A MULTICLASS DECISION TREE WITH BALANCED TOPOLOGY VIA LABEL-CONSTRAINED K-MEANS GROUPING AND SVM LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

In multiclass scenarios, effectively performing classification tasks with decision tree models is of great importance in fields such as medical analysis, intrusion detection, and data mining. This task faces numerous challenges, including accurate classification on large datasets and in imbalanced scenarios. Existing decision tree methods have the following shortcomings: (1) the learned tree topology is not sufficiently compact and requires pruning to control tree depth; (2) node splitting is performed through multiple sequential steps and has not been incorporated into a unified mathematical framework; (3) tree node splitting is often measured by distance or splitting criteria such as Shannon entropy, information gain, and the Gini index, without fully considering the global structure and information of nodes. To address these issues, we propose SLtree, a multiclass decision tree that obtains a balanced topology via label-constrained K-means grouping and learns nonlinear decision boundaries using support vector machines. During decision tree construction, SLtree introduces the clustering principle of the K-means algorithm, fully leveraging global relationships among samples. Moreover, label constraints are imposed to constrain the tree topology to a structurally optimized balanced binary tree, thereby yielding a compact topology construction. Node splitting can be mathematically represented by a constrained formulation and does not require multiple steps. To probe the classification limits of SLtree, we introduce an improved particle swarm optimization algorithm to individually optimize the parameters of each splitting node, yielding an improved variant of the algorithm, ISLtree. Experimental evaluation on 41 datasets demonstrates that our SLtree algorithm achieves a leading ranking among the compared algorithms, while its improved variant, ISLtree, significantly outperforms other methods. Moreover, ISLtree exhibits outstanding performance on imbalanced datasets and datasets with high classification difficulty, and it has clear advantages over other methods on datasets with large sample sizes.

1 INTRODUCTION

Decision trees, as a powerful tool in machine learning, have been widely applied in various domains such as medical analysis (Stasis et al., 2003; Kababulut et al., 2023; Wilson, 2023), image classification (Rodríguez et al., 2024; Oumaima et al., 2023), signal classification (Reddy & Priyanka, 2024), speech emotion recognition (Sun et al., 2019), intrusion detection (Benferhat et al., 2013), and data mining (Li et al., 2015). Owing to their high classification performance and concise, clear topological structures, numerous studies (Montañana et al., 2021; Yildiz & Alpaydin, 2005; Xian et al., 2024; Kantavat et al., 2022) have employed decision trees for multi-class classification tasks. Furthermore, due to the multi-layer discriminative propagation mechanism, decision trees are also regarded as hierarchical classifiers. Research on the fundamental architecture of decision trees is conducive to advancing their application and development across diverse fields.

Node splitting is central to decision trees, which are either binary or non-binary by topology. Traditional binary splitting uses measures such as Shannon entropy, information gain, Gini index, and

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

gain ratio (Montañana et al., 2021; Zhao et al., 2023; Yang et al., 2019; Breiman et al., 1984), or trains a binary classifier (e.g., SVM) (Kantavat et al., 2022) or uses class-center distances (Xian et al., 2024). For multi-class binary trees, class grouping can use One-vs-One (OvO), One-vs-Rest (OvR), or Many-vs-Many (MvM). With L classes, OvO trains $\mathrm{C}_L^2 = L(L-1)/2$ classifiers on class pairs and predicts by voting. OvR trains L classifiers, each separating one class from the rest, and selects the highest probability. With time T per binary problem, OvO and OvR have complexities $T \cdot L(L-1)/2$ and $T \cdot L$, respectively. MvM generalizes both: it assigns several classes to the positive side and several to the negative side; OvO is $M_1 = 1, M_2 = 1$, OvR is $M_1 = 1, M_2 = L-1$. SLtree uses MvM with $M_1 = \mathrm{round}((L-1)/2)$, $M_2 = L - M_1$; e.g., 7 classes split into 3 and 4, 10 into two groups of 5. Some methods use unsupervised K-means for multi-way splitting (Wang et al., 2020). However, the aforementioned traditional node splitting strategies often determine the partition roughly based on a single measure and fail to effectively exploit the similarity correlations among node samples. Moreover, most of them cannot determine the class grouping at a node in one step, and require multi-step search, class-by-class merging, or combining multiple binary classification results followed by voting or selection to obtain the final class partition. Although unsupervised iterative clustering-based node splitting takes sample correlations into account, it does not fully exploit the known label information, which may lead to underfitting.

To address the problems encountered in existing multi-class decision tree research, this study proposes a balanced-topology multi-class decision tree based on label-constrained K-means grouping and SVM learning, termed the **SLtree** algorithm, for multi-class classification tasks in general scenarios. During node grouping, it fully learns the global relationships among sample attribute values while also making full use of the label information provided by supervised learning. At the same time, the entire grouping process is unified into a mathematical optimization formulation, and the grouping is completed by solving this formulation.

The innovations of SLtree mainly lie in the learning of the decision tree structure:

- First, at each node to be split, we model the node partitioning problem as a label-constrained K-means-style optimization problem. This problem is solved by alternating iteration, and each subproblem is convex or has a closed-form solution, thereby directly yielding the class grouping matrix G. The final splitting result to be learned at that node is obtained without multi-step search or pairwise voting.
- Second, a class grouping balance constraint is introduced into the optimization problem, which controls the tree structure to be a balanced and compact binary tree while splitting nodes, without requiring any pruning operations to control tree depth or model complexity.
- Third, an SVM binary classifier is trained at each node to be split to learn a nonlinear decision boundary, adapting to complex data distributions.
- Furthermore, to further explore the multi-class performance limits of SLtree, and considering that each non-leaf node to be learned has distinct sample distribution characteristics, a swarm intelligence algorithm is used to search for an optimal pair of parameters (C^*, γ^*) when learning the SVM. Swarm intelligence is a distributed computing paradigm in bionics that studies how large numbers of simple individuals, through local interactions and cooperation, give rise to complex intelligent behaviors that transcend individual capabilities, featuring distributed control and decentralization (Tian et al., 2024; Kumar et al., 2023; 2020; Kennedy & Eberhart, 1995). Many excellent swarm intelligence algorithms have been proposed, such as artificial bee colony (Karaboga, 2005), ant colony optimization (Dorigo, 1992), firefly algorithm (Yang, 2009), and particle swarm optimization (Kennedy & Eberhart, 1995). Benefiting from its fast convergence and ease of implementation, particle swarm optimization has been widely applied across various domains (Kohler et al., 2019; Tian, 2018; Xue et al., 2014; Wan et al., 2013; Das et al., 2016). We complete the search using an improved particle swarm optimization algorithm (Shi & Eberhart, 1999; Sharma et al., 2026; Shi & Eberhart, 1998) and name this algorithm **ISLtree**. It is also used for multi-class classification tasks in general scenarios, and is particularly suitable for imbalanced scenarios, complex classification tasks, situations with high requirements for interpretability, or resource-constrained settings (limited memory and computing power).

In the remainder of this paper, we review recent studies on hierarchical classifiers in Section 2 and detail the main components of our proposed method in Section 3. Then, in Section 4, we describe

the experiments conducted to evaluate the proposed algorithm, involving extensive datasets and benchmarking against competing algorithms. Finally, our conclusions are presented in Section 5.

2 Related Work

Top-down decision tree construction recursively splits nodes from the root until a stopping condition is met, turning the node into a leaf. Node splitting strategies critically affect classification performance and are categorized into supervised and unsupervised ones based on whether label information is used. Supervised strategies include splitting directly by labels, using labels as reference, or training an initial classifier.

For direct label-based splitting, Linear Discriminant Trees (LDT) (Yildiz & Alpaydin, 2005) transform multi-class problems into binary ones via outer optimization of class grouping using selection and exchange methods. NDT-IFTSVM (Xian et al., 2024) can be viewed as an MVM strategy: it computes class centers, selects the farthest pair as initial positive/negative classes, and assigns remaining classes by distance to these centers. Other label-based criteria use measures such as information gain, Gini index, Shannon entropy, and gain ratio. Weighted Oblique Decision Trees (WODT) (Yang et al., 2019) locally optimize split parameters at each internal node using weighted information entropy as the objective. BoostTree (Zhao et al., 2023) uses gradient descent-based gain information maximization. MOCTSVM (Blanco et al., 2023) temporarily disregards class labels, using dynamically generated fictitious binary classifications to construct SVM hyperplanes, with both hyperplane parameters and fictitious labels optimized globally.Among unsupervised mechanisms, BDTKS (Wang et al., 2020) introduces K-means into node splitting (with $k = 2$ for binary trees), partitioning samples via intra-cluster distance minimization without labels. GMDT (Wang et al., 2026) replaces K-means with G-means (Hamerly & Elkan, 2003), enabling adaptive multi-way splitting without pre-specifying K.

Existing methods that rely on sample labels to indicate sample destinations or on splitting criteria typically require multiple steps to accomplish node splitting, making it difficult to integrate them into a unified mathematical framework and impossible to characterize the entire splitting process with a single formula; unsupervised node splitting mechanisms, in contrast, fail to fully exploit label information. SLtree incorporates class label constraints into the optimization framework of K-means, elevating "sample clustering" to "class grouping," thereby preserving the global structure discovery capability of unsupervised learning while injecting supervisory information. Furthermore, node splitting is modeled as a constrained optimization problem, and the optimal grouping of class labels is directly obtained by solving this formulation. Section 3 provides a detailed description of the implementation of SLtree.

3 Proposed method

3.1 A Balanced Binary Decision Tree Based on Clustering: SLtree

In this section, we propose a multi-class decision tree with balanced topology based on label-constrained K-means grouping and SVM learning, termed SLtree. Unlike BDTKS (Wang et al., 2020), which directly splits using K-means, SLtree seeks a decision boundary at each node via SVM. Since clustering alone ignores labels and yields imbalanced, irregular trees with excessive depth, SLtree does not train SVM directly on the K-means clusters. Instead, it minimizes the distance between each sample and one of the cluster centers, and under label supervision and tree-balance constraints, solves for a partition indicator matrix to split node samples into left and right subtrees. The resulting balanced binary tree is then used to train an SVM decision boundary at each node for multi-class classification.

