff between reducing bias and increasing variance, which is common in statistical methods. Despite this, the SRDO method shows a clear advantage in terms of prediction stability and accuracy, especially when the training and test distributions differ significantly. The theoretical results are supported by empirical experiments, which show that SRDO outperforms traditional methods in both regression and classification tasks.

Finally, the paper presents extensive experiments to validate the effectiveness of SRDO. The experiments are conducted using both synthetic and real-world datasets, demonstrating the method's ability to handle collinearity and distribution shifts. The results show that SRDO consistently reduces es-

timation errors and improves prediction accuracy, especially when training and test data distributions are mismatched. For instance, in the context of regression, SRDO significantly outperforms OLS and Lasso, especially in situations with strong collinearity. In classification tasks, the method also demonstrates improved stability when tested on diverse groups of users with varying behaviors. These findings highlight the practical applicability of SRDO in real-world machine learning scenarios, where data distributions are often subject to change. The authors conclude by emphasizing that SRDO is a versatile and valuable tool for improving the stability and reliability of linear models.

Critique: One area that could have been further explored in the paper is the practical application and limitations of SRDO in real-world datasets with varying feature types. While the authors demonstrate the method's effectiveness in handling collinearity and improving stability, the paper focuses primarily on linear models, which might not capture more complex patterns in the data. The authors could have discussed how SRDO performs when applied to datasets that involve both linear and non-linear relationships or datasets with categorical features. This would help extend the method's applicability and provide guidance for researchers working in diverse domains, especially those dealing with non-linear or high-dimensional data. A more in-depth exploration of SRDO's limitations in such contexts would be beneficial for understanding where the method might fall short or require adaptations.

Furthermore, the authors could have more thoroughly explored the computational complexity and scalability of the proposed method. Despite the strong focus on the theoretical benefits and empirical performance of SRDO, there is little discussion on the practical aspects of implementing the method at scale. The process of calculating and applying sample weights, especially for large datasets with high-dimensional feature spaces, could be computationally expensive. This challenge is particularly important for real-world applications where datasets can contain millions of samples and variables. An analysis of the time complexity of the method, along with potential optimizations or approximations, could provide valuable insights when applying SRDO in large-scale problems.

Another aspect that could have been addressed more thoroughly is the relationship between SRDO and existing regularization techniques for handling collinearity, such as Ridge or Lasso regression. While the paper emphasizes the novel approach of reweighting samples to reduce collinearity, it would have been useful to see a direct comparison with these established methods to better understand whether there are scenarios in which one method presents advantages relative to the others.

2. Implementation

In order to evaluate the impacts of the sampling algorithm proposed by Chowdhury & Ramuhalli (Chowdhury & Ramuhalli, 2024), as well as the simplified version proposed by this study, we present the implementation efforts described in this section.

2.1. Baseline Setup: Slow Logistic Regression

2.1.1. Overview

In this work, we set a performance baseline 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.

2.1.2. 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.

2.1.3. 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.

2.1.4. 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:

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

2.1.5. 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.

2.1.6. 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.

2.1.7. 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.

2.1.8. 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:

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

2.2. Randomized Sampling Algorithm (Simplified)

2.2.1. 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.

2.2.2. Leverage Scores Calculation

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.

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: $X\_sampled, y\_sampled$

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.

2.2.3. Sampling

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.

2.2.4. BasicLeverageScoresSampler versus Chowdhury & Ramuhalli's Approach

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.

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.

2.3. Initial Results

Preliminary experiments have been conducted, as detailed in Appendix A, to evaluate whether any impacts are observable with the integration of the sampling algorithm implemented in this study. Overall, positive results have been measured due to the application of the `BasicLeverageScoresSampler` algorithm.

