Mathematical Modelling of Engineering Problems
Vol. 11, No. 10, October, 2024, pp. 2823-2832
Journal homepage: http://iieta.org/journals/mmep

Stochastic Gradient Descents Optimizer and Its Variants: Performance of the Optimizers for Multinomial Logistic Models on Large Data Sets by Simulation

Sutarman[1*], Muhammad Alfan Irsyadi Hutagalung[1], Open Darnius[2], Muhammad Yandi Putra El Sya'ban[1]

[1] Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Sumatera Utara, Medan 20155, Indonesia
[2] Department of Statistics Diploma, Vocational Faculty, Universitas Sumatera Utara, Medan 20155, Indonesia

Corresponding Author Email: Sutarman@usu.ac.id

https://doi.org/10.18280/mmep.111025

Received: 1 August 2024
Revised: 2 October 2024
Accepted: 9 October 2024
Available online: 31 October 2024

Keywords:
optimizations, SGD, multinomial logistic regression, large datasets, stratified k-fold crossed validation, maximum likelihood

ABSTRACT

The exploration of Stochastic Gradient Descent (SGD) and its variants within the context of multinomial logistic models on large datasets represents a rich area of research. The existing literature underscores the strengths and weaknesses of various optimization techniques, paving the way for further investigations into their performance across diverse data environments. This article seeks to contribute to this ongoing discourse by systematically assessing the performance of these optimizers through simulations and real-world applications. By conducting simulations, the research will generate data-driven insights that can guide practitioners in selecting the most effective optimization methods for their specific applications. We explore how different SGD variants, including, Stochastic Gradient Descent (SGD), Stochastic Gradient Descent Momentum (SGDM), Adaptive Momentum (Adam), and Decaying Momentum Stochastic Gradient Descent Momentum (DemonSGDM), affect the convergence speed, accuracy, and ROC-AUC value as a generalization performance of the model on large simulated datasets. We compare their performance based on maximum likelihood methods for parameter estimations. The simulation framework allows us to control data characteristics and model parameters, allowing a systematic evaluation of the behavior of each SGD variant. Our findings can provide valuable insights into selecting the optimal SGD variant for modeling multinomial logistic models on large datasets through simulation. The simulation results show that both SGDM and DemonSGDM are more efficient and stable than SGD and Adam in terms of the number of epochs. The findings reveal that all optimizers can converge, but their effectiveness differs across datasets and complexities. SGDM and DemonSGDM perform well in simulations, while Adam has a slight advantage in challenging real-world scenarios.

1. INTRODUCTION

The concept of optimization is currently very developed and has been applied to various fields of science, engineering, medicine, computer science, and others. The concept of optimization has given birth to many methods and algorithms that help develop the theory and application of these fields. Some of them are least squares and maximum likelihood methods. In addition, gradient-based algorithms were also born, such as Newton. Especially in the field of statistics, the concept of optimization is very helpful in building models such as linear regression models, binary and multinomial logistic regression, time series, cox regression, and others. These models continue to grow, especially in the field of data science. These models can then be used for prediction and object classification.

One of the popular gradient-based optimization optimizers is SGD [1, 2]. This optimizer has an algorithm that iteratively optimizes a function that is differentiable or subdifferentiable. Thus, SGD will converge even if it is applied to functions that have derivatives or do not have derivatives. SGD works by replacing the actual gradient computed on the entire data set with an estimated gradient computed on a subset of the data set. This gives it an advantage over the use of Newton's method in solving the system of equations generated by maximum likelihood, which is the most widely used method for parameter estimation in statistical modeling. The method is very powerful for smaller data sets [3]. Of course, by using part of the data set to compute the gradient, it provides computational efficiency for large data sets [4, 5]. In its development, SGD [6] has several variants, including SGDM [7], DemonSGDM [8], Adaptive Moment Estimation (Adam) [9]. AdaGrad [10], Root Mean Square Propagation [11].

Evaluating SGD variants for multinomial logistic regression (MLR) on large datasets is a critical area of study due to several specific benefits and implications for practical applications. Some of them are handling high-dimensional data.

Furthermore, the impact of evaluating SGD on decision-making is giving information about accurate and efficient model [12]. Accurate and efficient models can lead to better decision-making in critical areas such as public health, finance, and policy-making. Evaluating SGD variants can enhance the predictive power of MLR models, ultimately benefiting stakeholders.

While there is extensive research on SGD and its variants, there is a lack of focused studies that compare the performance of these optimizers specifically for multinomial logistic regression. Most existing literature tends to concentrate on binary classification or does not adequately address the complexities of MLR [12, 13].

The rest of the paper is organized as follows. In Section 2, we briefly explore the related works of SGD and its most variants. In Section 3, we will explain about the optimization related SGD and its variants. Furthermore, the experiments and comparative results are discussed in Section 4. The last two sections are about the conclusions and future research.

2. RELATED WORKS

SGD and its variants have been extensively researched for optimizing multinomial logistic models, particularly in the context of large datasets. The foundational work [4], emphasizes the efficiency of SGD in large-scale optimization problems, highlighting its ability to converge faster than traditional batch gradient descent methods. This efficiency is particularly crucial when dealing with massive datasets, as SGD updates model parameters incrementally, allowing for quicker adjustments based on new data. Variants such as Mini-batch SGD, which processes small batches of data, have been shown to enhance convergence speed while maintaining computational efficiency.

In the context of multinomial logistic regression, the performance of these optimizers has been rigorously evaluated through both simulation studies and real-world applications. For instance, some researches [14, 15] conducted simulations that demonstrated the superiority of adaptive learning rate methods, such as Adam and RMSprop, over standard SGD in terms of convergence speed and accuracy when applied to synthetic datasets. Their findings suggest that adaptive methods can significantly reduce training time while achieving comparable or superior accuracy.

Conversely, real-world applications often present unique challenges that can affect optimizer performance [14]. They explored the efficacy of various optimizers on real-world datasets, revealing that while adaptive methods frequently outperform standard SGD in controlled environments, their performance can be inconsistent in practical scenarios due to factors such as feature sparsity and noise. This observation aligns with the work of, who introduced the Adam optimizer and demonstrated its robustness across diverse datasets. They found that while SGD remains a strong contender, the choice of optimizer can greatly influence model performance, particularly in high-dimensional and complex datasets.

