Pattern Recognition 45.12 (2012): 4414 – 4427

DOI: http://dx.doi.org/10.1016/j.patcog.2012.06.002

Hierarchical Linear Support Vector Machine

I. Rodriguez-Lujan[a,*], C. Santa Cruz[a], R. Huerta[b]

[a] *Departamento de Ingeniería Informática and Instituto de Ingeniería del Conocimiento, Universidad Autónoma de Madrid, 28049 Madrid, Spain.*
[b] *BioCircuits Institute, University of California San Diego, La Jolla CA 92093-0404, USA.*

Abstract

The increasing size and dimensionality of real-world datasets make it necessary to design efficient algorithms not only in the training process but also in the prediction phase. In applications such as credit card fraud detection, the classifier needs to predict an event in 10 milliseconds at most. In these environments the speed of the prediction constraints heavily outweighs the training costs. We propose a new classification method, called a Hierarchical Linear Support Vector Machine (H-LSVM), based on the construction of an oblique decision tree in which the node split is obtained as a Linear Support Vector Machine. Although other methods have been proposed to break the data space down in subregions to speed up Support Vector Machines, the H-LSVM algorithm represents a very simple and efficient model in training but mainly in prediction for large-scale datasets. Only a few hyperplanes need to be evaluated in the prediction step, no kernel computation is required and the tree structure makes parallelization possible. In experiments with medium and large datasets, the H-LSVM reduces the prediction cost considerably while achieving classification results closer to the non-linear SVM than that of the linear case.

Keywords: Large-Scale Learning, Real-Time Prediction, Support Vector Machine, Decision Tree, Pegasos Algorithm

1. Introduction

Support Vector Machines (SVMs) have been widely used in classification problems as a result of their effectiveness. However, the increasing size of real-world datasets in domains such as bioinformatics, document categorization or credit card fraud detection compromises their application. The computational complexity of the SVM decision function scales with respect to the number of support vectors n_{SV} and Steinwart [1] showed that the number of support vectors scales linearly with respect to the number of training patterns. Consequently, other machine learning techniques are preferred in those large-scale domains in which an

*Corresponding author. Instituto de Ingeniería delConocimiento; C/Francisco Tomás y Valiente,11; E.P.S., EdificioB; UAM-Cantoblanco, 28049Madrid,Spain. Tel: +34 91 497 2339; fax: +34 91 497 2334
Email addresses: irene.rodriguez@iic.uam.es (I. Rodriguez-Lujan), carlos.santacruz@iic.uam.es (C. Santa Cruz), rhuerta@ucsd.edu (R. Huerta)

efficient prediction step is needed, especially in real-time applications such as credit card fraud detection which requires a response time of less than 10 milliseconds.

Although the machine learning community has been mainly focused on speeding up the training of the SVM, the emergence of applications requiring fast classification makes the design of new algorithms necessary whilst maintaining as much as possible the effectiveness of non-linear SVMs and improving its classification complexity at the same time. Linear SVMs are the best alternative for fast execution because their decision boundary is made up of a single hyperplane. However, their performance for non-linear problems is uncompetitive and a compromise between performance and classification speed is needed.

The computational complexity of testing a pattern using a non-linear SVM is $O(n_{SV} \times d \times n_K)$ where n_{SV} are the number of support vectors, d is the dimension of the samples and n_K is the cost of evaluating the kernel function. In large-scale problems, the number of support vectors is usually much higher than the dimension of the problem ($n_{SV} \gg d$) which is why almost all methods proposed in the literature aim at reducing n_{SV}. The methods for reducing the number of support vectors can be divided into two groups [2]:

- **Numerical techniques** find a reduced set of basis functions necessary to classify a pattern. These algorithms usually consider all of the training patterns and find a sparse representation of the support vectors. A more detailed overview of these methods is given by Keerthi et al. [2]. According to Keerthi's categorization, the support vector reduction can be carried out as a post-processing phase after training the SVM model or during the training phase thus imposing a certain sparsity in the basis functions. The post-processing techniques ([3, 4]) reduce the number of support vectors once the SVM model has been trained. Therefore, they still depend on the standard SVM training which can be extremely costly in large-scale problems. Among the direct simplification approaches based on imposing sparseness on the basis functions in the primal space, several methods can be found in the literature [5, 6, 2]. These approaches considerably reduce the SVM prediction cost while having a competitive classification accuracy, but in some datasets the number of basis functions needed to maintain a competitive classification accuracy is still high for efficient training and prediction phases [2].

- **Data-reduction methods** reduce the number of SVM training patterns dividing the original training set into one or several smaller datasets to train an SVM in each partition. Boosting [7], bagging [8], parallel mixture of SVMs [9] and SVM-cascade [10] algorithms can be categorized into this group. A recent work [11] proposes the DTSVM (Decision Tree Support Vector Machine) algorithm to build a decision tree with axis-parallel splits via the CART method [12] and to train an SVM with an RBF kernel in each leaf of the tree. This method reports a significant reduction of the number of support vector evaluations in the test or prediction phase. However, the number is still too high for large-scale datasets at the level used in credit card fraud detection.

Our approach does not consider the use of non-linear SVMs because of their high classification and training cost. The aim of our work is to provide a model which generates non-linear decision boundaries via piecewise linear decision functions. This approach is motivated by the low classification cost of linear SVMs. Moreover, recent algorithms [13, 14] have shown the efficiency of stochastic gradient descent approaches for training linear SVMs and their usual fast convergence for large-scale datasets. Our work is not the first attempt at approximating non-linear SVMs through the linear case. Recent contributions have proposed the use of linear SVMs in the manifold coordinates such as sparse coding or local coordinate coding [15, 16, 17]. The MLSVM method [18] is based on a mixture of linear SVMs defining an underlying probabilistic model which implicitly selects the linear SVMs to be used to classify each pattern. A test sample is classified by the weighted average over the mixture of classifiers.

Our work approaches the task as the construction of a binary decision tree whose nodes are linear SVMs. The combination of linear SVMs and decision trees is motivated by the results of Bennett et al. [19] and some research combining decision trees and SVMs. An interesting comparison of the classification cost of decision trees and SVMs is given by Kumar and Gopal [20]. Basically, decision trees are much faster than SVMs in classifying new instances whereas the classification accuracy of SVMs is superior. Pursuing the objective of speeding up the prediction phase of a classifier, Zapién et al. [21] proposes a tree structure where the split of each node is a linear SVM. The tree presents a particular structure, which could be considered as a cascade of linear SVMs as the tree only expands the right branches. Then, it is assumed that each split in the tree is able to classify correctly all of the patterns belonging to the left child. The main difference with our method is that our tree is a *complete* binary decision tree in the sense that both children of each node can be expanded in the following steps. Although Zapién et al. provide the most straightforward approach, a balanced tree search is on average faster at classifying a datapoint since the cascade structure needs to run through all of the decision nodes to evaluate the worst datapoints. In addition, the hypothesis class (disjunctions of conjunctions) of H-LSVM is more general than that of the Zapién's model (conjunctions) because (i) the cascade structure (also known as decision list) can be viewed as a special type of decision trees [22] and (ii) the number of decision tree skeletons with k decision nodes is given by the k-th Catalan Number [19, 23] while the Zapien's cascade structure has only one possibility. The algorithms proposed by Fehr et al. [24] and Sun et al. [25] represent an extension of the Zapién model in which the linear SVM is the split in each node and nonlinear SVMs make up the leaves of the tree. These models still depend on a non-linear SVM which means a large number of support vector evaluations to classify a test sample. The DTO-SVM algorithm [26] builds an oblique decision tree whose node split is selected between the C4.5 [27] parallel-split calculated from the categorical variables and the SVM-SMO [28] classifier obtained from continuous attributes. The method still depends on the large number of support vectors given by the SMO which makes large-scale predictions costly.

Another interesting approach to combine decision trees and SVMs is the one proposed by Bennett and

Blue [29] in which the decision tree structure is set beforehand so that the model is formulated in the primal space as the minimization of a nonconvex objective function with respect to polyhedral constraints. This alternative is substantially different from the aforementioned ones and that adopted in this paper since they obtain the structure of the tree in an on-line manner. In addition, the margin is locally maximized in each node of the H-LSVM tree whereas the Bennett and Blue model looks for a global maximization.

The main advantage of our H-LSVM is its ability to classify a pattern in a few milliseconds even for large-scale datasets thus speeding up the prediction phase of the SVMs by several orders of magnitude while maintaining a classification accuracy close of that of the non-linear SVMs. Moreover, the decision tree structure is easily parallelizable which favors training acceleration [30]. As H-LSVM is a piecewise linear classifier, its classification accuracy is not as good as those models based on non-linear SVMs. However, in those systems which do require real-time predictions, the reduction in accuracy is bearable when compared to the runtime savings. As regards other combinations of decision trees and linear SVMs models, the proposed method represents an improvement in the state-of-the-art not only in terms of classification accuracy but also in terms of prediction cost.

The paper is organized as follows: Section 2 presents the H-LSVM algorithm including the explanation of several design aspects. Section 3 analyzes and compares the training and prediction complexities of linear SVMs, non-linear SVMs and H-LSVM. Section 4 provides a generalization error bound for the H-LSVM method and Section 5 presents the empirical results in terms of classification accuracy and prediction cost of the proposed method compared to linear and non-linear SVMs and other algorithms based on the speeding up of SVMs via linear SVMs. A numerical analysis of the H-LSVM scalability and generalization error bound are also given in this section.

2. The H-LSVM Algorithm

The proposed algorithm called a Hierarchical Linear Support Vector Machine (H-LSVM) is based on the construction of a decision tree. As described by Breiman et al. [12], four elements must be considered in the construction:

1. The goodness of the node split which needs to be evaluated in each node of the tree.
2. The type of test carried out in each node of the tree to decide whether a pattern belongs to the left or to the right child of the current node.
3. The stop-splitting rule.
4. The criteria for assigning the class label to a pattern when it reaches a leaf of the tree.

The H-LSVM algorithm node split is a linear SVM. The linear SVM is trained using the Pegasos algorithm [13] with weighted patterns. The weight of each pattern is not fixed and it depends on which node of the tree we are working on. For the rest of the elements, well-known techniques and criteria have been used.

Once the complete tree is trained, a pruning step can improve the generalization capability of the H-LSVM model. The four key elements for the construction of a decision tree with the pruning algorithm will be described in this section.

Let us establish some notation. Given a training set $S = \{(\vec{x_i}, y_i)\}_{i=1}^{N}$, where $\vec{x_i} \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$, we define:

- H_k: a node in the tree.
- S_{H_k}: subset of samples in the node H_k.
- $S_{H_k}^+$: subset of positive samples in the node H_k.
- $S_{H_k}^-$: subset of negative samples in the node H_k.
- $N_{H_k} = |S_{H_k}|$: number of samples in the node H_k.
- $N_{H_k}^+ = |S_{H_k}^+|$: number of positive samples in the node H_k.
- $N_{H_k}^- = |S_{H_k}^-|$: number of negative samples in the node H_k.
- $\vec{x_i}^{H_k}$: i-th sample in the subset S_{H_k}.
- $\vec{w}_{H_k}$: normal vector to the hyperplane associated to the node H_k.
- b_{H_k}: bias term of the hyperplane associated to the node H_k.
- $h_{H_k}(\vec{x_i})$: evaluation of the i-th pattern in the node H_k, that is $h_{H_k}(\vec{x_i}) = \vec{w}_{H_k} \cdot \vec{x_i} + b_{H_k}$.
- $S_{H_k}^l$: left child of the node H_k: $\{\vec{x} \in S_{H_k} \mid h_{H_k}(\vec{x}) \leq 0\}$.
- $S_{H_k}^r$: right child of the node H_k: $\{\vec{x} \in S_{H_k} \mid h_{H_k}(\vec{x}) > 0\}$.
- ${p_i}^{H_k}$: weight of the i-th pattern in the node H_k verifying $\sum_{i=1}^{N_{H_k}} p_i^{H_k} = 1 \; \forall k$.

