CAUCHY – Jurnal Matematika Murni dan Aplikasi
Volume 11 (2) (2026), Pages 2160–2180
p-ISSN: 2086-0382; e-ISSN: 2477-3344

Comparative Evaluation of Hyperparameter Optimization Strategies for k-Nearest Neighbors over Mixed Domain Search Spaces

Meta Kallista[1,4*], Ig. Prasetya Dwi Wibawa[2,5], and Heni Widayani[3]

[1] *Computer Engineering Study Program, School of Electrical Engineering, Telkom University, Main Campus (Bandung Campus), Jl. Telekomunikasi No. 1, Bandung 40257, West Java, Indonesia*
[2] *Electrical Engineering Study Program, School of Electrical Engineering, Telkom University, Main Campus (Bandung Campus), Jl. Telekomunikasi No. 1, Bandung 40257, West Java, Indonesia*
[3] *Mathematics Department, Universitas Islam Negeri Maulana Malik Ibrahim Malang, Indonesia*
[4] *Center of Excellence for Human Centric Engineering, Institute of Sustainable Society, Telkom University, Main Campus (Bandung Campus), Jl. Telekomunikasi No. 1, Bandung 40257, West Java, Indonesia*
[5] *Center of Excellence for Smart Transportation and Robotics, Research Institute for Connectivity and Convergence for Smart Living, Telkom University, Main Campus (Bandung Campus), Jl. Telekomunikasi No. 1, Bandung 40257, West Java, Indonesia*

Abstract

k-Nearest Neighbors (KNN) remains a strong baseline due to its simplicity, interpretability, and non-parametric structure. However, its predictive performance is highly sensitive to interacting hyperparameters. This paper studies search-constrained hyperparameter optimization for KNN over a mixed search space comprising an integer neighborhood size, a categorical distance metric, and a continuous Minkowski exponent. Several strategies are compared under a nested cross-validation framework with standardized preprocessing, including grid search, random search, Bayesian optimization, genetic algorithm (GA), surrogate optimization, particle swarm optimization (PSO), and grey wolf optimizer (GWO). Experiments on public classification datasets from UCI, OpenML, and Kaggle evaluate predictive performance using accuracy, macro AUC, and cross-entropy, together with validation loss and runtime. The results indicate that adaptive optimizers generally provide more favorable performance–cost trade-offs than exhaustive grids under limited evaluations, while no single method is uniformly best across all datasets. Rank-based statistical comparisons using non-parametric tests further support the observed differences and motivate practical recommendations for selecting tuning strategies as a function of evaluation search and mixed-domain search-space characteristics. Using 15 benchmark datasets, we compared seven KNN hyperparameter tuning strategies under five evaluation metrics consisting of accuracy, macro AUC, CV loss, CE loss, and runtime, and summarized performance using global mean ranks. PSO achieved the best overall composite rank, although post-hoc Wilcoxon signed-rank tests with Holm correction showed a statistically significant advantage only over random search.

Keywords: k-Nearest Neighbors; hyperparameter optimization; Bayesian optimization; surrogate optimization; grey wolf optimizer; particle swarm optimization

*Corresponding author. E-mail: metakallista@telkomuniversity.ac.id

Received: May 2, 2026 Revised: July 3, 2026 Accepted: September 16, 2026
DOI: https://doi.org/10.18860/cauchy.v11i2.42295

1. Introduction

The k-Nearest Neighbors (KNN) classifier remains a strong and widely adopted baseline in supervised learning because it is a *non-parametric*, instance-based method that requires minimal assumptions about the data-generating distribution [1, 2]. By making decisions using local neighborhoods, KNN can model complex, non-linear class boundaries when the data exhibit local structure that is difficult to capture with a fixed parametric form [2, 3]. Its continued relevance is supported by classical theory: nearest-neighbor decision rules admit risk bounds with respect to the Bayes optimal error under suitable regularity conditions, motivating KNN as a principled benchmark rather than a purely heuristic baseline [1].

KNN is additionally attractive due to its conceptual simplicity and transparency. Because it is commonly described as a "lazy" learner, training overhead is low compared to many parametric learners, and predictions can be interpreted directly by inspecting which training points become nearest neighbors and how they vote for the final label [2, 3]. Recent surveys further document that KNN continues to be used across domains and continues to evolve through metric and neighborhood refinements, reinforcing its role as a dependable baseline in empirical evaluation pipelines [2, 3]. In addition to methodological reviews, application-oriented studies continue to show that KNN remains practically relevant and can still be substantially improved through suitable preprocessing and ensemble design. For example, a recent water-quality classification study reported that integrating SMOTE with KNN as a bagging estimator markedly improved class discrimination compared with a single KNN model, especially in terms of ROC-AUC, highlighting that KNN performance is strongly affected not only by neighborhood choice but also by data imbalance handling and model aggregation strategies [4].

Despite these advantages, KNN performance is highly sensitive to hyperparameter choices, and tuning becomes substantially more difficult once multiple parameters interact [2, 5]. In practical implementations, predictive quality depends not only on the neighborhood size k (integer), but also on the distance metric (categorical) and—when using Minkowski distance—the exponent p (continuous), creating a mixed-domain search space that is challenging to optimize reliably [2, 5]. This difficulty is exacerbated because the validation objective can be non-convex and noisy, and because evaluating a single configuration typically requires repeated scoring under cross-validation, increasing computational cost [5].

Traditional tuning strategies such as manual trial-and-error or exhaustive grids become impractical as the number of candidate values grows, due to the combinatorial explosion in the number of configurations [5, 6]. Random search is often more search-efficient than grid search because it explores more distinct configurations under the same evaluation limit and avoids systematically wasting trials on uninformative dimensions [5, 6]. These constraints motivate automated hyperparameter optimization (HPO) methods that aim to improve the performance-cost trade-off under limited evaluation searches [5].

The hyperparameter optimization literature spans uninformed sampling (grid/random), model-based methods (Bayesian optimization), surrogate-assisted global optimization, and population-based metaheuristics (e.g., evolutionary and swarm intelligence) [5]. Bayesian optimization typically fits a probabilistic surrogate model of validation performance and uses an acquisition function to select promising trials sequentially, offering sample efficiency when evaluations are expensive [5, 7]. Surrogate optimization has a long-standing foundation in global optimization of expensive black-box functions and emphasizes explicitly balancing exploration and exploitation through a response surface and uncertainty-aware sampling [8, 9].

Population-based metaheuristics are attractive for rugged or mixed-variable search spaces because they are derivative-free and can be adapted to mixed domains via encoding/rounding strategies. Particle Swarm Optimization (PSO) uses collective dynamics guided by personal and global best solutions [10], while Grey Wolf Optimizer (GWO) simulates leadership hierarchy and hunting behavior to balance exploration and exploitation [11]. However, metaheuristics can suffer from premature convergence or stagnation without appropriate diversity mechanisms,

which motivates controlled comparisons under evaluation-based stopping criterion [12, 13]. In the context of KNN, recent reviews emphasize sensitivity to neighborhood size and metric choice, strengthening the case for systematically comparing tuning strategies rather than relying on a single optimizer or manual heuristics [2, 3].

This study focuses on broader search-constrained comparative evaluation of several hyperparameter tuning strategies for classical KNN over a mixed-domain search space consisting of (i) the integer neighborhood size k, (ii) a categorical distance metric, and (iii) the continuous Minkowski exponent p. Previous works generally considered narrower KNN tuning spaces, for example, by optimizing only a limited set of conventional parameters such as the number of neighbors, weighting scheme, and distance metric, or by embedding KNN hyperparameter tuning within broader procedures such as feature selection, instance selection, and data-preprocessing pipelines [14–17].

The novelty of this study is a unified, search-constrained comparative evaluation of several hyperparameter tuning strategies for classical KNN over a mixed-domain search space consisting of (i) the integer neighborhood size k, (ii) a categorical distance metric, and (iii) the continuous Minkowski exponent p. Existing studies often evaluate one or two tuning strategies or focus only on predictive performance. Fewer studies compare model-based, sampling-based, and metaheuristic optimizers under the same evaluation budget while considering both predictive quality and computational cost. Specifically, we evaluate the methods implemented in this study, namely grid search, random search, Bayesian optimization, genetic algorithm, surrogate optimization, particle swarm optimization, and grey wolf optimizer, under the same nested validation framework and evaluation-based stopping criterion, and we report both predictive performance and computational cost to support practical method selection [5–11]. The remainder of this paper is organized as follows: Section 2 formalizes the tuning objective, evaluation protocol, metrics, and tuning strategies; Section 3 presents experimental results on public datasets and evaluations; and Section 4 discusses conclusions, limitations, and future research directions.

