Fast SVM on Tree

Cong-Huan Tran[1], Thu-Le Tran* [2],
Ngan Nguyen[2], Khanh-Duy Le[2], and Clément Elvira[3]

[1]Faculty of xxx, College of Information and Communication Technology, Cantho University, Can Tho City, Vietnam
[2]Faculty of Mathematics and Informatics, Teacher College,, Cantho University, Can Tho City, Vietnam
[3]France

14/07/2025

Abstract

SVM is important. Solving SVM is time consuming. We approximate the points with a tree. We then propose a new fast computing method for SVM on tree.

In our experiments, we show that: 1) Our SVM model can be solved faster than the classical SVM, 2) Our SVM model achieve the same (or better) accuracy compared with classical SVM.

Contents

*Corresponding author, ttle@ctu.edu.vn

1 Notes

[LE: this section will be removed when we finish the project]

Notations. We will use the following notations for SVM model on $\mathbb{R}^n$

1. Training data $(x_i, y_i) \in \mathbb{R}^n \times \{-1, 1\}$ for $i = 1, ..., m$
2. Problem fomulation
$$\min \quad \sum_{i=1}^{m} \max(1 - y_i x_i^T w, 0) + \frac{\lambda}{2}\|w\|_2^2$$

We will use the following notations for SVM model on tree $\mathcal{T}$

1. Tree $\mathcal{T} = (V, E)$ be a tree with vertex set $V = \{x_1, ..., x_m\} \subset \mathbb{R}^n$ and edge set E, an edge e connecting x_i and x_j is denoted by $e = (x_i, x_j)$.
2. Training data $(x_i, y_i) \in \mathbb{R}^n \times \{-1, 1\}$ for $i = 1, ..., m$
3. Denote $V_+ = \{x_i \in V : y_i = +1\}, V_- = \{x_i \in V : y_i = -1\}$
4. The SVM on tree aims to minimize the objective function
$$\min_{u \in V_+, v \in V_-} \quad P(u, v) = f(u, v) - \lambda d(u, v)$$
where $d(u, v)$ stands for the margin of the model and f is the loss function defined as $f(u, v) = f_+(u, v) + f_-(u, v)$ with
$$f_+(u, v) = \sum_{x \in V_+(u,v)} d(x, u) \text{ and } f_-(u, v) = \sum_{x \in V_-(v,u)} d(x, v).$$
Here, $V_+(u, v)$ is the set of $+1$-labelled vertices of subtree rooted at u containing v and $V_-(v, u)$ is the set of -1-labelled vertices of subtree rooted at v containing u.
5. [LE: How to define separating plan?]

2 Introduction

Support Vector Machines (SVMs) have established themselves as one of the most powerful and theoretically well-founded approaches to machine learning since their introduction by Vapnik and colleagues in the 1990s. The fundamental principle of SVMs lies in finding an optimal separating hyperplane that maximizes the margin between different classes, providing both strong theoretical guarantees and excellent empirical performance across a wide range of applications.

Traditional SVMs excel in scenarios where data can be effectively separated by linear or kernel-induced nonlinear boundaries in high-dimensional feature spaces. However, many real-world datasets exhibit complex geometric structures that are not naturally captured by conventional distance metrics or kernel functions. In particular, when data points are embedded in or naturally associated with tree-like or hierarchical structures, standard SVM approaches may fail to exploit the inherent geometric relationships that could improve classification performance.

Tree structures arise naturally in numerous domains: phylogenetic trees in bioinformatics represent evolutionary relationships between species, social networks often exhibit hierarchical community structures, and many machine learning problems involve data with natural tree-like dependencies. In such contexts, the Euclidean distance used by traditional SVMs may not reflect the true similarity or dissimilarity between data points, as it ignores the underlying structural constraints.

This paper introduces a novel extension of Support Vector Machines that operates directly on tree structures, which we call **SVM on Tree**. Our approach constructs an augmented tree representation from the original data and formulates the classification problem as an optimization over support pairs within this tree structure. The key innovation lies in developing a tree-based distance metric that respects the geometric constraints of the tree while maintaining the theoretical foundations of SVM optimization.

Our main contributions are:

1. A novel tree-based SVM formulation that constructs an augmented tree from input data and optimizes support vector selection within this structure.
2. A theoretical analysis establishing the adjacency property, which dramatically reduces the computational complexity from $O(n^2)$ to $O(n)$ support pair candidates.
3. An efficient algorithm with $O(nd+n\log n)$ time complexity that leverages dynamic programming techniques for objective function evaluation.
4. Empirical validation demonstrating the effectiveness of our approach on datasets with inherent tree-like structures.

