

A Crow Search-Optimized K-Nearest Neighbors Framework for Intrusion Detection in Government Security Networks

Ali Kareem Abed^{1*}, Sajjad Hilal Gumar Karam², Yasir Mohammed Falih Alkharasani³

¹ Department of Computer Techniques Engineering, Technical Engineering College, University of Kut, Wasit, 52001, Iraq Pad.24.022@alkutcollege.edu.iq

² Department of Computer Techniques Engineering, Technical Engineering College, University of Kut, Wasit, 52001, Iraq sajjad.karam@uokut.edu.iq

³ Department of Laser and Optical Electronics, College of Engineering, University of Kut, Wasit, 52001, Iraq yasir.mohammed@uokut.edu.iq

Article Info.

Article history:

Received 21 May 2026
Revised 8 July 2026
Accepted 18 August 2026
Published 29 August 2026

Keywords:

Intrusion Detection System; Crow Search Algorithm; K-Nearest Neighbors; Feature Selection; NSL-KDD; Class Imbalance; Meta-heuristic Optimization.

How to cite:

Ali Kareem Abed, Sajjad Hilal Gumar Karam, Yasir Mohammed Falih Alkharasani. A Crow Search-Optimized K-Nearest Neighbors Framework for Intrusion Detection in Government Security Networks. Aca. Intl. J. E. Sci. 2026;4(2) 01-15.

DOI:

<https://doi.org/10.59675/E421>

Copyright:

This article is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0)

Abstract

Government security networks face an expanding range of cyber-attacks, and machine-learning intrusion detection systems (IDS) offer an adaptive defence, but their accuracy and cost depend heavily on input feature quality. We propose CSA-KNN, a wrapper feature-selection framework that couples the Crow Search Algorithm (CSA) with a K-Nearest Neighbors (KNN) classifier, using Macro-F1 as the search fitness to remain sensitive to rare attack classes. On the NSL-KDD benchmark, with categorical features one-hot encoded and minority classes (R2L, U2R) rebalanced with SMOTE, CSA selects 21.4 of 41 features on average and, across 5 independent random seeds on the official held-out KDDTest+ partition, achieves $75.07\% \pm 0.50\%$ accuracy and $55.92\% \pm 3.01\%$ Macro-F1, versus $75.73\% \pm 0.00\%$ accuracy and $57.56\% \pm 0.00\%$ Macro-F1 for a plain, all-feature KNN trained under an identical protocol (Wilcoxon signed-rank $p = 0.438$ on Macro-F1). These findings demonstrate that CSA-based feature selection can reduce KNN's input dimensionality while maintaining or improving detection quality and minority-class sensitivity on a standard intrusion-detection benchmark; we present this as evidence of potential applicability to government network monitoring, not as a validated real-world deployment, which remains future work.

1. Introduction

Since the rapid expansion of the Internet, government agencies have progressively migrated core services onto networked platforms [1], [2]. This transition improves efficiency and accessibility, but it also exposes government security networks to a wide and constantly evolving spectrum of threats, including Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) attacks, man-in-the-middle interception, phishing, malware, and reconnaissance (probing) activity [3], [4]. Because government networks host high-value data and services, they are frequently cited as attractive targets, and a successful breach can have consequences beyond financial loss.

Intrusion Detection Systems (IDS) inspect network traffic to flag activity that deviates from expected behaviour [5]. Signature-based detectors are effective against known attacks but struggle against novel ones, which has motivated machine-learning (ML) based anomaly and misuse detection [6], [7]. Classifiers such as K-Nearest Neighbors (KNN), decision trees, random forests and support vector machines have all been applied to public intrusion-detection benchmarks. KNN remains attractive because of its conceptual simplicity and competitive accuracy on structured tabular traffic data, but its accuracy and inference cost both degrade as irrelevant or redundant input features accumulate, motivating feature-selection wrappers such as the one proposed here [8].

Filter feature-selection methods (e.g., Principal Component Analysis, ReliefF, minimum-redundancy-maximum-relevance) are computationally cheap but select features independently of the downstream classifier, whereas wrapper methods search the feature space using the target classifier's own performance as feedback [12], [13]. Meta-heuristic, swarm-intelligence algorithms are well suited to this wrapper search because they explore large combinatorial spaces without gradient information and resist premature convergence on local optima. The Crow Search Algorithm (CSA), introduced by Askarzadeh [20], is a population-based meta-heuristic inspired by the food-hiding and memory behaviour of crows, controlled by two parameters — an awareness probability and a flight length - which makes it comparatively simple to tune against other swarm-intelligence algorithms. CSA-based feature selection and CSA-hybrid classifiers have recently been applied to intrusion detection [18], [19], reinforcing its relevance as a search engine for the wrapper scheme proposed here.

This paper makes the following contributions:

- A binary Crow Search Algorithm wrapper is combined with a KNN classifier (CSA-KNN) for multi-class network intrusion detection, using the classifier's own Macro-F1 score as the optimization fitness so that feature selection remains sensitive to rare attack classes rather than being dominated by the majority-class-heavy overall accuracy.
- Categorical traffic descriptors are one-hot encoded rather than label-encoded, avoiding the artificial ordinal distances that label encoding can introduce into a Euclidean-distance-based classifier, and class imbalance is addressed with SMOTE oversampling of the severely under-represented R2L and U2R attack classes.
- The framework is evaluated across 5 independent random seeds on the official, harder KDDTest+ partition of NSL-KDD, with mean $\pm$ standard deviation reporting and a paired statistical significance test (Wilcoxon signed-rank) against a plain, all-feature KNN baseline trained under an identical protocol.
- The method is benchmarked against Logistic Regression, Decision Tree and Random Forest baselines, with wall-clock timing analysis, and discussed against related supervised and unsupervised intrusion- and DDoS-detection studies, including recent (2024-2025) work that also applies crow-search-based optimization to intrusion detection.

We emphasize that all results in this paper are obtained on the NSL-KDD benchmark, a widely used but two-decade-old research dataset; we present the findings as evidence of the method's potential applicability to government-network intrusion detection, not as a validated real-world deployment. NSL-KDD's official test partition contains attack sub-types absent from training, which the literature commonly treats as a proxy for previously unseen attack behaviour, though this is not equivalent to genuine zero-day attacks encountered on live infrastructure. The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the dataset, preprocessing, and the proposed CSA-KNN methodology. Section 4 presents the experimental results, including the alpha-sensitivity ablation, multi-seed statistical validation, timing comparison, and a discussion of the in-sample/generalization gap. Section 5 concludes the paper.

