Multiscale and Multidisciplinary Modeling, Experiments and Design (2024) 7:3425–3445
https://doi.org/10.1007/s41939-024-00395-6

ORIGINAL PAPER

Estimation of California Bearing Ratio of stabilized soil with lime via considering multiple optimizers coupled by RBF neural network

Ling Yang[1]

Received: 20 November 2023 / Accepted: 6 February 2024 / Published online: 4 April 2024

Abstract
As an extensively used experiment, the California Bearing Ratio (CBR) tests the resistance of soils in subgrade layers and superstructure foundations often used to design flexible pavements. Practically, since CBR tests are time-consuming and costly, only a limited number of them could be performed over a road construction project. In these cases, artificial-based prediction methods will be helpful as they are quick and cheap. Artificial neural networks (ANNs), including Radial Basis Function (RBF), are powerful tools in prediction procedures employing modeling philosophy. On the other hand, recently, because meta-heuristics are very efficient, academics have focused more on optimization utilizing them, reasonable execution time, and significant convergence acceleration rate in solving real-world problems. In this study, three different hybrid models are introduced comprising the neural network approach along with three optimizers [including adaptive opposition slime mold algorithm (AOSMA), gradient-based optimizer (GBO), and Sine cosine algorithm (SCA)]. Predicted values of CBR in two categories of training and testing models have been compared with measured values of CBR tests. Finally, through some evaluators, the efficiency of hybrid models was evaluated, and the best-proposed model was presented for practical applications. In addition, RBAO obtained the most suitable prediction values compared to other developed models.

Keywords California Bearing Ratio · Radial basis function · Adaptive opposition slime mold algorithm · Gradient-based optimizer · Sine cosine algorithm

1 Introduction

The California Bearing Ratio (CBR) is a critical index in earth structures and geotechnical engineering, such as highway embankments, earth dams, the fills behind retaining walls, and bridge abutments. CBR tests can be conducted on compacted soil in the lab or on the ground surface. In practice (Ho and Tran 2022), since CBR tests are time-consuming and expensive to do, only a limited number of them can be conducted, for example, over a road construction project (Karunaprema 2002; Yildirim and Gunaydin 2011; Ho and Tran 2022). Therefore, artificial-based methods can be logical to be used in the CBR prediction procedures (Varol et al. 2021; Salehi et al. 2022; Xiao-xia 2022).

In predicting CBR value, validation of its correlation with other soil properties similar to the study conducted by Roy et al. (2006) can be helpful. In another study, Shukla and Kukalyekar (2004) developed a relationship between the compaction properties and CBR for compacted fly ash. Recently, Srinivasa Rao (2004) developed a relationship between the Group index and CBR. He developed tests on 150 soil samples, including various soil types. Additionally, Karunaprema and Edirisinghe (2002) and Nuwaiwu et al. (2006) conducted investigations to estimate the California Bearing Ratio from the Dynamic Cone Penetration (DCP) value and plasticity modulus.

The artificial neural network (ANN) is an extensively accepted simulation that can be used in various civil engineering branches. Therefore, the ANN is a precise solution for predicting in engineering fields (Masoumi et al. 2020; Nurlan 2022; behnam Sedaghat et al. 2023). It can be applied in various branches of geotechnical engineering (Cheng et al. 2022), such as the prediction of the bearing capacity of the pile (Das and Basudhar 2006) and slope stability (Erzin and Cetin 2013). Although ANNs are used widely in geotechnical engineering, research with the aim of CBR prediction is few, such as which estimated the California Bearing Ratio of

✉ Ling Yang
yangling869@126.com

[1] School of Informatics, Harbin Guangsha College, Harbin 150025, Heilongjiang, China

Table 1 The variables contained in the dataset and their statistical properties

Variables	Statistical properties			
	Max	Min	Avg	St. dev
Lime (LI)	20	0	5.630	4.029
Lime sludge (LS)	15	0	4.931	5.803
Curing period (CP)	1.39	1.14	1.242	0.058
Optimum moisture content (OMC)	32.65	13.7	24.058	3.880
Maximum dry density (MDD)	45	4	24.109	15.872
California bearing capacity (CBR)	156	2.2	52.980	34.387

stabilized soil by Si Ho and Quan Tran (2022) or single and multiple regression applications in CBR estimation of soil using a dataset from highways of Turkey located in different regions. The findings demonstrate that the neural network outperforms the statistical models (Yildirim and Gunaydin 2011). In a separate investigation, ANN and multiple regression approaches were employed to predict the CBR of soil mixed with lime and quarry (Sabat 2013).

Radial basis function (RBF) can be considered one of the widely used artificial neural networks to estimate the unconfined compressive strength of a soil–stabilizer mix (Bors and Pitas 1996; Heshmati et al. 2009; Yin et al. 2021). Heshmati et al. (2009) confirmed data from the results of previously published stabilization tests with previous laboratory tests. Also, they compared the RBF-based predictions with the numerical and experimental results of other researchers and found them more accurate. Sabour and Movahed (2017) developed a radial basis function neural network model to predict the soil sorption partition coefficient for about 800 organic varied and even unknown compounds. The obtained results indicated that the performance of the model is excellent.

Moreover, Shahani et al. (2021) developed four gradient-based machine learning algorithms to estimate the uniaxial compressive strength of soft sedimentary rocks at Thar Coalfield, Pakistan. They allocated a 106-point dataset identically for each algorithm into 30% for the testing and 70% for the training models. Also, Shangguan et al. (2010) applied the classical gradient-based optimization algorithm for estimating model parameters of conditioned soils in an EBP shield. They trained the neural network weights using a fast convergent approximation, namely the Levenberg–Marquardt. Comparing results from the model with simulated observations illustrates that the proposed neural network has better identification accuracy and higher computing efficiency. Chen et al. (2023) studied various advanced versions of this optimizer. Then, they sorted this algorithm's application domains and analyzed its shortcomings, development status, and role in each research domain. This review not only suggested possible future research directions in this field but also provided a complete source of information about the slime mold algorithm and its advanced versions.

In this current research investigation, three distinct hybrid models have been developed, employing the radial basis function (RBF) neural network approach and integrating three different optimizers: the adaptive opposition slime mold algorithm (AOSMA), the gradient-based optimizer (GBO), and the sine cosine algorithm (SCA). The primary aim of these models is to predict the CBR value with precision. The efficacy of these newly devised models in forecasting CBR values was rigorously assessed using a comprehensive set of five input variables. These variables encompassed essential parameters, such as optimum moisture content, lime percentage, maximum dry density, curing period, and lime sludge percentage. The combined power of these variables facilitated robust predictions of the CBR value. To validate the predictive prowess of the models, a comparison was made between the estimated CBR values and observed values. This comparison was performed in two distinct categories: training and testing models. Subsequently, the model demonstrating the most remarkable performance in CBR value estimation is presented as the optimal choice for practical applications. This research venture not only advances the understanding of CBR value prediction but also offers a reliable and potent tool for real-world applications in the field.

2 Materials and methodology

2.1 Dataset description

Experimental records presented in Table 1 were used to assess lime sludge's and lime's impact on CBR in different curing periods (Ikeagwuani 2021). Input parameters considered in measuring CBR values included five effective variables: curing period (CP(day)), lime sludge percentage (LS(%)), lime percentage (LI(%)), maximumdrydensity(MDD(g/cc))and, optimum moisture content [OMC (%)]. Statistical properties