The remainder of this paper is organized as follows: Section 3 reviews the foundations of classical SVM theory, Section 5 presents our SVM on Tree methodology, Section 6 provides experimental validation, and Section **??** concludes with directions for future work.

3 Classical SVM

In this section, we review the fundamental concepts of Support Vector Machines that serve as the theoretical foundation for our tree-based extension.

3.1 Hard margin SVM

Given a labeled training dataset $\{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathbb{R}^d$ are feature vectors and $y_i \in \{-1, +1\}$ are binary class labels, the goal of SVM is to find a hyperplane, say $w^T x + b = 0$ for $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$, that optimally separates the two classes, in the sense that

$$\begin{aligned} w^T x + b &\geq 1, \quad \forall y_i = +1 \\ w^T x + b &\leq -1, \quad \forall y_i = -1 \end{aligned}$$

The margin is defined as the distance between the two hyperplanes $w^T x + b = 1$ and $w^T x + b = -1$, which is $\frac{2}{\|w\|_2}$. SVM model seeks the separating hyperplane with maximal margin. This leads to the following optimization problem:

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

$$\text{s.t.} \quad y_i(w^T x_i + b) \geq 1, \quad i = 1, \ldots, n \tag{2}$$

The constraints ensure that all training points are correctly classified with at least unit distance from the decision boundary.

The point (or equivalently considered as a vector) $s \in V_+$ and $p \in V_+$ are said to be positive and negative support vector, respectively, if they are on the supporting hyperplane, i.e.,

$$\begin{aligned} w^s + b &= 1 \\ w^p + b &= -1. \end{aligned}$$

Hence the name SVM.

3.2 Soft Margin SVM

To handle non-separable cases, the soft margin SVM introduces slack variables $\xi_i \geq 0$:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i && (3) \\ \text{s.t.} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, n && (4) \\ & \xi_i \geq 0, \quad i = 1, \ldots, n && (5) \end{aligned}$$

where $C > 0$ is the regularization parameter that balances between margin maximization and training error minimization.

Remark 3.1 (Kernel method)**.** *To handle nonlinear classification, SVM can be extended using kernel functions $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ that implicitly map data to higher-dimensional feature spaces. Common kernels include:*

- ***Linear kernel**: $K(x_i, x_j) = x_i^T x_j$*
- ***RBF kernel**: $K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2)$*
- ***Polynomial kernel**: $K(x_i, x_j) = (x_i^T x_j + c)^p$*

4 New interpretation of SVM based on support vectors

In this section, we rewrite the soft-margin SVM model in term of support vectors. This will lay the foundation for our SVM model on tree in the next section.

Define

$$\sigma(x_i) = w^T x_i + b$$

and

$$d(x_i, x_j) = \sigma(x_i) - \sigma(x_j)$$

Let $s \in V_+$ and $p \in V_-$ be two support vectors, then the margin is given by the distance between s and p,

$$\frac{2}{\|w\|_2} = \frac{(w^T s + b) - (w^T p + b)}{\|w\|_2} = \frac{d(s, p)}{\|w\|_2}.$$

In the soft margin SVM, the noise can also be written in terms of s and p as follows. For $x_i \in V_+$, we should have

$$\begin{aligned} \xi_i &= 1 - (w^T x_i + b), \forall x_i \in V_+ \\ &= \sigma(s) - \sigma(x_i) \\ &= d(s, x_i) \end{aligned}$$

5 SVM on Tree

In this section, we present a novel approach to binary classification using Support Vector Machines (SVM) on tree structures. Our method constructs an augmented tree from the data and formulates the classification problem as an optimization over support pairs on this tree.

5.1 Problem Formulation

5.1.1 Data Representation and Spine Construction

Given a labeled dataset $\{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \{-1, +1\}$, we define the class means as:

$$\mu_- = \frac{1}{|V_-|} \sum_{y_i=-1} x_i, \tag{6}$$

$$\mu_+ = \frac{1}{|V_+|} \sum_{y_i=+1} x_i, \tag{7}$$

where $V_- = \{i : y_i = -1\}$ and $V_+ = \{i : y_i = +1\}$.

The unit spine direction is computed as:

$$w = \frac{\mu_+ - \mu_-}{\|\mu_+ - \mu_-\|}.$$

For each data point x_i, we compute its projection onto the line passing through μ_- in direction w:

$$x_i' = \mu_- + \langle x_i - \mu_-,\, w\rangle\, w, \tag{8}$$

$$t_i = \langle x_i - \mu_-,\, w\rangle. \tag{9}$$

The projected points $\{x_i'\}$ are then sorted by their projection coordinates t_i.