The general procedure is illustrated in Figure 1. The constructed decision tree is a balanced binary tree. Each non-leaf node stores the grouping matrix G for its class labels, and each sample x obtains a binary label y_{binary} from G and its class label y. Each leaf node stores the assigned class label l^*. For prediction, a new sample x enters the root node. The SVM at the root, S_1, directs it to the left or right child. At each subsequent node, the stored SVM classifier continues routing until a leaf node is reached, where the sample is assigned the stored class label l^*.

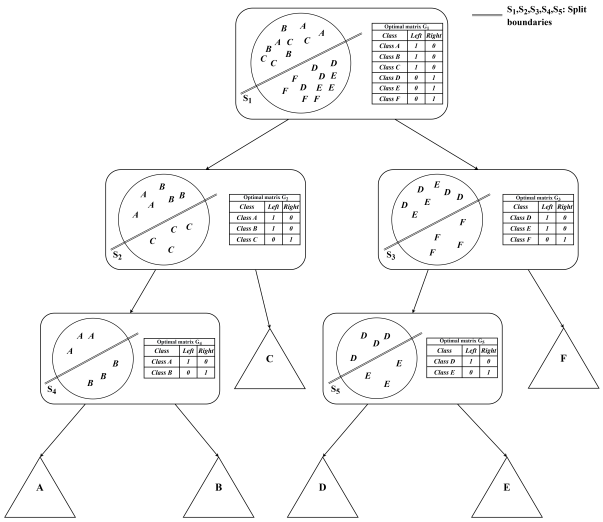

Figure 1: Illustration of the SLtree construction process on a six-class dataset (classes A–F). At each non-leaf node, an optimal grouping matrix G is learned under label constraints, which assigns each class to the left or right child, and an SVM classifier (split boundary S_i) is trained accordingly. The tree is recursively split until leaf nodes corresponding to individual class labels are obtained.

We now detail how the optimal grouping matrix G is obtained via label-constrained K-means. Consider a node to be split (including the root). Let the training samples entering this node be $X = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq \mathbb{R}^{(d+1)\times n}$, with index set $N = \{1, \ldots, n\}$, where $x_i \in \mathbb{R}^{d\times 1}$, $y_i \in \{1, \ldots, l\}$, and l is the total number of labels.

BDTKS uses unsupervised K-means to cluster samples into two clusters, exploiting intra-cluster compactness, which aligns with the requirement that similar samples be grouped together. This is formulated as:

$$\min_{C_1, C_2, \ldots, C_K} \sum_{i=1}^{K} \sum_{x \in C_i} \|x - c_i\|_2^2 \tag{3.1}$$

where c_i is the mean of cluster C_i, and $K = 2$ for BDTKS. However, this unsupervised method ignores label information. In contrast, NDT-IFTSVM (Xian et al., 2024) uses class-center distances and class sizes, seeking the two farthest classes to construct the tree.

Inspired by both BDTKS and NDT-IFTSVM, our method incorporates K-means clustering into tree construction, where the two groups after splitting should be close to their target centroids, capturing global data characteristics and enhancing generalization. Like NDT-IFTSVM, we determine sample assignment to left or right subtrees directly by class labels. However, unlike NDT-IFTSVM, which relies on class-center distances and multiple steps, we directly solve a minimization problem to obtain the partition. At each non-leaf node, for training samples x_i with labels $y_i \in \{1, \ldots, l\}$, the binary label y_{binary} is obtained via a label-constrained bipartitioning procedure. Specifically, we compute a balanced group size $h = \text{round}((l-1)/2)$, generate all possible grouping matrices $G \in \mathbb{R}^{l\times 2}$ with exactly h ones in the first column, initialize centroids via PCA, and for each G construct the label indicator matrix L, compute $F = LG$, reconstruct data as $\hat{X} = CF^\top$, and select the G^* minimizing the reconstruction error $\|X - \hat{X}\|_F$. We then iteratively optimize C and G until convergence and create binary labels $y_{\text{binary}} = \text{argmax}(F, \text{axis} = 1)$. This yields the optimal grouping matrix G and completes the node split.

We initialize two centroids $\mathbf{c}_1, \mathbf{c}_2 \in \mathbb{R}^{d\times 1}$ via PCA: project data onto the first principal component $\mathbf{w}$, split at the median projection value p_{median}, and compute the means of the two subsets as the initial centroids:

$$\begin{cases} X_{p1} = \{x_i \mid \langle x_i, \mathbf{w}\rangle > p_{\text{median}}\} \\ X_{p2} = \{x_i \mid \langle x_i, \mathbf{w}\rangle \leq p_{\text{median}}\} \end{cases} \tag{3.2}$$

Let $C = [\mathbf{c}_1, \mathbf{c}_2] \in \mathbb{R}^{d\times 2}$. For node data X with l classes, SLtree minimizes the distance between each sample x_i and its centroid, yielding a partition indicator matrix $F \in \mathrm{Ind}^{n\times 2}$ that splits samples into X_l and X_r:

$$F(i,j) = \begin{cases} 1, & \text{if } x_i \in X_l \text{ and } j = 1, \text{ or } x_i \in X_r \text{ and } j = 2, \\ 0, & \text{otherwise,} \end{cases} \quad i = 1, \ldots, n;\ j = 1, 2. \tag{3.3}$$

Extending K-means, we formulate:

$$\min_{C,F} \|X - CF^T\|_F^2, \tag{3.4}$$

where $\|\cdot\|_F$ is the Frobenius norm.

To incorporate label information, let $L \in \mathrm{Ind}^{n\times l}$ be the label indicator matrix:

$$L(i,j) = \begin{cases} 1, & \text{if } y_i = j, \\ 0, & \text{otherwise,} \end{cases} \quad i = 1, \ldots, n;\ j = 1, \ldots, l. \tag{3.5}$$

Since SLtree partitions node data by class labels, similar to NDT-IFTSVM (Xian et al., 2024), we introduce a label-to-group assignment matrix $G \in \mathrm{Ind}^{l\times 2}$ such that:

$$F = LG. \tag{3.6}$$

The optimization becomes:

$$\min_{C,L,G} \|X - CG^T L^T\|_F^2 \tag{3.7}$$

subject to:

$$\mathbf{1}^T G = [\, h,\ l - h\,], \quad h = \left\lfloor \frac{l-1}{2} \right\rceil, \tag{3.8}$$

where $\lfloor\cdot\rceil$ denotes rounding to the nearest integer. This constraint ensures a balanced tree topology, i.e., the height difference between left and right subtrees of any non-leaf node does not exceed 1. The matrix G can also be written element-wise as:

$$G(i,j) = \begin{cases} 1, & \text{if } y_i = i \text{ (class index), } x_i \in X_l \text{ and } j = 1, \text{ or } x_i \in X_r \text{ and } j = 2, \\ 0, & \text{otherwise,} \end{cases} \tag{3.9}$$

where $i = 1, \ldots, l$ and $j = 1, 2$.

Solving this model yields the optimal G, which determines the node split and constructs the tree.

After the label-constrained bipartitioning, the training data are partitioned and the binary labels are stored at the corresponding nodes. Then, support vector machines are employed to train a binary classifier at each split node (Figure 1), constructing nonlinear decision boundaries for hierarchical classification.

We define nodes at the same depth as belonging to the same level. For the balanced binary tree constructed by SLtree, let the root be level 0 with training samples $\mathbf{X}_0 = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq \mathbb{R}^{(d+1)\times n}$ and labels $y_i \in \{1, 2, \ldots, L\}$, where L is the total number of classes. Owing to the constraint $\mathbf{1}^T G = [h, l-h]$ with $h = [(l-1)/2]$, the tree has $D = \lceil \log_2 L \rceil$ levels, i.e., $D+1$ levels in total (Figure 2). Let $\mathcal{U} = \{u_0, u_1, \ldots, u_D\}$ denote the level set, where u_i is the set of nodes at level i, $i \in \{0, 1, \ldots, D\}$.

The root is u_0, and u_D is a possibly incomplete set of leaf nodes. The total number of SVM classifiers is $L-1$, denoted by $S = \{s_1, s_2, \ldots, s_{L-1}\}$. As shown in Figure 2, level u_0 contains one classifier s_1, level u_1 contains two classifiers s_2 and s_3, and so on. Level u_D contains no classifiers, while u_{D-1} contains at least one. In general, level u_i contains 2^i classifiers, where $0 \le i \le D-2$. Each SVM has two core parameters, the regularization parameter C and the kernel parameter γ, and after optimization, each s_i is unique, as explained below.

3.2 SVM Parameter Optimization via Particle Swarm Optimization

When using an RBF support vector machine for binary classification, its penalty parameter C and kernel parameter γ often have a considerable impact on the classification result, i.e., they exhibit hyperparameter sensitivity. For a decision tree with an SVM at each non-leaf node, uniformly applying

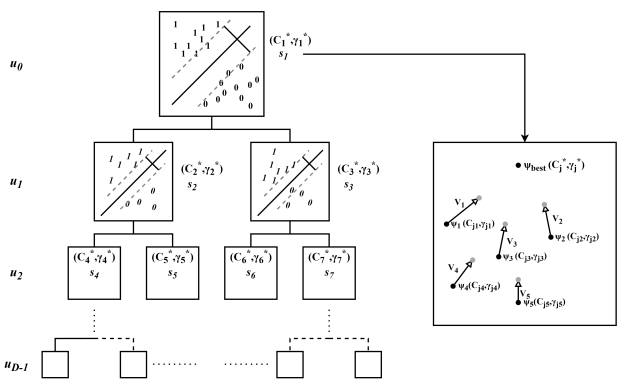

Figure 2: Level structure of the balanced binary decision tree constructed by SLtree, showing SVM classifiers at each non-leaf node. From the root level u_0 to the bottom level u_{D-1}, each node s_i is associated with an optimized SVM classifier parameterized by (C_i^*, γ_i^*). The right panel illustrates the improved PSO-based parameter optimization process for each node's SVM classifier.

