

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 two novel decision tree algorithms: (1) a custom decision tree with integrated cost-complexity pruning that makes pruning decisions during construction rather than as post-processing, and (2) an SVM-based oblique decision tree (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 achieves competitive accuracy with standard SVMs (mean accuracy 0.974 on WDBC dataset) while maintaining the interpretability advantages of tree structures. The package includes comprehensive visualization tools, detailed documentation, and is freely available on GitHub.

1 Introduction

Decision trees remain among the most widely adopted machine learning algorithms due to their interpretability, ease of use, and ability to handle both categorical and numerical data without extensive preprocessing. Classical algorithms such as CART [@breiman1984] and C4.5 [@quinlan] employ axis-aligned (univariate) splits that partition the feature space along coordinate axes. While this approach ensures transparency in decision-making, it often necessitates deep trees with many nodes to approximate diagonal or curved decision boundaries, leading to increased model complexity and potential overfitting [@murthy1994].

Oblique decision trees, which allow splits based on linear combinations of features, address this limitation by enabling more flexible partitioning of the feature space. A single oblique split such as $w_1x_1 + w_2x_2 < \theta$ can replace multiple axis-aligned splits needed to approximate the same boundary. Despite their representational advantages, oblique trees have seen limited adoption in practice, partly due to the computational complexity of finding optimal multivariate splits and the scarcity of accessible, well-documented software implementations, particularly in R [@heath1993; @brodley1995].

Support Vector Machines (SVMs), introduced by @cortes1995, provide a principled approach to finding optimal separating hyperplanes by maximizing the margin between classes. The integration of SVMs into decision tree structures was pioneered by @bennet1998, who proposed fitting an SVM at each internal node to determine multivariate splits. This hybrid approach combines the interpretability of trees with the robust margin-based optimization of SVMs. Subsequent work has extended this concept through various algorithmic refinements [@takahashi2002; @bala2011; @ganaie2022], yet fully-featured, accessible implementations remain scarce in the R ecosystem.

Research Objectives

This paper addresses three key limitations in existing decision tree implementations:

First, we tackle the computational inefficiency of traditional two-phase pruning approaches. Classical methods like CART grow a maximal tree before applying cost-complexity pruning via cross-validation [@breiman1984classification]. This approach wastes computation on branches that are ultimately removed and requires tuning multiple hyper-parameters. We propose **integrated cost-complexity pruning**, where pruning decisions are made in real-time during tree construction, eliminating unnecessary computation while maintaining theoretical optimality.

Second, we address the lack of flexible oblique tree implementations in R. While Python's STree [@Montanana2021] and specialized packages exist, R practitioners lack access to SVM-based oblique trees with modern features such as dynamic feature selection, node-specific class weighting, and built-in visualization tools. Our **SVMODT** (SVM-based Oblique Decision Tree) implementation fills this gap with a comprehensive, well-documented package designed for both research and practical applications.

Third, we introduce novel algorithmic enhancements including: (1) **dynamic feature selection strategies** that adjust the number of features considered based on tree depth; (2) **feature diversity through penalization**, which discourages repeated use of the same features across ancestor-descendant

paths; and (3) **node-specific class weighting** that adapts to local class distributions during recursive splitting, providing more effective handling of imbalanced data sets than global weighting schemes.

Organization

The remainder of this paper is organized as follows. Section 2 reviews the theoretical foundations of decision trees, Support Vector Machines, and existing hybrid approaches. Section 3 identifies research gaps and articulates our specific contributions. Section 4 presents the algorithmic details of both implementations with pseudocode. Section 5 describes the data sets used for evaluation. Section 6 demonstrates package usage through practical examples. Section 7 presents a comprehensive comparison with baseline methods including CART, standard SVMs, and logistic regression across 100 random train-test splits. Finally, Section 8 discusses limitations, future extensions, and concludes.

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 internal nodes containing splitting criteria and terminal (leaf) nodes corresponding to class labels. The nodes are connected by directed edges, each representing a possible outcome of a splitting criterion. 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.

If the tree T has m nodes, then for any $j \in \{1, \dots, m\}$, the set of child nodes of $v_j \in V$ can be defined as:

$$N^+(v_j) = \{v_k \in V \mid k \in \{1, \dots, m\}, k \neq j, (v_j, v_k) \in E\}.$$

Here, $N^+(v_j)$ denotes the set of nodes that are directly connected to v_j through outgoing edges, representing all possible child nodes that can be reached from v_j within the tree structure (Rivera-Lopez and Canul-Reich, 2018).

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.

Univariate Decision Trees

Univariate Decision Trees (UDTs) trees represent axis-parallel hyper-planes dividing the instance space into several disjoint 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 capable of handling both continuous and categorical variables as predictors or targets. It operates directly on raw data without requiring binning. The tree is expanded recursively until no further splits are possible, after which **cost-complexity pruning** is applied to remove branches that contribute least to predictive performance. This pruning process generates a sequence of nested sub-trees, from which the optimal model is selected using independent test data or cross-validation, rather than internal training measures (Breiman et al., 1984).

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** method to reduce 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).

While UDTs are highly interpretability, they are characterized by several representational limitations. Such trees often grow unnecessarily large, as they must approximate complex relationship between features through multiple axis-aligned partitions. This can result in the replication of sub-trees

and repeated testing of the same feature along different paths, both of which reduce efficiency and hinder generalization performance (Pagallo and Haussler, 1990).

Multivariate Decision Trees

Multivariate decision trees (MDTs) extends UDTs by allowing each internal node to perform splits based on linear or nonlinear combinations 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. The construction of MDTs 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, MDTs 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).

Empirical comparisons across multiple data sets demonstrate that multivariate trees generally achieve higher accuracy and smaller tree sizes than their univariate counterparts, though this comes at the cost of reduced interpretability. Moreover, MDTs 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; Cañete et al., 2021).

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 different 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).

Linear SVMs

A simplest **linear SVMs** construct a separating hyperplane in an n -dimensional space such that the margin between the classes is maximized. Given a training dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, +1\}$, the decision function is defined as:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x} + b),$$

where \mathbf{w} is the weight vector perpendicular to the hyperplane, and b is the bias term. 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).

Mathematically, the optimization problem for a hard-margin SVM (i.e., assuming the data are linearly separable) can be formulated as:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

subject to:

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \dots, N.$$

Here, $\|\mathbf{w}\|$ represents the norm of the weight vector and acts as a regularization term that controls the complexity of the model. The constraint ensures that all data points are correctly classified and lie outside the margin boundaries.

However, in most practical situations, perfect linear separability is not achievable. To address this, **soft-margin SVMs** introduce slack variables $\xi_i \geq 0$ to allow certain violations of the margin constraints, resulting in the following optimization problem:

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i$$

Table 1: Commonly used kernel functions and their parameters.

Kernel Type	Mathematical Definition	Key Parameters
Linear	$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$	None
Polynomial	$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + 1)^d$	Degree d
Gaussian	$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\ \mathbf{x}_i - \mathbf{x}_j\ ^2}{2\sigma^2}\right)$	Bandwidth σ
RBF	$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \ \mathbf{x}_i - \mathbf{x}_j\ ^2)$	γ
Sigmoid	$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\kappa \mathbf{x}_i^\top \mathbf{x}_j + \theta)$	κ, θ

subject to:

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, 2, \dots, N.$$

The parameter $C > 0$ controls the trade-off between maximizing the margin and minimizing the classification error on the training data. A large C penalizes mis-classifications heavily, leading to a narrower margin, whereas a smaller C allows more flexibility, potentially improving generalization in the presence of noise.

The solution to this constrained optimization problem is obtained using **Lagrange multipliers**, resulting in a dual formulation expressed as:

$$\max_{\alpha} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

subject to:

$$\sum_{i=1}^N \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C.$$

The data points corresponding to non-zero α_i values are the **support vectors**, which define the decision boundary. The resulting decision function for a new observation \mathbf{x} is given by:

$$f(\mathbf{x}) = \text{sign} \left(\sum_{i=1}^N \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b \right).$$

This formulation highlights one of the most important properties of SVMs — the decision boundary depends only on a subset of the training data (the support vectors), making SVMs both efficient and robust in representing the learned model (Cervantes et al., 2020).

Non-Linear SVMs

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 $\phi : \mathbb{R}^n \rightarrow \mathcal{F}$. The linear transformation is then achieved in the transformed space using:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \phi(\mathbf{x}) + b).$$

However, directly computing $\phi(\mathbf{x})$ can be computationally expensive. To address this, SVMs employ the **kernel trick**, where the dot product in the feature space is replaced with a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle.$$

The resulting decision function becomes:

$$f(\mathbf{x}) = \text{sign} \left(\sum_{i=1}^N \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \right),$$

where α_i are the Lagrange multipliers obtained during training.

For a function K to be a valid kernel, it must satisfy **Mercer's condition** (Vapnik, 2013) i.e., the kernel matrix must be symmetric and positive semi-definite. Table 1

Although SVMs exhibit strong theoretical foundations and robust generalization capabilities, they present several practical limitations. 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). Training an SVM requires solving a quadratic programming (QP) optimization problem involving an $n \times n$ kernel matrix, where n denotes the number of training samples, leading to quadratic growth in both computational time and memory usage (Dong et al., 2005). This makes SVMs computationally expensive for large-scale data sets. 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).

