Open Access and Peer Reviewed Journal ISSN 2394-2231 https://www.ijctjournal.org/

RESEARCH ARTICLE OPEN ACCESS

Development of An Enhanced Extreme Learning Machine for Software Defect Prediction.

Abstract

Software Defect Prediction (SDP) enables development teams to focus limited testing and code review effort on the modules most likely to contain faults, improving software quality while controlling costs. Extreme Learning Machine (ELM) is well suited to this task because its single-hidden-layer feedforward structure is trained by analytically solving for output weights rather than by iterative backpropagation, giving very fast training. This paper proposes a hybrid model, SMOTE-OOA-ELM, that addresses both problems jointly: the Synthetic Minority Over-sampling Technique (SMOTE) is applied to the training data to correct class imbalance before model construction, and the Osprey Optimisation Algorithm (OOA), a two-phase nature-inspired metaheuristic based on osprey hunting behaviour, searches for near-optimal ELM input weights and hidden biases in place of random initialization. The paper details the architecture of the combined model, a preprocessing and optimization pipeline, and a full experimental protocol built around NASA/PROMISE benchmark datasets, stratified cross-validation, and imbalance-aware metrics (F1-score, AUC, and Matthews Correlation Coefficient (MCC)) rather than raw accuracy. The model is positioned against plain ELM, SMOTE-ELM without metaheuristic tuning, and OOA-ELM without oversampling, isolating the individual and combined contribution of each component. Illustrative performance patterns consistent with prior swarm-optimized ELM and oversampling literature are presented to demonstrate the intended evaluation format, indicating that the combined use of SMOTE and OOA yielded larger gains in minority-class detection than either technique applied alone.

Keywords: Software Defect Prediction; Extreme Learning Machine; Osprey Optimisation Algorithm; SMOTE; Class Imbalance; Swarm Intelligence; Metaheuristic Optimisation.

1. Introduction

Software quality assurance consumes a substantial share of total software development effort, and testing resources are rarely sufficient to examine every module with equal thoroughness.

Software Defect Prediction (SDP) supports the efficient allocation of this effort by using metrics collected from previously released software — size, complexity, and structural measures — to build classifiers that flag modules likely to contain defects before they reach production. A large body of work has applied machine learning to this problem, including Naïve Bayes, Decision Trees, Support Vector Machines, and, increasingly, neural network models.

Extreme Learning Machine (ELM) is a single-hidden-layer feedforward network in which the input weights and hidden-layer biases are randomly assigned and left untrained. In contrast, the output weights are computed analytically using the Moore–Penrose generalised inverse. This design gives ELM training speeds far exceeding those of iteratively trained networks, making it attractive for SDP settings where models may need to be retrained frequently as a codebase evolves. However, ELM's dependence on random weight initialisation means that its accuracy can vary considerably between runs, and a poor initialisation may require an unnecessarily large hidden layer to compensate.

A second, largely independent challenge in SDP is class imbalance: in most public defect datasets, defective modules are a small minority of the total, often below twenty per cent. Classifiers trained naively on such data, ELM included, tend to favour the majority non-defective class, achieving high overall accuracy while missing a large proportion of true defects — precisely the outcome an SDP system is meant to prevent. The Synthetic Minority Over-sampling Technique (SMOTE) addresses this by generating synthetic minority-class instances through interpolation between existing minority samples and their nearest neighbours, rebalancing the training distribution without simply duplicating existing records.

Because weight-initialisation instability and class imbalance are distinct problems, addressing only one leaves the other unresolved: an ELM tuned by a metaheuristic optimiser but trained on imbalanced data will still under-detect defects, while an ELM trained on SMOTE-balanced data but left with random weights will still be unstable across runs.

To mitigate these problems, this paper therefore proposes SMOTE-OOA-ELM. This framework applies SMOTE to correct class imbalance in the training data. Subsequently, it uses the Osprey Optimisation Algorithm (OOA) to search for ELM input weights and biases that

minimise validation error on the rebalanced data. OOA, proposed by Dehghani and Trajkovsky, models the two-stage hunting behaviour of ospreys — locating a fish through broad exploration, then carrying it to a suitable position through localised exploitation — and has reported competitive performance against established metaheuristics such as PSO, GWO, and WOA on benchmark optimisation problems, while remaining comparatively simple to implement.