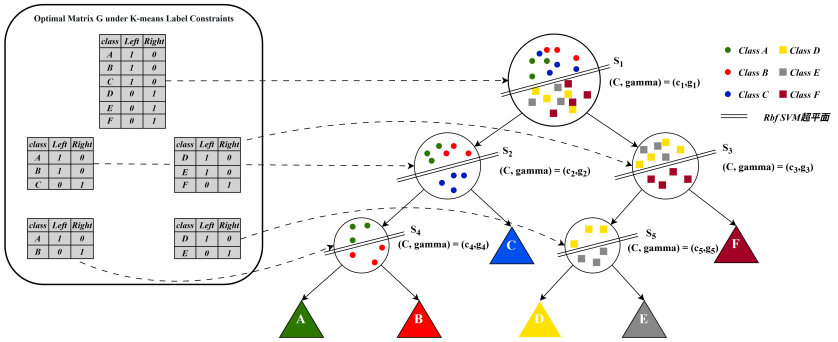

Figure 3: Overview of the ISLtree model construction on a six-class dataset. Each non-leaf node deploys an SVM classifier with independently optimized parameters $(c_1, g_1), \ldots, (c_5, g_5)$, and each leaf node stores one of the class labels A, B, C, D, E, F.

the same parameters cannot adapt well to the binary classification problem at each node. To address this, the proposed ISLtree method employs an Improved Particle Swarm Optimization (IPSO) algorithm to separately search for the optimal parameters (C, γ) for each node. The general procedure is illustrated in Figure 3, where each non-leaf node SVM possesses different (C, γ) parameters.

Consider a node with an SVM classifier to be optimized. Suppose the training samples entering this node are $X = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subseteq \mathbb{R}^{(d+1)\times n}$, with temporary binary labels y_{binary}. The training set for the node classifier s_j ($j \in \{1, \ldots, L-1\}$) is $T = \{(x_1, y_{b1}), \ldots, (x_n, y_{bn})\} \subseteq \mathbb{R}^{(d+1)\times n}$, where $x_i \in \mathbb{R}^{d\times 1}$, $y_{bi} \in \{0, 1\}$, $i = 1, \ldots, n$. The parameters to be optimized for s_j are (C_j, γ_j), and the corresponding hyperparameter set for $S = \{s_1, \ldots, s_{L-1}\}$ is $H = \{(C_1, \gamma_1), \ldots, (C_{L-1}, \gamma_{L-1})\}$. For a pair (C_j, γ_j) with $C_j \in [C_a, C_b]$ and $\gamma_j \in [\gamma_a, \gamma_b]$, the search space is $\Psi = \{(C_j, \gamma_j) \mid C_j \in [C_a, C_b], \gamma_j \in [\gamma_a, \gamma_b]\}$.

Let $\psi_k \in \Psi$ be a particle in the swarm. According to the standard PSO algorithm (Kennedy & Eberhart, 1995), each particle $\psi_k = (C_{jk}, \gamma_{jk})$ has a velocity $V_k = (\upsilon_{k1}, \upsilon_{k2})$. The fitness is defined as the average accuracy on a validation set disjoint from the test set:

$$\text{Fitness}(\psi_k) = \frac{1}{N_{\text{val}}} \sum_{i=1}^{N_{\text{val}}} \delta(\hat{y}_{bi}, y_{bi}) \tag{3.10}$$

where $\hat{y}_{bi} = \text{SVM}(\psi_k, T_{\text{val}})$ and y_{bi} is the true binary label. The optimization proceeds in three stages: initialization, iterative update of particle positions and velocities, and convergence to the optimal parameters.

Specifically, the population size is P and the search dimension is 2. Initial particles are randomly generated as $\psi_i = \{(C_{ji}, \gamma_{ji}) \mid C_{ji} \in [C_a, C_b], \gamma_{ji} \in [\gamma_a, \gamma_b]\}$, $i = 1, \ldots, P$, with random velocities $V_i = (v_{i1}, v_{i2})$. Initial fitness is computed via equation 3.10, and the global best G_{best} and individual best P_{best} are initialized.

In each iteration, the best position of particle ψ_i is $P_{\text{best},i} = (P_{\text{best},i1}, P_{\text{best},i2})$, and the swarm best is $G_{\text{best}} = (G_{\text{best1}}, G_{\text{best2}})$. Particles update as:

$$V_i(t+1) = \theta(t) \cdot V_i(t) + c_1 r_1 \big(P_{\text{best},i}(t) - \psi_i(t)\big) + c_2 r_2 \big(G_{\text{best}}(t) - \psi_i(t)\big) \tag{3.11}$$

$$\psi_i(t+1) = \psi_i(t) + V_i(t+1) \tag{3.12}$$

where $t < t_{\max}$, c_1, c_2 are learning coefficients, and r_1, r_2 are uniform random values. Fitness is then evaluated to update G_{best} and P_{best}.

Two improvement strategies are employed. First, a linearly decreasing inertia weight θ (Shi & Eberhart, 1999) is used:

$$\theta(t) = \theta_{\max} - (\theta_{\max} - \theta_{\min}) \left(\frac{t}{t_{\max}} \right) \tag{3.13}$$

Second, a genetic mutation operation (Shi & Eberhart, 1998) randomly perturbs ψ_i with a certain probability to increase diversity and avoid local optima.

The optimization terminates when the maximum number of iterations or a preset precision is reached, yielding optimal parameters (C_j^*, γ_j^*) and the optimized set $H^* = \{(C_1^*, \gamma_1^*), \ldots, (C_{L-1}^*, \gamma_{L-1}^*)\}$. ISLtree thus performs node-wise SVM parameter selection via PSO, improving model performance. To mitigate the increased training time, especially on large datasets, an early stopping mechanism is added when no improvement occurs for several consecutive iterations, and stratified sampling preserving the original class distribution is used for large-scale datasets.

4 Experimental evaluation

This section details a series of extensive and rigorous comparative experiments conducted to demonstrate the advantages of our proposed methods, SLtree and ISLtree.All experiments were conducted using Python 3.10 on the PyCharm platform under Windows 10. In addition, MATLAB 24.2.0.2863752 (R2024b) Update 5 was used to train and test the TBSVM decision tree model. The hardware configuration is as follows: Intel(R) Xeon(R) CPU E5-1620 v3 @ 3.50 GHz, 3500 MHz, 4 cores, 8 logical processors, and 32.0 GB of memory.

To evaluate the classification performance of the proposed algorithms on multi-class datasets, we selected 41 benchmark datasets. Datasets 7–14 were adopted from Ricardo et al. (Montañana et al., 2025), all from the UCI Machine Learning Repository (35). Datasets 31–41 were obtained by ResNet feature extraction on vegetable pest and disease images. The remaining datasets come from UCI, KEEL, and Kaggle, and all were unified into libsvm format. These datasets range from 3 to 18 classes, with sizes from 101 to 32,485, covering balanced, imbalanced, and complex cases. Imbalance is measured by IR and Avg_IR, and classification complexity by the MFII (Han et al., 2025), a continuous value in [0,1] where higher values indicate greater difficulty. The Supplementary Material provides the complete Python implementations of SLtree and ISLtree together with the libsvm files of all datasets, supporting reproducibility and interpretability.

4.1 Overall performance and statistical analysis

To evaluate each (algorithm, dataset) pair, we conducted ten runs of five-fold cross-validation (10×5 CV). Datasets were preprocessed for direct 10×5 CV, and all methods were evaluated on the processed datasets. Due to dataset imbalance, accuracy, precision, recall, and F1-score were used. ISLtree was evaluated under the same conditions as SLtree. For other algorithms, we used publicly released implementations (Python for STree, BDTKS.v2, BoostTree, WODT; Matlab for TBSVM_Tree).

SLtree and ISLtree achieve the best performance in both accuracy and F1-score. ISLtree improves over SLtree, surpassing it in accuracy on 23 datasets.The average metrics across all datasets are

shown in Table 1. SLtree significantly outperforms all comparison methods except ISLtree, with average accuracy 0.85 percentage points higher than the second-best method (STree) and Macro F1 2.14 percentage points higher. ISLtree achieves the best average performance, exceeding all comparison methods by 3.31 to 24.96 percentage points in accuracy and by 2.90 to 35.79 percentage points in Macro F1.

Table 1: Average performance (mean ± std) of each method over all datasets. The best value in each column is highlighted in bold.

Method	Accuracy	Precision	Recall	Macro F1	Training Time (s)	Avg Prediction Time (ms)
OURS-ISLTree	**0.8562±0.1498**	**0.7964±0.1958**	**0.7799±0.2009**	**0.7793±0.2042**	319.4225±1154.2939	0.3205±0.6584
OURS-SLtree	0.8316±0.1696	0.7572±0.2310	0.7183±0.2427	0.7199±0.2446	30.1601±109.7368	1.5754±0.9969
STree	0.8231±0.1760	0.7337±0.2380	0.7007±0.2474	0.6985±0.2525	21.6990±72.6949	0.7497±1.7331
bdtks	0.7994±0.1898	0.7228±0.2319	0.7068±0.2388	0.7054±0.2442	0.9317±0.6563	0.0955±0.0684
BoostTree	0.7971±0.1950	0.7175±0.2152	0.7049±0.2117	0.6965±0.2165	**0.3833±0.6537**	**0.0269±0.0334**
TBSVM_Tree	0.6056±0.2460	0.4236±0.2964	0.4779±0.2736	0.4214±0.2925	25.4837±65.7229	0.6922±0.5096
WODT	0.7733±0.2080	0.7015±0.2190	0.6789±0.2345	0.6744±0.2385	4.1213±6.2832	0.0311±0.0068

We adopted standard statistical analysis procedures (Demšar, 2006; García & Herrera, 2008). First, the Friedman test ($\alpha = 0.05$) was conducted. If rejected, Holm's procedure was used for post-hoc testing ($\alpha = 0.05$), with the highest-ranked algorithm as control. For each dataset, the seven algorithms were ranked by accuracy, and average ranks across the 41 datasets were computed. The Friedman test p-value is 4.3×10^{-23} (far less than 0.05), indicating significant overall differences.