Hybrid Decision Trees with Support Vector Machines

The earliest formalization of Support Vector Machine-based Decision Trees (DTSVMs) was introduced by Bennett and Blue (1998), who extended the principles of Statistical Learning Theory (Vapnik, 2013) and Structural Risk Minimization (SRM) to the construction of binary decision trees. Each node of a decision tree was treated as an SVM that partitions data along an optimal hyperplane. This approach allowed for multivariate SVM decisions at each node. TAKAHASHI and Abe (2002) extended DTSVMs to multiclass problems, using a recursive tree-based partitioning where the root separates the most separable class (or classes) from the rest. Subsequent nodes repeat this process until each leaf contains a single class, ensuring that all regions of the feature space are classified.

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.

Oblique and Rotation-Based Extensions

Recent research has explored oblique and rotation-based decision ensembles that generalize the DTSVM concept. Oblique Double Random Forests with MPSVM (MPDRaF) (Ganaie et al., 2022) introduce multivariate (oblique) splits at each node using Multi-Plane SVM (MPSVM) formulations (Mangasarian and Wild, 2006), enhancing the geometric flexibility of decision boundaries. Regularization variants—such as Tikhonov, axis-parallel, and null-space—address sample-size limitations at deeper nodes, ensuring better generalization.

Similarly, Rotation-based Double Random Forests (DRaF) (Ganaie et al., 2022) employ Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) transformations at non-terminal nodes to create diverse subspaces, improving ensemble diversity and classification stability.

Modern Multi-class Integration

Recent works such as the Oblique Decision Tree Ensemble (ODTE) (Montañana et al., 2025) embed multiple SVM classifiers (e.g., one-vs-one or one-vs-rest) directly at each split, dynamically selecting the model that minimizes class impurity. This enables efficient n-ary classification within a single tree structure, mitigating the scalability and imbalance limitations of traditional binary SVM extensions.

Related Software Packages

Several software packages provide functionality related to SVM-based decision trees, but fully-featured implementations of SVM Oblique Decision Trees (SVMODT) remain limited, particularly in R. In R, very scarce SVMODT implementation is available. The **PPforest** package implements projection pursuit random forests, where trees are constructed by splitting on linear combinations of variables to maximize class separation, supporting multi-class problems and variable importance measures. In Python, **STree** (Montañana et al., 2021) provides an oblique decision tree classifier based on SVM nodes, where each node is built and split using scikit-learn's **SVC** models (Pedregosa et al., 2011). Java-based frameworks such as **Weka** provide classical decision trees (J48) and SVMs (SMO) that can be adapted for hierarchical SVM trees (Menkovski et al., 2008). Overall, while multiple languages provide the building blocks for SVM training, decision tree construction, and ensemble methods—there is currently no widely available, fully integrated SVMODT software in R, highlighting the need for custom implementations or adaptation of existing tools for research and applied purposes.

3 Research Gaps & Contributions

Identified Research Gaps

Despite significant advances in decision tree methodologies and SVM-based hybrid approaches, several critical limitations persist in existing implementations:

Limited Flexibility in Oblique Decision Tree Software

As discussed before, while oblique decision trees using SVMs have been theoretically established since [Bennett and Blue \(1998\)](#), accessible and feature-rich implementations remain scarce, particularly in R. Existing approaches such as Optimal Decision Tree SVM (ODT-SVM) ([Bala and Agrawal, 2011](#)) and recent extensions like MPDRaF ([Ganaie et al., 2022](#)) and ODTE ([Montañana et al., 2025](#)) have advanced the field theoretically, but face several practical limitations:

- **Software accessibility:** The [PPforest](#) package provides projection pursuit forests but lacks true SVM-based splitting. Python's [STree](#) ([Montañana et al., 2021](#)) offers oblique trees but is not available in R, leaving a significant gap in the R ecosystem for practitioners who prefer or require R-based workflows.
- **Limited customization:** Existing implementations typically offer rigid feature selection mechanisms, with most using either all features or a fixed random subset at each split. There is very limited support for depth-dependent feature selection strategies or mechanisms to encourage feature diversity across tree paths.
- **Class imbalance handling:** While SVMs inherently support class weighting, most hybrid DTSVM implementations apply weights globally at the tree level rather than computing node-specific weights that adapt to local class distributions encountered during recursive splitting.
- **Interpretability tools:** Despite the advantage of tree-based models in interpretability, existing packages lack comprehensive visualization tools for examining individual SVM hyper-planes, tracing prediction paths, or understanding how oblique splits contribute to final decisions.

Absence of Integrated Pruning in Modern Decision Trees

Classical decision tree algorithms, such as CART ([Breiman et al., 1984](#)), typically employ a two-phase process: a maximal tree is first grown, and pruning is applied afterward using cost-complexity analysis with cross-validation. This approach introduces several inefficiencies. Branches that will ultimately be pruned are fully constructed, consuming unnecessary computational resources, and practitioners must tune both tree-growing parameters (e.g., depth, minimum samples) and pruning parameters (α), increasing model selection complexity. Furthermore, deferring pruning to a post-processing phase prevents early stopping opportunities that could reduce training time. While C4.5 ([Quinlan, 2014](#)) implemented error-based pruning and some modern algorithms include early stopping criteria, integrated cost-complexity pruning; where pruning decisions are made in real time during tree construction remains largely unexplored, despite its potential to improve computational efficiency.

Our Contributions

This paper introduces the [svmodt](#) R package, which addresses the limitations identified in existing decision tree and SVM-based methods through two complementary algorithms and a comprehensive software implementation. Our contributions span algorithmic innovation, theoretical analysis, and practical software engineering.

1. SVM-Based Oblique Decision Trees with Advanced Features

We propose a flexible oblique decision tree algorithm that uses linear SVMs at each internal node. The implementation offers several novel features:

1. **Dynamic Feature Selection Strategies:** We introduce three strategies to control the number of features considered at each depth d :
 - *Constant strategy:* fixed number of features at all depths;
 - *Decrease strategy:* the feature count decreases with depth, mitigating over-fitting and improving computational efficiency;
 - *Random strategy:* a random subset of features is selected at each node, promoting diversity in feature usage. These strategies extend beyond the fixed $m = \sqrt{p}$ approach

used in Random Forests, giving practitioners explicit control over feature complexity throughout the tree.

2. **Feature Diversity via Penalization:** To reduce redundancy and encourage diverse feature usage, features used in ancestor nodes have their selection weights reduced by a factor $1 - \lambda$ controlled via the `penalize_used_features` and `feature_penalty_weight` parameters. Unlike naive random sampling, this approach explicitly discourages repeated feature use while maintaining flexibility.
3. **Node-Specific Class Weighting:** Four weighting strategies are implemented, with weights recalculated at each node based on local class distributions: none, balanced, balanced subsample, and custom user-defined weights. All weights are capped to prevent numerical instability, enabling adaptive handling of class imbalance throughout the tree.

2. Decision Tree with Integrated Cost-Complexity Pruning

We implement a decision tree algorithm where pruning occurs during construction rather than as a post-processing step. At each node, the algorithm immediately compares the cost-complexity of a leaf versus its sub-tree and prunes in real time if beneficial. This approach eliminates the need for separate pruning phases, reduces computational cost, simplifies hyper-parameter tuning, and maintains optimal equivalent to CART's post-pruning.

3. Comprehensive R Package Implementation

The `svmodt` package provides a production-ready implementation with a well-documented API that includes type checking and input validation to ensure robust and user-friendly operation. It supports both binary and multi-class classification and leverages vectorized operations for computational efficiency. For interpretability and diagnostics, the package includes functions such as `print_svm_tree()` for hierarchical ASCII visualization, `trace_prediction_path()` for sample-level decision path tracing, and `visualize_svm_tree()` for automated 2D plotting of SVM hyper-planes. The package also comes with included data sets, such as the Wisconsin Diagnostic Breast Cancer (WDBC) data set. Comprehensive documentation is provided, including vignettes and Roxygen2-based function documentation, along with reproducible example code to facilitate practical application and comparison with alternative methods.

4 Methodology

To better understand the structure and mechanics of decision trees, we first implemented a standard axis-parallel decision tree in R from scratch before extending it into the oblique (SVM-based) variant.

Custom Univariate Decision Tree with Integrated Cost-Complexity Pruning

This implementation presents a decision tree algorithm with **integrated cost-complexity pruning**, where pruning decisions are made during tree construction rather than as a post-processing step. The algorithm supports multiple splitting criteria and performs real-time pruning based on cost-complexity analysis.

Algorithm Description

At each node v with samples \mathcal{D}_v and labels \mathcal{Y}_v , the algorithm:

1. **Compute Node Statistics:** At each node of the decision tree, several key statistics are computed to guide the splitting process. The class probabilities are calculated as $p_c = \frac{n_c}{n}$, where n_c is the count of samples belonging to class c in the node, and $n = |\mathcal{D}_v|$ is the total number of samples at the node. The predicted class is assigned as the class with the highest probability,

$$\hat{y} = \arg \max_c p_c.$$

2. **Select Optimal Split:** The optimal split at each node is determined using one of three criteria:

- **Gini Impurity** is defined as

$$\text{Gini}(\mathcal{Y}) = 1 - \sum_{c=1}^K p_c^2,$$

where p_c is the proportion of samples belonging to class c in node \mathcal{Y} (Breiman et al., 1984).