Moreover, recent studies have highlighted the importance of tuning hyperparameters for different optimizers to achieve optimal performance [16]. The research emphasized the role of learning rate schedules and momentum in enhancing the performance of SGD and its variants in large-scale multinomial logistic regression tasks. Their research indicates that careful tuning can lead to significant improvements in both convergence speed and model accuracy.

Overall, the literature indicates that while SGD and its variants are powerful tools for optimizing multinomial logistic models, the specific context of the dataset—whether simulated or real—plays a crucial role in determining the most effective optimization strategy. Future research should continue to explore the interplay between optimizer choice, hyperparameter tuning, and dataset characteristics to further enhance the performance of multinomial logistic models across various applications.

Moreover, the integration of advanced techniques such as regularization and dropout has been explored to further enhance the performance of SGD-based optimizers. For instance, introduction dropout [15] as a regularization technique that helps prevent overfitting in neural networks, which can be particularly beneficial when training multinomial logistic models on large datasets. Their work indicates that combining dropout with SGD variants can lead to improved generalization performance.

Many research papers have reported the use of SGD and its development in various fields. One of them is related to the stability of SGD for homogeneous neural networks and linear classifiers [16].

In addition, investigation of the effects of gradient descent optimizers and dropout techniques also conducted [17] on the performance of deep learning LSTMs in rainfall runoff modeling. Meanwhile, SGD continues to develop in recent years, both theoretically and in applications. Some of them are [18, 19] with the momentum on stochastic recursive gradient descent algorithm and brain tumor detection using adaptive Stochastic Gradient Descent on shared memory parallel environment. In addition, we find that Stochastic Gradient Descent with low noise [20] developed.

The next application is on modified time adaptive self-organizing map for automated food recognition system [21]. Unfortunately, there are not many studies on the use of the optimizer and its variants on large datasets. However, local SGD which is more optimal when using small mini-batches than when using large mini-batches in SGD [22] introduced. Another study is about the ability of SGD and some of its variants in non-convex settings, which is very common in machine learning. In this case, it has shown that SGD can converge to sharper minima, potentially yielding better generalization on unseen data [23].

3. METHODS AND ALGORITHMS

3.1 Multinomial logistic regression

Multinomial logistic regression [24, 25] is a statistical model used to predict categorical outcomes with more than two levels. It generalizes binary logistic regression by modeling multiple classes simultaneously. The mathematical formulation of the multinomial logistic regression model can be expressed as follows:

Assume we have K categories for the dependent variable Y, where Y can take on values $1,2,\dots,K$. Let X be a vector of predictor variables (features). The probability of outcome $Y = k$ for $k = 1,2,\dots,K$ given the predictor variables is modeled using the SoftMax function:

$$P(Y = k|\boldsymbol{X}) = \frac{\exp\left(\beta_{k,0} + \beta_{k,1}X_1 + \beta_{k,2}X_2 + \cdots + \beta_{k,p}X_p\right)}{\sum_{J=1}^{K} exp\left(\beta_{j,0} + \beta_{j,1}X_1 + \beta_{j,2}X_2 + \cdots + \beta_{j,p}X_p\right)} \quad (1)$$

where,

$\beta_{k,0}$ is the intercept term for category k,

$\beta_{k,j}$ are the coefficients corresponding to predictor variables X_j for category k,

p is the number of predictor variables.

To ensure identifiability, one category (often the last one, K) is chosen as the reference category, and its parameters are set to zero:

$$\beta_{K,0} = 0,\ \beta_{K,j} = 0 \text{ for } j = 1,2,\dots,p \tag{2}$$

The log-odds of being in category k versus the reference category K is given by:

$$log\left(\frac{P(Y=k|\boldsymbol{X})}{P(Y=K|\boldsymbol{X})}\right) = \beta_{k,0} + \beta_{k,1}X_1 + \beta_{k,2}X_2 + \cdots + \beta_{k,p}X_p \tag{3}$$

3.2 Maximum likelihood estimation

The parameters β of the model are estimated using the method of maximum likelihood. The likelihood function for the multinomial logistic regression can be expressed as:

$$L(\beta) = \prod_{i=1}^{n} P(Y_i|\boldsymbol{X_i}) = \prod_{i=1}^{n}\prod_{k=1}^{K} P(Y_i = k|\boldsymbol{X_i})^{I(Y_i=k)} \tag{4}$$

where, n is the number of observations and $I(Y_i = k)$ is an indicator function that is equal to 1 if $Y_i = k$ and 0 otherwise. The log-likelihood function is therefore:

$$\log L(\beta) = \sum_{i=1}^{n}\sum_{k=1}^{K} I(Y_i = k)\log P(Y_i = k|\boldsymbol{X_i}) \tag{5}$$

3.3 Stratified K-fold Cross Validation (SKCV)

SKCV [26-28] is one of the variants based on k-fold cross validation [29], where folds are created to preserve the target class distribution. SKCV is particularly useful when dealing with data with an unbalanced class distribution. It also ensures that each fold has the same percentage of each class label.

K-fold cross-validation

The dataset is divided into k subsets (folds). The model is trained on k-1 folds and validated on the remaining fold. This process is repeated k times, with each fold serving as the validation set once.

Stratified k-fold cross-validation

Similar to k-fold, but it ensures that each fold has the same proportion of class labels as the entire dataset. This is particularly useful for imbalanced datasets.

SKCV is a powerful tool in the scikit-learn library that enhances the model evaluation process, aids in hyperparameter tuning, and helps in selecting the best model

3.4 Algorithm

The gradient descent algorithm [30-32] is an optimization algorithm commonly used to solve an equation or a system of equations without constraints. Its application is widely found in statistics, research operations, machine learning, deep learning, and others [33]. Furthermore, the algorithm works iteratively to find estimated parameter values as a solution to the steepest descent equation.

Suppose one can compute the gradient of a function $\nabla f(\theta)$ at any point x. In this case, the simplest method for optimizing $f(\theta)$ is the gradient method, in which one constructs an iteration sequence starting from some initial approximation θ_0, one constructs an iteration sequence.

$$\theta(t+1) = \theta(t) - \eta\nabla f(\theta_t) \tag{6}$$

where, the parameter $\eta > 0$ is the step size, or what is usually called the learning rate in machine learning.