5.1.2 Augmented Tree Construction

We construct an augmented tree $T = (V, E)$ with two types of edges:

1. **Spoke edges**: Connect each original point x_i to its projection x_i', weighted by the Euclidean distance $\|x_i - x_i'\|$.
2. **Spine edges**: Connect consecutive projections in the sorted order, weighted by their Euclidean distance in $\mathbb{R}^d$.

For any vertices $u, v \in V$, let $d(u, v)$ denote the shortest path distance (sum of edge weights) between u and v in T.

5.1.3 Objective Function

For two vertices u, v with $y(u) \neq y(v)$, we define the noise terms:

$$f_+(u, v) = \sum_{z \in V_+(u,v)} d(z, u), \tag{10}$$

$$f_-(v, u) = \sum_{z \in V_-(v,u)} d(z, v), \tag{11}$$

where $V_+(u, v)$ (respectively $V_-(v, u)$) is the set of positive (respectively negative) labeled vertices in the subtree on the branch from u toward v (respectively from v toward u) when the tree is rooted at u (respectively at v).

The total noise is

$$f(u,v) = f_+(u,v) + f_-(v,u) \tag{12}$$

[LE: We define the margin as the distance between two support vectors

$$d(u,v).$$

Since we aim to minimize the noise while maximizing the margin, the total loss function is defined as

$$L_\lambda(u,v) = f(u,v) - \lambda \cdot d(u,v).$$

where $\lambda \geq 0$ is a parameter which controls the balance between the noise and the margin.]

In this work, we focus on the case $\lambda = 1$ and write $L(u,v) = f(u,v) - d(u,v)$. The decision boundary corresponding to a support pair (s,p) is the perpendicular bisector hyperplane of the segment sp in $\mathbb{R}^d$.

5.2 Theoretical Properties

In this subsection, we establish the key theoretical properties of our SVM on tree formulation, focusing on the case $\lambda = 1$.

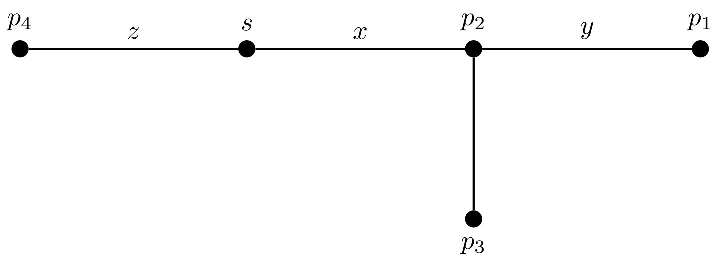

Figure 1: Tree structure illustrating nodes s, p_1, p_2, p_3, and p_4 with edge weights x, y, and z.

5.2.1 Adjacency Property

We now establish a fundamental property that significantly reduces the search space for optimal support pairs.

Let $s \in V_+$ and $p_1, p_2 \in V_-$ and p_2 is in the path connecting s and p_1, i.e.,

$$d(s,p_1) = d(s,p_2) + d(p_2,p_1) \tag{13}$$

Define the residual of vertex set

$$R = V_-(p_1,s) \setminus V_-(p_2,s).$$

The following result establish the difference between the two noise functions.

Lemma 5.1. *We have*

$$f(s,p_1) - f(s,p_2) = d(p_2,p_1)|V_-(p_2,s)| + \sum_{v \in R} d(v,p_1).$$

Proof. Recall the definition of total noise, see XXX, we have

$$f(s,p_1) = f_+(s,p_1) + f_-(p_1,s)$$
$$f(s,p_2) = f_+(s,p_2) + f_-(p_2,s).$$

Since $f_+(s,p_1) = f_+(s,p_2)$, [LE: Why? Explain.], then

$$f(s,p_1) - f(s,p_2) = f_-(p_1,s) - f_-(p_2,s).$$

Recall the definition of negative noise, we have

$$f_-(p_1,s) = \sum_{v \in V_-(p_1,s)} d(v,p_1)$$

Then

$$\begin{aligned} f_-(p_1,s) &= \sum_{v \in V_-(p_1,s)} d(v,p_1) \\ &= \sum_{v \in V_-(p_2,s)} d(v,p_1) + \sum_{v \in R} d(v,p_1) \\ &= \sum_{v \in V_-(p_2,s)} [d(v,p_2) + d(p_2,p_1)] + \sum_{v \in R} d(v,p_1) \\ &= f_-(p_2,s) + d(p_2,p_1)|V_-(p_2,s)| + \sum_{v \in R} d(v,p_1) \end{aligned}$$

or

$$f_-(p_1,s) - f_-(p_2,s) = d(p_2,p_1)|V_-(p_2,s)| + \sum_{v \in R} d(v,p_1).$$

□

Theorem 5.2 (Adjacency Property)**.** *For $\lambda = 1$, there exists an optimal support pair (s^*,p^*) consisting of two adjacent vertices on the spine.*