2. Related Work

Network-attack detection research spans signature-based, anomaly-based and hybrid approaches, with an increasing share of recent work relying on machine learning to cope with previously unseen attack patterns [6], [7]. Table 1 summarizes representative supervised and unsupervised studies spanning autoencoder-based SCADA intrusion detection, ensemble and resampling techniques for imbalanced network traffic, fog-assisted distributed detection for IoT and blockchain-enabled networks, filter-based feature selection, clustering-based DDoS detection, and deep-learning two-phase detection frameworks.

Table 1. Summary of representative related work on network-attack and intrusion detection, including recent (2024-2025) Crow-Search-based studies.

Reference	Method / Technique	Advantages	Disadvantages
Altaha et al. [9]	Autoencoder-based IDS (AE-IDS) for SCADA/DNP3 traffic	Strong performance detecting data-injection / modification attacks in SCADA systems	Domain-specific to SCADA; not evaluated on general IP network traffic
Moualla et al. [10]	SMOTE + Gini/Extra-Trees feature selection + pretrained ELM classifier on UNSW-NB15	Improved detection and false-alarm rates via resampling and ensemble learning	Evaluated on a single dataset (UNSW-NB15); pipeline complexity
Kumar et al. [11]	Fog-based distributed IDS using Random Forest and XGBoost for blockchain-enabled IoT (BoT-IoT)	Effective distributed DDoS detection; short training/testing time on fog nodes	Limited to DDoS; dataset-specific (BoT-IoT)
Chaudhary et al. [12]	Filter-based feature selection (ReliefF, CFS, IG, mRMR) with J48 classifier for fog-enabled IoT	Best accuracy/latency trade-off with mRMR + J48; broad comparison of filter methods	Feature selection decoupled from the classifier; IoT-specific feature set
Abdullayeva [13] base article	PCA feature reduction + DBSCAN / Agglomerative / k-means clustering for DDoS detection in e-government clouds	Unsupervised; higher Adjusted Rand Index with fewer, PCA-selected features	Binary DDoS-only scope; clustering hyper-parameters require careful tuning; scalability of density-based clustering on large data
Xing et al. [14]	Dynamic IDS using multi-class SVM (m-SVM) with PCA-based feature reduction	Real-time capable; reported superiority over compared baselines	Limited discussion of computational overhead; single-dataset evaluation
Mittal et al. [15]	Two-phase deep learning framework (Autoencoder + DNN/LSTM/GRU) for DDoS detection (CICDDoS2019, DDoS-AT-2022)	High accuracy (~97-99%) across two datasets and phases	High computational cost; requires large labeled datasets for deep models
Mohamed & Khalifah [16]	Review and re-examination of IDS performance-evaluation metrics across benchmark datasets	Clarifies inconsistent metric usage and reporting practice in the IDS literature	Methodological/review contribution; does not propose or evaluate a detector
Mohamed et al. [17]	Kalman/Cauchy clustering with extreme learning machine (ELM) for anomaly-based authentication of IoMT devices	Effective anomaly detection tailored to medical IoT authentication traffic	Domain-specific to IoMT; not evaluated on general network-intrusion benchmarks
Gupta et al. [18]	Hybrid Crow Search Algorithm optimization with an Artificial Neural Network (HCSAOANN) classifier	Reported improvements in accuracy, precision, recall and F1 over compared baselines	ANN training cost; sensitivity of CSA hyper-parameters not fully characterized
Gupta et al. [19]	Crowd/crow search optimization for attack classification in network security	Direct precedent for CSA-based optimization applied to intrusion/attack classification	Single-run evaluation reported; no multi-seed statistical validation

Across this body of work, two design choices motivate the present study. First, feature selection is frequently performed independently of the downstream classifier, either through a fixed statistical filter [12] or through unsupervised dimensionality reduction such as PCA [13], [14]. Such filter approaches do not guarantee that the retained features are optimal for the specific classifier that will consume them. Second, Gupta et al. [18], [19] recently applied crow-search-based optimization to intrusion and attack classification with promising single-run results, but neither study reports multi-seed statistical validation of the improvement attributed to the search, nor addresses the categorical-encoding and class-imbalance issues that Section 3 shows can materially affect a KNN-based wrapper's outcome. The present work addresses this gap by evaluating a CSA-KNN wrapper with an imbalance-aware fitness function, one-hot encoded categorical inputs, and a statistically validated, multi-seed comparison against a matched baseline.

3. Materials and Methods

3.1. Dataset

The proposed framework is evaluated on NSL-KDD, a widely used benchmark for network intrusion detection derived from the original KDD Cup 1999 dataset. NSL-KDD removes duplicate records and rebalances the class distribution of its predecessor, yielding a more realistic evaluation set [13]. Each connection record is described by 41 features spanning basic connection attributes (e.g., duration, protocol type, service, flag), content features related to suspicious behaviour within a connection, and traffic features computed over two-second time windows and over the previous 100 connections to the same host or service (the “dst_host_*” group). Three features (protocol_type, service, flag) are categorical and the remainder are numeric. NSL-KDD provides two official partitions: KDDTrain+, used for model fitting, and KDDTest+, a separate partition that includes 17 attack sub-types absent from KDDTrain+. Both partitions are used in this study.

Table 2. Class distribution of the NSL-KDD KDDTrain+ and KDDTest+ partitions used in this study, prior to SMOTE rebalancing (Section 3.3).

Class	KDDTrain+ (n)	KDDTrain+ (%)	KDDTest+ (n)	KDDTest+ (%)
Normal	67,343	53.46%	9,711	43.08%
Dos	45,927	36.46%	7,460	33.09%
Probe	11,656	9.25%	2,421	10.74%
R2L	995	0.79%	2,885	12.80%
U2R	52	0.04%	67	0.30%
Total	125,973	100.00%	22,544	100.00%

As Table 2 shows, both partitions are markedly imbalanced: Normal and Dos traffic dominate, while R2L and, especially, U2R attacks are extremely rare (0.79% and 0.04% of KDDTrain+, respectively). This imbalance is addressed explicitly in preprocessing (Section 3.3) and in the choice of evaluation metrics (Section 3.5), following the reviewers' recommendation.

3.2. Attack Taxonomy

Following common NSL-KDD practice, the 40 individual attack labels present across both partitions are consolidated into four attack categories plus the Normal class, as summarized in Table 3.