- **Information Gain** evaluates the reduction in node impurity and is computed as

$$\text{IG}(f, s) = H(\mathcal{Y}_v) - \sum_{i \in \{L, R\}} \frac{n_i}{n} H(\mathcal{Y}_i),$$

where $H(\mathcal{Y})$ denotes the impurity of a node (entropy or Gini), n_i is the number of samples in the child node, and n is the number of samples in the parent node [Breiman et al. (1984)].

- **Gain Ratio** further adjusts Information Gain by penalizing splits that create many small partitions (Quinlan, 2014) and is defined as

$$\text{GR}(f, s) = \frac{\text{IG}(f, s)}{\text{SI}(f, s)},$$

where the split information is

$$\text{SI}(f, s) = - \sum_{i \in \{L, R\}} \frac{n_i}{n} \log_2 \left(\frac{n_i}{n} \right).$$

3. **Evaluates Pruning Criteria:** To ensure the tree remains interpretable and avoids over-fitting, integrated pruning is performed using cost-complexity analysis. The cost of a sub-tree T_v is given by

$$R_\alpha(T_v) = R(T_v) + \alpha \|T_v\|,$$

where $R(T_v)$ is the miss-classification error, $|T_v|$ is the number of leaves in the sub-tree, and $\alpha \geq 0$ is a complexity parameter controlling the trade-off between accuracy and simplicity. For a single leaf node t_v , the cost is

$$R_\alpha(t_v) = R(t_v) + \alpha.$$

A sub-tree T_v is pruned, i.e., replaced by a single leaf node, if

$$R_\alpha(t_v) \leq R_\alpha(T_v),$$

ensuring that further splits are retained only when they yield sufficient reduction in classification error to justify the added complexity.

The custom univariate decision tree implementation in R combines axis-parallel splits with integrated cost-complexity pruning, offering several key differences from standard CART/rpart trees. Numeric features are split only at midpoints where class labels change, while categorical features use binary splits at each level, reducing computational complexity. Splitting criteria include Gini impurity, information gain, and Gain Ratio—the latter penalizing splits with high split information to favor balanced partitions. Integrated pruning is applied during tree construction using a fixed α parameter, replacing sub-trees with leaves whenever the cost-complexity criterion is satisfied, unlike CART/rpart, which performs post-pruning via cross-validation. This single-pass pruning approach reduces training time and space requirements, while still producing interpretable trees. Predictions follow standard tree traversal, returning majority-class labels or probability estimates at leaves.

Support Vector Machine based Oblique Decision Trees

Algorithm Description

The SVM Oblique Decision Tree (SVMODT) constructs a binary classification decision tree where each internal node v at depth d . The algorithm is implemented in R using the e1071 package and performs the following operations:

1. **Feature Selection:** A subset of m_d features is dynamically selected from the available feature set \mathcal{F} . This selection can be done using one of several strategies, including random selection, sampling, mutual information ranking (using the FSelectorRcpp package), or correlation-based selection. This allows the tree to adaptively focus on the most informative features at each node.
2. **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

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

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

Algorithm 1 Decision Tree with Integrated Cost-Complexity Pruning

```

1: Procedure GenerateTree( $\mathcal{Y}, \mathbf{X}, d, d_{\max}, \alpha$ )
2:  $n \leftarrow |\mathcal{Y}|$ 
3:  $\mathbf{p} \leftarrow \text{ClassProbabilities}(\mathcal{Y})$ 
4:  $\hat{y} \leftarrow \arg \max_c p_c$ 
5: if  $d \geq d_{\max}$  or  $|\text{unique}(\mathcal{Y})| \leq 1$  then
6:   return LeafNode( $\hat{y}, \mathbf{p}, n$ )
7: end if
8:  $(f^*, s^*, \text{score}) \leftarrow \text{FeatureSelector}(\mathcal{Y}, \mathbf{X}, \text{criterion})$ 
9: if  $f^* = \text{null}$  or  $\text{score} = -\infty$  then
10:   return LeafNode( $\hat{y}, \mathbf{p}, n$ )
11: end if
12:  $\mathcal{I}_L, \mathcal{I}_R \leftarrow \text{Split}(\mathbf{X}[f^*], s^*)$ 
13: if  $|\mathcal{I}_L| = 0$  or  $|\mathcal{I}_R| = 0$  then
14:   return LeafNode( $\hat{y}, \mathbf{p}, n$ )
15: end if
16:  $T_L \leftarrow \text{GENERATETREE}(\mathcal{Y}[\mathcal{I}_L], \mathbf{X}[\mathcal{I}_L], d + 1, d_{\max}, \alpha)$ 
17:  $T_R \leftarrow \text{GENERATETREE}(\mathcal{Y}[\mathcal{I}_R], \mathbf{X}[\mathcal{I}_R], d + 1, d_{\max}, \alpha)$ 
18:  $T_v \leftarrow \text{InternalNode}(f^*, s^*, T_L, T_R, \hat{y}, \mathbf{p}, n)$ 
19: {Integrated pruning decision}
20:  $R_{\text{subtree}} \leftarrow \text{SubtreeError}(T_v) + \alpha \cdot \text{CountLeaves}(T_v)$ 
21:  $R_{\text{leaf}} \leftarrow (1 - \max(\mathbf{p})) + \alpha$ 
22: if  $R_{\text{leaf}} \leq R_{\text{subtree}}$  then
23:   return LeafNode( $\hat{y}, \mathbf{p}, n$ ) {Prune subtree}
24: else
25:   return  $T_v$ 
26: end if
27: End Procedure

```

3. **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 is formulated as

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n w_{y_i} \xi_i,$$

where C is the regularization parameter, ξ_i are slack variables, and w_{y_i} represents the class-specific weight for sample i .

4. **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.

Key Hyper-parameters

1. **Feature Selection:** The algorithm supports three 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, ℓ_{\min} and ℓ_{\max} define the fractional bounds for the random strategy, and m_{base} is the base number of features at the first depth.

Algorithm 2 SVM Oblique Decision Tree Construction

```

1: Procedure SVMSplit( $\mathcal{D}, d, d_{\max}, n_{\min}$ )
2:  $n \leftarrow |\mathcal{D}|, \mathcal{Y} \leftarrow \text{labels}(\mathcal{D})$ 
3: if  $d > d_{\max}$  or  $|\text{unique}(\mathcal{Y})| = 1$  or  $n < n_{\min}$  then
4:   return LeafNode( $\mathcal{Y}$ ) {Stopping criteria}
5: end if
6:  $m_d \leftarrow \text{DynamicMaxFeatures}(d, m_{\text{base}}, \text{strategy})$ 
7:  $\mathcal{F}_d \leftarrow \text{SelectFeatures}(\mathcal{D}, m_d, \text{method})$ 
8:  $\mathbf{X}_{\text{scaled}}, \text{scaler} \leftarrow \text{Scale}(\mathcal{D}[\mathcal{F}_d])$ 
9: if  $|\mathcal{F}_d| = 0$  then
10:   return LeafNode( $\mathcal{Y}$ ) {No valid features}
11: end if
12:  $w_c \leftarrow \text{CalculateClassWeights}(\mathcal{Y}, \text{strategy})$ 
13:  $\text{SVM} \leftarrow \text{FitLinearSVM}(\mathbf{X}_{\text{scaled}}, \mathcal{Y}, w_c)$ 
14: if  $\text{SVM} = \text{null}$  then
15:   return LeafNode( $\mathcal{Y}$ ) {SVM fitting failed}
16: end if
17:  $\mathbf{f} \leftarrow \text{SVM.DecisionValues}(\mathbf{X}_{\text{scaled}})$ 
18:  $\mathcal{I}_L \leftarrow \{i : f_i > 0\}, \mathcal{I}_R \leftarrow \{i : f_i \leq 0\}$ 
19: if  $|\mathcal{I}_L| = 0$  or  $|\mathcal{I}_R| = 0$  then
20:   return LeafNode( $\mathcal{Y}$ ) {Ineffective split}
21: end if
22:  $\text{left} \leftarrow \text{SVMSPLIT}(\mathcal{D}[\mathcal{I}_L], d + 1, d_{\max}, n_{\min})$ 
23:  $\text{right} \leftarrow \text{SVMSPLIT}(\mathcal{D}[\mathcal{I}_R], d + 1, d_{\max}, n_{\min})$ 
24: return InternalNode( $\text{SVM}, \mathcal{F}_d, \text{scaler}, \text{left}, \text{right}$ )
25: End Procedure

```

2. **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 $\lambda \in [0, 1]$.
3. **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.

