Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2024 SCPE. Volume 25, Issues 1, pp. 55–64, DOI 10.12694/scpe.v25i1.2227

APPLICATION OF NONLINEAR BIG DATA ANALYSIS TECHNIQUES IN COMPUTER SOFTWARE RELIABILITY PREDICTION

LI GAO* AND HAI WANG†

Abstract. This research addresses the difficulties of underfitting, overfitting, and convergence to local minima in artificial neural networks for software dependability prediction. The work specifically focuses on enhancing the performance of the conventional PSO-SVM model for software reliability prediction. The analysis of the conventional PSO-SVM model and the special features of software reliability prediction serve as the foundation. An improved PSO-SVM software reliability prediction model is developed and the PSO-SVM model and a Backpropagation (BP) prediction model are compared experimentally. The critical metrics assessed include training error, and efficiency. The experimental results reveal that the training error of the enhanced PSO-LSSVM prediction model diminishes rapidly, levelling off after approximately 200 training generations. The BP prediction model requires 1,733 generations to meet training requirements. Furthermore, the improved PSO-LSSVM prediction model demonstrates significantly higher training efficiency than the BP prediction model. The optimized prediction model exhibits superior adaptability to small sample sizes, swift training, and high prediction accuracy, making it a more suitable choice for software reliability prediction applications.

Key words: Particle swarm optimization algorithm, Support vector machine, Software reliability, Prediction, Training error, Efficiency

1. Introduction. The rapid development of the Internet industry has unleashed of innovation and transformation across various sectors. This surge in technological advancement has empowered industries to harness the benefits of the Internet, such as its wide-reaching accessibility, and the seamless sharing of information. Consequently, numerous industries have seized the opportunity to develop software systems and products tailored to their specific needs, all in pursuit of greater efficiency and competitiveness. The evolution of computer software technology has emerged as a key player towards advancing information technology. These vital sectors now largely rely on sophisticated computer software systems to run their businesses and deliver services to a global client. High levels of reliability and security are necessary in a world where these systems and goods function in an open and linked network environment [10].

The dependability and safety of these software-driven systems and products are guaranteed by multifaceted strategy. First and foremost, innovative approaches and strict standards must be incorporated into the software development process. Considerable code reviews, test-driven development techniques, and adoption of alert development practices are a few examples. By doing this, programmers can guarantee the reliability and stability of their work, lowering the possibility of unanticipated failures. Second, security issues should be taken into account during the design and implementation of the product. Access control techniques and encryption technologies are essential for protecting sensitive data inside of these systems and goods. This proactive strategy assists in reducing potential weaknesses and prevents unauthorized access to sensitive information [16].

To assess security vulnerabilities within the software regularly and it is entailed by conducting routine vulnerability scans and swiftly implementing remedies to patch any identified weaknesses. Additionally, robust measures such as data backup and recovery systems should be in place to strengthen the system's flexibility in the face of unexpected disruptions or data loss. In today's digital era, the consequences of software failures cannot be underestimated. These failures can potentially cause significant disruptions and losses, affecting not only individual lives but also the trajectory of societal progress. For example, the tragic incident during the first Gulf War on February 25, 1991, where the U.S. Patriot missile system in Saudi Arabia failed to intercept

*Faculty of Computer and Software Engineering, HuaiYin Institute of Technology, Huai'an, 223003, China
†Civil and Architectural Engineering College, Nanning University, Nanning, 530000, China (Corresponding Author: haiwang39@126.com)

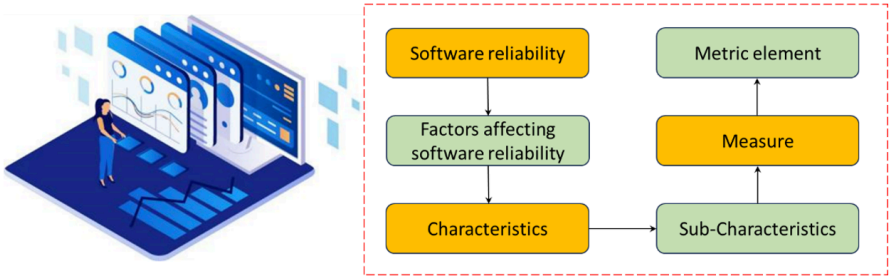

Fig. 1.1: Computation and utilization in forecasting software reliability

an Iraqi Scud missile by a mere 0.34-second error. This resulted in the tragic loss of 28 soldiers and underscored the critical importance of software reliability in military operations [1].