Splitting Goodness. The definition of the splitting goodness is based on the impurity function concept [12]. Two different concepts need to be defined: the impurity of a node and the impurity of a split. The impurity of a node H_k, $I(H_k)$, does not depend on the splits and it is a function of the number of patterns of each class in the node, $I(H_k) = I(N_{H_k}^+, N_{H_k}^-)$. The impurity of a split is the impurity induced by the node split which divides the samples into the subsets $S_{H_k}^l$ and $S_{H_k}^r$. The impurity of a split given by $\vec{w}_{H_k}$ and b_{H_k}, $I(\vec{w}_{H_k}, b_{H_k})$, can be defined straightforwardly from the impurity of the children, $I(H_k^l)$ and $I(H_k^r)$, as follows,

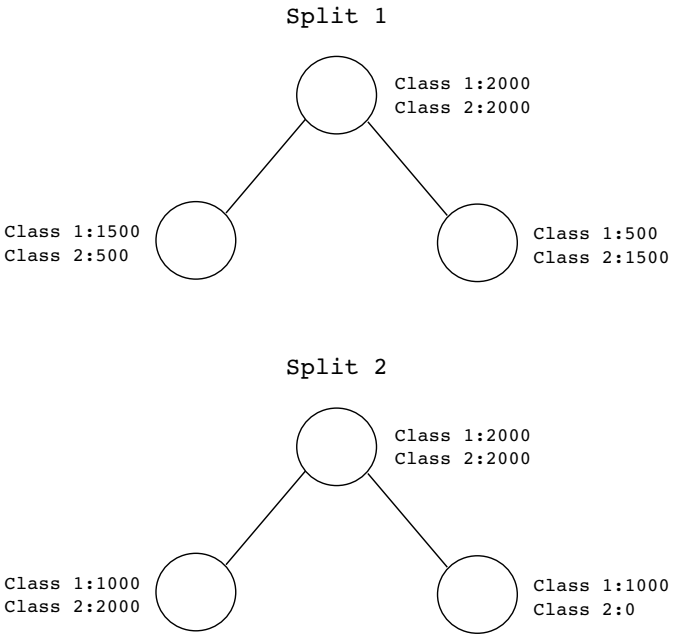


Figure 1: An example of two different splits in a decision tree. If the classification error is used as an impurity measure, both splits misclassified $1,000$ samples. Nevertheless, the second split seems more desirable for the future expansion of the tree.

$$I(\vec{w}_{H_k}, b_{H_k}) = \frac{|S^l_{H_k}|}{|S_{H_k}|}\, I(H^l_k) + \frac{|S^r_{H_k}|}{|S_{H_k}|}\, I(H^r_k). \tag{1}$$

As the aim of the decision tree is to minimize the overall misclassification rate of the tree, it would be natural to choose the classification error as an impurity measure. However, as pointed out by Breiman et al. [12, Chapter 4], this measure has two significant limitations: i) The improvement in the impurity can be zero for all the splits in S_{H_k}, and ii) the inadequacy for an iterative-split decision tree method (see Figure 1 extracted from [12, Chapter 4]).

As an alternative, entropy was chosen as impurity function because it is one of the most common impurity functions in recent methods. The entropy of a node H_k in a binary decision tree is formulated as follows,

$$I(H_k) = -\frac{|(S_{H_k})^+|}{|(S_{H_k})|} \times \log\left(\frac{|(S_{H_k})^+|}{|(S_{H_k})|}\right) - \frac{|(S_{H_k})^-|}{|(S_{H_k})|} \times \log\left(\frac{|(S_{H_k})^-|}{|(S_{H_k})|}\right) \tag{2}$$

where the superscripts $+$ and $-$ represents the category of the samples.

Splitting Criteria. The H-LSVM algorithm uses a linear SVM as splitting criteria because a single hyperplane vector $\vec{w}$ is obtained as a result of the training process which makes prediction much more efficient. More precisely, the Pegasos algorithm [13] was used because it is an efficient method for training linear SVMs in

large-scale datasets. The Pegasos algorithm minimizes the objective function of a linear SVM in the primal space,

$$\min_{\vec{w}} \frac{\lambda}{2}\|\vec{w}\|^2 + \frac{1}{N}\sum_{(x,y)\in S} \mathcal{L}\left(\vec{w};(\vec{x},y)\right) \tag{3}$$

where $\mathcal{L}\left(\vec{w};(\vec{x},y)\right)$ represents the loss function,

$$\mathcal{L}\left(\vec{w};(\vec{x},y)\right) = \max\left\{0, 1 - y(\vec{w}\cdot\vec{x})\right\}. \tag{4}$$

To solve the problem in Equations 3 and 4, the Pegasos algorithm alternates between stochastic gradient descent steps and projection steps:

- **Stochastic gradient descent**. On iteration t of the algorithm, a set $A_t \subset S$ of size k is chosen. Then, the objective function given in Equation 3 is approximated by,

$$\min_{\vec{w}} f(\vec{w};A_t) = \min_{\vec{w}} \frac{\lambda}{2}\|\vec{w}\|^2 + \frac{1}{k}\sum_{(\vec{x},y)\in A_t} \mathcal{L}\left(\vec{w};(\vec{x},y)\right). \tag{5}$$

  The update of the $\vec{w}$ based on the gradient descent method is given by $\vec{w}_{t+\frac{1}{2}} = \vec{w}_t - \eta_t \vec{\nabla}_t^w$, where $\eta_t = \frac{1}{\lambda t}$ is the learning-rate and $\vec{\nabla}_t^w$ is the subgradient of $f(\vec{w};A_t)$ with respect to $\vec{w}$ on the iteration t,

$$\vec{\nabla}_t^w \quad = \quad \lambda\vec{w}_t - \frac{1}{k}\sum_{(\vec{x},y)\in A_t^+} y\vec{x}\,, \tag{6}$$

  A_t^+ being the set of samples in A_t with non-zero loss that is, $A_t^+ = \{(\vec{x},y)\in A_t \mid y(\vec{w}_t\cdot\vec{x}) < 1\}$.

- **Projection step**. Projection of $\vec{w}_{t+\frac{1}{2}}$ onto the set $B = \left\{\vec{w} \mid \|\vec{w}\| \leq \frac{1}{\sqrt{\lambda}}\right\}$ since it can be shown that the optimal solution of SVM is in the set B [13].

The Pegasos algorithm has been used in the H-LSVM to obtain the oblique splitting hyperplane in each node of the tree but some changes have been applied:

- **Weighted-patterns**. The H-LSVM algorithm generates a piecewise linear model using a decision tree to divide the input space into disjoint regions. In each region, the proportion of patterns of each class might be unbalanced and might not necessarily be the same as in the original problem. In addition, some classification problems, such as fraud detection [31] or medical diagnosis [32], are unbalanced by nature. If the original formulation of the primal SVM objective function is used, the misclassification cost for each pattern is the same and independent of the class. However, this scheme

can give undesirable classifiers which assign the majority class label to all patterns [33] while we are interested in separating the classes with successive decision tree splits. To overcome the imbalance, the H-LSVM method computes the weight $p_i^{H_k}$ of the sample $\vec{x_i}$ in the node H_k according to,

$$p_i^{H_k} = \begin{cases} \frac{1}{2N_{H_k}^+} & \text{if } \vec{x_i} \in S_{H_k}^+ \\ \frac{1}{2N_{H_k}^-} & \text{if } \vec{x_i} \in S_{H_k}^- \end{cases} . \tag{7}$$

Now, the objective function of the Pegasos algorithm incorporates the sample weight in the loss term,

$$\begin{aligned} &\min_{\vec{w}} f(\vec{w}; A_t) = \\ &\min_{\vec{w}} \frac{\lambda}{2}\|\vec{w}\|^2 + \frac{1}{k} \sum_{(p,\vec{x},y)\in A_t} p \max\{0, 1 - y(\vec{w}\cdot\vec{x})\} \end{aligned} \tag{8}$$

and the subgradient of Equation 8 respect to $\vec{w}$ on the iteration t is given by,

$$\vec{\nabla}_t^w = \lambda\vec{w}_t - \frac{1}{k} \sum_{(p,\vec{x},y)\in A_t^+} py\vec{x} . \tag{9}$$

It can be easily shown that the Weighted-Pegasos algorithm still verifies that the norm of the optimum in Equation 8 is upper bounded by $\frac{1}{\sqrt{\lambda}}$ and the number of iterations required for achieving a solution of accuracy ϵ is $O(\frac{1}{\lambda\epsilon})$.

- **The bias term**. The presence of a bias term in the hyperplane is essential for the H-LSVM as a result of the multiple separation of the feature space. There are different approaches to estimate the bias term of the hyperplane [13]. Following the heuristics implemented in standard SGD packages[1], the bias is updated via a subgradient descent and by using a smaller learning rate (scaled by the heuristically chosen parameter τ) because the bias term is updated more frequently than the weights. At each epoch t, not only is the stochastic gradient descent applied to the $\vec{w}$ vector but also to the bias term b: $b_{t+1} = b_t - \tau\eta_t\nabla_t^b$. The subgradient of the bias is given by $\nabla_t^b = -\frac{1}{k}\sum_{(\vec{x},y)\in A_t^+} py$.

- **Pegasos Parameters**. Some meta parameters have to be set in the Weighted Pegasos Algorithm,
  - λ ***Regularization Parameter***: Obtained via a validation subset or cross validation (Section 5).
  - T ***Maximum number of iterations*** in Pegasos Training.
  - k ***size of the subset of samples*** A_t used to update the subgradient.

[1] `http://leon.bottou.org/projects/sgd`

- ϵ_P ***Tolerance*** in Pegasos Training. Allowable tolerance for the norm of the difference between $\vec{w}$ vectors in consecutive iterations.
- τ ***bias scale***. In our experiments we set $\tau = 0.01$.

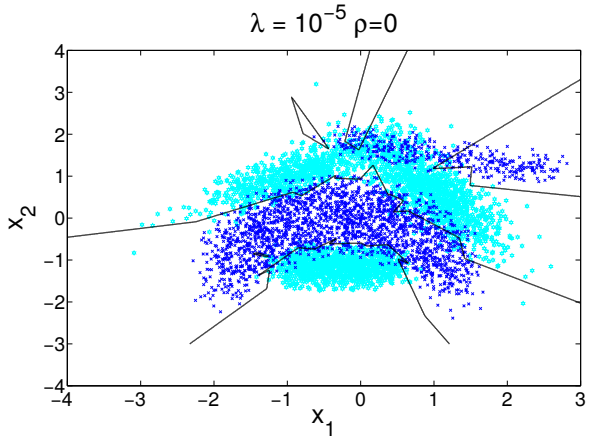
Splitting-Stop Criteria. A node split is stopped when it does not represent an improvement in the impurity measure or when the rate of training samples associated to this node is lower than a parameter δ. If δ is too high, the tree might be not expanded enough. Small values of δ which yield an overfitted tree are preferable because this tree will be pruned later. That is why, δ was set to $10^{-i}, i = \lfloor log_{10} N \rfloor$ in our experiments.