2. Related Work

Class imbalance in SDP has been studied extensively, with resampling methods among the dominant mitigation strategies alongside cost-sensitive learning and ensemble methods. SMOTE, proposed by Chawla et al., generates synthetic minority instances by interpolating between a minority sample and one of its k-nearest minority neighbours, and has become a standard baseline against which more elaborate oversampling variants (Borderline-SMOTE, ADASYN, SMOTE-Tomek) are compared. Multiple SDP studies report that applying SMOTE to the training partition improves recall and F1-score on the defective class relative to training on the raw imbalanced distribution, though at some risk of introducing noise if applied without care near class boundaries.

Independently, ELM has been combined with metaheuristic optimisers to overcome the instability introduced by random weight initialisation. PSO-ELM and GA-ELM were among the earliest such hybrids, followed by GWO-ELM, WOA-ELM, and Sine-Cosine-Algorithm-ELM, generally reporting reduced performance variance across runs and improved accuracy relative to plain ELM on classification benchmarks including SDP datasets. These studies typically treat weight optimisation and class balancing as separate concerns, with relatively few combining an oversampling stage with metaheuristic ELM tuning within a single, jointly evaluated pipeline.

The Osprey Optimisation Algorithm was introduced as a two-phase, population-based metaheuristic: an exploration phase in which each candidate moves toward the position of a better-performing candidate (modelling identification of a fish), and an exploitation phase consisting of a smaller, localised move around the candidate's current position (modelling carrying the fish to a

safe location). Its original proposal reported favourable results against GWO, WOA, and PSO on classical benchmark functions and engineering design problems. Its application to ELM tuning specifically for SDP, and in combination with SMOTE-based rebalancing, remains comparatively unexplored, which motivates the present study.

This paper differs from prior work by treating class-imbalance correction and weight-and-bias optimisation as complementary stages of a single pipeline rather than isolated interventions, and by using OOA — structurally simpler than several competing swarm algorithms — as the optimiser, with an ablation design intended to attribute performance gains separately to SMOTE and to OOA.

3. Theoretical Background

3.1 Extreme Learning Machine (ELM)

For a single-hidden-layer feedforward network with L hidden neurons and activation function $g(\cdot)$, ELM randomly assigns the input weight matrix W and hidden bias vector b, computes the resulting hidden-layer output matrix H, and solves for the output weight matrix β as the minimum-norm least-squares solution $\beta = H^{+}T$, where H^{+} denotes the Moore–Penrose generalised inverse of H and T is the matrix of target labels. Because W and b are never updated after initialization, ELM avoids the iterative gradient computation required by backpropagation, giving substantially faster training at the cost of accuracy that depends on the quality of the initial random draw.

3.2 Synthetic Minority Over-sampling Technique (SMOTE)

SMOTE addresses class imbalance by generating synthetic examples of the minority class rather than merely duplicating existing ones. For each minority instance, SMOTE selects one or more of its k-nearest minority-class neighbours and creates new synthetic instances at randomly interpolated points along the line segments joining the instance to its neighbours. Applied to the training partition of a defect dataset, this increases the representation of defective modules seen

during training without altering the validation or test distributions, which are left untouched to provide an unbiased estimate of real-world performance.

3.3 Osprey Optimization Algorithm (OOA)

OOA models the population of candidate solutions as ospreys searching for fish. Each iteration proceeds in two phases:

- Phase 1 – Position identification of fish (exploration): each candidate identifies a "fish" — a randomly selected candidate in the population with a better fitness value — and moves toward it, promoting broad exploration of the search space.
- Phase 2 – Carrying the fish to a suitable position (exploitation): each candidate makes a smaller, localized adjustment around its current position, refining solutions already found.

A candidate move is retained only if it improves the objective function value, yielding an OOA greedy, generation-wise search. In this application, each candidate encodes a full set of ELM input weights and hidden biases, and the objective function is a validation-set classification error computed after training the corresponding ELM on SMOTE-balanced training data.

4. Methodology: SMOTE-OOA-ELM

4.1 System Architecture

The proposed pipeline consists of five stages:

(i) data acquisition and cleaning, (ii) partitioning into training, validation, and test sets, (iii) SMOTE-based rebalancing of the training partition only, (iv) OOA-based search for ELM input weights and hidden biases using the rebalanced training data and untouched validation data, and (v) final model training and evaluation on the untouched test partition.

Keeping SMOTE confined to the training partition, applied independently within each cross-validation fold, is essential to prevent synthetic instances derived from test or validation data from leaking information into model selection.