Gradient descent uses all observation points to compute the gradient at each epoch. This makes it computationally inefficient for large data sets. To overcome this, several variants of gradient descent have been introduced. These variants of GD no longer use all observation points, but only some of them, always referred to as mini-batches. In general, the word stochastic is always associated with the names of these variants. One such variant is SGDM. This SGDM updates the gradient every epoch based on the momentum. In this case, momentum is an extension of the gradient descent optimization algorithm that allows the search to build up inertia in one direction in the search space and to overcome the oscillations of noisy gradients and coast over flat parts of the search space.

Another popular variant is Adam [34]. This optimizer is an algorithm for first-order gradient-based optimization of stochastic objective functions and relies on adaptive estimates of lower-order moments. The optimizer is easy to implement, computationally efficient, has a small memory footprint, is invariant to diagonal rescaling of the gradients, and is well suited for problems with large amounts of data and/or parameters. The method is also suitable for non-stationary targets and problems with very noisy and/or sparse gradients.

Recently, an increasingly popular variant of SGD is DemonSGD [35, 36]. DemonSGD uses linear learning rate decay models that reduce the influence of a gradient on current and future updates. By decaying the momentum parameter, the total contribution of a gradient to all future updates is decayed. In the following subsections, the algorithms used in the experiment are presented in pseudocode.

SGD

1. **Parameter**: learning rate η, timestep t
   a. $\theta_0 = 0$ or random
   b. $t = 0$
2. **While** θ_t not converged do
   $$\theta_{t+1} = \theta_t - \eta\nabla_{\theta_t} f(\theta_t)$$
3. **End while**

SGDM

1. **Parameter**:
   a. learning rate η
   b. momentum β
   c. timestep t
   d. $\theta_0 = 0$ or random
2. **While** θ_t not converged do
   $\beta_t = \beta_{t-1}v_{t-1} + (1-\beta_{t-1})\nabla_\theta f(\theta_{t-1})\theta_{t+1} = \theta_t - \eta\beta_t$
3. **End while**

Adam

1. **Parameters**:
   a. η: learning rate

b. $\beta_1, \beta_2 \in [0,1)$: Exponential decay rates for the moment estimates
c. $f(\theta)$: Stochastic objective function
d. θ_0: Initial parameter vector
e. $m_0 = 0$: Initial first moment vector
f. $v_0 = 0$: Initial second moment vector
g. $t = 0$: initial timestep
2. **While** θ_t not converged do
a. $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
b.$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$
c. $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$
d. $\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$
e. $\widehat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$
f. $\theta_t \leftarrow \theta_{t-1} - \eta \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\epsilon}$
3. **End while**

Demon SGDM
1. **Parameter**:
a. number of iterations T
b. learning rate η
c. β_{init} (Initial Momentum)
d. β_t momentum
e. $v_0 = \theta_0 = 0$ or random
2. **While** θ_t not converged do
a. $\beta_t = \frac{\beta_{init} \cdot (1-\frac{t}{T})}{(1-\beta_{init})+\beta_{init}(1-\frac{t}{T})}$
b. $\theta_{t+1} = \theta_t - \eta g_t + \beta_t v_t$
c. $v_{t+1} = \beta_t v_t - \eta g_t$
3. **End while**

3.5 Performance evaluation

Three metrics, loss function, accuracy, and ROC-AUC, are used to evaluate SGD variants in multinomial logistic regression. It is grounded in their relevance to the objectives of the study and their established significance in the literature. These metrics collectively provide a comprehensive evaluation of model performance, addressing both the optimization process and the practical implications of classification accuracy [37]. By utilizing these metrics, the study can effectively assess the strengths and weaknesses of different SGD variants [38, 39].

4. EXPERIMENTS

In general, this experiment was conducted based on two datasets, namely the simulation dataset and the real-world Red Wine Quality dataset. Subsequently, the learning rate and momentum values were estimated using SKCV by considering the Log-Loss test and test accuracy. The optimum learning rate and momentum values obtained were used by each optimizer to perform optimization on the multinomial logistic model. After convergence, the average run time, average number of epochs, and ROC-AUC scores were recorded. These values serve as the basis for assessing the performance of the optimizers on the multinomial logistic models.

4.1 Datasets

Table 1 provides a compilation of the datasets used to assess the performance of different optimizers in managing large data collections. We generated nine synthetic datasets, each varying in size and number of variables, to ensure a robust evaluation of the optimizers' performance under different conditions. All datasets contain three categories without outliers, and other nuance and were generated using the make_classification() function from scikit-learn (version 1.2..2) in Python 3.10.14.

For reproducibility, we specify the key parameters used in the make_classification() function:

- n_classes = 3 (fixed for all datasets)
- n_clusters_per_class = 1 (fixed for all datasets)
- n_samples: {5,000; 10,000; 20,000} (see Table 1)
- n_features: {30, 40, 50} (see Table 1)
- n_informative = 2
- n_redundant = 2
- n_repeated = 0 (no repeated features)
- class_sep = 1.0 (fixed for all datasets)
- flip_y = 0.01 (1% noise in class assignments)
- random_state = 42 (for reproducibility)

The function first creates clusters of points normally distributed (standard deviation = 1) around vertices of an n_informative-dimensional hypercube with sides of length 2*class_sep. It then assigns an equal number of clusters to each class.

Table 1. Datasets

Datasets	No. Observation	No. Variable
1	5,000	30
2	5,000	40
3	5,000	50
4	10,000	30
5	10,000	40
6	10,000	50
7	20,000	30
8	20,000	40
9	20,000	50

4.2 Real world datasets

For comparison, real world datasets red-wine-quality (source: https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009) will be used. This dataset is a subset of the wine quality datasets. This data consists of 1519 observations with 11 features and 6 classes. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g., there is no data about grape types, wine brand, wine selling price, etc.).

4.3 Experimental setting

In this experiment, the optimal learning rate and momentum are determined based on SKCV. In the case of the Adam algorithm,

The H hyperparameter grid space are Learning Rates, Momentum 1, and Momentum 2.