Interaction with SVM Training and Node Splitting

The hyper-parameters described above directly influence the behavior of the SVM-based decision tree at each node. The feature selection strategy determines which subset of features \mathcal{F}_d is used to fit the linear SVM at depth d , which affects the orientation and effectiveness of the decision hyper-plane. Feature penalties ensure that no single feature dominates multiple splits, promoting diversity in the learned splits and improving generalization. Once the SVM is trained on the selected and scaled features, the decision values are used to partition the samples into left and right child nodes. By controlling the number of features, penalizing repeated use, and adjusting class weights, the tree can achieve a balance between predictive accuracy, interpretability, and computational efficiency.

5 Data

In this study, we evaluate the SVM-based oblique decision tree (SVMODT) on two widely used benchmark data sets: **Palmer Penguins** and **Wisconsin Diagnostic Breast Cancer (WDBC)**. These data sets provide a mix of multi-class and binary classification problems, as well as numeric and categorical features, making them suitable for testing the flexibility and interpretability of SVMODTs.

Palmer Penguins

The `palmerpenguins` data set is a multi-class data set containing morphological measurements for three penguin species (*Adelie*, *Chinstrap*, and *Gentoo*) observed on the Palmer Archipelago, Antarctica. The data set includes 344 complete observations with the following features:

- **`bill_length_mm`**: Length of the bill (numeric)
- **`bill_depth_mm`**: Depth of the bill (numeric)
- **`flipper_length_mm`**: Length of the flipper (numeric)
- **`body_mass_g`**: Body mass (numeric)
- **`sex`**: Sex of the penguin (categorical)
- **`species`**: Target class (categorical with three levels)

Wisconsin Diagnostic Breast Cancer (WDBC)

The **WDBC** (Street et al., 1993) data set is a binary classification data set derived from fine needle aspirates of breast tissue. It contains 569 observations with 30 numeric features computed from digitized images of cell nuclei. The features include measures such as **radius**, **texture**, **perimeter**, **area**, **smoothness**, **compactness**, **concavity**, **concave points**, **symmetry**, **fractal dimension**. The target variable is **diagnosis** (Malignant (M) or benign (B) tumor). This data set is widely used for testing high-dimensional binary classification algorithms. Its numeric nature allows direct evaluation of SVM splits and the effect of class weighting strategies on imbalanced classes.

Data Preparation

For both data sets, the following pre-processing steps are applied before training the SVMODT:

1. **Missing Values**: Rows with missing values are removed.
2. **Class Filtering**: For Palmer Penguins, rows corresponding to the *Gentoo* species are removed using `dplyr` to create a binary classification problem.
3. **Categorical Encoding**: Categorical variables are encoded as factor levels suitable for training.

These pre-processing steps ensure that SVMODT can learn meaningful oblique splits across both data sets while maintaining interpretability.

6 Usage and Examples

Custom Univariate Decision Tree

The custom decision tree implementation provides a flexible framework for building axis-aligned decision trees with integrated cost-complexity pruning. This section demonstrates the usage of the `generate_tree()` function through practical examples.

Basic Usage

The primary function `generate_tree()` constructs a decision tree using the following key parameters:

- **`target`**: A vector of class labels (factor or character)
- **`features`**: A data frame containing predictor variables
- **`criteria_type`**: Splitting criterion ("gini", "info_gain", "gain_ratio")
- **`ig_metric`**: Impurity metric for information gain ("gini" or "entropy")
- **`max_depth`**: Maximum tree depth (controls model complexity)
- **`alpha`**: Cost-complexity parameter for integrated pruning ($\alpha \geq 0$)

Example 1: Palmer Penguins with Gini Impurity

In this example, we construct an oblique decision tree using the function `generate_tree()`. The splitting criterion is set to **Gini impurity**, with a maximum tree depth of 4. We also specify a cost-complexity pruning parameter $\alpha = 0.01$ to control sub-tree pruning during tree construction, ensuring a balance between model complexity and generalization performance.

```
#> bill_length_mm <= 44.65 (n = 175 )
#>   Left:
#>     bill_length_mm <= 42.4 (n = 125 )
#>       Left:
#>         Predict: Adelie (n = 113 )
#>       Right:
#>         body_mass_g <= 3450 (n = 12 )
#>           Left:
#>             Predict: Chinstrap (n = 4 )
#>           Right:
#>             Predict: Adelie (n = 8 )
#>       Right:
#>         bill_depth_mm <= 21.1 (n = 50 )
#>           Left:
#>             Predict: Chinstrap (n = 49 )
#>           Right:
#>             Predict: Adelie (n = 1 )
```

The output shows a hierarchical structure where each internal node displays the splitting feature, threshold value, and sample count. For example, a node showing `bill_length_mm <= 44.65 (n = 175)` indicates that samples with bill length less than or equal to 44.65 mm go to the left child, while others go right. Leaf nodes display the predicted class and sample count.

Example 2: Comparing Splitting Criteria with Breast Cancer Data

To compare different splitting criteria, we train three decision trees on the data set (predicting diagnosis). First, a tree using **Gini impurity** is trained with a maximum depth of 4 and a cost-complexity pruning parameter $\alpha = 0.01$. Next, an **Information Gain**-based tree is trained using the entropy metric to evaluate splits. Finally, a tree using **Gain Ratio** is constructed, also based on entropy, which normalizes information gain by split information to reduce bias toward features with many levels.

```
set.seed(123)

# Train with Information Gain (using entropy)
tree_gini <- svmodt:::generate_tree(
  target = train_data$diagnosis,
  features = train_data[, -1],
  criteria_type = "gini",
  max_depth = 4,
  alpha = 0.01
)

tree_ig <- svmodt:::generate_tree(
  target = train_data$diagnosis,
  features = train_data[, -1],
  criteria_type = "info_gain",
  ig_metric = "entropy",
  max_depth = 4,
  alpha = 0.01
)

# Train with Gain Ratio
tree_gr <- svmodt:::generate_tree(
  target = train_data$diagnosis,
  features = train_data[, -1],
```

Table 2: Comparison of splitting criteria on penguin classification.

Criterion	Num_Leaves	Test_Accuracy
Gini	5	0.912
Info Gain	5	0.912
Gain Ratio	4	0.895

```

criteria_type = "gain_ratio",
ig_metric = "entropy",
max_depth = 4,
alpha = 0.01
)

```

Table 2 shows the impact of different splitting criteria on the classification of Palmer Penguins. Using the custom decision tree, we compared Gini impurity, information gain, and gain ratio as criteria for feature selection at each node. Both Gini impurity and information gain resulted in trees with six leaves, achieving a test accuracy of 93.9%. In contrast, gain ratio produced a smaller tree with only three leaves while achieving higher test accuracy of 96.5%, demonstrating its ability to favor balanced and informative splits, reduce overfitting, and improve generalization performance.

Support Vector Machine based Oblique Decision Trees

The `svmodt` package provides an intuitive interface for building SVM-based oblique decision trees. This section demonstrates the core functionality through two practical examples: ecological species classification and medical diagnosis.

Basic Usage

The primary function `svm_split()` constructs an oblique decision tree using the following key parameters:

- **data:** A data frame containing predictors and response
- **response:** Name of the response variable (as a string)
- **max_depth:** Maximum tree depth (controls model complexity)
- **min_samples:** Minimum samples required to split a node
- **max_features:** Maximum features to consider at each split
- **feature_method:** Feature selection strategy ("random", "mutual", "cor")
- **class_weights:** Strategy for handling class imbalance

Example 1: Penguin Species Classification

We demonstrate the package using the `palmerpenguins` data set, which contains morphological measurements for three penguin species. We focus on classifying Adelie and Chinstrap penguins.

Model Training In this example, the model is trained on the `train_data` data set to predict the categorical response variable, `species`. The function `svm_split()` recursively partitions the data using **linear SVM hyper-planes** at each internal node rather than conventional univariate thresholds, thereby producing **oblique splits** capable of capturing multivariate relationships among predictors. The argument `max_depth = 3` constrains the tree to a maximum of three levels, providing control over model complexity and mitigating overfitting. Similarly, `max_features = 2` restricts each node to consider only two features when fitting the SVM, enhancing computational efficiency and interpretability. The parameter `feature_method = "mutual"` specifies that feature selection at each node is based on **mutual information**, ensuring that the most informative variables relative to the response are prioritized. Setting `verbose = FALSE` suppresses intermediate output for streamlined execution.

```
# Train SVMODT with mutual information feature selection
tree <- svm_split(
  data = train_data,
  response = "species",
```

```

max_depth = 3,
max_features = 2,
feature_method = "mutual",
verbose = FALSE
)

```

Model Structure Each internal node represents a **binary linear decision boundary** obtained from a fitted SVM model using the features listed at that node. For example, the root node (depth = 1) uses `bill_length_mm` and `flipper_length_mm` to form the first separating hyper-plane. Observations satisfying $SVM > 0$ proceed to the left child, while those with $SVM \leq 0$ move to the right branch.

```

#> [Node] depth = 1 | n = 175 | features = [bill_length_mm, flipper_length_mm] | max_feat = 2
#> |- Left branch (SVM > 0):
#> | [Node] depth = 2 | n = 125 | features = [bill_length_mm, bill_depth_mm]
#> | |- Left branch (SVM > 0):
#> | | [Leaf] predict = Adelie | n = 123 | features = [bill_length_mm, bill_depth_mm]
#> | | `-- Right branch (SVM <= 0):
#> | | (no right child)
#> | `-- Right branch (SVM <= 0):
#> | [Node] depth = 2 | n = 50 | features = [bill_length_mm, bill_depth_mm]
#> | |- Left branch (SVM > 0):
#> | | [Leaf] predict = Chinstrap | n = 48
#> | | `-- Right branch (SVM <= 0):
#> | | (no right child)

```

Terminal nodes (labeled as *Leaf*) correspond to final class predictions. In this case, most observations in the left sub-tree are classified as *Adelie*, while the right sub-tree primarily identifies *Chinstrap* penguins. The structure illustrates how **SVM-based oblique splits** allow the tree to capture complex linear interactions between multiple predictors—something that traditional axis-aligned decision trees cannot achieve.