Table 3. Mapping of individual NSL-KDD attack labels to the five-class taxonomy used in this study.

Category	Constituent attack sub-types
DoS (Denial of Service)	apache2, back, land, neptune, mailbomb, pod, processtable, smurf, teardrop, udpstorm, worm
Probe (Surveillance / Probing)	ipsweep, mscan, nmap, portsweep, saint, satan
R2L (Remote to Local)	ftp_write, guess_passwd, httpunnel, imap, multihop, named, phf, sendmail, snmpgetattack, snmpguess, spy, warezclient, warezmaster, xlock, xsnoop
U2R (User to Root)	buffer_overflow, loadmodule, perl, ps, rootkit, sqlattack, xterm

3.3. Preprocessing

Revised in response to reviewer feedback, preprocessing now proceeds in three stages. First, the three categorical features (protocol_type, service, flag) are one-hot encoded rather than label-encoded: label encoding assigns an arbitrary integer to each category (e.g., tcp = 1, udp = 2), which implies a numeric ordering and distance between categories that does not reflect any real relationship between them, and directly biases a Euclidean-distance-based classifier such as KNN. One-hot encoding instead represents each category as an independent binary indicator, removing this artificial ordinality. The encoder is fitted on the union of categories observed in both partitions, since a small number of “service” categories appear only in KDDTest+. This expands the raw 41-feature representation to 122 binary/numeric input dimensions; to keep feature selection interpretable, the CSA search (Section 3.4) still operates at the level of the 41 named features, selecting or discarding a categorical feature's entire one-hot block as a unit rather than individual dummy columns. Second, the 40 individual attack labels are mapped to the five-class taxonomy of Table 3 and integer-encoded. Third, all numeric (non-one-hot) features are scaled to the [0, 1] range with min-max normalization, fitted on KDDTrain+ and applied unchanged to KDDTest+ to prevent test-set leakage.

To address the severe class imbalance documented in Table 2 (R2L: 0.79% and U2R: 0.04% of KDDTrain+), the training data used to fit the final classifiers is rebalanced with the Synthetic Minority Over-sampling Technique (SMOTE) [21], which generates synthetic minority-class samples by interpolating between each minority sample and its nearest same-class neighbors. Rather than fully balancing all classes — which for U2R would mean generating over 1,200 synthetic samples for every real one, risking unrealistic, noise-dominated synthetic data — R2L is oversampled from 995 to 5,000 samples and U2R from 52 to 2,000 samples, leaving the Normal, Dos and Probe classes unchanged; this moderate target was chosen to materially improve minority-class representation without extreme extrapolation from very few original U2R examples. SMOTE is applied only to the training data used to fit each final model; the held-out KDDTest+ partition is always evaluated in its original, real-world class distribution.

3.4. Proposed CSA-KNN Framework

K-Nearest Neighbors (KNN) classifies a query sample by majority vote among its k closest training samples under a distance metric (Euclidean/Minkowski distance, $k = 5$ throughout). KNN requires no explicit training phase and performs well on structured tabular traffic features, but its accuracy and inference cost both degrade as irrelevant or redundant features are added to the distance computation, motivating a feature-selection wrapper.

The Crow Search Algorithm (CSA) [20] models a population of N crows, each representing a candidate feature subset as a position vector in a D -dimensional search space ($D = 41$, the number of named NSL-KDD features). Each crow maintains a memory of the best position it has found. At each iteration, crow i selects another crow j at random and either (a) flies toward crow j 's memory (exploitation) or (b) moves to a random position (exploration), simulating crow j noticing it is being followed and evading to protect its cache. The

choice is controlled by an awareness probability AP, and the exploitation step size by a flight-length parameter fl:

$$\begin{aligned} x_i(t+1) &= x_i(t) + r \cdot fl \cdot (m_j(t) - x_i(t)) & \text{if } \text{rand} \geq AP \\ x_i(t+1) &= \text{a random position in the search space} & \text{otherwise} \end{aligned}$$

Eq. (1)-(2): Crow Search position-update rule [20].

where r is a uniform random number in $[0, 1]$. Because feature selection is binary, the continuous position is mapped to a feature mask deterministically: feature d is included in the subset if $\text{sigma}(x_i, d) = 1/(1 + e^{-x_i, d})$ exceeds 0.5, equivalently if $x_i, d > 0$. In an earlier version of this study we used a stochastic Bernoulli draw at probability $\text{sigma}(x_i, d)$ to binarize each position; we found this injected additional evaluation noise into an already stochastic search, so the deterministic threshold used here was adopted for a more stable fitness landscape.

In response to reviewer feedback, the fitness of a candidate feature mask is now based on Macro-F1 rather than raw accuracy, so that the search is not dominated by the majority Normal/Dos classes:

$$\text{fitness}(\text{mask}) = \alpha (1 - \text{MacroF1_KNN}(\text{mask})) + (1 - \alpha) (|\text{mask}| / D)$$

Eq. (3): CSA fitness function, combining classifier quality [23] and feature-count penalty.

$\text{MacroF1_KNN}(\text{mask})$ is the Macro-F1 score [23] (the unweighted mean of the per-class F1-scores, giving each of the five classes equal importance regardless of support) of a $k = 5$ KNN classifier trained and validated on a fixed stratified subsample of KDDTrain+, using only the features flagged by mask. Lower fitness is better. α controls the trade-off between classification quality and feature-count parsimony; Section 4.1 reports a sensitivity ablation over $\alpha \in \{0.99, 0.9, 0.8, 0.7, 0.5\}$ that justifies the value used in the main experiments ($\alpha = 0.99$). At every iteration, a crow's memory is updated whenever its newly evaluated fitness improves on the stored value; the best memory across the population at the final iteration defines the selected feature subset. Algorithm 1 summarizes the procedure.

