svmodt: An R Package for Linear SVM-Based Oblique Decision Trees

by Aneesh Agarwal, Jack Jewson, and Erik Sverdrup

Abstract Decision trees are widely used for classification tasks due to their interpretability, but traditional axis-aligned splits often require deep trees to approximate complex decision boundaries. We introduce svmodt, an R package implementing SVM-based oblique decision trees (SVMODT) that uses linear Support Vector Machines at each node to create multivariate splits. The package supports dynamic feature selection strategies, node-specific class weighting for handling imbalanced data, and feature diversity mechanisms through penalization. We demonstrate that SVMODT outperforms axis-parallel decision trees while also maintaining shallower and more interpretable tree structures. The package includes comprehensive visualization tools, detailed documentation, and is freely available on GitHub.

1 Introduction

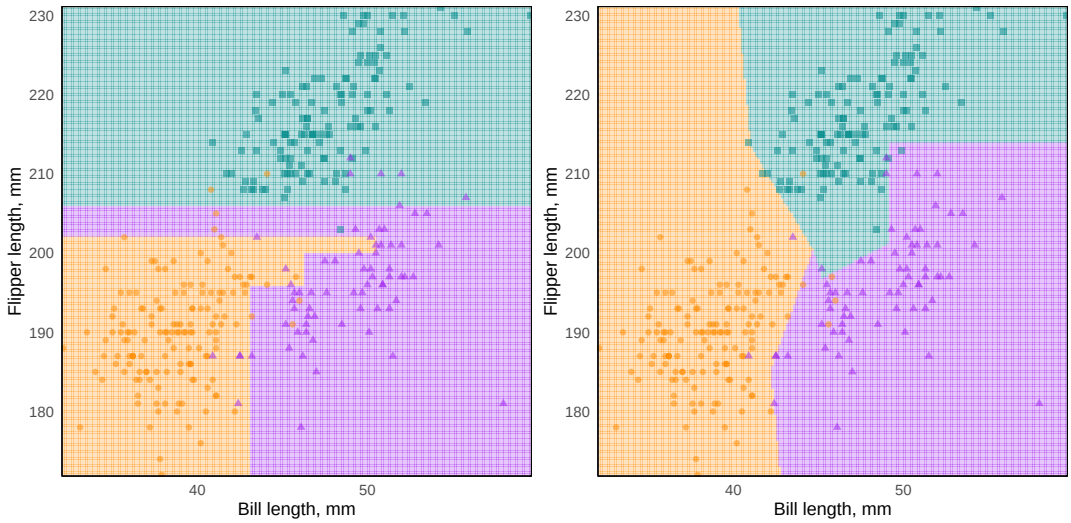
Decision trees remain widely used in machine learning because they are interpretable, straightforward to apply, and can accommodate both categorical and numerical predictors with minimal preprocessing. Classical algorithms such as CART (Breiman et al., 1984) and C4.5 (Quinlan, 2014) employ axis-aligned (univariate) splits that partition the feature space along coordinate axes, thereby preserving transparency in the resulting decision rules. Consider a supervised learning task where we wish to predict a categorical or continuous response $y \in \mathcal{Y}$ from d-predictor variables $x \in \mathcal{X} \subseteq \mathbb{R}^d$. In this setting an axis parallel split constitutes selecting one features $j \in \{1, \ldots, d\}$ and a splitting value $\theta \in \mathbb{R}$ and splits according to the test $x_j \leq \theta$. Throughout this work, we focus exclusively on the supervised learning problem of predicting y from d-dimensional predictors x, where the goal is to learn a decision function that maps $x \in \mathbb{R}^d \longrightarrow y$.

In R, traditional decision trees are readily implemented via packages such as **rpart**, **tree**, and **party**. However, the limited split structure of axis-parallel decision trees can promote overfitting, producing duplicated subtrees and repeated evaluations of the same predictor, which reduces efficiency and weakens generalization (Pagallo and Haussler, 1990). Oblique decision trees mitigate these limitations by permitting splits on linear combinations of features; for example, a single oblique test of the form $\sum_{j=1}^{d} w_j x_j \leq \theta$, where $w = (w_1, \ldots, w_d) \in \mathbb{R}^d$, can often replace multiple axis-aligned tests, producing more compact and geometrically simpler partitions of the feature space. Oblique trees are available in R through packages such as **ODRF**, **aorsf**, and **oblique.tree**. Empirical studies across diverse data sets show that oblique decision trees often achieve higher accuracy and yield more compact models than axis-parallel trees, though with a modest reduction in interpretability. (Cañete et al., 2021).

Figure 1 contrasts an axis-parallel decision tree with an axis-oblique decision tree. The axis-parallel tree displays (left panel) clear overfitting and a high rate of training misclassification, reflecting its inability to capture interactions among predictors that determine class membership. In contrast, the oblique tree (right panel) yields geometrically simpler decision boundaries and substantially fewer misclassifications on the training set, indicating a better fit to the underlying multivariate relationships. Together, these panels illustrate how oblique splits can more effectively represent linear combinations of features and thereby reduce spurious complexity and error in tree-based classifiers.

Support Vector Machines (SVMs), introduced by Cortes and Vapnik (1995), provide a principled way to obtain oblique splits by finding margin-maximizing hyperplanes. This margin-based approach enhances the generalization ability of the model, making SVMs robust and effective for many real-world problems (Cristianini, 2000). The idea of fitting SVMs at tree nodes was formalized by Bennett and Blue (1998) and extended to multiclass settings and other refinements in subsequent work (Takahashi and Abe, 2002; Bala and Agrawal, 2011; Ganaie et al., 2022). Python's STree implements SVM-node oblique trees using scikit-learn's SVC (Montañana et al., 2021; Pedregosa et al., 2011), while Java tools such as Weka offer components that can be adapted for hierarchical SVM trees (Menkovski et al., 2008); however, fully featured, native R implementations remain limited.

To address this gap, we advance two primary contributions. First, we introduce a fully native R implementation of an STree-style oblique decision tree that operates independently of Python. This implementation provides a production-ready API and leverages vectorized computation to ensure efficiency and seamless integration within the R ecosystem. Second, we develop **SVM-ODT**, an enhanced SVM-based oblique decision tree that extends the original STree framework through several practical innovations. These include feature sub-setting, embedded feature selection, feature penalization, and node-specific class weighting. Collectively, these enhancements are designed to



(a) axis-parallel split **(b)** axis-oblique split

Figure 1: Comparison of Axis-parallel and Axis-oblique splits on Palmerpenguins dataset.

improve predictive performance, interpretability, and robustness across both binary and multiclass classification settings.

