THE SUPPORT VECTOR MACHINE TREE

Anonymous authors
Paper under double-blind review

ABSTRACT

Support vector machines (SVMs) are an established binary classification model which enjoys good generalization performance. However, a long-standing problem is their high training and inference time. We propose a new model, the Support Vector Machine Tree, which applies a divide-and-conquer strategy but where, crucially, both the tree partition and the leaf SVMS are jointly optimized with an RKHS penalty. This brings several important improvements compared to the exact SVM as well as to existing approximate approaches. Computationally, the memory and runtime (at either training or inference) are far smaller, because the heavy SVM cost is confined to each leaf of the tree, which drastically reduces the number of support vectors in an instance-dependent way. The test accuracy is also improved, particularly in larger datasets, where other approaches cannot even run. Finally, the tree structure provides some interpretability.

1 INTRODUCTION

We consider binary classification with support vector machines (SVMs). Kernel SVMs provide an elegant solution to a nonparametric, infinite-dimensional function estimation over a reproducing kernel Hilbert space (RKHS). While they are widely praised for their generalization ability, they also suffer from a high computational cost: at least quadratic on the sample size N at training time in both memory and time, due to the kernel Gram matrix; and linear on N at inference time, since the classifier is a sum of basis functions (BFs) at the support vector (SVs), which are a (usually large) subset of the training instances.

We propose a new model to address this issue, the Support Vector Machine tree (SVM tree). We use a divide-and-conquer strategy, in the hope that simpler problems (operating on a local region of the input space and trained on the instances in it) should be easier and faster to solve—as long as we do not have too many, small problems that overfit. This idea has long been exploited in decision trees, which typically have a constant classifier in each leaf. But in our case each leaf classifier is a kernel SVM, which brings a side effect that is highly beneficial in computational cost. Since the SVM classifier is a sum over the SVs, each leaf SVM can only use (a subset of) the training instances within that leaf. Using a complete tree of depth Δ, each of the $m = 2^{\Delta}$ leaves receives on average N/m training instances. At training time, solving m subproblems of quadratic cost on the sample size is thus m times cheaper (m^2 times cheaper if solved in parallel). At inference time, the BF expansion is also m times cheaper. Thus, the deeper the tree the faster.

Naturally, the selection of individual, local leaf subproblems matters greatly. Some areas of input space may be easier to classify than others. Thus, it is critical to optimize the leaf SVMs jointly with the tree partition. Since the tree must make hard decisions (so that the leaf SVMs are independent and we reap the above speedups), the resulting training problem is not differentiable. However, we can adapt the framework of Tree Alternating Optimization (TAO) (Carreira-Perpiñán & Tavallali, 2018), which works by updating the parameters of one node at a time. In our case, we can reuse existing SVM algorithms to solve the subproblem at each leaf.

The above arguments suggest large computational savings are possible. The practical reality is more complex, though. The tree partition is usually not balanced, and some leaves may even receive no training instances, so they can be pruned. Some leaves may receive instances from only one class, so they become a constant classifier. If the tree is too big, the model will overfit. Ultimately, this is an empirical question. In a range of datasets of different size and dimensionality we find that, with hyperparameters determined by cross-validation, the SVM tree does indeed live up to this

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

promise: it produces classifiers whose test accuracy exceeds that of global, exact SVMs and existing fast approximate approaches, while being very fast at training and inference. The improvement becomes considerable for larger datasets. Also, we can achieve far faster models if we allow for a small decrease in accuracy. In addition, the tree structure introduces some interpretability into the partition, for example exposing subclass structure and how it relates to the input features.

Finally, we note that the predictive function of the SVM tree looks at first sight like that of the global SVM: it is a BF expansion over the training instances. However, the coefficients are constant in an SVM and instance-dependent and sparse in an SVM tree. Thus, the latter is neither an exact SVM (using some kernel in some RKHS space) nor an approximate SVM, but a really different type of model, where the complexity of individual leaf predictors adapts locally to the data. The SVM tree is also orthogonal to approaches that seek to approximate an exact SVM (see related work); such approaches could be applied to the tree leaves.

2 Related work

Low-rank approaches These methods apply some form of low-rank structure in the kernel matrix so fast linear algebra computations apply. For example, the *Nyström method* approximates the kernel matrix by sampling a subset of columns, which corresponds to selecting a subset of training instances as pseudo-SVs (Williams & Seeger, 2001; Zhang et al., 2008). The *Random Fourier Features* (RFF) method approximates shift-invariant kernels by mapping data to a finite-dimensional randomized feature space (Rahimi & Recht, 2008). These methods are computationally efficient but unsupervised and completely disjoint from the SVM's actual objective. Chen et al. (2017) define a single global kernel SVM but using a special type of kernel based on a hierarchical partitioning of the data domain, so the kernel matrix takes a recursively off-diagonal low-rank structure.

Subsampling and coreset approaches These seek a global SVM, i.e., a single BF expansion applied to any test instance, but with a smaller set of BFs. For example, *data squashing* reduces the training set by replacing dense regions with weighted pseudo-samples (Pavlov et al., 2000), without losing the statistican properties of the data, or cluster centroids (Boley & Cao, 2004; Yu et al., 2005). Conversely, *core-sets* select a subset of training instances that approximates the original SVM solution (Tsang et al., 2005).

Divide-and-Conquer Approximations Rather than routing samples during inference, this methods use localized training to accelerate the convergence of a single global model or ensemble. Cascade SVM (Graf et al., 2005) and its Spark-optimized variant PB-SVM (Li et al., 2023) hierarchically extract support vectors across local subsets and train SVM on aggregated samples. The DC-SVM (Hsieh et al., 2014b) seeks to approximate the exact, global SVM solution. They first divide the training samples with kernel clustering and fit a separate kernel SVM to each. Then they combine the cluster solutions (which are likely to contain SVs of the global SVM) as an initialization for a global SVM solver, which will thus converge faster. Alternatively, Parallel Mixture of SVMs (Collobert et al., 2002) dynamically re-routes samples to local experts via a neural gating function. Finally, (Zhang et al., 2015) consider kernel ridge regression (KRR) and use divide-and-conquer with a random balanced partition of the N samples into m subsets. They compute an independent KRR in each subset with a carefully chosen regularization parameter and estimate the output as the average of the m KRRs. Thus, the resulting model is not a global KRR, but an averaged ensemble of subsampled KRRs. They show this achieves computational speedups while retaining good statistical behavior as long as m is not too large.

Local SVMs These methods partition the feature space to route test instances to specific local SVM. Tree-based approaches like DTSVM (Chang & Liu, 2012) and CombNET-III (Kugler et al., 2006) partition the space hierarchically, using decision trees or sequential clustering networks. Clustering and projection methods partition data into disjoint sets using spatial proximity such as K-means in CSVM (Gu & Han, 2016) or boundary-focused clustering in BCSVM (Fayed & Atiya, 2021), or via linear/FLD projections (L-SVM (Song et al., 2019), FL-SVMs (Zeng et al., 2026)) to train localized models on balanced sub-regions. Spatial methods evaluate samples against defined spatial grids (Thomann et al., 2017) or use Partition of Unity (PU) to smoothly connect overlapping local decision boundaries into a unified classifier (LGSVM (Marchetti & Perracchione, 2022)).

The theoretical soundness of localized SVMs is well-established: they can achieve faster classification rates than standard global SVMs by adapting hyperparameters per cell (Blaschzyk & Steinwart, 2022), and they preserve L_p- and risk consistency even when partitions change dynamically as data grows (Köhler, 2024).

A critical limitation of existing localized SVMs is that the heuristic hard data partitioning and the classifier training are treated as completely disjoint steps. Therefore, suboptimal partitions cannot be corrected by the downstream classifiers. In contrast, our approach allows for the joint optimization of both the hard routing mechanism (an oblique decision tree) and the local experts (leaf SVMs), which leads to better local optima and highly efficient inference.