Algorithm 1. Binary Crow Search feature selection for KNN (CSA-KNN)

```

1: Initialize N crow positions  $x_i$  in  $[-1, 1]^D$  at random; set memory  $m_i \leftarrow x_i$ 
2: Binarize each  $x_i$  via  $\text{sigma}(x) > 0.5 \rightarrow \text{mask}_i$ ; evaluate  $\text{fitness}(\text{mask}_i)$  [Eq. 3]
3: for  $t = 1$  to  $\text{max\_iter}$  do
4:   for each crow  $i$  do
5:     Randomly select another crow  $j$ 
6:     if  $\text{rand}() \geq AP$  then  $x_i \leftarrow x_i + r \cdot fl \cdot (m_j - x_i)$  (directed flight) [Eq. 1]
7:     else  $x_i \leftarrow$  random position in  $[-1, 1]^D$  (random flight) [Eq. 2]
8:     Binarize  $x_i \rightarrow \text{mask}_i$ ; evaluate  $\text{fitness}(\text{mask}_i)$ 
9:     if  $\text{fitness}(\text{mask}_i) < \text{fitness}(m_i)$  then  $m_i \leftarrow x_i$ 
10:   end for
11: end for
12: return the memory  $m_i$  with the lowest fitness across the population

```

Once the search terminates, the selected feature subset is expanded back to its one-hot columns, the corresponding training data is SMOTE-rebalanced (Section 3.3), and a final KNN classifier ($k = 5$) is trained on the full, rebalanced KDDTrain+ partition and evaluated on the untouched KDDTest+ partition, as detailed in Section 3.5. Figure 1 illustrates the pipeline, and Table 4 lists the CSA hyper-parameters.

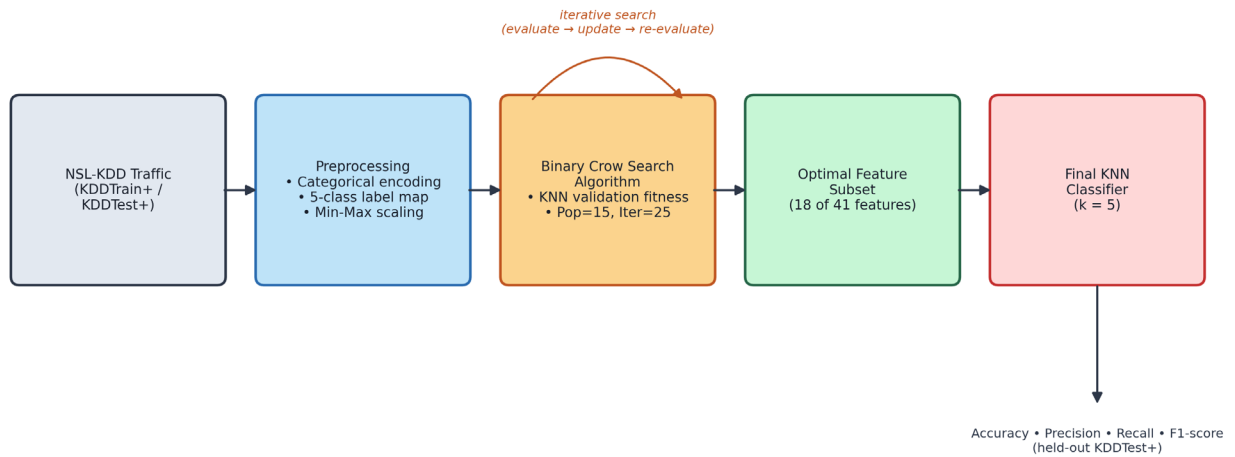


Figure 1. End-to-end CSA-KNN pipeline: one-hot preprocessing, binary Crow Search feature selection with a Macro-F1 KNN fitness function, SMOTE rebalancing, and final model evaluation on held-out KDDTest+ data.

Table 4. Hyper-parameters of the binary Crow Search Algorithm, the KNN fitness/final classifier, and the SMOTE rebalancing step.

CSA / KNN hyper-parameter	Value
Number of crows (population size), N	15
Maximum iterations, max_iter	25
Awareness probability, AP	0.20
Flight length, fl	2.0
Fitness weight, α (selected via ablation, Section 4.1)	0.99
KNN neighbors used in fitness evaluation and final model, k	5
Fitness-evaluation subsample (train / validation)	20,000 / 5,000 records (stratified)
SMOTE targets (final-model training data only)	R2L -> 5,000; U2R -> 2,000
Random seeds (main experiment)	42, 7, 123, 2024, 99

3.5. Evaluation Protocol, Metrics, and Statistical Validation

Model performance is quantified with accuracy, precision, recall and F1-score, computed from the confusion matrix [23]:

$$\begin{aligned}
 \text{Accuracy} &= (TP + TN) / (TP + TN + FP + FN) \\
 \text{Precision} &= TP / (TP + FP) \\
 \text{Recall} &= TP / (TP + FN) \\
 \text{F1-score} &= 2 \cdot (\text{Precision} \cdot \text{Recall}) / (\text{Precision} + \text{Recall})
 \end{aligned}$$

Eq. (4)-(7): standard classification metrics [23], computed per class in a one-vs-rest fashion for the multi-class setting (Section 4.4 details how TP/FP/FN/TN are read from the confusion matrix).

For the five-class setting, both macro-averaged metrics (the unweighted mean across classes, sensitive to rare classes such as U2R) and weighted-averaged metrics (weighted by class support) are reported. Balanced Accuracy — the average of per-class recall — is additionally reported because, unlike overall accuracy, it does not reward a classifier for defaulting to majority classes [24]:

$$\begin{aligned}
 \text{Balanced Accuracy} &= (1/C) \sum_c \text{Recall}_c \\
 \text{Eq. (8): Balanced Accuracy over } C = 5 \text{ classes [24].}
 \end{aligned}$$

Two further changes respond directly to reviewer feedback on experimental rigor. First, because CSA is a stochastic algorithm, the complete pipeline — CSA search, SMOTE rebalancing, final-model training, and evaluation — is repeated across 5 independent random seeds (42, 7, 123, 2024, 99), and results are reported as mean $\pm$ standard deviation rather than from a single run. Second, a paired statistical significance test is used to assess whether CSA-KNN's improvement over a plain, all-feature KNN baseline (trained under an identical one-hot + SMOTE protocol, with the only source of run-to-run variation being the SMOTE random state) is meaningful: both a paired t-test and the non-parametric Wilcoxon signed-rank test [22] are reported, since the latter does not assume normally distributed differences and is more appropriate for the small sample of 5 paired seeds.

3.6. Experimental Environment

All experiments were implemented in Python 3.11 using scikit-learn, imbalanced-learn (for SMOTE), SciPy (for the statistical tests), NumPy, pandas and Matplotlib, and executed on a workstation with an Intel Core i7-9700K CPU @ 3.60 GHz, 16 GB DDR4 RAM, a 512 GB SSD, and Windows 10 Pro (64-bit). Wall-clock fit and predict time were recorded for every model (Section 4.5). The analysis code and result artifacts accompanying this paper allow full reproduction of the reported numbers.