Together, these contributions aim to make the representational strengths of SVM-driven oblique splits accessible to R programmers while simultaneously addressing key computational and methodological limitations present in existing approaches.

2 Background

Decision Trees

Decision Trees (DTs) are interpretable classification models that represent their decision-making process through a hierarchical, tree-like structure. This structure comprises of internal nodes containing splitting criteria that divide up the predictor space and terminal (leaf) nodes corresponding to predicted outcomes (class labels and probabilities). The nodes are connected by directed edges, each representing a possible True or False outcome of a splitting criterion for observations whose feature reach this node. Formally, a DT can be expressed as a rooted, directed tree $T = (G(V, E), v_1)$, where V denotes the set of nodes, E represents the set of edges linking these nodes, and v_1 is the root node.

Decision tree algorithms can be categorized based on whether the same type of test is applied at all internal nodes. **Homogeneous trees** employ a single algorithm throughout (e.g., univariate or multivariate splits), whereas **hybrid trees** allow different algorithms such as linear discriminant functions, k-nearest neighbors, or univariate splits that can be used in different sub-trees (Brodley and Utgoff, 1995). Hybrid trees exploit the principle of *selective superiority*, allowing subsets of the data to be modeled by the most appropriate classifier, thereby improving flexibility and accuracy. However, hybrid tree models typically incur substantial computational overhead due to their structural complexity, and therefore, in this paper we restrict our focus to homogeneous trees.

Axis-Parallel Decision Trees

Axis-Parallel Decision trees represent hyper-planes dividing the instance space into several disjoint axis-parallel regions. Axis-parallel decision trees, such as CART and C4.5, represent two of the most widely used algorithms for classification tasks. The **CART (Classification and Regression Trees)** algorithm employs a binary recursive partitioning procedure at capable of handling both continuous and categorical variables as predictors or targets. At each node, the algorithm systematically evaluates every available variable $j \in \{1, \ldots, d\}$ and all of its potential split points θ i.e. between any two observations, to determine the optimal partition. In R, traditional decision trees are readily implemented via packages such as **rpart**, **tree**, and **party**.

In contrast, **C4.5**, an extension of the earlier **ID3** algorithm (Quinlan, 1986), utilizing information theory measures such as **information gain** and **gain ratio** to select the most informative attribute

for each split (Quinlan, 2014). C4.5 also includes mechanisms to handle missing attribute values by weighting instances according to the proportion of known data and employs an **error-based pruning** to eliminate subtrees that fail to improve predictive performance beyond a threshold (α), thereby mitigating overfitting. Although these techniques are effective across diverse data sets, studies have shown that the choice of pruning strategy and stopping criteria can significantly affect model performance across different domains (Mingers, 1989; Schaffer, 1992).

Axis-parallel decision trees offer strong interpretability but face inherent representational limits. Their constrained split structure often induces overfitting, leading to duplicated subtrees and repeated tests of the same predictor, ultimately reducing efficiency and weakening generalization (Pagallo and Haussler, 1990).

Axis-Oblique Decision Trees

Axis-oblique decision trees extends axis-parallel decision trees by allowing each internal node to perform splits based on linear or nonlinear combinations (Chabbouh et al., 2025) of multiple features. This flexibility enables the tree to form oblique decision boundaries that more accurately partition the instance space. For example, a single multivariate test such as $x + y < 8$ can replace multiple univariate splits needed to approximate the same boundary. Oblique trees are available in R through packages such as **ODRF**, **aorsf**, and **oblique.tree**.

The construction of Axis-oblique decision trees introduces several design considerations, including how to represent multivariate tests, determine their coefficients, select features to include, handle symbolic and missing data, and prune to avoid over-fitting (Brodley and Utgoff, 1995). Various optimization algorithms such as recursive least squares (Young, 1984), the pocket algorithm (Gallant, 1986), or thermal training (Frean, 1990) may be used to estimate the weights. However, Axis-oblique decision trees trade interpretability for representational power and often require additional mechanisms for **local feature selection**, such as *sequential forward selection* (SFS) or *sequential backward elimination* (SBE) (Kittler, 1986).

Although oblique splits introduce additional complexity and reduce interpretability, oblique decision trees retain key advantages of standard decision trees such as sequential split criteria evaluation and transparent decision procedures while offering improved modeling flexibility for complex data sets (Koziol and Wozniak, 2009; Friedl and Brodley, 1997; Liu and Setiono, 1998).

Alternate Approaches to Oblique Decision Boundaries

Employing linear classifiers at internal nodes can yield decision trees that are both accurate and diverse. Because each node receives a different subset of the training data, the most suitable separating hyperplane may vary from node to node. For example, (Menze et al., 2011; Lemmond et al., 2010) use LDA to construct linear splits, though their approach is limited to binary classification. Zhang and Suganthan (2015) adopts MPSVM to generate node-wise linear separators, while Truong (2009) fits multiple logistic-regression–based partitions (specifically $2^{K-1} - 1$ for K classes) and selects the split that best optimizes the impurity criterion.

Projection pursuit methods (Friedman and Tukey, 2006; Kruskal, 1969) have also been used to identify low-dimensional projections that expose structure in high-dimensional data. The **PPtree** algorithm extends this idea into a recursive partitioning framework by optimizing a class-sensitive projection index at each node and then applying standard split rules to the resulting one-dimensional projection. Similarly, Bulò and Kontschieder (2014) introduce a randomized multi-layer perceptron as a node-level splitting function. However, determining an appropriate network complexity remains challenging, as overly flexible models risk substantial overfitting.

Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are powerful supervised learning models used for classification and regression tasks. They aim to determine an optimal separating hyperplane that maximizes the margin between binary classes in the data. This margin-based approach enhances the generalization ability of the model, making SVMs robust and effective for many real-world problems (Cristianini, 2000).