Tables 2 and 3 present the post-hoc test results with SLtree and ISLtree as control, respectively. With SLtree as control, it significantly outperforms all comparison algorithms (STree, BoostTree, BDTKS, WODT, TBSVM_Tree) in accuracy, with highly significant differences from the latter three. With ISLtree as control, the p-value relative to SLtree is $0.0903 > 0.05$, which is not statistically significant; however, ISLtree wins more often than it loses and has higher average metrics, and it is significantly superior to all other algorithms except SLtree.

Table 2: Post-hoc test results with SLtree as the control algorithm (Holm-adjusted p-values).

Alg.	Rank	Win	Tie	Loss	Adj. p
OURS-SLtree*	2.451	–	–	–	–
STree	2.500	29	1	11	**3.57e-02**
BoostTree	3.000	29	0	12	**3.57e-02**
BDTKS	4.085	36	0	5	**5.91e-05**
WODT	4.317	36	1	4	**1.30e-06**
TBSVM_Tree	5.220	38	0	3	**1.69e-09**

Table 3: Post-hoc test results with ISLtree as the control algorithm (Holm-adjusted p-values).

Alg.	Rank	Win	Tie	Loss	Adj. p
OURS-ISLtree*	2.159	–	–	–	–
OURS-SLtree	2.451	23	1	17	**9.03e-02**
STree	3.280	32	0	9	**1.30e-03**
BoostTree	3.756	31	0	10	**4.00e-04**
BDTKS	4.963	36	0	5	**2.50e-05**
TBSVM_Tree	6.122	37	0	4	**2.03e-09**
WODT	5.268	39	0	2	**1.68e-09**

*Control algorithm; p-values are Holm-adjusted; bold values indicate significant differences ($p < 0.05$).

We also investigated tree complexity and training time. Average tree size (node count) normalized by SLtree is shown in Table 4. All other methods have normalized values greater than one, confirming SLtree yields a more compact tree. As a balanced binary tree with fixed structure, it naturally has the most compact topology and uses fewer SVM classifiers than STree. Training time for SLtree is comparable to other methods, while ISLtree requires more time due to multiple PSO searches. Overall, considering accuracy, F1-score, and tree size, SLtree and ISLtree remain the best-performing algorithms.

Table 4: Normalized average tree size (node count) with SLtree as the baseline.

	SLtree	STree	WODT	BDTKS	BoostTree	TBSVM_Tree
Size	1.00	3.36	11.27	487.43	1.38	10.93

4.2 PERFORMANCE ON CHALLENGING DATASETS

Based on intuitive observation, ISLtree outperforms other methods on large-scale datasets and performs particularly well on imbalanced datasets; the F1-score also demonstrates its stability and superior performance on imbalanced datasets. In the following, we analyze the performance of ISLtree and SLtree on imbalanced, complex, and large-scale datasets. Table 5 summarizes the average accuracy, Macro F1, training time, and prediction time under three scenarios: (i) imbalanced datasets with $IR > 10$ and $Avg_IR > 10$ (12 datasets), (ii) complex datasets with $MFII \geq 0.1$ (14 datasets), and (iii) large-scale datasets with more than 10,000 instances (6 datasets).

Table 5: Performance comparison of all algorithms on three challenging scenarios. The scenarios are imbalanced datasets ($IR > 10$ and $Avg_IR > 10$), complex datasets ($MFII \geq 0.1$), and large-scale datasets (#Instances $> 10{,}000$). For each scenario, average accuracy (Acc.), Macro F1, training time (Train., in seconds), and prediction time (Pred., in milliseconds) are reported. The best value of each metric within each scenario is highlighted in bold.

	Imbalanced				*Complex*				*Large-scale*			
Algorithm	**Acc.**	**F1**	**Train**	**Pred**	**Acc.**	**F1**	**Train**	**Pred**	**Acc.**	**F1**	**Train**	**Pred**
OURS-ISLtree	**0.8694**	**0.6547**	901.85	0.530	**0.7288**	**0.6311**	871.13	0.657	**0.8351**	**0.7911**	1849.70	0.9809
OURS-SLtree	0.8037	0.4875	86.20	1.920	0.6560	0.4999	83.02	2.180	0.7188	0.5503	176.20	2.7174
STree	0.7973	0.4452	54.65	1.395	0.6429	0.4682	59.99	1.793	0.7136	0.5123	111.60	2.7100
BoostTree	0.7946	0.5364	**0.35**	**0.024**	0.6608	0.5091	**0.30**	**0.025**	0.6694	0.5305	**0.19**	**0.0134**
WODT	0.7519	0.4815	7.05	0.033	0.5550	0.4122	6.93	0.037	0.6343	0.5033	14.10	0.0335
BDTKS	0.7365	0.4648	1.04	0.077	0.6052	0.4788	1.28	0.064	0.6044	0.4614	1.70	0.0624
TBSVM_Tree	0.6224	0.2318	33.05	0.345	0.4375	0.1975	38.26	0.449	0.3729	0.0894	103.00	0.1784

Table 5 shows that on imbalanced datasets, ISLtree significantly outperforms other methods, and SLtree also surpasses others in both accuracy and F1-score, although not significantly. On complex datasets, ISLtree achieves superior classification performance, while SLtree is inferior to BoostTree. On large-scale datasets, SLtree does not exhibit a clear advantage; in contrast, ISLtree achieves the highest accuracy, surpassing the second-best method (SLtree) by 11.63 percentage points, and its F1-score also significantly exceeds those of the other methods. However, the training time of ISLtree is correspondingly longer, indicating that it trades time for performance improvement. In summary, ISLtree performs excellently across various challenging scenarios, demonstrating superior performance, good stability, and robustness.

5 CONCLUSION

A novel nonlinear SVM decision tree multi-class algorithm, termed SLtree, is introduced. SLtree is based on the iterative idea of K-means and simultaneously incorporates label constraints into the iterative process, thereby achieving more thorough and comprehensive node splitting. Experiments demonstrate that the proposed method performs well across a wide range of domains (41 datasets), with its four average metrics surpassing those of other excellent decision tree multi-class algorithms, which also demonstrates the advancement of this novel node splitting strategy. The study further introduces the particle swarm optimization algorithm to tune the hyperparameters of the SVM that controls node splitting, exploring the performance limits of decision tree classification. The proposed ISLtree algorithm not only improves decision tree performance but also performs excellently on imbalanced datasets, complex datasets, and large-scale datasets. Future work can further investigate from the perspectives of large-scale or imbalanced datasets.

One limitation of the ISLtree method lies in its training time. Clearly, since multiple parameter searches are performed within a single training process, although the classification limits of the nonlinear SVM are fully explored and performance gains are thereby obtained, this comes at the cost of training time. At the same time, we observe that this limitation is not pronounced on small datasets. In future research, we plan to attempt an innovative ensemble learning optimization path to reduce training time as much as possible without compromising performance, or even while improving it.

REFERENCES

Uci machine learning repository. https://archive.ics.uci.edu.

S. Benferhat, A. Boudjelida, K. Tabia, and H. Drias. An intrusion detection and alert correlation approach based on revising probabilistic classifiers using expert knowledge. *Applied Intelligence*, 38(4):520–540, 2013.

V. Blanco, A. Japón, and J. Puerto. Multiclass optimal classification trees with svm-splits. *Machine Learning*, 112(12):4649–4678, 2023. doi: 10.1007/s10994-023-06366-1.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall/CRC, Boca Raton, 1984.

P. Das, H. Behera, and B. Panigrahi. A hybridization of an improved particle swarm optimization and gravitational search algorithm for multi-robot path planning. *Swarm and Evolutionary Computation*, 28:14–28, 2016. doi: 10.1016/j.swevo.2015.10.011.

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

M. Dorigo. *Optimization, learning and natural algorithms*. Ph.d. thesis, Politecnico di Milano, Dipartimento di Elettronica, 1992.

M. A. Ganaie, M. Tanveer, and P. N. Suganthan. Oblique decision tree ensemble via twin bounded svm. *Expert Systems with Applications*, 143:113072, 2019. doi: 10.1016/j.eswa.2019.113072.

S. García and F. Herrera. An extension on 'statistical comparisons of classifiers over multiple data sets' for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694, 2008.

Greg Hamerly and Charles Elkan. Learning the k in k-means. In *Advances in Neural Information Processing Systems*, volume 16, pp. 281–288, Vancouver, BC, Canada, 2003.

M. Han, H. Guo, and W. Wang. A new data complexity measure for multi-class imbalanced classification tasks. *Pattern Recognition*, 157:110881, 2025. doi: 10.1016/j.patcog.2024.110881.

Fevzi Yasin Kababulut, Damla Gürkan Kuntalp, Okan Düzyel, Nermin Özcan, and Mehmet Kuntalp. A new shapley-based feature selection method in a clinical decision support system for the identification of lung diseases. *Diagnostics*, 13(23):15, 2023. doi: 10.3390/diagnostics13233558.

P. Kantavat, P. Songsiri, and B. Kijsirikul. Efficient decision trees for multi-class support vector machines using large centroid distance grouping. *Engineering Journal*, 26(5):13–23, 2022. doi: 10.4186/ej.2022.26.5.13.

D. Karaboga. An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Department of Computer Engineering, 2005.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942–1948, 1995. doi: 10.1109/ICNN.1995.488968.