Proof. We prove by contradiction. Let $s \in V_+$ and $p_1 \in V_-$ and (s,p_1) is optimal but not adjacent, i.e., there exists $p_2 \in V_-$ [LE: why] in the path connecting s and p_1.

Now, we aim to show that (s,p_2) is also optimal but having p_2 closer to s in a comparison with p_1. To this end, we consider

$$\begin{aligned} L_1(s,p_1) - L_1(s,p_2) &= f(s,p_1) - f(s,p_2) - (d(s,p_1) - d(s,p_2)) \\ &= d(p_2,p_1)|V_-(p_2,s)| + \sum_{v \in R} d(v,p_1) - d(p_2,p_1) \\ &= d(p_2,p_1)(|V_-(p_2,s)| - 1) + \sum_{v \in R} d(v,p_1). \end{aligned}$$

Since $V_-(p_2,s)$ contains at least a vertex (say p_1), we should have $|V_-(p_2,s)| - 1 \geq 0$. Thus, we obtain the nonnegativity. Therefore,

$$L_1(s,p_1) \geq L_1(s,p_2).$$

Since (s,p_1) is optimal, (s,p_2) is also optimal. This completes the proof.

Note that, there exists an optimal pair (s^*,p^*) lying on the spine. Otherwise, assume that p^* not on the spine, i.e., it is the original point, let p' be its projection, then it is clear that p' has the same sign as p^* while closer to s. Thus (s^*,p') is also optimal with p' on spine. If s^* is not on the spine, we repeat the same argument. We complete the proof. □

5.3 Algorithm and Computational Complexity

5.3.1 Preprocessing

Recall $x'_1, ..., x'_n$ are projections of $x_1, ..., x_n$ and we assume that $x'_1, ..., x'_n$ have been sorted in a certain order, e.g., by their first coordinates. By theorem XXX, we know that support vectors is a pair (x'_i, x'_{i+1}) with opposite signs. The margin is clearly $d(x'_i, x'_{i+1}) = \|x'_i - x'_{i+1}\|_2$. To compute the total loss, we only need $f(x'_i, x'_{i+1})$ for $i = 1, .., n-1$. In the following, we will clarify how to compute these values efficiently using dynamic programming.

For $i = 1, ..., n-1$, we define

$$\begin{aligned}
N_+^{>i} &= |\{v \in V_+(x'_i, x'_{i+1})\}| = 2|\{j > i : y_j = +1\}| \\
N_-^{>i} &= |\{v \in V_-(x'_i, x'_{i+1})\}| = 2|\{j > i : y_j = -1\}| \\
N_+^{\leq i} &= |\{v \in V_+(x'_{i+1}, x'_i)\}| = 2|\{j \leq i : y_j = +1\}| \\
N_-^{\leq i} &= |\{v \in V_+(x'_{i+1}, x'_i)\}| = 2|\{j \leq i : y_j = -1\}|
\end{aligned}$$

and

$$\begin{aligned}
d_+^{>i} &= \sum_{v \in V_+(x'_i, x'_{i+1})} d(x'_i, v) &&= \sum_{j>i, y_j=+1} [d(x'_i, x'_j) + d(x'_i, x_j)], \\
d_-^{>i} &= \sum_{v \in V_-(x'_i, x'_{i+1})} d(x'_i, v) &&= \sum_{j>i, y_j=-1} [d(x'_i, x'_j) + d(x'_i, x_j)], \\
d_+^{\leq i} &= \sum_{v \in V_+(x'_{i+1}, x'_i)} d(v, x'_{i+1}) &&= \sum_{j \leq i, y_j=+1} [d(x'_j, x'_{i+1}) + d(x_j, x'_{i+1})] \\
d_-^{\leq i} &= \sum_{v \in V_-(x'_{i+1}, x'_i)} d(v, x'_{i+1}) &&= \sum_{j \leq i, y_j=-1} [d(x'_j, x'_{i+1}) + d(x_j, x'_{i+1})].
\end{aligned}$$