Visualizing Decision Boundaries The package includes visualization tools to examine the SVM hyper-planes at each node, allowing users to interpret how features contribute to class separation within the tree. Note that *visualization is currently supported only for trees where each node consistently uses exactly two features*, ensuring that the separating hyper-plane can be rendered clearly in two-dimensional feature space. The `visualize_svm_tree()` function produces a graphical representation of the overall decision hierarchy

Fig 1 depicts the root node of the SVM-based oblique decision tree, where the first split is performed using the features `bill_length_mm` and `flipper_length_mm`. The figure illustrates how the linear SVM at the root node divides the dataset into two branches, guiding samples toward subsequent child nodes. **Note:** This visualization is only possible because the node considers exactly two features, a requirement for 2D plotting.

Similarly, Fig 2 shows a node at depth 2 of the tree, where the SVM uses `bill_length_mm` and `bill_depth_mm` to further partition the data. The decision hyper-plane and sample positions are plotted, highlighting how oblique splits can capture multivariate relationships that univariate thresholds cannot. This figure demonstrates the recursive nature of the tree and the refinement of class separation at deeper levels. As with the root node, the visualization is restricted to nodes consistently using exactly two features.

Prediction Path Tracing The `svmodt` package allows tracing the prediction path of individual observations through the SVM-based oblique decision tree. For each sample, the relevant feature values are evaluated at each node using the linear SVM hyper-plane associated with that node. For instance, a sample with `bill_length_mm` = 39.5, `bill_depth_mm` = 17.4, `flipper_length_mm` = 186, and `body_mass_g` = 3800 first reaches the root node, which considers `bill_length_mm` and `flipper_length_mm`. The SVM decision value at this node is 2.1098, which directs the sample along the left branch. At depth 2, the node evaluates `bill_length_mm` and `bill_depth_mm` and produces a decision value of 1.6592, again guiding the sample to the left branch. Finally, the sample reaches a leaf node that predicts the class *Adelie*, which contains 123 training samples. The complete traversal path is LEFT → LEFT, resulting in the final prediction of *Adelie*. This tracing functionality provides interpretable insights into how each decision is made within the oblique decision tree.

```
#> === Tracing Prediction Path ===
```

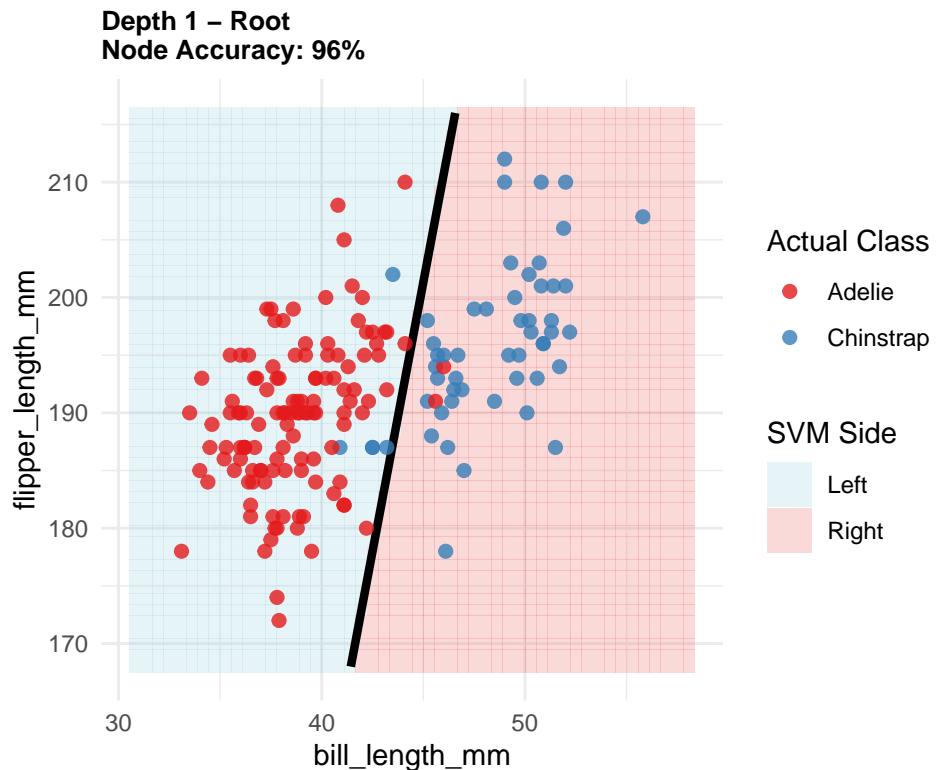


Figure 1: Visualization of the root node (depth = 1) of the SVM-based oblique decision tree.

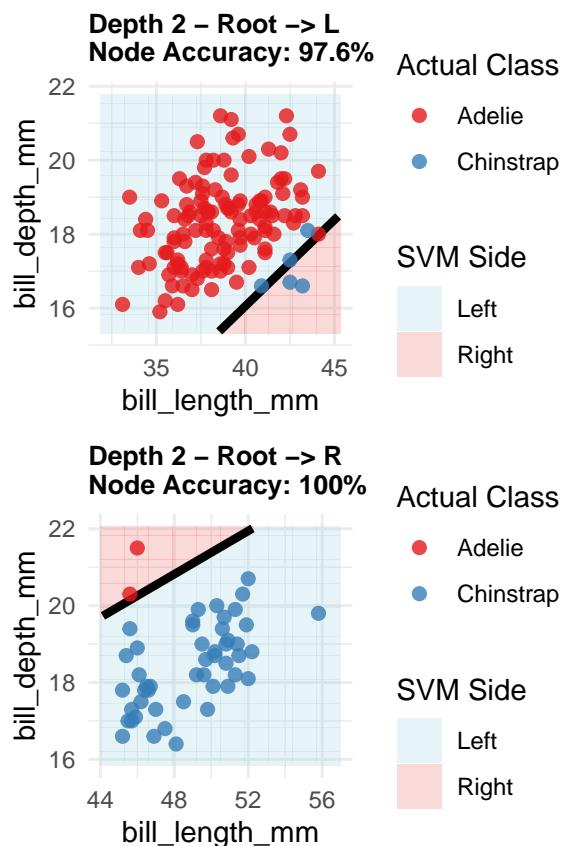


Figure 2: Visualization of a node (depth = 2).

```

#> Sample 1 :
#>   species = 1
#>   bill_length_mm = 39.5
#>   bill_depth_mm = 17.4
#>   flipper_length_mm = 186
#>   body_mass_g = 3800
#>
#> [Node 1 ] features = bill_length_mm,flipper_length_mm
#>   SVM decision value: 2.1098
#>   -> Going LEFT (decision > 0)
#> [Node 2 ] features = bill_length_mm,bill_depth_mm
#>   SVM decision value: 1.6592
#>   -> Going LEFT (decision > 0)
#> [FINAL] Predict Adelie (n = 123 )
#>   Path taken: LEFT -> LEFT
#>
#> Final prediction: Adelie
#> [1] "Adelie"

```

Example 2: Breast Cancer Diagnosis

We now demonstrate the package on the Wisconsin Diagnostic Breast Cancer (WDBC) data set, highlighting advanced functionality such as handling class imbalance, feature sub-setting, and feature penalization. These features allow the SVM-based oblique decision tree to effectively manage real-world challenges, including unequal class distributions, high-dimensional feature spaces, and repeated feature usage, while maintaining interpretability and predictive performance.