Learning Rates = {0.0001, 0.0003, 0.0005, 0.0008, 0.001, 0.003, 0.005, 0.008, 0.01, 0.03, 0.05, 0.08, 0.1, 0.3, 0.5, 0.8}

Momentum 1 = Momentum 2 = {0.7, 0.8, 0.9, 0.99, 0.999}

For all algorithms, the initial values of the parameters are set to zero. The following Table 2 shows the values of the learning rate, first momentum and second momentum for each data set used in the simulations.

Tabel 2. The values of the best learning rate, first momentum, and second momentum based on SKCV

No.	Optimizer	No. Observations	No. Variables	Learning Rate	Best First Momentum	Best Second Momentum	Test Log-Loss	Test Accuracy
1	SGD	5000	30	0.03	-	-	0.1111	0.9822
2	SGDM	5000	30	0.05	0.9	-	0.1036	0.9828
3	DemonSGDM	5000	30	0.01	0.8	-	0.1006	0.9824
4	Adam	5000	30	0.08	0.8	0.999	0.1198	0.9776
5	SGD	10000	30	0.001	-	-	0.3543	0.8726
6	SGDM	10000	30	0.005	0.8	-	0.3421	0.8747
7	DemonSGDM	10000	30	0.0008	0.9	-	0.3417	0.8753
8	Adam	10000	30	0.01	0.7	0.8	0.341	0.8758
9	SGD	20000	30	0.008	-	-	0.3855	0.86755
10	SGDM	20000	30	0.008	0.7	-	0.3858	0.8673
11	DemonSGDM	20000	30	0.0008	0.8	-	0.3839	0.86765
12	Adam	20000	30	0.01	0.9	0.8	0.3892	0.86655
13	SGD	5000	40	0.08	-	-	0.1589	0.9758
14	SGDM	5000	40	0.05	0.7	-	0.1568	0.9762
15	DemonSGDM	5000	40	0.01	0.7	-	0.1546	0.9776
16	Adam	5000	40	0.1	0.8	0.99	0.1808	0.9624
17	SGD	10000	40	0.005	-	-	0.3408	0.8741
18	SGDM	10000	40	0.003	0.7	-	0.3386	0.8739
19	DemonSGDM	10000	40	0.001	0.7	-	0.3388	0.8738
20	Adam	10000	40	0.01	0.7	0.9	0.342	0.8738
21	SGD	20000	40	0.005	-	-	0.3875	0.86725
22	SGDM	20000	40	0.005	0.8	-	0.3878	0.86715
23	DemonSGDM	20000	40	0.0005	0.9	-	0.3861	0.8684
24	Adam	20000	40	0.01	0.9	0.7	0.389	0.86755
25	SGD	5000	50	0.05	-	-	0.1494	0.9734
26	SGDM	5000	50	0.05	0.7	-	0.1562	0.9724
27	DemonSGDM	5000	50	0.005	0.8	-	0.1607	0.971
28	Adam	5000	50	0.08	0.8	0.999	0.1794	0.9592
29	SGD	10000	50	0.005	-	-	0.3453	0.8704
30	SGDM	10000	50	0.003	0.8	-	0.3415	0.8736
31	DemonSGDM	10000	50	0.0003	0.9	-	0.3433	0.8714
32	Adam	10000	50	0.008	0.7	0.99	0.3533	0.8701
33	SGD	20000	50	0.005	-	-	0.3899	0.8641
34	SGDM	20000	50	0.005	0.9	-	0.3875	0.86485
35	DemonSGDM	20000	50	0.0005	0.9	-	0.3851	0.86675
36	Adam	20000	50	0.01	0.9	0.8	0.3889	0.8636
37	SGD	Wine		0.003	-	-	0.9698	0.596
38	SGDM	Wine		0.003	0.7	-	0.969	0.5979
39	DemonSGDM	Wine		0.0005	0.99	-	0.9768	0.5985
40	Adam	Wine		0.0001	0.999	0.99	0.9714	0.6035

Each optimizer was evaluated based on its convergence speed, loss function stability, final accuracy, and ROC-AUC values with a tolerance value of 10-16 as the stopping rule. This is to determine the behavior of the optimizer after a solution is reached and over 100 epochs.

Furthermore, additional hyperparameters will be adjusted during the optimisation process, including the learning rate and momentum are determined using a combination of the python scikit-learn and numpy libraries with employed a grid search method coupled with k-fold cross-validation to determine the optimal learning rate and momentum. The dataset was initially partitioned into training and validation sets. Each hyperparameter combination within the predefined grid was evaluated using k-fold cross-validation on the training set. For each fold, the model was trained on the training subset and evaluated on the test subset using a specified scoring metric. The average score across all folds was computed for each hyperparameter combination. The combination yielding the best average score was selected as the optimal hyperparameters. Subsequently, the model was retrained on the entire training set using these optimal hyperparameters. Finally, the best model was evaluated on the held-out validation set to obtain an unbiased estimate of performance. These steps are shown as follows in pseudocode form:

Parameter:
a. dataset D
b. model M
c. hyperparameter_grid H
d. cross_validation_folds K
e. scoring_metric S

BEGIN
Split D into training set T and validation set V
FOR EACH combination C in H:
a. score_sum = 0
FOR i = 1 TO K:
a. Split T into K subsets
b. train_subset = T - T[i]
c. test_subset = T[i]
d. Train M on train_subset using hyperparameters C
e. Evaluate M on test_subset using S

f. Add evaluation score to score_sum

average_score = score_sum / K

IF average_score is better than best_score:

a. best_score = average_score

b. best_hyperparameters = C

Train M on entire T using best_hyperparameters

best_model = trained M

Evaluate best_model on V using S

final_score = evaluation result

RETURN

a. best_hyperparameters,

b. best_model,

c. best_score,

d. final_score

END

4.4 Model fitting

The model fitting process for evaluating SGD variants for MLR involves several key components, including the choice of software, convergence criteria, and regularization techniques. Open source Python; Buat sendiri; konvergence criteria: tolerance.