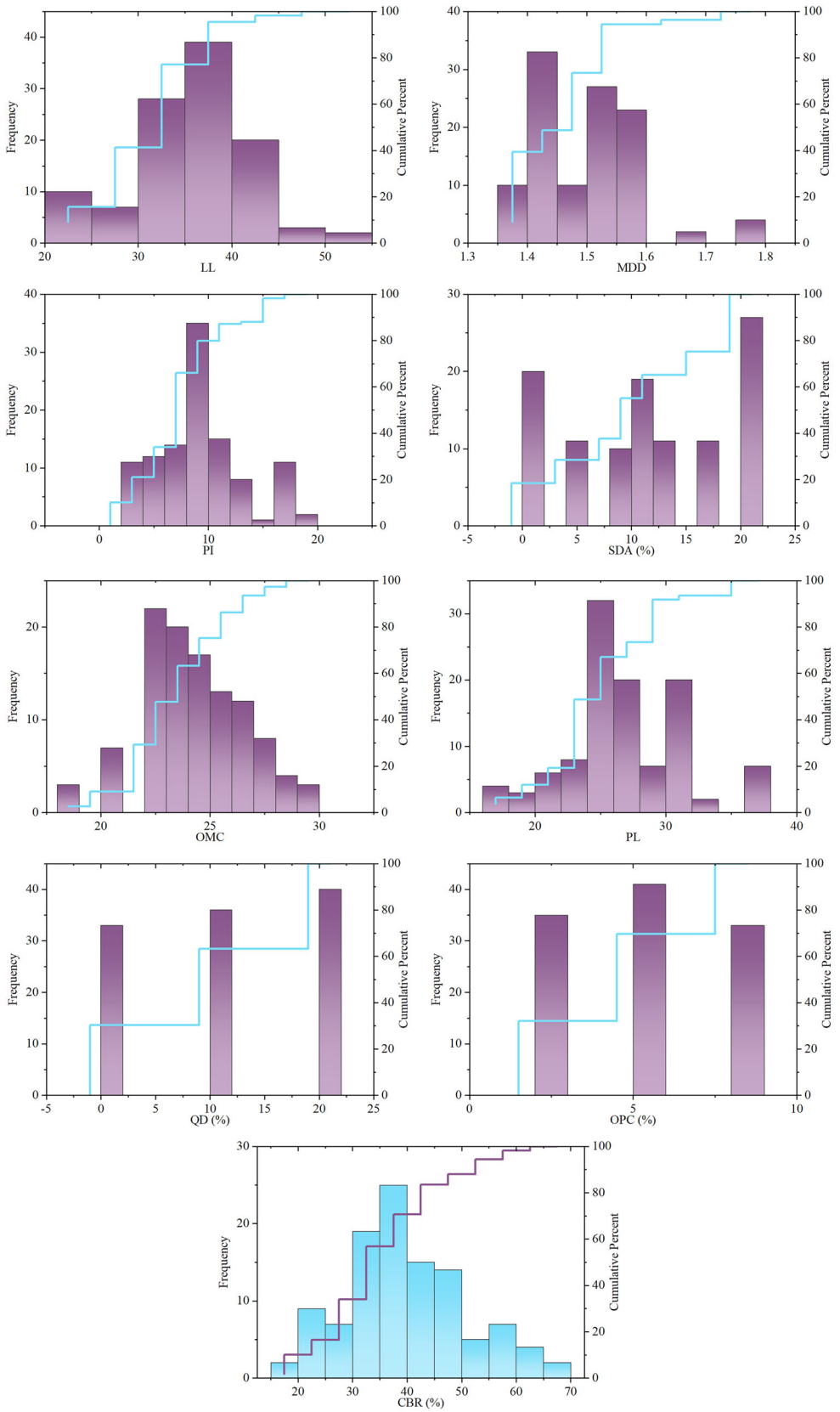

Fig. 1 Histogram cumulative plot for the input and output variables

of these variables in the dataset are reported in Table 1. Moreover, Fig. 1 shows the histogram and the cumulative plot of inputs and output.

The division of data into training and testing sets was identified as a crucial step in ensuring robust model evaluation in our study. To preserve statistical consistency and guarantee the representativeness of both sets, a random stratified sampling approach was employed. Consistency in the distribution of key variables between the training and testing sets is ensured by this method, thus, creating a reliable basis for evaluating the model's performance. This process has been clarified in the manuscript for improved transparency and understanding.

Stabilizing soil with lime enhances both physical and chemical properties. Physically, it increases density, reduces porosity, strengthens the soil, improves workability, and minimizes shrinkage. Chemically, lime raises pH, influences cation exchange capacity, stabilizes minerals, and reduces swelling potential. These modifications result in improved soil stability, reduced erosion, and enhanced nutrient availability. The effectiveness of lime stabilization depends on factors like soil type, lime dosage, and curing time, with testing required to determine optimal conditions for achieving desired results.

In this study, Python programming language was employed for data analysis and coding purposes. The choice of Python was driven by its versatility, extensive libraries, and robust capabilities in handling diverse data sets, ensuring the accuracy and efficiency of the analyses. In addition, the codes of model and optimizers mentioned in Appendix.

2.2 Radial basis function (RBF)

Artificial neural networks (ANNs), such as radial basis function (RBF) networks, are powerful tools for estimating nonlinearities by employing a modeling philosophy that does not rely on mathematical equations to define the relationship between model inputs and corresponding outputs, use the data alone to determine the structure of the model and unknown parameters. These methods can learn and upgrade as more data becomes accessible without repeating from the beginning. Thus, they outweigh the conventional methods (Alavi et al. 2009).

RBF consists of an input layer, a linear output layer, and a hidden layer. The hidden layer A Gaussian distribution based on a function that is utilized as an activation function is used to transform input vectors into RBF. Width and center are two important parameters that are associated with the Gaussian basis function (ψ_j) (Heshmati et al. 2009). This function is given in the following form:

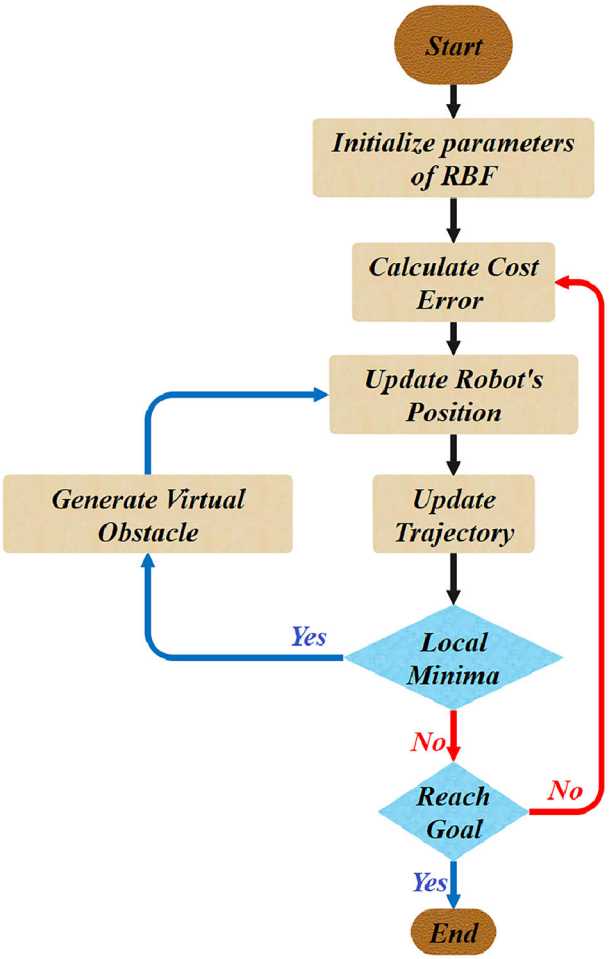

Fig. 2 Flowchart of RBF

$$\psi_j(x) = exp\left(\frac{|x-\lambda_j|^2}{2\sigma_j^2}\right) \tag{1}$$

where x is the input pattern, and σ_j and λ_j are the spread and center of the Gaussian basis function, respectively. The output neuron is written as:

$$y(x) = \sum_{j=1}^{n} \upsilon_j \psi_j(x) + a \tag{2}$$

Here, n is the number of hidden neurons, υ_j is the weight between jth hidden and the output neuron, and a is the bias term. Figure 2 shows the flowchart of RBF.

2.3 Adaptive opposition slime mold algorithm (AOSMA)

The SMA is inspired by the plasmodial SM's oscillation mode, which employs positive-negative feedback to find the

best path and oscillation to reach its nutrition source (Naik et al. 2021). In Li et al. (2020), the AOSMA is proposed, which enhances the SM's foraging behavior by incorporating adaptive decision-making based on the opposition.

To create a mathematical representation of AOSMA, it is postulated that a certain search area is inhabited by N slime molds bounded by a lower limit (LB) and an upper limit (UB). Then $X_i = (x_i^1, x_i^2, \ldots, x_i^d)$, $\forall i \in [1, N]$ is the position of ith slime mold in d-dimensions and $F(X_i)$, $\forall i = [1, N]$ represents the fitness of the ith slime. Therefore, the fitness and the position of N slime molds at the iteration t are presented in Eqs. (3) and (4):

$$X(x) = \begin{bmatrix} x_1^1 x_1^2 \cdots x_1^d \\ x_2^1 x_2^2 \cdots x_2^d \\ \vdots\vdots \\ x_N^1 x_N^2 \cdots x_N^d \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix} \tag{3}$$

$$F(X) = [F(X_1), F(X_2), \ldots, F(X_N)] \tag{4}$$

The upgraded position of the slime mold in the $(t + 1)$ iteration is as follows:

$$X_i(t+1) = \begin{cases} X_{LB}(t) + V_d(W.X_A(t) - X_B(t)) p_1 \geq \delta \quad and \quad p_2 < m_i \\ V_e.X_i(t) p_1 \geq \delta and p_2 \geq m_i, \quad \forall i \in [1, N] \\ rand.(UB - LB) + LB p_1 < z \end{cases} \tag{5}$$

X_{LB} is the best local slime mold, X_A and X_B are pooled individuals by random, W is the weight factor, and V_d and V_e are the random velocities. p_1 and p_2 are randomly chosen numbers in [0,1]. The chance of the slime mold, which is fixed at $\delta = 0.03$, refers to the initial search location that is selected randomly.

m_i is the threshold value of ith member in the population that helps opt for the position of slime mold using itself or the best alternative for the next iteration, which is evaluated in Eqs. (6–8):

$$m_i = tanh|F(X_i) - F_G|, \forall i \in [1, N] \tag{6}$$

$$F_G = F(X_G) \tag{7}$$

$$W(SortInd_F(i)) = \begin{cases} 1 + rand.\log\left(\frac{F_{LB} - F(X_i)}{F_{LB} - F_{Lw}} + 1\right) 1 \leq i \leq \frac{N}{2} \\ 1 - rand.\log\left(\frac{F_{LB} - F(X_i)}{F_{LB} - F_{Lw}} + 1\right) \frac{N}{2} < i \leq N \end{cases} \tag{8}$$

Here, F_G and X_G are the value of global best fitness and global best position, respectively. The variable $rand$ represents a randomly generated number within the interval [0,1]. F_{LB} and F_{Lw} are local best and worst fitness values. In a minimization problem in Eq. (9), an ascending order as follows can be used for sorting fitness values:

$$[Sort_F, SortInd_F] = sort(F) \tag{9}$$

Now, the local best and worst fitness also the local best slime mold X_{LB} can be extracted in Eqs. (10) to (12):

$$F_{LB} = F(Sort_F(1)) \tag{10}$$

$$F_{LW} = F(Sort_F(N)) \tag{11}$$

$$X_{LB} = X(SortInd_F(1)) \tag{12}$$

V_d and V_e as random velocities are defined as follows:

$$V_d \in [-d, d] \tag{13}$$

$$V_e \in [-e, e] \tag{14}$$

$$d = \text{arctanh}\left(-\left(\frac{t}{T}\right) + 1\right) \tag{15}$$

$$e = 1 - \frac{t}{T} \tag{16}$$

In Eq. (16), T is the maximum iteration, and the algorithm converges to a solution. In the context of engineering design problems and optimizations, the Slime Mold Algorithm (SMA) has shown potential for both exploration and exploitation. Several key factors determine the improvement of the slime mold rules in SMA:

Case 1: When $p_1 \geq z$ and $p_2 < m_i$, the search guided by the slime mould's local best, denoted as X_{LB} and two random individuals X_A and X_B with velocity V_d. This stage aids in maintaining a balance between exploitation and exploration.