2. Methodology

This section presents the methodological framework used to evaluate hyperparameter tuning strategies for KNN. It begins by introducing the problem formulation and notation, followed by the definition of the KNN hyperparameter optimization problem and the tuning objective under a nested cross-validation protocol. The tuning strategies considered in this study are then described, together with the implementation procedures for handling mixed-variable search spaces and evaluation-based stopping criteria.

2.1. Problem Setup and Notation

Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ denote a supervised classification dataset, where $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional feature vector and $y_i \in \mathcal{Y} = \{1, \ldots, C\}$ is the class label. We consider the classical k-Nearest Neighbors (KNN) classifier as a non-parametric baseline [1, 2]. Given a query point $\mathbf{x}$, KNN predicts by majority vote among the k closest training samples under a distance function $d(\cdot, \cdot)$ [2, 3]. In particular, the distance between two samples is computed using the Minkowski distance defined in Eq. (1).

$$d_p(\mathbf{x}, \mathbf{z}) = \left(\sum_{j=1}^{d} |x_j - z_j|^p \right)^{1/p}, \quad p \geq 1, \tag{1}$$

where $p = 2$ yields the Euclidean distance, $p = 1$ yields the Manhattan (cityblock) distance, and $p = \infty$ yields the Chebyshev distance, defined as $d_\infty(\mathbf{x}, \mathbf{z}) = \max_{1 \leq j \leq d} |x_j - z_j|$ [2].

2.2. KNN Hyperparameter Optimization

2.2.1. Model definition

Let $\mathcal{N}_k(\mathbf{x})$ denote the index set of the k nearest neighbors of $\mathbf{x}$ in the training set under $d(\cdot,\cdot)$. The predicted class is given by

$$\hat{y}(\mathbf{x}) = \arg\max_{c\in\{1,\dots,C\}} \sum_{i\in\mathcal{N}_k(\mathbf{x})} \mathbb{I}(y_i = c), \tag{2}$$

Eq. (2) assigns the query sample to the class receiving the largest number of votes among its k nearest neighbors. Here $\mathbb{I}(\cdot)$ denotes an indicator function. The nearest-neighbor rule is a classical nonparametric classifier with well-known theoretical properties [1].

2.2.2. Mixed-domain search space

We tune KNN over a mixed-domain hyperparameter space

$$h = (k, m, p) \in \mathcal{H} = \mathcal{K} \times \mathcal{M} \times \mathcal{P},$$

where k is the number of neighbors, $k \in \mathcal{K}$ and $k \in [1, 51] \cap \mathbb{Z}^+$, $m \in \mathcal{M}$ is the categorical distance metric, and $p \in \mathcal{P}$ is the Minkowski exponent with $p \in [1, 5]$ and $p \geq 1$. The categorical set of distance metrics is

$$\mathcal{M} = \{\texttt{euclidean}, \texttt{cityblock}, \texttt{chebychev}, \texttt{minkowski}\}.$$

The exponent p is *active* only when $m = \texttt{minkowski}$; otherwise, p is ignored (or set to a default such as $p = 2$) [5, 18]. In our implementation, (k, m, p) corresponds to the standard KNN parameters `n_neighbors`, `metric`, and `p` [19].

2.2.3. Hyperparameter selection objective

Let $\mathcal{L}(\cdot)$ denote a chosen loss function (e.g., classification error or cross-entropy). Hyperparameters are selected by minimizing the average K-fold cross-validation loss, as formulated in Eq.(3).

$$h^* = \arg\min_{h\in\mathcal{H}} \mathcal{L}_{\mathrm{CV}}(h) = \arg\min_{h\in\mathcal{H}} \frac{1}{K} \sum_{t=1}^{K} \mathcal{L}\left(\hat{f}_h^{(-t)}, \mathcal{D}^{(t)}\right), \tag{3}$$

where $\hat{f}_h^{(-t)}$ is the KNN model with hyperparameters h trained on all folds except t and evaluated on the validation fold $\mathcal{D}^{(t)}$. This validation-based selection is appropriate for KNN because performance can be sensitive to the distance definition and neighborhood size in high-dimensional feature spaces [20].

2.3. Tuning Objective and Nested Cross-Validation Protocol

We formulate hyperparameter tuning as an optimization problem over the mixed-domain search space $\mathcal{H}$:

$$h^\star = \arg\min_{h\in\mathcal{H}} \mathcal{L}_{\mathrm{CV}}(h; \mathcal{D}_{\mathrm{tr}}), \tag{4}$$

where $\mathcal{L}_{\mathrm{CV}}(h; \mathcal{D}_{\mathrm{tr}})$ is the inner cross-validation loss computed using only the outer training split $\mathcal{D}_{\mathrm{tr}}$ [5, 18]. Eq. (4) restricts the optimization process to the outer training data only, thereby preventing information leakage from the independent test partition. We adopted a nested validation protocol consisting of an outer stratified holdout split and an inner stratified K-fold cross-validation loop. For each dataset, the full dataset $\mathcal{D}$ was first divided into an outer training subset $\mathcal{D}_{\mathrm{tr}}$ and an independent test subset $\mathcal{D}_{\mathrm{te}}$ using a holdout ratio of $\alpha = 0.2$, corresponding to 80% training data and 20% test data. Stratification was applied to preserve the class proportions in both the outer training and test subsets. The outer training subset $\mathcal{D}_{\mathrm{tr}}$ was then used for hyperparameter selection through inner $K = 5$ stratified cross-validation.

Within $\mathcal{D}_{\text{tr}}$, the inner loop selects $h^\star$ by minimizing the average K-fold loss

$$\mathcal{L}_{\text{CV}}(h; \mathcal{D}_{\text{tr}}) = \frac{1}{K} \sum_{t=1}^{K} \mathcal{L}\Big(\hat{f}_h^{(-t)}, \mathcal{D}^{(t)}\Big), \tag{5}$$

where $\hat{f}_h^{(-t)}$ is the KNN model with hyperparameter configuration h trained on all inner folds except fold t and evaluated on the corresponding validation fold $\mathcal{D}^{(t)}$. As defined in Eq.(5), the objective value is obtained by averaging the validation loss across all K folds. Each objective evaluation corresponds to one complete inner 5-fold cross-validation assessment of a candidate KNN configuration.

For reproducibility, all data partitions were generated using a fixed random seed. For a given dataset, all tuning methods used the same outer holdout split and the same inner cross-validation partitions so that performance differences were attributable to the tuning strategy rather than to different data splits. To avoid data leakage, standardization was performed strictly within the validation procedure. Specifically, the standardization parameters were estimated only from the training portion of each inner fold and then applied to the corresponding validation fold. After the best hyperparameter configuration was selected, the final KNN model was retrained on the full outer training subset $\mathcal{D}_{\text{tr}}$ and evaluated only once on the independent test subset $\mathcal{D}_{\text{te}}$. This separation reduces optimistic bias introduced by hyperparameter optimization and supports an unbiased estimation of generalization performance [5].

For evaluation-limited methods, including random search, Bayesian optimization, genetic algorithm, surrogate optimization, particle swarm optimization, and grey wolf optimization, the maximum number of objective evaluations was fixed at $B = 40$. In contrast, grid search was treated as a deterministic exhaustive baseline because its evaluation count is determined by the predefined grid rather than by the common budget B.

2.4. Tuning Strategies

We evaluate seven tuning methods implemented in our framework, including grid search, random search, Bayesian optimization, genetic algorithm, surrogate optimization, particle swarm optimization, and grey wolf optimization. Each method proposes a sequence of candidate configurations $\{h_t\}_{t=1}^{B}$, evaluates $\mathcal{L}_{\text{CV}}(h_t)$, and returns the best observed configuration $h^\star$.

2.4.1. Grid Search

Grid Search evaluates a predefined discrete set $\mathcal{G} \subset \mathcal{H}$ and selects

$$h^\star = \arg\min_{h \in \mathcal{G}} \mathcal{L}_{\text{CV}}(h; \mathcal{D}_{\text{tr}}).$$

In our implementation, the grid is formed by discrete values of k, the categorical metric m, and a small set of p values if $m = \texttt{minkowski}$. Grid Search is simple, deterministic, and reproducible, but its computational cost grows rapidly with the number of grid points, often allocating evaluations inefficiently across dimensions that have limited its performance [5, 6].

2.4.2. Random Search

Random search samples $h_t \sim \pi(\mathcal{H})$ i.i.d. from a user-defined distribution π (uniform or mixed discrete-continuous), and returns the best observed configuration:

$$h^\star = \arg\min_{t \in \{1,\dots,B\}} \mathcal{L}_{\text{CV}}(h_t).$$

Each trial draws k uniformly from $[1, 51]$, picks m uniformly from $\mathcal{M}$, and samples p from $[1, 5]$. Random search is easy to implement and often more effective than grid search when only a subset of hyperparameters strongly affects performance [5, 6]. However, it does not learn from previous evaluations and can miss narrow optima under small search [5, 18].