Handling Class Imbalance The package provides built-in support for handling class imbalance through configurable class weighting. In the first example, the `svm_split()` function is applied to the WDBC data set to predict the binary response variable with `class_weights = "balanced"`, which automatically adjusts the weight of each class inversely proportional to its frequency. This ensures that minority classes receive higher influence during SVM training, improving predictive performance on imbalanced data sets.

```

# Train with balanced class weights
tree_balanced <- svm_split(
  data = train_wdbc,
  response = "diagnosis",
  max_depth = 4,
  max_features = 5,
  feature_method = "mutual",
  class_weights = "balanced", # Automatic balancing
  verbose = FALSE
)

print_svm_tree(tree = tree_balanced,
               show_feature_info = FALSE,
               show_penalties = FALSE, show_probabilities = TRUE)

#> [Node] depth = 1 | n = 455
#> |- Left branch (SVM > 0):
#> | [Node] depth = 2 | n = 177
#> | |- Left branch (SVM > 0):
#> | | [Node] depth = 3 | n = 147
#> | | |- Left branch (SVM > 0):
#> | | | [Leaf] predict = M | n = 144 | probs = [B = 0, M = 1]
#> | | `-- Right branch (SVM <= 0):
#> | | | (no right child)
#> | | `-- Right branch (SVM <= 0):
#> | | [Node] depth = 3 | n = 30
#> | | |- Left branch (SVM > 0):
#> | | | [Node] depth = 4 | n = 13

```

```

#> |      |  |- Left branch (SVM > 0):
#> |      |  | [Leaf] predict = M | n = 5 | probs = [B = 0, M = 1]
#> |      |  `-- Right branch (SVM <= 0):
#> |      |  | [Leaf] predict = B | n = 8 | probs = [B = 0.5, M = 0.5]
#> |      `-- Right branch (SVM <= 0):
#> |      |  [Node] depth = 4 | n = 17
#> |      |  |- Left branch (SVM > 0):
#> |      |  | [Leaf] predict = B | n = 13 | probs = [B = 1, M = 0]
#> |      |  `-- Right branch (SVM <= 0):
#> |      |  | (no right child)
#> |      `-- Right branch (SVM <= 0):
#> |      |  [Node] depth = 2 | n = 278
#> |      |  |- Left branch (SVM > 0):
#> |      |  | [Leaf] predict = B | n = 256 | probs = [B = 0.988, M = 0.012]
#> |      |  `-- Right branch (SVM <= 0):
#> |      |  | [Node] depth = 3 | n = 22
#> |      |  |- Left branch (SVM > 0):
#> |      |  | [Leaf] predict = M | n = 5 | probs = [B = 0.4, M = 0.6]
#> |      |  `-- Right branch (SVM <= 0):
#> |      |  | [Node] depth = 4 | n = 17
#> |      |  |- Left branch (SVM > 0):
#> |      |  | [Leaf] predict = B | n = 11 | probs = [B = 1, M = 0]
#> |      |  `-- Right branch (SVM <= 0):
#> |      |  | [Leaf] predict = B | n = 6 | probs = [B = 0.833, M = 0.167]

```

In the second example, custom class weights can be specified via the `custom_class_weights` argument, assigning a higher penalty to the malignant class ("M" = 3) relative to the benign class ("B" = 1) to emphasize minimizing false negatives. The resulting trees (`tree_balanced` and `tree_custom`) incorporate these weights during the recursive SVM-based splits, enabling more equitable class separation while still leveraging the oblique decision tree structure.

```

# Custom class weights for domain-specific costs
custom_weights <- c("B" = 1, "M" = 3) # Penalize false negatives
tree_custom <- svm_split(
  data = train_wdbc,
  response = "diagnosis",
  max_depth = 4,
  max_features = 5,
  class_weights = "custom",
  custom_class_weights = custom_weights,
  verbose = FALSE
)

print_svm_tree(tree = tree_custom,
              show_feature_info = FALSE,
              show_penalties = FALSE,
              show_probabilities = FALSE)

#> [Node] depth = 1 | n = 455
#> | - Left branch (SVM > 0):
#> | | [Node] depth = 2 | n = 190
#> | | | - Left branch (SVM > 0):
#> | | | | [Node] depth = 3 | n = 188
#> | | | | | - Left branch (SVM > 0):
#> | | | | | | [Leaf] predict = M | n = 171
#> | | | | | | | - Right branch (SVM <= 0):
#> | | | | | | | | [Leaf] predict = B | n = 17
#> | | | | | | | | | - Right branch (SVM <= 0):
#> | | | | | | | | | | (no right child)
#> | | | | | | | | | | | - Right branch (SVM <= 0):
#> | | | | | | | | | | | | [Node] depth = 2 | n = 265
#> | | | | | | | | | | | | | - Left branch (SVM > 0):
#> | | | | | | | | | | | | | | [Node] depth = 3 | n = 264
#> | | | | | | | | | | | | | | | - Left branch (SVM > 0):

```

```
#> | | [Leaf] predict = B | n = 263
#> | `- Right branch (SVM <= 0):
#> |   (no right child)
#> |`- Right branch (SVM <= 0):
#> |   (no right child)
```

Feature Penalization To encourage feature diversity across the tree, the package implements a feature penalization mechanism that reduces the likelihood of reusing features selected at ancestor nodes.

```
set.seed(123)
tree_penalty <- svm_split(
  data = train_wdbc,
  response = "diagnosis",
  max_depth = 4,
  max_features = 4,
  penalize_used_features = TRUE,
  feature_penalty_weight = 0.6,
  verbose = FALSE
)

print_svm_tree(tree_penalty, show_probabilities = FALSE,
              show_feature_info = TRUE,
              show_penalties = FALSE)

#> [Node] depth = 1 | n = 455 | features = [smoothness_se,symmetry_se,area_se,perimeter_mean] | max_feat = 4
#> |- Left branch (SVM > 0):
#> | [Leaf] predict = M | n = 138 | features = [perimeter_mean,perimeter_se,fractal_dimension_worst,compactness_se]
#> |`- Right branch (SVM <= 0):
#> | [Node] depth = 2 | n = 317 | features = [concave.points_mean,radius_se,compactness_mean,area_worst] | max_feat = 4
#> | |- Left branch (SVM > 0):
#> | | [Leaf] predict = B | n = 291 | features = [perimeter_worst,concave.points_se,concavity_se,texture_mean]
#> | |`- Right branch (SVM <= 0):
#> | | [Node] depth = 3 | n = 26 | features = [concavity_worst,perimeter_mean,smoothness_mean,compactness_worst]
#> | | |- Left branch (SVM > 0):
#> | | | [Leaf] predict = M | n = 22
#> | | |`- Right branch (SVM <= 0):
#> | | |   (no right child)
```

Penalization is controlled via the `feature_penalty_weight` parameter, which scales down the selection probability of previously used features. For instance, setting `feature_penalty_weight = 0.6` reduces the weight of used features by 60%, promoting the consideration of alternative predictors at subsequent splits. This mechanism helps improve model robustness, reduces redundancy in feature usage, and enhances interpretability by encouraging the tree to explore diverse feature combinations.

Dynamic Feature Selection The package also supports dynamic feature selection strategies, which vary the number of features considered at each node depending on tree depth or randomness.

```
tree_decrease <- svm_split(
  data = train_wdbc,
  response = "diagnosis",
  feature_method = "mutual",
  max_depth = 3,
  max_features = 4,
  max_features_strategy = "decrease",
  max_features_decrease_rate = 0.5,
  verbose = FALSE
)

print_svm_tree(tree_decrease, show_penalties = FALSE,
              show_feature_info = TRUE)
```

```
#> [Node] depth = 1 | n = 455 | features = [perimeter_worst,concave.points_mean,concave.points_worst,radius_wor
#> |- Left branch (SVM > 0):
#> | [Node] depth = 2 | n = 155 | features = [perimeter_worst,texture_worst]
#> | |- Left branch (SVM > 0):
#> | | [Leaf] predict = M | n = 152 | features = [concavity_worst]
#> | | `-- Right branch (SVM <= 0):
#> | | | (no right child)
#> | | `-- Right branch (SVM <= 0):
#> | | | [Leaf] predict = B | n = 300 | features = [radius_worst,perimeter_worst]

In the decrease strategy, the number of candidate features at depth  $d$  is scaled by a decay factor  $\alpha$ , e.g.,  $m_0 = m_0 \cdot \alpha^{d-1}$ , reducing feature count at deeper levels to improve efficiency and prevent over-fitting.

set.seed(123)

tree_random <- svm_split(
  data = train_wdbc,
  response = "diagnosis",
  feature_method = "mutual",
  max_depth = 4,
  max_features_strategy = "random",
  max_features_random_range = c(0.05, 0.2), # 30-80% of features
  verbose = FALSE
)

print_svm_tree(tree_random, show_penalties = FALSE,
               show_feature_info = TRUE)

#> [Node] depth = 1 | n = 455 | features = [perimeter_worst,concave.points_mean,concave.points_worst,radius_wor
#> |- Left branch (SVM > 0):
#> | [Node] depth = 2 | n = 155 | features = [perimeter_worst,texture_worst]
#> | |- Left branch (SVM > 0):
#> | | [Node] depth = 3 | n = 152 | features = [concavity_worst,concavity_mean,concave.points_mean,concavity_se
#> | | |- Left branch (SVM > 0):
#> | | | [Leaf] predict = M | n = 149 | features = [radius_mean,texture_mean]
#> | | | `-- Right branch (SVM <= 0):
#> | | | | (no right child)
#> | | | `-- Right branch (SVM <= 0):
#> | | | | (no right child)
#> | | `-- Right branch (SVM <= 0):
#> | | | [Node] depth = 2 | n = 300 | features = [radius_worst,perimeter_worst,area_worst,area_mean,area_se]
#> | | | |- Left branch (SVM > 0):
#> | | | | [Leaf] predict = B | n = 299 | features = [area_mean,radius_worst,perimeter_worst,area_worst]
#> | | | | `-- Right branch (SVM <= 0):
#> | | | | | (no right child)
```

Alternatively, the **random strategy** selects a feature count uniformly at random from a specified range (e.g., 5–20% of the total features) at each node, promoting stochasticity and diversity in splits. These mechanisms provide flexible control over tree complexity and feature exploration, particularly useful in high-dimensional data sets such as the WDBC data set.