The aerospace industry experienced a devastating software failure on June 4, 1996, during the inaugural launch of the Ariane 5 rocket. A software malfunction caused the rocket to turn off course and explode 37 seconds after liftoff, destroying four valuable solar wind observation satellites. This incident, one of the costliest software failures in the history of space exploration, served as a stark reminder of the need for meticulous software engineering in mission-critical systems [2].

Moving closer to home, the 2011 launch of China's 12306 network ticketing system, developed for 300 million RMB, was met with high demand during the Spring Festival. However, it yielded to many issues, including website crashes, prolonged response times, payment glitches, and ticketing problems. This unfortunate episode exposed the breakability of even well-funded software systems, shedding light on architectural flaws and inadequacies in handling a massive surge in ticket transactions [12].

The vulnerabilities within the overall system architecture can be attributed to various factors. Often, system designs are not sufficiently comprehensive to account for the interactions and dependencies among different functional modules, resulting in architectural weaknesses. Additionally, programming errors, algorithmic flaws, and other implementation defects can compromise a system's normal operation or efficiency. Improper system operation and maintenance practices, such as failing to consider the impact of architectural changes during upgrades or configuration adjustments, can also lead to system failures [15].

The early iteration of the online ticketing system failed to incorporate identity verification measures, enabling scalpers to exploit the system's vulnerabilities during the Spring Festival. Even today, the system grapples with recurring challenges during peak usage, including network congestion, connection instability, and queue management issues. This ongoing struggle highlights the pressing need for sustained efforts to ensure website reliability and security. Figure 1.1 details the impact of software failures in the aviation sector. Within the aviation sector, rigorous testing protocols are followed to evaluate the software's performance. Before the first flight of an aircraft type, more than 600 software failures are meticulously scrutinized on the ground. Software-related issues have emerged as the predominant cause of aviation incidents, emphasizing the critical importance of software reliability and safety in the aviation industry [4].

The paper is organized as follows. Section 2 presents a comprehensive literature review to contextualize the research within the existing body of knowledge. The proposed PSO-SVM network is examined in detail in Section 3. The research's findings are presented in Section 4 along with an explanation. Section 5 concludes with a summary of the main findings.

2. Literature Review. In the 1970s, as information technology continued to advance, the scope of computer applications expanded gradually, leading to a sharp increase in the demand for software. However, due to the relatively primitive state of software production and management, the field remained stagnant at a level of the 1960s when computers were first introduced. This inadequacy failed to keep pace with the rapidly growing requirements of computer software development, resulting in what became known as the software crisis [3].

The software crisis established itself across several critical dimensions. First, controlling the progress of software development became a difficult challenge. Given the intricate nature of software development,

its complex designs and testing procedures, and the need for collaboration among multiple team members, maintaining a firm grip on the development timeline proved elusive, often leading to unsatisfying delays. Second, ensuring software quality proved to be equally frustrating. The absence of standardized and normalized practices within the software development process hindered the effective guarantee of software quality. Consequently, software frequently exhibited vulnerabilities and errors, sometimes resulting in system crashes and data loss [11].

The managing software costs modelled yet another tough hurdle. Owing to the inherent difficulties and problems within the software development process, cost control remained elusive, often resulting in expenditures that exceeded the established budget. Meanwhile, software systems continued to grow in scale and complexity without commensurate improvements in reliability. This mismatch led to substantial economic losses and even casualties in significant accidents [14].

A poignant illustration of the consequences of software reliability issues occurred in 2016 when the japan aerospace exploration agency (JAXA) and national aeronautics and space administration (NASA) jointly launched the X-ray astronomy satellite "Hitomi". A month after its successful launch, "Hitomi" experienced a catastrophic ground communication failure, leading to a complete loss of contact with mission control. Two months later, JAXA declared they could not regain control of the X-ray satellite "Hitomi" and were forced to abandon it [8].

Another tragic incident that underscores the gravity of software-related issues occurred on March 10, 2019, when an Ethiopian Airlines Boeing 737-MAX crashed, resulting in the tragic loss of all 157 passengers and crew members. Investigations revealed a direct link between the crash and the incorrect activation of the automatic stall prevention software known as MCAS. These real-world instances underscore the critical significance of addressing software reliability concerns. As technology advances, the imperative to enhance software development processes, elevate quality control, and mitigate vulnerabilities becomes increasingly urgent. This ensures that software can serve as a catalyst for progress rather than a source of crises. Over the decades, these painful lessons have served as cautionary stories for software practitioners, prompting the industry to grow more concerned about software reliability [9].