3 Definition and Optimization of the SVM Tree

3.1 The SVM tree model

The SVM tree can be seen as an oblique decision tree having a classifier at each leaf that is a basis function (BF) expansion (a binary SVM). A test instance $\mathbf{x}$ follows a single root-leaf path and applies the leaf classifier. In detail, the SVM tree defines a mapping $T(\mathbf{x}; \mathbf{\Theta}) \in \mathbb{R}$ from a feature vector $\mathbf{x} \in \mathbb{R}^D$ to a scalar with parameters $\mathbf{\Theta} = \{\{(\mathbf{w}_i, w_{i0})\}_{i\in\mathcal{D}}, \{g_j\}_{j\in\mathcal{L}}\}$, as follows. Consider a binary tree (i.e., where each decision node has exactly two children) of fixed structure having decision nodes and leaves indexed by sets $\mathcal{D}$ and $\mathcal{L}$, respectively. (For example, for a complete tree of depth Δ, we could have $\mathcal{D} = \{1, \dots, 2^\Delta - 1\}$ and $\mathcal{L} = \{2^\Delta, \dots, 2^{\Delta+1} - 1\}$.) Decision node $i \in \mathcal{D}$ defines an oblique (hyperplane) split with parameters $\mathbf{w}_i \in \mathbb{R}^D$, $w_{i0} \in \mathbb{R}$: it sends input instance $\mathbf{x} \in \mathbb{R}^D$ to its right child if $\mathbf{w}_i^T\mathbf{x} + w_{i0} > 0$ and to its left child otherwise. Leaf $j \in \mathcal{L}$ applies a function $g_j \in \mathcal{H}$ to predict a class $g_j(\mathbf{x}) \in \mathbb{R}$ for instance $\mathbf{x}$, where $\mathcal{H}$ is a reproducing kernel Hilbert space (RKHS) with kernel $K(\cdot,\cdot)$ and norm $\|\cdot\|_\mathcal{H}$. Note the kernel (and its hyperparameters) are the same at each leaf. This will define a leaf SVM, i.e., as a consequence of the objective function (1) below, g_j will take the form of a sum of BFs $K(\cdot, \mathbf{x}_n)$ sitting at a certain subset of the leaf training instances (the support vectors of leaf j). Thus, like a global SVM, *the entire SVM tree model can be written as a sum of a basis function over each training instance,* $T(\mathbf{x}) = \sum_{n=1}^N \alpha_n(\mathbf{x})\, K(\mathbf{x}, \mathbf{x}_n)$*, but where the coefficients* $\alpha_n(\mathbf{x})$ *are 1) dependent on the input (test) instance* $\mathbf{x}$*, 2) sparse (only a small subset of coefficients are nonzero for any* $\mathbf{x}$*) and 3) fast to compute (by following a single root-leaf path in the tree).* Finally, we define the *reduced set (RS)* $\mathcal{R}_i \subset \{1, \dots, N\}$ of a node (decision node or leaf) i as the set of training instances that reach i under the current tree parameters.

3.2 The training problem

Consider a binary classification problem with training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N \subset \mathbb{R}^D \times \{-1, +1\}$. The SVM tree is learned as:

$$\min_{\mathbf{\Theta}} \sum_{n=1}^N L(y_n, T(\mathbf{x}_n; \mathbf{\Theta})) + \lambda \sum_{i\in\mathcal{D}} \phi_i(\mathbf{w}_i) + \mu \sum_{j\in\mathcal{L}} \|g_j\|_\mathcal{H}^2 \tag{1}$$

where $L(y, y') = \max(0, 1 - yy')$ for $y, y' \in \mathbb{R}$ is the hinge loss and ϕ_i is a regularization term on the hyperplane split (we use the ℓ_1 norm to encourage sparse splits), with hyperparameters $\lambda, \mu > 0$.

3.3 An optimization algorithm

Objective function (1) is not differentiable, but we can optimize it one node at a time, as proposed by Carreira-Perpiñán & Tavallali (2018) for decision trees. This relies on three theorems; the first two are as in (Carreira-Perpiñán & Tavallali, 2018) and apply to our case directly, and are as follows. First, a **separability condition** states that a set of nodes that are not co-descendant (e.g. all nodes at the same depth) can be optimized independently and in parallel given the parameters of all other nodes are fixed, because then (1) separates over such set. Second, the **decision node reduced problem** states that optimizing (1) over the parameters of a decision node $i \in \mathcal{D}$ is equivalent to a weighted 0/1 loss binary classification problem over $(\mathbf{w}_i, w_{i0})$, where the class labels are defined as the left or right child of i, each incurring a corresponding loss for each instance $\mathbf{x}_n \in \mathcal{R}_i$ (obtained by routing $\mathbf{x}_n$ under each subtree to a leaf). This problem is NP-hard but can be solved approximately

input binary oblique tree T with given structure and parameters $\boldsymbol{\Theta}$
at the nodes $\mathcal{N} = \mathcal{D} \cup \mathcal{L}$
repeat
  for $i \in \mathcal{N}$
    $\mathcal{R}_i \leftarrow$ reduced set of node i
  for $i \in \mathcal{N}$ in reverse BFS order
    if $i \in \mathcal{L}$ **then** leaf
      if $\mathcal{R}_i = \varnothing$ **then** $g_i \leftarrow 0$
      else if $y_n = y \; \forall n \in \mathcal{R}_i$ **then** $g_i \leftarrow y$ according to 4
      else $g_i \leftarrow \arg\min_{f \in \mathcal{H}} \sum_{n \in \mathcal{R}_i} L(y_n, f(\mathbf{x}_n)) + \mu \|f\|^2_{\mathcal{H}}$
    else decision node
      if $\mathcal{R}_i \neq \varnothing$ **then**
        compute pseudolabels z_n and weights $c_n \; \forall n \in \mathcal{R}_i$ (appendix B)
        solve surrogate of objective function (2)
          if node update increases $E(\boldsymbol{\Theta})$ **then** revert to previous $(\mathbf{w}_i, w_{i0})$
    until $E(\boldsymbol{\Theta})$ changes less than a set threshold
  postprocess T: remove redundant nodes
  return T

Figure 1: Pseudocode for the SVM tree training algorithm.

by a surrogate loss (e.g. ℓ_1-regularized logistic regression); see details in appendix B, objective function (2).

The third and novel theorem (whose proof is in appendix C) characterizes the solution of optimizing (1) over one leaf's function given the rest of the tree parameters are fixed.

Theorem 3.1 (Reduced problem at a leaf)**.** *Consider leaf $j \in \mathcal{L}$ with reduced set $\mathcal{R}_j$ under the current tree. Minimizing objective function* (1) *over $g_j \in \mathcal{H}$ given all other parameters $\boldsymbol{\Theta} \setminus \{g_j\}$ are fixed has the following solution g_j^*, with three cases:*

1. Empty leaf*: if $\mathcal{R}_j = \varnothing$ then $g_j^* \in \arg\min_f \|f\|^2_{\mathcal{H}}$ (usually $g_j^* \equiv 0$).*
2. Pure leaf*: if $\mathcal{R}_j \neq \varnothing$ and $y_n = y \; \forall n \in \mathcal{R}_j$ (where either $y = +1$ or $y = -1$) then $g_j^* \equiv y$ is a solution.*
3. Two classes*: otherwise, g_j^* is the solution of $\min_{f \in \mathcal{H}} \sum_{n \in \mathcal{R}_j} L(y_n, f(\mathbf{x}_n)) + \mu \|f\|^2_{\mathcal{H}}$. This is a usual SVM problem but over j's RS and thus has the form $g_j^*(\mathbf{x}) = \sum_{n \in \mathcal{R}'_j} \alpha_{jn} y_n K(\mathbf{x}, \mathbf{x}_n)$ for certain coefficients $\{\alpha_{jn}\} \subset \mathbb{R}$ where $\mathcal{R}'_j \subseteq \mathcal{R}_j$ are the support vectors (SVs) of leaf j.*

Remarks. Cases 1 and 2 are unusual in practical machine learning because we would not "train" an SVM if the training set is either empty or only contains one class, but they arise naturally in the SVM tree. Case 1 will lead to leaf j and its sibling being pruned and their parent becoming a leaf, as a consequence of the ℓ_1 penalty on the decision nodes; *this effectively learns the structure of the optimal tree* (as a substructure of the initial tree). Case 2 leads to a constant-label leaf, as in traditional decision trees. Case 3 is the usual kernel SVM, but *its SVs come only from the leaf j's RS (not the entire training set)*. This leads to a huge reduction in the SVM size. Finally, since the above-mentioned separability condition applies to the set of all leaves, theorem 3.1 leads to a **necessary optimality condition** for the SVM tree, as follows. Let $\boldsymbol{\Theta}^*$ be a local optimum of (1). Then $\{g_j^*\}_{j \in \mathcal{L}}$ are as above for each leaf using the RSs resulting from $\{(\mathbf{w}_i^*, w_{i0}^*)\}_{i \in \mathcal{D}}$.

3.4 Complete optimization algorithm and practicalities

The final algorithm simply updates each node in turn either by fitting an ℓ_1-regularized logistic regression (decision node approximate solution) or an SVM (leaf exact solution). We update the nodes in reverse breadth-first order (leaves to root) and stop iterating when the objective function (1) changes less than a set threshold. The initial tree is a random median tree, i.e., each decision node has a random direction and a bias to effect a 50/50 split. See the pseudocode in fig. 1. By only

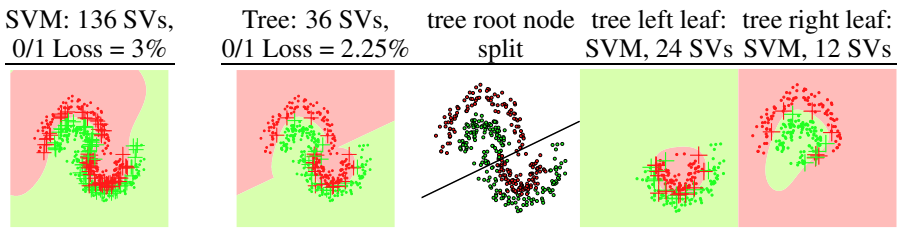

Figure 2: SVM and SVM tree of depth 1 on 2D toy dataset.

accepting the decision node update if it decreases its objective, we can guarantee that each node update (hence each iteration, or pass over all nodes) decreases (1) or leaves it unchanged. Leaf SVM QP is optimized using the SMO algorithm implemented in libsvm (Chang & Lin, 2011).