Case 2: When $p_1 \geq z$ and $p_2 \geq m_i$, the search is guided by the position of slime mold with a velocity V_e. This case assists in the exploiting process.

Case 3: where $p_1 < z$, the individual undergoes reinitialization within a defined search space. This step contributes to the exploration process.

In Case 1, it is demonstrated that when X_A and X_B represent two random slime molds, the probability of achieving optimal solutions through exploration and exploitation is not effectively controlled. To overcome this shortcoming, local best individual X_{LB} can be replaced byX_A. Therefore, the $ith (i = 1, 2, \ldots, N)$ member's position upgrading rule of Eq. (5) is remodeled as Eq. (17).

$$Xn_i(t) = \begin{cases} X_{LB}(t) + V_d(W.X_{LB}(t) - X_B(t)) p_1 \geq \delta \quad and \quad p_2 < m_i \\ V_e.X_i(t) p_1 \geq \delta \quad and \quad p_2 \geq m_i \\ rand.(UB - LB) + LB p_1 < \delta \end{cases} \tag{17}$$

Case 2 illustrates that slime mold exploits a place in the neighborhood. Thus, a path of lower fitness may be followed by that. To address this issue, opting for an adaptive decision mechanism proves to be a more effective alternative.

Case 3 demonstrates that although SMA provides criteria for exploration, the limited value of $\delta = 0.03$ restricts the level of exploration. To address this issue, an additional exploration supplement for SMA is required. A combined solution to Case 2's and Case 3's limitations involves using an adaptive decision strategy for determining if further exploration is necessary through opposition-based learning (OBL) [28]. OBL utilizes a defined Xop_i in the search space that is precisely opposite of the Xni for each member ($i = 1, 2, \cdots, N$) and compares it to improve the position of the following iterations. This approach helps improve convergence and prevent the likelihood of getting trapped in local minima. Therefore, the Xop_i for the ith individual in the jth ($j = 1, 2, \cdots, s$) dimension is defined in Eq. (18):

$$Xop_i^j = \min(Xn_i(t)) + max(Xn_i(t)) - Xn_i^j(t) \tag{18}$$

In Eq. (19), Xr_i represents the ith member's position in the minimization problem and is formulated as:

$$Xr_i = \begin{cases} Xop_i(t) F(Xop_i(t)) < F(Xn_i(t)) \\ Xn_i(t) F(Xop_i(t)) \geq F(Xn_i(t)) \end{cases} \tag{19}$$

When a descendant nutrient path is detected, an adaptive decision is made based on the old fitness value $f(Xi(t))$ and the current fitness value $f(Xni(t))$. This decision helps provide additional exploration when necessary. Subsequently, the next iteration's position undergoes an update as follows:

$$X_i(t+1) = \begin{cases} Xn_i(t) F(Xn_i(t)) \leq F(X_i(t)) \\ Xr_i(t) F(Xn_i(t)) > F(X_i(t)) \end{cases}, \forall i \in [1, N] \tag{20}$$

Related pseudocode is presented as follows and code of hybrid RBF-AOSMA is mentioned in Appendix.

2.4 Gradient-based optimizer (GBO)

GBO merges the population and GB approaches and the search direction to explore the search area using a pair of primary operators, namely the local escaping operators and gradient search rule specified by Newton's technique (Ahmadianfar et al. 2020), alongside a vector collection. In optimization problems, the aim is to minimize the objective function.

Begin

Inputs: N, s, T, δ and select an objective function f with search boundary range [LB, UB].

Outputs: X_G and F_G

Initialization: Randomly initialize the slime mould $X_i = (x_i^1, x_i^2, \cdots, x_i^d), \forall i \in [1, N]$ within the search boundary UB and LB for initial iteration $t = 1$.

while ($t \leq T$)

→ Calculate the fitness values $F(X)$ of N slime mould using Eq. (4).

→ Sort the fitness value using Eq. (9).

→ Update the local best fitness. F_{LB} using Eq. (10) and corresponding local best individual X_{LB} using Eq. (11).

→ Update the local worst fitness. F_{LW} using Eq. (12).

→ Update the global best fitness. F_G and corresponding global best individual X_G.

→ Update the weight W using Eq. (8).

→ Update the d using Eq. (15) and e using Eq. (16).

for (each slime mould $i = 1: N$)

○ Generate random numbers p_1 and p_2.

○ Generate the threshold value. m_i using Eq. (6).

○ Evaluate the new slime mould position. Xn_i using Eq. (17).

○ Evaluate the fitness value of new slime mold F(Xn_i).

if ($F(Xn_i) > F(X_i)$ // Adaptive decision strategy

- Estimate Xop_i using Eq. (14). //Opposition-based learning
- Select Xr_i using Eq. (19).

End

○ Update the next iteration of slime mould. X_i using Eq. (20).

end

→ Next iteration $t = t + 1$

end

Return: Global best solution space X_G.

2.4.1 Gradient search rule (GSR)

Generally, the GBO originates a speculated initial strategy and moves toward the following situation through a gradient-specified direction. The Taylor series related to $f(x + \Delta x)$ and $f(x - \Delta x)$ are definable in Eqs. (21) and (22):

$$f(x+\Delta x) = f(x) + f'(x_0)\Delta x + \frac{f''(x_0)\Delta x^2}{2!} + \frac{f^{(3)}(x_0)\Delta x^3}{3!} + \cdots \tag{21}$$

$$f(x-\Delta x) = f(x) + f'(x_0)\Delta x + \frac{f''(x_0)\Delta x^2}{2!} - \frac{f^{(3)}(x_0)\Delta x^3}{3!} + \cdots \tag{22}$$

Then, the following central differencing equation defines the first-order derivative as follows:

$$f''(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} \tag{23}$$

So, the new position is as follows:

$$x_{n+1} = x_n - \frac{2\Delta x \times f(x_n)}{f(x_n+\Delta x) - f(x_n-\Delta x)} \tag{24}$$

As justified in (Ahmadianfar et al. 2020), Eq. (25) must be rewritten as follows:

$$GSR = randn \times \frac{2\Delta x \times x_n}{(x_{worst} - x_{best} + \delta)} \tag{25}$$

Here $randn$ is a randomly chosen number, and δ determines a small value ranging between [0, 0.1].

To keep a balance between the local and global explorations for exploring promising regions within the search space and consequently to meet the optimal solution at the global level, in Eqs. (26–28), the GSR could be altered by utilizing an adaptive coefficient such as σ_1.

$$\sigma_1 = 2 \times \text{rand} \times \beta - \beta \tag{26}$$

$$\beta = \left| \alpha \times \sin\left(\frac{3\pi}{2} + \sin\left(\alpha \times \frac{3\pi}{2}\right)\right)\right| \tag{27}$$

$$\alpha = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min}) \times \left(1 - \left(\frac{m}{M}\right)^3\right)^2 \tag{28}$$

Here $\alpha_{\min} = 0.2$ and $\alpha_{\max} = 1.2$. m represent the iterations' number, and M represents the total amount. The maximum value for m is 1000.

In Eqs. (29) and (30), the motion's direction (M) is presented for exploiting the neighborhood area properly x_n:

$$M = \text{rand} \times \lambda \times (x_{\text{best}} - x_n) \tag{29}$$

$$\lambda = 2 \times \text{rand} \times \beta - \beta \tag{30}$$

Finally, based on the GSR and M, the following equation can be used to obtain an updated position (x'^m_n) of the current vector (x^m_n):

$$x'^m_n = x^m_n - GSR + M \tag{31}$$

2.4.2 Local escaping operator (LEO)

The LEO enhances the efficiency of the offered method in handling complicated issues as it assists in the rising diversity of the population in search space and goes far from local optimal solutions. For more details, see (Ahmadianfar et al. 2020).

In addition, the code of hybrid RBF-GBO is mentioned in the Appendix.