Consider $i = 1, ..., n-1$. If $y_i = -1$ and $y_{i+1} = +1$, we then have

$$f(x'_i, x'_{i+1}) = d_-^{>i} + d_+^{\leq i}.$$

Else, that is $y_i = +1$ and $y_{i+1} = -1$, then

$$f(x'_i, x'_{i+1}) = d_+^{>i} + d_-^{\leq i}.$$

Proposition 5.3. *For $i = 1, ..., n-1$, if we have $N_-^{\leq i}$ and $d_-^{\leq i}$, then we can update $N_-^{\leq i+1}$ and $d_-^{\leq i+1}$.*

Proof. By definition,

$$d_-^{\leq i} = \sum_{\substack{j \leq i \\ y_j=-1}} \big(d(x'_j, x'_{i+1}) + d(x_j, x'_{i+1})\big), \qquad d_-^{\leq i+1} = \sum_{\substack{j \leq i+1 \\ y_j=-1}} \big(d(x'_j, x'_{i+2}) + d(x_j, x'_{i+2})\big).$$

First consider the sum over indices $j \leq i$ with $y_j = -1$. Due to the tree structure consisting of the spine (connecting the x'_k in order) and spokes (connecting each x_k to its projection x'_k), for all $j \leq i$, the unique path from x'_j (or x_j) to x'_{i+2} passes through x'_{i+1}. Therefore,

$$d(x'_j, x'_{i+2}) = d(x'_j, x'_{i+1}) + d(x'_{i+1}, x'_{i+2}), \qquad d(x_j, x'_{i+2}) = d(x_j, x'_{i+1}) + d(x'_{i+1}, x'_{i+2}).$$

Summing over all $j \leq i$ with $y_j = -1$, the increase due to shifting the anchor from x'_{i+1} to x'_{i+2} is

$$N_-^{\leq i} \cdot d(x'_{i+1}, x'_{i+2}),$$

since $N_-^{\leq i}$ counts exactly the number of negative vertices participating in the sum (each index j contributes two vertices x'_j and x_j).

Additionally, if $y_{i+1} = -1$, two new vertices appear in the region $\leq i+1$, namely x'_{i+1} and x_{i+1}. Their contribution to $d_-^{\leq i+1}$ is

$$d(x'_{i+1}, x'_{i+2}) + d(x_{i+1}, x'_{i+2}) = d(x'_{i+1}, x'_{i+2}) + \big(d(x_{i+1}, x'_{i+1}) + d(x'_{i+1}, x'_{i+2})\big) = 2\,d(x'_{i+1}, x'_{i+2}) + d(x_{i+1}, x'_{i+1}).$$

Combining both parts, we obtain the update formulas for the two cases based on the label y_{i+1}:

$$N_-^{\leq i+1} = \begin{cases} N_-^{\leq i} + 2, & \text{if } y_{i+1} = -1, \\ N_-^{\leq i}, & \text{if } y_{i+1} = +1, \end{cases}$$

$$d_-^{\leq i+1} = \begin{cases} d_-^{\leq i} + N_-^{\leq i}\, d(x'_{i+1}, x'_{i+2}) + 2\,d(x'_{i+1}, x'_{i+2}) + d(x_{i+1}, x'_{i+1}), & \text{if } y_{i+1} = -1, \\ d_-^{\leq i} + N_-^{\leq i}\, d(x'_{i+1}, x'_{i+2}), & \text{if } y_{i+1} = +1. \end{cases}$$

Clearly, each update step only uses quantities known at step i and distances between consecutive spine points, so it can be performed in constant time. Therefore, knowing $N_-^{\leq i}$ and $d_-^{\leq i}$ is sufficient to update $N_-^{\leq i+1}$ and $d_-^{\leq i+1}$. □

By above proposition, we have

Proposition 5.4. *We can compute the following quantities in $O(n)$.*

1. $(N_+^{>i}, N_-^{>i}, N_+^{\leq i}, N_-^{\leq i})$ *for all* $i = 1, ..., n-1$
2. $(d_+^{>i}, d_-^{>i}, d_+^{\leq i}, d_-^{\leq i})$ *for all* $i = 1, ..., n-1$
3. $f(x'_i, x'_{i+1})$ *for all* $i = 1, ..., n-1$.

Proof. Proof. We prove this by applying the previous proposition in a systematic manner, using the dynamic programming technique on trees (Proposition 5.3).