M. Kohler, M. Vellasco, and R. Tanscheit. PSO+: a new particle swarm optimization algorithm for constrained problems. *Applied Soft Computing*, 85:105865, 2019. doi: 10.1016/j.asoc.2019.105865.

S. Kumar, G. Tejani, and N. Pholdee. Improved metaheuristics through migration-based search and an acceptance probability for truss optimization. *Asian Journal of Civil Engineering*, 21: 1217–1237, 2020. doi: 10.1007/s42107-020-00271-x.

S. Kumar, N. Panagant, and G. Tejani. A two-archive multi-objective multi-verse optimizer for truss design. *Knowledge-Based Systems*, 270:110529, 2023. doi: 10.1016/j.knosys.2023.110529.

P. Li, X. Wu, X. Hu, and H. Wang. An incremental decision tree for mining multilabel data. *Applied Artificial Intelligence*, 29(10):992–1014, 2015.

R. Montañana, José A. Gámez, and José M. Puerta. STree: a single multi-class oblique decision tree based on support vector machines. In *Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pp. 54–64. Springer, 2021. doi: 10.1007/978-3-030-85713-4_6.

R. Montañana, José A. Gámez, and José M. Puerta. ODTE—an ensemble of multi-class svm-based oblique decision trees. *Expert Systems with Applications*, 273:126833, 2025. doi: 10.1016/j.eswa.2025.126833.

A. Oumaima, H. Latifi, and L. Haltou. Classification of a quickbird satellite image by machine learning techniques: Mapping an urban environement by decision tree method. In *E3S Web of Conferences*, volume 364, pp. 7, 2023. doi: 10.1051/e3sconf/202336404001. Article No. 7.

M. Siva Kumar Reddy and R. Priyanka. Novel classification approach to minimize the false detection rate of epilepsy using decision tree classifier in comparison with artificial neural networks. In *AIP Conference Proceedings*, volume 2816, pp. 030007, 2024.

David M. Rodríguez, Manuel P. Cuéllar, and Diego P. Morales. Concept logic trees: enabling user interaction for transparent image classification and human-in-the-loop learning. *Applied Intelligence*, 54(5):3667–3679, 2024. doi: 10.1007/s10489-024-05321-4.

R. Sharma, J. S. Matharu, and K. S. Parmar. A survey on particle swarm optimization: Evolution, adaptations and practical implementations. *Applied Soft Computing*, 186:114016, 2026.

Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation*, pp. 69–73. IEEE, 1998.

Y. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pp. 1945–1950. IEEE, 1999.

A. C. Stasis, E. N. Loukis, S. A. Pavlopoulos, and D. Koutsouris. A decision tree-based method, using auscultation findings, for the differential diagnosis of aortic stenosis from mitral regurgitation. In *Proceedings of the IEEE International Conference on Computers in Cardiology*, 2003. doi: 10.1109/CIC.2003.1291270.

L. Sun, S. Fu, and F. Wang. Decision tree svm model with fisher feature selection for speech emotion recognition. *Eurasip Journal on Audio, Speech, and Music Processing*, 2019(1), 2019. doi: 10.1186/s13636-018-0145-5.

D. Tian. Particle swarm optimization with chaos-based initialization for numerical optimization. *Intelligent Automation and Soft Computing*, 24(2):331–342, 2018. doi: 10.1080/10798587.2017.1293881.

Dongping Tian, Qiu Xu, Xinhui Yao, Guangnan Zhang, Yafeng Li, and Chenghu Xu. Diversity-guided particle swarm optimization with multi-level learning strategy. *Swarm and Evolutionary Computation*, 86:101533, 2024. doi: 10.1016/j.swevo.2024.101533.

Z. Wan, G. Wang, and B. Sun. A hybrid intelligent algorithm by combining particle swarm optimization with chaos searching technique for solving nonlinear bilevel programming problems. *Swarm and Evolutionary Computation*, 8:26–32, 2013. doi: 10.1016/j.swevo.2012.08.001.

F. Wang, Q. Wang, and F. Nie. A linear multivariate binary decision tree classifier based on k-means splitting. *Pattern Recognition*, 107:107521, 2020. doi: 10.1016/j.patcog.2020.107521.

Quan Wang, Z. Zheng, Hao Lei, Fei Wang, Zitong Zhang, Xiaowu Zou, and Feiping Nie. An adaptive multi-splitting multivariate decision tree for multi-class classification applied on high-resolution and hyperspectral remote sensing images. *Remote Sensing*, 18(11):1790, 2026. doi: 10.3390/rs18111790.

Andrew M. Wilson. CORR synthesis: Can decision tree learning advance orthopaedic surgery research? *Clinical Orthopaedics & Related Research*, 481(12):6, 2023. doi: 10.1097/CORR.0000000000002820.

J. Xian, S. Rezvani, and D. Yang. A new decision tree based on intuitionistic fuzzy twin support vector machines. *IEEE Transactions on Intelligent Transportation Systems*, 2024. doi: 10.1109/TITS.2024.3445664.

B. Xue, M. Zhang, and W. Browne. Particle swarm optimisation for feature selection in classification: novel initialisation and updating mechanisms. *Applied Soft Computing*, 18:261–276, 2014. doi: 10.1016/j.asoc.2013.09.018.

B. B. Yang, S. Q. Shen, and W. Gao. Weighted oblique decision trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5621–5627, 2019. doi: 10.1609/aaai.v33i01.33015621.

X.-S. Yang. Firefly algorithms for multimodal optimization. In O. Watanabe and T. Zeugmann (eds.), *Stochastic Algorithms: Foundations and Applications*, volume 5792 of *Lecture Notes in Computer Science*, pp. 169–178. Springer, 2009.

O. T. Yildiz and E. Alpaydin. Linear discriminant trees. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(3):323–353, 2005.

C. Zhao, D. Wu, and J. Huang. Boosttree and boostforest for ensemble learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(7):8960–8976, 2023. doi: 10.1109/TPAMI.2022.3227370.

A Appendix

A.1 Algorithm

The general procedure of SLtree model construction is illustrated in Figure 1 and Algorithm 1. The constructed decision tree is a balanced binary tree. Algorithm 1 takes the training sample matrix X and label vector y as input and outputs a binary tree, where each split node stores a trained binary SVM classifier and each leaf node stores the assigned class label y^*. During training, each node invokes Algorithm 2 to obtain the binary labels y_{binary}.

Algorithm 1: SLtree Algorithm

Input: Training sample matrix $X \in \mathbb{R}^{n \times d}$ and corresponding label vector $y \in \mathbb{R}^n$.
Output: A binary tree, where each split node stores a trained binary SVM classifier, and each leaf node stores the assigned class label y^*.

Initialize Ω as an empty set;
Add the root node containing training data X to Ω;
while Ω *is not empty* **do**
    Select a node from Ω with sample matrix $X \in \mathbb{R}^{n \times d}$ and label vector $y \in \mathbb{R}^n$;
    Perform label-constrained binary partitioning (Algorithm 2) to obtain matrices G and F for deriving binary labels y_{binary};
    Train a binary SVM classifier on X and y_{binary} using fixed hyperparameters (e.g., RBF kernel, $C = 1.0$, $\gamma = 1.0$);
    Partition the node samples into two subsets X_1 and X_2 according to the binary labels y_{binary};
    for $i = 1, 2$ **do**
        if X_i *and* y_i *satisfy the non-splitting condition (e.g., insufficient samples or only one class)* **then**
            Set the child node containing X_i as a leaf node and assign the majority class label y^*;
        else
            Add the child node containing X_i to Ω;
    Remove the current node X from Ω;

We propose a novel multi-class classification method that constructs a balanced tree structure consisting of multiple binary support vector machines. The core of this construction is a label-constrained bipartitioning approach, implemented in Algorithm 2, which takes the sample data $\mathbf{X}$ and labels $\mathbf{y}$ of the current node as input and outputs the binary class labels $\mathbf{y}_{\text{binary}}$. This algorithm is the source of the y_{binary} used in Algorithm 1 to train the node's SVM classifier.

Algorithm 2: Label-Constrained Binary Partitioning

Input: Sample data $\mathbf{X}$ and labels $\mathbf{y}$ of the current node
Output: Binary class labels $\mathbf{y}_{\text{binary}}$

Obtain unique labels $L = \text{unique}(\mathbf{y})$, $l = |L|$;
if $l > 1$ **then**
  Compute balanced group size $h = \text{round}((l-1)/2)$;
  Generate all possible grouping matrices $\mathbf{G} \in \mathbb{R}^{l\times 2}$ where the first column has exactly h ones;
  Initialize centroids $\mathbf{c}_1, \mathbf{c}_2$ using a PCA-based method;
  for *each* $\mathbf{G}$ **do**
    Construct label indicator matrix $\mathbf{L}$;
    Compute group indicator $\mathbf{F} = \mathbf{LG}$;
    Reconstruct data: $\hat{\mathbf{X}} = \mathbf{CF}^\top$;
    Compute reconstruction error: $\|\mathbf{X} - \hat{\mathbf{X}}\|_F$;
  Select $\mathbf{G}^*$ with the minimum reconstruction error;
  Iteratively optimize $\mathbf{C}$ and $\mathbf{G}$ until convergence;
  Create binary labels $\mathbf{y}_{\text{binary}} = \text{argmax}(\mathbf{F}, \text{axis} = 1)$;

The ISLtree model extends SLtree by optimizing the SVM parameters at each node. Its overall procedure is illustrated in Algorithm 3 and Figure 3. Algorithm 3 shares the same input (training sample matrix $\mathbf{X}$ and label vector $\mathbf{y}$) and output (a binary tree with split nodes storing SVM models and leaf nodes storing class labels l^*) as Algorithm 1, but differs in that it invokes Algorithm 4 to train a unique RBF SVM classifier with optimized parameters (C^*, γ^*) for each node. As shown in Figure 3, each non-leaf node SVM possesses different (C, γ) parameters.