Figure 1 shows `SlowLogisticRegression`'s performance without any sampling over the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993). In **Figure 2**, we note the average accuracy of 0.97, and average training time of 0.96 seconds over 20 runs of the experiment. After repeating the experiment, this time employing `BasicLeverageScoresSampler`, we measured a substantial reduction of 80% in the training time, as illustrated in **Figure 3** and **Figure 4**. These experiments allowed us to isolate the sampling algorithm and evaluate its impact in a setting where no other optimizations exist. As demonstrated here, the accuracy impact has been negligible, down to 0.96% from 0.97%. While 1% may be significant in ultra-optimized implementations that aim for the highest possible accuracy, many practical scenarios might be satisfied with slightly lower accuracy levels.

Experiments 3 and 4, illustrated in **Figure 5** and **Figure 6**, respectively, apply `BasicLeverageScoresSampler` to the same dataset, but this time utilizes class `LogisticRegression` from the SciKit-Learn Python library. The experiments did not allow for statistically relevant observations, alluding to the possibility that `LogisticRegression` may count with optimizations that can potentially minimize the impact of dataset size when training the model. Further exploration is required to determine whether experiments conducted with larger datasets under the same conditions yield the same results.

Experiment 5, documented in **Figure 7**, amplifies the statistical relevance of the observed results by repeating experiments 1, 2, 3, and 4 under the same conditions, but this time over 1000 cycles. Equivalent results have been measured as a result of the experiment, confirming the observations made earlier.

2.4. Next Steps

In light of the positive preliminary results observed in this initial exploratory phase, we propose the following next steps to give continuity to this study:

- **Complete Sampling Algorithm Implementation:** As stated earlier, this preliminary exploration implemented a simplified version of the algorithm proposed by Chowdhury & Ramuhalli. To provide more depth and substance to this study, a full implementation of the algorithm is necessary. This effort will allow for more comprehensive benchmarking and trade-off analysis, as well as a better understanding of the circumstances in which the different variations demonstrate better (or worse) employability.
- **Sample Size Impacts:** To delve deeper into the subject of data-sampling, we plan on conducting an analysis to evaluate the impact of sample size in accuracy and training times for models trained with sampled data. In other words, we will seek to understand how different sample sizes affect the key metrics, and answer the question of whether an optimal point exists that allow for best prediction accuracy at a reasonable cost of training time.
- **Use-Case Diversification:** Chowdhury & Ramuhalli focus their work on logistic regression models. Given that dataset reduction is a topic of common interest in the Machine Learning field, an exploration of whether the sampling algorithms implemented in this project can be used to train other types of prediction models is justified. Therefore, we propose a more substantial analysis into this question as a natural next step for this study.

2.5. Proposal for Course Project

2.5.1. Overview

Our proposal for this study is multifold. Ultimately, we aim to provide (1) an implementation of the full sampling algorithm as originally proposed by Chowdhury & Ramuhalli, (2) a simplified version that aims to strike a reasonable balance between computational performance and prediction accuracy, and (3) an empirical evaluation of the technical feasibility of employing the sampling algorithms implemented here on the training process of other types of prediction models beyond logistic regression.

2.5.2. Methodology

To achieve the aforementioned goals, a thorough benchmark is necessary to evaluate how the simplified algorithm performs relative to the full implementation in diverse 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 will be 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 simplified and complete versions of the algorithms will be tested with different databases, such as those used by Chowdhury & Ramuhalli, and also well-known open-source datasets, such as the Breast Cancer Wisconsin Diagnostic Data Set (Wolberg et al., 1993), among others. Further details regarding experiment design are 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. At this preliminary stage, the types of predictions scoped for further exploration are **to be defined**.

2.5.3. Feasibility

Given the empirical nature of this study, the project so far has demonstrated to be technically feasible, from an implementation standpoint. The paper (Chowdhury & Ramuhalli, 2024) provides a thorough review of the mathematical background, as well as detailed information behind the algorithm's rationale and theoretical foundations. Additionally, the authors provide their own implementation in Python code, which is publicly available and may be used for benchmarking as part of this study.