Step 1: Computing all N quantities in $O(n)$. The counting quantities can be computed by simple linear scans:

- $N_+^{>i}, N_-^{>i}$: Scan from right to left, counting positive/negative labels.
- $N_+^{\leq i}, N_-^{\leq i}$: Scan from left to right, counting positive/negative labels.

Each quantity is updated in $O(1)$ per position, giving total $O(n)$ time.

Step 2: Computing all d quantities using dynamic programming on trees. This is where we apply Proposition 5.3 (dynamic programming on trees). The key insight is that our augmented tree has a spine structure, which allows efficient re-rooting operations.

Left-to-right pass (computing $d_+^{\leq i}, d_-^{\leq i}$):

- Initialize: $d_+^{\leq 1} = 2d(x'_1, x'_2) + d(x_1, x'_2)$ if $y_1 = +1$, else 0.
- For $i = 1, 2, \ldots, n-2$: Apply Proposition 1 (and its symmetric version for positive labels) to update $d_\pm^{\leq i+1}$ from $d_\pm^{\leq i}$ in $O(1)$ time.

Right-to-left pass (computing $d_+^{>i}, d_-^{>i}$):

- Initialize: $d_+^{>n-1} = 0, d_-^{>n-1} = 0$ (no points beyond position $n-1$).
- For $i = n-1, n-2, \ldots, 2$: Use the symmetric update formula to compute $d_\pm^{>(i-1)}$ from $d_\pm^{>i}$ in $O(1)$ time.

The symmetric update formula for the right-to-left pass is obtained by reversing the roles: when moving from anchor x'_{i+1} to x'_i, all points $j > i$ have their distances increased by $d(x'_i, x'_{i+1})$, and if $y_i = -1$, we add the contribution of the new point i.

Step 3: Computing $f(x'_i, x'_{i+1})$ in $O(n)$. Once all N and d quantities are computed, we can evaluate $f(x'_i, x'_{i+1})$ for each $i = 1, \dots, n-1$ using the given formulas:

- If $y_i = -1, y_{i+1} = +1$: $f(x'_i, x'_{i+1}) = d_-^{>i} + d_+^{\leq i}$.
- If $y_i = +1, y_{i+1} = -1$: $f(x'_i, x'_{i+1}) = d_+^{>i} + d_-^{\leq i}$.

Each evaluation takes $O(1)$ time, giving total $O(n)$ time.

Conclusion: By Proposition 5.3 (dynamic programming on trees), the re-rooting operations along the spine can be performed efficiently. Since each of the three steps requires $O(n)$ time, the total time complexity is $O(n)$. □

□

5.3.2 Dynamic Programming for Objective Evaluation

To efficiently compute $f(u, v)$ for any pair (u, v), we employ a dynamic programming approach using two depth-first search (DFS) passes along the branch $u \to v$:

1. **Size computation pass**: Root the tree T at vertex u and compute, for every node a, the subtree sizes:

$$\mathrm{sz}_+(a) = |\{z \in \mathrm{subtree}(a) : y(z) = +1\}|, \tag{14}$$

$$\mathrm{sz}_-(a) = |\{z \in \mathrm{subtree}(a) : y(z) = -1\}|. \tag{15}$$

2. **Distance accumulation pass**: Still with the tree rooted at u, accumulate the weighted distances:

$$\mathrm{dist}_+(a) = \sum_{\substack{z \in \mathrm{subtree}(a) \\ y(z) = +1}} d(z, a), \tag{16}$$

$$\mathrm{dist}_-(a) = \sum_{\substack{z \in \mathrm{subtree}(a) \\ y(z) = -1}} d(z, a). \tag{17}$$

   These values are then combined along the unique path $u \to v$ to obtain $f_+(u, v)$. By symmetry, rooting the tree at v yields $f_-(v, u)$.

Since our augmented tree has a spine-and-spoke structure (essentially a path with leaves), all tree traversals can be performed in linear time.

5.3.3 Complete Algorithm

5.3.4 Complexity Analysis

The computational complexity of our algorithm is analyzed as follows:

- **Projection step**: Computing projections for n points in $\mathbb{R}^d$ requires $O(nd)$ operations.
- **Sorting step**: Sorting n projections by their coordinates takes $O(n \log n)$ time.
- **Tree construction**: Building the augmented tree with n spoke edges and $O(n)$ spine edges requires $O(n)$ time.