Algorithm 3: ISLtree Algorithm

Input: Training sample matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ and corresponding class label vector $\mathbf{y} \in \mathbb{R}^n$.
Output: A binary tree, where split nodes store optimized SVM models with parameters (C^*, γ^*), and leaf nodes store assigned class labels l^*.

Initialize Ω as an empty set;
Add the node containing training sample matrix $\mathbf{X}$ as the root to Ω;
while Ω *is not empty* **do**
  Select a node from Ω with sample data matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ and corresponding class labels $\mathbf{y} \in \mathbb{R}^n$;
  Perform label-constrained binary partitioning (Algorithm 2) to obtain binary labels $\mathbf{y}_{\text{binary}}$;
  Perform classification hyperplane training (Algorithm 4) to obtain the trained node classifier $\mathbf{S}$;
  Partition the node samples into two subsets X_1 and X_2 according to the binary labels y_{binary};
  for $i = 1, 2$ **do**
    if $\mathbf{X}_i$ *and* $\mathbf{y}_i$ *satisfy the non-splitting condition* **then**
      Set the child node containing $\mathbf{X}_i$ as a leaf node and assign its majority class label l^*;
    else
      Add the child node containing $\mathbf{X}_i$ to Ω;
  Remove the current node $\mathbf{X}$ from Ω;

Algorithm 4 is responsible for training the node classifier in Algorithm 3. It takes the sample data $\mathbf{X}$ and the binary labels $\mathbf{y}_{\text{binary}}$ of the current node as input and outputs the trained binary classifier $\mathbf{S}$ together with its optimal parameters (C^*, γ^*). This algorithm is the source of the node classifier

$\mathbf{S}$ used in Algorithm 3. The optimization process consists of three stages: initialization, iterative updating of particle positions and velocities, and convergence to the optimal parameters.

Algorithm 4: Classification Hyperplane with IPSO-Optimized Parameters

Input: Sample data $\mathbf{X}$ and binary labels $\mathbf{y}_{\text{binary}}$ of the current node
Output: Trained node binary classifier $\mathbf{S}$ and its optimal parameters (C^*, γ^*)

```
Apply MinMaxScaler normalization to X;
Optimize SVM parameters using the improved particle swarm optimization (IPSO);
Initialize particle swarm with parameters (C, γ);
For each iteration, perform the following;
    Evaluate fitness using validation set accuracy;
    Update particle positions and velocities;
    Apply mutation operation with probability p_m;
Select the best parameters (C*, γ*);
Train an SVM model with RBF kernel using the optimized parameters;
```

A.2 Comparison Methods

The following algorithms are employed for comparison with our method to demonstrate the superiority of the proposed algorithms.

SLtree: This is our proposed method. For parameters that can be adjusted, the default settings are uniformly used for testing: max depth = 15, min samples = 5.
ISLtree: This is our proposed method. Since the IPSO algorithm is applied, manual tuning of the SVM classifier parameters is not required. For adjustable parameters, such as the sampling threshold, sample size, and early stopping rounds set for acceleration, the default settings are uniformly used for testing: sample threshold = 2000, sample size = 1000, early stop rounds = 3, max depth = 15, min samples = 5.
STree: An oblique decision tree classification algorithm based on support vector machines proposed by Ricardo et al. (Montañana et al., 2021), which uses Shannon entropy as the evaluation metric when applying the one-vs-rest (OVR) strategy for splitting at nodes. The authors provided tuned parameters for each dataset they used. Since the datasets used in our experiments are not exactly the same as theirs, the default parameters provided by the authors are adopted: kernel = rbf, $C = 1.0$, max iterations = 10^5, max features = all.
BDTKS.v2: A hierarchical binary decision classification tree based on K-means splitting, BDTKS, proposed by Wang et al. (Wang et al., 2020). According to different cluster center initialization methods, the authors divided the algorithm into two versions. In our experiments, the v2 version, which uses a deterministic method to initialize cluster centers, is selected for comparison. The authors performed parameter tuning in their experimental section. For convenience, a unified set of parameters is used for all datasets, i.e., the default original parameters in the provided code.
BoostTree: The base learner of the BoostForest ensemble learning method proposed by Zhao et al. (Zhao et al., 2023). BoostTree constructs tree models via gradient boosting and randomly samples cut points during node splitting to increase randomness. It is a decision tree model suitable for both regression and classification. In our experiments, the part of the provided code that uses BoostTree for multi-class classification is employed, and the original default parameters in the code are used.
TBSVM_Tree: The base learner used in the ensemble learning paper by Ganaie et al. (Ganaie et al., 2019), an oblique decision tree based on TBSVM (Twin Bounded Support Vector Machine). The code provided by the authors is in Matlab, so this method is specifically trained and tested on the Matlab platform, while the datasets are exactly the same as those used by the other methods. The code for the base learner part is used. The paper discusses the impact of parameter selection on experimental results and concludes that parameters need to be tuned to obtain the best accuracy. However, to facilitate experimental results, the original default parameters of the base learner are adopted.
WODT: Yang et al. (Yang et al., 2019) used weighted information entropy as the objective function and optimized this continuously differentiable objective function to find the best partition within each node, thereby constructing a novel weighted oblique decision tree method, WODT.

A.3 TABLE

The detailed information of each dataset is presented in Table 6, including the identifier (ID), name (Dataset), number of instances (#Ins.), number of attributes (#Att.), number of class labels (#Class), maximum/standard imbalance ratio (IR), and the average imbalance ratio (Avg_IR) extended to multi-class data, where a larger value of either imbalance ratio indicates a more uneven class distribution. In addition, the classification complexity is measured by the MFII (Multi-Factor Imbalance Index) proposed by Han et al. (Han et al., 2025), which comprehensively evaluates the impact of data imbalance and other intrinsic data characteristics, such as class overlap and sub-cluster structures, on classification difficulty. The evaluation result is a continuous value between 0 and 1, where a higher MFII value indicates higher data complexity and greater classification difficulty. The MFII values for all datasets were calculated using the code provided by the authors. It can be observed that the datasets employed in this experiment comprise a total of 41 multi-class datasets ranging from 3 to 18 classes, with dataset sizes varying from 101 to 32,485 and varying numbers of feature variables. They include both highly balanced and imbalanced datasets, as well as complex datasets with high classification difficulty and datasets with relatively low classification difficulty.

Table 6: Dataset Statistics

Id	Dataset	#Ins.	#Att.	#Class	IR	Avg_IR	MFII
1	student_prediction	145	14	5	51.50	20.21	0.202433
2	Mobile_Phone_Pricing	2000	20	4	1.00	1.00	0.142721
3	music	17947	14	11	12.35	5.39	0.462269
4	Student_performance	2392	12	5	11.32	5.46	0.500000
5	dermatology1	366	14	4	39.00	22.04	0.355753
6	Wine_Quality	32485	12	7	588.21	120.94	0.500000
7	led-display	1000	7	10	1.32	1.15	0.325310
8	statlog-image	2310	18	7	1.00	1.00	0.031392
9	libras	360	90	15	1.00	1.00	0.014853
10	cardiotocography-10clases	2126	21	10	10.92	5.38	0.262083
11	low-res-spect	531	100	9	138.00	52.53	0.029629
12	dermatology2	366	34	6	5.60	2.69	0.009667
13	synthetic-control	600	60	6	1.00	1.00	0.009256
14	zoo	101	16	7	10.25	5.48	0.001450
15	kr-vs-k	28056	6	18	168.63	22.13	0.374219
16	yeast	1484	8	10	92.60	18.87	0.325072
17	marketing	6876	13	9	2.49	1.90	0.447425
18	optdigits	5620	64	10	1.03	1.02	0.001060
19	page-blocks	5472	10	5	175.46	72.40	0.044540
20	satimage	6435	36	6	2.45	1.79	0.067287
21	texture	5500	40	11	1.00	1.00	0.002097
22	vowel	990	13	11	1.00	1.00	0.321855
23	shuttle	14500	9	7	4558.60	1546.87	0.000822
24	nursery	12960	8	5	2160.00	543.81	0.429775
25	Ecoli	336	7	8	71.50	26.78	0.063225
26	Wine	178	13	3	1.48	1.34	0.015347
27	Glass	214	9	6	8.44	4.49	0.107570
28	Page-block2	5473	10	5	175.46	72.24	0.047115
29	Statlog-segment	2310	19	7	1.00	1.00	0.070184
30	Iris	150	4	3	1.00	1.00	0.032844
31	Chili	483	32	5	1.20	1.20	0.034997
32	Corn	4188	32	4	2.28	1.51	0.026120
33	Cucumber	1289	32	8	1.06	1.06	0.001941
34	Lemon	1354	32	9	2.10	1.57	0.000211
35	Lettuce	2337	32	8	187.17	68.30	0.017560
36	Mango	4000	32	8	1.00	1.00	0.000000
37	PLD	4072	32	3	1.60	1.37	0.003055
38	Pumpkin	2000	32	5	1.00	1.00	0.021162
39	Rice	120	32	3	1.00	1.00	0.004306
40	Tea	885	32	8	1.93	1.39	0.004120
41	Tomato	25848	32	11	3.10	1.52	0.005827

The complete experimental results are provided in the appendix. Table 7 and Table 8 present the cross-validation results of seven decision tree algorithms in terms of accuracy and F1-score on all datasets, respectively. From these tables, we can directly observe that SLtree and ISLtree are the best-performing algorithms in terms of both accuracy and F1-score. The ISLtree algorithm significantly improves the performance of the SLtree decision tree.

Table 7: Accuracy of each method on each dataset (mean ± std). Bold indicates the best accuracy per dataset.