Class Assignment Criteria. Once a pattern reaches a leaf of the decision tree, it is assigned to the majority class in this leaf. It can be shown [12, Chapter 2] that this rule minimizes the expected misclassification probability of the leaf assuming that the cost of misclassifying a pattern is independent of its class.

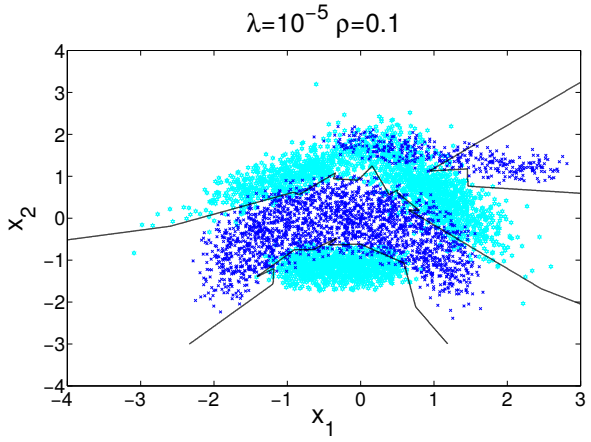
Pruning. Incorporating a pruning process into a decision tree algorithm reduces the risk of having an overfitted model [34, 35]. Although the SVM formulation already incorporates a regularization term which favors the generalization capability of the optimal hyperplane, a small value for δ in the splitting-stop criteria might imply an overfitted model. This point can be solved by setting different δ values and evaluating the performance of the model in a validation step. However, this approach is computationally costlier than using a small value for δ –that is, making the tree grow as much as possible– and then applying a pruning algorithm. The latter approach is used by H-LSVM and it uses the Cost Complexity (CC) pruning algorithm proposed by Breiman et al. [12]. The CC method requires a pruning set not used to train the tree. This set can be selected randomly or via cross-validation. The rate ρ of those training patterns kept away for the pruning phase is a parameter of the H-LSVM algorithm. The main idea of the CC pruning algorithm is to construct a set of decreasing-sized subtrees of the original tree and evaluate the goodness of each subtree as its classification accuracy on the pruning set. In the original CC method, the smallest subtree with a classification accuracy in k standard deviations of the original tree is selected. In our experiments, we set $k = 0$ and, therefore, the subtree selected is that which has the highest classification accuracy and, in the case of several subtrees with the highest accuracy, the smallest one is chosen. For more details, see [12, Chapters 10,11].

Figure 2 shows the decision boundary of the H-LSVM model on the synthetic banana dataset [2] for different pruning rates (ρ). The H-LSVM parameters were $\lambda = 10^{-5}$ and $\delta = 10^{-3}$. Blue and light blue points correspond to positive and negative samples. The problem is not linearly separable. The classification accuracy of the linear SVM is 54.44% while the Gaussian Kernel SVM achieves a classification rate of 90.60% and $1,152$ support vectors. Figure 2(a) shows the H-LSVM decision boundary when no pruning is applied

[2] Dataset available at `http://www.fml.tuebingen.mpg.de/Members/raetsch/benchmark`



(a)



(b)

Figure 2: Best viewed in color. The decision boundary of the H-LSVM model in the banana dataset. Figure 2(a) shows the decision boundary when no pruning is applied. Figure 2(b) shows the decision boundary after a pruning process with $\rho = 0.1$.

($\rho = 0.0$). The model is clearly overfitted. Figure 2(b) shows the H-LSVM decision boundary when pruning is applied ($\rho = 0.1$). This model only needs to evaluate at most 10 hyperplanes to classify a new pattern thus achieving a classification rate of 90.00%. In this case, H-LSVM obtains the same classification accuracy but with a classification time two orders of magnitude lower than the non-linear case.

2.1. Pseudocode

Once the model parameters $T, k, \epsilon_P, \delta, \tau$ have been fixed and the parameters λ and ρ have been estimated in the validation phase, the H-LSVM training procedure can be summarized as follows,

1. Select randomly $(1-\rho)N$ samples from the initial training set S to form the subset S_0. The remaining ρN samples, subset P, is used by the pruning algorithm.

2. Initialize the weight of each pattern in S_0 as described in Equation 7.
3. Train recursively the **H-LSVM Tree** following the steps given in Figure 3.
4. As a result of the H-LSVM tree construction, a set of N_H hyperplanes $\{\vec{w}_n, b_n\}_{n=1}^{N_H}$ is obtained.
5. **Pruning step**: If $\rho > 0$ apply the pruning algorithm on the set P to get $\left\{\left(\tilde{\vec{w}}_n, \tilde{b}_n\right)\right\}_{n=1}^{\tilde{N}_H}$ where $\tilde{N}_H \leq N_H$; otherwise, set $\{(\vec{w}_n, b_n)\}_{n=1}^{N_H} = \left\{\left(\tilde{\vec{w}}_n, \tilde{b}_n\right)\right\}_{n=1}^{\tilde{N}_H}$.
6. **Prediction step**: Let $\tilde{\vec{x}}$ be a new sample and the H-LSVM tree defined by $\left\{\left(\tilde{\vec{w}}_n, \tilde{b}_n\right)\right\}_{n=1}^{\tilde{N}_H}$. The target $\tilde{y}$ of the pattern $\tilde{\vec{x}}$ is calculated as the majority class in the leaf node of the tree associated to $\tilde{\vec{x}}$.

3. Training and Prediction Complexity

In this section we analyze the training and classification cost as a function of the number of operations needed by the linear SVM, the non-linear SVM and the proposed H-LSVM method. As already mentioned, the main advantage of the H-LSVM method is the speeding up of the prediction phase of non-linear SVMs. SVMs have very good results in performance in off-line problems, but when they are placed in a real time operation, such as the credit card fraud detection, they are not viable. Thus, focal attention has to be placed on prediction complexity. Training complexity of H-LSVM is also provided for completeness.

3.1. Training Complexity

The linear SVMs were trained using the popular LIBLINEAR classification package [36]. The algorithm behind LIBLINEAR is coordinate descent in the dual SVM formulation [37]. As pointed out by Menon [38], this algorithm is very attractive because it converges in only $O\left(\log \frac{1}{\epsilon}\right)$ iterations, ϵ being the optimization tolerance. Menon's experiments show that this algorithm achieves a lower generalization error solution faster than Pegasos. However, for large-scale datasets Pegasos' training time *decreases* to get a fixed generalization error [39] while this is not clearly true for LIBLINEAR. The use of the LIBLINEAR package does not affect to our analysis focused on the classification complexity. The non-linear SVMs have been trained using the SMO algorithm whose training cost is $\mathrm{O}(N^2\, d)$ [28] using N d-dimensional patterns. A detailed analysis of these costs can be found in [38].

The H-LSVM cost is that of training as many linear SVMs as nodes in the H-LSVM tree via the Weighted-Pegasos algorithm. More precisely, if the H-LSVM decision tree has N_H internal nodes, the training complexity is given by the cost of training N_H linear SVMs with the Weighted-Pegasos algorithm. Then, considering that the number of iterations needed by the Weighted-Pegasos algorithm to achieve a solution with tolerance ϵ is $O\left(\frac{1}{\lambda\epsilon}\right)$ and the cost per iteration is $O\left(kd\right)$, the total cost of H-LSVM is $O\left(\frac{N_H kd}{\lambda\epsilon}\right)$. For simplicity, the tolerance ϵ is fixed for every node in the tree, but as suggested by Shalev-Shwartz and Srebro [39], it could be adapted as a function of the number of training samples n_i reaching the i-th node to get some fixed generalization error in each node.

INPUT: $S_0, \lambda, T, k, \epsilon_P, \delta, \tau$

I_0=I(H_0)

if $I_0 = 0$ **then**

  FINISH {Homogeneous node}

end if

if $(\frac{|S_0|}{N} > \delta)$ **then**

  $\{\vec{w}, b\}$=Weighted-Pegasos($S_0, \lambda, T, k, \epsilon_P, \tau$)

else

  FINISH {There are not enough patterns.}

end if

if $I(\vec{w}, b) \geq I_0$ **then**

  FINISH {Cannot find any split}

end if

$S_{H^l} = \{\vec{x} \in S_0 \mid \vec{w} \cdot \vec{x} + b \leq 0\}$

$S_{H^r} = \{\vec{x} \in S_0 \mid \vec{w} \cdot \vec{x} + b > 0\}$

if $|S_{H^l}| > 0$ **then**

  Compute the weight of each pattern in S_{H^l} using Equation 7 where $H_k = H^l$

  H-LSVM_Tree($S_{H^l}, \lambda, T, k, \epsilon_P, \delta, \tau$)

end if

if $|S_{H^r}| > 0$ **then**

  Compute the weight of each pattern in S_{H^r} using Equation 7 where $H_k = H^r$

  H-LSVM_Tree($S_{H^r}, \lambda, T, k, \epsilon_P, \delta, \tau$)

end if

OUTPUT: $\{(\vec{w}_n, b_n)\}_{n=1}^{N_H}$

Figure 3: H-LSVM Tree Construction.

	Training	**Classification**
Linear SVM	$Nd\log\left(\frac{1}{\epsilon}\right)$	d
SMO-SVM	$N^2 \times d$	$n_{SV} \times n_K(d)$
H-LSVM	$\frac{N_H k d}{\lambda\epsilon}$	$N_H^P(\vec{x}) \times d$

Table 1: Number of operations needed to train a set S of N patterns in a d-dimensional space (*Training* column) and to classify a new pattern (*Classification* column) by Linear SVM, SVM-SMO and the H-LSVM algorithm. λ: regularization parameter in Pegasos formulation. ϵ: optimization tolerance. n_{SV}: number of support vectors of the non-linear SVM model. $n_K(d)$: operations are needed to compute the kernel between each support vector and the test pattern. N_H: total number of internal nodes in the H-LSVM tree. n_i: number of training samples which reach the node i in the H-LSVM tree. $N_H^P(\vec{x})$: number of nodes encountered by pattern $\vec{x}$ in the H-LSVM tree.

Table 1 (column *Training*) shows the training time complexities of the three algorithms: linear SVM, SVM-SMO and H-LSVM. The H-LSVM cost is highly dependent on each dataset as it is determined by the structure of the tree (N_H). As expected, the lowest training cost corresponds to the linear SVM. The comparison between the training times of non-linear SVM and H-LSVM is not straightforward as it depends on the H-LSVM tree structure and the λ and ϵ parameters. H-LSVM would be faster than SMO-SVM in the training phase if $\frac{N_H k}{\lambda\epsilon} \ll N^2$.