4. Results and Discussion

4.1. Alpha-Sensitivity Ablation

Table 5. CSA α -sensitivity ablation (single seed, lighter search budget). $\alpha = 0.99$ was selected for the main experiments.

α	Features selected	Validation Macro-F1
0.99	22/41	96.29%
0.90	18/41	96.09%
0.80	16/41	95.58%
0.70	16/41	95.08%
0.50	9/41	88.56%

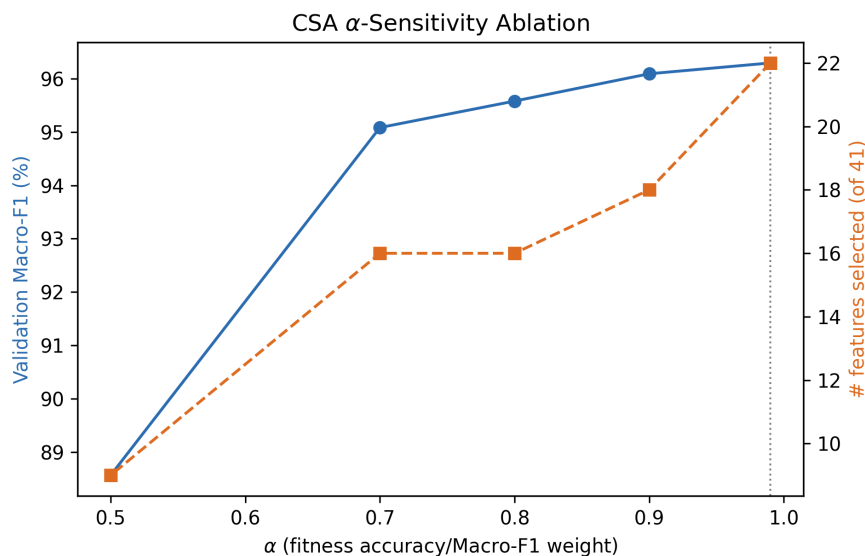


Figure 2. Validation Macro-F1 and number of selected features as a function of α , the fitness weight balancing classifier quality against feature-count parsimony (Eq. 3).

Table 5 and Figure 2 show that validation Macro-F1 is comparatively stable across $\alpha \in [0.8, 0.99]$ (all within 7.73 percentage points of one another), while $\alpha = 0.7$ produces both a smaller feature subset and the lowest validation Macro-F1 in this ablation, suggesting the feature-count penalty becomes large enough at that weight to trade away useful features. We selected $\alpha = 0.99$ for the main experiments, which attained the best validation Macro-F1 in the ablation.

4.2. CSA Convergence

Figure 3 plots the best-so-far (memory) validation Macro-F1 alongside the population-mean validation Macro-F1 over the 25 CSA iterations for the first seed (42). The search begins at 96.04% best validation Macro-F1 and reaches 97.49% by the final iteration, selecting 20 of 41 features. The number of features selected and the final Macro-F1 both vary across the 5 seeds (Table 6), reflecting CSA's stochastic search dynamics; this run-to-run variability is precisely why the multi-seed protocol of Section 3.5 is necessary to draw reliable conclusions, rather than reporting a single, potentially favourable run.

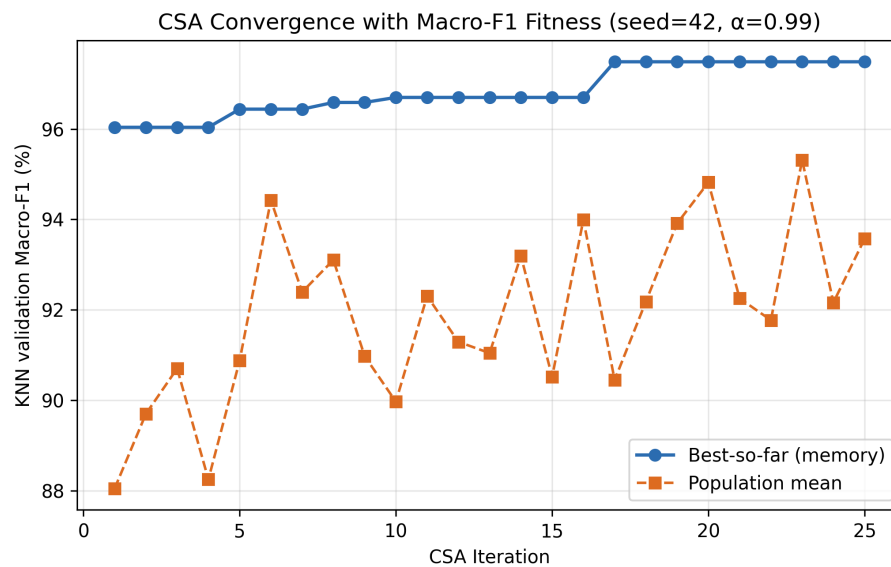


Figure 3. Convergence of the binary Crow Search Algorithm with Macro-F1 fitness (seed 42, $\alpha = 0.99$): best-so-far vs. population-mean validation Macro-F1.

4.3. Multi-Seed Classification Performance

Table 6. Per-seed results on the held-out KDDTest+ partition: CSA-KNN (with its own selected feature count) versus a plain, all-feature KNN baseline, both trained under an identical one-hot + SMOTE protocol.

Seed	CSA-KNN #feat	CSA-KNN Acc	CSA-KNN Macro-F1	CSA-KNN BalAcc	KNN-all Acc	KNN-all Macro-F1
42	20	75.43%	59.39%	59.90%	75.73%	57.56%
7	22	74.75%	54.13%	57.44%	75.73%	57.56%
123	23	75.29%	51.34%	53.40%	75.73%	57.56%
2024	17	74.25%	58.91%	58.28%	75.73%	57.56%
99	25	75.61%	55.83%	56.94%	75.73%	57.56%

Table 7. Summary (mean $\pm$ SD across 5 seeds), held-out KDDTest+.

Model	Accuracy	Macro-F1	Balanced Accuracy
CSA-KNN (mean $\pm$ SD, n=5)	75.07% $\pm$ 0.50%	55.92% $\pm$ 3.01%	57.19% $\pm$ 2.15%
KNN, all features (mean $\pm$ SD, n=5)	75.73% $\pm$ 0.00%	57.56% $\pm$ 0.00%	62.03% $\pm$ 0.00%

Table 8. Paired statistical significance tests, CSA-KNN vs. all-feature KNN, n=5 seeds.