A simplest **linear SVMs** construct a separating hyperplane in an d-dimensional space such that the margin between the classes is maximized. Given a training dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \{-1, +1\}$, the decision function is defined as $f(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$. The optimal hyperplane is the one that maximizes the distance between the closest points of each class (the **support vectors**) and the hyperplane itself (Cortes and Vapnik, 1995). Training an SVM entails solving a convex quadratic programming (QP) problem defined over an $(n \times n)$ kernel matrix, where (n) is the number of training

samples. Owing to the convexity of the objective, the QP admits a unique global optimum and can be solved reliably using standard optimization techniques. While linear classifiers provide useful insights, they are often inadequate for real-world data sets, where classes are not linearly separable. In such cases, SVMs can be extended to create **nonlinear decision boundaries** by mapping the input vectors into a higher-dimensional **feature space** using a nonlinear transformation $\phi : \mathbb{R}^n \to \mathcal{F}$. The linear transformation is then achieved in the transformed space using $f(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \phi(\mathbf{x}) + b)$.

Although SVMs exhibit strong theoretical foundations and robust generalization capabilities, they present several practical limitations, most notably their lack of interpretability. Model performance is highly dependent on the appropriate selection of hyperparameters such as the regularization term (C) and kernel parameters (e.g., γ), which govern the trade-off between margin maximization and misclassification tolerance (Nanda et al., 2018). For large-scale data sets, training an SVM can become computationally expensive, with both runtime and memory consumption growing quadratically in the number of training samples (Dong et al., 2005). Moreover, SVMs are inherently designed for binary classification, necessitating decomposition strategies such as One-vs-One and One-vs-All for multi-class problems (Hsu and Lin, 2002). Their performance also tends to degrade in imbalanced data settings, where the decision boundary becomes biased toward the majority class (Cervantes et al., 2020).

Oblique Decision Trees with Support Vector Machines

Support Vector Machine–based decision trees (DTSVMs) were first formalized by Bennett and Blue (1998), who adapted Statistical Learning Theory and Structural Risk Minimization to construct binary decision trees in which each internal node is an SVM that partitions the data by an optimal hyperplane. This formulation enabled multivariate, margin-based decisions at every split. Takahashi and Abe (2002) extended the idea to multiclass settings by using a recursive partitioning strategy: the root node isolates the most separable class or classes from the remainder, and the procedure recurses until each leaf contains a single class, thereby covering the entire feature space. Subsequent studies have built upon this foundation to address scalability, optimization, and generalization issues. Optimal Decision Tree SVM (ODT-SVM) (Bala and Agrawal, 2011) introduced split-selection criteria based on Gini index, information gain, and scatter matrix separability to balance tree interpretability and margin-based precision.

STree

More recent approaches embed multiple SVM classifiers directly within each split to handle multi-class problems more efficiently. STree (Montañana et al., 2021) is an oblique multi-class decision tree algorithm that integrates support vector machines (SVMs) to construct single margin-based splits capable of handling multiclass problems. Unlike traditional multi-class tree methods that rely on clustering or ensembles of binary models, STree builds a single decision tree. Its central innovation lies in using SVM-derived hyperplanes at each internal node, enabling more expressive splits than axis-aligned trees while also being interpretable and computationally simple.

Methodology The algorithm generates a binary tree recursively (see Algorithm 5 in Appendix) . At each node:

1) Stopping conditions are evaluated to check the depth of the tree and the class purity. If the stopping conditions are met, a leaf node is created, labelled with the most frequent class in the node.
2) If the stopping conditions haven't been met the algorithm then selects the best hyperplane to split the data into two partitions: T^+ (positive side) and T^- (negative side).

When all instances at a node belong to more than two classes ($k' > 2$), the approach generates multiple candidate binary splits: For each class label y_i, it constructs a one-vs-rest binary problem: Class y_i vs. all other classes. It trains an SVM for each of these k' problems, resulting in k' hyperplanes H_i. Each hyperplane partitions the data into T_i^+ and T_i^-. The algorithm computes the impurity (using Shannon entropy) of the class distribution within each partition. When only two classes remain, STree trains a single SVM to obtain the maximum-margin hyperplane. Instances are then routed left or right based on the sign of their distance to this hyperplane. Stree's performance depends heavily on hyperparameter tuning, including kernel choice (linear, polynomial or Gaussian), gamma, polynomial degree, regularization parameter C, and max iteration.

Synthesis and implications for future work

Across these developments, two recurring themes emerge: (1) leveraging multivariate, oblique decision boundaries to capture complex class geometry, and (2) combining transformation or ensemble strategies (rotations, multiple SVMs) to improve diversity and robustness. These trends point toward hybrid architectures that balance interpretability, margin-based generalization, and computational tractability—an agenda that motivates continued exploration of regularization schemes, split-selection heuristics, and efficient multi-class integration in SVM-based tree models.

3 Contributed Methods

STreeR

Algorithm 1: STreeR: Multi-class OVR SVM-based oblique decision tree

Input: $\mathcal{D}' = \{(\vec{x}_i, y_i)\}_{i=1}^{l'}$: Data
Output: *tree*: the root node of an SVM-based oblique decision tree

function STree($\mathcal{D}'$):
  if *stopping_condition*$(\{y_i\}_{i=1}^{t})$ **then**
    return *create_leaf_node*(*mode*$(\{y_i\}_{i=1}^{t})$) // Leaf node
  $k' \leftarrow$ *num_different_labels*$(\{y_i\}_{i=1}^{t})$
    $\mathcal{Y}' \leftarrow \{y'_1, \ldots, y'_{k'}\}$ // set of labels
  $I_{all} \leftarrow I(\{y_i\}_{i=1}^{t})$ // $I(\cdot)$ is an information theory measure
  $n_{models} \leftarrow k'$ // OvR
  for $j = 1$ **to** n_{models} **do**
    if $(j = k' = 2)$ **then**
      break // Two labels, one SVM is enough
    models$[j] \leftarrow$ SVM$(\mathcal{D}')$ using binary class y'_j (+) vs rest (–) // OvR
  $\mathcal{D}'^{+}, \mathcal{D}'^{-} \leftarrow models[j](\overrightarrow{x}_j) \quad \forall(\overrightarrow{x}) \in \mathcal{D}'$
    $I_j \leftarrow \frac{|\mathcal{D}'^{+}|}{|\mathcal{D}'|} I(\{y_i\}_{i=1}^{|\mathcal{D}'^{+}|}) + \frac{|\mathcal{D}'^{-}|}{|\mathcal{D}'|} I(\{y_i\}_{i=1}^{|\mathcal{D}'^{-}|})$
    $IG_j \leftarrow I_{all} - I_j$
  $b^* = \arg\max_{j=1,\ldots,n_{models}} IG_j$
  if $IG_{b^*} > 0$ **then**
    node $\leftarrow$ *create_node*(*models*$[b^*]$)
      node.left $\leftarrow$ *STree*$(\mathcal{D}'^{+})$
      node.right $\leftarrow$ *STree*$(\mathcal{D}'^{-})$
  else
    return *create_leaf_node*(*model*$(\{y_i\}_{i=1}^{t})$) // No split gain
  return *node*

A key contribution of this work is the first native R implementation of the Stree algorithm. Because no such implementation currently exists, we reconstructed the method in R using **e1071**, which interfaces with the LIBSVM C++ backend. This reconstruction provided clarity on the algorithm's internal workflow and ensured fair benchmarking against competing approaches. Algorithm 1 provides a concise summary of our R-based re-implementation of the Stree algorithm.