2.4.3. Bayesian Optimization

Bayesian optimization fits a surrogate probabilistic model to approximate $\mathcal{L}_{\mathrm{CV}}(h)$ and selects new evaluation points by maximizing an acquisition function [5, 7]. Let $\mathcal{S}_t = \{(h_i, \mathcal{L}_{\mathrm{CV}}(h_i))\}_{i=1}^{t}$ be observed evaluations. A common approach models $\mathcal{L}_{\mathrm{CV}}(h)$ as a Gaussian process (GP) surrogate, providing a posterior prediction with mean $\mu_t(h)$ and uncertainty $\sigma_t(h)$ over the search space [7]. The next configuration is chosen by

$$h_{t+1} = \arg\max_{h \in \mathcal{H}} a_t(h),$$

where a_t is an acquisition function such as Expected Improvement (EI) or one of its robust variants [7, 21]. We treat (k, m, p) as optimizable variables with mixed types (integer/categorical/real) and minimize inner-CV loss. Bayesian optimization is sample-efficient for expensive objectives because it uses posterior uncertainty to trade off exploration and exploitation [5, 7]. Its overhead (surrogate fitting and acquisition optimization) can be non-negligible, and mixed categorical variables require careful encoding, which can affect robustness [5, 21].

2.4.4. Genetic Algorithm (GA)

Genetic algorithms maintain a population $\{h^{(i)}\}_{i=1}^{N_p}$ and iteratively apply selection, crossover, and mutation to minimize $\mathcal{L}$:

$$\{h^{(i)}\}_{i=1}^{N_p} \xrightarrow{\text{selection/crossover/mutation}} \{h_{\text{new}}^{(i)}\}_{i=1}^{N_p}.$$

Fitness is defined as $f(h) = -\mathcal{L}_{\mathrm{CV}}(h)$ so that fitter individuals have lower loss [5, 22]. We encode $h = (k, m, p)$ as a mixed genome consisting of integer k, integer index for categorical m, and real p, with rounding for discrete components after genetic operations. GA is derivative-free and well suited to rugged or irregular search landscapes, naturally supporting mixed-variable optimization [5, 22]. However, it may require many evaluations for stable convergence, and performance depends on initial settings and population diversity [5, 18].

2.4.5. Surrogate Optimization

Surrogate optimization constructs an explicit approximation $\widehat{\mathcal{L}}(h)$ of the objective function from previously evaluated points and uses this model to guide subsequent evaluations [8, 9]. Classical Efficient Global Optimization (EGO) uses kriging models and expected improvement [8], whereas practical implementations of surrogate-based global search often employ radial basis function approximations together with merit functions that combine low predicted loss and distance from previously evaluated points [23, 24]. In our setting, we define the objective as inner-CV loss and provide bounded variables, where integer constraints are handled by rounding discrete dimensions. Surrogate optimization is designed for expensive black-box objectives and can be effective under strict evaluation limit [8, 9].

Bayesian optimization and surrogate optimization are both surrogate-based in a broad sense, but they differ in their search mechanisms. Bayesian optimization uses a probabilistic surrogate model and an acquisition function to balance exploration and exploitation when selecting the next hyperparameter configuration. In contrast, surrogate optimization is considered a derivative-free surrogate-assisted global search method for expensive black-box objectives, where the surrogate model guides the search over the bounded hyperparameter space without necessarily following the Bayesian acquisition-function framework. Therefore, both methods are evaluated as separate tuning methods in this study.

2.4.6. Particle Swarm Optimization (PSO)

PSO maintains a swarm of particles with positions $\mathbf{x}_i^t$ and velocities $\mathbf{v}_i^t$ in a continuous embedding of $\mathcal{H}$ [10]. At each iteration, particle velocities are updated by combining inertia, cognitive, and

social components according to Eq. (6).

$$\begin{aligned} \mathbf{v}_i^{t+1} &= \omega \mathbf{v}_i^t + c_1 r_1 (\mathbf{p}_i^t - \mathbf{x}_i^t) + c_2 r_2 (\mathbf{g}^t - \mathbf{x}_i^t), \\ \mathbf{x}_i^{t+1} &= \mathbf{x}_i^t + \mathbf{v}_i^{t+1}, \end{aligned} \tag{6}$$

where $\mathbf{p}_i^t$ is the personal best, $\mathbf{g}^t$ is the global best, ω is the inertia weight, c_1 and c_2 are acceleration coefficients, and $\mathbf{r}_1, \mathbf{r}_2 \sim U(0,1)$[10]. We embed $h = (k, m, p)$ as a vector $\mathbf{x} = [k, \mathrm{idx}(m), p]$, apply bound constraints, and decode mixed variables by rounding k and $\mathrm{idx}(m)$ to valid integers, then mapping $\mathrm{idx}(m)$ back to a metric label; p remains continuous. PSO is simple, derivative-free, and often converges quickly in early iterations [10, 25]. However, it can prematurely converge or stall in local optima, motivating hybrid and adaptive PSO variants in recent optimization literature [18, 25].

2.4.7. Grey Wolf Optimizer (GWO)

GWO models the search process through a leadership hierarchy in which the three best solutions, denoted by α, β, and δ, guide the population [11]. Given a candidate position $\mathbf{x}^t$, updates are computed using coefficient vectors

$$A = 2ar_1 - a, \qquad C = 2r_2, \tag{7}$$

where a decreases from 2 to 0 over iterations and $r_1, r_2 \sim U(0,1)$ [11]. As indicated by Eq. (7), the control parameter a decreases linearly from 2 to 0 throughout the optimization process. Using leaders $\mathbf{x}_\alpha, \mathbf{x}_\beta, \mathbf{x}_\delta$, three candidate moves are

$$\begin{aligned} \mathbf{D}_\alpha &= |C_1 \mathbf{x}_\alpha - \mathbf{x}|, \quad \mathbf{x}_1 = \mathbf{x}_\alpha - A_1 \mathbf{D}_\alpha, \\ \mathbf{D}_\beta &= |C_2 \mathbf{x}_\beta - \mathbf{x}|, \quad \mathbf{x}_2 = \mathbf{x}_\beta - A_2 \mathbf{D}_\beta, \\ \mathbf{D}_\delta &= |C_3 \mathbf{x}_\delta - \mathbf{x}|, \quad \mathbf{x}_3 = \mathbf{x}_\delta - A_3 \mathbf{D}_\delta, \end{aligned} \tag{8}$$

and the updated position is the average

$$\mathbf{x}^{t+1} = \frac{\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3}{3}. \tag{9}$$

As in PSO, we use a continuous embedding $\mathbf{x} = [k, \mathrm{idx}(m), p]$ and decode via rounding for discrete components and mapping back to the categorical metric. As can be seen in Eqs.7-9, GWO has few parameters and often provides a balanced exploration–exploitation search [11, 26]. Nevertheless, recent studies note limitations such as stagnation and insufficient diversity, motivating improved multi-strategy or hybrid GWO variants that enhance convergence and escape local optima [12, 26].

2.5. Implementation Notes for Mixed Variables and Evaluation-Based Stopping Criteria

To ensure fairness across methods, all optimizers are allocated the same evaluation limit B (except exhaustive grid search, which evaluates a fixed grid). For population-based methods (GA, PSO, GWO), mixed variables are handled by continuous embedding and decoding via rounding and categorical indexing, a standard approach for adapting swarm/evolutionary optimizers to mixed domains [5, 18]. Finally, the best configuration $h^\star$ returned by each tuning method is refit on the outer training split and evaluated once on the held-out test split to produce unbiased test metrics.

3. Results and Discussion

This section presents the experimental framework and analyzes the results of the comparative evaluation of the KNN hyperparameter tuning methods. The discussion begins with the

experimental configuration, followed by the initialization settings of each tuning method and the definition of the KNN hyperparameter search space. Subsequently, the data preprocessing procedures and evaluation outputs are described to provide the methodological context for interpreting the results. Finally, the relative performance of the tuning methods is examined through a rank-based comparison across the evaluated datasets.