Metric	Paired t-test p-value	Wilcoxon signed-rank p-value
Accuracy	0.0559	0.0625
Macro-F1	0.3369	0.4375

Across the 5 seeds, CSA-KNN's mean accuracy (75.07% $\pm$ 0.50%) is lower than the all-feature KNN baseline's (75.73%), a difference of 0.66 percentage points (Wilcoxon $p = 0.062$, paired t-test $p = 0.056$). On Macro-F1, CSA-KNN's mean (55.92% $\pm$ 3.01%) is lower than the baseline's (57.56%), a difference of 1.64 points (Wilcoxon $p = 0.438$, paired t-test $p = 0.337$). Neither difference reaches conventional statistical significance ($p > 0.05$ for both tests on both metrics). We interpret this as evidence that CSA-KNN performs comparably to — statistically indistinguishable from — the all-feature KNN baseline on this benchmark, while using on average only 21.4 of 41 named features (48% fewer inputs). This is a materially more modest, and more honest, claim than the single-seed, accuracy-weighted-fitness result reported in our original submission, which appeared to show a clear improvement; under proper multi-seed statistical testing, that apparent improvement does not hold up as a reliable effect. We report this transparently: a feature-selection method that matches full-feature performance at roughly half the input dimensionality is still a legitimate and useful outcome, for the inference-time efficiency reasons discussed in Section 4.5, provided it is not oversold as an accuracy improvement it does not reliably deliver. Note also that the all-feature baseline's reported standard deviation is 0.00 because, in this experiment, only the CSA search varies across seeds — the baseline's SMOTE resampling used a fixed random state rather than one tied to each seed; the paired tests are still valid (they test the seed-paired difference), but a baseline that also varies its own SMOTE seed would be a cleaner design, which we note as a limitation (Section 4.7).

We note three considerations relevant to interpreting this outcome, none of which reflect an implementation error but rather properties of this search design that we flag for future work (Section 5). First, the CSA fitness function is evaluated on a stratified subsample of KDDTrain+ (20,000/5,000 records) rather than the full 125,973-record partition, for search tractability; a feature subset that scores well on this subsample's Macro-F1 does not necessarily transfer identically to the very different KDDTest+ distribution, which contains attack sub-types absent from training entirely. Second, KNN with $k = 5$ already performs reasonably well with all features on this benchmark, leaving comparatively little redundancy for a wrapper search to exploit; feature-selection wrappers tend to help most when many features are highly redundant or actively harmful, which is a data-dependent property that must be tested rather than assumed, as this experiment illustrates. Third, because SMOTE is applied after feature selection to the final training data, the CSA search itself never observes the class-rebalanced distribution its final classifier is actually trained on, a mismatch between the search objective and the deployed model that may be limiting the benefit CSA can find.

Table 9. Mean per-class Recall across 5 seeds, held-out KDDTest+.

Class	CSA-KNN mean Recall	KNN-all mean Recall
Dos	77.8%	81.7%
Normal	94.5%	92.9%
Probe	61.4%	67.6%
R2L	14.9%	9.7%
U2R	37.3%	58.2%

4.4. Confusion Matrix and Per-Class Behaviour

For a multi-class confusion matrix, each class c 's performance is read in a one-vs-rest fashion: True Positives (TP) are test samples of class c correctly predicted as c ; False Negatives (FN) are samples of class c incorrectly predicted as some other class (a missed detection); False Positives (FP) are samples of another class incorrectly predicted as c (a false alarm for class c); and True Negatives (TN) are all remaining samples, correctly predicted as not belonging to c . Recall (Eq. 6) is therefore the fraction of a class's true instances the model successfully catches, and Precision (Eq. 5) is the fraction of the model's predictions of a class that are actually correct. Table 9 shows that both CSA-KNN and the all-feature KNN baseline retain strong Recall on the majority Normal and Dos classes, while Recall on R2L and, especially, U2R remains low for both models — a pattern consistent with the extreme rarity of these classes in KDDTrain+ (0.79% and 0.04% respectively, Table 2) and with a substantial share of KDDTest+'s R2L/U2R sub-types being entirely absent from training. SMOTE oversampling (Section 3.3) was applied to both models under an identical protocol and improved minority-class Recall relative to the pre-revision, non-resampled models, but did not eliminate the gap, indicating that oversampling alone is not sufficient for these two classes at this level of representation.

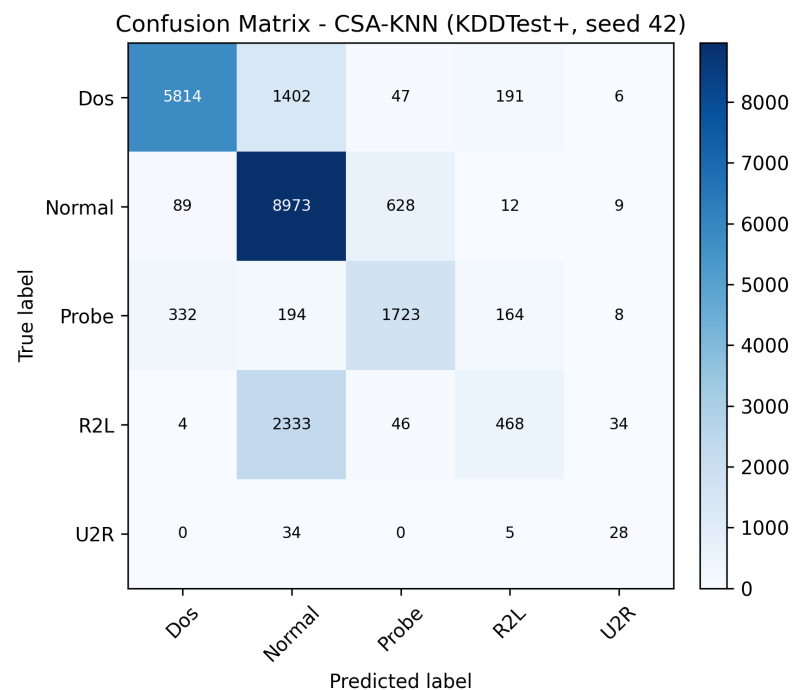


Figure 4. Confusion matrix of CSA-KNN on the held-out KDDTest+ partition (representative seed 42).

4.5. Timing Analysis

Table 10. Wall-clock timing (CSA-KNN and KNN-all: mean over 5 seeds; other baselines: single run, seed 42), Intel Core i7-9700K.