This heightened awareness has led to continuous development and the gradual integration of engineering principles into the software realm. In 1968 and 1969, the concept of software engineering was introduced during consecutive north atlantic treaty organization (NATO) meetings. This marked a pivotal shift as engineering principles began to guide software development practices. With software's evolution into software engineering, significant advancements were made in development technologies and management tools. Simultaneously, expectations for software reliability soared [6].

Seizing the momentum created by the burgeoning field of software engineering and drawing inspiration from traditional reliability engineering techniques, software reliability engineering emerged as a distinct discipline. This research uses SVM and PSO algorithms to investigate software reliability prediction. The inherent limitations and drawbacks of conventional PSO and SVM algorithms is investigated by examining potential optimization strategies. Consequently, an optimized PSO-SVM software reliability prediction model is established. The model's impressive predictive accuracy and applicability is demonstrated through practical examples involving limited early-stage software data samples [17, 7].

3. Research Methodology.

3.1. Analysis of traditional PSO-SVM features. The traditional PSO-SVM model is a SVM prediction model that relies on the PSO algorithm. This model possesses several noteworthy characteristics. Firstly, it is distinguished by its simplicity and ease of implementation, requiring no complicated mathematical theories or algorithmic foundations. Understanding the fundamental principles and implementation procedures of both PSO and SVM is sufficient for its deployment. Secondly, the PSO algorithm's inherent global optimization capabilities are a key feature, enabling the model to evade local optima and enhance prediction accuracy. The PSO-SVM model consistently leverages the PSO algorithm to optimize model parameters and SVM kernel parameters, improving SVM's predictive accuracy by identifying and employing the most favourable parameter combinations. Initially designed for binary classification problems, SVM later expanded into the nonlinear regression prediction. A support vector machine for nonlinear regression prediction closely resembles a classification problem as it computes a decision function based on provided data, leading to classification and prediction

outcomes. However, the regression problem retains the core characteristics of maximizing the convex function over time, with the capability to obtain nonlinear functions directly through specialized kernel functions [13].

Suppose the given data set is $\{(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), \ldots, (x_l, y_l)\}$, where $x_i \in R^n, y_i \in R, i = 1, 2, \ldots, l$. The value of y_i for classification problems is the number of corresponding data types. For example, -1 and 1 can be used for binary classification problems, while any actual number can be used for regression problems. The original SVM problem can be expressed as Equation (3.1).

$$\begin{aligned} &\min_{\boldsymbol{\omega}\in\mathbf{R}^n, b\in\mathbf{R}} J(\boldsymbol{\omega}, \xi) = \frac{1}{2}\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{\omega} + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ &\text{s.t.} \quad (\boldsymbol{\omega}\cdot\boldsymbol{\varphi}(x_i) + b) - y_i \leqslant \xi_i^* + \varepsilon; i = 1, 2, \cdots, l \\ &\quad y_i - (\boldsymbol{\omega}\cdot\boldsymbol{\varphi}(x_i) + b) \leqslant \xi_i + \varepsilon^*; i = 1, 2, \cdots, l \\ &\quad \xi_i, \xi_i^* \geqslant 0; i = 1, \cdots, l \end{aligned} \tag{3.1}$$

Given the substantial size of the eigenspace and the non-convergent nature of the objective function, we incorporate point object kernel function techniques and Wolff dual theory. These additions partition the problem into two sets of computationally manageable subproblems. One of these subproblems involves transforming the original challenge into a quadratic programming problem.

By using PSO, the model C, ε of SVM and kernel parameters σ^2 are optimized, the population continuously learns from the most available position in the current generation and the global most available position during the iterative update. Suppose the population size is m and the d^{th} dimensional space position of the ith particle is $x_i = \{x_{i1}, x_{i2}, \ldots, x_{id}\}$, and the velocity is $v_i = \{v_{i1}, v_{i2}, \ldots, v_{id}\}$, the optimal position of this generation is $pbest_i = \{p_{i1}, p_{i2}, \ldots, p_{id}\}$, and the global optimal position is $gbest = \{g_1, g_2, \ldots, g_d\}$, and d is determined by the dimensionality of the characteristic attributes of the target problem, and the fitness function is set according to the function for which the target problem is being optimized [18].