3.1. Experimental Configuration

Table 1 summarizes the datasets used in this study, including the number of instances, features, classes, task type (binary/multiclass), and the corresponding data sources. All tuning methods were evaluated under an identical nested validation protocol: an outer holdout split for unbiased test evaluation and an inner K-fold cross-validation for model selection. This design follows common recommendations for reliable hyperparameter evaluation under limited computational resources [5, 18]. Unless otherwise stated, the outer holdout ratio was set to $\alpha = 0.2$ and the inner CV used $K = 5$ folds. Each stochastic method was run once per dataset using a fixed random seed, `RngSeed = 1`, to ensure reproducibility under the same experimental protocol. For evaluation-limited methods (random search, Bayesian optimization, GA, surrogate optimization, PSO, and GWO), the maximum number of objective evaluations was fixed to $B =$ `MaxEvals` (e.g., $B = 40$), ensuring fair comparison in terms of evaluation count [5, 18]. Meanwhile, Grid Search performed 182 objective evaluations, corresponding to 26 candidate values of k and seven metric-related configurations, i.e., three non-Minkowski metrics and four Minkowski exponent values, as shown in Table 2. Grid Search evaluates a predefined exhaustive grid, so its evaluation count is determined by grid size rather than the common budget $B = 40$ [6]. Therefore, it is included as a deterministic exhaustive baseline and is not fully evaluation-matched with the evaluation-limited methods.

Table 1: Datasets used in this study and preprocessing checklist.

No	Dataset	#data	#features	#classes	Classification	Has missing value?	Has duplicate data?	Has categorical feature?	Is imbalanced class?	Ref.
1	heart disease	1025	13	2	Binary	×	✓	✓	×	[27]
2	ionosphere	351	34	2	Binary	×	✓	×	×	[28]
3	spambase	4601	57	2	Binary	×	✓	×	×	[29]
4	haberman	306	3	2	Binary	×	✓	×	✓	[30]
5	blood transfusion service center	748	4	2	Binary	×	✓	×	✓	[31]
6	sonar	208	60	2	Binary	×	×	×	×	[32]
7	breast cancer	286	9	2	Binary	✓	✓	✓	✓	[33]
8	diabetes	768	8	2	Binary	×	×	×	×	[34]
9	iris	150	4	3	Multiclass	×	✓	×	×	[35]
10	wine	178	13	3	Multiclass	×	×	×	×	[36]
11	satellite image	6435	36	6	Multiclass	×	×	×	✓	[37]
12	glass	214	9	6	Multiclass	×	✓	×	✓	[38]
13	winequality red	1599	11	6	Multiclass	×	✓	×	✓	[39]
14	winequality white	4898	11	7	Multiclass	×	✓	×	✓	[39]
15	yeast	1484	8	10	Multiclass	×	✓	×	✓	[40]

*) We use an imbalance ratio threshold, $IR \geq 2.0$, where $IR = n_{\max}/n_{\min}$, $n_{\max}$ and $n_{\min}$ denote the largest and smallest class counts, respectively.

3.2. Initialization Settings for Each Tuning Method

All experiments were implemented in MATLAB R2022b and executed on a desktop computer running Windows 11 Education 64-bit. The hardware platform consisted of an Intel Core i7-8700, 16 GB RAM, and an NVIDIA GeForce GTX 1050 Ti. For evaluation-limited tuning methods, namely Random Search, Bayesian Optimization, Genetic Algorithm, Surrogate Optimization, PSO, and GWO, the maximum number of objective-function evaluations was fixed at $B = 40$ to ensure a fair comparison across methods, whereas grid search evaluated all predefined grid combinations. Table 2 summarizes the main initialization and control settings used for each tuning method. Grid search and random search were configured according to their conventional

search procedures [6]. Bayesian optimization and surrogate optimization followed standard sequential and surrogate-based optimization settings [7, 21, 23, 24]. The population-based methods, i.e., PSO and GWO, were initialized using commonly adopted parameter settings based on their original formulations [10, 11].

Table 2: Initialization settings used for each hyperparameter tuning method.

Method	Initialization / Control Parameters
Grid Search	$k \in \{1, 3, \ldots, 51\}$; $m \in \{\texttt{euclidean}, \texttt{cityblock}, \texttt{chebychev}, \texttt{minkowski}\}$; $p \in \{1, 2, 3, 4\}$ only if $m = \texttt{minkowski}$
Random Search	Evaluation-limited with budget B; randomly sample integer k from $[1, 51]$, categorical distance metric m from $\mathcal{M}$, and real-valued Minkowski exponent p from $[1, 5]$; return the best configuration.
Bayesian Optimization	Budget B evaluations; mixed search space with integer, categorical, and real-valued variables; acquisition function: `expected-improvement-plus`.
Genetic Algorithm	Population size $N_p = 30$; number of generations $\max(5, \lceil B/10 \rceil)$; best individual selected based on fitness $-\mathcal{L}(h)$.
Surrogate Optimization	Budget B evaluations; surrogate model used to approximate the objective function and guide the search.
PSO	Swarm size $N = \min(30, \max(10, \lfloor 4\sqrt{B} \rfloor))$; inertia weight $\omega = 0.72$; acceleration coefficients $c_1 = c_2 = 1.49$; velocity $v_{\max} = 0.2(\text{ub} - \text{lb})$.
GWO	Pack size $N = \min(30, \max(10, \lfloor 4\sqrt{B} \rfloor))$; control parameter a decreases linearly from 2 to 0; leadership update based on α, β, and δ wolves; mixed-variable decoding by rounding.

3.3. Hyperparameter Space Tuned for KNN

Table 3 lists the KNN search space used consistently across all tuning methods. This mixed-domain formulation is typical in hyperparameter optimization because it includes discrete and categorical decisions as well as continuous parameters [5, 18]. The Minkowski exponent p is active only when $m = \texttt{minkowski}$. For `euclidean`, `cityblock`, and `chebychev`, the value of p is ignored by the KNN model and is therefore reported as inactive in the result tables. The term "inactive" indicates that the Minkowski exponent p is not used because the selected distance metric is not `minkowski`.

Table 3: KNN hyperparameter search space used in all tuning methods.

Hyperparameter	Domain / Notes
k (neighbors)	Integer: $k \in [1, 51] \cap \mathbb{Z}$
m (distance metric)	Categorical: $\{\texttt{euclidean}, \texttt{cityblock}, \texttt{chebychev}, \texttt{minkowski}\}$
p (Minkowski exponent)	Real: $p \in [1, 5]$. The parameter p is active only when $m = \texttt{minkowski}$; for `euclidean`, `cityblock`, and `chebychev`, p is ignored and reported as inactive in the result tables.
Objective $\mathcal{L}_{\text{CV}}(h)$	Inner K-fold CV loss on the training split. Each objective evaluation corresponds to one complete inner CV assessment of a candidate KNN configuration. [5]

3.4. Data Preprocessing and Evaluation Outputs

Data preprocessing was performed before model evaluation. Numerical predictors were retained or converted from numeric-like strings when appropriate, while missing numerical values were handled using linear interpolation followed by mean imputation. Missing categorical values were replaced by a fixed token, and categorical predictors were encoded using one-hot encoding. Duplicate records were inspected and reported but not removed to preserve the original benchmark datasets. Class imbalance was assessed using the imbalance ratio and handled through stratified

splitting and macro-level metrics, without resampling. To avoid data leakage, global normalization was disabled, and z-score standardization was performed within the cross-validation and final training procedures using training data only.

For numerical results, we evaluate best-found hyperparameters, best inner-CV loss, and test metrics including accuracy, cross-entropy loss, and macro-AUC. These metrics provide complementary views of discrimination, probabilistic calibration (via cross-entropy), and class-balanced separability (macro-AUC) [5, 18]. The detailed dataset list (UCI/OpenML/Kaggle) and preprocessing steps are provided in Section 2. Predicted class probabilities were obtained from normalized neighbor vote proportions, or equivalently, from the class-score outputs. The cross-entropy loss was computed as

$$\text{CE Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\hat{p}_{ic}),$$

where $\hat{p}_{ic}$ denotes the predicted probability of class c for sample i, and y_{ic} is the one-hot encoded true label. Class probabilities were obtained from the class-score outputs, corresponding to the normalized proportions of neighboring samples assigned to each class.

As shown in Table 8, the perfect performance observed on the heart disease dataset should be interpreted cautiously. This dataset version contains 1025 instances and includes 723 duplicate records, which may cause an instance-based classifier such as KNN to produce overestimated performance estimates when similar or identical samples appear in both the training and test subsets.

The performance differences among tuning methods can be interpreted in relation to the characteristics of the datasets. For small and low-dimensional datasets, such as iris, wine, haberman, and glass, most tuning methods achieved relatively comparable predictive performance because the KNN search space could be explored sufficiently within the limited evaluation budget. In contrast, larger datasets or datasets with higher feature dimensionality, such as spambase, satellite image, sonar, and ionosphere, increased the computational cost of each KNN evaluation, making runtime and evaluation efficiency more influential in the overall comparison. Multiclass and imbalanced datasets, such as glass, winequality red, winequality white, yeast, and satellite image, also posed additional challenges.