2.5 Sine cosine algorithm (SCA)

The Sine cosine algorithm (SCA) utilizes the rules of trigonometric sine and cosine to update the individuals' position toward the ideal solution. Updating equations in SCA are presented in Eq. (32) (Mirjalili 2016):

$$x^{t+1}_{ij} = \begin{cases} x^t_{ij} + p_1 * cos(p_2) * \left| p_3 b^t_{ij} - x^t_{ij} \right|, & p_4 \geq 0.5 \\ x^t_{ij} + p_1 * sin(p_2) * \left| p_3 b^t_{ij} - x^t_{ij} \right|, & p_4 < 0.5 \end{cases} \tag{32}$$

where b^t_{ij} represents the best individual's position and p_1, p_2, p_3, p_4 are random to avoid trapping into local optima. These parameters can be described in Eq. (33) (Gabis et al. 2021):

- p_1 decides whether an updated position is the best solution ($p_1 < 1$) or outwards it ($p_1 > 1$).

$$p_1 = b - t\frac{b}{T_{max}} \tag{33}$$

  where t is the present iteration,b is a constant, and T_{max} represents the maximum iterations.
- p_2 is set in [0, 2π], which dictates if the movement of a solution is toward the destination or outward it.

- p_3 assigns a weight to the terminus randomly. This permits $de-$ emphasizing ($p_3 < 1$) or emphasizing ($p_3 > 1$) the influence of the terminus of the position updating of other answers. This parameter is in the range of [0, 2].
- p_4 is in the range of [0, 1]. It acts as a switch to opt between the trigonometric functions of sine or cosine.

Related pseudocode is presented as follows, and the code of hybrid RBF-SCA is mentioned in the Appendix.

Initialize a set of solutions $X_i(i = 1,2,\ldots,n)$ *randomly*
While t is less than T_{max} *do*
Calculate the objective value for each solution
Update the destination $(P = X)$
Update the random parameters $p_1, p_2, p_3,$ *and* p_4
Update the solutions using equation (31)
End while
Return the destination P

2.6 Performance evaluation methods

The metrics for evaluation of model performance are:

- The coefficient of determination (R^2) indicates the extent to which the estimated values match the observed values, and it can be calculated using Eq. (34).

$$R^2 = \left(\frac{\sum_{i=1}^{n}(t_i - \bar{t})(e_i - \bar{e})}{\sqrt{\left[\sum_{i=1}^{n}(e_i - t)^2\right]\left[\sum_{i=1}^{n}(e_i - \bar{e})^2\right]}} \right)^2 \tag{34}$$

- The RMSE is explained in Eq. (35):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(e_i - t_i)^2} \tag{35}$$

- Normalized Root Mean Square Error (NRMSE) is as follows in Eq. (36):

$$\text{NRMSE} = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(e_i - t_i)^2}}{\frac{1}{n}\sum_{i=1}^{n}(t_i)} \tag{36}$$

- The Mean Absolute Error (MAE) represents the mean of the absolute differences between the predicted and observed values, as shown in Eq. (37):

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|e_i - t_i| \tag{37}$$

- The scatter index (SI) is defined as a function of RMSE in Eq. (38):

$$\text{SI} = \frac{\text{RMSE}}{\bar{t}} \tag{38}$$

In all the five equations, n is the number of samples, e_i represents the estimated value and $\bar{e}$ is the average of the estimated value. t_i and $\bar{t}$ represent the experimental value and the average of the practical value, respectively. In the introduced metrics, except for R^2, which is the highest desired value, the rest of the metrics should have the lowest value.

2.7 Hybridization

In this study, a novel hybrid architecture is proposed, wherein the Radial Basis Function (RBF) is combined with three distinct optimization algorithms: the adaptive opposition slime mold algorithm (AOSMA), gradient-based optimizer (GBO), and sine cosine algorithm (SCA). The unique strengths of each component are aimed to be leveraged for the enhancement of overall performance and robustness. The steps of integration are as follows:

- *Identification of components*: Identify the radial basis function (RBF) as the primary modeling component and the three chosen optimizers: Adaptive opposition slime mold algorithm (AOSMA), gradient-based optimizer (GBO), and sine cosine algorithm (SCA).

Table 2 Evaluation of introduced models

Models	Evaluators				
	R^2	RMSE	NRMSE	MAE	SI
Training stage					
RBAO	0.994	2.658	0.052	2.100	0.048
RBGB	0.973	6.212	0.121	3.463	0.114
RBSC	0.991	3.389	0.066	2.687	0.062
Testing stage					
RBAO	0.977	4.195	0.190	2.863	0.084
RBGB	0.969	5.115	0.232	3.666	0.103
RBSC	0.963	5.337	0.242	3.645	0.107

- *Understanding characteristics of components*: Attain a comprehensive understanding of the characteristics, strengths, and weaknesses inherent in each optimizer (AOSMA, GBO, and SCA) and the RBF within the specific problem domain.
- *Definition of integration strategy*: Ascertain the manner in which the RBF will be coupled with each optimizer. Consider whether the integration will follow a sequential, parallel, or hierarchical approach. Define how the optimization process will influence the parameters of the RBF.
- *Adjustment of parameters*: Modify the parameters of each optimizer and the RBF to ensure compatibility and optimal performance within the hybrid system. Fine-tune parameters based on the characteristics of the data and the specific requirements of the problem.
- *Implementation of integration*: Execute the defined integration strategy by combining the RBF with each optimizer. Develop interfaces or connectors to facilitate communication and ensure seamless interaction between the RBF and the optimizers.
- *Evaluation of performance*: Evaluate the performance of each hybrid model (RBF-AOSMA, RBF-GBO, RBF-SCA) using appropriate evaluation metrics. Assess how well the integrated components operate together and determine whether the hybridization achieves improvements over the standalone RBF.
- *Refinement through iteration*: Based on the performance evaluation, iteratively refine the hybrid models. Adjust integration parameters, revisit the selection of optimizers, and fine-tune the hybridization strategy to enhance overall performance and convergence.
- *Documentation process*: Thoroughly document the hybridization process. Include details about the RBF and each optimizer, the integration strategy, parameter settings, and any insights gained during the iterative refinement. This documentation is crucial for transparency and replicability.
- *Testing procedures*: Conduct thorough testing of each hybrid model to ensure robustness and generalizability. Verify the performance on both training and unseen data, ensuring that the hybrid models effectively leverage the optimizers for improved results.
- *Guidelines for application*: Provide clear guidelines on how each hybrid model (RBF-AOSMA, RBF-GBO, RBF-SCA) can be applied in practical scenarios. Specify input requirements, steps for obtaining predictions, and any considerations for real-world applications, taking into account the integration with the respective optimizers.

3 Results and discussion

This work presents three kinds of hybrid models that compare the observed experimental findings with anticipated values of CBR using RBF in conjunction with $AOSMA$, GBO, and SCA. The coupled models were created in the framework of RBF + AOSMA (RBAO), RBF + GBO (RBGB), and RBF + SCA (RBSC). In these hybrid models, data are split into training and test sets, which make up 70% and 30% of the final models, respectively. In order to ascertain if one version works better than another, this part compares statistical identifiers created for the produced research.

Table 2's R^2 values may be compared to see which method produces the greatest results. RBAO, for example, has the highest R^2 values (0.994 and 0.977, respectively) in both the training and testing phases. All three models' test portion R^2 values are greater than the train portion, indicating inadequate training. It is clear by comparing values of the scatter Index (SI), where lower SI denotes the maximum model accuracy, that RBAO again yields the best results in both the training and testing stages. Comparing three types of errors, including RMSE, MAE, and NRMSE, in all three criteria, testing and training RBAO with smaller values of errors show the best results.

