Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2024) 32: 68 – 92

doi:10.55730/1300-0632.4056

Fast grid search: A grid search-inspired algorithm for optimizing hyperparameters of support vector regression

Mustafa AÇIKKAR*
Department of Software Engineering, Faculty of Computer and Informatics Engineering,
Adana Alparslan Türkeş Science and Technology University, Adana, Turkiye

Received: 12.04.2022 • **Accepted/Published Online:** 23.11.2023 • **Final Version:** 07.02.2024

Abstract: This study presents a fast hyperparameter optimization algorithm based on the benefits and shortcomings of the standard grid search (GS) algorithm for support vector regression (SVR). This presented GS-inspired algorithm, called fast grid search (FGS), was tested on benchmark datasets, and the impact of FGS on prediction accuracy was primarily compared with the GS algorithm on which it is based. To validate the efficacy of the proposed algorithm and conduct a comprehensive comparison, two additional hyperparameter optimization techniques, namely particle swarm optimization and Bayesian optimization, were also employed in the development of models on the given datasets. The evaluation of the models' predictive performance was conducted by assessing root mean square error, mean absolute error, and mean absolute percentage error. In addition to these metrics, the number of evaluated submodels and the time required for optimization were used as determinative performance measures of the presented models. Experimental results proved that the FGS-optimized SVR models yield precise performance, supporting the reliability, validity, and applicability of the FGS algorithm. As a result, the FGS algorithm can be offered as a faster alternative in optimizing the hyperparameters of SVR in terms of execution time.

Key words: Fast grid search, support vector regression, hyperparameter optimization, grid search

1. Introduction

In general, optimal results with machine learning methods are not attained unless their hyperparameters are appropriately fine-tuned. In order to build an accurate classification or regression model, it is essential to select the optimal hyperparameters of the machine learning method. Thus, the optimization of these hyperparameters is the key point to improve the method's training ability. Hyperparameter optimization can be very time-consuming if done manually or unsystematically, especially when the learning method has many hyperparameters.

Support vector machines (SVMs) represent a machine learning algorithm rooted in statistical learning theory and the principle of structural risk minimization, showcasing robust generalization capabilities. Furthermore, SVMs embody rigorous supervised learning models, and the incorporation of the kernel trick facilitates the development of nonlinear classifiers suitable for diverse input data types [1]. In essence, an SVM model nonlinearly transforms the initial data into a higher-dimensional feature space and subsequently performs linear regression within this feature space [2].

*Correspondence: macikkar@atu.edu.tr

SVMs, initially designed for classification problems, have undergone adaptations to effectively address regression problems [3]. A notable regression technique arising from this is support vector regression (SVR), introduced by Drucker et al. [4]. SVR has gained widespread adoption due to its ability to enhance prediction accuracy across various domains [5–9]. Extensive studies on SVR underscore the pivotal role of hyperparameter selection in augmenting prediction accuracy [10–12]. Consequently, similar to other machine learning methods, the precise selection of hyperparameters significantly influences the performance of the developed SVR models.

The performance of SVR is significantly influenced by key hyperparameters, including the capacity value (C), the epsilon value (ϵ), the type of kernel function, and, when applicable, the associated parameters of the kernel function. As a result, optimal selection of the kernel function and precise determination of its parameter values are crucial. Kernel functions commonly utilized in SVR can be broadly categorized into four main types: linear, polynomial, radial basis function (RBF), and sigmoid kernels. In previous studies [13–15], the RBF kernel was found to be mostly superior to other kernels with satisfactory generalization capabilities. Therefore, RBF, which requires optimizing the γ hyperparameter, is used as the kernel function in this study. To create a precise SVR model using the RBF kernel function, it must be determined which values of (C, ϵ, γ) are most suitable for the provided regression problems.

The predictive accuracy of SVR is significantly impacted by the selection of hyperparameters. Unfortunately, there is no established mathematical model or formula available to precisely determine the optimal values for these hyperparameters. In order to achieve this objective, it is advisable to employ a suitable hyperparameter optimization algorithm to identify the desired values of the hyperparameters. This approach aims to maximize the prediction accuracy of the SVR-based model.

In the academic literature, similar to various machine learning techniques, SVM hyperparameters are commonly optimized through the utilization of swarm intelligence, probabilistic, physics-based, and evolutionary algorithms. The primary objective of employing these algorithms is to enhance the predictive efficiency of the model. Among the frequently utilized algorithms for optimizing SVM, particle swarm optimization (PSO) [16–22], Bayesian optimization (BO) [13, 14, 23, 24], genetic algorithms [25–28], ant colony optimization [11, 33, 34], the sine-cosine algorithm [30–32], grey wolf optimization [35–37], the gravitational search algorithm [38], and the black hole algorithm [39] can be listed. Moreover, a wide variety of general-purpose algorithms have been put forth with the goal of determining the hyperparameters of SVR. These algorithms encompass novel approaches, hybrid methodologies, and enhancements of existing techniques [40–49].

In addition to the aforementioned approaches, the trial-and-error-based grid search (GS) algorithm is extensively employed for hyperparameter tuning of SVR [50–54]. GS is an iterative process of recalculating a given model using various combinations of hyperparameters. The objective is to identify the hyperparameter set that results in the highest accuracy rate in model performance. This technique guarantees that GS has a high level of accurate prediction, but it also leads to protracted computation time, even for small datasets, because of the potentially large number of possibilities that must be investigated.

Several GS-based algorithms have been proposed recently, improving the traditional GS, to optimize the hyperparameters of SVMs. Sun et al. [15] presented an improved GS (IGS) algorithm to optimize C and γ by dynamically adjusting the search range and step iteratively. Beltrami et al. [55] proposed a new hyperparameter optimizer for SVM-based classification problems called grid-quadtree (GQ), which integrates the quadtree technique with GS. They used ten benchmark datasets to assess the GQ performance.

In addition to the studies referenced above, Bao and Liu [56] introduced another fast grid search approach (FGS_{ts}) for optimizing the hyperparameters C and γ of SVR with the RBF kernel in the context of time-series forecasting. The basic idea of the suggested technique involves traversing the hyperparameter grid map in the direction that exhibits the greatest reduction in error. In order to achieve this, the errors derived from the prediction models utilizing eight points (hyperparameter sets) adjacent to the current point are calculated. This process continues until it either reaches the boundary or returns to a previously tested point. After this exploration process, the algorithm selects the best C and γ based on prediction measures obtained during the training process. While there is similarity in nomenclature between the present study and the previous study described here, it is important to note that the underlying working principles diverge significantly. Comprehensive details regarding this algorithm can be obtained from the referenced publication [56].

The motivation for this study, which aims to optimize the hyperparameters of SVR, consists of several important factors. Understanding the underlying motivations will help contextualize the study's significance and potential impact on SVR research.

- Optimizing model performance: Hyperparameter tuning significantly affects the performance of machine learning models. Our motivation is to improve the prediction accuracy and generalization of SVR models by developing a more efficient and effective hyperparameter optimization technique.
- Overcoming GS limitations: GS can suffer from a large number of evaluations, especially in high-dimensional spaces. Our motivation is to address the limitations of GS and create a more efficient and effective algorithm in finding optimal hyperparameters.
- Reducing computational burden: GS, while straightforward, can become computationally expensive, especially with large datasets or complex models. Therefore, the goal is to design an alternative algorithm that reduces the computational burden and allows for faster and more scalable hyperparameter optimization.
- One of the motivations of this study is to develop an optimization algorithm that is not limited to SVR but can be applied to other machine learning models with discretized search space and regression or classification problems.
- Another motivation of this paper is to provide researchers with a valuable tool to improve the performance of SVR models by developing a new hyperparameter optimization algorithm.

The main objective of this study is to present a fast hyperparameter optimization technique for SVR inspired by GS, which we call fast grid search (FGS). The importance of this study could be summarized as follows: Up to the present, no GS-based algorithm that scans the hyperparameter search space so fast has been proposed for the hyperparameter optimization of SVR in the literature. With the introduction of this proposed algorithm, it can be asserted that it offers an intelligent method to bypass unnecessary hyperparameter evaluations through GS. This, in turn, enhances the efficiency of SVR while maintaining model performance, rendering it suitable for handling large-sized datasets. To illustrate the effectiveness of the presented method on SVR optimization, the FGS algorithm was first compared with its constituent algorithm, GS, and also two other most used and robust optimization techniques, namely PSO and BO, for the provided benchmark datasets.

While this study introduces a novel hyperparameter optimization algorithm for SVR, it is important to acknowledge certain potential limitations to ensure the reliability and transparency of the research. These limitations include:

- Limited scope of datasets: This study focuses on a set of benchmark datasets, which might not fully represent the diversity and complexity of real-world data. The performance of the FGS algorithm could vary on different types of datasets not considered in this study.
- The FGS algorithm was designed and tested specifically for SVR hyperparameter optimization. It might not be directly applicable or optimal for hyperparameter optimization with other machine learning algorithms.
- Hyperparameter search space: The effectiveness of any hyperparameter optimization algorithm is highly dependent on the chosen search space. This study uses a predefined search space for SVR hyperparameters, which might not be the most suitable for all datasets or SVR models.
- Limited hyperparameter combinations: Due to its time and computational advantages, FGS does not exhaustively investigate all possible hyperparameter combinations and can potentially miss optimal hyperparameter sets.
- Lack of comparison with other algorithms: While this study compares the FGS algorithm with GS, PSO, and BO, it may not have included other potentially competitive hyperparameter optimization algorithms that could offer further insights into the FGS algorithm's performance.
- This study has not explored the FGS algorithm's scalability concerning large datasets. Some massive datasets may impact the FGS algorithm's performance.

This paper is structured into four distinct sections and the subsequent discussion is organized as follows. Section 2 elaborates on the datasets employed and outlines the methodology, including specifics regarding the proposed technique for optimizing hyperparameters. Section 3 focuses on presenting the FGS-optimized SVR models, comparing them with models optimized using the techniques outlined in this paper, and evaluating their performances on the respective datasets. The experimental results obtained from these comparisons are thoroughly analyzed. Finally, in Section 4, a summary of the key points and findings of the paper is provided.

2. Methodology

The present study employed four benchmark datasets comprising five outputs in order to construct prediction models based on SVR. To achieve better prediction accuracy, it is necessary to tune the hyperparameters of each model. For the purpose of finding the optimal values of C, ϵ, and γ, a modified fast hyperparameter optimization technique, FGS, inspired by GS, is proposed in this study. In this study, three different optimization techniques, namely GS, PSO, and BO, were employed to identify the most effective hyperparameters of SVR for the purpose of comparison. The effect of the presented hyperparameter optimization techniques on prediction accuracy was compared separately for each dataset. All experiments conducted in this work were run on a PC with a 4.20 GHz CPU and 16 GB of memory using the MATLAB R2022a[1] environment.