The behavior of the model depends heavily on the dataset and the chosen hyperparameters, which control the balance between the tree's routing structure and the SVMs at its leaves. At one extreme, using a depth of zero or a very large penalty parameter (λ) prevents any branching, turning the model into a single, global SVM. At the opposite extreme, using a large depth with a small λ forces the tree to keep splitting until every leaf contains only one class. This makes the leaf SVMs unnecessary, reducing the model to a standard decision tree. We discuss this special case in section C (eq. 4). How often these extreme cases occur depends entirely on the data being used. However, intermediate hyperparameter values are generally the most effective in practice (Figure 10). They strike a practical balance, resulting in a moderately deep tree that routes data to relatively small, well-regularized SVMs at its leaves to achieve the best overall accuracy. We discuss the effect of hyperparameters in more detail in the Appendix (section C.3).

4 Computational complexity for training and inference

Assume a dataset with N instances and D features. Let the model be a complete tree of depth Δ where leaves are reached with equal probability. Decision nodes use sparse weight vectors with sD nonzeros ($s \in [0, 1]$), and each leaf contains $\rho N 2^{-\Delta}$ support vectors ($\rho \in [0, 1]$). While our unoptimized Python implementation is sequential (see Table 1), all nodes at the same depth can theoretically be updated in parallel to accelerate training.

Inference Inference involves routing an instance to a leaf ($\mathcal{O}(sD\Delta)$) and applying the local kernel SVM ($\mathcal{O}(D\rho N 2^{-\Delta})$), for a total time of $\mathcal{O}((s\Delta + \rho N 2^{-\Delta})D)$. Because deep trees partition data into small regions, each leaf contains very few support vectors, making inference exponentially faster with depth.

Training decision nodes The cost of one training iteration (updating each node in the tree once) is decomposed into three main components, as illustrated in Figure 3: pseudolabel computation, training linear model in the decision nodes, and leaf SVMs. First, in each decision node we construct RP (compute pseudolabels) and solve ℓ_1-regularized logistic regression. *Pseudolabel computation* involves sending each training sample down to left or right subtree (opposite to the currently chosen) to compute its loss. At each depth all N training instances are sent down the tree and the depth span from 0 to $\Delta - 1$, therefore the total cost is $\mathcal{O}\left(\frac{1}{2}sND\Delta(\Delta - 1) + \rho N^2 D\Delta 2^{-\Delta}\right)$. Second, the RP solution is proportional to running Δ logistic regressions on the whole dataset, which is $\mathcal{O}(c_1 ND\Delta)$ for some constant c_1. Asymptotically, the total cost is $\mathcal{O}\left(ND\Delta(\Delta + \rho N 2^{-\Delta})\right)$.

Training leaf nodes The cost of training a non-linear leaf SVM with a solver like SMO with $N2^{-\Delta}$ instances is roughly $\mathcal{O}((N2^{-\Delta})^2 D + c_2(N2^{-\Delta})^\alpha)$ where $\alpha \in [2, 3]$ and c_2 is a constant. The total cost is $\mathcal{O}\left(N^2 D 2^{-\Delta} + c_2 N^\alpha 2^{-\Delta(\alpha-1)}\right)$.

As shown in Figure 3 (first column), the leaf node training time decreases *exponentially* ($[2^{-\Delta}, 2^{-2\Delta}]$) as Δ grows. For shallow trees, training is dominated by the leaf SVMs, whose complexity scales superlinearly with the number of samples. For deep trees, the kernel SVM cost plummets rapidly, and the overall cost is eventually dominated by the decision nodes, $\mathcal{O}(ND\Delta^2)$.

Table 1: Comparison of training time (s) for different methods on large datasets.

Dataset	SVM tree	RFF	Nyström	CascadeSVM	DCSVM	hSVM
Covertype (500K,10,2)	520	2268	2077	58687	34522	41
Poker Hand (871K,10,2)	587	1846	1815	78988	65731	71
ADC (1.5M,28,2)	1390	1441	1077	88642	82655	239
SUSY (4M,18,2)	2900	3388	3189	timeout	timeout	800

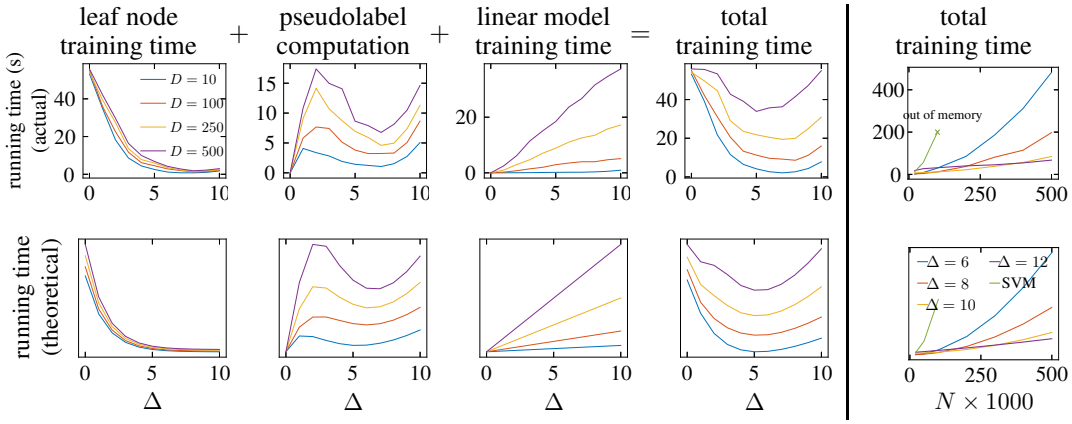

Figure 3: *Left panels:* training time over tree depth (Δ) and feature dimensionality (D), on MNIST (even vs odd digits). *Right panel:* training time over sample size (N) and tree depth (Δ), on SUSY.

This property makes the algorithm highly scalable to large sample sizes without the need for any approximation. As seen in the right panel of Figure 3, the total training time for a fixed, sufficiently large Δ (e.g., $\Delta = 10$ or 12) scales linearly with the sample size N, avoiding the intractability of global kernel SVMs (green line in Figure 3 right panel). The total training cost is bounded by $\mathcal{O}\left(\rho N^2 D \Delta 2^{-\Delta}\right) \leq \mathcal{O}\left(c_2 N^{\alpha} 2^{-\Delta(\alpha-1)} + \rho N^2 D \Delta 2^{-\Delta}\right) \leq \mathcal{O}\left(N^3 2^{-2\Delta}\right)$.

Memory Complexity During *training*, standard kernel SVMs suffer from $\mathcal{O}(N^2)$ memory bottleneck when caching the global kernel matrix. SVM tree trains locally, reducing the maximum local matrix size to $\mathcal{O}(N^2 2^{-2\Delta})$ and eliminating out-of-memory errors. During *inference*, global SVMs require $\mathcal{O}(\rho_{global} N D)$ memory. SVM tree instead stores sparse decision weights and leaf support vectors, totaling $\mathcal{O}((s2^{\Delta} + \rho_{tree} N)D)$. For large N, the tree's routing overhead is negligible, and both models scale linearly as $\mathcal{O}(ND)$. Because deep trees isolate highly separable regions, SVM tree requires significantly fewer support vectors ($\rho_{tree} \ll \rho_{global}$), ensuring linear storage.

5 Experiments

We empirically validate SVM tree across various datasets (Section 5.1). We then detail its training and inference efficiency (Section 5.2), and joint optimization benefits (Section 5.3). Details on hyperparameter dynamics (Section C.3), insights into the interpretability of the sparse oblique decision process (Section C.2) are provided in the appendix A. These results convincingly demonstrate the computational speedups and improved accuracy over both the exact and approximate SVM algorithms.

5.1 SVM trees produce more accurate classifiers than other SVM models

SVM tree demonstrates the best generalization and inference efficiency against exact SVM, kernel approximations (RFF, Nyström), and scalable baselines (CascadeSVM, DC-SVM, hSVM) across all datasets (3.9K to 4M samples) as detailed in Table 2. It consistently outperforms exact SVM on feasible datasets (e.g., 13.03% vs 14.79% on Electricity and 4.92% vs 5.93% on Spambase) and scales robustly to massive regimes where CascadeSVM and DC-SVM fail due to timeouts. On large-scale datasets, SVM tree remains dominant, reducing test error by ≈2% on Covertype and

Table 2: Comparison of different methods. **Bold blue** = best, and blue = second best. The number of support vectors is counted on the test set for all methods. For RFF/Nyström, this number represents the equivalent dual inference complexity. For tree-based methods (SVM tree and hSVM) we report the depth (Δ) and total number of nodes in the tree.