After getting the values of learning rate and momentum the multinomial logistic model is fitted using the following steps (pseudocode):

1. Initializing the weight matrix W randomly.
2. Depending on the chosen optimizer, we initialize additional variables (velocity for SGDM, first and second moments for Adam).
3. We then enter the main training loop, which iterates for a specified number of epochs.
4. In each epoch, we shuffle the data and process it in mini-batches.
5. For each mini-batch, we perform a forward pass, compute the gradient, and then update the weights based on the chosen optimizer:
    a. For SGD, we simply subtract the learning rate multiplied by the gradient.
    b. For SGDM, we update the momentum and then use it to update the weights.
    c. For DemonSGDM, we update the momentum and then use it to update the weights and decay momentum.
    d. For Adam, we update the first and second moments, compute their bias-corrected versions, and use them to update the weights.
6. After training, we return the final weight matrix.

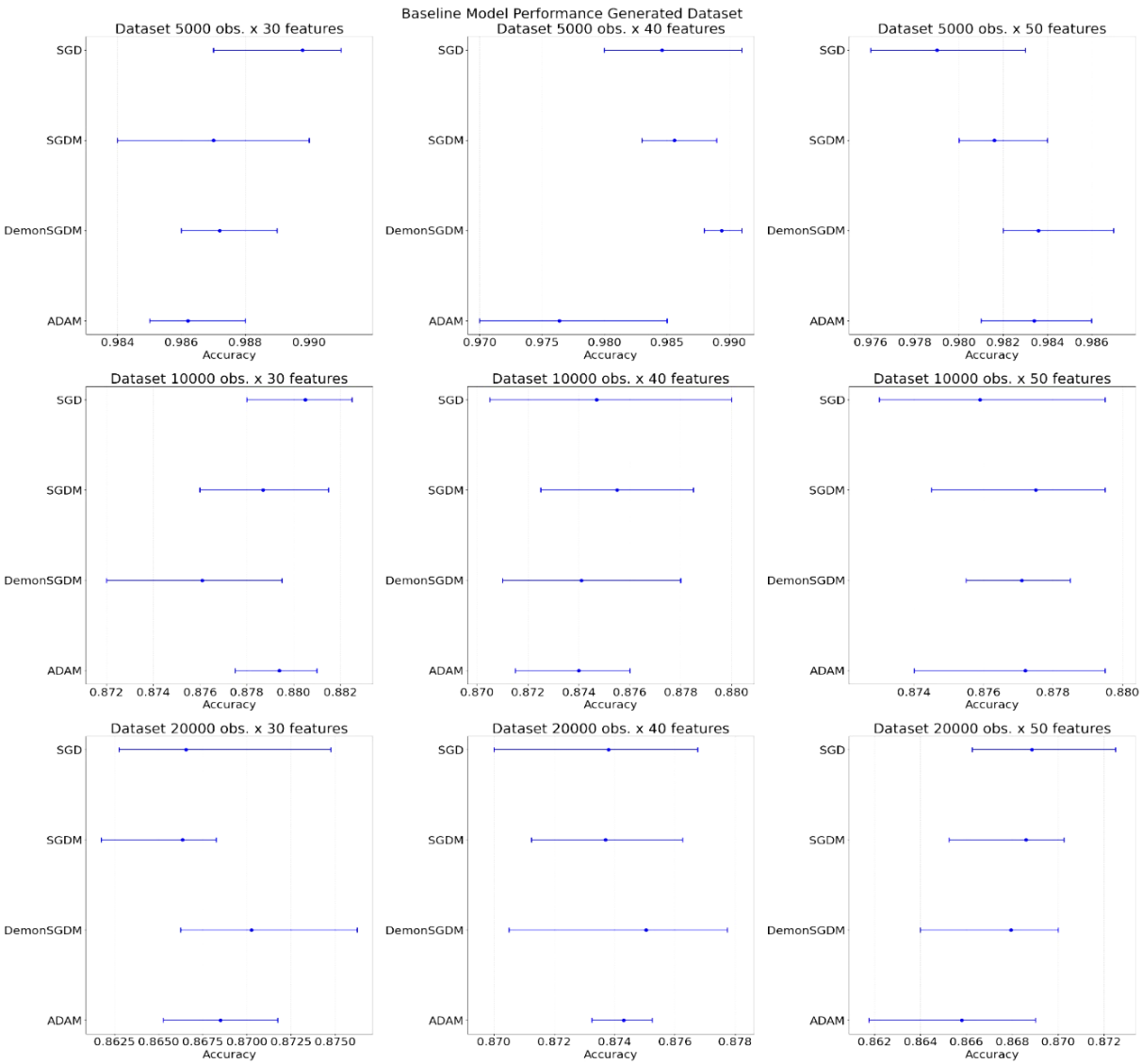

Figure 1. Model accuracy for simulation datasets

The multinomial logistic model is defined separately as it's used in the forward pass to compute the predicted probabilities.

This pseudocode provides a high-level overview of the process. In a real implementation, you would also need to handle data preprocessing, model evaluation, and potentially early stopping or learning rate scheduling.

5. RESULT

In general, all optimizers are capable of achieving convergence for both simulation datasets and the Red-Wine dataset. The following Table 2 shows the values of the learning rate, first momentum and second momentum for each dataset used in the simulations and real-world. Next, Table 3 presents the average run times of the teams (s), the average number of epochs, and the ROC-AUC scores. Additionally, Figures 1 and 2 display the accuracy interval values for each optimizer and dataset used. The ROC-AUC score and accuracy intervals will be used as the basis for assessing the performance of each optimizer.