2.5.4. Relevance

Our literature review indicates numerous efforts in the machine learning community in pushing the barriers of predictive algorithms, and in seeking creative solutions to improve accuracy and optimize computational efficiency of known methods. In particular, the core contribution of this project involves addressing questions left open by Chowdhury & Ramuhalli, such as (1) are the theoretical guarantees provided by the original implementation indispensible, or could a simplified version of the algorithm provide comparable performance? And (2) can this sample method contribute positively to the training process of different machine learning models? By addressing these questions, our work further expands the body of knowledge proposed by the authors, and provides the academic community with further insights into the use of randomized sampling algorithms in reducing large training datasets.

3. Conclusion

This project proposes an exploration of data-sampling as a creative strategy for reducing large bodies of data to representative subsets that allow for the training of accurate prediction models at a smaller computational cost. The integration of a sampling algorithm based on leverage scores, according to this preliminary evaluation, provided valuable insights into the potential benefits of employing this method in the context of logistic regression models, specifically in reducing the training dataset without substantial loss of prediction quality. In our view, this work justifies further efforts to analyze different implementations, as well as address the open question of whether similar impacts can be observed when training other types of models. We are confident that this project can provide meaningful contributions to the field of Machine Learning, specifically in addressing open questions regarding dimensionality reduction as a training optimization strategy.

Software and Data

Download links to the code that accompanies this study to be provided after the anonymous peer review.

Impact Statement

This paper presents a study which goal is to advance the field of Machine Learning, particularly in what is known regarding dataset reduction. There are many potential societal implications of our work, none which we feel must be specifically highlighted here.

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**.

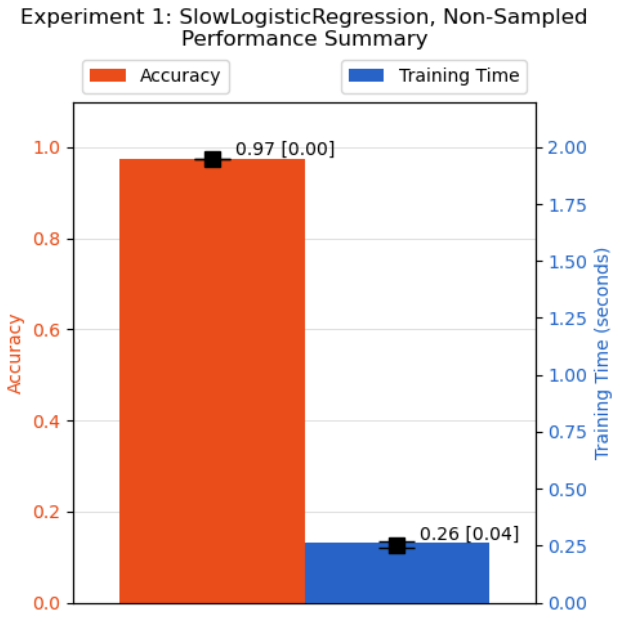

Figure 1. Mean accuracies and training times observed over 100 training cycles of `SlowLogisticRegression`, without any sampling, using the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993).

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**.

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.

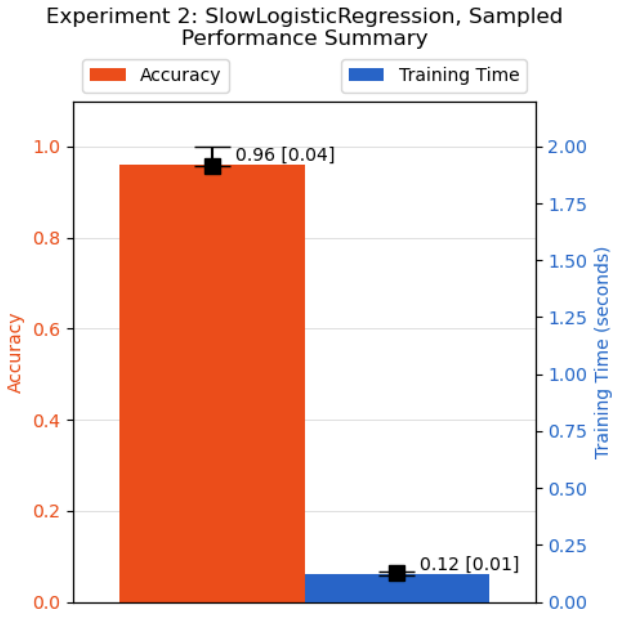

Figure 2. Mean accuracies and training times observed over 100 training cycles of `SlowLogisticRegression`, with `BasicLeverageScoresSampler` sampling, using the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993).