		SVM tree	SVM	RFF	Nyström	CascadeSVM	DCSVM	hSVM
Spambase (3910,57,2)	$E_{\texttt{train}}$	2.43	4.04	2.51	2.68	4.04	3.78	3.58
	$E_{\texttt{test}}$	**4.92**	5.93	6.94	6.22	5.94	6.08	7.09
	Δ / #nodes	4 / 29	—	—	—	—	—	2 / 7
	# SVs on test	**116**	861	718	694	831	249	249
Electricity (32702,8,2)	$E_{\texttt{train}}$	3.26	6.36	19.44	19.23	10.63	10.28	9.83
	$E_{\texttt{test}}$	**13.03**	14.79	20.79	20.53	15.66	15.74	16.29
	Δ / #nodes	6 / 29	—	—	—	—	—	3 / 15
	# SVs on test	1228	11189	15877	14004	12107	6474	1687
Compass (14147,17,2)	$E_{\texttt{train}}$	17.16	15.12	19.96	21.65	5.75	5.58	5.09
	$E_{\texttt{test}}$	25.15	**25.19**	27.2	27.11	26.67	26.71	27.35
	Δ / #nodes	5 / 63	—	—	—	—	—	2 / 7
	# SVs on test	**613**	8148	12477	7103	9893	5797	2531
House 16H (11464,16,2)	$E_{\texttt{train}}$	6.93	10.5	9.49	12.1	11.54	12.53	10.44
	$E_{\texttt{test}}$	**13.7**	13.9	14.62	14.4	14.17	15.96	14.97
	Δ / #nodes	3 / 7	—	—	—	—	—	2 / 7
	# SVs on test	**1054**	4265	6693	4216	3780	3452	1145
Bank Marketing (38429,31,2)	$E_{\texttt{train}}$	6.47	10.2	9.56	8.55	7.34	10.16	10.09
	$E_{\texttt{test}}$	**9.44**	9.68	9.69	9.74	9.92	9.81	9.94
	Δ / #nodes	3 /15	—	—	—	—	—	2 / 7
	# SVs on test	2338	8559	6981	7266	8200	**1497**	2156
MiniBooNE (62048,50,2)	$E_{\texttt{train}}$	6.31	7.74	7.87	7.80	7.78	4.49	10.16
	$E_{\texttt{test}}$	**7.23**	7.92	8.12	7.89	9.25	8.94	10.45
	Δ / #nodes	2 / 7	—	—	—	—	—	2 / 7
	# SVs on test	**3304**	14824	13373	13397	17141	4119	4658
Covertype (500K,10,2)	$E_{\texttt{train}}$	3.63	OOM	17.82	16.38	5.95	3.74	13.23
	$E_{\texttt{test}}$	**5.63**		17.99	16.5	7.45	7.79	14.47
	Δ / #nodes	13 / 4441	—	—	—	—	—	8 / 511
	# SVs on test	**111**		186918	183700	210211	929	751
Poker Hand (871K,10,2)	$E_{\texttt{train}}$	20.7	OOM	37.66	37.14	37.7	28.04	25.76
	$E_{\texttt{test}}$	**25.1**		37.98	37.50	38.7	35.59	38.61
	Δ / #nodes	11 / 1361	—	—	—	—	—	10 / 2047
	# SVs on test	2557		661614	650818	546540	1400	**665**
ADC (1.5M,28,2)	$E_{\texttt{train}}$	11.01	OOM	14.09	17.00	15.95	8.46	12.3
	$E_{\texttt{test}}$	**11.86**		14.11	17.02	16.1	12.71	12.85
	Δ / #nodes	9 / 1023	—	—	—	—	—	8 / 511
	# SVs on test	**960**		465597	574384	390629	624263	2373
SUSY (4M,18,2)	$E_{\texttt{train}}$	19.87	OOM	23.17	22.76	timeout	timeout	29.89
	$E_{\texttt{test}}$	**19.96**		23.16	23.06			29.91
	Δ / #nodes	11 / 6747	—	—	—	—	—	9 / 511
	# SVs on test	15836		1879330	267095			**11403**

≈10% on Poker Hand. On the 4M-sample SUSY dataset, it surpasses the strongest approximation baseline (Nyström) by ≈4% and tree-based hSVM by ≈10%, emphasizing the importance of joint optimization.

Crucially, SVM tree drastically reduces inference costs without sacrificing accuracy. It requires up to three orders of magnitude fewer SVs than exact SVM, DCSVM or CascadeSVM (e.g., 111 vs 210,211 on Covertype). While hSVM occasionally uses fewer SVs on massive datasets, it suffers accuracy drop (Poker Hand: 25.1 error vs hSVM's 38.61). By counting test samples within the approximated margin, we extract an "equivalent SVs" metric for RFF and Nyström that isolates theoretical dual inference cost and enables space-agnostic (apples-to-apples) sparsity comparisons. More detailed analysis of inference dynamics in provided in Section 5.2.

5.2 SVM TREES NOT ONLY ACCURATE BUT VERY FAST AT TRAINING AND TEST TIME

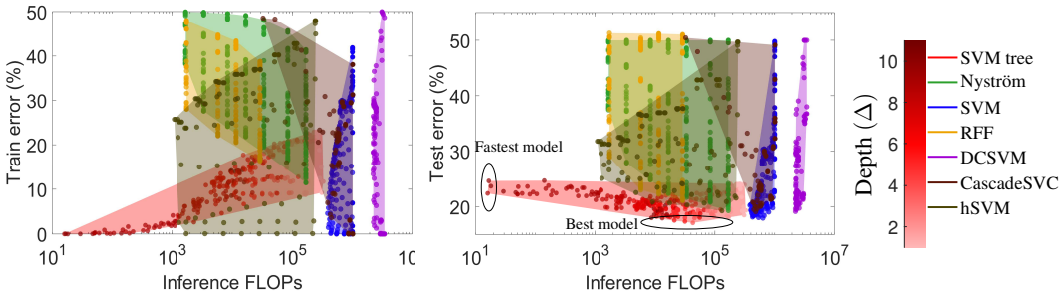

Figure 4: 0/1 Loss (training, test) vs inference FLOPs for various SVM methods (color-coded markers and their convex hull regions) for the subsampled Covertype. Each marker represents a model with a particular hyperparameter choice. For the SVM tree each marker is colored according to tree depth Δ.

As previously discussed in section 4, SVM tree scales gracefully to large scale problems both in terms of memory and computational efficiency. Despite an unoptimized Python implementation, SVM tree scales highly efficiently (Table 1). It is orders of magnitude faster than CascadeSVM and DC-SVM, which require tens of thousands of seconds or timeout entirely on massive datasets like SUSY. It trains as fast or faster than highly optimized RFF and Nyström baselines across most datasets (e.g., 820s vs. 2077s on Covertype). While hSVM is faster, it suffers severe accuracy drops compared to SVM tree (e.g., 38.6% vs. 25.1% error on Poker Hand).

To evaluate inference efficiency, we analyze the rate–distortion trade-off by plotting test error against inference FLOPs (Figure 4) and number of support vectors (Figure 8). Convex hulls over hyperparameter sweeps highlight the optimal configurations for each method.

Across both metrics, SVM tree (red hull) strictly dominates all baselines. The optimal region (bottom-left: low error, low cost) is occupied exclusively by SVM tree models. Exact SVM (blue), DC-SVM (purple), and Cascade SVM (brown) achieve competitive errors (≈19%) but are computationally intensive, requiring $> 10^5$ FLOPs and $> 10^3$ SVs. Cascade SVM performs no localized routing and remains a global model with aggregated SVs. DC-SVM uses kernel clustering for routing, which is expensive and ignores the decision boundary. Cluster splits can intersect optimal hyperplanes, forcing local SVMs to model highly complex overlapping regions, increasing SV count and errors.

hSVM (dark green), Nyström (bright green), and RFF (yellow) span a wide cost range but do not reduce test error below ≈20%. Although hSVM enables faster hierarchical routing than DC-SVM and Cascade SVM, its routing remains suboptimal, leading to poorly partitioned, overlapping subsets of data with noisy local distributions. Kernel approximations require many components to match exact SVM accuracy; without conditional computation, inference cost scales linearly with components, increasing FLOPs and SVs.