3.4.1. Friedman test

To assess whether there are overall differences among k tuning methods across N datasets, we apply the Friedman test, a non-parametric test for repeated measures [41]. For each dataset i, the k methods are ranked, yielding r_{ij} for dataset i and method j. Let

$$R_j = \sum_{i=1}^{N} r_{ij}$$

denote the rank sum of method j. The Friedman chi-square statistic is

$$\chi_F^2 = \frac{12}{Nk(k+1)}\sum_{j=1}^{k} R_j^2 - 3N(k+1), \tag{10}$$

the statistic in Eq. (10) is approximately chi-square distributed with $k-1$ degrees of freedom under the null hypothesis of equal method performance [41]. If the Friedman test is significant, post-hoc analyses such as Wilcoxon signed-rank test with Holm correction can be used to localize differences [41].

3.4.2. Wilcoxon signed-rank test

To compare two tuning methods A and B across the same N datasets, we use the Wilcoxon signed-rank test, a non-parametric paired test based on ranks [41]. Let x_i and y_i denote the

per-dataset performance values (e.g., composite ranks or a single metric value) of methods A and B on dataset i, and define the paired differences

$$d_i = x_i - y_i, \qquad i = 1, \ldots, N. \tag{11}$$

After discarding zero differences in Eq. 11 ($d_i = 0$), let n be the number of remaining pairs. We rank the absolute differences $|d_i|$ in ascending order (ties receive midranks), obtaining ranks R_i. We then compute the sums of ranks for positive and negative differences:

$$W^+ = \sum_{i:d_i>0} R_i, \qquad W^- = \sum_{i:d_i<0} R_i.$$

A common Wilcoxon statistic is $T = \min(W^+, W^-)$, and the p-value is obtained from the exact distribution (small n) or a normal approximation (large n) [41]. A simple descriptive summary of the typical direction and magnitude of the paired effect is the sample median of the differences,

$$\Delta = \text{median}\{d_1, \ldots, d_N\},$$

which is commonly reported alongside the Wilcoxon test to indicate whether method A tends to yield larger or smaller values than method B across datasets.

3.4.3. Holm correction for multiple Wilcoxon tests

When multiple pairwise Wilcoxon tests are performed, we control the family-wise error rate (FWER) using Holm's step-down procedure [42]. Given m p-values $p_1, \ldots, p_m$, sort them as $p_{(1)} \leq \cdots \leq p_{(m)}$. Holm's decision rule rejects $H_{(i)}$ if

$$p_{(i)} \leq \frac{\alpha}{m - i + 1}, \qquad i = 1, \ldots, m,$$

and stops at the first index for which the inequality fails [42]. Equivalently, Holm-adjusted p-values can be reported as

$$p_{(i)}^{\text{Holm}} = \max_{1 \leq j \leq i} \left[(m - j + 1)\, p_{(j)} \right], \qquad p_{(i)}^{\text{Holm}} \leftarrow \min\left(p_{(i)}^{\text{Holm}}, 1 \right).$$

3.5. Performance Rank Comparison of Tuning Methods)

From Table 8, we compared seven tuning methods across 15 datasets using a rank-based non-parametric protocol. Composite ranks were computed per dataset by averaging ranks over five metrics (Accuracy and Macro AUC: higher is better; CV Loss, CE Loss, and Runtime: lower is better), as seen in Table 4. For simplicity, we assume all evaluation metrics contribute equally by assigning the same weight to each metric when computing the overall score/rank. Tied observations are handled by assigning average ranks to all methods sharing the same value.

Table 4: Evaluation metrics and their optimization directions.

Metric	Better direction	Reason
Accuracy	Higher is better	Classification performance
Macro AUC	Higher is better	Class-balanced discrimination
CV Loss	Lower is better	Validation error
CE Loss	Lower is better	Test error
Runtime	Lower is better	Computational cost

Table 5 shows the global mean ranks of the hyperparameter tuning methods across all datasets, where lower ranks indicate better performance. Particle Swarm Optimization (PSO) achieves the best overall rank, primarily driven by its strong performance in computational efficiency

Table 5: Global mean ranks of tuning methods across all datasets (lower is better).

Tuning method	Acc rank	Macro AUC rank	CV Loss rank	CE Loss rank	Runtime rank	Overall rank
PSO	4.500	3.567	5.633	**2.933**	**1.267**	**3.580**
Grid Search	3.533	**3.200**	**2.633**	3.567	5.467	3.680
Bayesian Optimization	**3.367**	3.667	2.967	3.600	5.200	3.760
Surrogate Optimization	3.500	4.533	4.000	4.833	2.867	3.947
Grey Wolf Optimizer	4.167	4.767	4.433	4.700	1.867	3.987
Genetic Algorithm	4.100	4.400	2.767	4.467	4.333	4.013
Random Search	4.833	3.867	5.567	3.900	7.000	5.033

(runtime) and competitive error-related metrics, despite relatively weaker accuracy-based ranks. Grid Search and Bayesian Optimization follow closely, exhibiting balanced performance across predictive and loss-based metrics, with Grid Search performing particularly well in Macro AUC and CV loss, and Bayesian Optimization attaining the best average rank in accuracy. In contrast, Random Search consistently yields the highest (worst) ranks, especially in runtime and loss metrics, indicating inferior overall performance. These results suggest that metaheuristic and population-based approaches, particularly PSO, provide a favorable trade-off between predictive performance and computational cost when evaluated across multiple metrics and datasets.

Table 6: Friedman test on global ranks across datasets.

Item	Value
Number of datasets (N)	15
Number of methods (k)	7
Degrees of freedom ($k-1$)	6
Friedman statistic (χ^2_F)	22.859
p-value	0.000845
Significance level (α)	0.05
Decision	**Reject** H_0

As shown in Table 6, the Friedman test indicates statistically significant differences among the seven tuning methods when evaluated via global composite ranks across the 15 datasets ($\chi^2_F = 22.859$, $p = 0.000845$). Therefore, the null hypothesis that all methods have equal performance (in terms of average ranks) is rejected at $\alpha = 0.05$, implying that at least one method differs significantly from the others and motivating post-hoc pairwise analysis. In this study, we use Wilcoxon (with multiplicity correction/Holm correction) to analyze the differences. To localize differences after ranking, we computed paired per-dataset differences in composite ranks between the best overall method (PSO) and each alternative method and applied the Wilcoxon signed-rank test to these paired differences across the same $N = 15$ datasets. As shown in Table 7, PSO is significantly better than Random Search ($p_{\text{Holm}} = 0.000366$), while the remaining comparisons are not significant after Holm correction ($p_{\text{Holm}} \geq 0.316$).

Table 7: Wilcoxon signed-rank tests on paired per-dataset rank differences.

Comparison	N	Median Δ	W	p	p_{Holm}	Significant?
PSO vs random search	15	−1.2	0	0.000061	**0.000366**	**Yes**
PSO vs grey wolf optimizer	15	−0.2	23	0.063254	0.316268	No
PSO vs surrogate optimization	15	−0.4	31	0.106995	0.427979	No
PSO vs genetic algorithm	15	−0.6	41	0.302795	0.908386	No
PSO vs Bayesian optimization	15	−0.2	48	0.524475	1.000000	No
PSO vs grid search	15	−0.3	47	0.729635	1.000000	No

4. Conclusion and Future Works

This paper evaluated hyperparameter tuning strategies for k-Nearest Neighbors (KNN) over a mixed-domain search space consisting of an integer neighborhood size (k), a categorical distance metric, and a continuous Minkowski exponent (p). We ensured a fair and unbiased comparison among grid search, random search, Bayesian optimization, genetic algorithms, surrogate optimization, particle swarm optimization, and grey wolf optimization. The empirical results indicate that no single tuner consistently dominates across all datasets; instead, the best choice depends on the evaluation-limited and dataset properties such as dimensionality, class structure, and objective irregularity. Overall, adaptive optimizers tend to offer more favorable performance–cost trade-offs than exhaustive grids under limited evaluations, while population-based methods remain competitive on rugged or mixed-variable landscapes.

Future work will extend the analysis to multi-objective tuning to explicitly capture trade-offs between predictive performance and computational cost. Sensitivity analysis will also be conducted to examine the effect of alternative composite-ranking weights, including separate rankings for predictive performance and runtime. Since the stochastic methods in this study were evaluated using a single random seed, future experiments will use multiple random seeds and repeated outer splits to assess ranking stability and performance variability. In addition, future work will investigate hybrid strategies that combine surrogate modeling with population-based search, as well as recent metaheuristics with diverse search operators and adaptive mechanisms, to evaluate their robustness across datasets, search budgets, and mixed-variable KNN hyperparameter spaces [43–45].

