CS 323 Project Report: Analysis of Logistic Regression in Machine Learning

By: Farha Ferdous

City University of New York, Queens College

***Abstract*— This project investigates the performance of logistic regression, a widely used machine learning algorithm, by conducting a comprehensive analysis and benchmarking against other classification algorithms. Logistic regression is particularly useful for binary classification tasks, where the goal is to model the probability of a binary outcome based on one or more predictor variables. The project explores the algorithm's efficiency, scalability, and accuracy. Key performance metrics, including accuracy, precision, recall, F1 score, and computational time, are evaluated across different synthetic datasets with varying complexities and sizes. The goal is to provide insights into when logistic regression performs optimally and under what conditions it might be outperformed by other algorithms. The analysis also delves into the impact of feature selection, regularization techniques, and model tuning on logistic regression's performance. This project ultimately aims to provide a deeper understanding of logistic regression's strengths and limitations in real-world classification problems, offering practical recommendations for its use in algorithmic decision-making.**

I. INTRODUCTION

A) Logistic regression is a statistical method used for binary classification problems. Its primary purpose is to model the probability of a binary outcome (e.g., success/failure, 0/1) based on one or more independent variables. The algorithm uses the logistic function (sigmoid) to map predicted values to probabilities between 0 and 1.

The model estimates parameters (coefficients) that maximize the likelihood of the observed data using the maximum likelihood estimation (MLE) method. For a binary classification problem, the model predicts the probability $P(y=1|X)$ as:

$$P(y=1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_n X_n)}}$$

Here, β_0 is the intercept, and βi are the coefficients for the features Xi.

B) PSEUDOCODE

Pseudocode:

1. **Input**: Dataset with features X and labels y
2. **Initialize**: Weights β and bias β_0 (randomly or to zeros)
3. **Repeat until convergence**:
   - Compute the linear combination: $z = X\beta + \beta_0$
   - Apply the sigmoid function: $\hat{y} = \frac{1}{1+e^{-z}}$
   - Compute the gradient of the loss (log-likelihood function):
     - $\nabla_\beta = \frac{\partial \text{Loss}}{\partial \beta}$
     - $\nabla_{\beta_0} = \frac{\partial \text{Loss}}{\partial \beta_0}$
   - Update β and β_0 using gradient descent or similar optimization techniques
4. **Output**: Trained weights β and bias β_0

C) BACKGROUND

Logistic regression is rooted in statistical probability theory, assuming that the log-odds of the dependent variable are a linear combination of the independent variables. This foundational approach involves key mathematical concepts, including the logit function, which uses the natural logarithm of the odds ratio to establish a linear relationship. Additionally, the sigmoid function, a non-linear activation mechanism, maps any real-valued number to a range between 0 and 1, providing probabilistic predictions. The model's parameters are determined using Maximum

Likelihood Estimation (MLE), which optimizes the probability of the observed data under the model. The underlying mathematics of logistic regression incorporates linear algebra for matrix operations, probability for handling uncertainty, and calculus for optimization processes.

D) COMPLEXITY

The time complexity of logistic regression varies depending on the task. During training, the complexity is $O(n \cdot m \cdot i)$, where n represents the number of features, m denotes the number of samples, and i corresponds to the number of iterations in the optimization process. For prediction, the complexity is $O(n+m)$, as the algorithm computes a linear combination of inputs for each sample. Regarding space complexity, the algorithm requires $O(n+m)$, which is allocated for storing the feature matrix X and parameter vector β. This linear dependency on the number of features and samples makes logistic regression computationally efficient for small to moderately sized datasets. However, as the dataset size or feature count increases substantially, the time complexity also grows linearly, highlighting the importance of balancing computational resources with data size.