2.1. Overview of datasets

In this study, we utilize four distinct medium-scaled datasets sourced from the UCI Machine Learning Repository. These datasets were employed to assess and benchmark the performance of the presented hyperparameter

[1]MathWorks Inc. (2022). Statistics and Machine Learning Toolbox [online]. Website https://www.mathworks.com/help/stats/ [accessed 05 January 2024]

optimization algorithm for SVR. These datasets are the Concrete Slump Test (CST) [57], Energy Efficiency (EE) [58], Concrete Compressive Strength (CCS) [59], and Airfoil Self-Noise (ASN) [60]. As the EE dataset comprises two distinct outputs, namely cooling load and heating load, two separate datasets were derived: EEC and EEH, corresponding to the EE dataset with cooling load and heating load, respectively. The specifics including the dataset name, number of instances, number of features, target variable name, and descriptive statistics of the target variable for each dataset are summarized in Table 1.

Table 1. Specifics of the datasets.

Name	#Instances	#Features	Target			
			Name	Min.	Max.	Avg. ± Std.
CST	103	8	Compressive Strength (MPa)	17.19	58.53	36.0 ± 7.8
EEC	768	9	Cooling Load (kW)	10.90	48.03	25.6 ±9.5
EEH			Heating Load (kW)	6.01	43.01	22.3 ±10.1
CCS	1030	9	Compressive Strength (MPa)	2.33	82.60	35.8 ± 16.7
ASN	1503	6	Scaled Sound Pressure Level (dB)	103.38	140.99	124.8 ± 6.9

2.2. Dataset partitioning and k-fold cross-validation

In machine learning, the dataset is typically divided at random into training and testing subsets. Cross-validation is commonly employed to enhance the generalizability of predictive models and attain more robust results by minimizing interference from randomness. k-fold cross-validation arbitrarily divides the initial dataset into k equally sized subsets. One of the subsets is allocated as the testing data to validate the model's performance, whereas the remaining $k - 1$ subsets serve as the training data. This technique is executed iteratively for a total of k iterations, whereby each iteration involves the utilization of one of the k subsets as the testing data. One advantageous aspect of this technique is that it ensures the utilization of all instances for both training and testing purposes, with each instance being exclusively employed for testing only once. The final assessment of the performance measures for each model is established by calculating the average of the results obtained from each fold.

2.3. Performance metrics

In the hyperparameter optimization phase of the SVR model, the performance of each submodel was calculated using mean square error (MSE), and the overall prediction performance of each SVR model was calculated using root mean square error ($RMSE$), mean absolute error (MAE), and mean absolute percentage error ($MAPE$). The respective formulas for these metrics are provided in Eqs. (1) through (4).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2 \tag{1}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2} \tag{2}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \widehat{Y}_i| \tag{3}$$

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\frac{|Y_i - \widehat{Y}_i|}{Y_i} \tag{4}$$

Here, n is the number of instances, and Y_i and $\widehat{Y}_i$ represent the actual and predicted values of the ith instance in the testing set, respectively.

Furthermore, throughout the optimization phase, the performance of the models was assessed by considering the number of evaluations of the objective function ($Cost_{eval}$) and the optimization time (t_{opt}), in addition to the aforementioned metrics. *RMSE*, *MAE*, *MAPE*, t_{opt}, and $Cost_{eval}$ are established as measures that quantify SVR-based model performance.

2.4. SVR-based prediction models

The steps involved in implementing the SVR model are outlined in Figure 1. At the outset, the dataset was subjected to cross-validation, leading to the generation of fresh training and testing sets. Subsequently, each set underwent preprocessing via standardization to achieve zero mean and unit variance. This preprocessing step is advantageous as it scales variables with high values, consequently reducing the computational power needed to construct the SVR prediction model. By performing the hyperparameter optimization technique on the training set, the best values for C, ϵ, and γ were determined during the optimization phase. To meet this prerequisite, we employed a 5-fold cross-validation technique, partitioning the training set into new training and testing subsets. This approach aimed to mitigate the influence of randomness and bolster the generalization capability of the optimization process. To derive the ultimate *MSE* (MSE_{cost}), we computed the *MSE* values for each hyperparameter set (C, ϵ, and γ) within their respective subfolds and subsequently averaged these values. The hyperparameter set yielding the lowest MSE_{cost} value was then selected to build the prediction model, which was applied to predict target variable values in the testing set. Finally, the prediction model's performance was evaluated by computing the testing set's metrics, namely *RMSE*, *MAE*, and *MAPE*. The t_{opt} and $Cost_{eval}$ of each model were also recorded. This depicted process, as shown in Figure 1, was independently conducted for each dataset and optimization technique.

2.5. Hyperparameter optimization algorithms

As previously established, SVR requires a precise choice of hyperparameters to achieve high generalization performance. Thus, an efficient search technique is required to assess the optimal values for these hyperparameters. In the existing literature, numerous techniques have been designed to tune the hyperparameters of the SVR, with the overall goal of increasing precision and improving the reliability of predictions [40–49]. In this study, three different optimization techniques were used apart from the proposed FGS algorithm to determine the optimal hyperparameters of SVR. These techniques are GS, PSO, and BO, and this section gives adequate information about them.

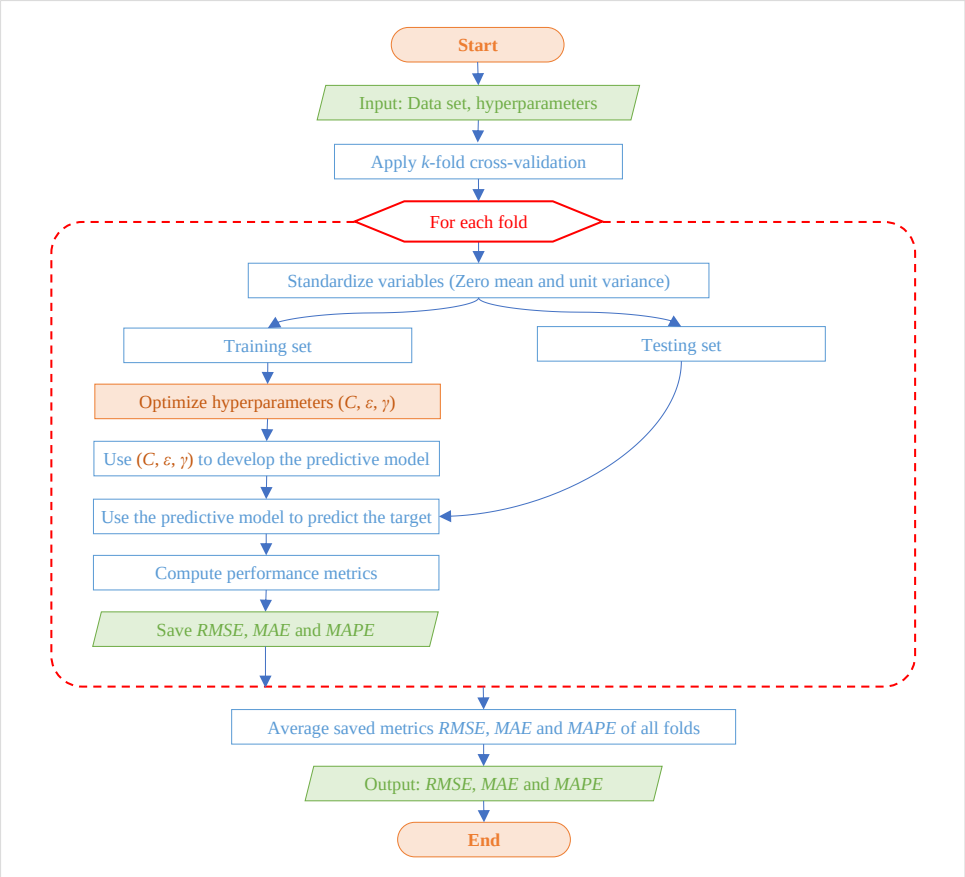

Figure 1. Flowchart of the SVR-based model.

2.5.1. Grid search (GS)

GS is a widely employed technique to optimize hyperparameters of SVM in regression and classification problems. The idea behind GS is simple and it relies on a trial-and-error process. The range of values for each hyperparameter is discretized by partitioning it into a series of logarithmic-scaled equidistant intervals ($GridSize_{C,\epsilon,\gamma}$), forming a structured search space. GS then evaluates every possible combination of hyperparameter values inside this space. This involves a cross-product of all intervals, and so the computational expense is exponential in the number of hyperparameters. While GS is known for its high accuracy, it is characterized by a computational time requirement deemed excessive. Consequently, it is often described as a brute-force or exhaustive search technique.

The performance of each set of hyperparameters can be assessed through the evaluation of the error obtained (MSE_{cost}). At the completion of the search procedure in GS, the set with the smallest error is chosen as containing the optimal hyperparameters.

Table 2 presents the hyperparameters' ranges, *GridSize*, and scale types. The range of each hyperparameter was also employed in the FGS, PSO, and BO algorithms.

Table 2. Overview of SVR hyperparameters.

Hyperparameter	Range	*GridSize*	Scale Type
Capacity (C)	$[2^{-3}, 2^{14}]$	18	Logarithmic
Epsilon (ϵ)	$[2^{-10}, 2^{5}]$	16	
Gamma (γ)	$[2^{-5}, 2^{8}]$	14	

2.5.2. Particle swarm optimization (PSO)

Kennedy and Eberhart [61] proposed the PSO algorithm, a population-based stochastic optimization technique based on the theory of swarm intelligence, in 1995. This algorithm is inspired by the social interactions of birds or fish flocks in nature, and each individual representing a bird or fish in the PSO algorithm is called a *particle* and has its own velocity V_i and position X_i [49]. Particles (or particle swarms) seek the optimal solution through collaboration and mutual learning in a d-dimensional space. Here, the position of each particle embodies a potential candidate solution to the problem and d signifies the number of particles ($SwarmSize$).

When implementing PSO, the particles are initially assigned to random locations throughout the solving space. Subsequently, an iterative search is conducted to identify the best solution that corresponds to the optimal hyperparameters of SVR and minimizes the validation error. Each particle searches for the optimal solution by constantly adjusting its position and remembers its own best solution (B_i) and the best solution of the swarm (B_g), as the most optimal solution ever found, experienced by the entire particle swarm. During this iterative process, the velocity V_i and position X_i of each particle are updated with the following formulas:

$$V_{id}^{(t+1)} = wV_{id}^{t} + c_1 r_1 (B_{id}^{t} - X_{id}^{t}) + c_2 r_2 (B_{gd}^{t} - X_{id}^{t}) \tag{5}$$

$$X_{id}^{(t+1)} = X_{id}^{t} + V_{id}^{(t+1)} \tag{6}$$

In Eq. (5), w represents the inertia weight, c_1 and c_2 denote acceleration coefficients, and r_1 and r_2 are random numbers within the range of $(0, 1)$. For this study, c_1 and c_2 were adopted from the extensively acknowledged research conducted by Clerc and Kennedy [62]. The strategy of linear decreasing inertia weight [63] was utilized to balance the exploration and exploitation process of PSO. During each iteration, the value of w was decremented accordingly with the following equation:

$$w_i = w_{max} - [(w_{max} - w_{min}) \times \frac{i}{MaxIter}] \tag{7}$$

Here, w_{max} is 1.1, w_{min} is 0.3, and $MaxIter$ represents the number of iterations.