CRediT Authorship Contribution Statement

Meta Kallista: Conceptualization, Methodology, Formal Analysis, Writing–Original Draft, Editing & Validation. **Ig. Prasetya Dwi Wibawa**: Methodology, Formal Analysis, Software, Writing–Review, Editing & Validation. **Heni Widayani**: Validation, Writing–Review & Editing.

Declaration of Generative AI and AI-assisted technologies

During the preparation of this work, the authors used Copilot in order to improve the clarity, readability, and language of the manuscript. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Declaration of Competing Interest

The authors declare no competing interests.

Funding and Acknowledgments

The authors gratefully acknowledge Research and Community Service, Telkom University's support through Research Funding Grant No. RCMS/RES/Periode 1/14943/2/2026

Data and Code Availability

The dataset can be seen in Table 1. The code used in this study is publicly available at https://github.com/metakallista/Tuning-KNN-Hyperparameter.

References

[1] Thomas M. Cover and Peter E. Hart. “Nearest Neighbor Pattern Classification”. In: *IEEE Transactions on Information Theory* 13.1 (1967). Accessed: 2026-04-30, pp. 21–27. DOI: 10.1109/TIT.1967.1053964. https://isl.stanford.edu/~cover/papers/transIT/0021cove.pdf.

[2] Panos K. Syriopoulos, Nektarios G. Kalampalikis, Sotiris B. Kotsiantis, and Michael N. Vrahatis. “kNN Classification: a review”. In: *Annals of Mathematics and Artificial Intelligence* (2023). DOI: 10.1007/s10472-023-09882-x.

[3] Rajib Kumar Halder, Mohammed Nasir Uddin, Md. Ashraf Uddin, Sunil Aryal, and Ansam Khraisat. “Enhancing K-nearest neighbor algorithm: a comprehensive review and performance analysis of modifications”. In: *Journal of Big Data* 11 (2024), p. 113. DOI: 10.1186/s40537-024-00973-y.

[4] Ivana Meiska, Syifa Melinda Naf’an, Ig Prasetya Dwi Wibawa, Meta Kallista, Brilliant Friezka Aina, and Asif Awaludin. “Improving k-Nearest Neighbors Model using SMOTE with Bagging Ensemble”. In: *2023 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*. IEEE. 2023, pp. 195–201.

[5] Luca Franceschi, Michele Donini, Valerio Perrone, Aaron Klein, Cédric Archambeau, Matthias Seeger, Massimiliano Pontil, and Paolo Frasconi. “Hyperparameter Optimization in Machine Learning”. In: *Foundations and Trends in Machine Learning* 18.6 (2025). DOI: 10.1561/2200000088. https://arxiv.org/abs/2410.22854.

[6] James Bergstra and Yoshua Bengio. “Random Search for Hyper-Parameter Optimization”. In: *Journal of Machine Learning Research* 13 (2012), pp. 281–305. https://jmlr.org/papers/v13/bergstra12a.html.

[7] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. “Practical Bayesian Optimization of Machine Learning Algorithms”. In: *Advances in Neural Information Processing Systems*. Vol. 25. 2012. https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf.

[8] Donald R. Jones, Matthias Schonlau, and William J. Welch. “Efficient Global Optimization of Expensive Black-Box Functions”. In: *Journal of Global Optimization* 13.4 (1998), pp. 455–492. DOI: 10.1023/A:1008306431147.

[9] Dat Ha and Josephine Carstensen. “Automatic hyperparameter tuning of topology optimization algorithms using surrogate optimization”. In: *Structural and Multidisciplinary Optimization* 67 (2024), p. 157. DOI: 10.1007/s00158-024-03850-7.

[10] James Kennedy and Russell Eberhart. “Particle Swarm Optimization”. In: *Proceedings of the IEEE International Conference on Neural Networks*. IEEE, 1995, pp. 1942–1948. DOI: 10.1109/ICNN.1995.488968.

[11] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. “Grey Wolf Optimizer”. In: *Advances in Engineering Software* 69 (2014), pp. 46–61. DOI: 10.1016/j.advengsoft.2013.12.007.

[12] Binbin Tu, Fei Wang, Yan Huo, and Xiaotian Wang. “A hybrid algorithm of grey wolf optimizer and harris hawks optimization for solving global optimization problems with improved convergence performance”. In: *Scientific Reports* 13 (2023), p. 22909. DOI: 10.1038/s41598-023-49754-2.

[13] Mingyang Yu, Jing Xu, Weiyun Liang, Yu Qiu, Sixu Bao, and Lin Tang. “Improved multi-strategy adaptive Grey Wolf Optimization for practical engineering applications and high-dimensional problem solving”. In: *Artificial Intelligence Review* 57 (2024), p. 277. DOI: 10.1007/s10462-024-10821-3.

[14] Yaqeen Saad Ali, Sura Mahroos Searan, Rihab Hazim Qasim, and Salah Awad Aliesawi. "Hyperparameter Optimization for CNN, K-NN, and Decision Tree in Handwritten Digit Classification". In: *Ingénierie des Systèmes d'Information* 30.5 (2025), pp. 1201–1207. DOI: 10.18280/isi.300508.

[15] Dimas Trianda, Dedy Hartama, and Solikhun Solikhun. "Comparison of Hyperparameter Tuning Methods for Optimizing K-Nearest Neighbor Performance in Predicting Hypertension Risk". In: *JURNAL TEKNIK INFORMATIKA* 18.1 (2025), pp. 111–121. DOI: 10.15408/jti.v18i1.42260.

[16] Jeena Joseph and K. Kartheeban. "Visualizing the Full Spectrum Optimization of K-Nearest Neighbors From Data Preprocessing to Hyperparameter Tuning and K-Fold Validation for Cardiovascular Disease Prediction". In: *Informatica* 49.2 (2025). DOI: 10.31449/inf.v49i2.7774.

[17] Rasool Seyghaly, Jordi Garcia, Xavi Masip-Bruin, and Jovana Kuljanin. "SBNNR: Small-Size Bat-Optimized KNN Regression". In: *Future Internet* 16.11 (2024), p. 422. DOI: 10.3390/fi16110422.

[18] Jia Mian Tan, Haoran Liao, Wei Liu, Changjun Fan, Jincai Huang, Zhong Liu, and Junchi Yan. "Hyperparameter optimization: Classics, acceleration, online, multi-objective, and tools". In: *Mathematical Biosciences and Engineering* 21.6 (2024), pp. 6289–6335. DOI: 10.3934/mbe.2024275.

[19] scikit-learn developers. *KNeighborsClassifier–scikit-learn documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. Accessed: 2026-04-30. 2026.

[20] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Accessed: 2026-04-30. Springer, 2009. DOI: 10.1007/978-0-387-84858-7. https://link.springer.com/book/10.1007/978-0-387-84858-7.

[21] MathWorks. *bayesopt — Select optimal machine learning hyperparameters using Bayesian optimization (Documentation)*. https://www.mathworks.com/help/stats/bayesopt.html. Accessed: 2026-04-29.

[22] Yagyanath Rimal, Navneet Sharma, and Abeer Alsadoon. "The accuracy of machine learning models relies on hyperparameter tuning: student result classification using random forest, randomized search, grid search, bayesian, genetic, and optuna algorithms". In: *Multimedia Tools and Applications* 83 (2024), pp. 74349–74364. DOI: 10.1007/s11042-024-18426-2.

[23] MathWorks. *Surrogate Optimization Algorithm (Documentation)*. https://www.mathworks.com/help/gads/surrogate-optimization-algorithm.html. Accessed: 2026-04-29.

[24] MathWorks. *surrogateopt — Surrogate optimization for global minimization (Documentation)*. https://www.mathworks.com/help/gads/surrogateopt.html. Accessed: 2026-04-29.

[25] Jicheng Yao, Xiaonan Luo, Fang Li, Ji Li, Jundi Dou, and Hongtai Luo. "Research on hybrid strategy Particle Swarm Optimization algorithm and its applications". In: *Scientific Reports* 14 (2024), p. 24928. DOI: 10.1038/s41598-024-76010-y.

[26] Mingyang Yu, Jing Xu, Weiyun Liang, Yu Qiu, Sixu Bao, and Lin Tang. "Improved multi-strategy adaptive Grey Wolf Optimization for practical engineering applications and high-dimensional problem solving". In: *Artificial Intelligence Review* 57 (2024), p. 277. DOI: 10.1007/s10462-024-10821-3.