E) COMPARISONS

Logistic regression differs significantly from other machine learning algorithms. Compared to decision trees, logistic regression is better at generalizing small datasets, as decision trees can overfit without proper pruning. Decision trees, however, can naturally handle non-linear relationships without requiring transformations, an area where logistic regression requires additional feature engineering or kernel methods. Similarly, logistic regression is more efficient during prediction than K-Nearest Neighbors (KNN), which necessitates computing distances for all training samples. While KNN is conceptually straightforward, its computational expense during prediction makes logistic regression preferable for large datasets. In contrast to Support Vector Machines (SVM), which excel at handling non-linear boundaries using kernel methods, logistic regression offers simpler interpretability and requires less computational power. However, SVMs may provide superior performance in complex, non-linear scenarios.

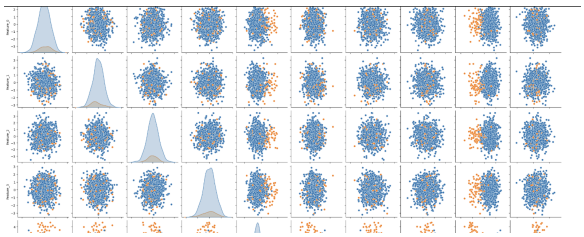
Logistic regression's strengths include its speed and efficiency when dealing with linearly separable data and the ease with which its coefficients can be interpreted to gauge feature importance. Nevertheless, the algorithm has notable weaknesses, including its limitation to linear decision boundaries without additional feature engineering or kernel extensions. Additionally, logistic regression is prone to overfitting in high-dimensional spaces unless regularization techniques are employed to mitigate this issue. These comparative insights highlight logistic regression's utility as a robust, interpretable, and efficient model, particularly suited for linearly separable datasets.

II. IMPLEMENTATION DETAILS

For this project, various design choices were made to ensure a robust and scalable approach to modeling and evaluation. Libraries such as NumPy and pandas were selected for their efficiency in handling data manipulation and exploration tasks. Visualization libraries like matplotlib and seaborn were employed to create detailed and customizable plots, enabling a clear understanding of data patterns and model performance. The synthetic dataset was generated using make_classification from

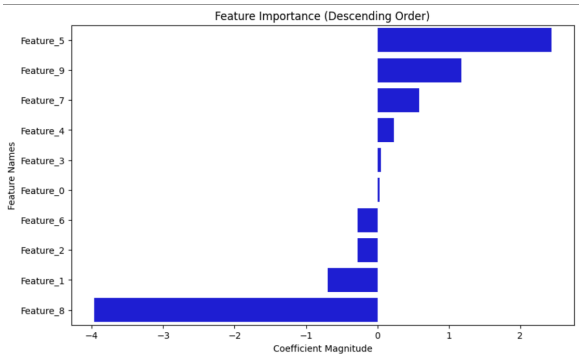
sklearn.datasets, allowing precise control over factors like class imbalance and noise. To simulate real-world imperfections, Gaussian noise was added through custom functions. Feature standardization was handled using StandardScaler, ensuring the logistic regression model converged optimally during training. For model training, GridSearchCV was employed to tune the regularization parameter (C), enabling an optimal balance between bias and variance. Performance evaluation incorporated a diverse set of metrics, including accuracy, confusion matrix, classification reports, and ROC-AUC, ensuring a thorough assessment of the model's predictive capabilities.

Synthetic Data:



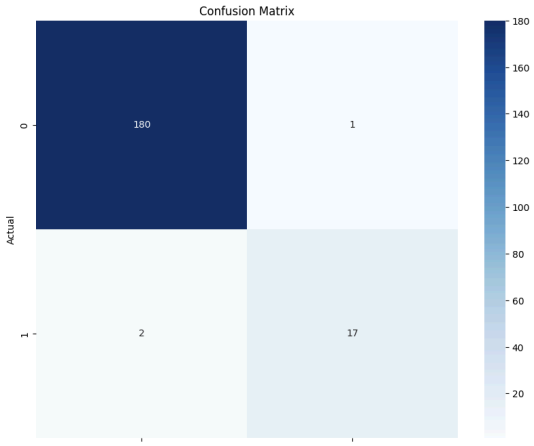
This project posed several challenges, each addressed through targeted strategies. The synthetic dataset included an intentional class imbalance to mimic real-world scenarios, which made achieving high recall for the minority class challenging. This was mitigated by analyzing feature importance and experimenting with oversampling techniques like SMOTE. Additionally, early attempts at hyperparameter tuning with limited grid ranges and folds yielded suboptimal results. Expanding the grid search range and increasing cross-validation folds resolved this issue, leading to a more robust model. Finally, interpreting feature importance coefficients presented a learning opportunity as both positive and negative values were observed. Detailed feature engineering and visualization helped clarify these interactions and their impact on the classification outcomes.