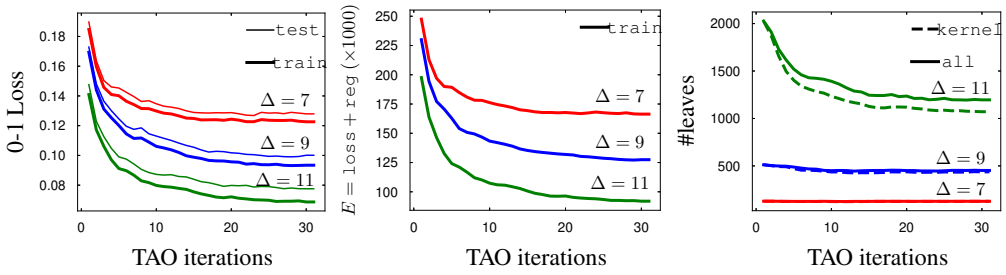

Figure 5: Learning curves of SVM tree on Covertype dataset

SVM tree achieves the best performance (≈17% error, "Best model") while reducing FLOPs by two orders of magnitude and using roughly 30× fewer SVs. Notably, it enables extreme sparsity: models between "Fastest" and "Best" operate with ∼10 SVs (10^2–10^3 FLOPs), while maintaining ≈20% error. This capability to scale down SV count without catastrophic accuracy degradation is not matched by baselines.

5.3 Behavior of the optimization algorithm

We analyze how jointly optimizing routing nodes and leaf models drives the tree toward better local optima (Figures 5 and 6). As shown in Figure 5, training expressive leaf SVMs on the initial random median splits (iteration 1) already provides a reasonable baseline. However, the sharp reduction in both loss and objective value during the first iterations shows that jointly adapting the routing substantially outperforms fixed random partitions. Moreover, the regularized objective E and overall tree 0/1 loss decrease nearly monotonically across iterations (Figure 5 and left panels of Figure 6), demonstrating stable optimization behavior.

In contrast, the average 0/1 loss within individual leaves varies significantly across iterations, while the distributions of support vectors and reduced-set sizes also change considerably (Figure 6). This indicates that the algorithm does not merely refine fixed leaf models, but continuously restructures the routing, altering the data distribution reaching each leaf.

The optimization progressively simplifies the classification problem by isolating increasingly homogeneous subsets of data. As shown in Figure 5, particularly for deeper trees (e.g., $\Delta = 11$), the number of active kernel leaves decreases over training. The model learns partitions that produce pure single-class leaves, substantially reducing inference complexity without sacrificing accuracy.

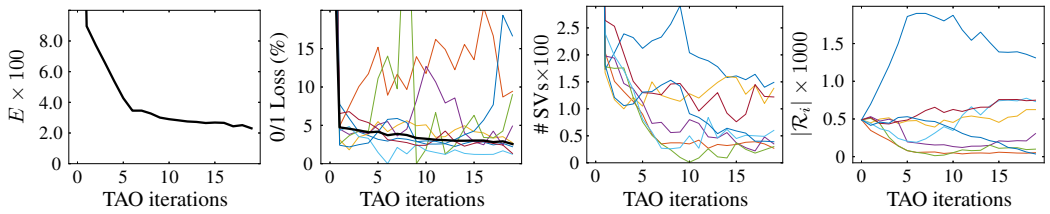

Figure 6: *Left panel:* Learning curve of the whole tree. *3 right panels:* Leaf-specific learning curves: 0/1 loss, # SVs, and reduced set size. Thick black line shows metrics of the whole tree. Tree is trained on the Spambase dataset.

6 Conclusion

We have proposed the SVM tree, a new binary classification model that inherits the generalization ability of SVMs and the efficiency of divide-and-conquer techniques. The SVM tree minimizes the hinge loss over RKHS classifiers at the leaves of a parameterized decision tree, all of which is optimized jointly. Empirically, the SVM tree typically improves over SVMs (trained exactly or using existing fast approximations) both in accuracy and in computational cost in training and inference, and it scales to larger datasets.

AI USE STATEMENT

In this work, we have not used generative AI tools. We take responsibility for the final content of this work, including text, claims or artifacts.

REFERENCES

Ingrid Blaschzyk and Ingo Steinwart. Improved classification rates for localized SVMs. *J. Machine Learning Research*, 23(165):1–59, 2022.

Daniel Boley and Dongwei Cao. Training support vector machine using adaptive clustering. In Michael W. Berry, Umeshwar Dayal, Chandrika Kamath, and David Skillicorn (eds.), *Proc. of the 2004 SIAM Int. Conf. Data Mining (SDM 2004)*, pp. 126–137, Lake Buena Vista, FL, April 22–24 2004.

Miguel Á. Carreira-Perpiñán and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pp. 1211–1221. MIT Press, Cambridge, MA, 2018.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intelligent Systems and Technology*, 2(3):27, April 2011.

Fu Chang and Chan-Cheng Liu. Decision tree as an accelerator for support vector machines. In Xiaoqing Ding (ed.), *Advances in Character Recognition*, chapter 8, pp. 139–154. Intech, 2012.

Jie Chen, Haim Avron, and Vikas Sindhwani. Hierarchically compositional kernels for scalable nonparametric learning. *J. Machine Learning Research*, 18(66):1–42, 2017.

Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of SVMs for very large scale problems. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani (eds.), *Advances in Neural Information Processing Systems (NIPS)*, volume 14, pp. 633–640. MIT Press, Cambridge, MA, 2002.

Naiyang Deng, Yingjie Tian, and Chunhua Zhang. *Support Vector Machines. Optimization Based Theory, Algorithms, and Extensions*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. CRC Publishers, 2012.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *J. Machine Learning Research*, 9:1871–1874, August 2008.

Hatem A. Fayed and Amir F. Atiya. Decision boundary clustering for efficient local SVM. *Applied Soft Computing*, 110:107628:1–17, October 2021.

Hans Peter Graf, Eric Cosatto, León Bottou, Igor Durdanovic, and Vladimir Vapnik. Parallel support vector machines: The cascade SVM. In Lawrence K. Saul, Yair Weiss, and Léon Bottou (eds.), *Advances in Neural Information Processing Systems (NIPS)*, volume 17, pp. 521–528. MIT Press, Cambridge, MA, 2005.

Quanquan Gu and Jiawei Han. Clustered support vector machines. In Arthur Gretton and Christian C. Robert (eds.), *Proc. of the 19th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2016)*, pp. 307–315, Cádiz, Spain, May 9–11 2016.

Trevor J. Hastie, Robert J. Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning—Data Mining, Inference and Prediction*. Springer Series in Statistics. Springer-Verlag, second edition, 2009.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *Annals of Statistics*, 36(3):1171–1220, 2008.

Cho-Jui Hsieh, Si Si, and Inderjit Dhillon. A divide-and-conquer solver for kernel support vector machines. In Eric P. Xing and Tony Jebara (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 566–574, Bejing, China, 22–24 Jun 2014a. PMLR. URL https://proceedings.mlr.press/v32/hsieha14.html.

Cho-Jui Hsieh, Si Si, and Inderjit Dhillon. A divide-and-conquer solver for kernel support vector machines. In Eric P. Xing and Tony Jebara (eds.), *Proc. of the 31st Int. Conf. Machine Learning (ICML 2014)*, pp. 566–574, Beijing, China, June 21–26 2014b.

Hannes Köhler. Lp- and risk consistency of localized SVMs. *Neurocomputing*, 598:128060:1–13, September 14 2024.

Mauricio Kugler, Susumu Kuroyanagi, Anto Satriyo Nugroho, and Akira Iwata. CombNET-III: A support vector machine based large scale classifier with probabilistic framework. *IEICE Trans. Inf. & Syst. D*, E89-D(9):2533–2541, September 2006.

Jianjiang Li, Jinliang Shi, Zhiguo Liu, and Can Feng. A parallel and balanced SVM algorithm on spark for data-intensive computing. *Intelligent Data Analysis*, 27(4):1065–1086, July 2023.

F. Marchetti and E. Perracchione. Local-to-global support vector machines (LGSVMs). *Pattern Recognition*, 132:108920:1–11, December 2022.

Dmitry Pavlov, Darya Chudova, and Padhraic Smyth. Towards scalable support vector machines using squashing. In *Proc. of the 6th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2000)*, pp. 295–299, Boston, MA, August 20–23 2000.

Ali Rahimi and Ben Recht. Random features for large-scale kernel machines. In John C. Platt, Daphne Koller, Yoram Singer, and Sam Roweis (eds.), *Advances in Neural Information Processing Systems (NIPS)*, volume 20, pp. 1177–1184. MIT Press, Cambridge, MA, 2008.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels. Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA, 2001.

Yunsheng Song, Jiye Liang, and Feng Wang. An accelerator for support vector machines based on the local geometrical information and data partition. *Int. J. Machine Learning and Cybernetics*, 10(9):2389–2400, September 2019.

Philipp Thomann, Ingrid Blaschzyk, Mona Meister, and Ingo Steinwart. Spatial decompositions for large scale SVMs. In Aarti Singh and Jerry Zhu (eds.), *Proc. of the 20th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2017)*, pp. 1329–1337, Fort Lauderdale, FL, April 20–22 2017.

Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. Core vector machines: Fast SVM training on very large data sets. *J. Machine Learning Research*, 6(13):363–392, 2005.