[27] UCI Machine Learning Repository. *Heart Disease (Dataset)*. Tech. rep. Accessed: 2026-04-30. University of California, Irvine, 1988. https://archive.ics.uci.edu/ml/datasets/heart%20disease.

[28] UCI Machine Learning Repository. *Ionosphere (Dataset)*. Tech. rep. Accessed: 2026-04-30. University of California, Irvine, 1988. https://archive.ics.uci.edu/ml/datasets/Ionosphere.

[29] UCI Machine Learning Repository. *Spambase (Dataset)*. Tech. rep. Accessed: 2026-04-30. University of California, Irvine, 1999. https://archive.ics.uci.edu/ml/datasets/Spambase.

[30] UCI Machine Learning Repository. *Haberman's Survival (Dataset)*. Tech. rep. Accessed: 2026-04-30. University of California, Irvine, 1999. https://archive.ics.uci.edu/dataset/43/haberman+s+survival.

[31] OpenML. *blood-transfusion-service-center (OpenML Dataset 1464)*. Tech. rep. Accessed: 2026-04-30. OpenML, 2015. https://www.openml.org/d/1464.

[32] UCI Machine Learning Repository. *Connectionist Bench (Sonar, Mines vs. Rocks) (Dataset)*. Tech. rep. Accessed: 2026-04-30. University of California, Irvine, 1988. https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar%2C+Mines+vs.+Rocks%29.

[33] UCI Machine Learning Repository. *Breast Cancer (Ljubljana) (Dataset)*. Tech. rep. Accessed: 2026-04-30. University of California, Irvine, 1988. https://archive.ics.uci.edu/dataset/14/breast+cancer.

[34] mlbench package authors. *mlbench: Machine Learning Benchmark Problems — PimaIndiansDiabetes*. Accessed: 2026-04-30. 2026. https://search.r-project.org/CRAN/refmans/mlbench/html/PimaIndiansDiabetes.html.

[35] UCI Machine Learning Repository. *Iris (Dataset)*. Tech. rep. Accessed: 2026-04-30. University of California, Irvine, 1988. https://archive.ics.uci.edu/ml/datasets/Iris.

[36] UCI Machine Learning Repository. *Wine (Dataset)*. Tech. rep. Accessed: 2026-04-30. University of California, Irvine, 1991. https://archive.ics.uci.edu/ml/datasets/wine.

[37] UCI Machine Learning Repository. *Statlog (Landsat Satellite) (Dataset)*. Tech. rep. Accessed: 2026-04-30. University of California, Irvine, 1993. https://archive.ics.uci.edu/dataset/146/statlog+landsat+satellite.

[38] UCI Machine Learning Repository. *Glass Identification (Dataset)*. Tech. rep. Accessed: 2026-04-30. University of California, Irvine, 1987. https://archive.ics.uci.edu/ml/datasets/Glass+Identification.

[39] UCI Machine Learning Repository. *Wine Quality (Dataset)*. Tech. rep. Accessed: 2026-04-30. University of California, Irvine, 2009. https://archive.ics.uci.edu/dataset/186/wine+quality.

[40] UCI Machine Learning Repository. *Yeast (Dataset)*. Tech. rep. Accessed: 2026-04-30. University of California, Irvine, 1996. https://archive.ics.uci.edu/ml/datasets/Yeast.

[41] Janez Demšar. "Statistical Comparisons of Classifiers over Multiple Data Sets". In: *Journal of Machine Learning Research* 7.1 (2006), pp. 1–30. https://jmlr.org/papers/volume7/demsar06a/demsar06a.pdf.

[42] Sture Holm. "A Simple Sequentially Rejective Multiple Test Procedure". In: *Scandinavian Journal of Statistics* 6.2 (1979), pp. 65–70. https://www.jstor.org/stable/4615733.

[43] Purba Daru Kusuma and Meta Kallista. "Stochastic Komodo Algorithm." In: *International Journal of Intelligent Engineering & Systems* 15.4 (2022).

[44] Purba Daru Kusuma and Meta Kallista. "Swarm Space Hopping Algorithm: A Swarm-based Stochastic Optimizer Enriched with Half Space Hopping Search." In: *International Journal of Intelligent Engineering & Systems* 17.2 (2024).

[45] Purba Daru Kusuma and Meta Kallista. "Migration-Crossover Algorithm: A Swarm-based Metaheuristic Enriched with Crossover Technique and Unbalanced Neighbourhood Search." In: *International Journal of Intelligent Engineering & Systems* 17.1 (2024).

Table 8: KNN hyperparameter tuning results across datasets.

Dataset	Hyperparameter Tuning				Acc.	Macro AUC	CV Loss	CE Loss	Runtime (s)
	Tuning Method	k	m	p					
glass (UCI)									
	Bayesian Optimization	1	cityblock	4.927	0.785714	0.805225	0.313953	5.920933	9.1264
	Genetic Algorithm	4	cityblock	2.5794	0.785714	0.950962	0.30814	1.647074	6.9579
	Grey Wolf Optimizer	1	cityblock	2.6962	0.785714	0.805225	0.313953	5.920933	1.8447
	Grid Search	1	euclidean	2	0.785714	0.827557	0.313953	5.920933	7.5469
	PSO	10	cityblock	4.1832	0.738095	0.924303	0.331395	1.814474	1.8554
	Random Search	1	euclidean	2.8343	0.785714	0.827557	0.313953	5.920933	9.4207
	Surrogate Optimization	1	euclidean	1	0.785714	0.827557	0.313953	5.920933	5.8777
blood transfusion service center (openML)									
	Bayesian Optimization	23	euclidean	4.9781	0.785235	0.675313	0.186978	0.521779	8.4883
	Genetic Algorithm	19	minkowski	3.8702	0.771812	0.65589	0.186978	0.539072	7.3635
	Grey Wolf Optimizer	23	euclidean	3.2678	0.785235	0.675313	0.186978	0.521779	1.804
	Grid Search	19	minkowski	4	0.771812	0.657895	0.185309	0.534433	7.8483
	PSO	27	cityblock	4.2026	0.778523	0.681955	0.193656	0.514938	1.8025
	Random Search	27	cityblock	4.2026	0.778523	0.681955	0.193656	0.514938	9.6559
	Surrogate Optimization	23	euclidean	1.75	0.785235	0.675313	0.186978	0.521779	5.6342
breast cancer (UCI)									
	Bayesian Optimization	31	chebychev	1.0093	0.649123	0.576049	0.209607	0.581509	8.7999
	Genetic Algorithm	7	minkowski	4.2296	0.719298	0.559441	0.200873	1.893002	6.9696
	Grey Wolf Optimizer	31	chebychev	2.9914	0.649123	0.576049	0.209607	0.581509	1.7127
	Grid Search	7	minkowski	4	0.719298	0.564685	0.200873	1.889804	7.291
	PSO	35	chebychev	2.0965	0.719298	0.59965	0.213974	0.579112	1.7021
	Random Search	35	chebychev	2.0965	0.719298	0.59965	0.213974	0.579112	8.978
	Surrogate Optimization	7	minkowski	3.5	0.701754	0.558566	0.20524	1.894851	5.4701
diabetes (Kaggle)									
	Bayesian Optimization	34	cityblock	1.0489	0.718954	0.782358	0.222764	0.698335	9.0137
	Genetic Algorithm	30	cityblock	2.2986	0.718954	0.78283	0.219512	1.008101	8.2474
	Grey Wolf Optimizer	29	cityblock	1	0.712418	0.786887	0.227642	1.003894	2.2231
	Grid Search	29	cityblock	2	0.712418	0.786887	0.227642	1.003894	9.0976
	PSO	34	cityblock	1.4315	0.718954	0.782358	0.222764	0.698335	2.2285
	Random Search	24	cityblock	1.1159	0.718954	0.770377	0.229268	1.013449	11.2724