Upon completing this iterative procedure, the solution exhibiting the lowest validation error is regarded as the most optimal, signifying the suitable values of SVR hyperparameters for the present study.

2.5.3. Bayesian optimization (BO)

Recent years have witnessed a surge in the adoption of the Bayesian optimization (BO) algorithm, recognized for its prowess as a powerful, versatile, and efficient tool. It excels in furnishing optimal and practical solutions, particularly for the optimization of computationally complex, noisy, and resource-intensive objective (cost) functions [14]. BO has therefore become a powerful approach for optimizing machine learning methods due to

its ability to efficiently navigate complex and high-dimensional parameter spaces [24, 64]. Consistent with this, the goal of BO in this work is to select the optimal hyperparameters for an SVR model, and some of the main reasons why we consider BO are as follows:

- BO uses probabilistic models to balance exploration and exploitation. This is crucial for efficient optimization in machine learning, where we need to find the best-performing configurations within a limited number of evaluations.
- BO is sample-efficient, requiring fewer evaluations of the objective function compared to traditional optimization methods. This is particularly beneficial when evaluating the machine learning method is time-consuming.
- BO is adept at finding global optima rather than getting stuck in local optima, which is a common challenge in optimizing complex machine learning models with many hyperparameters.
- BO allows for an adaptive approach where the choice of the next set of hyperparameters to be evaluated is influenced by the outcomes of previous evaluations. This sequential aspect is highly advantageous for dynamically adjusting the search based on the observed performance.
- BO employs Gaussian processes or other probabilistic models to model the objective function. This allows for uncertainty estimation, aiding in making informed decisions about where to sample next.
- In many real-world scenarios, evaluating a machine learning method can be noisy or expensive. BO accommodates this by modeling the noise and adjusting the exploration accordingly.
- Machine learning models often have hyperparameters that significantly impact their performance. BO can efficiently tune these hyperparameters to enhance the model's performance.

Assuming the solution space of potential hyperparameters is denoted as X and the cost function utilized to minimize validation error is f, the BO algorithm can be succinctly represented with the equation below:

$$x_{opt} = arg\, min_{x \in X} f(x) \tag{8}$$

Here, x_{opt} denotes the hyperparameter set yielding the smallest value of the cost function while x denotes any value within the solution space X.

BO utilizes Bayes' theorem to discern the minima or maxima of the cost function. This approach employs a surrogate probability model, guiding the selection of the next evaluation point in the solution space through an iterative procedure. During each iteration, the algorithm refines its surrogate model based on existing prior knowledge, a phase often referred to as the "learning phase." Subsequently, BO employs an acquisition function to evaluate the objective function in what is termed the "optimization phase." This function scrutinizes the surrogate model, aiding in the prediction of the next candidate best solution. The algorithm then updates the surrogate model with the newly obtained result to prepare for the subsequent iteration. This iterative process, alternating between learning and optimization phases, continues until sufficient data are generated, allowing the surrogate model to evolve and converge towards the global optimum [14]. Through this approach, BO demonstrates a capacity to effectively produce the optimal solution while minimizing the number of evaluations. The essence of this theorem can be mathematically expressed as follows:

$$P(H \mid e) = \frac{P(e \mid H)\, P(H)}{p(e)} \tag{9}$$

Here, $P(e \mid H)$ represents the likelihood, $P(H)$ indicates the prior probability, $P(e)$ stands for the marginal probability (evidence), and $P(H \mid e)$ signifies the posterior probability.

2.5.4. Fast grid search (FGS)

Appropriately determining the hyperparameters of SVR significantly improves the prediction accuracy. The difficulty in choosing the correct values of hyperparameters of the SVR model may require testing many possible values of hyperparameters, and the GS algorithm does so. Therefore, the GS-optimized SVR model (GS-SVR) can be very time-consuming in optimizing hyperparameters even when applied to small-sized datasets. To avoid this drawback of the GS algorithm, this study suggests a fast optimal hyperparameter search algorithm inspired by GS (FGS). This new and improved algorithm mainly aims to reduce the computational cost of GS-SVR, but, at the same time, it matches its accuracy by using the same discretized search space.

To reveal our motivation in this study, preliminary work was carried out to show the effect of the changes in the values of the hyperparameters on the MSE_{cost} values of the submodels during the optimization phase. For this purpose, we developed a 10-fold applied GS-SVR model on the CCS dataset by utilizing the hyperparameters given in Table 2 and following the flowchart of the SVR model illustrated in Figure 1. For each fold, we recorded the MSE_{cost} values of the developed submodels using each hyperparameter combination set (4032 sets) in the optimization phase. After optimization (for the first fold), the set of hyperparameters giving the minimum MSE_{cost} was determined as the optimal hyperparameter set (i.e. $C = 2048$, $\epsilon = 4$, and $\gamma = 4$). To show the effect of the hyperparameters on MSE_{cost}, the values of the hyperparameter sets and the corresponding MSE_{cost} values were used to form Figure 2. Each subfigure in Figure 2 was constructed using the values (in logarithmic scale) of the selected hyperparameters and their corresponding MSE_{cost} values. Here, in each subfigure, while the value of the selected hyperparameter changes over its specified range, fixed values are assigned to the other two hyperparameters. In order to show the effect of the change in the values of the selected hyperparameter more clearly, the optimal values of the other two hyperparameters are chosen as fixed values of these hyperparameters. As can be easily seen in each subfigure, the best value of the selected hyperparameter is considered to be the value that converges MSE_{cost} to the global minimum. As a result, this proposed algorithm follows an iterative process during the optimization phase of the model for each hyperparameter. It focuses on quickly finding the minimum value of MSE_{cost} using the discretized values of the selected hyperparameter in the given range while fixing the values of the other two hyperparameters.

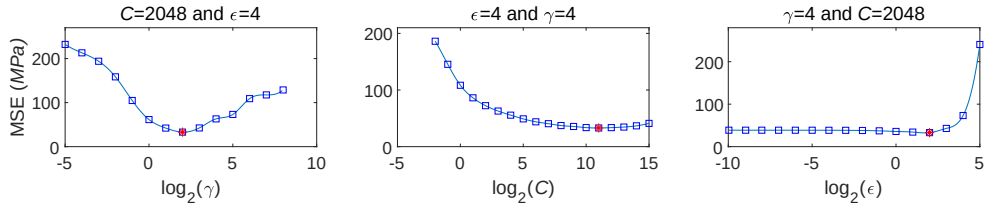

Figure 2. Effects of changes in the values of hyperparameters on MSE_{cost} values during the optimization phase for the CCS dataset.

In this iterative process, initially, two submodels with different hyperparameter sets are created by fixing the values of two hyperparameters and choosing the leftmost (lower bound) and rightmost (upper bound) values of the given range for the selected hyperparameter, and the MSE_{cost} values obtained from these two submodels are compared. As a result of this comparison, the selected hyperparameter value that gives the lower MSE_{cost} is preserved and the predetermined range is narrowed by shifting the value of the selected hyperparameter on the other side by the number of adaptively varied steps (*StepSize*). Thus, another new value is obtained for the selected hyperparameter, leading to a new submodel. This comparison process is repeated continuously until the MSE_{cost} values of the submodels obtained using the final left and right hyperparameter values encounter each other, and the value of the selected hyperparameter that gives the smallest MSE_{cost} value is accepted as the best value for that hyperparameter.

This process is the core part of the proposed algorithm, which sequentially and alternately switches to the next hyperparameter after a certain number of evaluations (*SwitchSize*) and repeats one or more times for each hyperparameter. In this way, the proposed algorithm finds global minima, or the optimized values of the hyperparameters, without falling into local minima, with this fast and straightforward search process, the details of which are given below.

According to the author's previous experiences in optimizing the hyperparameters of SVR-based models, all hyperparameters exhibit robust interactions with each other. Therefore, the idea of the proposed FGS algorithm relies on the partial and sequential optimization of each hyperparameter and a step-by-step search to find the minimum value of MSE_{cost} during the optimization phase of the model.

Before explaining the detailed implementation steps of the proposed algorithm for SVR optimization, the following observations must be made in order to understand the general framework of the FGS algorithm:

- The same discretized search space used in the GS algorithm was also used in the FGS algorithm. However, GS examines all possible points (hyperparameter sets) within the discretized search space, whereas FGS examines only a limited number of points, which will be detailed below.
- The FGS algorithm has three parameters, which are the maximum number of evaluations (*MaxEval*), the number of evaluations to switch to the next hyperparameter (*SwitchSize*), and the maximum number of steps ($StepSize_{max}$).
- In order to facilitate the understanding and implementation of the FGS algorithm, the values of each hyperparameter were indexed from 1 to *GridSize*.
- The lower bound index (*LBi*) and the upper bound index (*UBi*) of each hyperparameter correspond to 1 and *GridSize*, respectively. The values of each hyperparameter corresponding to *LBi* and *UBi* are defined as the lower bound value (*LBv*) and the upper bound value (*UBv*), respectively.
- FGS operates on the indexes of each hyperparameter. If the hyperparameter's value is required for calculation, the corresponding index value is used. For this reason, the process is explained through indexes so that the values are not confused and the algorithm can be better understood.
- Each hyperparameter is optimized partially and in order. The default order of the hyperparameters to be optimized is chosen as γ, C, and ϵ. The current hyperparameter to be optimized is abbreviated as *HPc*.

As stated before, in the FGS algorithm, the most appropriate values of the hyperparameters are found by using a search process on the discretized search space based on partial and sequential optimization of each hyperparameter. This search process is repeated by replacing the *HPc* with the next one in the predefined order

so that each hyperparameter is ensured to be the *HPc* at least once or more. Each partial search process, which is the core of the proposed FGS algorithm, is referred to as a partial optimization process (POP).

The flowchart of the FGS algorithm is illustrated in detail in Figure 3. To perform the overall optimization process (OP), first of all, the training data and attributes of the hyperparameters are utilized as input, as in the GS algorithm. Secondly, the hyperparameter γ is marked as the first *HPc*. To construct the new hyperparameter sets (*HPS*s), the *UBv* of hyperparameter C and the *LBv* of hyperparameter ϵ are considered the best, by default. Furthermore, the evaluation count (*evalCount*) is set to zero.