A.3.2. 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**.

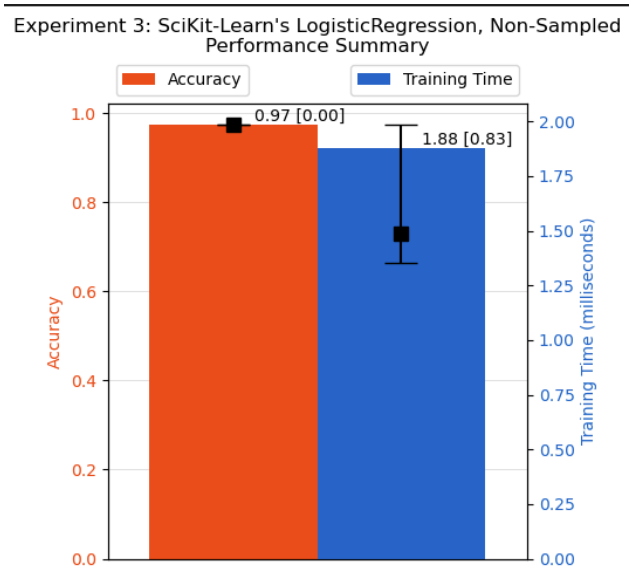

Figure 3. Mean accuracies and training times observed over 100 training cycles of SciKit-Learn's `LogisticRegression` class, without any sampling, using the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993).

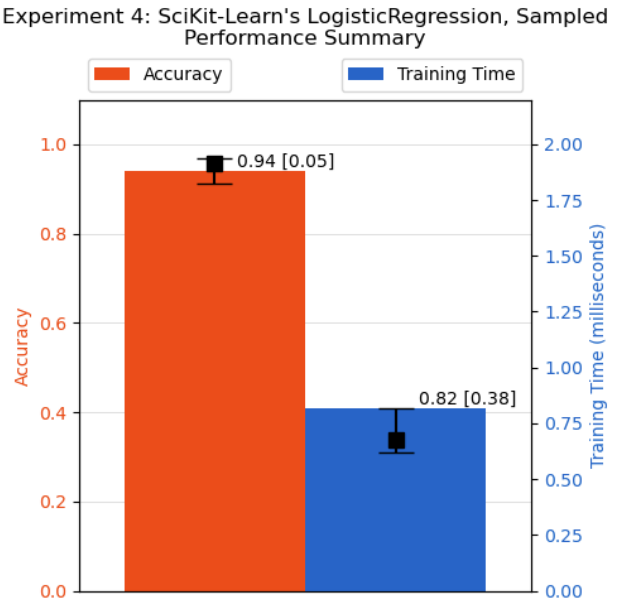

Figure 4. Mean accuracies and training times observed over 100 training cycles of SciKit-Learn's `LogisticRegression` class, with `BasicLeverageScoresSampler` sampling, using the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993).

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**.