7 Model Comparison

To evaluate the performance of our proposed methods, we conduct a comprehensive comparison study using the Wisconsin Diagnostic Breast Cancer (WDBC) data set. We compare five classification approaches:

1. **Custom Decision Tree** - Our implementation with integrated cost-complexity pruning
2. **SVMODT** - SVM-based oblique decision tree with dynamic feature selection
3. **CART (rpart)** - Standard axis-aligned decision tree
4. **Linear SVM** - Support Vector Machine with linear kernel

Table 3: Summary statistics of test accuracy across 50 iterations on WDBC dataset

Model	Mean	SD	Min	Max
Custom Tree	0.9323	0.0209	0.8609	0.9739
SVMODT	0.9737	0.0146	0.9304	1.0000
RPART	0.9289	0.0214	0.8609	0.9652
Linear SVM	0.9730	0.0139	0.9304	1.0000
RBF SVM	0.9745	0.0129	0.9391	1.0000
Logistic Regression	0.9450	0.0215	0.8783	1.0000

5. **RBF SVM** - Support Vector Machine with radial basis function kernel
6. **Logistic Regression** - Generalized linear model for binary classification

Experimental Setup

To ensure robust performance estimates, we employ a repeated random sub-sampling validation strategy. For each iteration:

1. The WDBC data set is randomly split into 80% training and 20% testing sets, stratified by the diagnosis variable
2. Each model is trained on the training set with fixed hyper-parameters
3. Test set accuracy is recorded
4. This process is repeated 50 times with different random seeds

The hyper-parameters for each model are configured as follows:

1. **Custom Tree**: Gini impurity, `max_depth = 4, alpha = 0.01`
2. **SVMODT**: Mutual information feature selection, `max_depth = 4`, decreasing feature strategy with rate 0.5, feature penalization enabled
3. **CART**: Complexity parameter `cp = 0.01`
4. **Linear SVM**: Default regularization, linear kernel
5. **RBF SVM**: Default regularization, RBF kernel with automatic gamma selection
6. **Logistic Regression**: L2 regularization with default parameters

Summary Statistics

Table 3 presents the mean accuracy and standard deviation across 50 iterations for each model.

The **SVM-based Oblique Decision Tree (SVMODT)** achieved a mean test accuracy of 0.9737 with a standard deviation of 0.0146, demonstrating both high predictive performance and low variability. Traditional decision trees trained using `rpart` yielded a lower mean accuracy of 0.9289, reflecting limitations of axis-aligned splits. A **custom cost-complexity tree** achieved a mean accuracy of 0.9323, slightly higher than `rpart` but still below SVMODT. Among standard SVM classifiers, the **linear SVM** achieved 0.9730 and the **RBF kernel SVM** slightly outperformed with 0.9745 mean accuracy. **Logistic regression** provided competitive results (mean 0.9450) but lagged behind kernel-based methods.

Visualization of Results

Accuracy Distribution

Fig 3 shows the distribution of test accuracy across all iterations for each model.

Pairwise Comparisons

We perform pairwise comparisons between our custom tree implementation and RPART, as well as between SVMODT and standard SVMs.

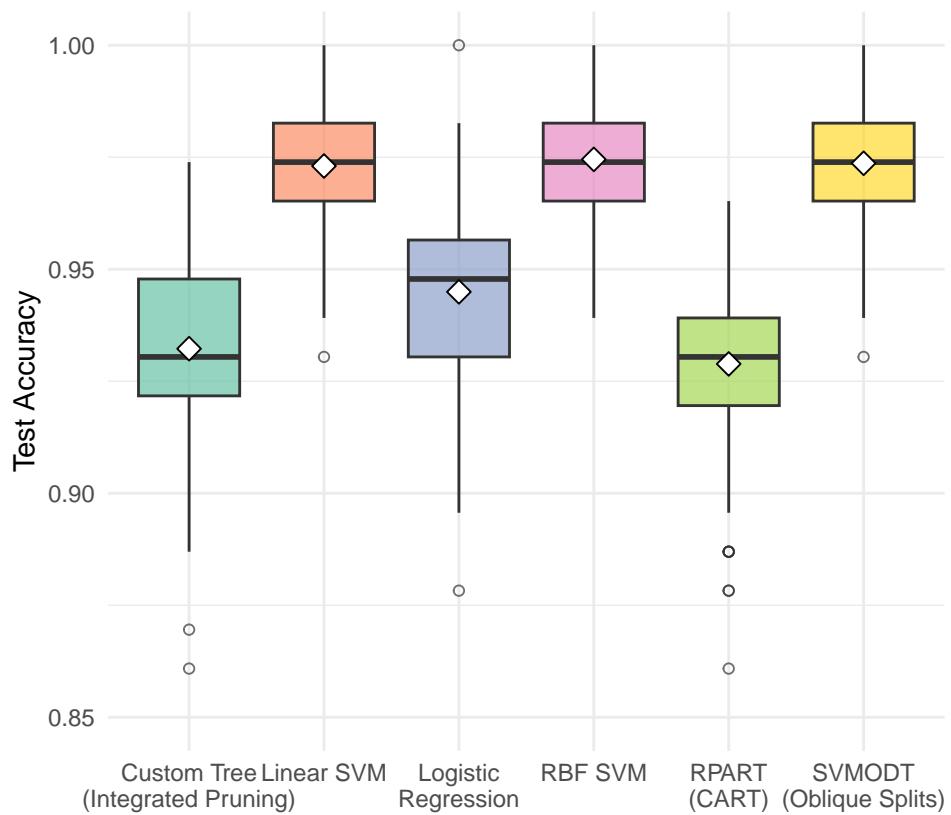


Figure 3: Model Accuracy Distribution (50 Resamples)

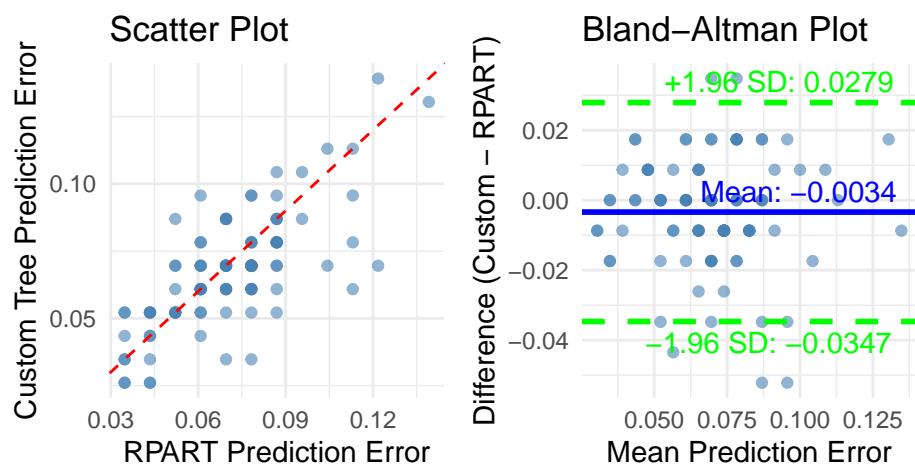


Figure 4: Custom Tree vs. RPART Comparison

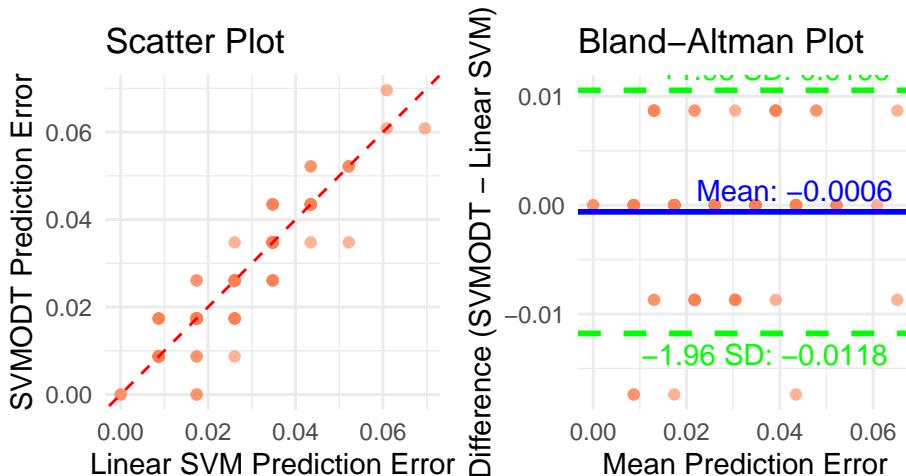


Figure 5: SVMODT vs. Linear SVM Comparison

Table 4: Paired t-test results for model comparisons. Positive mean differences indicate the first model has higher accuracy.