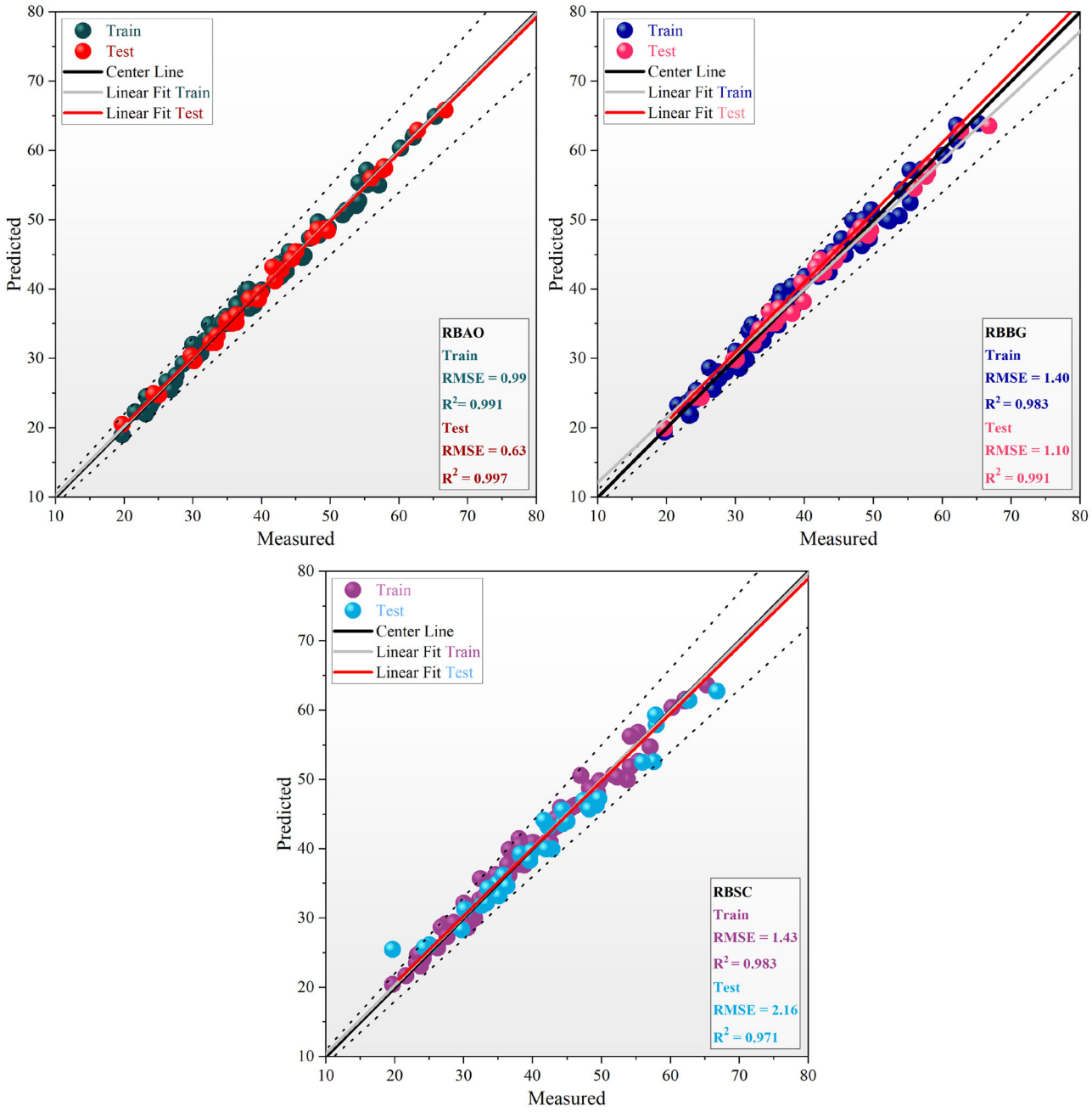

Fig. 3 Scatter plot depicting measured and predicted values

Figure 3 depicts the scatter plot of the relationship between CBR's measured and predicted values regarding RMSE and R^2 evaluators reported for each data point. The RMSE is used as a measure of dispersal, with lower values indicating higher density and less variation in the results. However, the validation and learning points are brought closer to the centerline by the R^2. A linear fit, two lines above and below the centerline at $Y = 0.9X$ and $Y = 1.1X$, and the centerline at $Y = X$ are among the other variables shown in the picture. The intersection of the upper and lower lines indicates overestimation and underestimation. Figure 3 depicts three models generated by combining three optimizers during the testing and training frameworks. The R^2 of RBAO and RBSC models are satisfactory, particularly in the training phase, where the points are clustered close to the centerline and aligned in the same direction.

The excellent match between the observed and projected CBR values in each of the three models is clearly seen in Fig. 4, indicating the viability of the established algorithms that have been suggested for CBR value estimation. The

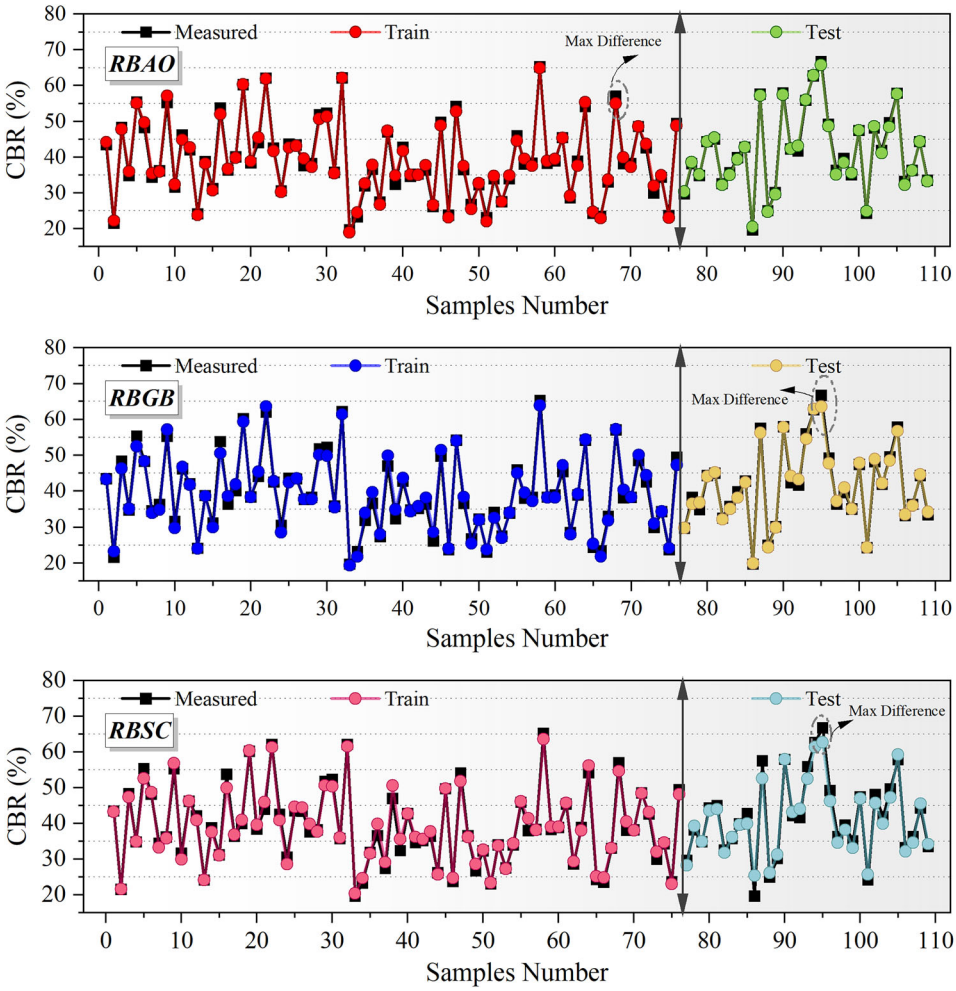

Fig. 4 Line-symbol plot for contrasting the measured values with the predicted ones

RBAO model's training section is where this agreement is most noticeable, while the RBGB training models are linked to the largest discrepancy between anticipated and observed values. Overall, it can be inferred that the suggested models demonstrate accurate performance with minimal error in predicting CBR, thereby demonstrating their potential effectiveness for practical applications.

Discrete training and testing stages of three generated models are shown in Fig. 5's histogram-distribution plot as the normal distribution of errors. Less precision in the outcome is associated with a wider tendency of spreading error. After training and testing, the narrow bell-shaped normal distribution diagram is the result of the improved performance of the RBAO hybrid model. Errors are in the 0 percent range in all three models, with RBAO being the most accurate. Through diagram comparison, it is evident that, with the exception of RBGB, where the training phase diagram has a spreading error trend that has covered a broader range than the testing diagram, the training has a spreading error trend that is as wide as the testing phase diagram's covered range.

Across all three models that were developed, it is notable that the highest prediction errors seen in the testing process are either equivalent to or less than the maximum errors encountered in the training phase. This consistency between the training and testing phases is a positive indicator of model robustness. When comparing the prediction errors of these models, as depicted in Fig. 6, it becomes evident that in the case of the RBAO model, the errors exhibit a relatively narrow range, staying consistently below the 20% threshold. In contrast, the RBGB and RBSC models display prediction errors that fluctuate at significantly higher levels, with RBGB around 60% and RBSC around 40%. This analysis leads to the conclusion that the RBAO model outperforms the other two models in accurately estimating the CBR value in practical applications. Its more stable and lower error range suggests that RBAO is the most suitable choice for accurate CBR value predictions, which can be crucial in various engineering and construction scenarios.