The FGS algorithm then proceeds with the POP applied to the *HPc*, and the POP is implemented by the following steps. At the beginning of the POP, it is initially checked whether the OP is completed. If all hyperparameters are marked as optimized, or *evalCount* has reached *MaxEval*, the proposed FGS algorithm automatically terminates the OP. In both cases, the final best value of each hyperparameter is considered to be the most suitable value of that hyperparameter. However, it should be noted that none of these criteria could be met at the beginning of the OP.

If the OP is not completed, initially, the iteration count for *SwitchSize* (*switchCount*) is reset. After that, it is checked whether the *HPc* is marked as optimized. If the *HPc* is considered as optimized before, the new *HPc* is obtained by switching to the next hyperparameter (*HPn*) according to the predefined order of hyperparameters described above. This criterion could also not be met at the beginning of the OP. Otherwise, two *HPS*s are constructed by using the *LBv* and *UBv* of the *HPc* and the best values of the other two hyperparameters. These *HPS*s are named HPS_{LBv} and HPS_{UBv}.

If these *HPS*s have not been used before, the *MSE*s, namely MSE_{LBv} and MSE_{UBv}, are obtained by using HPS_{LBv} and HPS_{UBv}, respectively. After each model is created, *switchCount* and *evalCount* are increased by 1 separately. Otherwise, the corresponding *MSE* value of the *HPS* is utilized for comparison purposes as will be explained as follows. Next, if the difference between *UBi* and *LBi* is equal to 1, the *HPc* is marked as optimized. This will be used to determine if the POP ends in the following steps. MSE_{LBv} and MSE_{UBv} are then compared with each other to decide which one of the *LBv* and *UBv* is the best. If MSE_{LBv} is greater than MSE_{UBv}, the *UBv* is considered the best. Otherwise, the *LBv* is considered the best.

To end the POP, at least one of the following three conditions must be satisfied. First, the *HPc* must be marked as optimized. Second, the *switchCount* value must be greater than or equal to *SwitchSize*. Third, the direction of the *HPc* must be changed (i.e. while the *LBi* increases, the direction is considered changed if *UBi* starts to decrease in the next evaluation, or vice versa). If one of these conditions is met, the current POP ends and the new one starts after assigning the *HPn* as the new *HPc*. If none of the above conditions are met, the new *stepSize* that belongs to the *HPc* needs to be calculated dynamically. The new *LBi* or *UBi* must then be calculated in order to construct a new *HPS*. If MSE_{LBv} is greater than the MSE_{UBv}, the new *LBi* is obtained by increasing the value of the current *LBi* with the new calculated *stepSize*, and otherwise, the new *UBi* is obtained by decreasing the value of the current *LBi* with the new calculated *stepSize*. Finally, the new *HPS* is constructed using the new corresponding *LBv* or *UBv* of the *HPc*. Considering the difference between *UBi* and *LBi*, the *stepSize* is dynamically recalculated at each POP. In the end, when the difference between *UBi* and *LBi* and the final value of the *stepSize* is 1, then the *HPc* is marked as optimized.

This process is repeated at least once or more, but only for the new *HPS*. Eventually, the *LBi* and *UBi* come close to each other. Therefore, the partial optimization for the *HPc* is considered complete. As stated before, this subprocess is maintained and repeated until one of the criteria for the termination of the OP is met.

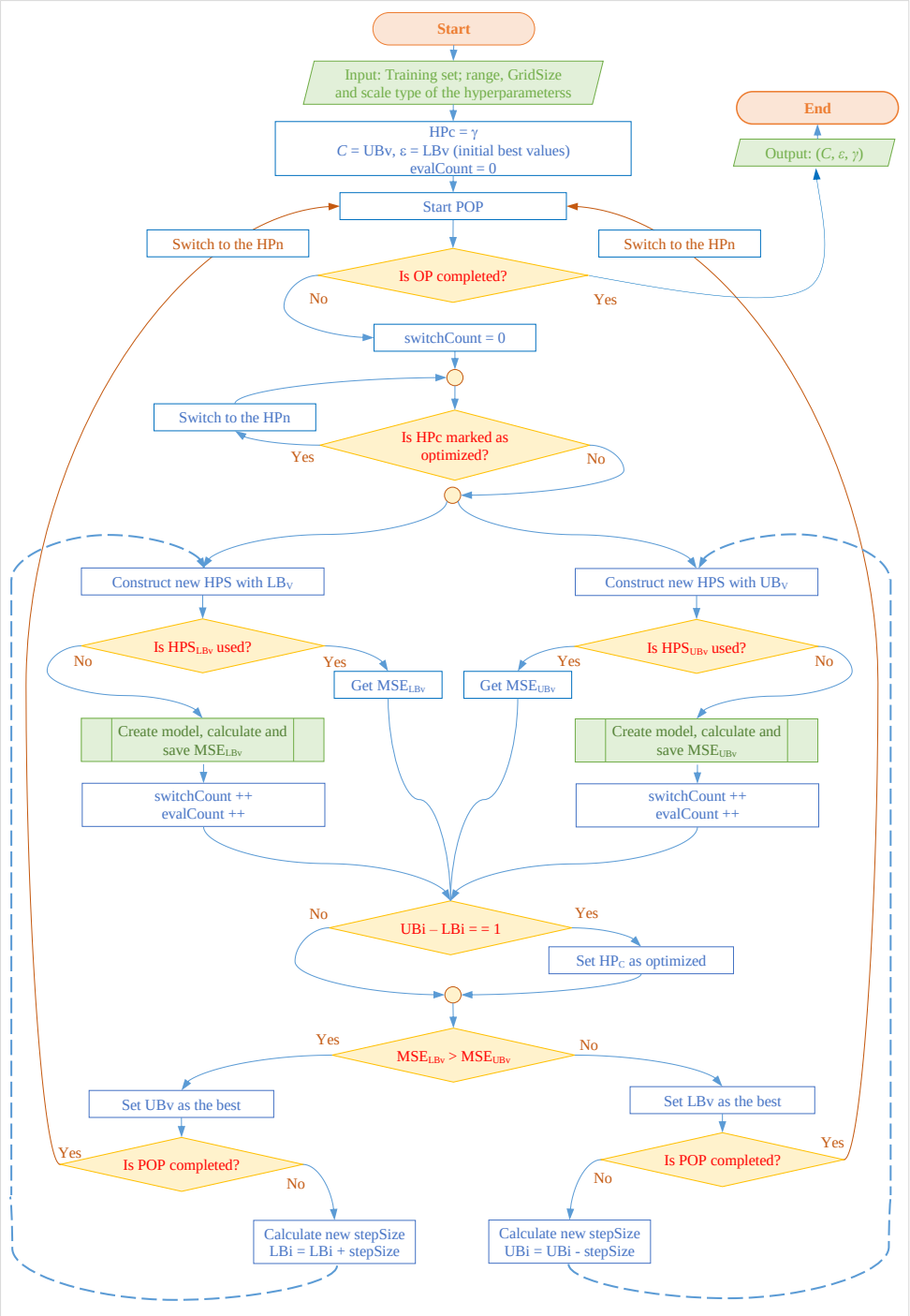

Figure 3. Flowchart of the FGS algorithm.

3. Experimental analysis

The present study introduces an improved hyperparameter optimization algorithm inspired by GS for SVR. To assess the efficacy of the proposed FGS algorithm on prediction accuracy, its impact on model performance was

first benchmarked against that of the GS algorithm. Additionally, for comparative analysis, the performances of FGS-optimized SVR models were also compared with those of models employing two other widely used and robust optimization techniques, namely BO and PSO, across various datasets.

To ensure the generalizability of the findings, the datasets underwent evaluation via 10-fold cross-validation. Moreover, to enhance robustness and dependability, the entire process illustrated in Figure 1 was executed independently for each dataset and optimization technique a total of 20 times. This repetition aimed to mitigate the impact of randomness and bolster the reliability of the reported results. Additionally, for each dataset and run, indices of all folds were randomly generated before developing the prediction models. This methodology guaranteed that all models were trained and tested on precisely the same data segments, thus enhancing the reliability of the obtained results. The ultimate performance value for each model was determined by calculating the mean of the results from each fold and then aggregating them across each run.

Before undertaking a comparative analysis of the models presented with various optimizers, it is essential to address several key considerations. First, lower values of *RMSE*, *MAE*, and *MAPE* indicate superior model performance. Second, lower values of t_{opt} and $Cost_{eval}$ signify faster model optimization.

This section commences with a preliminary investigation aimed at identifying the optimal user-defined parameters for the FGS, PSO, and BO algorithms. Subsequently, SVR-based models are constructed using the most favorable parameter configurations obtained from the optimization algorithms. Finally, this section presents and discusses the experimental findings.

3.1. Parameter settings

To analyze the effects of the user-defined parameters of the optimization algorithms, a preliminary study was conducted on the datasets. For this purpose, different values of the parameters of the algorithms were tested by running each model 5 times. In order to decide which parameter values to choose according to the results to be obtained from datasets, it is necessary to consider the smallest possible values of the MSE_{cost} and $Cost_{eval}$. Small values of MSE_{cost} indicate better optimization, and small values of $Cost_{eval}$ indicate rapid optimization. It should also be noted that small values of $Cost_{eval}$ may mean that the optimization is not fully completed, while large values may cause the optimization process to be time-consuming without a significant reduction in MSE_{cost}. As a result, if the MSE_{cost} values obtained from parameter sets are comparable, the set with the smaller $Cost_{eval}$ value will be considered to reduce the computational cost.

3.1.1. Parameters of FGS

The FGS algorithm has three parameters, *MaxEval*, *SwitchSize*, and $StepSize_{max}$, for which optimal values need to be determined. The value of *MaxEval* should be set to a large value because if the value is set lower than required, the number of evaluations will be reached quickly and the optimization process will end undesirably before the optimization is complete. However, this parameter can be used if the optimization process is desired to be terminated at a certain number of evaluations, although it is not suitable. Therefore, the default value of *MaxEval* was accepted to be 100 without conducting any experimental analysis. The values of *SwitchSize* and $StepSize_{max}$ have to be adjusted manually. Small values of these parameters result in an increase in the number of submodels. However, large *SwitchSize* values may prevent the hyperparameters from being optimized simultaneously and accurately. On the other hand, large $StepSize_{max}$ values may reduce the model accuracy due to the possibility of missing the optimal values of the hyperparameters.