Continued on next page

Dataset	Hyperparameter Tuning				Acc.	Macro AUC	CV Loss	CE Loss	Runtime (s)
	Tuning Method	k	m	p					
	Surrogate Optimization	30	cityblock	2.25	0.718954	0.78283	0.219512	1.008101	6.2854
haberman (UCI)									
	Bayesian Optimization	23	cityblock	1.0018	0.737705	0.723611	0.232653	0.526902	8.6729
	Genetic Algorithm	22	euclidean	1.8902	0.737705	0.692361	0.232653	0.530267	7.2266
	Grey Wolf Optimizer	22	euclidean	3.2227	0.737705	0.692361	0.232653	0.530267	1.7798
	Grid Search	23	cityblock	2	0.737705	0.723611	0.232653	0.526902	7.4038
	PSO	18	chebychev	3.8841	0.803279	0.688889	0.240816	0.511029	1.771
	Random Search	30	euclidean	4.1485	0.737705	0.696528	0.240816	0.528322	9.1177
	Surrogate Optimization	22	euclidean	1.733	0.737705	0.692361	0.232653	0.530267	5.3579
heart diseases (UCI)									
	Bayesian Optimization	1	euclidean	1.9174	1	1	0.02439	0	8.6825
	Genetic Algorithm	1	euclidean	1	1	1	0.02439	0	7.6083
	Grey Wolf Optimizer	1	euclidean	1.4172	1	1	0.02439	0	1.9106
	Grid Search	1	euclidean	2	1	1	0.02439	0	9.4949
	PSO	1	cityblock	2.6762	1	1	0.028049	0	1.8617
	Random Search	1	euclidean	1.4695	1	1	0.02439	0	11.6496
	Surrogate Optimization	1	euclidean	1	1	1	0.02439	0	5.2919
ionosphere (UCI)									
	Bayesian Optimization	1	cityblock	4.927	0.942857	0.928889	0.120996	1.578915	8.4956
	Genetic Algorithm	1	cityblock	3.3734	0.942857	0.928889	0.120996	1.578915	6.9875
	Grey Wolf Optimizer	1	minkowski	4.4316	0.885714	0.857778	0.145907	3.157831	1.8471
	Grid Search	1	cityblock	2	0.942857	0.928889	0.120996	1.578915	7.8567
	PSO	1	minkowski	4.4316	0.885714	0.857778	0.145907	3.157831	1.8034
	Random Search	1	minkowski	4.4316	0.885714	0.857778	0.145907	3.157831	9.6673
	Surrogate Optimization	1	euclidean	1	0.885714	0.857778	0.156584	3.157831	5.4872
iris (UCI)									
	Bayesian Optimization	9	euclidean	1.149	0.966667	0.995	0.016667	0.138808	8.2911
	Genetic Algorithm	20	euclidean	4.6102	0.866667	0.98	0.025	0.233425	6.6059
	Grey Wolf Optimizer	18	euclidean	2.5117	0.9	0.985	0.025	0.215713	1.7129
	Grid Search	9	euclidean	2	0.966667	0.995	0.016667	0.138808	7.0466
	PSO	18	euclidean	2.5117	0.9	0.985	0.025	0.215713	1.6776
	Random Search	18	euclidean	2.5117	0.9	0.985	0.025	0.215713	8.7036

Continued on next page

Dataset	Hyperparameter Tuning				Acc.	Macro AUC	CV Loss	CE Loss	Runtime (s)
	Tuning Method	k	m	p					
	Surrogate Optimization	15	minkowski	3.0966	0.966667	0.993333	0.025	0.187217	5.1815
satellite image (UCI)									
	Bayesian Optimization	5	minkowski	1.0079	0.90902	0.980816	0.097589	0.643869	50.7098
	Genetic Algorithm	5	cityblock	2.0943	0.909798	0.981074	0.096812	0.623981	88.3349
	Grey Wolf Optimizer	3	cityblock	2.8823	0.90591	0.974879	0.097589	0.929217	19.2838
	Grid Search	5	cityblock	2	0.909798	0.981074	0.096812	0.623981	164.5119
	PSO	3	cityblock	2.8823	0.90591	0.974879	0.097589	0.929217	18.1452
	Random Search	3	cityblock	4.1606	0.90591	0.974879	0.097589	0.929217	191.1661
	Surrogate Optimization	5	cityblock	1.1984	0.909798	0.981074	0.096812	0.623981	21.0704
sonar (Kaggle)									
	Bayesian Optimization	1	cityblock	4.892	0.756098	0.751196	0.125749	6.739273	8.3351
	Genetic Algorithm	5	euclidean	2.9974	0.658537	0.782297	0.131737	1.154825	7.09
	Grey Wolf Optimizer	1	cityblock	1	0.756098	0.751196	0.125749	6.739273	2.0083
	Grid Search	1	cityblock	2	0.756098	0.751196	0.125749	6.739273	7.6448
	PSO	5	cityblock	1.4361	0.634146	0.789474	0.143713	0.508271	1.7776
	Random Search	5	cityblock	1.4361	0.634146	0.789474	0.143713	0.508271	9.3896
	Surrogate Optimization	5	euclidean	1	0.658537	0.782297	0.131737	1.154825	5.6253
spambase (UCI)									
	Bayesian Optimization	7	cityblock	1.5146	0.922826	0.963222	0.090465	0.524793	20.2506
	Genetic Algorithm	7	cityblock	3.5915	0.922826	0.963222	0.090465	0.524793	59.7536
	Grey Wolf Optimizer	1	minkowski	2.7529	0.903261	0.896478	0.09291	2.673001	15.8436
	Grid Search	7	cityblock	2	0.922826	0.963222	0.090465	0.524793	88.1614
	PSO	7	cityblock	3.0037	0.922826	0.963222	0.090465	0.524793	14.8831
	Random Search	1	minkowski	2.7529	0.903261	0.896478	0.09291	2.673001	105.0789
	Surrogate Optimization	7	euclidean	1	0.905435	0.951895	0.093725	0.65453	13.7806
wine (UCI)									
	Bayesian Optimization	17	minkowski	1.9028	0.914286	1	0.027972	0.148397	8.2991
	Genetic Algorithm	18	euclidean	2.3698	0.914286	0.998889	0.027972	0.152227	6.6811
	Grey Wolf Optimizer	16	euclidean	4.6052	0.914286	0.998813	0.027972	0.155937	1.675
	Grid Search	3	euclidean	2	0.942857	1	0.027972	0.085947	7.138
	PSO	16	euclidean	4.6052	0.914286	0.998813	0.027972	0.155937	1.6664
	Random Search	16	euclidean	4.6052	0.914286	0.998813	0.027972	0.155937	8.8212

Continued on next page

Dataset	Hyperparameter Tuning				Acc.	Macro AUC	CV Loss	CE Loss	Runtime (s)
	Tuning Method	k	m	p					
	Surrogate Optimization	30	`euclidean`	2.4939	0.942857	0.998813	0.027972	0.16923	5.1955
winequality red (UCI)									
	Bayesian Optimization	1	`minkowski`	1.1601	0.642633	0.644657	0.394531	9.874409	8.7195
	Genetic Algorithm	1	`euclidean`	4.5071	0.633229	0.638819	0.397656	10.134262	8.7423
	Grey Wolf Optimizer	1	`minkowski`	1	0.636364	0.644283	0.400781	10.047644	2.1572
	Grid Search	1	`euclidean`	2	0.633229	0.638819	0.397656	10.134262	10.9455
	PSO	11	`euclidean`	2.222	0.561129	0.649095	0.401562	2.21446	2.1792
	Random Search	11	`euclidean`	2.222	0.561129	0.649095	0.401562	2.21446	13.555
	Surrogate Optimization	1	`euclidean`	1	0.633229	0.638819	0.397656	10.134262	5.7706
winequality white (UCI)									
	Bayesian Optimization	1	`cityblock`	3.0196	0.629213	0.647665	0.38122	10.24521	11.2209
	Genetic Algorithm	1	`euclidean`	1	0.629213	0.643205	0.383006	10.24521	21.0828
	Grey Wolf Optimizer	1	`minkowski`	2.4814	0.631256	0.647242	0.384792	10.188763	6.2658
	Grid Search	1	`cityblock`	2	0.629213	0.647665	0.38122	10.24521	36.4988
	PSO	17	`cityblock`	3.2274	0.560776	0.701414	0.448073	1.523006	5.1996
	Random Search	23	`cityblock`	2.1906	0.54239	0.696543	0.446798	1.419247	46.1109
	Surrogate Optimization	1	`euclidean`	1	0.629213	0.643205	0.383006	10.24521	8.1213
yeast (UCI)									
	Bayesian Optimization	18	`euclidean`	4.9785	0.601351	0.864772	0.393939	1.984875	10.0445
	Genetic Algorithm	18	`minkowski`	1.393	0.591216	0.864405	0.389731	1.983761	10.5252
	Grey Wolf Optimizer	18	`cityblock`	2.2601	0.608108	0.865615	0.399832	1.905662	2.8582
	Grid Search	21	`cityblock`	2	0.597973	0.867812	0.393939	1.670115	11.8685
	PSO	18	`cityblock`	2.2601	0.608108	0.865615	0.399832	1.905662	2.8185
	Random Search	18	`cityblock`	2.2601	0.608108	0.865615	0.399832	1.905662	14.8533
	Surrogate Optimization	18	`cityblock`	4.4582	0.608108	0.865615	0.399832	1.905662	6.9177