Dataset	OURS-SLtree	OURS-ISLtree	STree	BDTKS	BoostTree	TBSVM_Tree	WODT
1	0.7172 ± 0.0517	**0.7220 ± 0.0552**	0.7103 ± 0.0388	0.6483 ± 0.0747	0.6800 ± 0.0737	0.7007 ± 0.0609	0.6317 ± 0.0783
2	0.7861 ± 0.0180	0.7549 ± 0.1096	0.6872 ± 0.0231	**0.8633 ± 0.0212**	0.8159 ± 0.0205	0.5000 ± 0.0000	0.3552 ± 0.0764
3	**0.4640 ± 0.0147**	**0.4640 ± 0.0250**	0.4044 ± 0.0205	0.3322 ± 0.0053	0.4622 ± 0.0062	0.2982 ± 0.0371	0.3884 ± 0.0215
4	0.6521 ± 0.0138	**0.7462 ± 0.0148**	0.6467 ± 0.0146	0.6493 ± 0.0158	0.6780 ± 0.0188	0.6826 ± 0.0196	0.6084 ± 0.0417
5	0.8773 ± 0.0181	0.8656 ± 0.0233	**0.8850 ± 0.0174**	0.8459 ± 0.0259	0.8609 ± 0.0322	0.8541 ± 0.0358	0.8544 ± 0.0278
6	0.5431 ± 0.0089	**0.8848 ± 0.0710**	0.5582 ± 0.0115	0.4863 ± 0.0064	0.5622 ± 0.0070	0.4346 ± 0.0001	0.5061 ± 0.0201
7	**0.7192 ± 0.0240**	0.7084 ± 0.0243	0.7055 ± 0.0232	0.7048 ± 0.0231	0.7159 ± 0.0229	0.4280 ± 0.1296	0.6480 ± 0.0519
8	0.9171 ± 0.0112	**0.9658 ± 0.0106**	0.9396 ± 0.0107	0.9043 ± 0.0157	0.8251 ± 0.0100	0.5211 ± 0.0821	0.9171 ± 0.0175
9	0.8233 ± 0.0558	**0.8508 ± 0.0438**	0.7578 ± 0.0541	0.7539 ± 0.0523	0.7694 ± 0.0517	0.5186 ± 0.1481	0.7103 ± 0.0579
10	0.7931 ± 0.0152	**0.8232 ± 0.0182**	0.7969 ± 0.0164	0.6550 ± 0.0222	0.7837 ± 0.0207	0.5016 ± 0.1266	0.6948 ± 0.0356
11	0.8669 ± 0.0209	0.8811 ± 0.0293	0.8625 ± 0.0231	0.8190 ± 0.0314	**0.8813 ± 0.0251**	0.7932 ± 0.0602	0.8426 ± 0.0323
12	0.9544 ± 0.0217	0.9479 ± 0.0254	0.9434 ± 0.0249	0.9456 ± 0.0235	0.9505 ± 0.0200	**0.9721 ± 0.0183**	0.8973 ± 0.0285
13	0.9865 ± 0.0106	0.9737 ± 0.0215	**0.9915 ± 0.0096**	0.9735 ± 0.0139	0.9622 ± 0.0181	0.9507 ± 0.0401	0.8840 ± 0.0291
14	0.9348 ± 0.0556	0.8982 ± 0.0608	0.9327 ± 0.0433	0.9485 ± 0.0413	0.9465 ± 0.0373	**0.9624 ± 0.0394**	0.9266 ± 0.0533
15	0.4452 ± 0.0160	**0.7450 ± 0.0407**	0.4656 ± 0.0153	0.3436 ± 0.0067	0.3450 ± 0.0074	0.1623 ± 0.0001	0.3442 ± 0.0158
16	0.5787 ± 0.0256	**0.5902 ± 0.0265**	0.5220 ± 0.0323	0.5107 ± 0.0293	0.5726 ± 0.0268	0.3686 ± 0.0317	0.5201 ± 0.0414
17	**0.3393 ± 0.0088**	0.3386 ± 0.0079	0.3259 ± 0.0108	0.2999 ± 0.0107	0.3279 ± 0.0096	0.3067 ± 0.0096	0.3084 ± 0.0117
18	0.9843 ± 0.0043	**0.9898 ± 0.0027**	0.9497 ± 0.0083	0.9150 ± 0.0105	0.6711 ± 0.0225	0.9411 ± 0.0546	0.8378 ± 0.0193
19	0.9418 ± 0.0053	**0.9678 ± 0.0045**	0.9391 ± 0.0055	0.9362 ± 0.0063	0.9674 ± 0.0044	0.9166 ± 0.0163	0.9602 ± 0.0048
20	**0.8965 ± 0.0076**	0.8284 ± 0.0107	0.8936 ± 0.0072	0.8730 ± 0.0076	0.8203 ± 0.0114	0.3618 ± 0.0099	0.7579 ± 0.0138
21	0.9837 ± 0.0040	**0.9957 ± 0.0020**	0.9813 ± 0.0053	0.9121 ± 0.0089	0.6477 ± 0.0093	0.7939 ± 0.0991	0.9125 ± 0.0184
22	0.7075 ± 0.0651	**0.9302 ± 0.0259**	0.7760 ± 0.0353	0.8883 ± 0.0276	0.8466 ± 0.0311	0.1988 ± 0.0204	0.6028 ± 0.0603
23	0.9873 ± 0.0028	0.9977 ± 0.0009	0.9866 ± 0.0031	0.9872 ± 0.0027	**0.9986 ± 0.0006**	0.7916 ± 0.0002	0.9819 ± 0.0209
24	0.9302 ± 0.0047	**0.9757 ± 0.0044**	0.9271 ± 0.0050	0.5660 ± 0.0836	0.8805 ± 0.0058	0.3333 ± 0.0000	0.6625 ± 0.0672
25	0.8541 ± 0.0364	**0.8746 ± 0.0423**	0.8107 ± 0.0304	0.7997 ± 0.0434	0.8598 ± 0.0394	0.7118 ± 0.1541	0.8014 ± 0.0496
26	0.9877 ± 0.0170	0.9652 ± 0.0351	**0.9894 ± 0.0159**	0.9557 ± 0.0303	0.9636 ± 0.0320	0.9685 ± 0.0328	0.8966 ± 0.0599
27	0.6313 ± 0.0591	0.6547 ± 0.0751	0.5897 ± 0.0857	0.6795 ± 0.0482	**0.7200 ± 0.0641**	0.3556 ± 0.0099	0.6455 ± 0.0767
28	0.9433 ± 0.0055	**0.9681 ± 0.0051**	0.9415 ± 0.0064	0.9367 ± 0.0067	0.9676 ± 0.0038	0.9212 ± 0.0133	0.9601 ± 0.0055
29	0.9200 ± 0.0145	**0.9572 ± 0.0605**	0.9396 ± 0.0107	0.8881 ± 0.0142	0.8252 ± 0.0100	0.5447 ± 0.0632	0.8816 ± 0.0345
30	**0.9667 ± 0.0294**	0.9533 ± 0.0375	**0.9667 ± 0.0294**	0.9427 ± 0.0330	0.9493 ± 0.0400	0.9640 ± 0.0355	0.9380 ± 0.0415
31	**0.6992 ± 0.0499**	0.6860 ± 0.0483	0.6934 ± 0.0473	0.6688 ± 0.0502	0.6973 ± 0.0399	0.3870 ± 0.0767	0.6800 ± 0.0472
32	**0.9319 ± 0.0168**	0.9262 ± 0.0515	0.9318 ± 0.0170	0.9275 ± 0.0188	0.9299 ± 0.0188	0.6210 ± 0.1096	0.9274 ± 0.0189
33	**0.8849 ± 0.0223**	0.8781 ± 0.0250	0.8743 ± 0.0232	0.8632 ± 0.0307	0.8039 ± 0.0714	0.5572 ± 0.2000	0.8661 ± 0.0276
34	**0.9756 ± 0.0127**	0.9677 ± 0.0126	0.9617 ± 0.0169	0.9592 ± 0.0175	0.7801 ± 0.0556	0.5596 ± 0.2500	0.9536 ± 0.0180
35	0.9590 ± 0.0051	**0.9597 ± 0.0052**	0.9592 ± 0.0048	0.9582 ± 0.0065	0.9589 ± 0.0064	0.4805 ± 0.0008	0.9577 ± 0.0065
36	**0.9965 ± 0.0030**	0.9963 ± 0.0028	0.9931 ± 0.0034	0.9913 ± 0.0050	0.9291 ± 0.0784	0.5022 ± 0.2753	0.9916 ± 0.0044
37	0.9851 ± 0.0093	**0.9859 ± 0.0082**	0.9850 ± 0.0100	0.9838 ± 0.0100	0.9844 ± 0.0094	0.8862 ± 0.1530	0.9839 ± 0.0100
38	**0.8303 ± 0.0190**	0.8222 ± 0.0188	0.8288 ± 0.0173	0.8130 ± 0.0219	0.8123 ± 0.0195	0.4302 ± 0.0918	0.8152 ± 0.0205
39	0.8733 ± 0.0714	**0.8875 ± 0.0701**	0.8800 ± 0.0697	0.8742 ± 0.0840	0.8758 ± 0.0676	0.7850 ± 0.1624	0.8742 ± 0.0778
40	**0.8832 ± 0.0282**	0.8725 ± 0.0405	0.8720 ± 0.0300	0.8603 ± 0.0332	0.8826 ± 0.0251	0.6432 ± 0.2041	0.8584 ± 0.0318
41	0.9427 ± 0.0043	**0.9432 ± 0.0352**	0.9399 ± 0.0047	0.9112 ± 0.0093	0.7680 ± 0.0868	0.2171 ± 0.0293	0.9224 ± 0.0063
Average	0.8316 ± 0.1696	**0.8562 ± 0.1498**	0.8231 ± 0.1760	0.7994 ± 0.1898	0.7971 ± 0.1950	0.6056 ± 0.2460	0.7733 ± 0.2080

Table 8: Macro F1 of each method on each dataset (mean ± std). Bold indicates the best Macro F1 per dataset (all methods included).