3.2. Analysis of model applicability and optimization counter measures. The conventional PSO-SVM model enhances final prediction results by optimizing the model and SVM kernel parameters using the PSO algorithm. This model exhibits numerous notable advantages that align seamlessly with the prediction attributes of software reliability. The benefits of this prediction model correspond to the distinctive characteristics of software reliability prediction in the following ways [5]:

1) SVM achieves the adjustment of the proportional relationship between structural risk and empirical risk via the modulation of model parameters. This mechanism effectively mitigates the issue of over-learning, subsequently enhancing the prediction model's capacity for generalization. Such an approach is particularly well-suited for addressing the challenges of acquiring software reliability samples, especially in cases with limited data.

2) Incorporating a kernel function into the prediction model, SVM efficiently transforms the multidimensional input space into a higher-dimensional space for prediction purposes. This transformation effectively addresses the challenges modelled by high dimensionality within the input space, making it highly compatible with the abundance of parameters typically associated with software reliability features.

3) The conventional PSO-SVM model conducts predictions in a high-dimensional space, initially transforming the original nonlinear problem into a prediction problem and subsequently deducing solutions for the nonlinear problem, thereby enhancing efficiency. This approach aligns well with the inherently nonlinear nature of software predictions.

However, despite the advantages of the traditional PSO-SVM prediction model, it grapples with computational limitations and inherent deficiencies in the PSO and SVM algorithms, rendering the model somewhat inadequate. The PSO-SVM model necessitates the adjustment of numerous parameters, including inertia weight, learning factor, and kernel function parameters, among others. Moreover, these parameters exhibit interdependencies, mandating extensive experimentation and adjustments, thereby compounding the model's complexity. The interaction between design deficiencies and proposed improvement strategies is explained as follows.

a) Coding rules originate from the experimental model. In the prediction model, when the kernel is not processed through SVM, it introduces significant randomness, thereby constraining the model's ability to

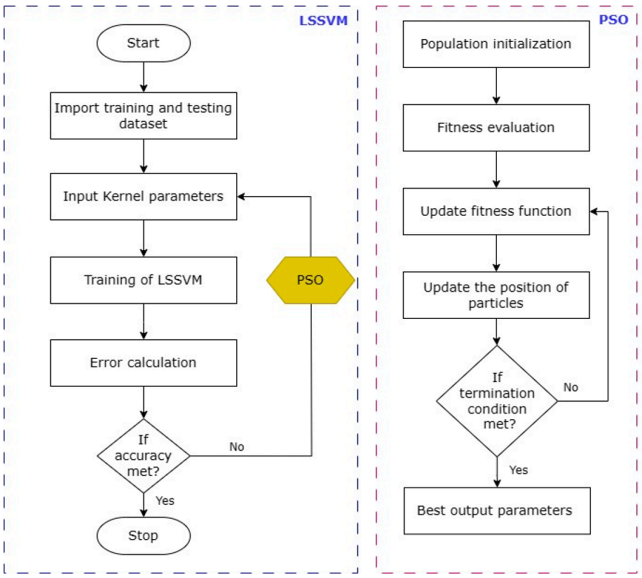

Fig. 3.1: Flowchart of improved PSO-LSSVM prediction model

undertake comprehensive global exploration in search of optimal solutions. Consequently, this can lead to a reduction in prediction accuracy and efficiency. To mitigate these limitations, a block scheme is employed to divide the initial population into several smaller subsets, facilitating the initialization of parameters. This approach enhances the diversity of the initial population, resulting in improved prediction quality and accuracy.

b) The inherent uncertainty associated with PSO within the prediction model and its constrained adaptability for conducting both global and local investigations curtails the model's capacity to achieve optimal outcomes. A variable inertia approach is employed to address this limitation, facilitating timely updates of inertia factors and optimal parameter solutions. This adjustment extends the duration of global and local surveys, enhancing the model's ability to reach optimal solutions.

c) This model transforms a low-dimensional input space into a higher-dimensional, followed by the resolution of quadratic programming problems. However, as the number of dimensions in the input space increases, computational demands grow exponentially, a significant bottleneck to the model's performance. Given the abundance of characteristic parameters in software reliability, employing the traditional PSO-SVM prediction model results in substantial computational overhead. Adapting the least squares support vector machine (LSSVM) as a modification to the SVM approach serves to streamline the SVM's details by optimizing its internal structure. Consequently, this reduces the computational burden on the prediction model and, in turn, enhances prediction efficiency.