To search for the optimal values of *SwitchSize* and $StepSize_{max}$, the range for each of these parameters was determined as [3, 6] and [2, 5], respectively. The MSE_{cost} and $Cost_{eval}$ values obtained from combinations of these parameters are shown in Table 3 for all datasets. For each of the EEC and ASN datasets, the same MSE_{cost} value was obtained for different values of $StepSize_{max}$. Therefore, small $StepSize_{max}$ values are preferred to minimize the possibility of unintentionally missing the optimum value. According to the results, since it is not possible to determine the optimal values of *SwitchSize* and $StepSize_{max}$ as being the same for all datasets, they were determined separately for each dataset. Therefore, the parameter pairs were decided to be (3, 3), (6, 3), (4, 3), (5, 3), and (6, 3) for the CST, EEC, EEH, CCS, and ASN datasets, respectively.

Table 3. Optimization performances of the FGS algorithm with different parameter pairs on datasets.

SwitchSize	$StepSize_{max}$	MSE_{cost}					$Cost_{eval}$ (*Min-Max*)
		CST (*MPa*)	EEC (*kW*)	EEH (*kW*)	CCS (*MPa*)	ASN (*dB*)	
3	2	0.40	1.57	0.27	32.33	5.15	[29, 35]*
4	2	0.41	1.64	0.27	32.93	5.12	
5	2	0.39	1.42	0.29	32.88	5.14	
6	2	0.38	1.42	0.29	32.82	5.07	
3	**3**	**0.36**✓	1.55	0.26	33.31	5.12	
4	**3**	0.39	1.56	**0.25**✓	33.20	5.12	
5	**3**	0.44	1.42	0.28	**32.13**✓	5.10	
6	**3**	0.37	**1.40**✓	0.28	32.48	**5.04**✓	
3	4	0.41	1.60	0.26	32.51	5.16	
4	4	0.38	1.60	0.26	32.70	5.10	
5	4	0.39	1.42	0.28	32.31	5.10	
6	4	0.43	**1.40**	0.28	32.44	**5.04**	
3	5	0.39	1.66	0.26	32.50	5.16	
4	5	0.37	1.66	0.26	32.76	5.10	
5	5	0.37	1.42	0.28	32.30	5.10	
6	5	0.43	**1.40**	0.28	32.28	**5.04**	

*Since the value of $Cost_{eval}$ adaptively changes according to the *GridSize* of each hyperparameter, *SwitchSize* and $StepSize_{max}$, a range is given for $Cost_{eval}$ that encompasses the ranges obtained from all optimization phases.

3.1.2. Parameters of PSO

The PSO algorithm has some user-defined parameters, such as w, c_1, c_2, r_1, r_2, w_{max}, and w_{min} given in Eqs. (5) and (7). The values of these parameters and how these values are determined are detailed in Section 2.5.2. In addition, there are two more variables, *MaxIter* and *SwarmSize*, that need to be determined in order for the PSO algorithm to reach the optimum solution. The values of *MaxIter* and *SwarmSize* were formed in pairs as (20, 10), (30, 10), (40, 10), (30, 15), and (40, 15) and examined to search for the optimal values of these parameters. Table 4 shows the obtained MSE_{cost} and $Cost_{eval}$ values from the pairs for all datasets. The smallest MSE_{cost} value belonging to each dataset is indicated in bold. The results showed that selecting the pair (40, 10) produced the smallest MSE_{cost} values for the CST, EEC, and EEH datasets. For the CCS and ASN datasets, primarily, it was found that there was no significant difference between the smallest MSE_{cost} values and those obtained from the pair (40, 10). Secondly, it was considered that it would be appropriate to

choose large values for *MaxIter* and small values for *SwarmSize* in order to increase the performances of the models. As a result of the determination made from the results, the optimal values of *MaxIter* and *SwarmSize* for all datasets could be considered as 40 and 10, respectively, to make a fair comparison between all models.

Table 4. Optimization performances of the PSO algorithm with different parameter pairs on datasets.

MaxIter	*SwarmSize*	MSE_{cost}					$Cost_{eval}$
		CST (MPa)	EEC (kW)	EEH (kW)	CCS (MPa)	ASN (dB)	
20	10	0.58	1.63	0.214	31.32	5.43	200
30	10	0.47	1.63	0.212	**30.33**✓	5.09	300
40	**10**	**0.41**✓	**1.51**✓	**0.209**✓	30.42	5.04	400
30	15	0.50	1.65	0.211	30.70	**5.03**✓	450
40	15	0.46	1.53	**0.209**	30.58	6.05	600

3.1.3. Parameters of BO

For the BO algorithm, no internal parameters need to be set, but the number of evaluations ($Cost_{eval}$) must be determined. The values of $Cost_{eval}$ were determined as (40, 50, 60, 70, 80, 90, and 100) to search for the best-fitting $Cost_{eval}$. The MSE_{cost} and $Cost_{eval}$ values are given in Table 5 for each dataset. The lowest MSE_{cost} value of each dataset is indicated in bold. For all datasets, it was found that there was no significant difference between the MSE_{cost} values obtained from $Cost_{eval}$ values 80, 90, and 100. According to the results, to reduce the computational cost and to make a straightforward comparison between the models, the optimal value of $Cost_{eval}$ for all datasets was set to be 90.

Table 5. Optimization performances of the BO algorithm with different numbers of evaluations on datasets.

#Evaluations	MSE_{cost}					$Cost_{eval}$
	CST (MPa)	EEC (kW)	EEH (kW)	CCS (MPa)	ASN (dB)	
40	0.41	2.46	1.26	32.75	4.37	40
50	0.45	2.15	0.95	32.19	4.18	50
60	0.39	1.86	0.43	32.20	4.11	60
70	0.40	1.97	0.33	32.62	4.05	70
80	0.38	1.60	0.27	31.88	3.99	80
90	**0.37**✓	1.61	**0.26**✓	31.82	4.01	90
100	0.38	**1.59**✓	0.26	**31.79**✓	**3.98**✓	100

3.2. Experimental results

This section presents the performances of the FGS algorithm and three other optimization algorithms on SVR models. For this purpose, SVR-based models optimized with FGS, GS, PSO, and BO were developed for each dataset. Table 6 presents the impact of the proposed hyperparameter optimization algorithms on the prediction performance of the SVR models, measured regarding *RMSE*, *MAE*, and *MAPE*. Additionally, the table includes the average values along with the corresponding standard deviations obtained from 20 runs for each performance metric.

Table 6. Performance metrics of the models presented for datasets.

Data Set	Model	*RMSE*		*MAE*		*MAPE* (%)	
		Avg.	Std.	Avg.	Std.	Avg.	Std.
CST (*MPa*)	GS-SVR	**0.56**	0.042	**0.42**	0.028	**1.18**	0.082
	PSO-SVR	0.57	0.040	0.43	0.029	1.21	0.081
	BO-SVR	**0.56**	0.047	**0.42**	0.039	1.20	0.108
	FGS-SVR	**0.56**	0.044	**0.42**	0.031	1.20	0.094
EEC (*kW*)	GS-SVR	**1.21**	0.050	**0.79**	0.022	**3.12**	0.074
	PSO-SVR	1.30	0.086	0.83	0.045	3.19	0.170
	BO-SVR	1.30	0.057	0.84	0.038	3.33	0.155
	FGS-SVR	**1.21**^{PSO,BO}	0.041	**0.79**^{PSO,BO}	0.017	3.13^{BO}	0.066
EEH (*kW*)	GS-SVR	**0.46**	0.009	**0.31**	0.008	**1.54**	0.044
	PSO-SVR	**0.46**	0.013	0.32	0.007	1.56	0.038
	BO-SVR	0.51	0.068	0.36	0.044	1.78	0.206
	FGS-SVR	0.50	0.015	0.36	0.012	1.80	0.079
CCS (*MPa*)	GS-SVR	**5.57**	0.133	3.90	0.113	13.20	0.563
	PSO-SVR	5.60	0.163	3.94	0.091	13.40	0.381
	BO-SVR	5.65	0.152	3.87	0.086	13.05	0.368
	FGS-SVR	5.64	0.150	**3.61**^{PSO,BO}	0.082	**12.04**^{PSO,BO}	0.337
ASN (*dB*)	GS-SVR	**2.01**	0.048	1.37	0.031	**1.10**	0.022
	PSO-SVR	2.27	0.164	1.52	0.060	1.23	0.049
	BO-SVR	**2.01**	0.069	**1.36**	0.028	**1.10**	0.023
	FGS-SVR	2.11^{PSO}	0.050	1.43^{PSO}	0.032	1.15^{PSO}	0.025

Due to the extensive computational time (encompassing weeks) required for GS-SVR models, the models developed for the CST, EEH, EEC, and CCS datasets, along with the model for the ASN dataset, were executed 10 and 5 times, respectively, instead of the standard 20 runs.

Although the GS-SVR models achieved slightly better performances for all datasets, in general, the model performances obtained for each dataset were comparable. On the other hand, one-way ANOVA tests were carried out to assess whether the impacts of the FGS algorithm on model performances were substantially different compared to the PSO and BO algorithms to reveal the performances of the FGS algorithm statistically. As seen in Table 6, FGS achieved statistically higher model performances than PSO and BO for the EEC, CCS, and ASN datasets (as indicated with superscripts ^{PSO} and ^{BO}).

In addition to the performance metrics, the t_{opt} and $Cost_{eval}$ values obtained during the optimization phase of the developed models are presented in Table 7. In general, the execution time required for optimizing the hyperparameters is directly related to the number of developed submodels (evaluated cost functions), as well as the complexity of the problem, the size of the dataset, and the mathematical model of the optimizer. For GS, according to the *GridSize* of each hyperparameter (see Table 2), 4032 ($= 18 \times 16 \times 14$) different submodels have to be developed to determine the best hyperparameter set for this study. That is why GS becomes a time-consuming process. On the other hand, for the PSO and BO algorithms using the values of the parameters decided in the preliminary study, the $Cost_{eval}$ value was determined as 400 and 90, respectively.

Table 7. Execution times and $Cost_{eval}$ values of the optimization phase of the models on datasets.

Model	t_{opt} $(minutes)$					t_{opt} $(times)$*	$Cost_{eval}$
	CST	EEC	EEH	CCS	ASN		
GS-SVR	18.41	880.63	930.41	407.95	2645.95	125.5	4032
PSO-SVR	2.90	103.71	125.58	53.62	361.13	17.0	400
BO-SVR	0.62	12.21	13.98	5.41	38.83	2.3	90
FGS-SVR	0.14	6.96	7.15	3.33	22.65	1.0	31.3 ([28, 35]**)

* Average optimization time of the GS-, PSO-, and BO-SVR models relative to that of FGS-SVR for all datasets.
** The ranges obtained from all FGS-SVR models were considered to get the final $Cost_{eval}$ value.