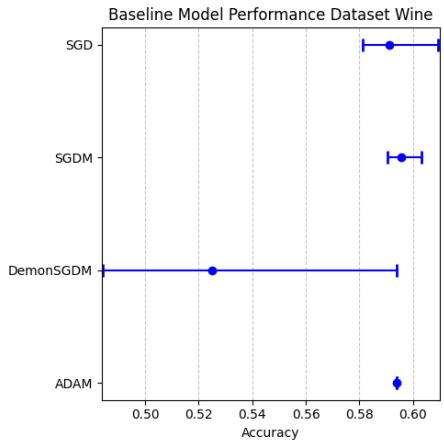

Figure 2. Model performance for red-wine quality

Table 3. Average run time, average number of epoch, ROC-AUC score values

Datasets	Optimizer	Average Run Time (s)	Average Number of Epochs	ROC-AUC Score
Simulation Dataset 5000 obs. × 30 features	SGD	0.758038	78.6	0.9974
	SGDM	0.509473	49.2	0.9971
	DemonSGDM	3.204334	300	0.9973
	Adam	3.696493	300	0.9972
Simulation Dataset 5000 obs. × 40 features	SGD	0.215203	21.4	0.9962
	SGDM	0.317307	28.8	0.9966
	DemonSGDM	3.295011	300	0.9969
	Adam	3.717411	300	0.996
Simulation Dataset 5000 obs. × 50 features	SGD	0.403949	39.8	0.9915
	SGDM	0.462271	42.2	0.9918
	DemonSGDM	2.711366	251.2	0.9919
	Adam	3.814062	300	0.9922
Simulation Dataset 10000 obs. × 30 features	SGD	1.910756	108	0.9605
	SGDM	0.49598	26.8	0.9601
	DemonSGDM	5.386132	300	0.9588
	Adam	0.917388	42.6	0.9596
Simulation Dataset 10000 obs. × 40 features	SGD	0.722471	42.4	0.959
	SGDM	1.193789	64.2	0.9597
	DemonSGDM	0.841327	46.4	0.9599
	Adam	1.61335	77.8	0.9599
Simulation Dataset 10000 obs. × 50 features	SGD	1.308712	36.4	0.9594
	SGDM	2.258024	53.6	0.9609
	DemonSGDM	12.10365	300	0.9603
	Adam	14.19419	300	0.9601
Simulation Dataset 20000 obs. × 30 features	SGD	0.764878	11.6	0.9628
	SGDM	0.82038	11.4	0.9627
	DemonSGDM	18.53604	243.4	0.9639
	Adam	7.485278	86.4	0.9638
Simulation Dataset 20000 obs. × 40 features	SGD	1.106545	17.2	0.9625
	SGDM	1.205817	17	0.9625
	DemonSGDM	23.68813	300	0.9624
	Adam	8.17894	90.2	0.9625
Simulation Dataset 20000 obs. × 50 features	SGD	1.26268	16.8	0.964
	SGDM	1.211844	16.2	0.9645
	DemonSGDM	21.9884	300	0.9639
	Adam	7.994103	90.6	0.9643
Red Wine Quality Dataset	SGD	2.698811	300	0.8325
	SGDM	2.900424	300	0.8322
	DemonSGDM	2.964975	300	0.8001
	Adam	3.498089	300	0.8344

For 5000 observations, the models generally performed best, with test accuracies often above 97%. Performance decreased for 10000 and 20000 observations, with accuracies dropping to around 87% and 86% respectively. There's no clear trend in performance as the number of variables increases from 30 to 50. For 5000 observations, SGDM and

DemonSGDM generally outperformed SGD and Adam. For 10000 and 20000 observations, the differences between optimizers were less pronounced. Adam often had slightly worse performance compared to the other optimizers, especially for 5000 observations.

All optimizers performed similarly on this real-world dataset. Test accuracies were much lower (around 60%) compared to the generated dataset, indicating this is a more challenging classification task. Adam slightly outperformed the other optimizers, with the highest test accuracy of 60.35%. The log-loss values were significantly higher for this dataset compared to the generated data. Furthermore, Figure 1 shows a series of plots comparing the performance of different optimization optimizers (SGD, SGDM, DemonSGDM, and Adam) across various datasets and conditions. The plots are organized in a 3×3 grid, with each row representing a different number of observations (5000, 10000, 20000) and each column representing a different number of features (30, 40, 50).

Each individual plot displays the accuracy of the four optimization algorithms for a specific combination of observations and features. The optimizers are listed vertically, and horizontal lines with dots represent the accuracy range and mean for each algorithm. The relative performance of the algorithms seems fairly consistent across different numbers of observations and features. In most cases, the order of performance from best to worst appears to be: DemonSGDM ≈ SGDM > SGD > Adam. The accuracy values generally fall between 0.86 and 0.98, indicating high performance across all conditions. There seems to be slightly more variation in results for the 5000 observation datasets compared to the 10000 and 20000 observation datasets. The number of features (30, 40, 50) doesn't seem to dramatically change the relative performance of the optimizers. Adam consistently shows the widest range of accuracy values and often performs slightly worse than the other optimizers. These two algorithms frequently have very similar performance, often overlapping in their accuracy ranges.

Figure 2 displays the performance comparison of four different optimization algorithms (SGD, SGDM, DemonSGDM, and ADAM) on a wine dataset. SGD shows a wide range of performance, with accuracy varying from about 0.58 to 0.61. SGDM has a narrower performance range, clustered around 0.59 to 0.60. DemonSGDM displays the widest range of performance, spanning from about 0.50 to 0.60. **A**DAM shows the most consistent performance with the smallest range, achieving an accuracy of about 0.60. ADAM seems to perform the best with the highest and most consistent accuracy has decent performance but with more variability than SGDM and ADAM. DemonSGDM shows the most unpredictable performance, with a very wide range of possible accuracies.

In general, the performance differences between these algorithms on this wine dataset are relatively small, with all achieving accuracies between 50% and 60%. This suggests that the classification task for this wine dataset might be challenging, as none of the algorithms achieve very high accuracy.

6. DISCUSSION

It is noteworthy that all optimizers used in the study can achieve convergence, both in simulation datasets and the more challenging real-world Red-Wine dataset. This suggests that despite differences in their underlying mechanics, each optimizer possesses the fundamental characteristics necessary to minimize loss functions effectively across diverse data types.

The evaluation of the optimizers is supported by two main metrics: the ROC-AUC score and accuracy intervals. Table 2 provides insights into the learning parameters for each simulation, while Table 3 and Figures 1 and 2 present more comprehensive performance metrics like average run times, epochs, and the ROC-AUC scores, which together create a robust framework for comparison.

Analyzing the performance based on the number of observations reveals intriguing insights. With 5000 observations, the models achieve high test accuracies often exceeding 97%. However, as the number of observations increases to 10,000 and 20,000, a significant drop in accuracy to around 87% and 86%, respectively, occurs. This decline may indicate that the models struggle to generalize as the dataset becomes larger and potentially more complex. The absence of a clear performance trend with the increase in feature variables (30 to 50) further complicates the narrative, indicating that merely increasing complexity does not necessarily enhance model performance.