3.3. Improved PSO-LSSVM deformation strategy. The PSO-LSSVM model represents a highly effective machine learning algorithm rooted in the SVM model's principles while optimizing model parameters through the least squares method is depicted in Figure 3.1. Additionally, it boasts robust generalization capabilities and exceptional classification performance. In the proposed model, each training sample necessitates the determination of a corresponding coefficient, ultimately forming a coefficient matrix.

The least-squares support vector machine (LSSVM) comprises two fundamental elements. Firstly, it adopts the most minor square linear system as its loss function, shifting the variable risk to the quadratic side and reformulating the support vector machine model's inequalities into equations, simplifying the constraints. Secondly, it replaces the quadratic programming approach with a system of equations to address efficiency concerns, reducing the learning complexity significantly. This transformation substantially enhances both learning efficiency and accuracy. Consequently, the original problem of the standard SVM can be simplified to the Equation (3.2)

outlined below.

$$\begin{aligned} &\min J(\boldsymbol{\omega}, \xi) = \frac{1}{2}\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{\omega} + C\sum_{i=1}^{n}\xi_i^2 \\ &\text{s.t.} \quad y_i = \boldsymbol{\varphi}(x_i)\cdot\boldsymbol{\omega} + b + \xi_i + \varepsilon \end{aligned} \tag{3.2}$$

The block population initialization mechanism is a method employed to optimize genetic algorithms. Genetic algorithms heuristic optimization techniques simulating biological evolution rely heavily on population initialization as a pivotal step. The quality of this initialization process directly impacts the effectiveness of genetic algorithms in optimization tasks. Consequently, adopting an efficient population initialization mechanism constitutes a crucial strategy for addressing optimization challenges using genetic algorithms.

In contrast, the particle swarm algorithm (PSO) is a premier global search engine, distinguishing itself from traditional search algorithms by its exceptional global search capabilities. Therefore, when employing this algorithm, active participation in its global search process is imperative to achieve favourable global outcomes swiftly. However, during the initial stages of the algorithm, traditional particles are distributed randomly across the search space, leading to an uneven population distribution. This initial inequality can be alleviated by partitioning the search space. The main idea of using chunked population initialization is to make each particle almost uniformly distributed, assuming that the number of population particles is n, then the whole search space is divided into n regions. Each small region is randomly generated one particle is shown in Equation (3.3).

$$\begin{aligned} &X_{i,k} \in [a_k + f_k(b_k - a_k), a_k + f_{k+1}(b_k - a_k)]; k = 1, 2, \cdots, D \\ &\begin{cases} f_k = (i-1) \bmod C^{D-k} \\ f_D = (i-1) - \sum_{j=1}^{D-1} f_j C^{D-j} \\ C = \sqrt[D]{n} \end{cases} \end{aligned} \tag{3.3}$$

In Equation (3.3), a_k and b_k denote the range of values on the particle position vector in the k^{th} dimension, and mod denotes the modulus. Then the initial position of the i^{th} particle can be generated according to Equation (3.4).

$$X_{i,k} = a_k + f_k(b_k - a_k) + \frac{1}{\sqrt[D]{n}}(b_k - a_k)\cdot r \tag{3.4}$$

In Equation (3.4), r is a random number taking values within $[0, 1]$.

3.4. Inertia factor approach for adaptability. In the particle swarm algorithm, the inertia factor plays a critical role by imbuing the algorithm with historical memory during the update of particle velocities. It effectively maintains the connection between historical velocities and the global and local optima, thereby influencing the balance between global and local search capabilities. The adaptive inertia factor strategy dynamically adjusts the inertia factor's magnitude based on the current search state of the particle swarm. Consequently, it assigns a higher value to the inertia factor during the early stages of the search to stimulate global exploration. As the search progresses, the inertia factor gradually diminishes, facilitating localized exploration and ultimately improving search outcomes.

During the initial iteration, boosting the inertia weight enhances the global search capability of the PSO algorithm. This allows for exploration across a broader region, enabling the rapid identification of potential solutions in the vicinity. Subsequently, maintaining a low inertia weight throughout the iteration empowers the PSO algorithm's local search capabilities. It achieves this by decelerating particle velocities, thereby fostering effective local exploration. It is evident that the initial iterations significantly influence the algorithm's global search capabilities, while the later iterations dictate the extent of local exploration. Extending the search duration between these early and later phases can enhance the overall algorithmic performance. This is precisely where the concept of weighted arm variation, as presented in Equation (3.5).

$$w = w_{\min} + (w_{\max} - w_{\min}) \times \exp\left(-20 \times (t/t_{\max})^n\right) \tag{3.5}$$