3.2. Prediction Complexity

The cost of classifying a new pattern $\vec{x} \in \mathbb{R}^d$ by a linear SVM is the cost of computing the dot product between the model hyperplane and the pattern to be classified: $O(d)$. In the case of non-linear SVMs, the classification of the pattern $\vec{x}$ is carried out according to: $\sum_{i=1}^{n_{SV}} \alpha_i \times K(\vec{x_i}, \vec{x})$, n_{SV} being the number of support vectors. If $n_K(d)$ is the number of operations needed to compute $K(\vec{x_i}, \vec{x})$, the SVM prediction complexity is $n_{SV} \times n_K(d)$. The proposed H-LSVM algorithm needs to find the leaf of the tree for the pattern $\vec{x}$ which leads to $N_H^P(\vec{x}) \times d$ operations, $N_H^P(\vec{x})$ being the number of internal nodes –oblique hyperplanes– evaluated by the algorithm until the pattern $\vec{x}$ reaches a leaf in the tree.

The summary of the number of operations needed by each algorithm to classify a new pattern $\vec{x}$ is given in Table 1 (column *Classification*).

Obviously the lowest classification cost corresponds to the linear SVM but it will be shown in Section 5 that the linear model is not usually competitive enough for real-world datasets. As regards the non-linear models, it is reasonable to assume that the number of kernel operations $n_K(d)$ is at least d. In that case, H-LSVM has the lowest cost if the number of node evaluations needed to classify the pattern $\vec{x}$, $N_H^P(\vec{x})$, is lower than the number of support vector encountered by SVM, n_{SV}. In Section 5, the values of n_{SV} and $N_H^P(\vec{x})$ for real-world datasets are given, and it is shown that, in practice, the number of evaluations needed by H-LSVM is indeed several orders of magnitude lower.

4. Generalization Error Bound for the H-LSVM Algorithm

In this section we provide a generalization error bound for the H-LSVM algorithm. First, we show that the H-LSVM learning algorithm always converges and produces a decision tree as a final model. The number of nodes to be generated is finite and upper bounded by the number of training samples because of the stopping criteria commonly used in learning decision tree schemes: the tree expansion is finished when there is no improvement in the impurity measure or when there are not enough patterns in a node. The convergence properties of the model can be obtained by considering each node separately and applying the Pegasos convergence bounds [13] which hold in the weighted version.

The generalization error bound is obtained based on the results given by Golea et al. in [40]. Although other bounds for decision trees have been proposed in the literature [41, 42], some of them tighter than those of Golea et al., the latter has been considered in this paper because of its simplicity and its explicit dependence on the decision tree parameters, favoring the understandability of the empirical results obtained in Section 5.6. Among the alternative bounds, the work of Shah [42] based on the Sample Compression (SC) paradigm deserves a special mention because it generally yields tighter bounds and sparse models. These bounds assume axis-parallel decision trees and their application to H-LSVM trees is not straightforward. The formulation of the SC bounds for oblique decision trees is a direction for future work which might also help to alleviate (or even eliminate) the cost of the pruning phase by using these bounds to guide the learning process in a similar way as in [42, 43, 44, 45].

Suppose a two-class decision tree T whose internal decision nodes are labeled with boolean functions from some class $\mathcal{U}$ and whose leaves are labeled as -1 or 1. The bounds obtained depend on the effective number of leaves L_{eff}, a data-dependent quantity which reflects how uniformly the training data covers the tree's leaves and which can be considerably smaller than the total number of leaves in the tree, L [46]. This bound is different from the Vapnik−Chervonenkis one, which depends on the total number of leaves in the tree [47, 48].

Formally, let $P = (P_1, \ldots, P_L)$ the probability vector which represents the probability of a pattern $\vec{x}$ reaching leaf i for $i = 1 \ldots L$. Then, the quadratic distance between the probability vector P and the uniform probability vector $U = (1/L, \ldots, 1/L)$ is given by $\rho(P, U) = \sum_{i=1}^{L} (P_i - 1/L)^2$ and the effective number of leaves in the tree is defined by $L_{\text{eff}} \equiv L(1 - \rho(P, U))$.

A bound of misclassification probability under the distribution $\mathcal{D}$, $P_{\mathcal{D}}\left[T(\vec{x}) \neq y\right]$, can be estimated using the following theorem [40]:

Theorem 1. *For a fixed $\xi > 0$, there is a constant c that satisfies the following. Let $\mathcal{D}$ be a distribution on $X \times \{-1, +1\}$. Consider the class of decision trees of a depth of up to D, with decision functions in $\mathcal{U}$. With a probability of at least $1 - \xi$ on the training set S (of size N), every decision tree T that is consistent with S has*

$$P_{\mathcal{D}}\left[T(\vec{x}) \neq y\right] \leq c\left(\frac{L_{\mathit{eff}}\ VCdim(\mathcal{U})\ log^2 N\ log\ D}{N}\right)^{\frac{1}{2}}$$

where L_{eff} is the effective number of leaves of T and $VCdim$ is the Vapnik Dimension.

The H-LSVM algorithm is in line with this framework identifying the class $\mathcal{U}$ with the Linear SVM. It is known that the Vapnik Dimension of a hyperplane in a d-dimensional space is $(d+1)$ [49] therefore, the error bound for the H-LSVM method is reformulated as,

Lemma 2. *For a fixed $\xi > 0$, there is a constant c that satisfies the following. Let $\mathcal{D}$ be a distribution on $X \times \{-1, +1\}$. Consider the class of decision trees of a depth of up to D, with H-LSVM decision functions. With a probability of at least $1-\xi$ on the training set S (of size N), every decision tree T that is consistent with S has*

$$P_{\mathcal{D}}\left[T(\vec{x}) \neq y\right] \leq c\left(\frac{L_{\mathit{eff}}\,(d+1)\ log^2 N\ log\ D}{N}\right)^{\frac{1}{2}} \tag{10}$$

where L_{eff} is the effective number of leaves of T.

In practice it is quite difficult to have a consistent tree with the training data S. In that case, a bound of the misclassification probability can be obtained as a function of the misclassification probability in S, $P_S\left[T(\vec{x}) \neq y\right]$. Now, the probability vector P is reformulated according to the training set as:

$$P'_i = \frac{P_i P_S\left[T(\vec{x}) = y \mid \vec{x} \text{ reaches leaf } i\right]}{P_S\left[T(\vec{x}) = y\right]}$$

By applying the theorem given in [40] for the particular case of the H-LSVM tree, we obtain the following result,

Lemma 3. *For a fixed $\xi > 0$, there is a constant c that satisfies the following. Let $\mathcal{D}$ be a distribution on $X \times \{-1, +1\}$. Consider the class of decision trees of a depth of up to D with H-LSVM internal node decision functions. With a probability of at least $1-\xi$ on the training set S (of size N), every decision tree T has*

$$P_{\mathcal{D}}\left[T(\vec{x}) \neq y\right] \leq P_S\left[T(\vec{x}) \neq y\right] + c\left(\frac{L'_{\mathit{eff}}\,(d+1)\ log^2 N\ log\ D}{N}\right)^{\frac{1}{3}} \tag{11}$$

where c is a universal constant, and $L'_{\mathit{eff}} = L(1-\rho(P', U))$ is the empirical effective number of leaves of T.

Therefore, for a given training set S of N patterns, the parameters of the tree which determine the error bound for the H-LSVM algorithm are the depth D of the tree and the effective number of leaves L_{eff}: the lower these parameters are, the better generalization error. The parameter values and an estimation of the model complexity according to Equation 11 are given in Section 5.

5. Experiments

The aim of the experiments described in the following subsections is fourfold:

- Compare H-LSVM with linear SVMs and non-linear SVMs in terms of classification accuracy and prediction complexity (Section 5.2).
- Compare H-LSVM with Zapién's [21, 24] and Adaboost [50] algorithms in terms of classification accuracy and prediction complexity (Sections 5.3 and 5.4).
- Analyze numerically the H-LSVM scalability (Section 5.5).
- Analyze numerically the H-LSVM error bound studied in Section 4 (Section 5.6).

The H-LSVM has been implemented in C language and the code can be found at:

`https://sites.google.com/site/irenerodriguezlujan/HLSVM-1.1.zip`.

As the H-LSVM algorithm has been designed for binary classification domains, the experiments have been conducted in large-scale binary classification problems. We have considered the large-scale datasets used by Keerthi et al. [2]: *IJCNN*, *Shuttle*, *M3VO* and *Vehicle*. The *Shuttle* dataset has been converted to a binary classification problem by differentiating class 1 from the rest. In the same way, the *Vehicle* dataset has been reformulated as a binary classification task consisting of differentiating class 3 from the rest. The *M3VO* dataset corresponds to differentiate digit 3 from all the other digits in the *MNIST* problem. An extension of the *MNIST* dataset with $8,100,000$ patterns (*MNIST8m*) has also been included since it represents a very large-scale classification problem. Again, the classification of digit 3 from all of the others has been considered (*M3VOm8*). In order to compare the performance of our method with the Zapién algorithm [21, 24], initially we chose the binary datasets used in this work: *Heart* and *Faces*. However in the case of the *Heart* dataset, the classification accuracy of the linear SVM is the same as that of the non-linear SVM and thus, we decided not to include this dataset in our experiments. Finally, we added the binary version of the covtype dataset (class 2 versus others) because of its large number of patterns. The characteristics of the datasets are shown in Table 2 as well as the repositories where they are available.

In most of the datasets (*IJCNN*, *Shuttle*, *M3VO* and *Vehicle*), the training and test subsets are given beforehand. In the *Faces* dataset, we followed the experimental setup described in [52] which uses two

	# Train	# Test	# Feat.	Repository
IJCNN	49,990	91,701	22	LIBSVM [51]
Shuttle	43,500	14,500	9	LIBSVM [51]
M3VO	60,000	10,000	780	LIBSVM [51]
M3VOm8	810,000	7,290,000	784	LIBSVM [51]
Vehicle	78,823	19,705	100	LIBSVM [51]
Faces	8,525	4,263	576	`http://www.cs.ubc.ca/~pcarbo/#data`
Binary Covtype	522,910	58,102	54	LIBSVM Reposity [51]

Table 2: Binary datasets used to compare H-LSVM with linear SVMs and non-linear SVMs.

thirds of the observations for the training and the rest as a testing set. Moreover, data was normalized to minimum and maximum feature values. We run the experiments on 10 different randomly chosen training-test partitions of the dataset. In the case of the *M3VOm8* and *Covtype* datasets, we have tried to use as many training patterns as possible in order to simulate a large-scale system with a high number of support vectors [3]. Then, the first 810,000 patterns in the *M3VOm8* dataset were used for training and the remaining samples for test. In the *Covtype* dataset, according to the experiments carried out in [9, 53], 9/10 of the samples for training and the remaining patterns for test. In both cases, the experiments were run on 10 different randomly chosen training-test partitions of the dataset.

In all of the experiments, linear SVMs and non-linear SVMs implemented in LIBLINEAR [36] and LIBSVM [51] packages were used. The Gaussian kernel, $k(x_i, x_j) = \exp\left(-\gamma\|x_i - x_j\|^2\right)$, was used for non-linear SVMs.