To verify the correctness of our R implementation of STree, we conducted a benchmarking comparison against the original Python version using a 10 × 5-fold cross-validation scheme applied to 10 benchmark datasets from the UCI Machine Learning Repository. For comparability, both algorithms were run with the same default hyperparameters: cost-complexity $C = 1$, a linear kernel, a maximum of 1 x 10^7 training iterations, and a maximum depth of 10. Prediction accuracies were averaged across all folds for each dataset.

Table 1 compares the average mean accuracy of both the algorithms. Across most datasets, StreeR exhibits slightly stronger performance than its Python counterpart. Although both implementations rely on the same underlying `LIBSVM C++` library; Python via the `SVC` classifier and R via the `e1071::svm` interface, there are subtle but meaningful differences in how each framework calls the underlying LIBSVM code that fits the underlying SVM decision boundary and applies scaling parameters during

Table 1: Comparison of Mean Prediction Accuracy and Median Training Time for STreeR and STree. N denotes the number of observations, X the number of features, and L the number of classes.

Dataset	N	X	L	StreeR	STree	StreeR(Med)	Stree(Med)
WDBC Diagnosis	569	30	2	**0.970 ± 0.004**	**0.970 ± 0.003**	0.043ms	0.019ms
Iris	150	4	3	**0.966 ± 0.006**	0.965 ± 0.010	0.011ms	0.007ms
Echocardiogram	131	10	2	**0.846 ± 0.004**	0.845 ± 0.008	0.007ms	0.006ms
Fertility	100	9	2	**0.876 ± 0.011**	0.874 ± 0.010	0.003ms	0.005ms
Wine	178	12	3	**0.978 ± 0.008**	0.959 ± 0.008	0.023ms	0.007ms
Cardiotography-3	2126	21	3	0.901 ± 0.003	**0.903 ± 0.003**	0.97ms	0.242ms
Cardiotography-10	2126	21	10	0.792 ± 0.004	**0.795 ± 0.018**	4.391ms	0.474ms
Ionosphere	351	33	2	**0.896 ± 0.009**	0.892 ± 0.015	0.049ms	0.021ms
Dermatology	366	34	6	**0.972 ± 0.006**	0.959 ± 0.004	0.126ms	0.019ms
Statlog Australian Credit	690	14	2	**0.678 ± 0.000**	**0.678 ± 0.000**	0.033ms	0.018ms

prediction. These implementation-level discrepancies likely contribute to the observed performance variation.

Given computational and scope constraints, our re-implementation focuses only on the components of the STree algorithm that are directly relevant to our research objectives. In particular, we restrict our analysis to the one-vs-rest splitting strategy. The alternative one-vs-one scheme is not included, as it scales poorly with the number of classes: for problems with large L, the number of pairwise SVMs grows quadratically, substantially increasing training time without offering clear benefits for the aspects of the algorithm we aim to study. We also implement only a linear kernel, both to maintain the interpretability of the resulting tree structure and to avoid the considerable computational overhead associated with non-linear kernels. At present, the implementation does not include hyperparameters for feature subsetting, although users can replicate this functionality by manually providing a reduced feature set.

Support Vector Machine based Oblique Decision Trees: SVMODT

Algorithm 2: SVMODT: SVM-based Oblique Decision Tree with Enhanced Splitting

Input: $\mathcal{D} = \{(\vec{x}_i, y_i)\}_{i=1}^{n}$: training dataset, d: current depth, $d_{\max}$: maximum depth, $n_{\min}$: minimum samples, m_{base}: base number of features, $\mathcal{C}$: set of all possible class labels

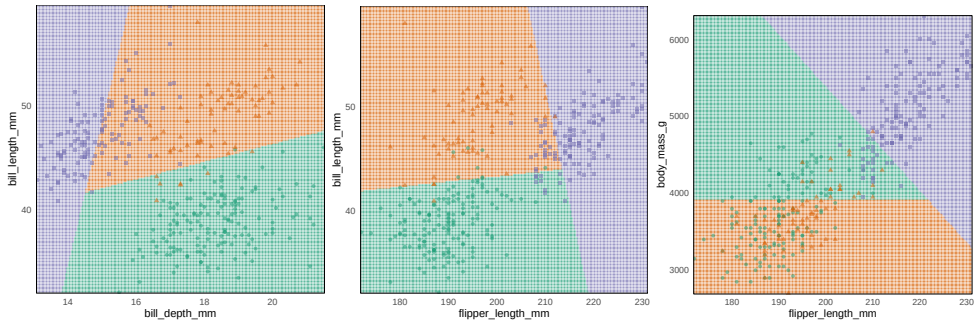
Output: *tree*: root node of the SVM-based oblique decision tree