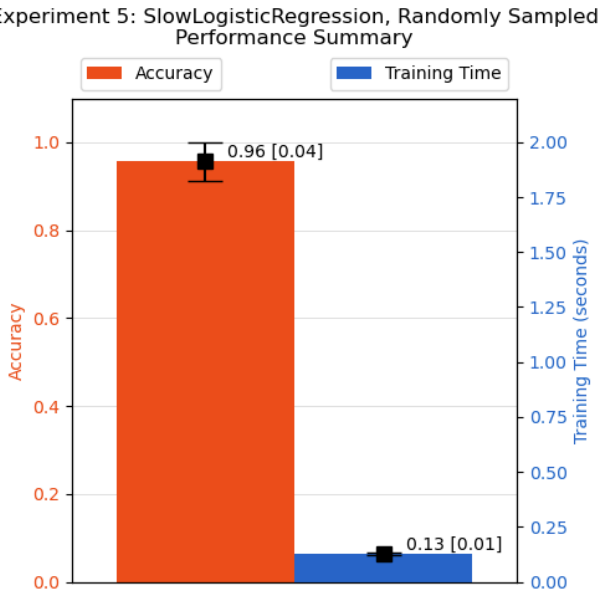

Figure 5. Mean accuracies and training times observed over 20 training cycles of `SlowLogisticRegression`, with `RandomSampler` sampling, using the Breast Cancer Wisconsin Diagnostic Dataset (Wolberg et al., 1993).

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**.

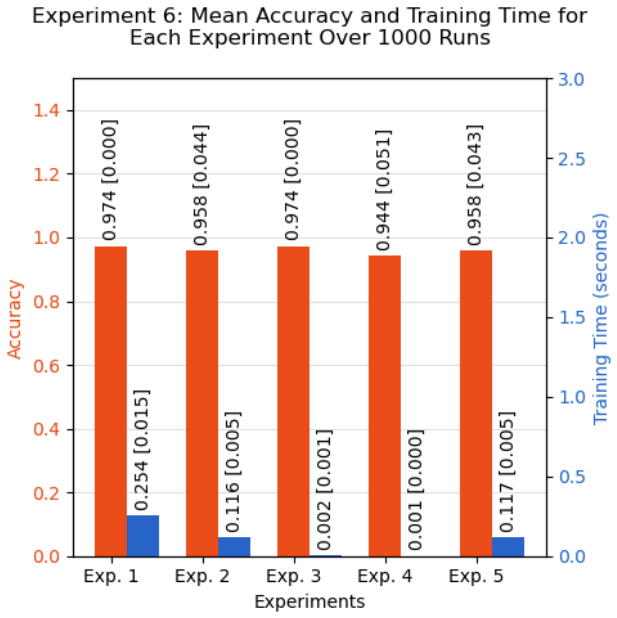

Figure 6. Mean accuracies and training times observed when repeating experiments 1, 2, 3, 4, and 5, this time with 1000 training cycles each.

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 100 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**.

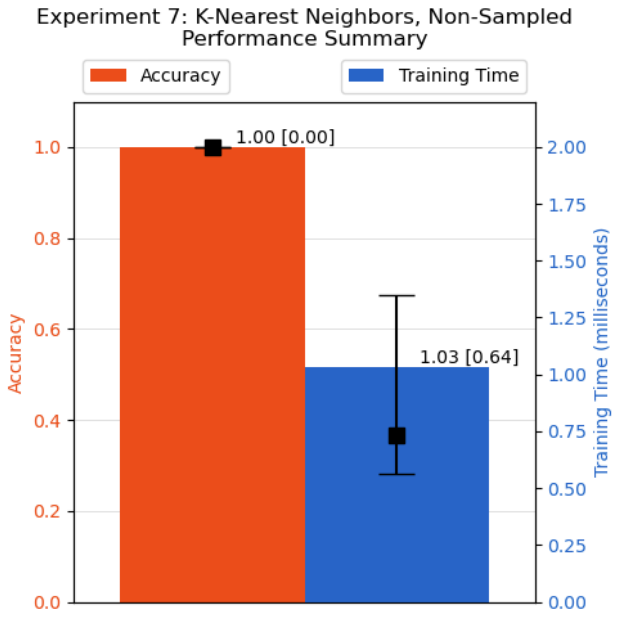

Figure 7. Mean accuracies and training times observed over 100 training cycles of `KNeighborsClassifier`, without any sampling, using the Iris Dataset (Fisher, 1936).

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 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. 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**.