Algorithm 1 SVM on Tree Algorithm

```
1: Input: Labeled dataset $\{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \{-1, +1\}$
2: Output: Optimal support pair $(s^*, p^*)$ and decision boundary
3: Compute class means $\mu_-$, $\mu_+$ and spine direction $w$
4: Project all points: $x'_i = \mu_- + \langle x_i - \mu_-, w\rangle w$ for $i = 1, \dots, n$
5: Sort projections $\{x'_i\}$ by coordinate $t_i = \langle x_i - \mu_-, w\rangle$
6: Construct augmented tree $T$ with spoke and spine edges
7: Initialize $L_{\min} = \infty$, $(s^*, p^*) = \emptyset$
8: for each pair of adjacent spine vertices $(s, p)$ with $y(s) \neq y(p)$ do
9:    Compute $f(s, p)$ using two DFS passes
10:   Calculate $L(s, p) = f(s, p) - d(s, p)$
11:   if $L(s, p) < L_{\min}$ then
12:      $L_{\min} = L(s, p)$
13:      $(s^*, p^*) = (s, p)$
14:   end if
15: end for
16: return $(s^*, p^*)$ and perpendicular bisector of segment $s^*p^*$
```

- **Objective evaluation**: Each DFS pass takes $O(n)$ time, and there are $O(n)$ adjacent pairs to evaluate due to Theorem 5.2.

Theorem 5.5 (Complexity)**.** *The overall time complexity of Algorithm 1 is $O(nd + n \log n)$, or $O(n \log n)$ for fixed dimension d.*

This represents a significant improvement over exhaustive search, which would require $O(n^2)$ pair evaluations, each taking $O(n)$ time for objective computation.

5.3.5 Optimization Problem Statement

The core optimization problem solved by our algorithm is:

Problem 1 (SVM on Tree)**.** *Given the augmented tree T and loss function $L(\cdot, \cdot)$, find:*

$$(s^*, p^*) = \arg \min_{(s,p):y(s)\neq y(p)} L(s, p) = \arg \min_{(s,p):y(s)\neq y(p)} [f(s, p) - d(s, p)].$$

By Theorem 5.2, this optimization reduces to considering only adjacent opposite-label projection pairs on the spine, dramatically reducing the search space from $O(n^2)$ to $O(n)$ candidates.

6 Experiments

In this section, we evaluate the performance of our SVM on Tree method compared to the classical SVM implementation from scikit-learn. Our experiments demonstrate two key advantages: (1) significantly faster training and prediction times, and (2) comparable or better classification accuracy.

6.1 Experimental Setup

We implemented our SVM on Tree algorithm in C++ with Python bindings using pybind11 for efficient computation. The experiments compare our method against scikit-learn's SVC with RBF kernel on both synthetic and real-world datasets.

All experiments were conducted on synthetic parametric datasets with the following parameters:

- Spine separation: sep = 6.0

Table 1: Performance comparison on synthetic datasets

N (samples)	**Training Time (s) Tree / SVC**	**Prediction Time (s) Tree / SVC**	**Accuracy (%) Tree / SVC**	**Speedup Train / Pred**
100	0.000078 / 0.000820	0.000114 / 0.001065	88.0 / 87.0	10.5x / 9.4x
200	0.000088 / 0.001038	0.000072 / 0.002742	90.5 / 90.5	11.7x / 37.8x
400	0.000165 / 0.002175	0.000085 / 0.009270	89.0 / 89.0	13.2x / 108.8x
1000	0.000402 / 0.009031	0.000114 / 0.045735	89.1 / 88.4	22.5x / 401.2x
2000	0.002271 / 0.138671	0.000289 / 0.649851	88.85 / 88.85	61.1x / 2249.6x
5000	0.009452 / 0.908959	0.000618 / 4.074999	88.69 / 88.70	96.2x / 6595.8x

- Parallel noise: $\sigma_{\text{para}} = 2.5$
- Perpendicular noise: $\sigma_{\text{perp}} = 2.5$
- Correlation: $\rho = 0.0$

Each experiment was repeated 3 times and results were averaged to ensure statistical reliability.

6.2 Synthetic Dataset Results

Table 1 shows the performance comparison on synthetic datasets of varying sizes. The results demonstrate consistent speedups in both training (FIT) and prediction (PRED) phases while maintaining competitive accuracy.

6.3 Performance Analysis

The experimental results reveal several important findings:

1. **Training Speedup**: Our SVM on Tree method achieves consistent training speedups ranging from 10.5x to 96.2x compared to scikit-learn's SVC. The speedup increases dramatically with dataset size, demonstrating exceptional scalability.
2. **Prediction Speedup**: The prediction phase shows even more dramatic improvements, with speedups ranging from 9.4x to an impressive 6595.8x for the largest dataset (N=5000). This extraordinary speedup makes our method ideal for real-time applications.
3. **Accuracy Preservation**: Our method maintains competitive accuracy across all dataset sizes, consistently matching the classical SVM performance (around 88-90% accuracy).
4. **Scalability**: The computational advantage becomes more pronounced with larger datasets, with the most significant improvements observed at N=5000 where prediction is over 6500 times faster than classical SVM.