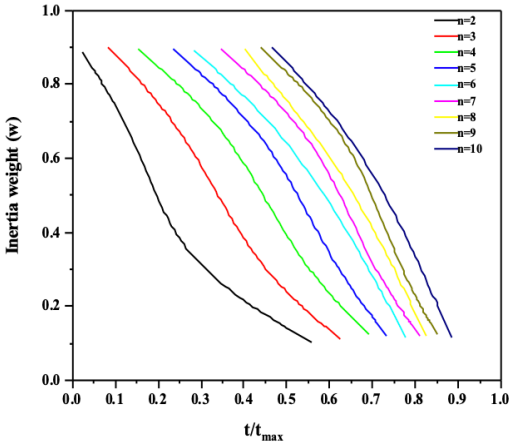

Fig. 3.2: Dynamic inertia weight adjustment curve

In Equation (3.5), $w_{\max}$ and $w_{\min}$ are taken as 0.9 and 0.1, respectively. The variation curves of the indices in Equation (3.5) are shown from the left to the right in Figure 3.1 when the indices are taken as 2 to 10 in turn. The corresponding inertia weight variation curves are shown in Figure 3.1.

As depicted in Figure 3.2, it becomes evident that when the parameter is configured as 6, both the global search duration and particle local search duration are considerably longer than other parameter values. This configuration leads to the inertia weight maintaining a high value during the initial phases of the loop and a lower value as the loop progresses. This extended duration of high inertia weight substantially strengthens the early-stage global search and late-stage local search. Consequently, this configuration achieves a well-balanced equilibrium between early-stage global search and late-stage local search, enhancing the overall global search responsiveness while concurrently augmenting local search capabilities.

4. Result Analysis and Discussions. To assess the new model's performance and compare its strengths and weaknesses against the traditional model, we conduct a series of comparative simulation experiments. In this context, we use a military software system as an illustrative example. Table 4.1 provides an overview of several metrics, including the number of module defects, for 13 modules within this military software system. Here, "SN" denotes the module number, "LOC" represents the module size in terms of lines of code, "FO" indicates module fan-out, "FI" stands for module fan-in, "PATH" reflects module control flow paths, and "FAULTS" represents the module defect count.

Evaluating the precision of an optimization model for predicting software module defect counts involved two distinct experimental trials.

In Experiment 1, all 13 data samples were partitioned into two sets. The first 10 samples were allocated for model construction and training, while the remaining 3 were reserved as test samples for assessing the model's predictive accuracy. Experiment 2 employed a different approach. The initial 6 samples were designated as training data for model development, and the subsequent 3 samples were employed as independent test data to evaluate the model's performance.

The training and prediction models under consideration encompass the BP network prediction model, PSO-SVM prediction model, and the improved PSO-LSSVM prediction model. For each of these models, normalization procedures are applied through the following steps: First, for each input feature, calculate the mean and standard deviation within both the training and testing datasets, which represent the feature's mean and variance, respectively. Next, standardize the input features by subtracting the mean of each feature in both the training and testing datasets and dividing by the standard deviation. This ensures that all input feature values are within the same order of magnitude, preventing any influence on the model's training and prediction due to differences in numerical ranges. Additionally, standardize the target variables in both the training and testing datasets to ensure they share the same order of magnitude.

Table 4.1: Summary of software metrics and defect information

Module number	Lines of code	Fan out	Fan in	Flow paths	Module defect count
1	30	3	2	3	1
2	30	3	2	3	3
3	33	1	3	1	2
4	34	2	28	3	2
5	38	6	19	15	2
6	42	6	2	13	5
7	56	1	2	11	3
8	65	5	2	13	1
9	70	2	2	7	2
10	102	3	5	11	6
11	121	2	11	21	7
12	165	13	11	220	12
13	271	8	2	79	18

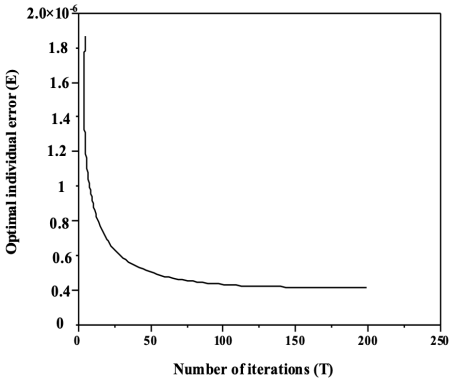

Fig. 4.1: Improved PSO-LSSVM error curve

Following data standardization, training is carried out for the BP neural network models, traditional PSO-SVM models, and improved PSO-LSSVM models. Subsequently, predictions are made on the test set using these standardized models. Specifically, the BP prediction model employs a structure with 18 hidden layers and a training target threshold of 0.00001. For the traditional PSO-SVM model, after two rounds of analysis using cross-validation and depth-first analysis, the model parameters are set to 499, and the kernel parameters are set to 5. It's important to note that both the PSO-SVM prediction model and the improved PSO-LSSVM prediction model utilize the radial basis function (RBF) as their kernel function.