In the GS-inspired FGS algorithm, there is no need to evaluate all 4032 submodels during the optimization phase to ascertain the optimal hyperparameters, unlike the GS algorithm. The FGS algorithm performs only a fast search on the discretized search space, comparing the performance of a limited number of submodels (detailed in Section 2.5.4). This adaptive fast search algorithm automatically ends when the most appropriate hyperparameters are found. For FGS, the number of evaluations is highly dependent on the *GridSize* of each hyperparameter and secondarily on the FGS parameters *SwitchSize* and $StepSize_{max}$. Considering all FGS-SVR models in this study, the FGS algorithm only used up to 35 hyperparameter sets.

To better understand how the FGS algorithm works, Figure 4 shows a sample output of the optimization phase of the FGS-SVR model for a single fold on dataset CS103. This sample output reveals that only 34 of the 4032 different hyperparameter sets used in the GS algorithm were used to optimize the hyperparameters of the SVR model by using the FGS algorithm.

Moreover, Figure 5 visually represents the average convergence rates of the FGS, PSO, and BO algorithms over 20 runs for all datasets. The presented figure illustrates that the FGS algorithm has a notable ability for convergence, requiring significantly fewer evaluations compared to the PSO and BO algorithms.

Overall, it can be concluded from the above results that the FGS algorithm is capable of optimizing SVR hyperparameters safely and quickly.

3.3. Comparing results

Although the optimization of SVMs is found attractive by researchers and there are many studies on this subject, as given in the introduction, a limited number of modified GS-based algorithms have been proposed in the literature recently for the optimization of SVMs. In addition, it is not possible to directly compare this study with previous studies on SVM optimization due to the difference in the optimizers utilized, the parameters of the optimizers utilized, the datasets utilized, the performance metrics depending on the problem type (regression or classification), and the number of parameters to be optimized. However, apart from such incomparable studies, there are several robust studies suggesting an improved GS algorithm for SVM optimization, which are similar in principle to this study. From this point of view, the present study can be compared superficially with those studies regarding model performance, execution/optimization time, and/or convergence rate/number of evaluations.

Sun et al. [15] introduced a customized version of the GS algorithm, denoted as IGS, aimed at hyperparameter optimization for SVR involving parameters C and γ. This modified IGS-SVR model was specifically applied to forecast levels within soil and plant analyzer development pertaining to rice leaves. The authors noted that the convergence rate of the IGS algorithm remained relatively sluggish compared to well-established

optimization techniques such as PSO and GA. Furthermore, they observed instances where the prediction performance of the IGS algorithm might not surpass that of the conventional GS algorithm. In our study, PSO was also used for comparison purposes. When the results obtained in our study and [15] are evaluated, considering the execution times of the models, the IGS-SVR model is 1.61 times more time-consuming than the PSO-SVR model, while on the contrary, the FGS-SVR model is 17 times faster than the PSO-SVR on average. Other than this, the convergence rate of the FGS algorithm is 12.78 times faster than the PSO algorithm in our study.

```
#DateTime: 16-Sep-2022 15:26:18
  #Fold: 1/10 #1#DataSet: CS103 #Optimizer: FGS-GridS[14 18 16]-StepS[3]-ShiftS[3]
#Optimizing [g] ------------------------------------------------------------------------------------
   #Eval=  1 | g=   0.0313 | b= 32768.000 | e=   0.0010 | indx=  1 | err=  59.494
   #Eval=  2 | g= 256.0000 | b= 32768.000 | e=   0.0010 | indx= 14 | err=   6.687
   #Eval=  3 | g=   0.2500 | b= 32768.000 | e=   0.0010 | indx=  4 | err=  54.456
   #Eval=  4 | g=   2.0000 | b= 32768.000 | e=   0.0010 | indx=  7 | err=   4.538
#Optimizing [b] ------------------------------------------------------------------------------------
   #Eval=  5 | g=   2.0000 | b=     0.250 | e=   0.0010 | indx=  1 | err=  53.858
   #Eval=  - | g=   2.0000 | b= 32768.000 | e=   0.0010 | indx= 18 | err=   4.538
   #Eval=  6 | g=   2.0000 | b=     2.000 | e=   0.0010 | indx=  4 | err=  25.613
   #Eval=  7 | g=   2.0000 | b=    16.000 | e=   0.0010 | indx=  7 | err=   4.741
   #Eval=  8 | g=   2.0000 | b=   128.000 | e=   0.0010 | indx= 10 | err=   4.538
#Optimizing [e] ------------------------------------------------------------------------------------
   #Eval=  - | g=   2.0000 | b= 32768.000 | e=   0.0010 | indx=  1 | err=   4.538
   #Eval=  9 | g=   2.0000 | b= 32768.000 | e=  32.0000 | indx= 16 | err=  59.519
   #Eval= 10 | g=   2.0000 | b= 32768.000 | e=   4.0000 | indx= 13 | err=  16.417
   #Eval= 11 | g=   2.0000 | b= 32768.000 | e=   0.5000 | indx= 10 | err=   5.520
   #Eval= 12 | g=   2.0000 | b= 32768.000 | e=   0.1250 | indx=  8 | err=   4.736
#Optimizing [g] ------------------------------------------------------------------------------------
   #Eval=  - | g=   2.0000 | b= 32768.000 | e=   0.0010 | indx=  7 | err=   4.538
   #Eval=  - | g= 256.0000 | b= 32768.000 | e=   0.0010 | indx= 14 | err=   6.687
   #Eval= 13 | g=  64.0000 | b= 32768.000 | e=   0.0010 | indx= 12 | err=   6.029
   #Eval= 14 | g=  32.0000 | b= 32768.000 | e=   0.0010 | indx= 11 | err=   1.712
#Optimizing [b] ------------------------------------------------------------------------------------
   #Eval= 15 | g=  32.0000 | b=   128.000 | e=   0.0010 | indx= 10 | err=  10.731
   #Eval=  - | g=  32.0000 | b= 32768.000 | e=   0.0010 | indx= 18 | err=   1.712
   #Eval= 16 | g=  32.0000 | b=   512.000 | e=   0.0010 | indx= 12 | err=   6.379
   #Eval= 17 | g=  32.0000 | b=  1024.000 | e=   0.0010 | indx= 13 | err=   6.094
   #Eval= 18 | g=  32.0000 | b=  2048.000 | e=   0.0010 | indx= 14 | err=   5.997
#Optimizing [e] ------------------------------------------------------------------------------------
   #Eval=  - | g=  32.0000 | b= 32768.000 | e=   0.0010 | indx=  1 | err=   1.712
   #Eval= 19 | g=  32.0000 | b= 32768.000 | e=   0.1250 | indx=  8 | err=   1.798
   #Eval= 20 | g=  32.0000 | b= 32768.000 | e=   0.0313 | indx=  6 | err=   1.720
   #Eval= 21 | g=  32.0000 | b= 32768.000 | e=   0.0156 | indx=  5 | err=   1.710
#Optimizing [g] ------------------------------------------------------------------------------------
   #Eval= 22 | g=   2.0000 | b= 32768.000 | e=   0.0156 | indx=  7 | err=   4.562
   #Eval=  - | g=  32.0000 | b= 32768.000 | e=   0.0156 | indx= 11 | err=   1.710
   #Eval= 23 | g=   4.0000 | b= 32768.000 | e=   0.0156 | indx=  8 | err=   0.770
#Optimizing [b] ------------------------------------------------------------------------------------
   #Eval= 24 | g=   4.0000 | b=  2048.000 | e=   0.0156 | indx= 14 | err=   0.747
   #Eval=  - | g=   4.0000 | b= 32768.000 | e=   0.0156 | indx= 18 | err=   0.770
   #Eval= 25 | g=   4.0000 | b= 16384.000 | e=   0.0156 | indx= 17 | err=   0.770
   #Eval= 26 | g=   4.0000 | b=  8192.000 | e=   0.0156 | indx= 16 | err=   0.770
   #Eval= 27 | g=   4.0000 | b=  4096.000 | e=   0.0156 | indx= 15 | err=   0.770
    #[b= 2048.0000] optimized succesfully...
#Optimizing [e] ------------------------------------------------------------------------------------
   #Eval= 28 | g=   4.0000 | b=  2048.000 | e=   0.0010 | indx=  1 | err=   0.755
   #Eval=  - | g=   4.0000 | b=  2048.000 | e=   0.0156 | indx=  5 | err=   0.747
   #Eval= 29 | g=   4.0000 | b=  2048.000 | e=   0.0020 | indx=  2 | err=   0.755
   #Eval= 30 | g=   4.0000 | b=  2048.000 | e=   0.0039 | indx=  3 | err=   0.753
   #Eval= 31 | g=   4.0000 | b=  2048.000 | e=   0.0078 | indx=  4 | err=   0.749
    #[e=    0.0156] optimized succesfully...
#Optimizing [g] ------------------------------------------------------------------------------------
   #Eval=    | g=   4.0000 | b=  2048.000 | e=   0.0156 | indx=  8 | err=   0.747
   #Eval= 32 | g=  32.0000 | b=  2048.000 | e=   0.0156 | indx= 11 | err=   6.011
   #Eval= 33 | g=  16.0000 | b=  2048.000 | e=   0.0156 | indx= 10 | err=   1.582
   #Eval= 34 | g=   8.0000 | b=  2048.000 | e=   0.0156 | indx=  9 | err=   0.279
    #[g=    8.0000] optimized succesfully...
#EndDateTime: 16-Sep-2022 15:26:20
#Optimization Ended--------------------------------------------------------------------------------
```

Figure 4. Sample output of the optimization phase of the FGS-SVR model on dataset CS103.

Beltrami et al. [55] presented a hyperparameter optimizer abbreviated as GQ, which combines the quadtree technique with GS for SVM classification. In order to exhibit GQ's performance on SVMs, several classification benchmark datasets were utilized. They stated that the results demonstrated GQ's capability to identify hyperparameters that are comparable to those of GS, but with a substantial reduction of 78.81% to 85.12% in the number of evaluations needed. In our study, considering the results from all datasets, it was seen

that the FGS algorithm had an average of 99.22% less computational cost than the GS algorithm in optimizing SVR-based models.