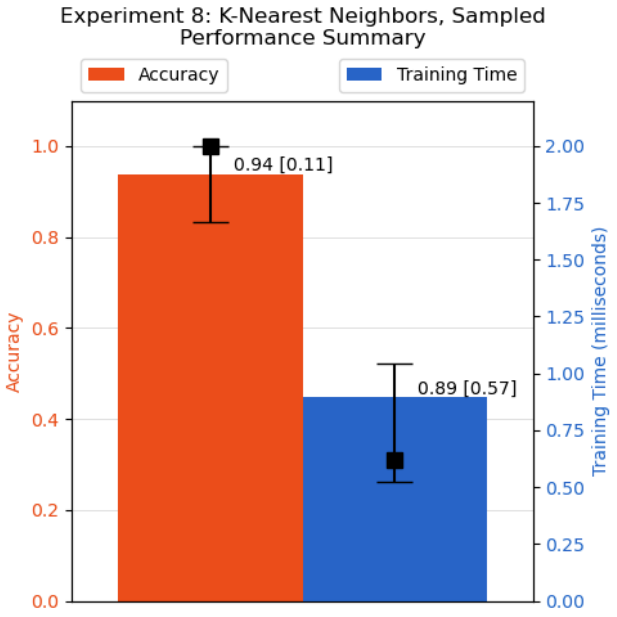

Figure 8. Mean accuracies and training times observed over 100 training cycles of `KNeighborsClassifier`, with `BasicLeverageScoresSampler` sampling, using the Iris Dataset (Fisher, 1936).

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 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. 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**.

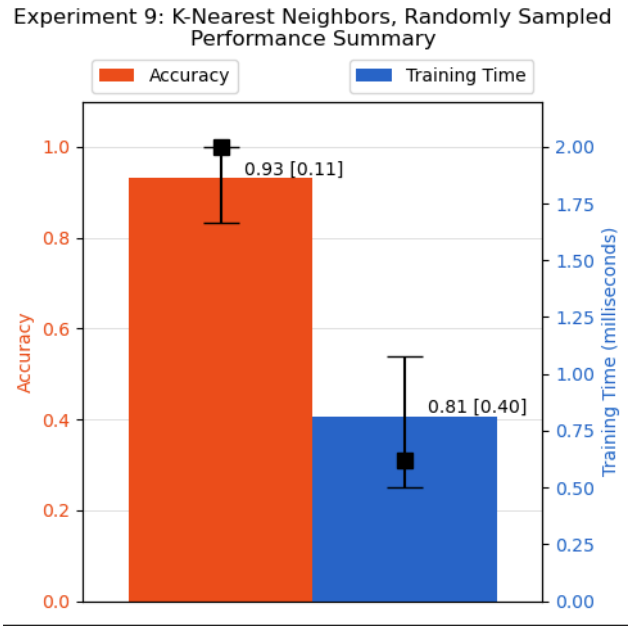

Figure 9. Mean accuracies and training times observed over 100 training cycles of `KNeighborsClassifier`, with `RandomSampler` sampling, using the Iris Dataset (Fisher, 1936).

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 100 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**.

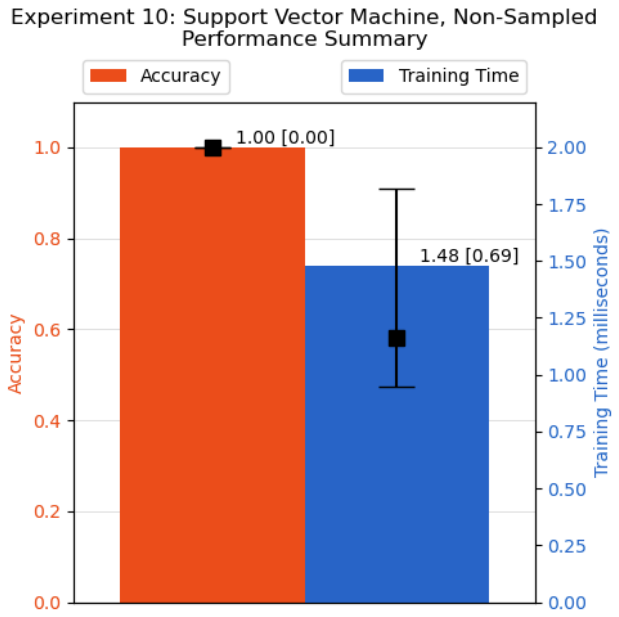

Figure 10. Mean accuracies and training times observed over 100 training cycles of `SVC`, without any sampling, using the Iris Dataset (Fisher, 1936).