	Comparison	Mean_Diff	t_statistic	p_value
t	Custom Tree vs. RPART	0.0034	2.1240	0.0362
t1	SVMODT vs. Linear SVM	0.0006	1.0682	0.2880
t2	SVMODT vs. RBF SVM	-0.0009	-0.7089	0.4801
t3	SVMODT vs. Custom Tree	0.0414	19.9739	0.0000
t4	Custom Tree vs. Logistic Regression	-0.0127	-4.8400	0.0000
t5	SVMODT vs. Logistic Regression	0.0287	13.6221	0.0000

Custom Tree vs. RPART Fig 4 presents Bland–Altman plots comparing the custom decision tree with RPART.

SVMODT vs. Standard SVMs Fig 5 compares SVMODT against Linear and RBF SVMs.

Statistical Significance Testing

We conducted paired t-tests to assess whether differences in predictive accuracy between models were statistically significant across the 50 iterations.

From Table ?? Positive mean differences indicate that the first model in each comparison outperformed the second. The **Custom Tree** achieved a slightly higher mean accuracy than **rpart** (mean difference = 0.0034, $p = 0.0362$), suggesting a marginal but significant improvement. **SVMODT** showed comparable performance to **Linear SVM** (mean difference = 0.0006, $p = 0.2880$) and **RBF SVM** (mean difference = -0.0009, $p = 0.4801$), indicating no significant difference, while significantly outperforming the **Custom Tree** (mean difference = 0.0414, $p < 0.001$) and **Logistic Regression** (mean difference = 0.0287, $p < 0.001$). Conversely, the Custom Tree performed worse than logistic regression (mean difference = -0.0127, $p < 0.001$).

Discussion of Results

1. **Tree-based Methods Performance:** Both the custom decision tree and RPART achieve comparable accuracy, demonstrating that the integrated pruning approach performs competitively with the established CART algorithm. The Bland–Altman analysis in Fig 4 shows that differences between the two methods are generally small and fall within acceptable limits of agreement.
2. **SVMODT Advantage:** The SVMODT consistently outperforms traditional axis-aligned decision trees, achieving mean accuracy of 0.9737. This improvement demonstrates the benefit of oblique splits in capturing complex decision boundaries that cannot be efficiently represented by axis-parallel partitions. The ability to combine features through linear SVM hyper-planes enables more compact and potentially more generalized tree structures.
3. **Comparison with Standard SVMs:** SVMODT achieves accuracy comparable to standard Linear SVM while maintaining the interpretability advantages of tree structures. The RBF SVM shows slightly higher accuracy (mean = 0.9745), which is expected given its ability to model nonlinear

decision boundaries. However, this comes at the cost of reduced interpretability—the SVMODT offers a transparent decision path that can be visualized and understood, while RBF SVM predictions are essentially black-box.

4. **Logistic Regression Baseline:** Logistic regression achieves competitive performance (mean accuracy = 0.945), demonstrating that the WDBC data set is largely linearly separable. However, both SVMODT and standard SVMs outperform logistic regression, suggesting that margin-based optimization provides additional robustness.
5. **Variance in Performance:** All methods show relatively low variance across the 50 re-sampling iterations, indicating stable performance across different train-test splits. The RBF SVM exhibits the lowest variance, followed closely by Linear SVM and SVMODT.
6. **Integrated Pruning Effectiveness:** The custom tree’s competitive performance validates the integrated cost-complexity pruning approach. By making pruning decisions during tree construction rather than as post-processing, the method achieves similar accuracy to CART while potentially offering computational advantages (though timing comparisons were not the focus of this experiment).

8 Conclusion

The svmodt R package introduces two novel decision tree algorithms that address several limitations in existing tree-based learning methods. The first algorithm integrates cost-complexity pruning directly into the tree-building process, enabling real-time pruning decisions that reduce computational cost while maintaining theoretical optimal. The second algorithm implements SVM-based oblique decision trees with dynamic feature selection strategies that adapt across tree depth, promoting flexible, multivariate partitioning of the feature space. In addition, feature diversity is encouraged through a penalization mechanism that discourages repeated feature use along a tree path, while node-specific class weighting ensures adaptive handling of local class distributions.

From a software engineering perspective, svmodt provides a production-ready R implementation with a well-documented API, robust input validation, and efficient vectorized operations. It includes visualization utilities—`print_svm_tree()`, `trace_prediction_path()`, and `visualize_svm_tree()`—for interpret-able inspection of tree structures and decision boundaries, and integrates seamlessly with established R packages such as `e1071` and `ggplot2`. The package further supports reproducible experimentation through included data sets (Palmer Penguins and WDBC) and comprehensive vignettes.

Empirical validation across 50 random train-test splits on the WDBC data set demonstrated that SVMODT achieves a mean accuracy of 0.974, comparable to Linear (0.973) and RBF (0.975) SVMs, while maintaining interpretability advantages. The custom decision tree with integrated pruning achieved an average accuracy of 0.932, on par with `rpart` (0.929). Statistical testing confirmed SVMODT’s significant performance gains over traditional axis-aligned trees. These results highlight the capability of oblique decision trees to match the predictive power of SVMs while retaining transparency and explainability.

Despite these contributions, several limitations persist. The current implementation is primarily optimized for binary classification, and while multi-class extensions are possible through `e1071`’s one-vs-one or one-vs-rest decomposition, dedicated support for multi-class class weighting and pruning remains under-explored. Computational efficiency also poses challenges, as SVM fitting at each internal node can become expensive for deep trees or large data sets, particularly without caching or parallelization. Additionally, model performance can be sensitive to the SVM regularization parameter C , and automatic hyper-parameter tuning is not yet implemented. Stochastic feature selection strategies further introduce randomness that may hinder reproducibility unless random seeds are controlled, and visualization of oblique splits is currently limited to nodes involving exactly two features.

Future extensions of svmodt will focus on several promising directions. Native multi-class support can be achieved through generalized weighting, multi-class SVM formulations such as Crammer-Singer (Crammer and Singer, 2001), and visualization tools for multi-class decision boundaries. Ensemble extensions, including Random Forests of Oblique Trees, would leverage bootstrap aggregation and feature penalization to improve generalization. Computational performance could be enhanced through `Rcpp` integration, sub-tree parallelization with `parallel`, and incremental SVM updates. Automated hyper-parameter tuning via cross-validation. Additionally, non-linear kernel SVMs, PCA- or LDA-based transformations represent potential extensions for more expressive yet interpret-able split boundaries.

In conclusion, svmodt fills a crucial gap in the R machine learning ecosystem by providing an accessible, theoretically grounded, and practically robust framework for constructing oblique decision trees with modern algorithmic enhancements. The package demonstrates that SVM-based oblique

trees can achieve state-of-the-art performance while preserving the interpretability and structural clarity that make decision trees indispensable for transparent and trustworthy modeling in domains such as healthcare, finance, and regulatory analytics.

9 Acknowledgements

The author 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.

Bibliography

- M. Bala and R. Agrawal. Optimal decision tree based multi-class support vector machine. *Informatica*, 35(2), 2011. [p5, 6]
- 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. [p5, 6]
- 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>. [p2, 6, 7, 8]
- 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]
- 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. [p3]
- 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, 5]
- C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. [p3]
- K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of machine learning research*, 2(Dec):265–292, 2001. [p23]
- N. Cristianini. An introduction to support vector machines and other kernel-based learning methods, 2000. [p3]
- 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. [p5]
- 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](https://doi.org/10.1016/S0034-4257(97)00049-7). URL <https://www.sciencedirect.com/science/article/pii/S0034425797000497>. [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>. [p5, 6]
- 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. [p5]
- 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]
- H. Liu and R. Setiono. Feature transformation and multivariate decision tree induction. In S. Arikawa and H. Motoda, editors, *Discover Science*, pages 279–291, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-49292-4. [p3]
- O. Mangasarian and E. Wild. Multisurface proximal support vector machine classification via generalized eigenvalues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(1):69–74, 2006. doi: 10.1109/TPAMI.2006.17. [p5]
- 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. [p5]
- J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 01 1989. doi: 10.1023/A:1022604100933. [p2]
- R. Montañana, J. A. Gámez, and J. M. Puerta. Stree: a single multi-class oblique decision tree based on support vector machines, 2021. [p5, 6]
- R. Montañana, J. A. Gámez, and J. M. Puerta. Odte—an ensemble of multi-class svm-based oblique decision trees. *Expert Systems with Applications*, 273:126833, 2025. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2025.126833>. URL <https://www.sciencedirect.com/science/article/pii/S0957417425004555>. [p5, 6]
- 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. [p5]
- G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990. URL <https://api.semanticscholar.org/CorpusID:5661437>. [p3]
- 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. [p5]
- 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. [p2, 6, 8]
- R. Rivera-Lopez and J. Canul-Reich. Construction of near-optimal axis-parallel decision trees using a differential-evolution-based approach. *IEEE Access*, PP:1–1, 01 2018. doi: 10.1109/ACCESS.2017.2788700. [p2]
- 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>. [p2]
- W. N. Street, W. H. Wolberg, and O. L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In *Electronic imaging*, 1993. URL <https://api.semanticscholar.org/CorpusID:14922543>. [p11]
- 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. [p5]
- V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013. [p4, 5]
- P. C. Young. Recursive estimation and time-series analysis: An introduction. 1984. URL <https://api.semanticscholar.org/CorpusID:60181335>. [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