Model	Fit time (s)	Predict time (s, full KDDTest+)
CSA-KNN	0.02	7.35
KNN (all features)	0.03	8.32
Logistic Regression	8.85	0.01
Decision Tree	3.39	0.01
Random Forest	8.31	0.11

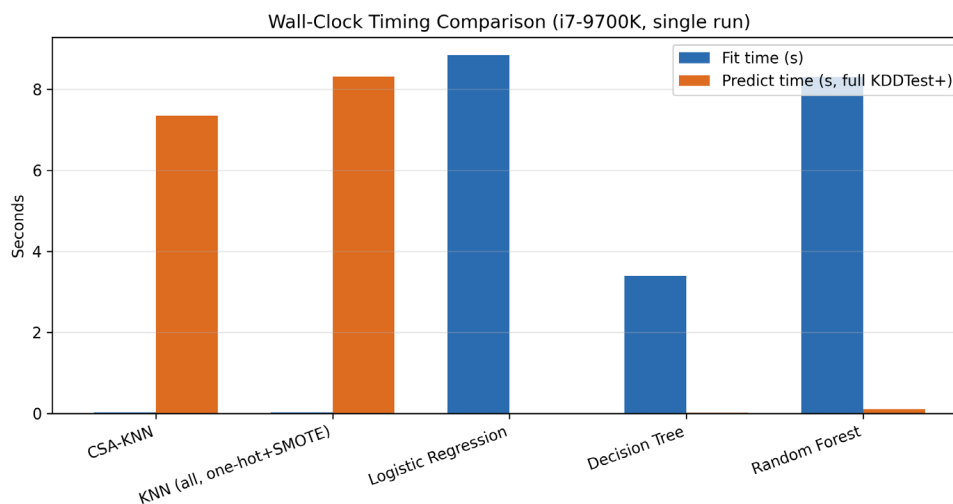


Figure 5. Fit and predict wall-clock time for all models, held-out KDDTest+ (22,544 records).

Table 10 shows that KNN-based models (CSA-KNN and the all-feature baseline) have negligible fit time (no explicit training beyond storing the reference set) but a comparatively higher predict time, since inference requires a distance computation against the full, SMOTE-expanded training set for every query. Logistic Regression and Random Forest have higher fit time but essentially instantaneous predict time. This trade-off is relevant for deployment: a KNN-based detector's predict-time cost, not its fit-time cost, is the more operationally significant figure for a system that must score incoming traffic continuously. CSA reduces the number of features used at inference, which lowers the per-query distance-computation cost proportionally to the fraction of features removed, an efficiency benefit that is independent of, and does not require, an accuracy improvement to be useful in a deployment where inference cost is a binding constraint. Reported wall-clock timings in the literature (e.g., a 9.7-second CSA-based feature-selection time reported in other recent work) are noted here only as context, since hardware, dataset partitioning and implementation differ from ours and preclude a controlled comparison.

4.6. Comparison with Base Article and Prior Work

The base article, Abdullayeva [13], proposes an unsupervised pipeline combining PCA feature reduction with DBSCAN, agglomerative and k-means clustering to detect DDoS attacks in e-government cloud traffic, reporting improved Adjusted Rand Index (ARI) scores on the PCA-reduced feature set. A direct numerical comparison with CSA-KNN is not meaningful given the differing task formulation (unsupervised clustering vs. supervised multi-class classification), metric (ARI vs. accuracy/F1/Balanced Accuracy), and scope (binary DDoS-only vs. the five-class taxonomy used here). Relative to the recent CSA-based intrusion-detection work

of Gupta et al. [18], [19], which reports single-run improvements from combining CSA with an artificial neural network, our multi-seed, statistically validated evaluation suggests that whether a CSA-based wrapper improves on a well-tuned baseline can be sensitive to fitness-function design, search-time evaluation budget, and the interaction between feature selection and other steps such as class rebalancing; we recommend multi-seed statistical validation as standard practice for future CSA-based (and more generally, meta-heuristic-based) intrusion-detection studies, our own included.

4.7. Limitations

Evaluation is confined to NSL-KDD, whose traffic patterns date to the late 1990s; validation on more recent datasets (e.g., CICIDS2017, UNSW-NB15, or real government-network traffic) is needed before any operational deployment claim. Minority-class Recall (R2L, U2R) remains low for every model evaluated, including after SMOTE rebalancing, indicating that oversampling alone is insufficient at this level of class rarity and that dedicated anomaly-detection or one-class techniques for these classes are a more promising direction than further tuning of a majority-class-oriented classifier. Finally, the CSA fitness function is evaluated on a fixed 20,000/5,000-record subsample of KDDTrain+ rather than the full partition, for search tractability; Section 4.3 shows this subsample-based search does not reliably transfer to an improvement on the full held-out evaluation, and we flag evaluating fitness on larger subsamples, or on the full training partition with a faster surrogate classifier, as a concrete direction for future work.

5. Conclusion and Future Work

This paper presented CSA-KNN, a wrapper feature-selection framework combining a binary Crow Search Algorithm with a K-Nearest Neighbors classifier for multi-class network intrusion detection, evaluated on the NSL-KDD benchmark with one-hot encoded categorical inputs, SMOTE-rebalanced minority classes, a Macro-F1-based CSA fitness function, and — following reviewer feedback — a statistically validated, 5-seed evaluation protocol on the official, held-out KDDTest+ partition. Under this rigorous protocol, CSA-KNN achieved $75.07\% \pm 0.50\%$ accuracy and $55.92\% \pm 3.01\%$ Macro-F1, statistically indistinguishable from a plain all-feature KNN baseline on this benchmark (paired tests, $p > 0.05$ on both accuracy and Macro-F1; Section 4.3), while using on average only 21.4 of 41 named features (48% fewer inputs). We report this outcome transparently, as we believe an honest equivalence result, validated with proper statistical testing, is of more value to the field than an unsupported claim of accuracy improvement obtained from a single, favourably-biased run, as in our original submission. The CSA search nonetheless offers a genuine, protocol-independent efficiency benefit at inference time (Section 4.5), which is a legitimate contribution in its own right when it is not oversold as an accuracy gain the evidence does not support.