Dataset	OURS-SLtree	OURS-ISLtree	STree	BDTKS	BoostTree	TBSVM_Tree	WODT
1	0.3203 ± 0.0813	0.3137 ± 0.0971	0.2437 ± 0.0604	0.2833 ± 0.1156	**0.3495 ± 0.1236**	0.2584 ± 0.0763	0.2671 ± 0.0806
2	0.7874 ± 0.0175	0.7542 ± 0.1114	0.6940 ± 0.0220	**0.8633 ± 0.0212**	0.8166 ± 0.0202	0.3334 ± 0.0001	0.3200 ± 0.0815
3	**0.4523 ± 0.0214**	0.4818 ± 0.0385	0.3454 ± 0.0356	0.2630 ± 0.0110	0.4386 ± 0.0157	0.1008 ± 0.0204	0.3432 ± 0.0409
4	0.4281 ± 0.0172	**0.5984 ± 0.0296**	0.4094 ± 0.0325	0.4733 ± 0.0265	0.4942 ± 0.0355	0.4056 ± 0.0215	0.4094 ± 0.0596
5	0.3751 ± 0.0606	0.3814 ± 0.0706	0.3754 ± 0.0419	**0.4200 ± 0.0396**	0.3990 ± 0.0774	0.3659 ± 0.0432	0.3674 ± 0.0731
6	0.2062 ± 0.0136	**0.8572 ± 0.1309**	0.2536 ± 0.0405	0.1952 ± 0.0135	0.2368 ± 0.0094	0.0866 ± 0.0000	0.2331 ± 0.0350
7	**0.7195 ± 0.0240**	0.7080 ± 0.0242	0.7041 ± 0.0230	0.7053 ± 0.0230	0.7156 ± 0.0233	0.3553 ± 0.1490	0.6384 ± 0.0617
8	0.9149 ± 0.0122	**0.9659 ± 0.0105**	0.9391 ± 0.0108	0.9034 ± 0.0161	0.7826 ± 0.0196	0.4120 ± 0.0851	0.9168 ± 0.0178
9	**0.8182 ± 0.0588**	0.8489 ± 0.0449	0.7481 ± 0.0596	0.7475 ± 0.0539	0.7635 ± 0.0547	0.4817 ± 0.1502	0.6993 ± 0.0620
10	0.6614 ± 0.0246	**0.7816 ± 0.0264**	0.6518 ± 0.0281	0.5929 ± 0.0290	0.7025 ± 0.0400	0.3028 ± 0.1288	0.5964 ± 0.0412
11	0.4516 ± 0.0628	0.6065 ± 0.0484	0.5081 ± 0.0686	0.5608 ± 0.0977	**0.5604 ± 0.0901**	0.4203 ± 0.0619	0.5295 ± 0.0993
12	0.9433 ± 0.0277	0.9376 ± 0.0384	0.9344 ± 0.0309	0.9393 ± 0.0258	0.9422 ± 0.0257	**0.9690 ± 0.0202**	0.8784 ± 0.0342
13	0.9864 ± 0.0107	0.9734 ± 0.0221	**0.9915 ± 0.0095**	0.9734 ± 0.0139	0.9622 ± 0.0180	0.9482 ± 0.0499	0.8834 ± 0.0294
14	0.8531 ± 0.1396	0.7568 ± 0.1331	0.8176 ± 0.1200	0.8512 ± 0.1113	0.8346 ± 0.1104	**0.8986 ± 0.1049**	0.8198 ± 0.1318
15	0.3234 ± 0.0186	**0.7034 ± 0.0529**	0.3638 ± 0.0290	0.2546 ± 0.0149	0.2319 ± 0.0091	0.0155 ± 0.0000	0.2779 ± 0.0230
16	**0.5410 ± 0.0518**	0.5303 ± 0.0549	0.4101 ± 0.0678	0.4790 ± 0.0576	0.4173 ± 0.0357	0.1286 ± 0.0824	0.4377 ± 0.0716
17	0.2459 ± 0.0118	0.2485 ± 0.0128	**0.2466 ± 0.0128**	0.2399 ± 0.0114	0.2227 ± 0.0142	0.1248 ± 0.0232	0.2234 ± 0.0125
18	**0.9843 ± 0.0042**	0.9898 ± 0.0027	0.9497 ± 0.0082	0.9152 ± 0.0103	0.6467 ± 0.0265	0.9362 ± 0.0658	0.8383 ± 0.0192
19	0.6007 ± 0.0621	**0.7560 ± 0.0571**	0.5586 ± 0.0653	0.6071 ± 0.0649	0.7828 ± 0.0505	0.3472 ± 0.0830	0.6966 ± 0.0651
20	**0.8745 ± 0.0093**	0.7832 ± 0.0167	0.8673 ± 0.0099	0.8445 ± 0.0092	0.7767 ± 0.0164	0.2902 ± 0.0174	0.6868 ± 0.0160
21	0.9837 ± 0.0040	**0.9957 ± 0.0020**	0.9813 ± 0.0053	0.9128 ± 0.0087	0.5915 ± 0.0102	0.7509 ± 0.1186	0.9129 ± 0.0185
22	0.6828 ± 0.0716	**0.9294 ± 0.0266**	0.7690 ± 0.0392	0.8880 ± 0.0281	0.8452 ± 0.0314	0.0829 ± 0.0233	0.5962 ± 0.0635
23	0.6084 ± 0.0766	0.7878 ± 0.0835	0.5107 ± 0.0568	0.7183 ± 0.1023	**0.9429 ± 0.0651**	0.1431 ± 0.0085	0.7406 ± 0.1144
24	**0.7679 ± 0.0772**	0.9740 ± 0.0063	0.6606 ± 0.0914	0.4243 ± 0.0901	0.6149 ± 0.0663	0.1150 ± 0.0124	0.5021 ± 0.0757
25	0.7190 ± 0.0766	**0.8034 ± 0.0959**	0.5556 ± 0.0871	0.6253 ± 0.0850	0.7287 ± 0.0694	0.4850 ± 0.2150	0.6248 ± 0.1050
26	0.9878 ± 0.0168	0.9643 ± 0.0362	**0.9894 ± 0.0158**	0.9579 ± 0.0288	0.9642 ± 0.0321	0.9693 ± 0.0325	0.8980 ± 0.0596
27	0.4876 ± 0.0973	0.5735 ± 0.1205	0.4274 ± 0.0998	0.6217 ± 0.0843	**0.6430 ± 0.0932**	0.0890 ± 0.0122	0.5587 ± 0.1028
28	0.6192 ± 0.0638	**0.7631 ± 0.0630**	0.5874 ± 0.0787	0.6222 ± 0.0552	0.7897 ± 0.0412	0.3349 ± 0.0544	0.7178 ± 0.0626
29	0.9181 ± 0.0155	0.9554 ± 0.0736	**0.9391 ± 0.0108**	0.8876 ± 0.0142	0.7827 ± 0.0194	0.4358 ± 0.0606	0.8798 ± 0.0363
30	**0.9665 ± 0.0295**	0.9530 ± 0.0379	**0.9665 ± 0.0295**	0.9425 ± 0.0331	0.9491 ± 0.0402	0.9636 ± 0.0361	0.9378 ± 0.0418
31	**0.6955 ± 0.0492**	0.6816 ± 0.0479	0.6891 ± 0.0483	0.6658 ± 0.0505	0.6925 ± 0.0396	0.2689 ± 0.0979	0.6749 ± 0.0467
32	**0.9148 ± 0.0250**	0.9058 ± 0.0768	0.9146 ± 0.0260	0.9100 ± 0.0251	0.9123 ± 0.0258	0.4447 ± 0.1297	0.9094 ± 0.0259
33	**0.8822 ± 0.0236**	0.8764 ± 0.0257	0.8713 ± 0.0243	0.8605 ± 0.0317	0.7864 ± 0.0920	0.4776 ± 0.2386	0.8635 ± 0.0285
34	**0.9711 ± 0.0148**	0.9632 ± 0.0144	0.9559 ± 0.0187	0.9532 ± 0.0192	0.6731 ± 0.0850	0.4432 ± 0.2970	0.9470 ± 0.0201
35	0.3168 ± 0.0373	0.3794 ± 0.0596	0.3145 ± 0.0372	**0.3872 ± 0.0598**	0.3830 ± 0.0686	0.0811 ± 0.0001	0.3834 ± 0.0682
36	**0.9965 ± 0.0030**	0.9963 ± 0.0028	0.9931 ± 0.0034	0.9913 ± 0.0050	0.9121 ± 0.1011	0.4294 ± 0.3086	0.9916 ± 0.0044
37	0.9848 ± 0.0093	0.9859 ± 0.0082	0.9848 ± 0.0100	0.9838 ± 0.0100	**0.9845 ± 0.0094**	0.8618 ± 0.1924	0.9839 ± 0.0101
38	**0.8301 ± 0.0191**	0.8220 ± 0.0189	0.8286 ± 0.0172	0.8128 ± 0.0220	0.8124 ± 0.0195	0.3120 ± 0.1115	0.8148 ± 0.0206
39	0.8722 ± 0.0722	**0.8867 ± 0.0699**	0.8792 ± 0.0697	0.8726 ± 0.0853	0.8739 ± 0.0696	0.7566 ± 0.2025	0.8728 ± 0.0789
40	**0.8807 ± 0.0283**	0.8696 ± 0.0422	0.8694 ± 0.0304	0.8578 ± 0.0329	0.8805 ± 0.0253	0.5746 ± 0.2377	0.8554 ± 0.0325
41	**0.9435 ± 0.0045**	0.9422 ± 0.0471	0.9398 ± 0.0050	0.9128 ± 0.0086	0.7177 ± 0.1146	0.0756 ± 0.0213	0.9231 ± 0.0066
Average	0.7199 ± 0.2446	**0.7793 ± 0.2042**	0.6985 ± 0.2525	0.7054 ± 0.2442	0.6965 ± 0.2165	0.4214 ± 0.2925	0.6744 ± 0.2385