Christopher K. I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In Todd K. Leen, Tom G. Dietterich, and Volker Tresp (eds.), *Advances in Neural Information Processing Systems (NIPS)*, volume 13, pp. 682–688. MIT Press, Cambridge, MA, 2001.

Hwanjo Yu, Jiong Yang, Jiawei Han, and Xiaolei Li. Making SVMs scalable to large data sets using hierarchical cluster indexing. *Data Mining and Knowledge Discovery*, 11(3):295–321, November 2005.

Jingjing Zeng, Yimo Qin, Bin Zou, Jie Xu, and Peipei Dong. Hybrid learning framework integrating Fisher linear discriminant and localized support vector machines. *Expert Systems with Applications*, 299 Part A:129990:1–12, March 1 2026.

Kai Zhang, Ivor W. Tsang, and James T. Kwok. Improved Nyström low-rank approximation and error analysis. In Andrew McCallum and Sam Roweis (eds.), *Proc. of the 25th Int. Conf. Machine Learning (ICML'08)*, pp. 1232–1239, Helsinki, Finland, July 5–9 2008.

Yuchen Zhang, John Duchi, and Martin Wainwright. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *J. Machine Learning Research*, 16(102): 3299–3340, 2015.

A TECHNICAL APPENDICES AND SUPPLEMENTARY MATERIAL

B REDUCED PROBLEM AT A DECISION NODE

Optimizing the objective function over a single decision node $i \in \mathcal{D}$ while fixing the rest of the tree relies on isolating the subset of training instances that reach i, denoted as the reduced set $\mathcal{R}_i$. For instances not in $\mathcal{R}_i$, the routing and subsequent loss are independent of the parameters $(\mathbf{w}_i, w_{i0})$. Thus, the global objective reduces to minimizing the loss over $\mathcal{R}_i$: $E(\mathbf{w}_i, w_{i0}) = \sum_{n\in\mathcal{R}_i} L(y_n, T(\mathbf{x}_n; \mathbf{w}_i, w_{i0})) + \lambda\phi_i(\mathbf{w}_i)$. Because the rest of the tree is fixed, the final prediction $T(\mathbf{x}_n)$ for any $\mathbf{x}_n \in \mathcal{R}_i$ depends strictly on whether the instance is routed to the left or right child.

Let $L_{n,\text{right}}$ and $L_{n,\text{left}}$ denote the respective losses incurred if $\mathbf{x}_n \in \mathcal{R}_i$ is deterministically routed to the right or left subtree and evaluated at the corresponding fixed leaf SVMs. We assign a pseudolabel $z_n \in \{0, 1\}$ indicating the routing direction that minimizes the loss, such that $z_n = 1$ if $L_{n,\text{right}} \leq L_{n,\text{left}}$, and 0 otherwise. Defining the instance weight as the absolute difference in loss, $c_n = |L_{n,\text{right}} - L_{n,\text{left}}|$, we can rewrite the routing loss using the 0/1 loss function as:

$$L(y_n, T(\mathbf{x}_n; \mathbf{w}_i, w_{i0})) = z_n L_{n,\text{right}} + (1 - z_n)L_{n,\text{left}} + c_n \mathbb{I}\Big(z_n \neq \mathbb{I}(\mathbf{w}_i^T \mathbf{x}_n + w_{i0} > 0)\Big)$$

Dropping the constant term, the reduced problem is exactly equivalent to learning a linear classifier on the weighted dataset $\{(\mathbf{x}_n, z_n, c_n)\}_{n\in\mathcal{R}_i}$:

$$\min_{\mathbf{w}_i, w_{i0}} \sum_{n\in\mathcal{R}_i} c_n \mathbb{I}\Big(z_n \neq \mathbb{I}(\mathbf{w}_i^T \mathbf{x}_n + w_{i0} > 0)\Big) + \lambda\phi_i(\mathbf{w}_i) \tag{2}$$

Because optimizing the 0/1 loss is NP-hard, we use ℓ_1-regularized logistic regression as a a convex surrogate approximation with sample weights c_n and binary labels z_n: $\min_{\mathbf{w}_i, w_{i0}} \sum_{n\in\mathcal{R}_i} c_n \log(1 + \exp(-z_n(\mathbf{w}_i^T \mathbf{x}_n + w_{i0}))) + \lambda\|\mathbf{w}_i\|_1$, and solve it using LIBLINEAR (Fan et al., 2008).

C PROOFS

Proof of theorem 3.1.

Proof. Minimizing the SVM tree objective function (1) over g_j given that all other parameters in the tree are fixed reduces to the following:

$$\min_{g_j\in\mathcal{H}} \sum_{n\in\mathcal{R}_j} L(y_n, g_j(\mathbf{x}_n)) + \mu\|g_j\|_{\mathcal{H}}^2. \tag{3}$$

Note how the hinge loss sum is over the reduced set $\mathcal{R}_j$ only (rather than over the whole training set), and how we have dropped other terms that are additive constants (regularization terms ϕ_i over the decision nodes and $\|\cdot\|_{\mathcal{H}}^2$ over the other leaves).

First, let's consider the case where $\mu \to +\infty$. Let $g_j(\mathbf{x}) = f_j(\mathbf{x}) + b_j$, where $f_j \in \mathcal{H}$ is the regularized functional component and $b_j \in \mathbb{R}$ is the unregularized bias term. Let (f_j^*, b_j^*) be a minimizer of (3). This implies:

$$\mu\big\|f_j^*\big\|_{\mathcal{H}}^2 \leq \sum_{n\in\mathcal{R}_j} L(y_n, f_j^*(\mathbf{x}_n) + b_j^*) + \mu\big\|f_j^*\big\|_{\mathcal{H}}^2 \leq |\mathcal{R}_j|$$

$$\big\|f_j^*\big\|_{\mathcal{H}}^2 \leq \frac{|\mathcal{R}_j|}{\mu}$$

Taking the limit as $\mu \to +\infty$:

$$\lim_{\mu\to+\infty} \big\|f_j^*\big\|_{\mathcal{H}}^2 = 0$$

By the reproducing property of the RKHS, evaluation functionals are bounded, meaning $|f_j(\mathbf{x})| \leq \|f_j\|_{\mathcal{H}}\sqrt{k(\mathbf{x}, \mathbf{x})}$. Therefore, $\big\|f_j^*\big\|_{\mathcal{H}} \to 0 \implies f_j^*(\mathbf{x}) \to 0$ for all $\mathbf{x} \in \mathcal{R}_j$.

Let N_+ and N_- denote the cardinalities of the positive and negative samples in $\mathcal{R}_j$. Since $g_j(\mathbf{x})$ simplifies to $g_j(\mathbf{x}) = b_j$, we can rewrite optimal scalar bias as:

$$b_j^* = \arg\min_{b\in\mathbb{R}} \sum_{n\in\mathcal{R}_j} \max(0, 1 - y_n b) = N_+ \max(0, 1 - b) + N_- \max(0, 1 + b)$$

The solution is:

$$b_j^* = \begin{cases} +1 & \text{if } N_+ > N_- \\ -1 & \text{if } N_+ < N_- \\ c \in [-1, +1] & \text{if } N_+ = N_- \end{cases} \tag{4}$$

For general case where $\mu > 0$, we consider first case 3, where the set $\mathcal{R}_j$ is not empty and it contains instances of both classes. As is well known (Schölkopf & Smola, 2001; Deng et al., 2012; Hastie et al., 2009; Hofmann et al., 2008), the solution g_j^* of (3) has the form

$$g_j^*(\mathbf{x}) = \sum_{n\in\mathcal{R}'_j} \alpha_{jn} y_n K(\mathbf{x}, \mathbf{x}_n) \tag{5}$$

for certain coefficients $\{\alpha_{jn}\} \subset \mathbb{R}$ and where $\mathcal{R}'_j \subseteq \mathcal{R}_j$ are the support vectors (SVs) of leaf j. Numerically, the solution for the coefficients can be obtained with different types of algorithms, operating either directly on the hinge-loss formulation, or on the dual quadratic program that is equivalent to it.

There are two special cases. Firstly, in case 1 we have $\mathcal{R}_j = \varnothing$. Then the objective (3) becomes

$$\min_{g_j\in\mathcal{H}} \|g_j\|_{\mathcal{H}}^2 \tag{6}$$

and so any function in $\arg\min_f \|f\|_{\mathcal{H}}^2$ is a solution; typically this always contains the identically-zero function $f(\mathbf{x}) = 0\ \forall \mathbf{x} \in \mathbb{R}^D$.