function SVMODT($\mathcal{D}, d, d_{\max}, n_{\min}, m_{base}, method_{\mathcal{F}}, measure_I, strat_m, strat_w, \mathcal{F}_{used}, \lambda_{pen}, \tau_I, \mathcal{C}$):
- $n \leftarrow |\mathcal{D}|$
- $\mathcal{Y} \leftarrow \{y_i\}_{i=1}^{n}$ // Extract labels
- // Stopping conditions
- **if** $d > d_{\max}$ ***or*** $|unique(\mathcal{Y})| = 1$ ***or*** $n < n_{\min}$ **then**
  - **return** create_leaf_node(mode($\mathcal{Y}$), $n, \mathcal{C}$)
- // Dynamic feature selection and scaling
- $m_d \leftarrow$ calculate_dynamic_max_features($d, m_{\text{base}}, \text{strat}_m$)
- $\mathcal{F}_d \leftarrow$ select_features_with_penalty($\mathcal{D}, m_d, \text{method}_{\mathcal{F}}, \mathcal{F}_{\text{used}}, \lambda_{\text{pen}}$)
- $\mathbf{X}_{\text{scaled}}, \text{scaler} \leftarrow$ scale_node($\mathcal{D}[\mathcal{F}_d]$)
- $\mathcal{F}_d \leftarrow$ features($\mathbf{X}_{\text{scaled}}$) // Update to non-constant features
- **if** $|\mathcal{F}_d| = 0$ **then**
  - **return** create_leaf_node(mode($\mathcal{Y}$), $n, \mathcal{C}$) // No valid features
- $k \leftarrow |\text{unique}(\mathcal{Y})|$ // Number of classes at node
- $I_{\text{parent}} \leftarrow I(\mathcal{Y}, \text{measure}_I)$ // Parent impurity
- // Binary vs Multiclass handling
- **if** $k = 2$ **then**
  - (best_model, $\mathcal{I}_L, \mathcal{I}_R, c_{\text{split}}$) $\leftarrow$ BinarySplit($\mathbf{X}_{scaled}, \mathcal{Y}, strat_w$)
- **else**
  - (best_model, $\mathcal{I}_L, \mathcal{I}_R, c_{\text{split}}$) $\leftarrow$ MulticlassSplit($\mathbf{X}_{scaled}, \mathcal{Y}, strat_w, I_{parent}, measure_I, \tau_I, n$)
- **if** *best_model* $= null$ ***or*** $|\mathcal{I}_L| = 0$ ***or*** $|\mathcal{I}_R| = 0$ **then**
  - **return** create_leaf_node(mode($\mathcal{Y}$), $n, \mathcal{C}$)
- // Handle small children
- **if** $|\mathcal{I}_L| < n_{\min}$ ***and*** $|\mathcal{I}_R| < n_{\min}$ **then**
  - **return** create_leaf_node(mode($\mathcal{Y}$), $n, \mathcal{C}$) // Both children too small
- // Update used features for penalty mechanism
- $\mathcal{F}'_{\text{used}} \leftarrow \mathcal{F}_{\text{used}} \cup \mathcal{F}_d$
- // Create internal node
- *node* $\leftarrow$ create_internal_node(best_model, $\mathcal{F}_d$, scaler, c_{split}, d, n)
- // Recursively build children
- **if** $|\mathcal{I}_L| \geq n_{\min}$ **then**
  - *node*.left $\leftarrow$ SVMODT($\mathcal{D}[\mathcal{I}_L], d+1, d_{\max}, n_{\min}, m_{\text{base}}, \ldots, \mathcal{F}'_{\text{used}}, \ldots, \mathcal{C}$)
- **else**
  - *node*.left $\leftarrow$ create_leaf_node(mode($\mathcal{Y}[\mathcal{I}_L]$), $|\mathcal{I}_L|, \mathcal{C}$)
- **if** $|\mathcal{I}_R| \geq n_{\min}$ **then**
  - *node*.right $\leftarrow$ SVMODT($\mathcal{D}[\mathcal{I}_R], d+1, d_{\max}, n_{\min}, m_{\text{base}}, \ldots, \mathcal{F}'_{\text{used}}, \ldots, \mathcal{C}$)
- **else**
  - *node*.right $\leftarrow$ create_leaf_node(mode($\mathcal{Y}[\mathcal{I}_R]$), $|\mathcal{I}_R|, \mathcal{C}$)
- **return** *node*

The Support Vector Machine based Oblique Decision Tree (SVMODT) constructs a binary classification decision tree where each internal node v at depth d. The algorithm (see Algorithm 2) is implemented in R using the **e1071** performs the following operations:

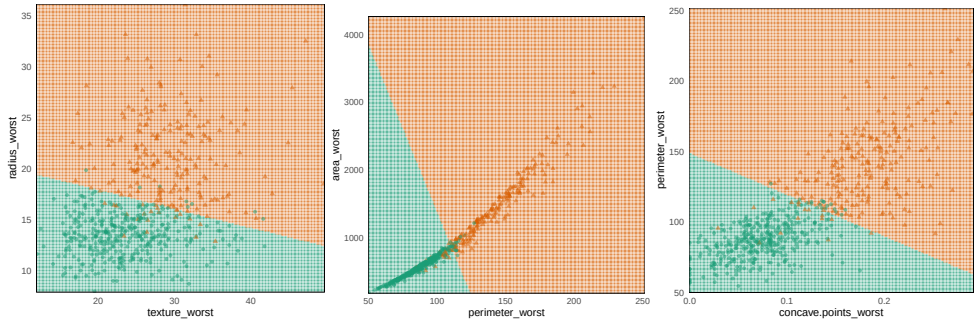
Key Features

Feature Selection At each node, a subset of m_d features is dynamically drawn from the full feature set $\mathcal{F}$. Several selection strategies are supported, including random sampling, mutual-information ranking (via the **FSelectorRcpp** package), and correlation-based filtering. This mechanism enables the tree to concentrate on the most informative predictors at each stage of the recursion. For random-feature strategies, multiple candidate feature subsets are generated; an SVM is fitted on each subset, and the subset yielding the highest information gain is selected for the node.



(a) random **(b)** mutual **(c)** correlated

Figure 2: Comparison of Random, Mutual and Correlated Feature Selection by SVMODT on Palmer-penguins dataset.



(a) random **(b)** mutual **(c)** correlated

Figure 3: Comparison of Random, Mutual and Correlated Feature Selection by SVMODT on Wisconsin Breast Cancer Diagnosis dataset.

The algorithm also supports three different strategies for selecting features at each node. The maximum number of features considered at depth d is defined as:

$$m_d = \begin{cases} m_{\text{base}} & \text{constant strategy} \\ \lfloor m_{\text{base}} \cdot \alpha^{d-1} \rfloor & \text{decrease strategy} \\ \text{Uniform}(\lfloor p \cdot \ell_{\min} \rfloor, \lfloor p \cdot \ell_{\max} \rfloor) & \text{random strategy} \end{cases}$$

Here, $\alpha \in (0, 1]$ is the decrease rate for the "decrease strategy," p is the total number of features, $\ell_{\min}$ and $\ell_{\max}$ define the fractional bounds for the random strategy, and m_{base} is the base number of features at the first depth.