During the model training process, error curves for both the BP prediction model and the improved PSO-LSSVM model are simulated. The results of these simulations are visually presented in Figure 3.2 and Figure 4.1, respectively.

Observing Figures 4.1 and 4.2, it becomes evident that the training error of the improved PSO-LSSVM prediction model exhibits rapid reduction and eventually stabilizes after approximately 200 training generations. In contrast, the BP prediction model meets the training criteria after significantly more iterations, specifically, 1,733 generations.

Following the training process, predictive models suitable for the sample data are established using the three prediction methods. Subsequently, the predictive samples are fed into their respective models for making predictions. A smoothing approach is employed to mitigate the influence of initial parameters on the BP prediction model and reduce prediction variability. This involves conducting 10 consecutive predictions using

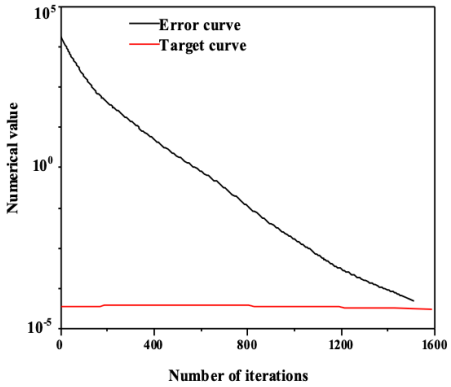

Fig. 4.2: Back propagation (BP) network error curve

Table 4.2: Comparison of prediction results of three prediction models

Experiment serial number	Test set serial number	True value	BP prediction model	Traditional PSO-SVM prediction model	Improved PSO-LSSVM prediction model
1	1	6	5. 7204	6. 2664	6. 100 9
	2	11	10. 1666	11. 4748	11. 192 5
	3	17	16. 0433	17. 6283	17. 203 7
2	1	6	5. 2234	6. 6326	6. 134 5
	3	11	9. 1999	12. 3895	11. 207 8
	4	17	14. 5860	19. 0068	17. 325 7

Table 4.3: Comparison of prediction errors of three prediction models

Experiment serial number	Test set serial number	BP prediction model	Traditional PSO-SVM prediction model	Improved PSO-LSSVM prediction model
1	1	4.66	4.44	1.68
	2	7.57	5.23	1.75
	3	5.62	3.69	1.21
	4	5.95	4.45	1.54
2	1	12.94	10.54	2.24
	2	16.37	12.62	1.89
	3	14.20	11.76	1.93
	4	14.52	11.64	2.02

the BP network and calculating the mean percentage error for the prediction outcomes. The results of these predictions are presented in Table 4.2 and Table 4.3.

In summary, enhancing and optimizing the PSO-LSSVM prediction model can be achieved by introducing several key strategies, including the adaptive inertia weight strategy, multi-objective optimization strategy, kernel function adaptive selection strategy, parallel computing strategy, and deep learning strategy. These strategies elevate the prediction model's accuracy and its generalization capabilities.

5. Conclusion. The development of the improved PSO-LSSVM prediction model through optimization techniques and non-overlapping kernels, becomes evident that the proposed model excels in prediction accuracy compared to the PSO-SVM method and the BP prediction model. Furthermore, as the number of models involved in the training process decreases, the improved PSO-LSSVM prediction model demonstrates superior overall performance compared to the PSO-SVM prediction model with fewer training samples. The predictive accuracy of the proficient PSO-LSSVM prediction model surpasses that of both the traditional PSO-SVM prediction model and the BP prediction model when trained with fewer models. Consequently, the improved PSO-LSSVM prediction model outperforms the traditional PSO-SVM and BP prediction models in terms of both training and prediction performance. This model effectively addresses the current software prediction requirements in the domestic market, especially in scenarios with limited historical data. Moreover, it can be applied to similar projects with historical models, further emphasizing its versatility and effectiveness.