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 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. 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

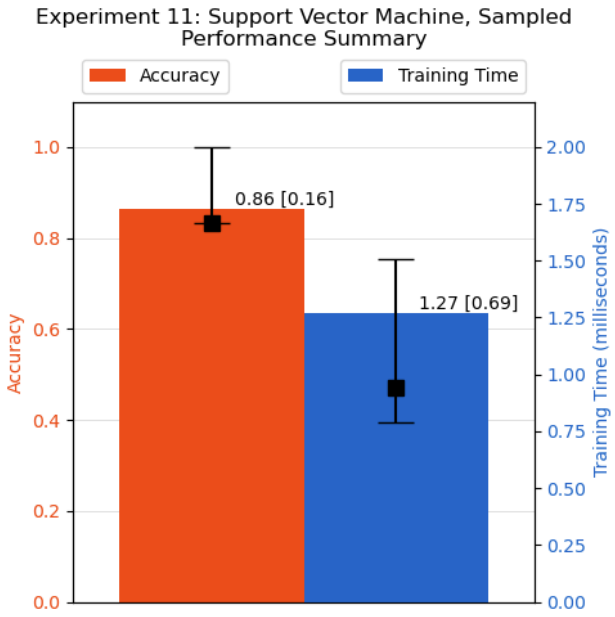

Figure 11. Mean accuracies and training times observed over 100 training cycles of `SVC`, with `BasicLeverageScoresSampler` sampling, using the Iris Dataset (Fisher, 1936).

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 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. 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.

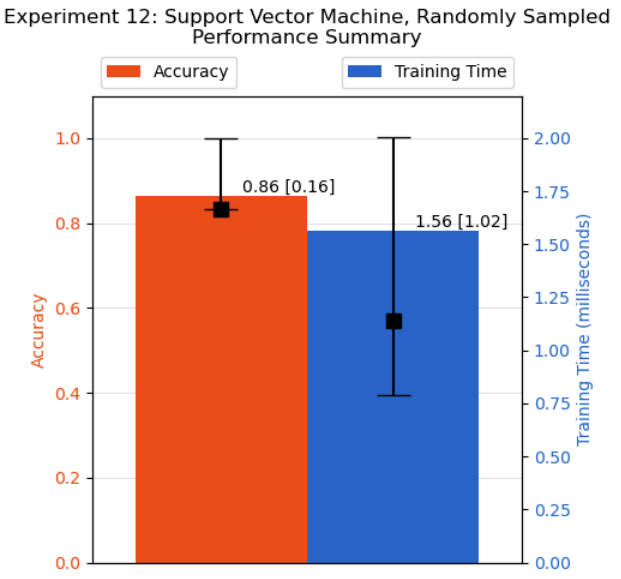

Figure 12. Mean accuracies and training times observed over 100 training cycles of `SVC`, with `RandomSampler` sampling, using the Iris Dataset (Fisher, 1936).