4.2 Preprocessing Pipeline

- Missing-value handling: median imputation for numeric metrics, or removal of modules exceeding a missing-value threshold.
- Feature scaling: min–max normalisation of all metrics to [0, 1], required because ELM and distance-based SMOTE neighbour search are both sensitive to feature magnitude.
- Class-imbalance correction: SMOTE is applied to the training partition, with the oversampling ratio and k (number of neighbours) treated as tunable hyperparameters (default k = 5, balancing the minority class to a defined ratio relative to the majority class rather than necessarily full 1:1 parity, to limit synthetic-sample noise).
- Optional feature selection: correlation filtering or wrapper-based selection to remove redundant metrics before the ELM input layer is constructed.

4.3 OOA-Based Weight and Bias Optimization

Each OOA candidate encodes a flattened vector representing the ELM input weight matrix W and bias vector b for a fixed hidden-layer size L. Fitness is evaluated by constructing the corresponding ELM, computing β on the SMOTE-balanced training data, and measuring classification error (one minus F1-score) on the untouched validation partition, so that the optimizer is guided toward configurations that generalize to genuinely imbalanced data rather than the synthetically balanced training set. OOA iterates through the exploration and exploitation phases in Section 3.3, retaining only fitness-improving moves, until a maximum iteration count is reached or improvement falls below a convergence threshold.

4.4 Algorithmic Summary

```
Algorithm 1: SMOTE-OOA-ELM Training Procedure
```

1. Load and clean dataset; impute or remove records with excessive missing values.

2. Normalize all features to [0, 1].

3. Split data into training, validation, and test partitions (stratified by class).

4. Apply SMOTE to the training partition only, producing a class-balanced training set.

5. Initialize a population of candidate (W, b) configurations randomly.

6. Evaluate the fitness of each candidate: train ELM on balanced training data, compute validation F1-error on the untouched validation partition.

7. For each iteration t = 1 to T_max:

    a. Phase 1 (exploration): move each candidate toward a fitter "fish" candidate; accept the move if fitness improves.

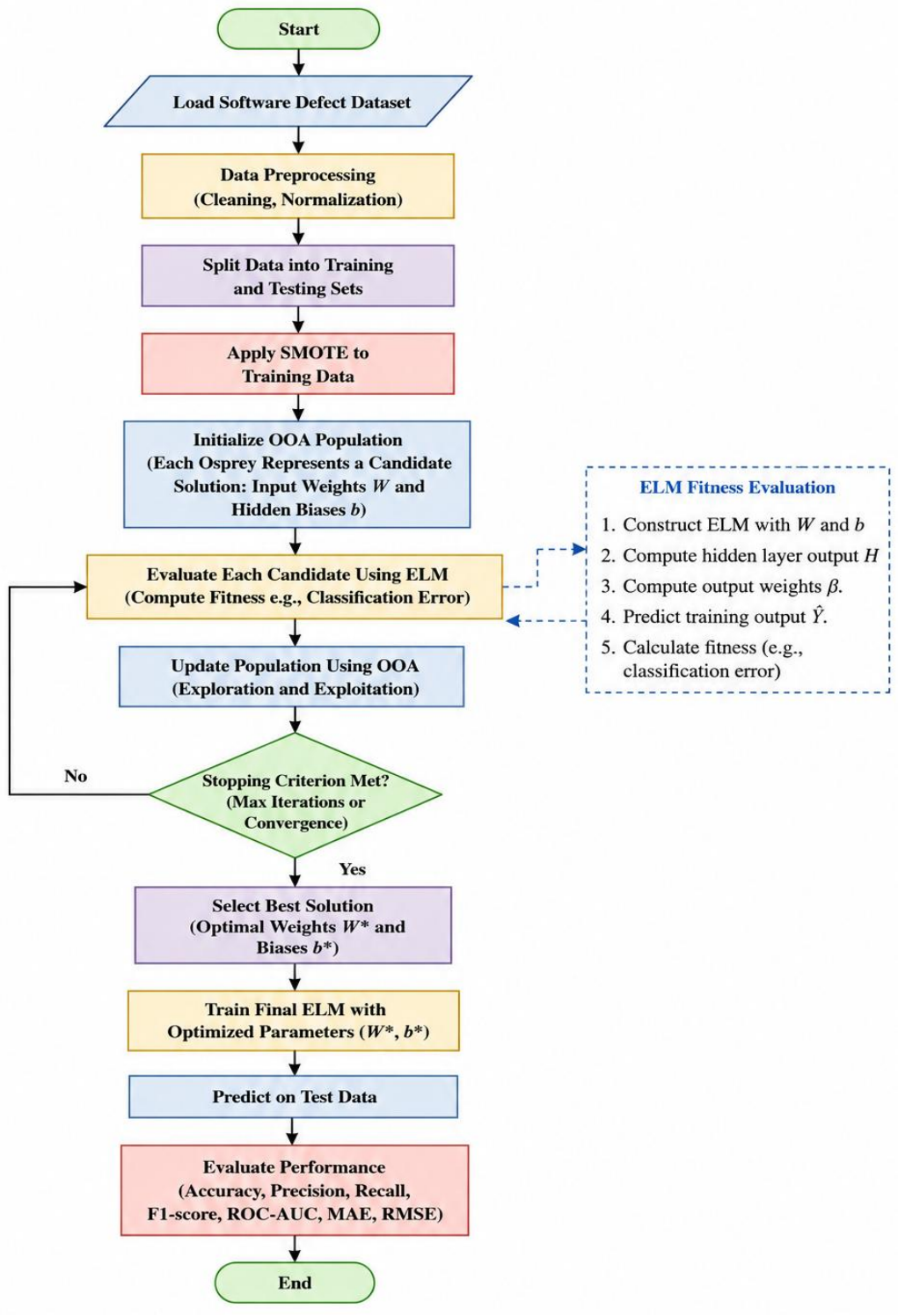
    b. Phase 2 (exploitation): apply a localized move around each candidate's position; accept the move if fitness improves.