REFERENCES

[1] M. K. Bhuyan, D. P. Mohapatra, and S. Sethi, *Software reliability prediction using fuzzy min-max algorithm and recurrent neural network approach.*, International Journal of Electrical & Computer Engineering, 6 (2016).

[2] G. Carrozza, R. Pietrantuono, and S. Russo, *A software quality framework for large-scale mission-critical systems engineering*, Information and Software Technology, 102 (2018), pp. 100–116.

[3] G. Coleman and R. V. O'Connor, *An investigation into software development process formation in software start-ups*, Journal of Enterprise Information Management, 21 (2008), pp. 633–648.

[4] C. Diwaker, P. Tomar, A. Solanki, A. Nayyar, N. Jhanjhi, A. Abdullah, and M. Supramaniam, *A new model for predicting component-based software reliability using soft computing*, IEEE Access, 7 (2019), pp. 147191–147203.

[5] Y. Duan, N. Chen, L. Chang, Y. Ni, S. S. Kumar, and P. Zhang, *CAPSO: Chaos adaptive particle swarm optimization algorithm*, IEEE Access, 10 (2022), pp. 29393–29405.

[6] M. P. Efthymiopoulos, *A cyber-security framework for development, defense and innovation at NATO*, Journal of Innovation and Entrepreneurship, 8 (2019), pp. 1–26.

[7] A. G. Gad, *Particle swarm optimization algorithm and its applications: a systematic review*, Archives of Computational Methods in Engineering, 29 (2022), pp. 2531–2561.

[8] J. Homola, M. Johnson, P. Kopardekar, A. Andreeva-Mori, D. Kubo, K. Kobayashi, and Y. Okuno, *UTM and D-NET: NASA and JAXA's collaborative research on integrating small UAS with disaster response efforts*, in Proceedings of the Aviation Technology, Integration, and Operations Conference, Atlanta, Georgia, 2018, p. 3987.

[9] P. Johnston and R. Harris, *The boeing 737 MAX saga: lessons for software organizations*, Software Quality Professional, 21 (2019), pp. 4–12.

[10] X. Li, H. Liu, W. Wang, Y. Zheng, H. Lv, and Z. Lv, *Big data analysis of the internet of things in the digital twins of smart city based on deep learning*, Future Generation Computer Systems, 128 (2022), pp. 167–177.

[11] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, *An empirical study of the impact of modern code review practices on software quality*, Empirical Software Engineering, 21 (2016), pp. 2146–2189.

[12] N. Medvidovic and R. N. Taylor, *Software architecture: foundations, theory, and practice*, in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, Cape Town, South Africa, 2010, ACM, pp. 471–472.

[13] C. Pozna, R.-E. Precup, E. Horváth, and E. M. Petriu, *Hybrid particle filter–particle swarm optimization algorithm and application to fuzzy controlled servo systems*, IEEE Transactions on Fuzzy Systems, 30 (2022), pp. 4286–4297.

[14] S. A. Ruk, M. F. Khan, S. G. Khan, and S. M. Zia, *A survey on adopting agile software development: issues & its impact on software quality*, in Proceedings of the 6th International Conference on Engineering Technologies and Applied Sciences, Kuala Lumpur, Malaysia, 2019, IEEE, pp. 1–5.

[15] T. Sommestad, M. Ekstedt, and H. Holm, *The cyber security modeling language: A tool for assessing the vulnerability of enterprise system architectures*, IEEE Systems Journal, 7 (2012), pp. 363–373.

[16] B. Wang, Q. Hua, H. Zhang, X. Tan, Y. Nan, R. Chen, and X. Shu, *Research on anomaly detection and real-time reliability evaluation with the log of cloud platform*, Alexandria Engineering Journal, 61 (2022), pp. 7183–7193.

[17] G. Wang, B. Zhao, B. Wu, C. Zhang, and W. Liu, *Intelligent prediction of slope stability based on visual exploratory data analysis of 77 in situ cases*, International Journal of Mining Science and Technology, 33 (2023), pp. 47–59.

[18] Z. Yu, Z. Si, X. Li, D. Wang, and H. Song, *A novel hybrid particle swarm optimization algorithm for path planning of UAVs*, IEEE Internet of Things Journal, 9 (2022), pp. 22547–22558.

Edited by: Venkatesan C
Special issue on: Next Generation Pervasive Reconfigurable Computing for High Performance Real Time Applications
Received: Apr 10, 2023
Accepted: Sep 15, 2023