5.1. Hyperparameter Tuning

Linear SVMs, non-linear SVMs and H-LSVM need to determine the values of a few parameters. In all datasets, except *Covtype*, the hyperparameter selection has been made using a 5-fold cross validation on the training set. The cost parameters in linear SVMs and non-linear SVMs were selected from the grid $10^i, i = -6, \ldots, 6$. The γ parameter of the Gaussian kernel was taken from the range $10^i, i = -3, \ldots, 3$. Finally, for the H-LSVM model, we fixed the maximum number of Pegasos iterations $T = 10^7$ with a tolerance of $\epsilon_P = 10^{-4}$ and the minimum proportion of patterns needed to split a node δ was chosen as 10^{-i} with $i = \lfloor \log_{10} N \rfloor$ to guarantee that the H-LSVM grows to a sufficient size (pruning is applied if necessary). The regularization parameter λ was chosen from the grid $\frac{10^i}{N}, i = -6, \ldots, 6$, N being the number of training

[3] LIBSVM for the *M3VOm8* dataset did not finish in reasonable time when training with all the available patterns.

	LIBLINEAR	**LIBSVM**		**H-LSVM**	
	c	c	γ	λ	ρ
IJCNN	10^1	10^1	10^0	10^{-5}	0
Shuttle	10^2	10^6	10^0	10^{-7}	0.2
M3VO	10^0	10^2	10^{-2}	10^{-4}	0.2
M3VOm8	10^0	10^2	10^{-2}	10^{-5}	0.2
Vehicle	10^{-1}	10^1	10^{-1}	10^{-6}	0.1
Faces	10^{-1}	10^1	10^{-2}	10^{-5}	0.1
Covtype	10^0	10^0	0.346	10^{-7}	0.0

Table 3: Parameters used in the linear SVM, non-linear SVM and H-LSVM models for each binary dataset.

samples. The grid was obtained from the equivalence $\lambda = \frac{1}{CN}$ between the LIBLINEAR and LIBSVM cost parameter c and the λ regularizer in H-LSVM. The prune rate ρ took values in $[0.0, 0.1, 0.2]$. For the *Covtype* dataset, we used the non-linear SVM hyperparameters provided in [9]. The resulting parameters for each dataset and each model are given in Table 3.

5.2. Results

The results in terms of classification error (*Error (%)*) and classification cost are shown in Table 4. In the case of the linear SVM, the number of hyperplane evaluations is shown whereas the number of support vectors is indicated for the non-linear SVM (n_{SV} *or Hyp*). While the classification cost of linear/non-linear SVMs is independent of the test sample, the H-LSVM prediction cost depends on the path of the pattern in the H-LSVM tree. Thus, the mean number of H-LSVM hyperplanes encountered per test sample together with the maximum number of H-LSVM hyperplane evaluations written in parentheses are shown. In those cases in which there were several training/test partitions, the average and standard deviation on the 10 runs of the experiment are shown.

The quantification of the performance of the algorithms considering the linear and non-linear SVMs as the points of reference is given by the quantities *Relative Error (RE)* and *Relative Complexity (RC)*,

$$RE = \frac{e_{\text{LSVM}} - e}{e_{\text{LSVM}} - e_{\text{SVM}}} \tag{12}$$

$$RC = \frac{Hyp - 1}{n_{SV} - 1}, \tag{13}$$

where e represents the classification error rate. A value equals 0 at these magnitudes *RE/RC* indicate that the classification accuracy/complexity is the same as that of the linear SVM while a value of 1 represents

the equivalence with the non-linear case. Therefore, it would be desirable to have a *Relative Error* close to 1 and a *Relative Complexity* close to 0.

As expected, the classification results of the non-linear SVMs are greater than those of the linear SVM and H-LSVM. However, the classification accuracy of the H-LSVM is significantly better than that of the linear model in all cases. These results are not surprising because the proposed H-LSVM method is simpler than the non-linear SVM but more sophisticated than linear SVMs. While the classification error of H-LSVM is closer to that of the non-linear SVM in most cases, the H-LSVM classification accuracy is closer to the linear model for the *Faces* and *M3VO* datasets. Nevertheless, in the case of the *Faces* dataset the H-LSVM model represents an improvement of 41% in respect to the linear SVM and it will be shown later that it yields significantly better results than the Zapién et al. [21] algorithm. These results show that the non-linear SVMs cannot be approximated by the proposed method in certain domains. It is worth pointing out that H-LSVM outperforms the non-linear SVM in the *Covtype* dataset. Although, the classification error obtained for the non-linear SVM is comparable to the results reported in [9], a thorough search of the non-linear SVM parameters might provide better results. Unfortunately, to apply the hyperparameter procedure described in Section 5.1 is unfeasible because of the size of the dataset and the number of support vectors.

Our main interest is not having the best classification error rates but providing a method capable of classifying a pattern in few milliseconds while obtaining a competitive performance. In this respect, the non-linear SVM needs the largest number of operations in prediction while the lowest cost is that of the linear SVM. However, the performance of the linear SVM can be extremely poor as in the *IJCNN* or *Covtype* datasets. The classification complexity of H-LSVM is between these two models: it is higher than that of the linear SVM –in the worst case it increases the cost of the linear model in one order of magnitude– but much lower than the cost of the non-linear SVM –H-LSVM can accelerate the prediction cost of the non-linear SVM even by a factor of 10^4 as in the case of the *M3VOm8* and *Covtype* datasets. In fact, the *Relative Complexity* is lower than 10^{-1} in all cases.

5.3. Results: Comparison with SVM Trees Algorithm

Having compared H-LSVM to baseline models, we can contrast the results with the Zapién decision tree [21]. As mentioned above, only one of the two binary classification problems used in this work cannot be classified accurately by a linear SVM (*Faces*). Despite the fact that our method has been designed for binary classification problems, the performance of our model in the multiclass USPS dataset was measured. The USPS dataset for handwritten text recognition is available in the LIBSVM Repository [51]. It consists of 7291 training samples and 2007 test samples. Each example is described by 256 features. Following the methodology described in [52], we normalized the data to minimum and maximum feature values and we applied *one against one* approach (1A1) for the multiclass problem. The 1A1 strategy consists of

IJCNN

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	7.82	1.01	2.36
n_{SV} *or Hyp*	1	$3,154$	7.28 (16)
RE / RC	0 / 0	1 / 1	0.80 / $2.0 \cdot 10^{-3}$

Shuttle

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	2.21	0.062	0.10
n_{SV} *or Hyp*	1	66	5.18 (12)
RE / RC	0 / 0	1 / 1	0.98 / $6.43 \cdot 10^{-2}$

M3VO

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	2.09	0.33	1.79
n_{SV} *or Hyp*	1	$2,873$	3.15 (8)
RE / RC	0 / 0	1 / 1	0.17 / $7.49 \cdot 10^{-4}$

M3VOm8

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	3.90	0.03	1.43 ± 0.02
n_{SV} *or Hyp*	1	$13,471$	4.73 ± 0.003 (11.00 ± 0.00)
RE / RC	0 / 0	1 / 1	0.64 / $2.77 \cdot 10^{-4}$

Vehicle

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	14.18	11.88	12.61
n_{SV} *or Hyp*	1	$23,642$	2.84 (10)
RE / RC	0 / 0	1 / 1	0.68 / $7.78 \cdot 10^{-5}$

Faces

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	8.81 ± 0.35	2.97 ± 0.24	6.39 ± 0.43
n_{SV} *or Hyp*	1	$1,260.3 \pm 14.81$	2.59 ± 0.06 (5.30 ± 0.15)
RE / RC	0 / 0	1 / 1	0.41 / $1.26 \cdot 10^{-3}$

Covtype

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	23.66 ± 0.21	18.57 ± 0.20	11.39 ± 0.08
n_{SV} *or Hyp*	1	$245,687.2 \pm 167.8$	12.93 ± 0.087 (44.00 ± 1.08)
RE / RC	0 / 0	1 / 1	2.41 / $4.86 \cdot 10^{-5}$

Table 4: Test error rate (*Class. Err (%)*) and classification complexity (n_{SV} *or Hyp*) of Linear SVMs, non-linear SVMs and H-LSVM. The mean number of hyperplane evaluations per test sample is indicated for linear SVMs and H-LSVM. The maximum number of H-LSVM hyperplane evaluations is shown in parentheses. In the case of non-linear SVMs, the number of support vectors (n_{SV}) is shown. The reference measures RE and RC (Equations 12 and 13) are also provided.

Faces

	Linear SVM	*Non-linear SVM*	*SVM Trees*	*H-LSVM*
Class. Err (%)	8.81	2.97	8.99	6.39
n_{SV} *or Hyp*	1	1260.3	4	2.59 (5.30)
RE / RC	0 / 0	1 / 1	−0.03 / 0.002	0.41 / 0.001

USPS

	Linear SVM	*Non-linear SVM*	*SVM Trees*	*H-LSVM*
Class. Err (%)	8.67	4.53	6.24	5.38
n_{SV} *or Hyp*	1	1521	49	64.77 (117)
RE / RC	0 / 0	1 / 1	0.59 / 0.03	0.79 / 0.04

Table 5: Comparison of the SVM Trees method by Zapién et al. [21, 52] and H-LSVM. Misclassification error (*Class. Err (%)*) and the mean number of hyperplane evaluations per test sample (*Hyp*) are shown for both methods and for the linear and non-linear SVMs (n_{SV}). The maximum number of H-LSVM hyperplane evaluations is indicated in parentheses. The number of hyperparameter evaluations was computed as the sum of the hyperplanes evaluated in every binary classifier. The *Relative Error (RE)* and the *Relative Complexity (RC)* of the SVM Trees method and H-LSVM are also given.

training a classifier for every pair of classes and classifying a new pattern based on majority voting. The hyperparameters were chosen by using 5-CV as described above. For the linear SVM, the cost parameter was set to $c = 1$, for the non-linear SVM $c = 10^1$ and $\gamma = 10^{-2}$ and for the H-LSVM algorithm $\lambda = 10^{-5}$ and $\rho = 0$. The results in terms of the misclassification error and classification cost for both methods are given in Table 5. The performance of the Zapién method has been extracted from [21, 52].