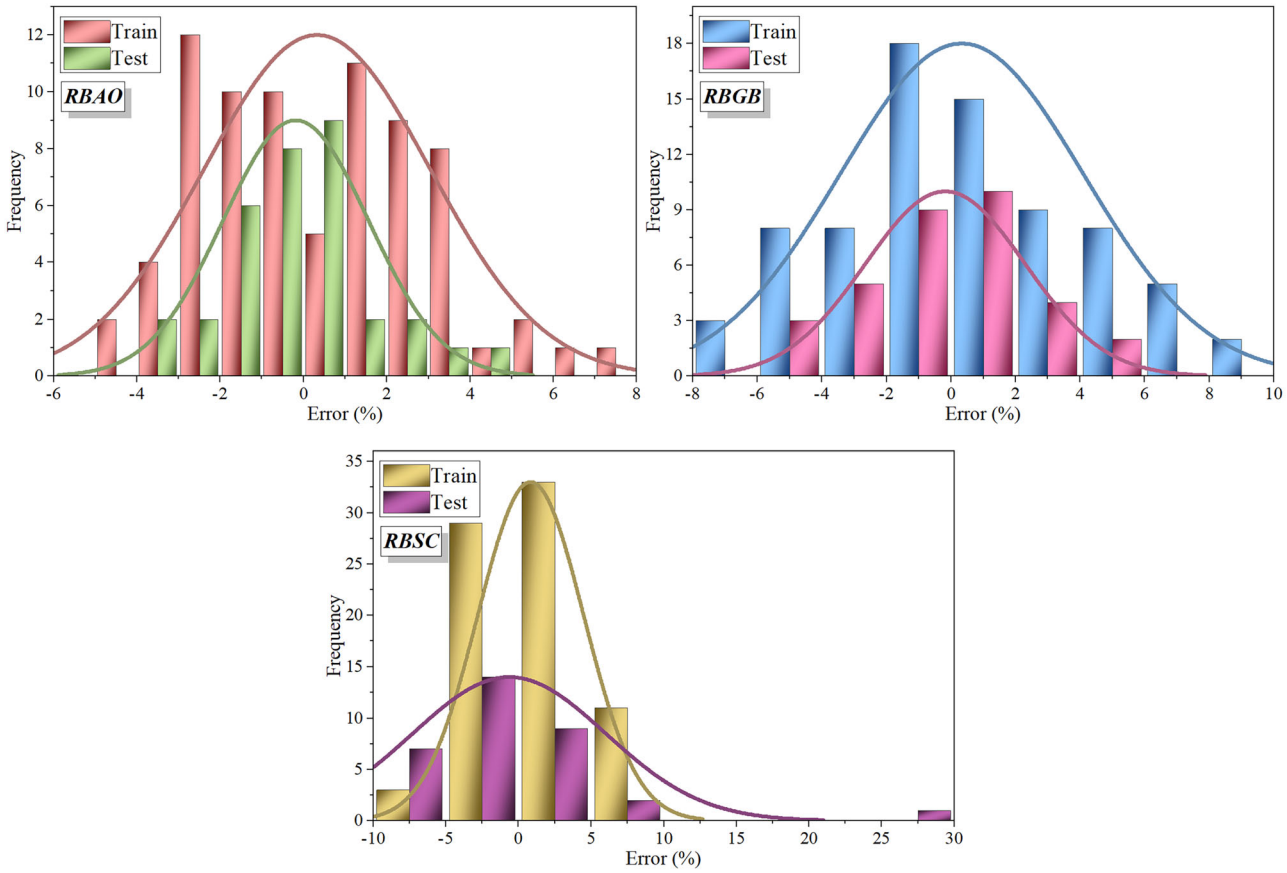

Fig. 5 Histogram-distribution plot for the error percentage of presented models

4 Conclusion

In the present research, three different hybrid models using the radial basis function (RBF) neural network are developed along with three optimizers containing adaptive opposition slime mold algorithm (AOSMA), gradient-based optimizer (GBO), and sine cosine algorithm (SCA). The performance of the models was assessed based on five input variables: maximum dry density, optimum moisture content, lime percentage, curing period, and lime sludge percentage. Finally, through some evaluators, the predicted values of CBR in two categories of training and testing models have been compared between the developed models, and the best-proposed model is presented for practical applications. Generally, RBAO had the best performance, with lower errors in training (RMSE = 2.658, MAE = 2.10, and NRMSE = 0.052), and testing models (RMSE = 4.195, MAE = 2.863, and NRMSE = 0.242). Furthermore, this model prepares accurate results with high density as it has lower values of SI in both the training and testing phases. The predicted CBR values by all of the developed models agree well with the predicted values, which demonstrates the offered hybrid algorithms' workability in the CBR estimation procedure. Therefore, all models, especially RBAO, conclude the exact anticipation having the smallest error in the CBR forecasting procedure, making them efficient for practical applications. The combination of the RBF with Adaptive AOSMA, GBO, and SCA offers a range of advantages for predicting CBR. This synergy holds the potential to significantly improve the accuracy, adaptability, and robustness of predictive models. Firstly, it enhances prediction accuracy by efficiently optimizing model parameters, leading to highly accurate forecasts. The versatility of this approach allows for the selection of different optimizers based on specific dataset characteristics, adapting to various scenarios. Moreover, it expedites model training and convergence, saving computational resources and time. The risk of overfitting is reduced through fine-tuning, ensuring the model performs well on unseen data. Optimal parameter selection and adaptability to diverse data contribute to improved robustness, making the model resilient to data variations and outliers. Lastly, this approach efficiently utilizes available data, even with noise or limited datasets, resulting in more meaningful predictions.

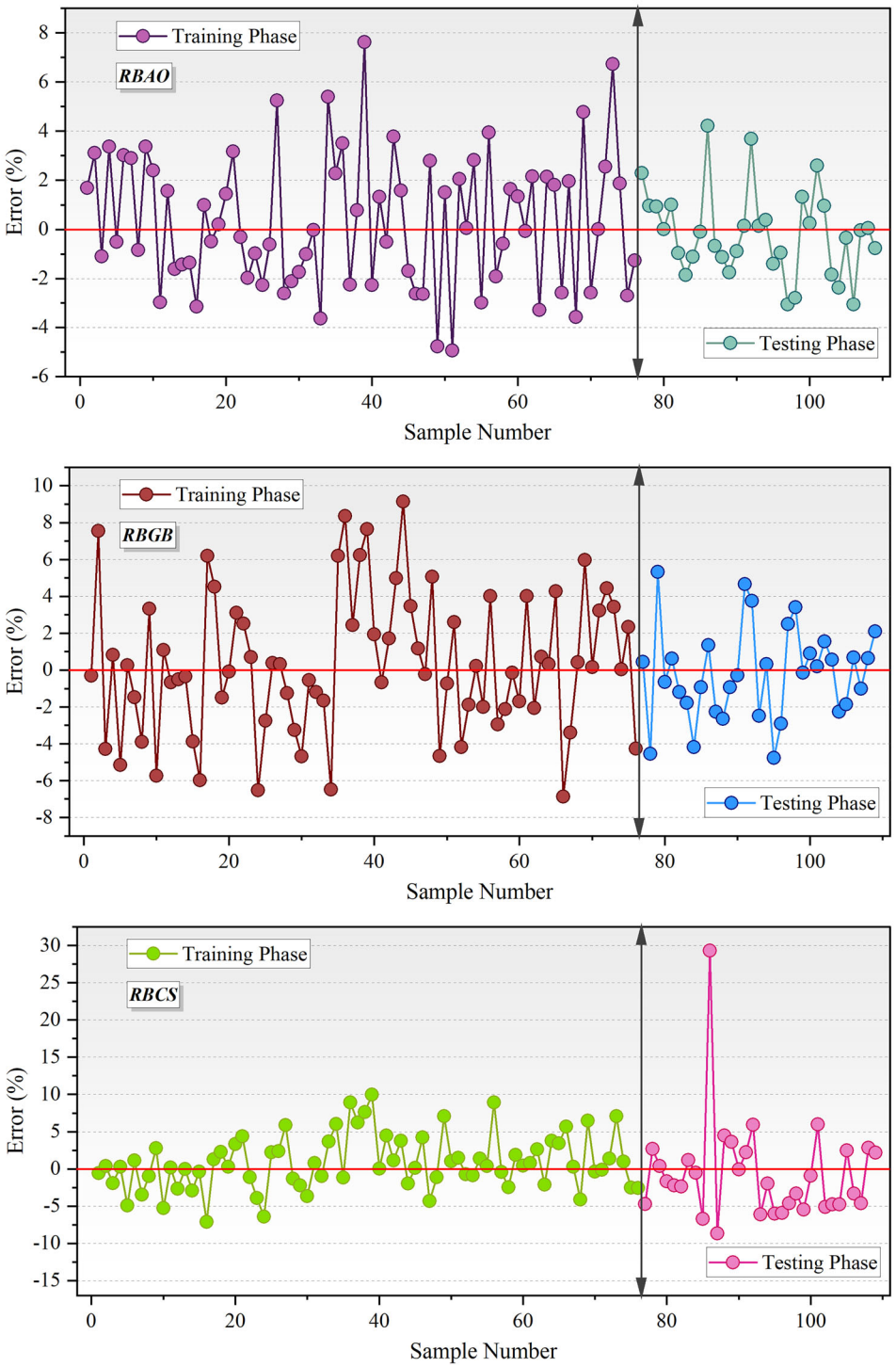

Fig. 6 Line-symbol plot for error percentage of developed hybrid models

Author contributions All authors contributed to the study's conception and design. Data collection, simulation and analysis were performed by "Ling Yang".

Funding This work was supported by Heilongjiang Higher Education Teaching Reform Project(SJGY20220741): Research and Practice on the New Student-Centered Innovation and Entrepreneurship Education System in the Field of Information Technology in Private Universities.

Data availability The authors do not have permission to share data.

Declarations

Conflict of interest The author declares no competing interests.

Appendix

RBF Code

```
clc;
clear;
close all;

%Read data
%X=(xlsread(''))';
%Y=(xlsread(''))';

net= newrb(X,Y);
ytr=sim(net,X);
ytr=ytr';
```