Secondly, in case 2 we have w.l.o.g. $y_n = +1\ \forall n \in \mathcal{R}_j$. Then the objective (3) becomes

$$\min_{f_j\in\mathcal{H}, b_j\in\mathbb{R}} \sum_{n\in\mathcal{R}_j} \max(0, 1 - (f_j(\mathbf{x}_n) + b_j)) + \mu\|f_j\|_{\mathcal{H}}^2$$

which becomes $\min_{f_j\in\mathcal{H}, b_j\in\mathbb{R}} \sum_{n\in\mathcal{R}_j} \max(0, 1 - (f_j(\mathbf{x}_n) + b_j))$, with the solution $f_j^*(\mathbf{x}) = 0$ $\forall \mathbf{x} \in \mathbb{R}^D$ and $b_j^* = +1$ (the opposite happens if all labels are negative). This is consistent with the limit case of infinite regularization in (4). □

C.1 Setup

All algorithms and experiments were implemented in Python 3.11.11. Depedning on experiments, computational evaluations were conducted on multiple machines: first machine equipped with an Intel Xeon E5-2699 v3 CPU (2.30 GHz) and 256 GB of RAM, second one Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz and 24 GB of RAM. To ensure fair runtime evaluations and exact extraction of support vectors during inference, we utilized both standard libraries and custom implementations. Hyperparameter search spaces were adapted according to the scale of each dataset.

- **SVM tree:** Trees were initialized with a complete structure using random median splits. The hyperparameter grid included $\lambda \in \{1, 10\}$, $C \in \{0.1, 1, 10, 100\}$, and $\gamma \in \{0.001, 0.01, 0.1, 1, 10\}$, where the scale parameter is defined.
- **Standard SVM:** We utilized the scikit-learn implementation. We performed a grid search over $C \in [10^{-3}, 10^2]$ and $\gamma \in [10^{-3}, 10^5]$. The granularity of the grid (ranging from discrete selections to 10–20 intervals) was scaled based on dataset size.
- **DC-SVM, CascadeSVM:** We use Python implementation of CascadeSVM and we developed custom Python implementation of DC-SVM ("early" version from (Hsieh et al., 2014a)) to guarantee a controlled, one-to-one runtime comparison and to extract intermediate support vectors. Implementation of both DC-SVM and CascadeSVM was based on the

scikit-learn framework. The search space contains $C \in [10^{-2}, 10^{2}]$ and $\gamma \in [10^{-4}, 10^{2}]$. Algorithm-specific parameters included the maximum number of hierarchy levels (up to 3) and partitioning criteria (fold size for CascadeSVM, and number of clusters for DC-SVM). It is worth noting that both algorithms require thorough hyperparameter tuning to achieve optimal performance, and their sensitivity to these parameters can significantly impact runtime and accuracy, furthermore, the optimal settings can vary widely across different datasets.

- **Kernel Approximations (Nyström & RFF):** The C and γ ranges are the same as those of the Standard SVM. The maximum number of components was tuned proportionally to dataset complexity (e.g., up to 300 components for SUSY and 4000 for Covertype).
- **Hierarchical SVM:** Oblique node splits were generated via random projections, hierarchical K-Means, or random median splits. The latter ensured equal reduced set sizes to balance training. Hyperparameters for the leaf-node SVMs were optimized over a grid identical to that of SVM tree.

C.2 Interpretability of the SVM tree

We demonstrate the interpretability of our model by visualizing a decision tree trained on a subsampled FashionMNIST dataset. The dataset was separated into 2 classes: winter clothing (class 0: Pullover, Ankle boot, Coat) and summer clothing (class 1: T-shirt/top, Shirt, Sandal). Figure 7 reveals some interesting insights into classification process.

Global level structure. The final tree is of complete structure with depth 3, and train accuracy 99.73%. The weights of the decision nodes are of high sparsity between 35 ($\approx 5\%$) and 122 ($\approx 16\%$) non-zero elements. The tree effectively performs a semantic, hierarchical clustering of the dataset before applying complex non-linear boundaries. Rather than attempting to separate all winter items from all summer items simultaneously, the early routing decisions separate the data into broad morphological categories (e.g., footwear vs. upper-body garments). This drastically simplifies the classification problem for the deeper nodes and the leaves (keeps the number of SVs low), allowing them to act as highly specialized local experts on visually homogeneous subsets of the data.

Local level structure. By inspecting the sparse weight visualizations (28×28 matrices where red denotes positive weights, blue negative, and white zero), we can directly interpret the features constructed by decision node optimization. These splits behave as meaningful, highly localized feature detectors. **Node 0 (Root)** acts as a primary filter to separate footwear from upper-body garments. The blue (negative) weights clearly outline the silhouette of a boot or high-ankle shoe. Consequently, the left branch receives a reduced set heavily skewed toward footwear, while the right branch processes mostly upper-body garments. **Node 1** refines the initial separation by checking for pixel activations in the top-left corner (circled in blue), effectively pushing remaining garments to the left and isolating footwear to the right (uses only $\approx 9\%$ of weights). This split is of high accuracy, as Node 2 does not contain any shoe samples. Deeper in this branch, **node 3 (footwear branch)** distinguishes between different shoe types using only $\approx 5\%$ of weights. Its weights focus on the heel and upper ankle regions (circled in red), effectively routing such shoes to the right (Leaf D), while shoes with a protruding lacing area go left (Leaf C).

For upper-body items, the tree constructs spatial filters (**node 2, 4, 5 & 6 (garment branch)**) corresponding to specific garment components. Node 4 performs the split into 2 clusters of garment items with different intensity (images in the right subtree are generally darker). Node 2 focuses on the bottom edge (extended hems), which separates T-shirts/tops and shirts almost entirely to the right leaf node. Both Node 5 and 6 use longer sleeves which predominantly performs impure linear separation, where samples of T-shirts/tops are separated almost entirely to the left leaf, while only partially separating the other.

In **leaf node**, we choose to demonstrate a random support vector with the largest weight in the leaves, since mean of a high dimensional non-convex dataset leads to a poor representation. Figure 9 illustrates actual support vector examples from leaf support vectors. **Leaves C & D** are both specialized on footwear. Because the tree successfully isolated specific shoe types, the SVMs face a much simpler task. In Leaf D, the SVM is specifically tasked with the hard margin between winter high-heeled boots (class 0) and summer high-heeled sandals (class 1). The support vectors for this leaf

show a tight semantic cluster of heeled footwear. **Leaves F & H** contain visually similar upper-body garments. The SVM in Leaf F handles the nuanced boundary between winter coats/pullovers (class 0) and long-sleeve summer shirts (class 1). The support vectors in Figure 9 highlight the thickness of the torso and the cut of the collar—that the Kernel SVM relies upon to separate these garments.

C.3 Effect of hyperparameters

The joint optimization exposes a clear spectrum between structural complexity and leaf-level functional complexity. A deep tree relies entirely on its hierarchical routing (high Δ, pure leaves, 0% kernel usage), while a shallow tree relies on the more powerful non-linear model of its leaf nodes (low Δ, 100% kernel usage, high support vectors). As Δ increases, the tree partitions the input space so finely that the leaves become pure (or contain too few samples) so classification does not require complex decision boundaries.

The results in Figure 10 demonstrate that a hybrid approach, where a shallow oblique tree ($\Delta \approx 4$) combined with well-regularized kernel SVM leaves yields the best generalization (test error of $\approx 17\%$). Furthermore, Figure 4 and Figure 8 demonstrate using depth gradient overlaid on the SVM tree points the architectural mechanism controlling this trade-off. Darker points (deeper trees) cluster in the lower FLOPs and SVs region (low train error, high test error), while shallow trees (light points) cluster close to SVM (highest FLOPs and SVs). The models with the best test error (annotated as "Best model") are of intermediate depth, demonstrating the importance of balancing routing and leaf complexity to achieve optimal generalization.

At high C (upper rows), expressive leaf models produce distinct losses between child subtrees, generating numerous carepoints that preserve decision splits. Conversely, low C underfits leaf models, resulting in similar child losses. In deeper nodes, the shrinking reduced set size artificially inflates regularization, causing redundant splits to be pruned and yielding a sparser tree.