In both cases H-LSVM is superior in terms of classification accuracy whereas the classification cost is in the same order of magnitude. Specifically, their classification complexity is quite similar in the *Faces* dataset but the SVM Trees algorithm is slightly faster for the *USPS* database. In any case, the classification cost of both algorithms is of the same order of magnitude. In summary, the H-LSVM decision tree, expanding both children of each node as well as the weighted patterns used in the linear SVM training, provide advantages in terms of classification accuracy while maintaining the classification cost. It is also worth noting that in all cases the maximum depth of the tree is lower than the number of internal nodes (the number of linear SVMs), which means that the structure of the tree is far from being a cascade of classifiers as in [21, 24, 25, 52]. The superiority of the H-LSVM tree against the Zapién's algorithm in terms of classification accuracy is not surprising given that, as already mentioned in Section 1, the hypothesis class of H-LSVM (disjunctions of conjunctions) is more general than that of the SVM Trees algorithm (conjunctions) [22]. What is more, this difference can be quantified taking into account that the number of decision tree skeletons with k decision

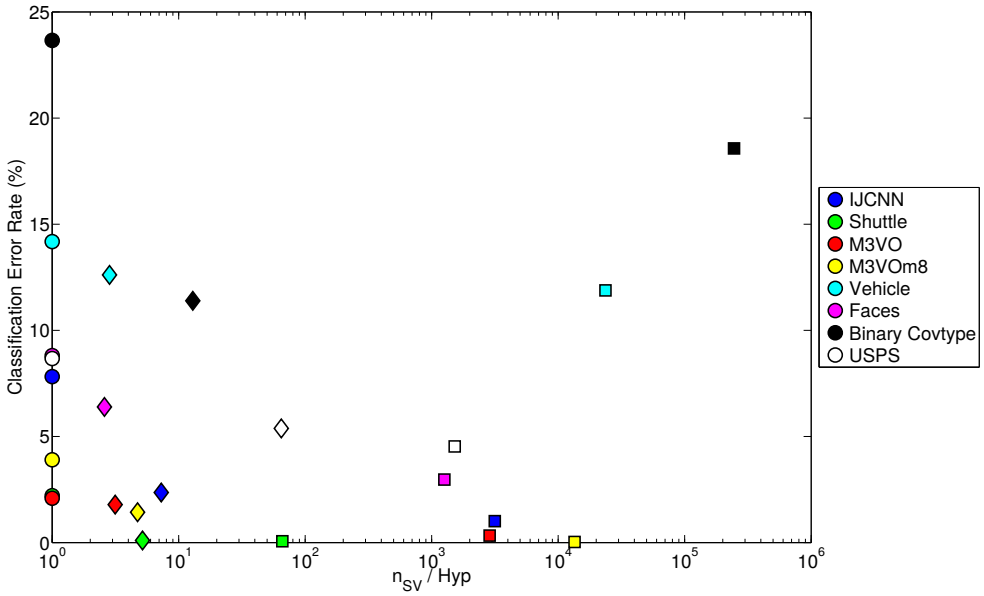


Figure 4: Best viewed in color. Classification complexity (n_{SV} / *Hyp*) versus classification error rate for different datasets. • Linear SVM ■ Non-linear SVM ♦ H-LSVM.

nodes is given by the k-th Catalan Number [19, 23] in comparison to the only one possibility for the Zapién's method.

In order to visualize the trade-off between the misclassification error vs. classification cost, Figure 4 shows the dependence between these two magnitudes for the linear SVM, non-linear SVM and H-LSVM. The x-axis represents the number of support vectors or hyperplanes encountered by each method in logarithmic scale. The y-axis shows the classification error rate. Each dataset is represented by a color according to the legend. Circles, squares and diamonds represent the linear SVM, non-linear SVM and H-LSVM models, respectively. The lower left-hand area is associated to the best scenario: the lowest classification error and the lowest classification complexity. In this Figure, three clusters can be easily identified according to the classifier (circles, squares and diamonds). Clearly, the non-linear SVMs have the highest classification complexity while the H-LSVM cost is closer to the linear one. Looking at the classification error, in all cases the non-linear SVM is superior – except the *Covtype* dataset – and the H-LSVM effectiveness is greater than that of the linear model.

Finally, to give an idea of the quality of the H-LSVM algorithm with regard to the prediction time, Table 6 shows the time in seconds needed by a linear SVM, a non-linear SVM and H-LSVM to classify a new pattern in an Intel(R) Core(TM) i7 CPU 920 at 2.67GHz. The training time is also included for completeness. As expected, the lowest training and test times correspond to the linear SVM. As regards the

	Linear SVM		**Nonlinear SVM**		**H-LSVM**	
	Training	*Test*	*Training*	*Test*	*Training*	*Test*
IJCNN	$4.00 \cdot 10^{-1}$	$1.22 \cdot 10^{-7}$	$2.54 \cdot 10^{1}$	$2.17 \cdot 10^{-4}$	$1.44 \cdot 10^{3}$	$2.21 \cdot 10^{-6}$
Shuttle	$4.77 \cdot 10^{-1}$	$1.09 \cdot 10^{-7}$	$4.83 \cdot 10^{0}$	$3.91 \cdot 10^{-6}$	$2.37 \cdot 10^{4}$	$2.76 \cdot 10^{-6}$
M3VO	$4.76 \cdot 10^{0}$	$6.47 \cdot 10^{-7}$	$6.40 \cdot 10^{3}$	$3.27 \cdot 10^{-3}$	$6.25 \cdot 10^{3}$	$3.61 \cdot 10^{-5}$
M3VOm8	$3.38 \cdot 10^{2}$	$3.00 \cdot 10^{-6}$ $\pm 1.28 \cdot 10^{-8}$	$7.71 \cdot 10^{4}$	$2.77 \cdot 10^{-2}$ $\pm 5.88 \cdot 10^{-6}$	$7.19 \cdot 10^{4}$	$2.88 \cdot 10^{-5}$ $\pm 4.90 \cdot 10^{-8}$
Vehicle	$2.82 \cdot 10^{0}$	$4.83 \cdot 10^{-7}$	$1.85 \cdot 10^{3}$	$9.02 \cdot 10^{-3}$	$2.99 \cdot 10^{4}$	$2.67 \cdot 10^{-6}$
Faces	$1.04 \cdot 10^{0}$ $\pm 5.28 \cdot 10^{-2}$	$2.41 \cdot 10^{-6}$ $\pm 5.38 \cdot 10^{-9}$	$2.39 \cdot 10^{1}$ $\pm 9.61 \cdot 10^{-2}$	$2.52 \cdot 10^{-3}$ $\pm 1.66 \cdot 10^{-5}$	$4.70 \cdot 10^{3}$ $\pm 5.98 \cdot 10^{1}$	$1.36 \cdot 10^{-5}$ $\pm 4.71 \cdot 10^{-7}$
Covtype	$6.65 \cdot 10^{1}$ $\pm 1.05 \cdot 10^{-1}$	$1.30 \cdot 10^{-7}$ $\pm 1.76 \cdot 10^{-9}$	$2.10 \cdot 10^{4}$ $\pm 1.31 \cdot 10^{2}$	$1.95 \cdot 10^{-2}$ $\pm 7.43 \cdot 10^{-5}$	$3.13 \cdot 10^{4}$ $\pm 8.47 \cdot 10^{1}$	$6.83 \cdot 10^{-6}$ $\pm 3.86 \cdot 10^{-8}$
USPS	$5.46 \cdot 10^{0}$	$3.58 \cdot 10^{-6}$	$5.21 \cdot 10^{0}$	$1.17 \cdot 10^{-3}$	$5.88 \cdot 10^{3}$	$1.59 \cdot 10^{-4}$

Table 6: Training and testing time in seconds required by LIBLINEAR, LIBSVM and H-LSVM.

training cost discussed in Section 3.1, the differences between the training cost of the non-linear SVM and H-LSVM are given by the structure of the H-LSVM tree. Therefore, depending on the dataset either the non-linear SVM or H-LSVM is faster in the training phase. Focusing on the aim of speeding up the non-linear SVM prediction cost, the H-LSVM classification time is always in the order of tenths of milliseconds at most and significantly lower than those of the non-linear SVM.

Several techniques based on the use of linear SVMs on the manifold coordinates have come out recently [15, 16, 17]. In particular, the Locally Linear SVM (LLSVM) model proposed by Ladicky et al. [17] reports results for the USPS dataset. The classification accuracy of H-LSVM is slightly better than that of the LLSVM and the LLSVM algorithm needs to compute the distance to 100 k-means centroids while H-LSVM evaluates on average 64.77 hyperplanes (maximum 117). That is, both methods are comparable in terms of classification accuracy and prediction complexity.

5.4. Results: Comparison with Adaboost Algorithm

Other natural competitors for H-LSVM are boosting algorithms [54] since they create piecewise linear functions with a good generalization performance [55] and low classification cost. In particular, we have considered the most known boosting algorithm: AdaBoost (Adaptive Boosting) [50]. Decision stumps were used in accordance with the Adaboost algorithm originally proposed by its authors Freund and Schapire [50] and motivated by its successful application in the state-of-the-art Viola-Jones face detection algorithm [56]. The main advantage of Adaboost with decision stumps on competitors is the speed of learning and prediction, which is particularly critical in large-scale problems.

Adaboost requires the establishment of the maximum number of weak classifiers H to be used. Since our paper focuses on accelerating the classification times, H was fixed to make the prediction cost of Adaboost comparable to that of H-LSVM. Then, by taking into account that Adaboost needs to evaluate all the weak learners to classify a test pattern and considering that the prediction complexity of each decision stump is $O(1)$, H is computed as the mean number of hyperplanes evaluated by H-LSVM multiplied by the dimension of the patterns. The comparison of both methods in terms of misclassification rate and classification cost as well as the value of the parameter H for each dataset are given in Table 7.

The results show that Adaboost has a better performance in the *Shuttle* and *Vehicle* datasets, the difference in classification accuracy being 0.15% at most. However, in some cases such as *IJCNN* and *Covtype*, H-LSVM significantly outperforms Adaboost. On average, Adaboost and H-LSVM have misclassification rates of 7.81% and 5.15%, respectively on all the datasets. Overall, the H-LSVM yields a better performance/classification speed ratio than Adaboost with decision stumps.

5.5. Numerical Analysis of H-LSVM Scalability

To illustrate the applicability of the H-LSVM algorithm to real large-scale scenarios, we show the scalability in the training time and convergence of the test error rates as the number of training samples increases. In this regard, Figures 5a – 5c show the training complexity of H-LSVM in terms of the number of hyperplanes in the H-LSVM tree and the computational time as a function of the number of training samples N. Figure 5d shows the training and test classification accuracies as a function of N. The results represent the average on the 10 training/test partitions of the *Binary Covtype* dataset. In turn, 4 subsets of size $10,000$, $50,000$, $100,000$ and $200,000$, respectively, have been randomly chosen from each training partition.

According to the training cost of H-LSVM presented in Section 3.1, $O(\frac{N_H kd}{\lambda\epsilon})$, by maintaining λ, ϵ and d constant for the different training sizes, the training cost of H-LSVM depends on the subsampling rate k and the number of internal nodes in the H-LSVM tree N_H as $O(N_H k)$. Unfortunately, N_H depends inextricably on the problem in question and thus, a general estimation of N_H based on N cannot be provided. However, it is possible to compare the number of internal nodes in the H-LSVM tree against those corresponding to the best tree (balanced tree) and the worst one (a linear or cascade tree). In this regard, Figures 5a (linear

	Adaboost			**H-LSVM**		
	Class. Err (%)	*H*	*Cost*	*Class. Err (%)*	*Hyp*	*Cost*
IJCNN	6.50	170	170	2.36	7.28	160.16
Shuttle	0.08	50	50	0.10	5.18	46.62
M3VO	2.69	2460	2460	1.79	3.15	2457.00
M3VOm8	3.61 ± 0.01	3710	3710	1.43 ± 0.02	4.73	3708.32
Vehicle	12.46	290	290	12.61	2.84	284.00
Faces	6.39 ± 0.33	1500	1500	6.39 ± 0.43	2.59	1491.84
Binary Covtype	22.95 ± 0.23	700	700	11.39 ± 0.08	12.93	698.22
Average	**7.81**	**1268.57**	**1268.57**	**5.15**	**5.53**	**1263.74**

Table 7: Test error rate (*Class. Err (%)*) and classification complexity (*Cost*) of Adaboost and H-LSVM. The number of weak learners (H) are indicated for Adaboost and the mean number of hyperplane evaluations per test sample is indicated for H-LSVM (*Hyp*). Note that the classification cost is computed by considering that the classification complexity of each decision stump is $O(1)$ whereas it is $O(d)$ for each hyperplane in the H-LSVM tree.