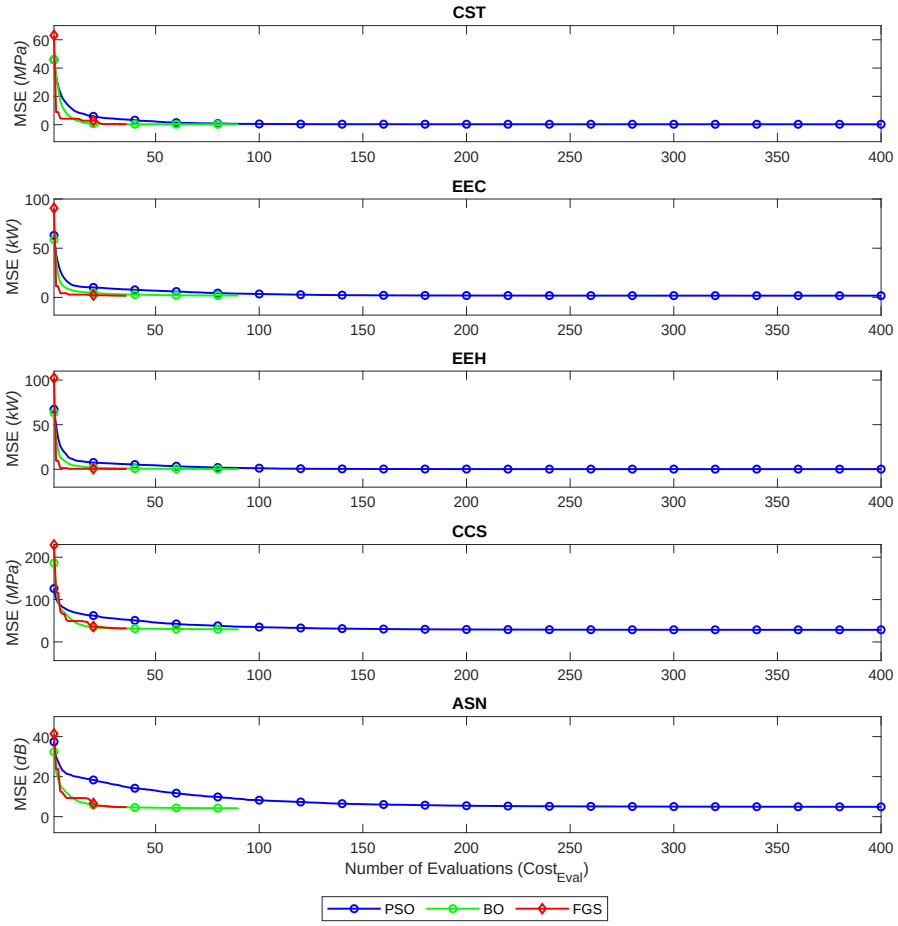

Figure 5. Average rate of convergence of the optimization algorithms over 20 runs for all datasets.

Bao and Liu [56] proposed the FGS$_{ts}$ method to enhance the optimization of hyperparameters in SVR using the RBF kernel within the domain of time-series forecasting. They used four different benchmark datasets to demonstrate how FGS$_{ts}$ performed on SVR. Their study did not provide any specific information regarding the number of evaluations conducted. However, they provided the execution times of both the GS and FGS-based SVR models for each dataset. Upon evaluation of the findings, it was seen that the proposed FGS$_{ts}$ algorithm exhibited a significant reduction in time consumption, amounting to 80% less than that of the GS method. As previously mentioned, upon examining the outcomes derived from all datasets included in our study, it is obvious that our FGS algorithm exhibits an average reduction of 99.22% in computational time compared to the GS algorithm when optimizing SVR-based models.

In addition, when the GS was compared with the effectiveness of the GS-based proposed algorithms of our study, [15], [55], and [56] in terms of SVR model performance, it was seen that comparable/similar model performances were obtained between the GS and each of the proposed algorithms.

4. Conclusions

In this study, an improved GS-inspired algorithm called FGS was proposed to obtain the most suitable hyperparameters of SVR. The primary objective of this study was to assess the efficacy of the suggested algorithm through a comparative analysis with other optimization algorithms.

In this respect, the primary comparison was conducted to evaluate the impact of the FGS algorithm on prediction performance as opposed to the impact of the conventional GS algorithm. This evaluation was performed on five different medium-sized datasets. To compare the performance of the FGS algorithm on the datasets, the well-known PSO and BO algorithms were subsequently used for SVR optimization. Through the implementation of 10-fold cross-validation, the prediction performances of all models utilizing various hyperparameter optimization algorithms were evaluated by computing the *RMSE*, *MAE*, and *MAPE* values. In addition to these metrics, t_{opt} and $Cost_{eval}$ were utilized as determinative performance measures of the models. To mitigate the impact of randomness and enhance the dependability of the outcomes, the model was executed a total of 20 times and the final results were computed by averaging the outcomes throughout those 20 runs.

When all results were evaluated according to the *RMSE*, *MAE*, and *MAPE*, it was seen that the FGS-SVR models generally produced comparable results with the GS-SVR, PSO-SVR, and BO-SVR models. In addition, the FGS algorithm obtained statistically significant model performances compared to PSO and BO for the EEC, CCS, and ASN datasets. On the other hand, the optimization phase of these models was 128.8, 12.8, and 2.9 times more time-consuming on average in terms of $Cost_{eval}$, respectively, than that of the FGS-based models. In conclusion, the results of this work revealed that the FGS-SVR models demonstrate apparent performance benefits, supporting the reliability and validity of the FGS algorithm. Therefore, the FGS algorithm could be safely used as a hyperparameter optimizer of SVR, taking advantage of much shorter computational times. Accordingly, the FGS algorithm can be used for the hyperparameter optimization of other machine learning methods having multiple discrete hyperparameters in future studies.

References

[1] Tsirikoglou P, Abraham S, Contino F, Lacor C, Ghorbaniasl G. A hyperparameters selection technique for support vector regression models. Applied Soft Computing 2017; 61: 139-148. https://doi.org/10.1016/j.asoc.2017.07.017

[2] Yoo KH, Back JH, Na MG, Kim JH, Hur S et al. Prediction of golden time using SVR for recovering SIS under severe accidents. Annals of Nuclear Energy 2016; 94: 102-108. https://doi.org/10.1016/J.ANUCENE.2016.02.029

[3] Vapnik V. The Nature of Statistical Learning Theory. New York, NY, USA: Springer, 1995.

[4] Drucker H, Burges CJC, Kaufman L, Smola A, Vapnik V. Support vector regression machines. Advances in Neural Information Processing Systems 1997; 28 (7): 779-784.

[5] Ayodeji A, Liu Y. SVR optimization with soft computing algorithms for incipient SGTR diagnosis. Annals of Nuclear Energy 2018; 121: 89-100. https://doi.org/10.1016/J.ANUCENE.2018.07.011

[6] Terrault NA, Hassanein TI. Management of the patient with SVR. Journal of Hepatology 2016; 65 (1): 120-129. https://doi.org/10.1016/J.JHEP.2016.08.001

[7] Oyehan TA, Alade IO, Bagudu A, Sulaiman KO, Olatunji SO et al. Predicting of the refractive index of haemoglobin using the hybrid GA-SVR approach. Computers in Biology Medicine 2018; 98: 85-92. https://doi.org/10.1016/j.compbiomed.2018.04.024

[8] Boegli M, Stauffer Y. SVR based PV models for MPC based energy flow management. Energy Procedia 2017; 122: 133-138. https://doi.org/10.1016/J.EGYPRO.2017.07.317

[9] Jebadurai J, Peter JD. SK-SVR: Sigmoid kernel support vector regression based in-scale single image super-resolution. Pattern Recognition Letters 2017; 94: 144-153. https://doi.org/10.1016/J.PATREC.2017.04.013

[10] Zhang WY, Hong WC, Dong Y, Tsai G, Sung JT et al. Application of SVR with chaotic GASA algorithm in cyclic electric load forecasting. Energy 2012; 45 (1): 850-858. https://doi.org/10.1016/j.energy.2012.07.006

[11] Zhang X, Chen X, He Z. An ACO-based algorithm for parameter optimization of support vector machines. Expert Systems with Applications 2010; 37 (9): 6618-6628. https://doi.org/10.1016/J.ESWA.2010.03.067

[12] Wei J, Zhang R, Yu Z, Hu R, Tang J et al. A BPSO-SVM algorithm based on memory renewal and enhanced mutation mechanisms for feature selection. Applied Soft Computing 2017; 58: 176-192. https://doi.org/10.1016/J.ASOC.2017.04.061

[13] Alam MS, Sultana N, Hossain SMZ. Bayesian optimization algorithm based support vector regression analysis for estimation of shear capacity of FRP reinforced concrete members. Applied Soft Computing 2021; 105: 107281. https://doi.org/10.1016/j.asoc.2021.107281

[14] Elsayad AM, Nassef AM, Al-Dhaifallah M. Bayesian optimization of multiclass SVM for efficient diagnosis of erythemato-squamous diseases. Biomedical Signal Processing and Control 2022; 71 (Part B): 103223. https://doi.org/10.1016/j.bspc.2021.103223

[15] Sun Y, Ding S, Zhang Z, Jia W. An improved grid search algorithm to optimize SVR for prediction. Soft Computing 2021; 25: 5633-5644. https://doi.org/10.1007/s00500-020-05560-w

[16] Huang CL, Dun JF. A distributed PSO-SVM hybrid system with feature selection and parameter optimization. Applied Soft Computing 2008; 8 (4): 1381-1391. https://doi.org/10.1016/J.ASOC.2007.10.007

[17] Harish N, Mandal S, Rao S, Patil SG. Particle swarm optimization based support vector machine for damage level prediction of non-reshaped berm breakwater. Applied Soft Computing 2015; 27: 313-321. https://doi.org/10.1016/J.ASOC.2014.10.041

[18] Lin SW, Ying KC, Chen SC, Lee ZJ. Particle swarm optimization for parameter determination and feature selection of support vector machines. Expert Systems with Applications 2008; 35 (4): 1817-1824. https://doi.org/10.1016/J.ESWA.2007.08.088

[19] Narayanan G, Joshi M, Dutta P, Kalita K. PSO-tuned support vector machine metamodels for assessment of turbulent flows in pipe bends. Engineering Computations 2020; 37 (3): 981-1001. https://doi.org/10.1108/EC-05-2019-0244

[20] Li Z, Zhang J, Liu T, Wang Y, Pei J et al. Using PSO-SVR algorithm to predict asphalt pavement performance. Journal of Performance of Constructed Facilities 2021; 35 (6): 04021094. https://doi.org/10.1061/(ASCE)CF.1943-5509.0001666

[21] Chu XL, Zhao RJ. A building carbon emission prediction model by PSO-SVR method under multi-criteria evaluation. Journal of Intelligent & Fuzzy Systems 2021; 41 (6): 7473-7484. https://doi.org/10.3233/JIFS-211435

[22] Wang J, Wang X, Li X, Yi J. A hybrid particle swarm optimization algorithm with dynamic adjustment of inertia weight based on a new feature selection method to optimize SVM parameters. Entropy 2023; 25 (3): 531. https://doi.org/10.3390/e25030531

[23] Kouziokas GN. A new W-SVM kernel combining PSO-neural network transformed vector and Bayesian optimized SVM in GDP forecasting. Engineering Applications of Artificial Intelligence 2020; 92: 103650. https://doi.org/10.1016/j.engappai.2020.103650