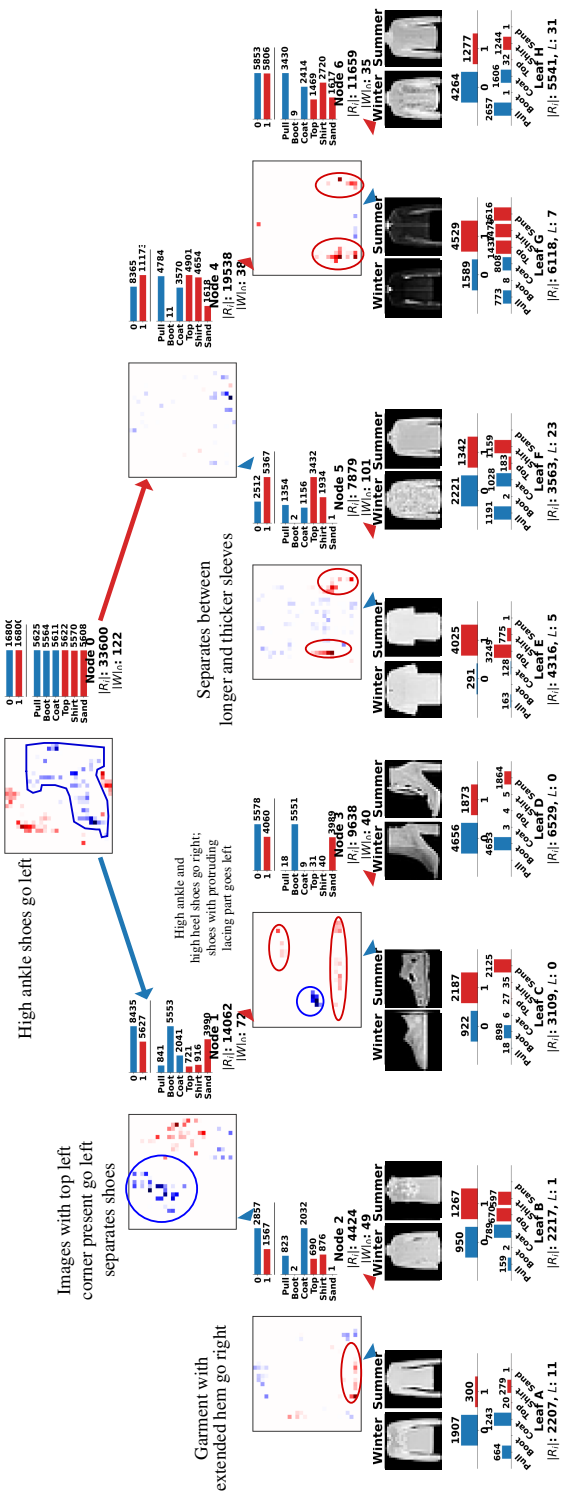

Figure 7: Visual representation of a decision tree trained on subsampled FashionMNIST (Winter cloth vs. Summer cloth). For each node we demonstrate the RS size ($|\mathcal{R}_i|$). Internal decision nodes display # non-zero weights ($\|\mathbf{w}\|_0$) and weight visualizations (red for positive, blue for negative, and white for zero). Leaf nodes depict a random sample of support vectors with highest weights for the two predicted classes and # misclassifications (L). Local class distributions within the reduced dataset are illustrated via bar charts at each node.

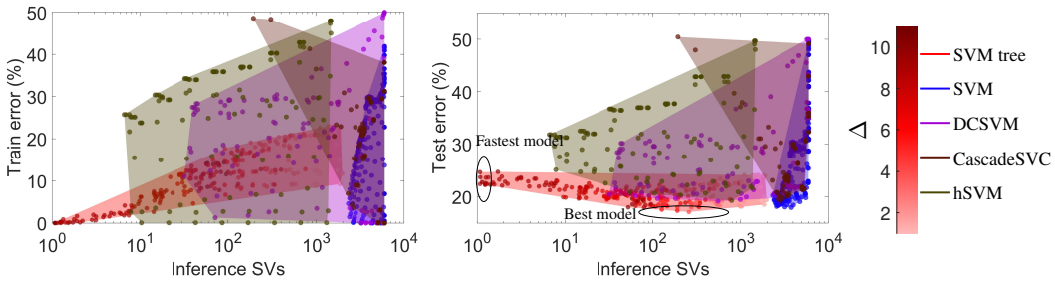

Figure 8: 0/1 Loss (training, test) vs inference SVs for various SVM methods (color-coded markers and their convex hull regions) for the subsampled Covertype. Each marker represents a model with a particular hyperparameter choice. For the SVM tree each marker is colored according to tree depth Δ. Similar to Figure 4.

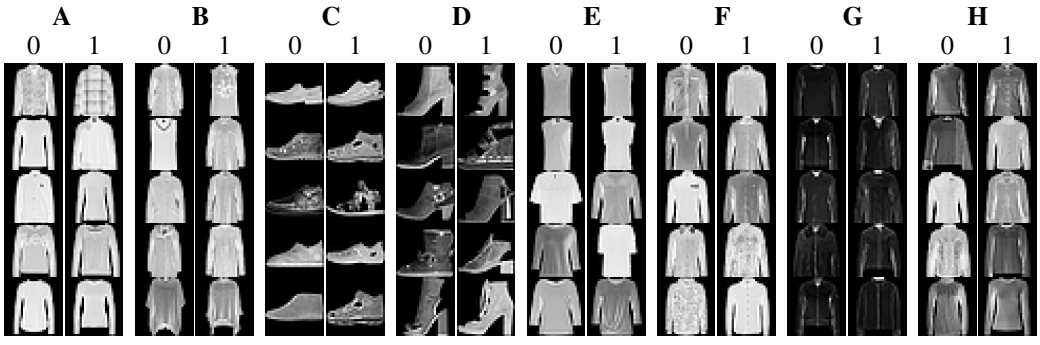

Figure 9: Support vector examples of reduced sets from the leaf nodes in Figure 7. Colors denote different classes, illustrating the class separation achieved within each leaf's subset.

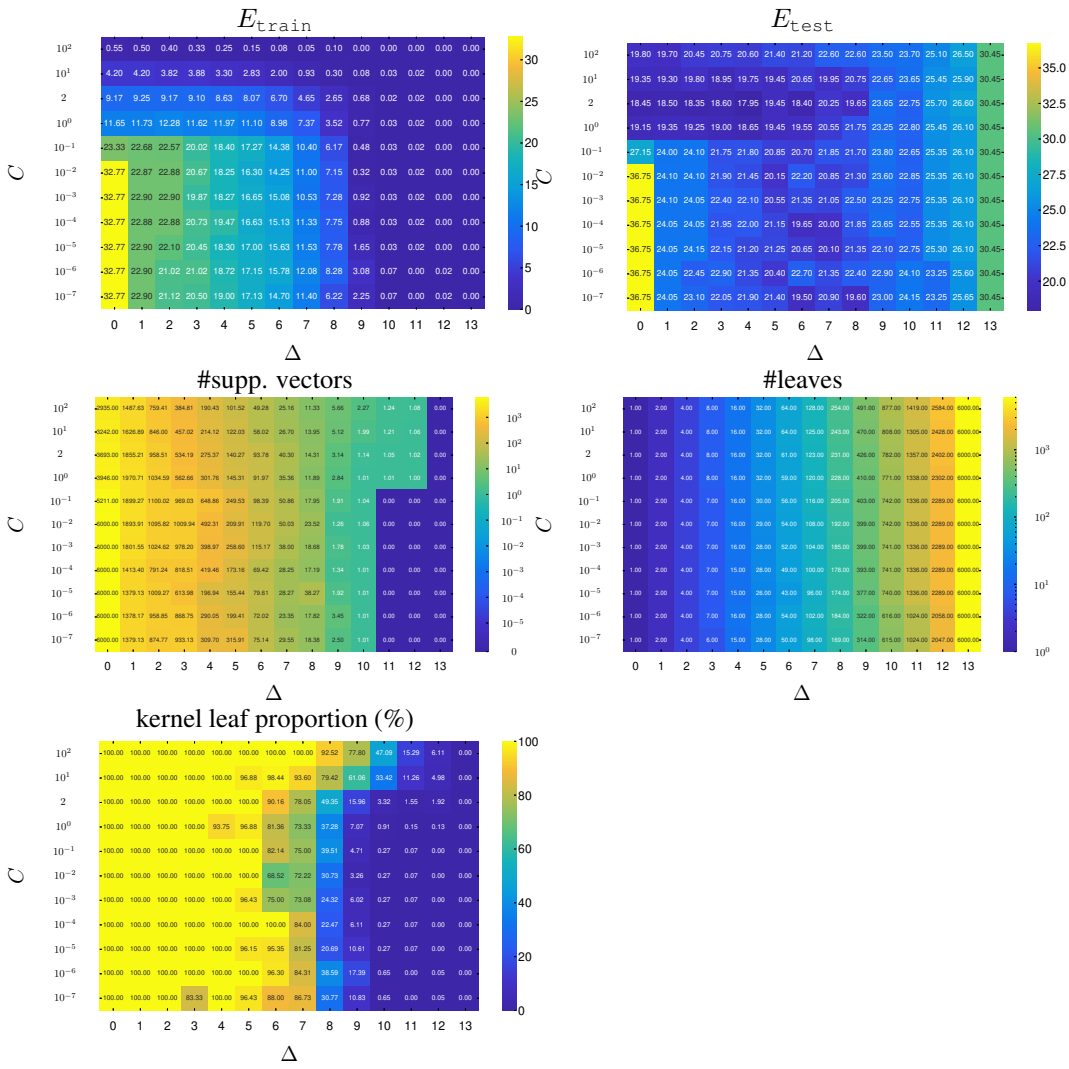

Figure 10: Regularization paths computed on Covertype dataset (subsampled to 6,000 instances). To isolate the effect of the regularization parameter C for a given tree depth Δ, γ and λ are held constant. The sparsity penalty, λ, is set to a near-zero value to eliminate sparsity constraints during the parameter sweep.