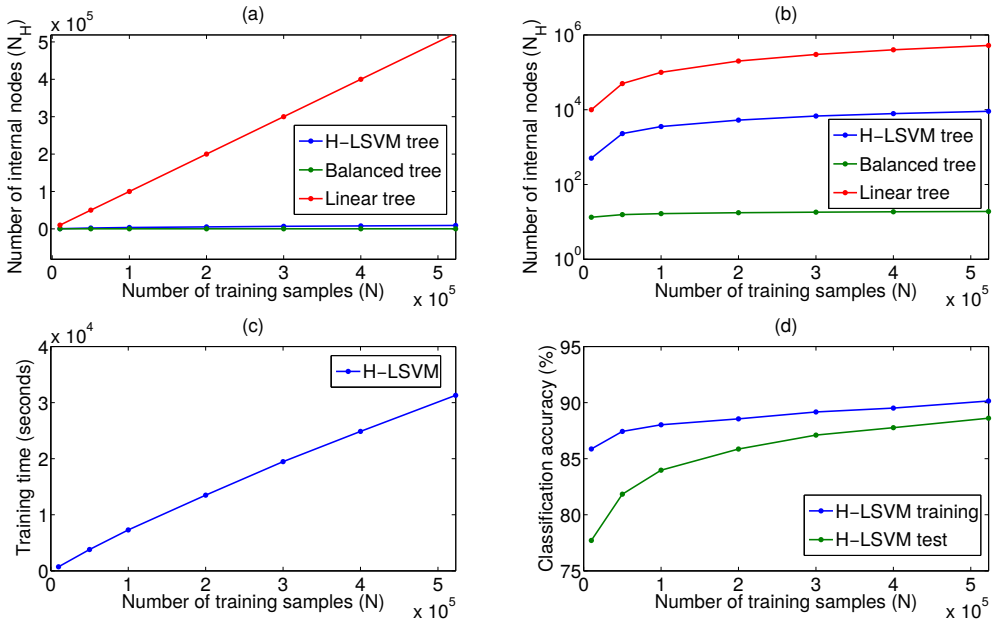


Figure 5: H-LSVM training complexity and classification rate convergence as a function of the number of training samples (N) for the *Covtype* dataset. Figure 5a: number of internal nodes N_H in the H-LSVM tree. Reference values corresponding to a balanced and cascade tree are also included. Figure 5b: Figure 5a using logarithmic axis in the y-axis. Figure 5c: training time of the H-LSVM algorithm. Figure 5d: training and test classification accuracies.

y-axis) and 5b (logarithmic y-axis) include the number of internal nodes associated with the balanced decision tree ($\log_2(N)$) and those encountered in the cascade structure (N). In this case, the complexity of the H-LSVM tree is closer to that of the balanced tree. Similar results are expected for the other datasets since in all cases the maximum depth of the tree is much lower than the number of internal nodes.

Furthermore, the variability in the distribution of the training samples throughout the decision tree also affects the exact computation of a general training cost of the Pegasos algorithm in each node. Although the subsample size k is fixed at the beginning of the algorithm – in this experiment, it was set $50,000$ –, the effective subsampling size in each node is determined online as the minimum between k and the number of samples reaching the current node, which is totally linked to each particular dataset. Nevertheless, the empirical measure of the training time of H-LSVM as a function of the number of training samples N shown in Figure 5c seems to have a linear growth and, in fact, it has been proven that the polynomial curve fitting the points with the lowest error is that of degree 1. Again, a comparison with the training complexity of the balanced and the linear decision trees can be valuable. Under the assumption that the cost of computing the split of the i-th node is proportional to the number of samples n_i reaching the node, the balanced tree has a linear cost with respect to N:

$$\sum_{i=0}^{\log_2(N)} i\, n_i = \sum_{i=0}^{\log_2(N)} i\, \frac{i}{2^i} = O(N)\ ,$$

while the linear tree has a quadratic dependence:

$$\sum_{i=0}^{N-1} n_i = \sum_{i=0}^{N-1} (N-i) = O(N^2)\ .$$

Therefore, the complexity of H-LSVM training is closer to the best scenario. Finally, Figure 5d reveals that the gap between training and test errors converges with approximately $300,000$ patterns. Although the classification accuracy in the test set increases with the number of training samples, the improvement becomes smaller as N grows, especially when N is larger than $300,000$ in which case the difference with respect to the model trained with all the training samples is 0.52%.

The preceding results corroborate the applicability of H-LSVM to large-scale scenarios.

5.6. Numerical Analysis of H-LSVM Generalization Error Bound

Lemma 3 provides a generalization error bound for the H-LSVM method as a function of some data-dependent parameters according to the equation,

$$P_{\mathcal{D}}\left[T(\vec{x}) \neq y\right] \leq P_S\left[T(\vec{x}) \neq y\right] + c\left(\frac{L'_{\text{eff}}\,(d+1)\,log^2 N\,log\,D}{N}\right)^{\frac{1}{3}}$$

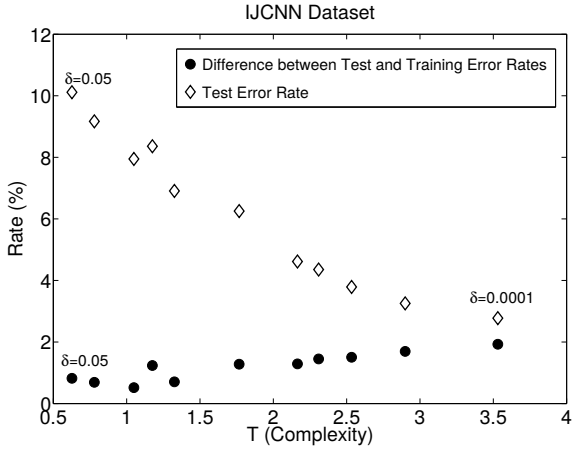
which establishes a linear dependence between the difference $P_{\mathcal{D}}\left[T(\vec{x}) \neq y\right] - P_S\left[T(\vec{x}) \neq y\right]$ and the complexity of the tree given by $T = \left(\frac{L'_{\text{eff}}\ (d+1)\ log^2 N\ log\ D}{N}\right)^{\frac{1}{3}}$. In this section, an empirical analysis of the above equation is provided.

The misclassification probability under a distribution $\mathcal{D}$, $P_{\mathcal{D}}\left[T(\vec{x}) \neq y\right]$ has been approximated with the error rate in the test set: $\hat{P}_{\mathcal{D}}\left[T(\vec{x}) \neq y\right]$. The range of values of the complexity measure T depends on the characteristic of each dataset making the comparison between the different datasets impossible. However, an interesting point of analysis is to determine whether in practice a linear correlation exists between the difference of the test and training error rates and the complexity of the model T. This relationship is analyzed for the *IJCNN* and *Faces* datasets by varying the values of the δ parameter to obtain the values for T, $P_S\left[T(\vec{x}) \neq y\right]$ and $\hat{P}_{\mathcal{D}}\left[T(\vec{x}) \neq y\right]$. The δ parameter allows the complexity of the model to be measured and controlled. If δ takes values in the grid $\{\delta_1 > \delta_2 > \ldots > \delta_M\}$, the obtained trees $\mathcal{T}_{\delta_m}$ verify $\mathcal{T}_{\delta_1} \subseteq \mathcal{T}_{\delta_2} \subseteq \ldots \subseteq \mathcal{T}_{\delta_M}$. In our experiment, the δ grid was: $\{5 \cdot 10^{-2}$, $2.5 \cdot 10^{-2}$, $1 \cdot 10^{-2}$, $7.5 \cdot 10^{-3}$, $5 \cdot 10^{-3}$, $2.5 \cdot 10^{-3}$, $1 \cdot 10^{-3}$, $7.5 \cdot 10^{-4}$, $5 \cdot 10^{-4}$, $2.5 \cdot 10^{-4}$, $1 \cdot 10^{-4}\}$, the prune rate was fixed to $\rho = 0.0$ in both cases and the λ parameter was selected as in Table 3. The obtained results are shown in Figure 6 in which the high correlation between the complexity term T and the gap between the training and test errors is shown by the points representing the difference between the training and test errors. More precisely, the linear correlation between the H-LSVM tree complexity T and the difference between the test and training error rates is 0.91 for the *IJCNN* dataset and it is 0.97 for the *Faces* dataset. These high correlations show that the generalization error bound given in Lemma 3 holds in practice.

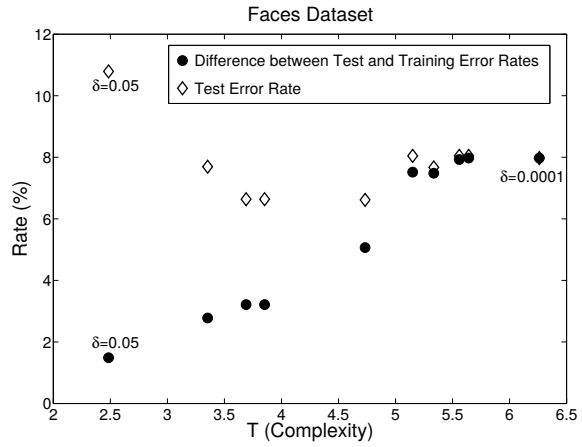
Finally, it is interesting to see how the underfitting and overfitting effects are reflected in Figure 6. In the case of the *IJCNN* dataset, the differences between the test and training error rates are small for the largest values of δ while the test error rate is the worst. It is a case of underfitting. On the other hand, the lowest values for δ have the largest differences between the test and training error rates but the test error rate is the lowest. This scenario is preferable to that with large values of δ. In the *Faces* dataset the underfitting/overfitting are clearly reflected for high/small δ values, respectively. Regarding how the δ parameter was chosen in the experiments (see Table 3), it makes sense that the optimal pruning rate for the *Faces* dataset was $\rho = 0.1$ in order to avoid overfitting.

6. Conclusions

This paper has presented and analyzed a new classification method for medium and large-scale datasets. As the application of non-linear SVMs in these problems is prohibitive because it generates a large number of support vectors, the proposed method takes advantage of the efficiency of linear SVMs to construct a piecewise linear model. The new algorithm, called a Hierarchical Linear Support Vector Machine (H-LSVM), is based on the construction of a decision tree whose node splits are Linear Support Vector Machine trained



(a)



(b)

Figure 6: Difference between the test and training error rates and the test error rate as a function of the complexity T of the H-LSVM model. Figure 6(a) IJCNN dataset; Figure 6(b) Faces dataset.

with a modified version of the Pegasos algorithm with weighted patterns.

Here, we provide a description of the H-LSVM algorithm, an upper bound of the H-LSVM generalization error and an analysis of the H-LSVM prediction cost compared with those of linear SVM and non-linear SVM. The experiments carried out in medium and large datasets show that the H-LSVM algorithm improves the classification accuracy of linear SVMs. Compared with the existing methods based on the construction of a decision tree with linear SVMs as splitting criteria, the H-LSVM model is superior in terms of classification accuracy while maintaining a classification complexity of the same order of magnitude.

In summary, the H-LSVM method is an attempt at a solution to the problem of applying SVM technology to industrial settings with high loads in real-time classification. In online industrial environments when decisions have to be taken in a hundredth of a second, non-linear SVMs are just impossible to apply. H-LSVMs may bridge this gap because it is simple and efficient.