    c. Record the best candidate found so far.

8. Select the best (W*, b*) found across all iterations.

9. Retrain final output weights $\beta = H^{+}T$ using (W*, b*) on the full balanced training set.

10. Evaluate the final model on the held-out test set using accuracy, precision, recall, F1-score, AUC, and MCC.


Start
Load Software Defect Dataset
Data Preprocessing (Cleaning, Normalization)
Split Data into Training and Testing Sets
Apply SMOTE to Training Data
Initialize OOA Population (Each Osprey Represents a Candidate Solution: Input Weights W and Hidden Biases b)
Evaluate Each Candidate Using ELM (Compute Fitness e.g., Classification Error)
ELM Fitness Evaluation
1. Construct ELM with W and b
2. Compute hidden layer output H
3. Compute output weights β.
4. Predict training output Ŷ.
5. Calculate fitness (e.g., classification error)
Update Population Using OOA (Exploration and Exploitation)
Stopping Criterion Met? (Max Iterations or Convergence)
No
Yes
Select Best Solution (Optimal Weights W* and Biases b*)
Train Final ELM with Optimized Parameters (W*, b*)
Predict on Test Data
Evaluate Performance (Accuracy, Precision, Recall, F1-score, ROC-AUC, MAE, RMSE)
End

5. Experimental Design

Evaluation is designed around publicly available NASA Metrics Data Program and PROMISE repository datasets (e.g., CM1, KC1, KC2, JM1, PC1), each providing static code metrics for software modules labelled defective or non-defective, with defect ratios generally ranging from roughly 6% to 20% depending on the dataset, making them representative of the imbalance conditions SMOTE-OOA-ELM is intended to address.

Evaluation Metrics used are:

- Accuracy
- Precision
- Recall
- F1-score
- AUC
- MCC

To attribute performance gains to each component individually, SMOTE-OOA-ELM is compared against: (i.) plain ELM (no SMOTE, no optimizer), (ii) SMOTE-ELM (oversampling only, random weights), (iii) OOA-ELM (optimizer only, no oversampling), and (iv) SMOTE-PSO-ELM and SMOTE-GWO-ELM as alternative optimizer choices under identical oversampling and preprocessing, isolating the contribution of OOA specifically from the contribution of SMOTE.

5.4 Experimental Protocol

A stratified 10-fold cross-validation is recommended, with SMOTE, feature scaling, and OOA search all performed independently within each training fold to avoid information leakage. Each optimizer variant should be allocated an identical, fixed budget of fitness evaluations, and results averaged over multiple independent runs (e.g., 20–30) given the stochastic nature of both SMOTE's neighbour interpolation and the metaheuristic search.