GBO Code

```
clc;
clear;
close all;
disp('AOSMA is optimizing your problem');

CostFunction=@COST_Function;        % Cost Function

dim=2;            % Number of Decision Variables
lb1= 1;           % Decision Variables Lower Bound
ub1= 80;         % Decision Variables Upper Bound
nP= 5;            % The swarm size.
MaxIt= 10;     % The number of iterations.

nV = dim;                        % Number f Variables
pr = 0.5;                        % Probability Parameter
lb = ones(1,dim).*lb1;            % lower boundary
ub = ones(1,dim).*ub1;            % upper boundary
X = initialization(nP,nV,ub,lb); %Initialize the set of random solutions
Convergence_curve = zeros(1,MaxIt);

%% RBF-Interies
Goal=0;
% Spread=100;
% MaxNeuron=20;
DisplayAt=400;

%%
for i=1:nP
%     Cost(i) = fobj(X(i,:));      % Calculate the Value of Objective Function
     [fitness,POP(i).pre,POP(i).Net]=CostFunction(X(i,:),XX,Y,Goal,DisplayAt);
     POP(i).Cost=fitness;
end
[~,Ind] = sort([POP.Cost]);
Best_Cost = POP(Ind(1)).Cost;        % Determine the value of Best Fitness
Best_X = X(Ind(1),:);
Worst_Cost=POP(Ind(end)).Cost;       % Determine the value of Worst Fitness
Worst_X=X(Ind(end),:);
%% Main Loop
for it=1:MaxIt

    beta = 0.2+(1.2-0.2)*(1-(it/MaxIt)^3)^2;
    alpha = abs(beta.*sin((3*pi/2+sin(3*pi/2*beta))));
    for i=1:nP
        A1 = fix(rand(1,nP)*nP)+1;                                    % Four positions randomly selected
from population
        r1 = A1(1);r2 = A1(2);
        r3 = A1(3);r4 = A1(4);

        Xm = (X(r1,:)+X(r2,:)+X(r3,:)+X(r4,:))/4;                     % Average of Four positions randomly
selected from population
        ro = alpha.*(2*rand-1);ro1 = alpha.*(2*rand-1);
```

```
        eps = 5e-3*rand;                                                % Randomization Epsilon

        DM = rand.*ro.*(Best_X-X(r1,:));Flag = 1;                       % Direction of Movement Eq.(18)
        GSR=GradientSearchRule(ro1,Best_X,Worst_X,X(i,:),X(r1,:),DM,eps,Xm,Flag);
        DM = rand.*ro.*(Best_X-X(r1,:));
        X1 = X(i,:) - GSR + DM;

        DM = rand.*ro.*(X(r1,:)-X(r2,:));Flag = 2;
        GSR=GradientSearchRule(ro1,Best_X,Worst_X,X(i,:),X(r1,:),DM,eps,Xm,Flag);
        DM = rand.*ro.*(X(r1,:)-X(r2,:));
        X2 = Best_X - GSR + DM;

        Xnew=zeros(1,nV);
        for j=1:nV
            ro=alpha.*(2*rand-1);
            X3=X(i,j)-ro.*(X2(j)-X1(j));
            ra=rand;rb=rand;
            Xnew(j) = ra.*(rb.*X1(j)+(1-rb).*X2(j))+(1-ra).*X3;
        end

        % Local escaping operator(LEO)
        if rand<pr
            k = fix(rand*nP)+1;
            f1 = -1+(1-(-1)).*rand();f2 = -1+(1-(-1)).*rand();
            ro = alpha.*(2*rand-1);
            Xk = unifrnd(lb,ub,1,nV);%lb+(ub-lb).*rand(1,nV);
            L1=rand<0.5;u1 = L1.*2*rand+(1-L1).*1;u2 = L1.*rand+(1-L1).*1;
            u3 = L1.*rand+(1-L1).*1;
            L2=rand<0.5;
            Xp = (1-L2).*X(k,:)+(L2).*Xk;

            if u1<0.5
                Xnew = Xnew + f1.*(u1.*Best_X-u2.*Xp)+f2.*ro.*(u3.*(X2-X1)+u2.*(X(r1,:)-X(r2,:)))/2;
            else
                Xnew = Best_X + f1.*(u1.*Best_X-u2.*Xp)+f2.*ro.*(u3.*(X2-X1)+u2.*(X(r1,:)-X(r2,:)))/2;
            end
        end

        % Check if solutions go outside the search space and bring them back
        Flag4ub=Xnew>ub;
        Flag4lb=Xnew<lb;
        Xnew=(Xnew.*(~(Flag4ub+Flag4lb)))+ub.*Flag4ub+lb.*Flag4lb;
%         Xnew_Cost = fobj(Xnew);
        [Xnew_Cost,New_POP.pre,New_POP.Net]=CostFunction(Xnew,XX,Y,Goal,DisplayAt);
        New_POP.Cost=Xnew_Cost;
        % Update the Best Position
        if Xnew_Cost<POP(i).Cost
            X(i,:)=Xnew;
            POP(i).Cost=Xnew_Cost;
            if POP(i).Cost<Best_Cost
                Best_X=X(i,:);
                Best_Cost=POP(i).Cost;

                bestSol.position=Best_X;
                bestSol.Cost=Best_Cost;
                bestSol.pre=POP(i).pre;
            end
        end
       % Update the Worst Position
        if POP(i).Cost>Worst_Cost
            Worst_X= X(i,:);
            Worst_Cost=POP(i).Cost;
        end
    end

    Convergence_curve(it) = Best_Cost;
    % Show Iteration Information
    disp(['Iteration ' num2str(it) ': Best Fitness = ' num2str(Convergence_curve(it))]);
```

```
end
```

SCA Code

```
clc;
clear;
close all;
disp('SCA is optimizing your problem');

%% DATA LOADING
X=(xlsread(''))';
Y=(xlsread(''))';


%% Problem Definition

CostFunction=@COST_Function;        % Cost Function

maxneurons=25;
N=maxneurons;
q=[5,N];
nVar=2;             % Number of Decision Variables
VarSize=[1 nVar];   % Decision Variables Matrix Size
VarMin=1;          % Decision Variables Lower Bound
VarMax=N;          % Decision Variables Upper Bound
Params.VarMin = VarMin;
Params.VarMax = VarMax;

nPop=10;      % The swarm size.
MaxIt=50;    % The number of iterations.

%% RBF-Interies
Goal=0;
% Spread=100;
% MaxNeuron=20;
DisplayAt=100;
%%
%Initialize the positions of search agents
Positions=initialization(nPop,nVar,VarMax,VarMin);

Convergence_curve=zeros(1,MaxIt);


% Calculate the fitness of the first set and find the best one
for i=1:size(Positions,1)
%     Objective_values(1,i)=fobj(X(i,:));
       [fitness,POP(i).pre,POP(i).Net]=CostFunction(Positions(i,:),X,Y,Goal,DisplayAt);
       POP(i).Cost=fitness;
    if i==1
        Destination_position=Positions(i,:);
        Destination_fitness=POP(1,i).Cost;
    elseif POP(1,i).Cost<Destination_fitness
        Destination_position=Positions(i,:);
        Destination_fitness=POP(1,i).Cost;
    end

    All_objective_values(1,i)=POP(1,i).Cost;
end

%Main loop
t=2; % start from the second iteration since the first iteration was dedicated to calculating the
fitness
while t<=MaxIt

    a = 2;
    MaxIt = MaxIt;
    r1=a-t*((a)/MaxIt); % r1 decreases linearly from a to 0
```

```
    % Update the position of solutions with respect to destination
    for i=1:size(Positions,1) % in i-th solution
        for j=1:size(Positions,2) % in j-th dimension

            % Update r2, r3, and r4
            r2=(2*pi)*rand();
            r3=2*rand;
            r4=rand();


            if r4<0.5
                Positions(i,j)= Positions(i,j)+(r1*sin(r2)*abs(r3*Destination_position(j)-
Positions(i,j)));
            else
                Positions(i,j)= Positions(i,j)+(r1*cos(r2)*abs(r3*Destination_position(j)-
Positions(i,j)));
            end

        end
    end

    for i=1:size(Positions,1)

        % Check if solutions go outside the search spaceand bring them back
        Flag4ub=Positions(i,:)>VarMax;
        Flag4lb=Positions(i,:)<VarMin;
        Positions(i,:)=(Positions(i,:).*(~(Flag4ub+Flag4lb)))+VarMax.*Flag4ub+VarMin.*Flag4lb;

        % Calculate the objective values
        [fitness,POP(i).pre,POP(i).Net]=CostFunction(Positions(i,:),X,Y,Goal,DisplayAt);
        POP(i).Cost=fitness;

        % Update the destination if there is a better solution
        if  POP(1,i).Cost<Destination_fitness
            Destination_position=Positions(i,:);
            Destination_fitness=POP(1,i).Cost;

            bestSol.position=Destination_position;
            bestSol.Cost=Destination_fitness;
            bestSol.pre=POP(i).pre;
            bestSol.Net=POP(i).Net;
        end
    end

    Convergence_curve(t)=Destination_fitness

    % Display the iteration and best optimum obtained so far
    if mod(t,50)==0
        display(['At iteration ', num2str(t), ' the optimum is ', num2str(Destination_fitness)]);
    end

    % Increase the iteration counter
    t=t+1;
end
```