6.4 Real-world Dataset Evaluation

We further evaluate our method on real-world datasets to demonstrate its practical applicability beyond synthetic data.

6.4.1 Iris Dataset

The Iris dataset, a classic benchmark in machine learning, contains 150 samples with 4 features representing sepal and petal measurements of iris flowers. For our experiments, we applied PCA to reduce the data to 2 dimensions and converted it to a binary classification problem.

Table 2: Performance comparison on Iris dataset (median of 10 runs)

Metric	SVM on Tree	SVC (RBF)	Speedup/Difference
Training Time	0.000052s	0.000643s	12.33x
Prediction Time	0.000101s	0.000159s	1.58x
Accuracy	100.0%	100.0%	+0.0%

Table 2 shows the performance comparison on the Iris dataset over 10 runs:
The Iris dataset demonstrates excellent suitability for our tree-based approach with:

- **Suitability Score**: 5.358 (indicating high compatibility with tree structure)
- **Spine Dominance**: 211.306 (showing clear linear separability along the principal direction)
- **Zero Overlap**: 0.000 overlap on spine, indicating clean separation

Both methods achieved perfect classification accuracy (100%), but our SVM on Tree method was 12.33 times faster in training and 1.58 times faster in prediction.

6.4.2 Wine Dataset

The Wine dataset contains chemical analysis of wines from three different cultivars. We converted this to a binary classification problem and reduced the dimensionality to 2D using PCA for compatibility with our tree-based approach.

Table 3 shows the performance comparison on the Wine dataset over 10 runs:

Table 3: Performance comparison on Wine dataset (median of 10 runs)

Metric	SVM on Tree	SVC (RBF)	Speedup/Difference
Training Time	0.000075s	0.000713s	9.52x
Prediction Time	0.000101s	0.000182s	1.79x
Accuracy	88.89%	88.89%	+0.0%

The Wine dataset presents a more challenging scenario with:

- **Suitability Score**: 0.728 (moderate compatibility with tree structure)
- **Spine Dominance**: 1.885 (lower than Iris, indicating less clear linear separability)
- **Class Imbalance**: 0.331 (indicating uneven class distribution)
- **Spine Overlap**: 0.225 (some overlap along the principal direction)

Despite the more challenging characteristics of the Wine dataset, our SVM on Tree method achieved identical accuracy (88.89%) to the classical SVM while providing 9.52x speedup in training and 1.79x speedup in prediction.

6.5 Discussion

The experimental results across both synthetic and real-world datasets demonstrate the effectiveness of our SVM on Tree approach:

1. **Computational Efficiency**: Consistent speedups ranging from 1.58x to 6595.8x in prediction time and 9.52x to 96.2x in training time across different dataset sizes and types. The most impressive results are observed with larger synthetic datasets where prediction speedup exceeds 6500x.

2. **Accuracy Preservation**: Our method maintains identical accuracy to classical SVM across all tested datasets. On well-structured data like Iris, both methods achieve perfect classification (100%), while on more complex datasets like Wine, both achieve competitive performance (88.89%).

3. **Scalability**: The performance advantage becomes more pronounced with larger datasets, making our approach particularly suitable for big data applications. Synthetic experiments show speedups exceeding 6500x for prediction on the largest datasets (N=5000).

4. **Dataset Adaptability**: The method works effectively across datasets with varying characteristics:

    - High suitability (Iris: score 5.358) with perfect linear separability
    - Moderate suitability (Wine: score 0.728) with class imbalance and spine overlap
    - Synthetic datasets with controlled noise and separation parameters

5. **Practical Applicability**: The method works effectively on real-world datasets, not just synthetic ones, demonstrating its practical value for machine learning applications.

These results validate our theoretical analysis and confirm that the tree-based formulation provides a computationally efficient alternative to classical SVM without sacrificing classification quality. The suitability score metric effectively predicts when our method will perform optimally, helping practitioners decide when to apply this approach.

The synthetic results demonstrate that our SVM on Tree method successfully addresses the computational challenges of traditional SVM while preserving classification quality. The significant speedups in both training and prediction phases, combined with maintained accuracy, make our approach particularly suitable for large-scale applications and real-time systems.