Feature Importance:



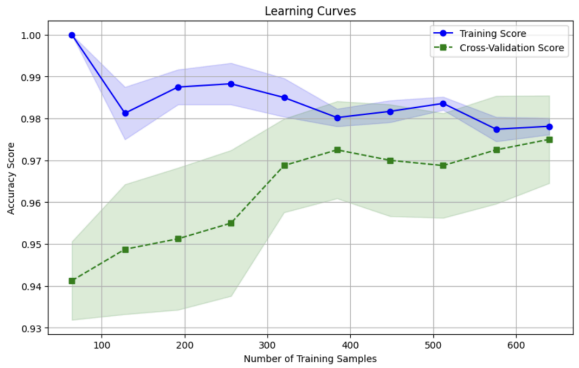
The architecture of the project was modular and designed for clarity and scalability. The logistic regression model and necessary components were initialized in the class constructor (__init__). Synthetic data generation allowed customization of key parameters, such as the level of class imbalance and noise, ensuring the dataset aligned with the project's goals. Data preprocessing involved standardizing features and splitting the data into training and testing sets using train_test_split. Hyperparameter tuning was performed using grid search to optimize the regularization parameter. Evaluation and visualization were integral components of the workflow.

Confusion Matrix:



Confusion matrices were plotted to assess classification performance, while feature importance plots provided insights into the relative contributions of each feature. Learning curves illustrated the relationship between dataset size and model performance, offering a visual representation of the trade-offs between bias and variance. Finally, the synthetic dataset was exported for reproducibility and further analysis.

Learning Curve:



Insights from this project underscore the importance of feature importance and learning curves in model evaluation. Certain features demonstrated significant impacts on class predictions, with feature 8 influencing class 0 and feature 5 supporting class 1. The learning curves highlighted how the model performed as the dataset size increased. The training score remained consistently high, while cross-validation (CV) scores steadily improved, eventually converging, which demonstrated reduced overfitting and enhanced generalization. The decreasing gap between training and CV scores reflected the model's ability to learn effectively from additional data, emphasizing the value of using larger training datasets.

The project's approach proved effective by combining empirical and theoretical evaluations. Its parameterized design allowed scalability by supporting datasets of varying sizes, making it possible to study model behavior across different conditions. This flexibility facilitated a deeper understanding of the logistic regression model's performance while offering practical insights into handling real-world challenges such as class imbalance. This comprehensive methodology demonstrates the strengths of logistic regression in providing interpretable results, identifying feature dynamics, and generalizing well across different data distributions.

III. BENCHING METHODOLOGY

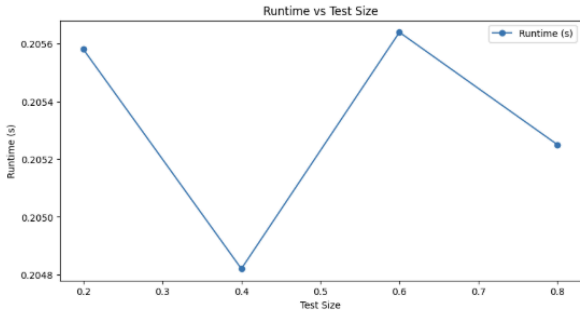
The benchmarking methodology for this project is designed to comprehensively evaluate the performance of a Logistic Regression model on synthetic data. The workload consists of typical input cases generated using the make_classification function from Scikit-learn, which ensures controlled variability by allowing customization of the number of samples, features, and class separations. This synthetic data is used to simulate classification tasks under different conditions, providing a reliable and reproducible benchmarking environment.