Within the 5000 observation datasets, SGDM and DemonSGDM generally demonstrate superior performance over SGD and Adam. However, the differences among optimizers become less pronounced with larger dataset sizes of 10,000 and 20,000 observations. Interestingly, Adam often lagged behind the others, especially in smaller datasets. This suggests that while Adam is a popular choice in various contexts, it may require more data to express its full potential effectively.

The real-world wine dataset presents an entirely different challenge, with all optimizers achieving lower accuracies (around 60%). Adam stands out slightly with the highest accuracy of 60.35%, although this is modest overall. The increased log-loss values further indicate that this dataset is inherently more difficult, reinforcing the idea that complex real-world data may not always yield favorable results, regardless of the optimizer.

Figures 1 and 2 collectively illustrate the performance of each optimizer across varying conditions. The first figure compares the algorithms through a grid of accuracy plots, highlighting that DemonSGDM and SGDM consistently outperform others within the 5000 observations dataset. Figure 2 delves deeper into the wine dataset, illustrating that while ADAM offers consistency, it doesn't reach the levels of variability seen with SGDM and DemonSGDM. The accuracies for the wine dataset, hovering between 50% and 60%, imply that all algorithms are relatively ineffective under these classification conditions and may suggest a need for alternative modeling strategies or feature engineering.

7. CONCLUSION

In conclusion, the findings indicate that while all optimizers can achieve convergence, their effectiveness varies significantly across datasets and task complexities. SGDM and DemonSGDM stand out in simulations, while Adam garners a slight edge in challenging real-world applications, albeit with modest performance. The varying results highlight the need for careful optimizer selection based on dataset characteristics

and required performance thresholds particularly in predicting outcomes, where subtle distinctions in methodology can influence results substantially. Further exploration into hybrid approaches or alternative optimization techniques may yield enhanced performance, particularly for complex, real-world datasets like the wine classification case.

8. FUTURE RESEARCH

Future research could focus on developing and evaluating hybrid optimization algorithms that combine the strengths of existing optimizers like SGDM, DemonSGDM, and Adam. By integrating different optimization strategies, researchers can investigate whether hybrid approaches can achieve superior convergence rates and accuracy across various datasets, particularly in complex real-world scenarios.

Investigating adaptive optimization techniques that dynamically adjust learning rates and momentum parameters based on the characteristics of the dataset could be beneficial. This research could explore how adaptive mechanisms can improve performance in datasets with varying complexities and sizes, potentially leading to more robust models.

Given the observed performance variations with different feature sets, future studies should delve into advanced feature engineering and selection techniques. Research could focus on identifying which features contribute most significantly to model performance and how to preprocess data effectively to enhance optimizer efficacy.

Conducting extensive benchmarking of optimization algorithms across a wider range of datasets, including both synthetic and real-world data, would provide deeper insights into their performance. This research could help establish guidelines for selecting optimizers based on specific dataset characteristics, such as size, dimensionality, and inherent complexity.

Investigating alternative optimization techniques, such as evolutionary algorithms, swarm intelligence, or reinforcement learning-based optimizers, could yield new insights into their effectiveness compared to traditional methods. This research could assess whether these novel approaches can outperform established optimizers in specific contexts.

Future studies should consider a broader range of performance metrics beyond accuracy and ROC-AUC scores. Metrics such as precision, recall, F1-score, and computational efficiency could provide a more comprehensive evaluation of optimizer performance, particularly in imbalanced datasets or those with specific classification challenges.

Conducting case studies that apply various optimization algorithms to real-world problems, such as medical diagnosis, financial forecasting, or image classification, would provide practical insights into their effectiveness. These studies could highlight the nuances of optimizer performance in real-world applications and inform best practices for their deployment.

Implementing longitudinal studies that track the performance of different optimizers over time as new data becomes available could provide valuable insights into their adaptability and robustness. This research could help identify which optimizers maintain performance consistency in evolving datasets.

REFERENCES

[1] Li, L., Wang, Y.L. (2022). On uniform-in-time diffusion approximation for stochastic gradient descent. arXiv preprint arXiv.2207.04922. http://doi.org/10.48550/arXiv.2207.04922

[2] Toulis, P., Airoldi, E.M. (2015). Scalable estimation strategies based on stochastic approximations: Classical results and new insights. Statistics and Computing, 25: 781-795. https://doi.org/10.1007/s11222-015-9560-y

[3] Ghazali, K., Sulaiman, J., Dasril, Y., Gabda, D. (2020). Newton-2EGSOR method for unconstrained optimization problems with a block diagonal hessian. SN Computer Science, 1: 21. https://doi.org/10.1007/s42979-019-0021-0

[4] Bottou, L., Bousquet, O. (2007). The tradeoffs of large scale learning. In Advances in Neural Information Processing Systems (Vol. 20).

[5] Shen, L., Sun, Y., Yu, Z.Y., Ding, L., Tian, X.M., Tao, D.C. (2023). On efficient training of large-scale deep learning models: A literature review. arXiv preprint arXiv:2304.03589. https://doi.org/10.48550/arXiv.2304.03589

[6] Soydaner, D. (2020). A comparison of optimization algorithms for deep learning. International Journal of Pattern Recognition and Artificial Intelligence, 34(13): 2052013. https://doi.org/10.1142/S0218001420520138

[7] Bottou, L., Curtis, F.E., Nocedal, J. (2018). Optimization methods for large-scale machine learning. SIAM Review, 60(2): 223-311. https://doi.org/10.1137/16M1080173

[8] Sutskever, I., Martens, J., Dahl, G., Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In Proceedings of the 30th International Conference on International Conference on Machine Learning, Atlanta, USA, pp. 1139-1147.

[9] Buduma, N., Buduma, N., Papa, J. (2022). Fundamentals of Deep Learning. O'Reilly Media, Inc.

[10] Duchi, J., Hazan, E., Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(7): 2121-2159.