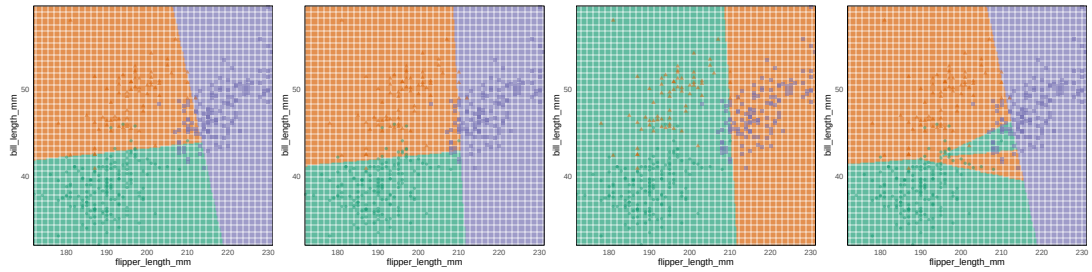
Class Weight Handling To address class imbalance, the algorithm allows four weighting schemes: - **None:** $w_c = 1$ for all classes.

- **Balanced:** $w_c = \frac{n}{K \cdot n_c}$, where K is the total number of classes and n_c is the number of samples in class c.
- **Balanced sub-sample:** $w_c = \frac{1}{n_c} \cdot \frac{K}{\sum_{c'} 1/n_{c'}}$, which adjusts weights for sub-sampled data.
- **Custom:** Users can define their own class weights.

Feature Scaling Once the features are selected, z-score normalization is applied to standardize them. For the selected feature matrix $\mathbf{X}_v \in \mathbb{R}^{n \times m_d}$, the scaled features are computed as

$$\mathbf{X}_v^{\text{scaled}} = \frac{\mathbf{X}_v - \boldsymbol{\mu}_v}{\boldsymbol{\sigma}_v},$$

where $\boldsymbol{\mu}_v$ and $\boldsymbol{\sigma}_v$ denote the mean and standard deviation of the features in node v.



(a) none **(b)** balanced **(c)** balanced subsample **(d)** custom

Figure 4: Comparison of No, Balanced, Balanced-Subsample and Custom Weights by SVMODT on Palmerpenguins dataset.

Feature Penalty To encourage diversity in feature usage across the tree, previously used features have their selection weight reduced by a factor of $(1 - \lambda)$, where $\lambda \in [0, 1)$.

SVM Training A linear SVM is then trained on the scaled features. Optional class weights w_c can be applied to handle imbalanced data. The SVM optimization problem at each node v is formulated as

$$\min_{\mathbf{w_v}, b_v} \frac{1}{2} \|\mathbf{w_v}\|^2 + C \sum_{i \in D_v}^{n} w_{y_i} \xi_i,$$

where C is the regularization parameter, ξ_i are slack variables, and w_{y_i} represents the class-specific weight for sample i in the set of training samples D_v reaching node v.

Algorithm 3: Binary Classification Split

Input: $\mathbf{X}_{\text{scaled}}$: scaled features, $\mathcal{Y}$: labels, strat_w: class weighting strategy
Output: $(\text{model}, \mathcal{I}_L, \mathcal{I}_R, c_{\text{split}})$: SVM model, left indices, right indices, split class

function `BinarySplit`($\mathbf{X}_{scaled}, \mathcal{Y}, strat_w$)**:**
  $w_c \leftarrow \text{calculate\_class\_weights}(\mathcal{Y}, \text{strat}_w)$
  $\text{model} \leftarrow \text{fit\_linear\_SVM}(\mathbf{X}_{\text{scaled}}, \mathcal{Y}, w_c)$
  if *model* $=$ *null* **then**
    return $(\text{null}, \emptyset, \emptyset, \text{null})$
  $\mathbf{f} \leftarrow \text{model.decision\_values}(\mathbf{X}_{\text{scaled}})$
  $\mathcal{I}_L \leftarrow \{i : f_i > 0\}, \quad \mathcal{I}_R \leftarrow \{i : f_i \leq 0\}$
  return $(\text{model}, \mathcal{I}_L, \mathcal{I}_R, \text{null})$

Table 2: Comparing Mean Prediction Accuracy with default arguments - STree(R), STree(Python), SVMODT

Dataset	Stree(R)	STree(Python)	SVMODT
WDBC Diagnosis	**0.970 ± 0.004**	**0.970 ± 0.003**	0.969 ± 0.003
Iris	**0.966 ± 0.006**	0.965 ± 0.010	**0.966 ± 0.006**
Echocardiogram	**0.846 ± 0.004**	0.845 ± 0.008	**0.846 ± 0.006**
Fertility	**0.876 ± 0.011**	0.874 ± 0.010	0.874 ± 0.010
Wine	**0.978 ± 0.008**	0.959 ± 0.008	**0.978 ± 0.008**
Cardiotography-3	0.901 ± 0.003	**0.903 ± 0.003**	0.902 ± 0.004
Cardiotography-10	0.792 ± 0.004	**0.795 ± 0.018**	0.792 ± 0.004
Ionosphere	**0.896 ± 0.009**	0.892 ± 0.015	**0.896 ± 0.011**
Dermatology	**0.972 ± 0.006**	0.959 ± 0.004	**0.972 ± 0.006**
Statlog Australian Credit	**0.678 ± 0.000**	**0.678 ± 0.000**	**0.678 ± 0.000**

Algorithm 4: Multiclass One-vs-Rest Split

Input: $\mathbf{X}_{\text{scaled}}$: scaled features, $\mathcal{Y}$: labels, strat_w: class weighting strategy, I_{parent}: parent impurity, measure_I: impurity measure, τ_I: minimum information gain, n: number of samples

Output: $(\text{best\_model}, \mathcal{I}_L, \mathcal{I}_R, c_{\text{split}})$: best SVM model, left indices, right indices, split class

function `MulticlassSplit`$(\mathbf{X}_{scaled}, \mathcal{Y}, strat_w, I_{parent}, measure_I, \tau_I, n)$**:**