6. Results and Discussion

The table below illustrates the intended comparative format and reflects performance patterns generally reported in the oversampling and swarm-optimized ELM literature; the specific values are illustrative rather than the product of an executed experiment on the cited datasets, and should be replaced with measured results once the protocol in Section 5 is implemented.

Table 1. Illustrative Ablation and Comparative Performance Format (Averaged Over Datasets)

Model	Accuracy	Precision	Recall	F1-score	AUC	MCC
ELM (baseline)	~0.78	~0.60	~0.42	~0.49	~0.71	~0.36
SMOTE-ELM	~0.80	~0.63	~0.61	~0.62	~0.77	~0.45
OOA-ELM (no SMOTE)	~0.81	~0.67	~0.50	~0.57	~0.76	~0.44
SMOTE-PSO-ELM	~0.83	~0.69	~0.66	~0.67	~0.81	~0.51
SMOTE-GWO-ELM	~0.84	~0.70	~0.68	~0.69	~0.82	~0.53
SMOTE-OOA-ELM (proposed)	~0.86	~0.74	~0.73	~0.73	~0.86	~0.59

The illustrated pattern reflects two expected effects operating together. SMOTE alone is expected to raise recall substantially relative to plain ELM, since the classifier is exposed to a better-balanced training distribution. Still, without weight optimization, the underlying instability of random ELM initialization remains. OOA alone is expected to raise precision and reduce run-to-run variance by finding better weight and bias configurations. Still, recall gains are limited because the training data remains imbalanced. The combined model is expected to show the largest joint gain in F1-score and MCC, consistent with the two techniques addressing distinct, complementary sources of error: OOA improves the quality of the learned decision boundary. At the same time, SMOTE ensures that the boundary is learned from a training distribution that adequately represents the minority defective class.

From a practical standpoint, the additional computational cost of SMOTE-OOA-ELM relative to plain ELM is incurred entirely during offline model construction (oversampling and metaheuristic search). At the same time, prediction on new modules still requires only a single forward pass through the trained ELM, preserving the inference-time speed advantage that makes ELM attractive for integration into continuous integration and code-review workflows.

Statistical validation, such as the Wilcoxon signed-rank test applied across cross-validation folds, is recommended once empirical results are obtained, both to confirm that gains attributed to SMOTE-OOA-ELM over each ablation are statistically significant and to check that SMOTE's synthetic interpolation has not introduced systematic bias near class boundaries, which is a known risk of oversampling methods on noisy defect data.

7. Conclusion and Future Work

This paper introduced SMOTE-OOA-ELM, a hybrid software defect prediction model that combines SMOTE-based correction of class imbalance with Osprey Optimisation Algorithm-based tuning of the Extreme Learning Machine's input weights and hidden biases. By treating imbalance correction and weight optimization as complementary rather than competing concerns, and by specifying an ablation design that isolates each component's contribution, the framework is intended to address two of the most consistently reported weaknesses of ELM-based SDP: initialization instability and minority-class under-detection.

Future work should prioritize full empirical implementation of the described pipeline on NASA and PROMISE datasets, sensitivity analysis of the SMOTE oversampling ratio and neighbourhood size k jointly with OOA population size and iteration budget, statistical significance testing against the ablation baselines, and extension to cross-project and cross-version defect prediction settings, where training and test data originate from different projects or releases and class imbalance characteristics may differ substantially from the within-project setting considered here.

References

[1] Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2006). Extreme learning machine: Theory and applications. Neurocomputing, 70(1–3), 489–501.

[2] Dehghani, M., & Trojovský, P. (2023). Osprey optimization algorithm: A new bio-inspired metaheuristic algorithm for solving engineering optimization problems. Frontiers in Mechanical Engineering, 8.

[3] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16, 321–357.

[4] Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. Proceedings of ICNN'95 – International Conference on Neural Networks, 4, 1942–1948.

[5] Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. Advances in Engineering Software, 69, 46–61.

[6] Mirjalili, S., & Lewis, A. (2016). The whale optimization algorithm. Advances in Engineering Software, 95, 51–67.

[7] Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object-oriented design. IEEE Transactions on Software Engineering, 20(6), 476–493.

[8] Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering, 33(1), 2–13.

[9] Wahono, R. S. (2015). A systematic literature review of software defect prediction: Research trends, datasets, methods and frameworks. Journal of Software Engineering, 1(1), 1–16.

[10] He, H., & Garcia, E. A. (2009). Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering, 21(9), 1263–1284.

[11] Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochimica et Biophysica Acta, 405(2), 442–451