[11] Xu, D.P., Zhang, S.D., Zhang, H.S., Mandic, D.P. (2021). Convergence of the RMSProp deep learning method with penalty for nonconvex optimization. Neural Networks, 139: 17-23. https://doi.org/10.1016/j.neunet.2021.02.011

[12] Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, 27(8): 861-874. https://doi.org/10.1016/j.patrec.2005.10.010

[13] Mai, X., Liao, Z., Couillet, R. (2019). A large scale analysis of logistic regression: Asymptotic performance and new insights. In ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, pp. 3357-3361. https://doi.org/10.1109/ICASSP.2019.8683376

[14] Smith, L.N., Topin, N. (2018). Super-convergence: Very fast training of neural networks using large learning rates. arXiv preprint arXiv:1708.07120. https://doi.org/10.48550/arXiv.1708.07120

[15] Xie, Z.K., Wang, Z.X., Zhang, H.S., Sato, I., Sugiyama, M. (2024). Adaptive inertia: Disentangling the effects of adaptive learning rate and momentum. https://proceedings.mlr.press/v162/xie22d/xie22d.pdf, accessed on Oct. 13, 2024.

[16] Paquin, A.L., Chaib-Draa, B., Giguère, P. (2023). Stability analysis of Stochastic Gradient Descent for

homogeneous neural networks and linear classifiers. Neural Networks, 164: 382-394. https://doi.org/10.1016/j.neunet.2023.04.028
[17] Anh, D.T., Thanh, D.V., Le, H.M., Sy, B.T., Tanim, A.H., Pham, Q.B., Dang, T.D., Mai, S.T., Dang, N.M. (2023). Effect of gradient descent optimizers and dropout technique on deep learning LSTM performance in rainfall-runoff modeling. Water Resources Management, 37(2): 639-657. https://doi.org/10.1007/s11269-022-03393-w
[18] Qin, C., Li, B., Han, B. (2023). Fast brain tumor detection using adaptive Stochastic Gradient Descent on shared-memory parallel environment. Engineering Applications of Artificial Intelligence, 120: 105816. https://doi.org/10.1016/j.engappai.2022.105816
[19] Yang, Z. (2024). SARAH-M: A fast stochastic recursive gradient descent algorithm via momentum. Expert Systems with Applications, 238: 122295. https://doi.org/10.1016/j.eswa.2023.122295
[20] Wang, P.Y., Lei, Y.W., Ying, Y.M., Zhou, D.X. (2024). Differentially private Stochastic Gradient Descent with low-noise. Neurocomputing, 585: 127557. https://doi.org/10.1016/j.neucom.2024.127557
[21] Kotwal, J.G., Koparde, S., Jadhav, C., Bharati, R., Somkunwar, R. (2024). A modified time adaptive self-organizing map with Stochastic Gradient Descent optimizer for automated food recognition system. Journal of Stored Products Research, 107: 102314. https://doi.org/10.1016/j.jspr.2024.102314
[22] Lin, T., Stich, S.U., Patel, K.K., Jaggi, M. (2018). Don't use large mini-batches, use local SGD. arXiv preprint arXiv:1808.07217. https://doi.org/10.48550/arXiv.1808.07217
[23] Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836. https://doi.org/10.48550/arXiv.1609.04836
[24] Hosmer, D.W., Lemeshow, S. (2000). Applied Logistic Regression. John Wiley & Sons. Inc.: New York, NY, USA.
[25] Agresti, A. (2012). Categorical Data Analysis (Vol. 792). John Wiley & Sons.
[26] Allen, J., Liu, H., Iqbal, S., Zheng, D., Stansby, G. (2021). Deep learning-based photoplethysmography classification for peripheral arterial disease detection: A proof-of-concept study. Physiological Measurement, 42(5): 054002. https://doi.org/10.1088/1361-6579/abf9f3
[27] Hastie, T., Tibshirani, R., Friedman, J.H., Friedman, J.H. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (Vol. 2). New York: Springer.
[28] Theodoridis, S., Koutroumbas, K. (2009). Pattern Recognition (4th ed.). Academic Press.
[29] Szeghalmy, S., Fazekas, A. (2023). A comparative study of the use of stratified cross-validation and distribution-balanced stratified cross-validation in imbalanced learning. Sensors, 23(4): 2333. https://doi.org/10.3390/s23042333
[30] Lemaréchal, C. (2012). Cauchy and the gradient method. Documenta Mathematica Extra Volume ISMP, pp. 251-254.
[31] Maitanmi, O.S., Ogunyolu, O.A., Kuyoro, A.O. (2024). Evaluation of financial credit risk management models based on gradient descent and meta-heuristic algorithms. Ingénierie des Systèmes d'Information, 29(4): 1441-1452. https://doi.org/10.18280/isi.290417
[32] Farchi, C., Farchi, F., Touzi, B., Mousrij, A. (2023). A sustainable performance assessment system for road freight transport based on artificial neural networks. Ingénierie des Systèmes d'Information, 28(3): 647-653. https://doi.org/10.18280/isi.280313
[33] Pedregal, P. (2004). Introduction to Optimization (Vol. 46). New York: Springer.
[34] Kingma, D.P. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. https://doi.org/10.48550/arXiv.1412.6980
[35] John, C., Cameron, W., Zhao, L., Anastasios, K. (2019). Demon: Improved neural network training with momentum decay. In arXiv [cs.LG]. https://akyrillidis.github.io/pubs/Conferences/Demon.pdf.
[36] Chen, J., Kyrillidis, A. (2019). Decaying momentum helps neural network training. Opt-ml.org. https://www.opt-ml.org/papers/2019/paper_12.pdf, accessed on Oct. 16, 2024.
[37] Gupta, M., Yadav, D., Khan, S.S., Kumawat, A.K., Chourasia, A., Rane, P., Ujlayan, A. (2024). Modeling the detection and classification of tomato leaf diseases using a robust deep learning framework. Traitement du Signal, 41(4): 1667-1678. https://doi.org/10.18280/ts.410403
[38] Berbiche, N., El Alami, J. (2023). Enhancing anomaly-based Intrusion Detection Systems: A hybrid approach integrating feature selection and Bayesian Hyperparameter Optimization. Ingénierie des Systèmes d'Information, 28(5): 1177-1195. https://doi.org/10.18280/isi.280506
[39] Dasari, K., Mekala, S., Kaka, J.R. (2024). Evaluation of UDP-based DDoS attack detection by neural network classifier with convex optimization and activation functions. Ingénierie des Systèmes d'Information, 29(3): 1031-1042. https://doi.org/10.18280/isi.290321