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 100 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

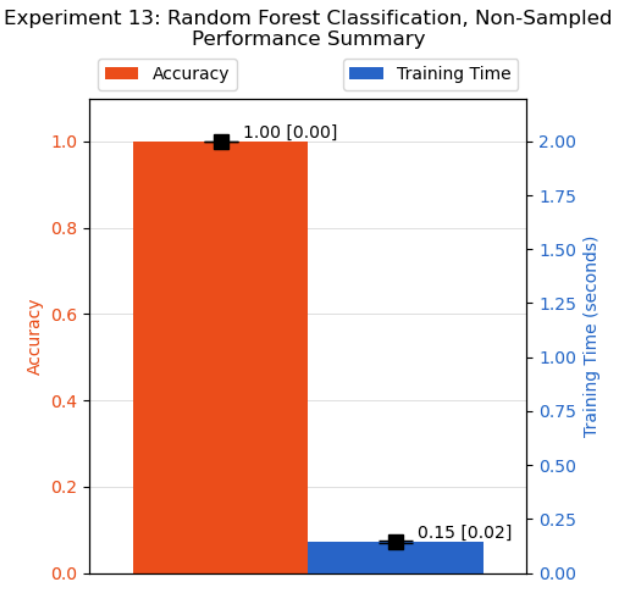

Figure 13. Mean accuracies and training times observed over 100 training cycles of `RandomForestClassifier`, without any sampling, using the Wine Dataset (Aeberhard & Forina, 1992).

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 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. 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).

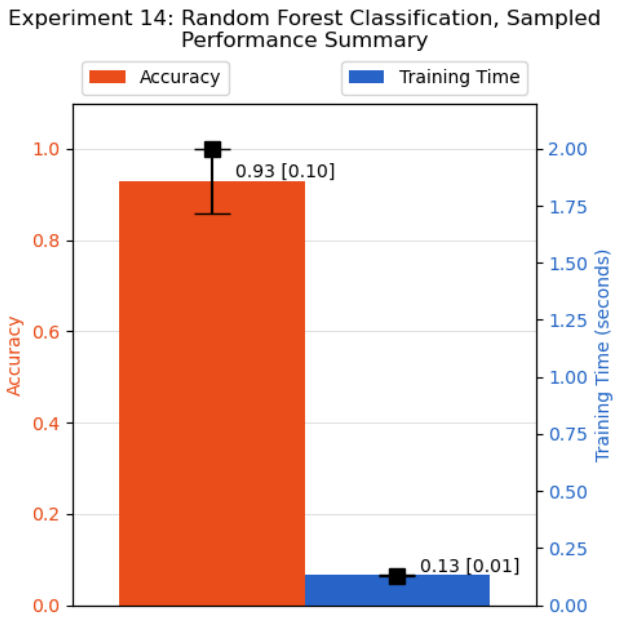

Figure 14. Mean accuracies and training times observed over 100 training cycles of `RandomForestClassifier`, with `BasicLeverageScoresSampler` sampling, using the Wine Dataset (Aeberhard & Forina, 1992).

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 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. 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.

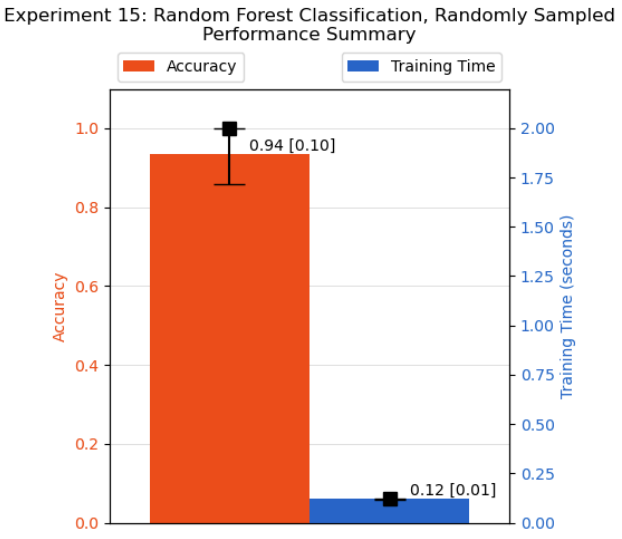

Figure 15. Mean accuracies and training times observed over 100 training cycles of `RandomForestClassifier`, with `RandomSampler` sampling, using the Wine Dataset (Aeberhard & Forina, 1992).

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**.