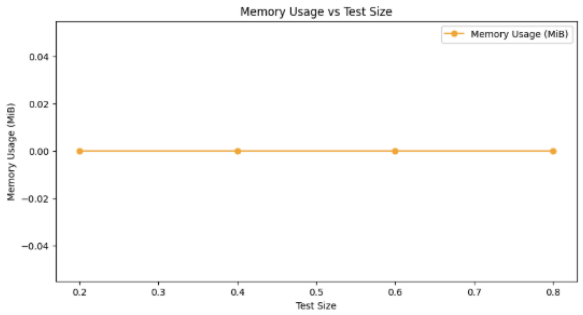
The dataset for benchmarking comprises a synthetic classification problem where the features and labels are generated programmatically. This approach eliminates external noise and variability that might arise from real-world data, ensuring that the evaluation focuses solely on the algorithm's performance. Each dataset is split into training and testing subsets using varying test sizes, ranging from 20% to 80%, enabling an analysis of the model's behavior under different proportions of training and testing data.

The benchmarking is conducted on a standard hardware and software setup, using

a machine with a typical modern CPU and adequate memory for running machine learning tasks. The software environment includes Google Colab as the notebook, Python 3 as the programming language, with necessary libraries such as Scikit-learn for model implementation, NumPy for data manipulation, and Memory Profiler for measuring memory usage. This combination of tools ensures that the evaluation is both robust and relevant to common machine learning workflows.

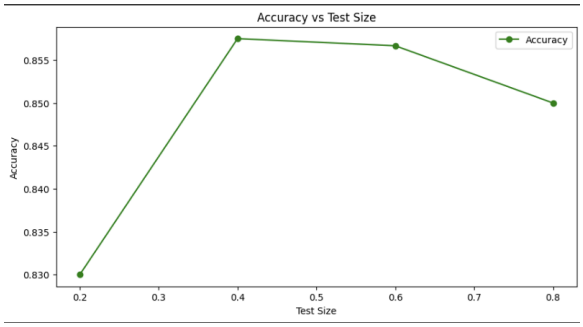
The results of the benchmarking are presented with clear empirical metrics, including runtime, memory usage, and model accuracy. These metrics are complemented by theoretical insights into time and space complexity. Runtime is measured by recording the duration of the fit and predict processes, showing the time required for training and making predictions. Memory usage is tracked by monitoring the memory before and after training, ensuring a precise measurement of the resources consumed. Accuracy, computed as the proportion of correctly classified samples, demonstrates the model's predictive power.





These empirical results are paired with theoretical complexity analysis: the time complexity of logistic regression is defined as $O(nsamples \times nfeatures \times iterations)$, while the space complexity is $O(nsamples \times nfeatures)$. This dual perspective bridges the gap between expected and observed performance.

The benchmarking results indicate that the Logistic Regression model performs consistently across different test sizes, with runtime stable at approximately 0.205 seconds and minimal or undetectable memory usage. Accuracy fluctuates slightly, peaking at 85.75% for a 40% test split, suggesting the model's robustness to varying data distributions.



These observations align well with the theoretical complexities, which remain constant irrespective of the test size.

Overall, this methodology provides a comprehensive evaluation of the model's performance by integrating empirical measurements and theoretical analysis. The flexible, parameterized design allows for scaling the benchmarking process to datasets of varying sizes, facilitating a deeper understanding of the model's scalability and resource utilization. The results demonstrate that the Logistic Regression model is efficient and well-suited for this synthetic dataset, with consistent speed, low memory usage, and reliable accuracy across different testing scenarios.