AOSMA Code

```
%% The Adative Opposition Slime Mould Algorithm (AOSMA) function
%function [Destination_fitness,bestPositions,Convergence_curve]=AOSMA(N,Max_iter,lb,ub,dim,fobj)
clc;
clear;
close all;
disp('AOSMA is optimizing your problem');

%% DATA LOADING
%XX=(xlsread(''))';
%Y=(xlsread(''))';
```

```
%% Problem Definition

CostFunction=@COST_Function;        % Cost Function

dim=2;             % Number of Decision Variables
lb1= 1;            % Decision Variables Lower Bound
ub1= 25;          % Decision Variables Upper Bound
N= 10;             % The swarm size.
Max_iter= 50;      % The number of iterations.

bestPositions=zeros(1,dim);
Destination_fitness=inf;%change this to -inf for maximization problems
AllFitness = inf*ones(N,1);%record the fitness of all slime mold
weight = ones(N,dim);%fitness weight of each slime mold
%Initialize the set of random solutions
X=initialization(N,dim,ub1,lb1);
Convergence_curve=zeros(1,Max_iter);
lb=ones(1,dim).*lb1; % lower boundary
ub=ones(1,dim).*ub1; % upper boundary
del=0.03; % parameter

%% RBF-Interies
Goal=0;
% Spread=100;
% MaxNeuron=20;
DisplayAt=110;
%%
for i=1:N
        % Check if solutions go outside the search space and bring them back
        Flag4ub=X(i,:)>ub;
        Flag4lb=X(i,:)<lb;
        X(i,:)=(X(i,:).*(~(Flag4ub+Flag4lb)))+ub.*Flag4ub+lb.*Flag4lb;
%         AllFitness(i) = fobj(X(i,:));
        [fitness,POP(i).pre,POP(i).Net]=CostFunction(X(i,:),XX,Y,Goal,DisplayAt);
        POP(i).Cost=fitness;

end
% Main loop
it=1;
while  it <= Max_iter
    oldfitness=[POP.Cost];
    [SmellOrder,SmellIndex] = sort(oldfitness);
    worstFitness = SmellOrder(N);
    bestFitness = SmellOrder(1);
    S=bestFitness-worstFitness+eps;  % plus eps to avoid denominator zero
    %calculate the fitness weight of each slime mold
    for i=1:N
        for j=1:dim
            if i<=(N/2)
                weight(SmellIndex(i),j) = 1+rand()*log10((bestFitness-SmellOrder(i))/(S)+1);
            else
                weight(SmellIndex(i),j) = 1-rand()*log10((bestFitness-SmellOrder(i))/(S)+1);
            end
        end
    end

    %update the best fitness value and best position
    if bestFitness < Destination_fitness
        bestPositions=X(SmellIndex(1),:);
        Destination_fitness = bestFitness;

        bestSol.position=bestPositions;
        bestSol.Cost=Destination_fitness;
        bestSol.pre=POP(:,SmellIndex(1)).pre;
    end
    a = atanh(-(it/Max_iter)+1);
    b = 1-it/Max_iter;
    % Update the Position of search agents
```

```
    for i=1:N
        if rand<del
            X(i,:) = (ub-lb)*rand+lb;
        else
            p =tanh(abs(oldfitness(i)-Destination_fitness));
            vb = unifrnd(-a,a,1,dim);
            vc = unifrnd(-b,b,1,dim);
            A = randi([1,N]);  % one positions randomly selected from population
            r2 = rand();
            for j=1:dim
                if r2<p
                    X(i,j) = bestPositions(j)+ vb(j)*(weight(i,j)*bestPositions(j)-X(A,j));
                else
                    X(i,j) = vc(j)*X(i,j);
                end
            end
        end
    end
    for i=1:N
        % Check if solutions go outside the search space and bring them back
        Flag4ub=X(i,:)>ub;
        Flag4lb=X(i,:)<lb;
        X(i,:)=(X(i,:).*(~(Flag4ub+Flag4lb)))+ub.*Flag4ub+lb.*Flag4lb;
%         AllFitness(i) = fobj(X(i,:));
        [fitness,POP(i).pre,POP(i).Net]=CostFunction(X(i,:),XX,Y,Goal,DisplayAt);
        POP(i).Cost=fitness;
    end
    for i=1:N
        if POP(i).Cost>oldfitness(i)
            D(i,:)=min(X(i,:))+max(X(i,:))-X(i,:);
            Flag4ub=D(i,:)>ub;
            Flag4lb=D(i,:)<lb;
            D(i,:)=(D(i,:).*(~(Flag4ub+Flag4lb)))+ub.*Flag4ub+lb.*Flag4lb;
            [D_fitness,D_POP(i).pre,D_POP(i).Net]=CostFunction(D(i,:),XX,Y,Goal,DisplayAt);
            D_POP(i).Cost=D_fitness;
            if D_POP(i).Cost<AllFitness(i)
                X(i,:)=D(i,:);
            end
        end
    end
    Convergence_curve(it)=Destination_fitness;
    it=it+1;
end
```

References

Ahmadianfar I, Bozorg-Haddad O, Chu X (2020) Gradient-based optimizer: a new metaheuristic optimization algorithm. Inf Sci 540:131–159

Alavi AH, Gandomi AH, Gandomi M, Sadat Hosseini SS (2009) Prediction of maximum dry density and optimum moisture content of stabilised soil using RBF neural networks. IES J Part a: Civ Struct Eng 2:98–106

Behnam S, Tejani GG, Kumar S (2023) Predict the maximum dry density of soil based on individual and hybrid methods of machine learning. Adv Eng Intell Syst. https://doi.org/10.22034/aeis.2023.414188.1129

Bors AG, Pitas I (1996) Median radial basis function neural network. IEEE Trans Neural Netw 7:1351–1364

Chen H, Li C, Mafarja M et al (2023) Slime mould algorithm: a comprehensive review of recent variants and applications. Int J Syst Sci 54:204–235

Cheng H, Kitchen S, Daniels G (2022) Novel hybrid radial based neural network model on predicting the compressive strength of long-term HPC concrete. Adv Eng Intell Syst. https://doi.org/10.22034/aeis.2022.340732.1012

Das SK, Basudhar PK (2006) Undrained lateral load capacity of piles in clay using artificial neural network. Comput Geotech 33:454–459

Erzin Y, Cetin T (2013) The prediction of the critical factor of safety of homogeneous finite slopes using neural networks and multiple regressions. Comput Geosci 51:305–313

Gabis AB, Meraihi Y, Mirjalili S, Ramdane-Cherif A (2021) A comprehensive survey of sine cosine algorithm: variants and applications. Artif Intell Rev 54:5469–5540

Heshmati RAA, Alavi AH, Keramati M, Gandomi AH (2009) A radial basis function neural network approach for compressive strength prediction of stabilized soil. In: Road Pavement Material Characterization and Rehabilitation: Selected Papers from the 2009 GeoHunan International Conference. pp 147–153

Ho LS, Tran VQ (2022) Machine learning approach for predicting and evaluating California Bearing Ratio of stabilized soil containing industrial waste. J Clean Prod 370:133587

Ikeagwuani CC (2021) Estimation of modified expansive soil CBR with multivariate adaptive regression splines, random forest and gradient boosting machine. Innov Infrastruct Solut 6:199

Karunaprema KAK (2002) Some useful relationships for the use of dynamic cone penetrometer for road subgrade evaluation

Li S, Chen H, Wang M et al (2020) Slime mould algorithm: a new method for stochastic optimization. Futur Gener Comput Syst 111:300–323

Masoumi F, Najjar-Ghabel S, Safarzadeh A, Sadaghat B (2020) Automatic calibration of the groundwater simulation model with high parameter dimensionality using sequential uncertainty fitting approach. Water Supply 20:3487–3501. https://doi.org/10.2166/ws.2020.241

Mirjalili S (2016) SCA: a sine cosine algorithm for solving optimization problems. Knowl-Based Syst 96:120–133

Naik MK, Panda R, Abraham A (2021) Adaptive opposition slime mould algorithm. Soft Comput 25:14297–14313

Nurlan Z (2022) A novel hybrid radial basis function method for predicting the fresh and hardened properties of self-compacting concrete. Adv Eng Intell Syst. https://doi.org/10.22034/aeis.2022.148305

Nwaiwu CMO, Alkali IBK, Ahmed UA (2006) Properties of ironstone lateritic gravels in relation to gravel road pavement construction. Geotech Geol Eng 24:283–298

Roy TK, Chattopadhyay BC, Roy SK (2006) Prediction of CBR for subgrade of different materials from simple test. In: Proc. International Conference on 'Civil Engineering in the New Millennium–Opportunities and Challenges, BESUS, West Bengal. pp 2091–2098

Sabat AK (2013) Prediction of California bearing ratio of a soil stabilized with lime and quarry dust using artificial neural network. Electron J Geotech Eng 18:3261–3272

Sabour MR, Movahed SMA (2017) Application of radial basis function neural network to predict soil sorption partition coefficient using topological descriptors. Chemosphere 168:877–884

Salehi M, Bayat M, Saadat M, Nasri M (2022) Prediction of unconfined compressive strength and California bearing capacity of cement-or lime-pozzolan-stabilised soil admixed with crushed stone waste. Geomech Geoeng 18:1–12

Shahani NM, Kamran M, Zheng X et al (2021) Application of gradient boosting machine learning algorithms to predict uniaxial compressive strength of soft sedimentary rocks at Thar Coalfield. Adv Civ Eng 2021:1–19

Shangguan Z, Li S, Sun W, Luan M (2010) Estimating model parameters of conditioned soils by using artificial network. J Softw 5:296–303

Shukla SK, Kukalyekar MP (2004) Development of CBR correlations for the compacted fly ash. In: Proceedings of the Indian Geotechnical Conference. Warangal, pp 53–56

Srinivasa Rao K (2004) Correlation between CBR and Group Index. In: Proceedings of the Indian Geotechnical Conference. Warangal, pp 477–480

Varol T, Ozel HB, Ertugrul M et al (2021) Prediction of soil-bearing capacity on forest roads by statistical approaches. Environ Monit Assess 193:527. https://doi.org/10.1007/s10661-021-09335-0

Xiao-xia L (2022) Predicting California-bearing capacity value of stabilized pond ash with lime and lime sludge applying hybrid optimization algorithms. Multiscale Multidiscip Model, Exp Des 5:157–166

Yildirim B, Gunaydin O (2011) Estimation of California bearing ratio by using soft computing systems. Expert Syst Appl 38:6381–6391

Yin H, Liu S, Lu S et al (2021) Prediction of the compressive and tensile strength of HPC concrete with fly ash and micro-silica using hybrid algorithms. Adv Concrete Constr 12:339–354