[24] Sultana N, Hossain SMZ, Abusaad M, Alanbar N, Senan Y et al. Prediction of biodiesel production from microalgal oil using Bayesian optimization algorithm-based machine learning approaches. Fuel 2022; 309: 122184. https://doi.org/10.1016/j.fuel.2021.122184

[25] Chen KY, Wang CH. Support vector regression with genetic algorithms in forecasting tourism demand. Tourism Management 2007; 28 (1): 215-226. https://doi.org/10.1016/j.tourman.2005.12.018

[26] Jubert de Almeida B, Ferreira Neves R, Horta N. Combining support vector machine with genetic algorithms to optimize investments in Forex markets with high leverage. Applied Soft Computing 2018; 64: 596-613. https://doi.org/10.1016/J.ASOC.2017.12.047

[27] Chen KY. Forecasting systems reliability based on support vector regression with genetic algorithms. Reliability Engineering & System Safety 2007; 92 (4): 423-432. https://doi.org/10.1016/J.RESS.2005.12.014

[28] Zhan A, Du F, Chen Z, Yin G, Wang M et al. A traffic flow forecasting method based on the GA-SVR. Journal of High Speed Networks 2022; 28: 97-106. https://doi.org/10.3233/JHS-220682

[29] Luo Z, Hasanipanah M, Bakhshandeh Amnieh H, Brindhadevi K, Tahir MM. GA-SVR: A novel hybrid data-driven model to simulate vertical load capacity of driven piles. Engineering with Computers 2021; 37: 823-831. https://doi.org/10.1007/s00366-019-00858-2

[30] Zhang A, Yu D, Zhang Z. TLSCA-SVM fault diagnosis optimization method based on transfer learning. Processes 2022; 10 (2): 362. https://doi.org/10.3390/pr10020362

[31] Liu G, Jia W, Wang M, Heidari AA, Chen H et al. Predicting cervical hyperextension injury: a covariance guided sine cosine support vector machine. IEEE Access 2020; 8: 46895-46908. https://doi.org/10.1109/ACCESS.2020.2978102

[32] Li S, Fang H, Liu X. Parameter optimization of support vector regression based on sine cosine algorithm. Expert Systems with Applications 2018; 91: 63-77. https://doi.org/10.1016/j.eswa.2017.08.038

[33] Pan M, Li C, Gao R, Huang Y, You H et al. Photovoltaic power forecasting based on a support vector machine with improved ant colony optimization. Journal of Cleaner Production 2020; 277: 123948. https://doi.org/10.1016/j.jclepro.2020.123948

[34] Raji M, Tahroudi MN, Ye F, Dutta J. Prediction of heterogeneous Fenton process in treatment of melanoidin-containing wastewater using data-based models. Journal of Environmental Management 2022; 307: 114518. https://doi.org/10.1016/j.jenvman.2022.114518

[35] Ahmed HU, Mostafa RR, Mohammed A, Sihag P, Qadir A. Support vector regression (SVR) and grey wolf optimization (GWO) to predict the compressive strength of GGBFS-based geopolymer concrete. Neural Computing and Applications 2023; 35: 2909-2926. https://doi.org/10.1007/s00521-022-07724-1

[36] Wu W, Zhang M, Zhao L, Dong H, Zhang J. CFD-DPM data-driven GWO-SVR for fast prediction of nitrate decomposition in blast furnaces with nozzle arrangement optimization. Process Safety and Environmental Protection 2023; 176: 438-449. https://doi.org/10.1016/j.psep.2023.06.029

[37] Fan C, Zheng Y, Wang S, Ma J. Prediction of bond strength of reinforced concrete structures based on feature selection and GWO-SVR model. Construction and Building Materials 2023; 400: 132602. https://doi.org/10.1016/j.conbuildmat.2023.132602

[38] Gauthama Raman MR, Somu N, Jagarapu S, Manghnani T, Selvam T et al. An efficient intrusion detection technique based on support vector machine and improved binary gravitational search algorithm. Artificial Intelligence Review 2020; 53 (5): 3255-3286. https://doi.org/10.1007/s10462-019-09762-z

[39] Alrefaee S, Al Bakal S, Algamal Z. Hyperparameters optimization of support vector regression using black hole algorithm. International Journal of Nonlinear Analysis and Applications 2022; 13 (1): 3441-3450. https://doi.org/10.22075/ijnaa.2022.6107

[40] Huang Q, Wang C, Ye Y, Wang L, Xie N. Recognition of EEG based on improved black widow algorithm optimized SVM. Biomedical Signal Processing and Control 2023; 81: 104454. https://doi.org/10.1016/j.bspc.2022.104454

[41] Algamal ZY, Qasim MK, Lee MH, Ali TTM. Improving grasshopper optimization algorithm for hyperparameters estimation and feature selection in support vector regression. Chemometrics and Intelligent Laboratory Systems 2021; 208: 104196. https://doi.org/10.1016/j.chemolab.2020.104196

[42] Faris H, Hassonah MA, Al-Zoubi AM, Mirjalili S, Aljarah I. A multi-verse optimizer approach for feature selection and optimizing SVM parameters based on a robust system architecture. Neural Computing & Applications 2018; 30 (8): 2355-2369. https://doi.org/10.1007/s00521-016-2818-2

[43] Syarif I, Prugel-Bennett A, Wills G. SVM parameter optimization using grid search and genetic algorithm to improve classification performance. TELKOMNIKA 2016; 14 (4): 1502. https://doi.org/10.12928/telkomnika.v14i4.3956

[44] Zeng N, Qiu H, Wang Z, Liu W, Zhang H et al. A new switching-delayed-PSO-based optimized SVM algorithm for diagnosis of Alzheimer's disease. Neurocomputing 2018; 320: 195-202. https://doi.org/10.1016/J.NEUCOM.2018.09.001

[45] Tao Y, Yan H, Gao H, Sun Y, Li G. Application of SVR optimized by modified simulated annealing (MSA-SVR) air conditioning load prediction model. Journal of Industrial Information Integration 2018; 15: 247-251. https://doi.org/10.1016/J.JII.2018.04.003

[46] Liu Y, Wang R. Study on network traffic forecast model of SVR optimized by GAFSA. Chaos, Solitons & Fractals 2016; 89: 153-159. https://doi.org/10.1016/J.CHAOS.2015.10.019

[47] Zhao W, Tao T, Zio E. System reliability prediction by support vector regression with analytic selection and genetic algorithm parameters selection. Applied Soft Computing 2015; 30: 792-802. https://doi.org/10.1016/J.ASOC.2015.02.026

[48] Kalita DJ, Singh VP, Kumar V. A lightweight knowledge-based PSO for SVM hyper-parameters tuning in a dynamic environment. Journal of Supercomputing 2023; 79 (16): 18777-18799. https://doi.org/10.1007/s11227-023-05385-y

[49] Açıkkar M, Altunkol Y. A novel hybrid PSO- and GS-based hyperparameter optimization algorithm for support vector regression. Neural Computing and Applications 2023; 35 (27): 19961-19977. https://doi.org/10.1007/s00521-023-08805-5

[50] Wang X, Zhang F, Kung H, Johnson VC, Latif A. Extracting soil salinization information with a fractional-order filtering algorithm and grid-search support vector machine (GS-SVM) model. International Journal of Remote Sensing 2020; 41 (3): 953-973. https://doi.org/10.1080/01431161.2019.1654142

[51] Tang F, Wu Y, Zhou Y. Hybridizing grid search and support vector regression to predict the compressive strength of fly ash concrete. Advances in Civil Engineering 2022; 2022: 3601914. https://doi.org/10.1155/2022/3601914

[52] Abbaszadeh M, Soltani-Mohammadi S, Ahmed AN. Optimization of support vector machine parameters in modeling of Iju deposit mineralization and alteration zones using particle swarm optimization algorithm and grid search method. Computers & Geosciences 2022; 165: 105140. https://doi.org/10.1016/j.cageo.2022.105140

[53] Chi Y, Zhang Y, Li G, Yuan Y. Prediction method of Beijing electric-energy substitution potential based on a grid-search support vector machine. Energies 2022; 15 (11): 3897. https://doi.org/10.3390/en15113897

[54] Mao Y, Wang T, Duan M, Men H. Multi-objective optimization of semi-submersible platforms based on a support vector machine with grid search optimized mixed kernels surrogate model. Ocean Engineering 2022; 260: 112077. https://doi.org/j.oceaneng.2022.112077

[55] Beltrami M, Silva ACL. A grid-quadtree model selection method for support vector machines. Expert Systems with Applications 2020; 146: 113172. https://doi.org/10.1016/j.eswa.2019.113172

[56] Bao Y, Liu Z. A fast grid search method in support vector regression forecasting time series. In: Intelligent Data Engineering and Automated Learning - IDEAL 2006: 7th International Conference; Burgos, Spain; 2006. pp. 504-511. https://doi.org/10.1007/11875581_61

[57] Yeh IC. Concrete Slump Test. UCI Machine Learning Repository. Irvine, CA, USA: UCI, 2009. https://doi.org/10.24432/C5FG7D

[58] Tsanas A, Xifara A. Energy Efficiency. UCI Machine Learning Repository. Irvine, CA, USA: UCI, 2012. https://doi.org/10.24432/C51307

[59] Yeh IC. Concrete Compressive Strength. UCI Machine Learning Repository. Irvine, CA, USA: UCI, 2007. https://doi.org/10.24432/C5PK67

[60] Brooks T, Pope D, Marcolini M. Airfoil Self-Noise. UCI Machine Learning Repository. Irvine, CA, USA: UCI, 2014. https://doi.org/10.24432/C5VW2C

[61] Kennedy J, Eberhart R. Particle swarm optimization. In: Proceedings of ICNN'95 - International Conference on Neural Networks; Perth, Australia; 1995. pp. 1942-1948. https://doi.org/10.1109/ICNN.1995.488968

[62] Clerc M, Kennedy J. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation 2002; 6 (1): 58-73. https://doi.org/10.1109/4235.985692

[63] Bansal JC, Singh PK, Saraswat M, Verma A, Jadon SS et al. Inertia weight strategies in particle swarm optimization. In: Third World Congress on Nature and Biologically Inspired Computing; Salamanca, Spain; 2011. pp. 633-640. https://doi.org/10.1109/NaBIC.2011.6089659

[64] Hossain SMZ, Sultana N, Irfan MF, Haque SM, Nasr N et al. Artificial intelligence-based super learner approach for prediction and optimization of biodiesel synthesis - A case of waste utilization. International Journal of Energy Research 2022; 46 (14): 20519-20534. https://doi.org/10.1002/er.7764