- $k \leftarrow |\text{unique}(\mathcal{Y})|$
- $n_{\text{models}} \leftarrow k$
- **for** $j = 1$ **to** n_{models} **do**
  - $c_j \leftarrow \text{unique}(\mathcal{Y})[j]$ `// Target class for OvR`
  - $\mathcal{Y}_{\text{binary}} \leftarrow \begin{cases} + & \text{if } y_i = c_j \\ - & \text{otherwise} \end{cases}$
  - $w_c \leftarrow \text{calculate\_class\_weights}(\mathcal{Y}_{\text{binary}}, \text{strat}_w)$
  - $\text{model}_j \leftarrow \text{fit\_linear\_SVM}(\mathbf{X}_{\text{scaled}}, \mathcal{Y}_{\text{binary}}, w_c)$
  - **if** $model_j = null$ **then**
    - **continue** `// Skip failed SVM`
  - $\mathbf{f}_j \leftarrow \text{model}_j.\text{decision\_values}(\mathbf{X}_{\text{scaled}})$
  - $\mathcal{I}_L^j \leftarrow \{i : f_{j,i} > 0\}, \quad \mathcal{I}_R^j \leftarrow \{i : f_{j,i} \leq 0\}$
  - **if** $|\mathcal{I}_L^j| = 0$ ***or*** $|\mathcal{I}_R^j| = 0$ **then**
    - **continue** `// Skip degenerate split`
  - $I_L^j \leftarrow I(\mathcal{Y}[\mathcal{I}_L^j], \text{measure}_I)$
  - $I_R^j \leftarrow I(\mathcal{Y}[\mathcal{I}_R^j], \text{measure}_I)$
  - $I_j \leftarrow \frac{|\mathcal{I}_L^j|}{n} \cdot I_L^j + \frac{|\mathcal{I}_R^j|}{n} \cdot I_R^j$
  - $IG_j \leftarrow I_{\text{parent}} - I_j$
  - **if** $IG_j < \tau_I$ **then**
    - **continue** `// Insufficient information gain`
- $b^* \leftarrow \arg\min_j I_j$ `// Select best split`
- **if** $model_{b^*} = null$ **then**
  - **return** $(\text{null}, \emptyset, \emptyset, \text{null})$
- **return** $(\text{model}_{b^*}, \mathcal{I}_L^{b^*}, \mathcal{I}_R^{b^*}, c_{b^*})$

Node Splitting After training, the decision values for each sample are computed as $f(x_i) = \mathbf{w}^T x_i + b$. Samples are then partitioned into left and right child nodes according to the sign of $f(x_i)$:

- `Left child: $\{i : f(x_i) > 0\}$`
- `Right child: $\{i : f(x_i) \leq 0\}$`

This process recursively continues for each child node until stopping criteria are met.

4 Experiment

5 Acknowledgements

The authors would like to thank Professor **Natalia Da Silva** and Professor **Dianne Helen Cook** for their constructive discussions and valuable insights regarding oblique decision trees and their implementations in R. The authors also acknowledges the use of OpenAI's GPT-5 model (ChatGPT) for editorial assistance and technical writing support during the preparation of this manuscript. All conceptual development, analysis, data processing, coding, and interpretation of results were performed independently by the author. The AI tool was used solely to refine language clarity, improve structure, and ensure stylistic consistency in the documentation.

6 Appendix

Algorithm 5: STree (Multi-class SVM-based oblique decision tree)

Input: $\mathcal{D}' = \{(\vec{x}_i, y_i)\}_{i=1}^{l'}$: Data
Output: *tree*: the root node of an SVM-based oblique decision tree

function `STree`($\mathcal{D}'$)**:**
  if $stopping\_condition(\{y_i\}_{i=1}^{t})$ **then**
    return $create\_leaf\_node(mode(\{y_i\}_{i=1}^{t}))$ `// Leaf node`
  $k' \leftarrow num\_different\_labels(\{y_i\}_{i=1}^{t})$
    $r \leftarrow \frac{k'(k'-1)}{2}$
    $\mathcal{Y}' \leftarrow \{y'_1, \ldots, y'_{k'}\}$ `// set of labels`
  $I_{all} \leftarrow I(\{y_i\}_{i=1}^{t})$ `// I(·) is an information theory measure`
  if (OvO) **then**
    $n_{models} \leftarrow r$
  else
    $n_{models} \leftarrow k'$ `// OvR`
  for $j = 1$ **to** n_{models} **do**
    if $(j = k' = 2)$ **then**
      break `// Two labels, one SVM is enough`
    if (OvO) **then**
      Let c_a and c_b the pair of labels corresponding to index j
      $models[j] \leftarrow \text{SVM}\left(\mathcal{D}'^{\downarrow c_a} \cup \mathcal{D}'^{\downarrow c_b}\right)$
    else
      $models[j] \leftarrow \text{SVM}(\mathcal{D}')$ using binary class y'_j (+) vs rest (–) `// OvR`
  $\mathcal{D}'^{+}, \mathcal{D}'^{-} \leftarrow models[j](\overrightarrow{x}_j) \quad \forall(\overrightarrow{x}) \in \mathcal{D}'$
  $I_j \leftarrow \frac{|\mathcal{D}'^{+}|}{|\mathcal{D}'|} I(\{y_i\}_{i=1}^{|\mathcal{D}'^{+}|}) + \frac{|\mathcal{D}'^{-}|}{|\mathcal{D}'|} I(\{y_i\}_{i=1}^{|\mathcal{D}'^{-}|})$
  $IG_j \leftarrow I_{all} - I_j$
  $b^* = \arg\max_{j=1,\ldots,n_{models}} IG_j$
  if $IG_{b^*} > 0$ **then**
    $node \leftarrow create\_node(models[b^*])$
    $node.left \leftarrow STree(\mathcal{D}'^{+})$
    $node.right \leftarrow STree(\mathcal{D}'^{-})$
  else
    return $create\_leaf\_node(model(\{y_i\}_{i=1}^{t}))$ `// No split gain`
  return *node*

Bibliography

M. Bala and R. Agrawal. Optimal decision tree based multi-class support vector machine. *Informatica*, 35(2), 2011. [p1, 4]

K. Bennett and J. Blue. A support vector machine approach to decision trees. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, volume 3, pages 2396–2401 vol.3, 1998. doi: 10.1109/IJCNN.1998.687237. [p1, 4]

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984. URL http://lyle.smu.edu/~mhd/8331f06/cart.pdf. [p1]

C. E. Brodley and P. E. Utgoff. Multivariate decision trees. *Machine Learning*, 19:45–77, 1995. URL https://api.semanticscholar.org/CorpusID:16836572. [p2, 3]

S. Bulò and P. Kontschieder. Neural decision forests for semantic image labelling. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 81–88, 2014. doi: 10.1109/CVPR.2014.18. [p3]

L. Cañete, R. Monroy, and M. Medina-Pérez. A review and experimental comparison of multivariate decision trees. *IEEE Access*, PP:1–1, 08 2021. doi: 10.1109/ACCESS.2021.3102239. [p1]