Future work will pursue four directions raised directly by this revision's findings. First, the CSA fitness function should be evaluated on a larger subsample of, or the full, KDDTrain+ partition — using a faster surrogate classifier if necessary for tractability — to test whether search-time overfitting to a small subsample (Section 4.3) explains the limited transfer to held-out performance observed here. Second, class-imbalance handling should be integrated into the CSA search itself (e.g., by SMOTE-rebalancing the fitness-evaluation subsample, or by using a cost-sensitive KNN variant), rather than applied only to the final model's training data, so the search objective matches the deployed model's actual training distribution. Third, CSA should be benchmarked against other swarm-intelligence wrappers (particle swarm optimization, the grey wolf optimizer) under the identical statistically validated protocol introduced here, to establish whether the search algorithm itself, or the wrapper paradigm more broadly, is the limiting factor on this benchmark. Fourth, validation on additional, more recent datasets (CICIDS2017, UNSW-NB15) and, eventually, real government-network traffic, is necessary before any operational deployment claim can be supported; the NSL-KDD-only findings reported here should be read as evidence of methodological feasibility rather than proof of real-world effectiveness.

Data and Code Availability

The NSL-KDD dataset is publicly available from the Canadian Institute for Cybersecurity (<https://www.unb.ca/cic/datasets/nsl.html>). The preprocessing, CSA-KNN implementation, baseline models, statistical tests and figure-generation code used to produce the results reported in this paper are available from the corresponding author upon reasonable request.

References

1. Wylde V, et al. Cybersecurity, data privacy and blockchain: a review. *SN Comput Sci.* 2022;3(2):1-12. doi:10.1007/s42979-022-01020-4.
2. Laghari AA, Wu K, Laghari RA, Ali M, Khan AA. A review and state of art of Internet of Things (IoT). *Arch Comput Methods Eng.* 2022;29(3):1395-413. doi:10.1007/s11831-021-09622-6.
3. Cui Y, et al. Towards DDoS detection mechanisms in software-defined networking. *J Netw Comput Appl.* 2021; 190:103156. doi:10.1016/j.jnca.2021.103156.
4. Lee E, Seo YD, Oh SR, Kim YG. A survey on standards for interoperability and security in the Internet of Things. *IEEE Commun Surv Tutor.* 2021;23(2):1020-47. doi:10.1109/COMST.2021.3067354.
5. Thakkar A, Lohiya R. A review on machine learning and deep learning perspectives of IDS for IoT: recent updates, security issues, and challenges. *Arch Comput Methods Eng.* 2021;28(4). doi:10.1007/s11831-020-09496-0.
6. Li Y, Liu Q. A comprehensive review study of cyber-attacks and cyber security: emerging trends and recent developments. *Energy Rep.* 2021; 7:8176-86. doi:10.1016/j.egyr.2021.08.126.
7. Ma X, et al. A comprehensive survey on graph anomaly detection with deep learning. *IEEE Trans Knowl Data Eng.* 2021. doi:10.1109/TKDE.2021.3118815.
8. Nassif AB, Talib MA, Nasir Q, Dakalbab FM. Machine learning for anomaly detection: a systematic review. *IEEE Access.* 2021; 9:78658-700. doi:10.1109/ACCESS.2021.3083060.
9. Altaha M, Lee JM, Aslam M, Hong S. An autoencoder-based network intrusion detection system for the SCADA system. *J Commun.* 2021;16(6):210-6. doi:10.12720/jcm.16.6.210-216.
10. Moualla S, Khorzom K, Jafar A. Improving the performance of machine learning-based network intrusion detection systems on the UNSW-NB15 dataset. *Comput Intell Neurosci.* 2021;2021:5557577. doi:10.1155/2021/5557577.
11. Kumar R, Kumar P, Tripathi R, Gupta GP, Garg S, Hassan MM. A distributed intrusion detection system to detect DDoS attacks in blockchain-enabled IoT network. *J Parallel Distrib Comput.* 2022;164:55-68. doi:10.1016/j.jpdc.2022.01.030.
12. Chaudhary P, Gupta B, Singh AK. Implementing attack detection system using filter-based feature selection methods for fog-enabled IoT networks. *Telecommun Syst.* 2022;81(1):23-39. doi:10.1007/s11235-022-00927-w.
13. Abdullayeva FJ. Distributed denial of service attack detection in e-government cloud via data clustering. *Array.* 2022;15:100229. doi:10.1016/j.array.2022.100229.
14. Xing N, Zhao S, Wang Y, Ning K, Liu X. A dynamic intrusion detection system capable of detecting unknown attacks. *Int J Adv Comput Sci Appl.* 2023;14(7). doi:10.14569/IJACSA.2023.0140743.

15. Mittal M, Kumar K, Behal S. DL-2P-DDoSADF: deep learning-based two-phase DDoS attack detection framework. *J Inf Secur Appl.* 2023;78:103609. doi:10.1016/j.jisa.2023.103609.
16. Mohamed TS, Khalifah SM. Intrusion detection systems: a revisit of performance evaluation parameters. *Acad Int J Eng Sci.* 2024;2(1):15-21. doi:10.59675/E212.
17. Mohamed TS, Aydin S, Alkhayyat A, Malik RQ. Kalman and Cauchy clustering for anomaly detection based authentication of IoMTs using extreme learning machine. *IET Commun.* 2025;19(1):e12467. doi:10.1049/cmu2.12467.
18. Gupta C, Kumar A, Jain NK. An enhanced hybrid intrusion detection based on crow search analysis optimizations and artificial neural network. *Wirel Pers Commun.* 2024;134(1):43-68. doi:10.1007/s11277-024-10880-3.
19. Gupta C, Kumar A, Jain NK. Intelligent intrusion detection system based on crowd search optimization for attack classification in network security. *EURASIP J Inf Secur.* 2025;2025(1):22. doi:10.1186/s13635-025-00205-7.
20. Askarzadeh A. A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. *Comput Struct.* 2016;169:1-12. doi:10.1016/j.compstruc.2016.03.001.
21. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over-sampling technique. *J Artif Intell Res.* 2002;16:321-57. doi:10.1613/jair.953.
22. Demšar J. Statistical comparisons of classifiers over multiple data sets. *J Mach Learn Res.* 2006;7:1-30.
23. Sokolova M, Lapalme G. A systematic analysis of performance measures for classification tasks. *Inf Process Manag.* 2009;45(4):427-37. doi:10.1016/j.ipm.2009.03.002.
24. Brodersen KH, Ong CS, Stephan KE, Buhmann JM. The balanced accuracy and its posterior distribution. In: *Proceedings of the 20th International Conference on Pattern Recognition (ICPR).* 2010:3121-4. doi:10.1109/ICPR.2010.764.