J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez. A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, 408:189–215, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2019.10.118. URL https://www.sciencedirect.com/science/article/pii/S0925231220307153. [p4]

M. Chabbouh, S. Bechikh, E. Mezura-Montes, and L. Ben Said. Evolutionary optimization of the area under precision-recall curve for classifying imbalanced multi-class data. *Journal of Heuristics*, 31, 01 2025. doi: 10.1007/s10732-024-09544-z. [p3]

C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. [p1, 3]

N. Cristianini. An introduction to support vector machines and other kernel-based learning methods, 2000. [p1, 3]

J.-x. Dong, A. Krzyzak, and C. Y. Suen. Fast svm training algorithm with decomposition on very large data sets. *IEEE transactions on pattern analysis and machine intelligence*, 27(4):603–618, 2005. [p4]

M. R. Frean. *Small nets and short paths: Optimising neural computation*. PhD thesis, 1990. [p3]

M. Friedl and C. Brodley. Decision tree classification of land cover from remotely sensed data. *Remote Sensing of Environment*, 61(3):399–409, 1997. ISSN 0034-4257. doi: https://doi.org/10.1016/S0034-4257(97)00049-7. URL https://www.sciencedirect.com/science/article/pii/S0034425797000497. [p3]

J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on computers*, 100(9):881–890, 2006. [p3]

S. I. Gallant. Optimal linear discriminants. *Eighth International Conference on Pattern Recognition*, pages 849–852, 1986. URL https://cir.nii.ac.jp/crid/1573668924476144256. [p3]

M. Ganaie, M. Tanveer, P. Suganthan, and V. Snasel. Oblique and rotation double random forest. *Neural Networks*, 153:496–517, 2022. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2022.06.012. URL https://www.sciencedirect.com/science/article/pii/S0893608022002258. [p1]

C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425, 2002. [p4]

J. Kittler. Feature selection and extraction. *Handbook of Pattern Recognition and Image Processing*, pages 59–83, 1986. URL https://cir.nii.ac.jp/crid/1573950400671608448. [p3]

M. Koziol and M. Wozniak. *Multivariate Decision Trees vs. Univariate Ones*, pages 275–284. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-540-93905-4. doi: 10.1007/978-3-540-93905-4_33. URL https://doi.org/10.1007/978-3-540-93905-4_33. [p3]

J. B. Kruskal. Toward a practical method which helps uncover the structure of a set of multivariate observations by finding the linear transformation which optimizes a new "index of condensation". In *Statistical computation*, pages 427–440. Elsevier, 1969. [p3]

T. D. Lemmond, B. Y. Chen, A. O. Hatch, and W. G. Hanley. *An Extended Study of the Discriminant Random Forest*, pages 123–146. Springer US, Boston, MA, 2010. ISBN 978-1-4419-1280-0. doi: 10.1007/978-1-4419-1280-0_6. URL https://doi.org/10.1007/978-1-4419-1280-0_6. [p3]

H. Liu and R. Setiono. Feature transformation and multivariate decision tree induction. In S. Arikawa and H. Motoda, editors, *Discovey Science*, pages 279–291, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-49292-4. [p3]

V. Menkovski, I. T. Christou, and S. Efremidis. Oblique decision trees using embedded support vector machines in classifier ensembles. In *2008 7th IEEE International Conference on Cybernetic Intelligent Systems*, pages 1–6, 2008. doi: 10.1109/UKRICIS.2008.4798937. [p1]

B. H. Menze, B. M. Kelm, D. N. Splitthoff, U. Koethe, and F. A. Hamprecht. On oblique random forests. In D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 453–469, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-23783-6. [p3]

J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 01 1989. doi: 10.1023/A:1022604100933. [p3]

R. Montañana, J. A. Gámez, and J. M. Puerta. Stree: A single multi-class oblique decision tree based on support vector machines. In E. Alba, G. Luque, F. Chicano, C. Cotta, D. Camacho, M. Ojeda-Aciego, S. Montes, A. Troncoso, J. Riquelme, and R. Gil-Merino, editors, *Advances in Artificial Intelligence*, pages 54–64, Cham, 2021. Springer International Publishing. ISBN 978-3-030-85713-4. [p4]

R. Montañana, J. A. Gámez, and J. M. Puerta. Stree: a single multi-class oblique decision tree based on support vector machines, 2021. [p1]

M. Nanda, K. Seminar, D. Nandika, and A. Maddu. A comparison study of kernel functions in the support vector machine and a comparison study of kernel functions in the support vector machine and its application for termite detection, 2018. [p4]

G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990. URL https://api.semanticscholar.org/CorpusID:5661437. [p1, 3]

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. [p1]

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986. URL https://api.semanticscholar.org/CorpusID:189902138. [p2]

J. R. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014. [p1, 3]

C. Schaffer. Deconstructing the digit recognition problem. In D. Sleeman and P. Edwards, editors, *Machine Learning Proceedings 1992*, pages 394–399. Morgan Kaufmann, San Francisco (CA), 1992. ISBN 978-1-55860-247-2. doi: https://doi.org/10.1016/B978-1-55860-247-2.50056-5. URL https://www.sciencedirect.com/science/article/pii/B9781558602472500565. [p3]

F. Takahashi and S. Abe. Decision-tree-based multiclass support vector machines. volume 3, pages 1418 – 1422 vol.3, 12 2002. ISBN 981-04-7524-1. doi: 10.1109/ICONIP.2002.1202854. [p1, 4]

A. K. Y. Truong. Fast growing and interpretable oblique trees via logistic regression models. 2009. URL https://api.semanticscholar.org/CorpusID:57884720. [p3]

P. C. Young. Recursive estimation and time-series analysis: An introduction. 1984. URL https://api.semanticscholar.org/CorpusID:60181335. [p3]

L. Zhang and P. N. Suganthan. Oblique decision tree ensemble via multisurface proximal support vector machine. *IEEE Transactions on Cybernetics*, 45(10):2165–2176, 2015. doi: 10.1109/TCYB.2014.2366468. [p3]

Aneesh Agarwal
Monash University

aaga0022@student.monash.edu

Jack Jewson
Monash University
Department of Econometrics and Business Statistics, Monash University, Australia
Jack.Jewson@monash.edu

Erik Sverdrup
Monash University
Department of Econometrics and Business Statistics, Monash University, Australia
Erik.Sverdrup@monash.edu