Acknowledgements

The authors would like to thank the anonymous reviewers for their comments that help improve the manuscript. I.R.-L. is supported by an FPU grant from Universidad Autónoma de Madrid, and partially supported by the Universidad Autónoma de Madrid-IIC Chair and TIN 2010-21575-C02-01. R.H. acknowledges partial support by ONR N00014-07-1-0741, USARIEM-W81XWH-10-C-0040 (ELINTRIX) and JPL-1396686.

References

[1] I. Steinwart, Sparseness of support vector machines—some asymptotically sharp bounds, Neural Information Processing Systems 16 (2004) 1069–1076.

[2] S. S. Keerthi, O. Chapelle, D. DeCoste, Building support vector machines with reduced classifier complexity., Journal of Machine Learning Research 7 (2006) 1493–1515.

[3] C. J. C. Burges, B. Schölkopf, Improving the accuracy and speed of support vector learning machines, in: Advances in Neural Information Processing Systems 9, MIT Press, Cambridge, MA, 1997, pp. 375–381.

[4] T. Downs, K. E. Gates, A. Masters, Exact simplification of support vector solutions, Journal of Machine Learning Research 2 (2002) 293–297.

[5] E. E. Osuna, F. Girosi, Reducing the run-time complexity in support vector machines, Advances in kernel methods, MIT Press, Cambridge, MA, USA, 1999, pp. 271–283.

[6] M. Wu, B. Schölkopf, G. H. Bakir, Building sparse large margin classifiers., in: ICML '05: Proceedings of the 22th international conference on Machine learning, Vol. 119, 2005, pp. 996–1003.

[7] R. E. Schapire, Y. Singer, Improved boosting algorithms using confidence-rated predictions, Machine Learning 37 (3) (1999) 297–336.

[8] L. Breiman, Bagging predictors, Machine Learning 24 (2) (1996) 123–140.

[9] R. Collobert, S. Bengio, Y. Bengio, A parallel mixture of SVMs for very large scale problems., Neural Computation 14 (5) (2002) 1105–1114.

[10] H. P. Graf, E. Cosatto, L. Bottou, I. Durdanovic, V. Vapnik, Parallel support vector machines: The cascade SVM, in: Advances in Neural Information Processing Systems, MIT Press, 2005, pp. 521–528.

[11] C. G. X. L. F.Chang, C. Lu, Tree decomposition for large-scale SVM problems, Journal of Machine Learning Research 11 (2010) 2935–2972.

[12] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, Classification and Regression Trees, Chapman & Hall, New York, NY, 1984.

[13] Y. Singer, N. Srebro, Pegasos: Primal estimated sub-gradient solver for SVM, in: In ICML, 2007, pp. 807–814.

[14] A. Bordes, L. Bottou, P. Gallinari, Sgd-qn: Careful quasi-newton stochastic gradient descent, Journal of Machine Learning Research 10 (2009) 1737–1754.

[15] X. Zhou, N. Cui, Z. Li, F. Liang, T. S. Huang, Hierarchical gaussianization for image classification., in: ICCV, IEEE, 2009, pp. 1971–1977.

[16] K. Yu, T. Zhang, Y. Gong, Non-linear learning using local coordinate coding, in: Advances in Neural Information Processing Systems 22, 2009, pp. 2223–2231.

[17] L. Ladicky, P. Torr, Locally Linear Support Vector Machines, in: ICML '11: Proceedings of the 28th international conference on Machine learning, 2011, pp. 985–992.

[18] Z. Fu, A. Robles-Kelly, J. Zhou, Mixing linear SVMs for nonlinear classification., IEEE Transactions on Neural Networks 21 (12) (2010) 1963–1975.

[19] K. P. Bennett, N. Cristianini, J. Shawe-Taylor, D. Wu, Enlarging the margins in perceptron decision trees, Machine Learning 41 (2000) 295–313.

[20] M. Arun Kumar, M. Gopal, A hybrid SVM based decision tree, Pattern Recognition 43 (2010) 3977–3987.

[21] K. Z. Arreola, J. Fehr, H. Burkhardt, Fast support vector machine classification using linear SVMs., in: ICPR (3), IEEE Computer Society, 2006, pp. 366–369.

[22] M. Anthony, Generalization error bounds for threshold decision lists, Journal of Machine Learning Research 5 (2004) 189–217.

[23] J. R. Quinlan, R. L. Rivest, Inferring decision trees using the minimum description length principle, Information and Computation 80 (3) (1989) 227–248.

[24] J. Fehr, K. Z. Arreola, H. Burkhardt, Fast support vector machine classification of very large datasets., in: GfKl, Studies in Classification, Data Analysis, and Knowledge Organization, Springer, 2007, pp. 11–18.

[25] J. Su, G. Wang, Q. Hu, S. Li, A novel SVM decision tree and its application to face detection., in: SNPD (1), IEEE Computer Society, 2007, pp. 385–389.

[26] V. Menkovski, I. Christou, S. Efremidis, Oblique decision trees using embedded Support Vector Machines in classifier ensembles, in: 7th IEEE International Conference on Cybernetic Intelligent Systems, 2008. CIS 2008., 2008, pp. 1–6.

[27] J. R. Quinlan, C4.5: programs for machine learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[28] J. C. Platt, Fast training of support vector machines using sequential minimal optimization, in: Advances in Kernel Methods: Support Vector Learning, MIT Press, Cambridge, MA, 1998, pp. 185–208.

[29] K. P. Bennett, J. Blue, A support vector machine approach to decision trees, in: The 1998 IEEE International Joint Conference on Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence., 1998.

[30] Y. Ben-Haim, E. Tom-Tov, A streaming parallel decision tree algorithm., Journal of Machine Learning Research 11 (2010) 849–872.

[31] S. Viaene, R. A. Derrig, G. Dedene, Cost-sensitive learning and decision making for massachusetts pip claim fraud data., International Journal of Intelligent Systems 19 (12) (2004) 1197–1215.

[32] S.-B. Park, S. Hwang, B.-T. Zhang, Mining the risk types of human papillomavirus (hpv) by adacost., in: DEXA, Vol. 2736 of Lecture Notes in Computer Science, Springer, 2003, pp. 403–412.

[33] Y. Huang, S. Du, Weighted Support Vector Machine for classification with uneven training class sizes, in: Proceedings of 2005 International Conference on Machine Learning and Cybernetics, Vol. 7, 2005, pp. 4365–4369.

[34] J. Quinlan, Simplifying decision trees., International Journal of Man-Machine Studies 27 (1987) 221–234.

[35] W. W. Cohen, Efficient pruning methods for separate-and-conquer rule learning systems., in: IJCAI, 1993, pp. 988–994.

[36] R. Fan, K. Chang, C. Hsieh, X. Wang, C.-J. Lin, LIBLINEAR: A library for large linear classification., Journal of Machine Learning Research 9 (2008) 1871–1874, software available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`.

[37] C. Hsieh, K. Chang, C. Lin, S. S. Keerthi, S. Sundararajan, A dual coordinate descent method for large-scale linear SVM., in: ICML '08: Proceedings of the 25th international conference on Machine learning, Vol. 307, 2008, pp. 408–415.

[38] A. K. Menon, Large-scale support vector machines: Algorithms and theory, Tech. rep., University of California, San Diego (2009).

[39] S. Shalev-Shwartz, N. Srebro, SVM optimization: inverse dependence on training set size., in: Proceedings of the 25th International Conference on Machine Learning (ICML'08), Vol. 307, 2008, pp. 928–935.

[40] M. G. Peter, I. Llew Mason, Generalization in decision trees and dnf: Does size matter?, in: Advances in Neural Information Processing Systems, The MIT Press, 1997, pp. 259–265.

[41] Y. Mansour, D. A. McAllester, Generalization bounds for decision trees., in: COLT, 2000, pp. 69–74.

[42] M. Shah, Sample compression bounds for decision trees, in: ICML, Vol. 227, 2007, pp. 799–806.

[43] M. Marchand, M. Sokolova, Learning with decision lists of data-dependent features, Journal of Machine Learning Research 6 (2005) 427–451.

[44] F. Laviolette, M. Marchand, M. Shah, S. Shanian, Learning the set covering machine by bound minimization and margin-sparsity trade-off, Machine Learning 78 (1-2) (2010) 175–201.

[45] M. Shah, M. Marchand, J. Corbeil, Feature selection with conjunctions of decision stumps and learning from microarray data, IEEE Trans. Pattern Anal. Mach. Intell. 34 (1) (2012) 174–186.

[46] R. C. Holte, Very simple classification rules perform well on most commonly used datasets, Machine Learning 11 (1993) 63–90.

[47] U. M. Fayyad, K. B. Irani, What should be minimized in a decision tree?, in: Association for the Advancement of Artificial Intelligence (AAAI), 1990, pp. 749–754.

[48] A. Ehrenfeucht, D. Haussler, M. Kearns, Learning decision trees from random examples needed for learning, Information and Computation 82 (1989) 231–246.

[49] V. N. Vapnik, Statistical Learning Theory, Wiley-Interscience, 1998.

[50] Y. Freund, R. E. Schapire, Experiments with a new boosting algorithm., in: International Conference on Machine Learning (ICML), 1996, pp. 148–156.

[51] C.-C. Chang, C.-J. Lin, LIBSVM: a library for support vector machines, software available at `http://www.csie.ntu.edu.tw/\~cjlin/libsvm` (2001).

[52] J. Fehr, K. Z. Arreola, H. Burkhardt, Fast support vector machine classification of very large datasets, Tech. rep., University of Freiburg, Department of Computer (2007).

[53] I. W. Tsang, J. T. Kwok, P.-M. Cheung, Core Vector Machines: Fast SVM Training on Very Large Data Sets., Journal of Machine Learning Research 6 (2005) 363–392.

[54] Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, in: Proceedings of the Second European Conference on Computational Learning Theory, 1995, pp. 23–37.

[55] R. E. Schapire, Y. Freund, P. Bartlett, W. S. Lee, Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods., The Annals of Statistics 26 (5) (1998) 1651–1686.

[56] P. Viola, M. Jones, Robust real-time face detection, International Journal of Computer Vision 57 (2) (2004) 137–154.

***Author Biography**

Irene Rodriguez-Lujan received her degree in Computer Engineering and Mathematics from the Universidad Autonoma de Madrid (UAM) in 2007 and the Master degree in Computer Science in 2009 from the same university. Currently she is working on her Ph.D. thesis in the UAM Computer Science Department and she collaborates with the Instituto de Ingenieria del Conocimiento (IIC). Her research interests include feature selection and classification in real-time large-scale systems.

Carlos Santa Cruz received the physics degree from the Universidad Autonoma of Madrid (UAM) in 1987. He received the Ph.D. degree in physics from the UAM in 1991. Currently, he is a Professor at the Computer Science Department of the UAM and head of the Quantitative Methods Group of the Instituto de Ingenieria del Conocimiento (IIC). His research interests are in pattern recognition, model building, and time series forecasting using neural networks.

Ramon Huerta received the B.S. and M.S. degrees in physics from the Universidad Autonoma de Madrid, Spain, and the Ph.D. degree in Computer Science from the same University in 1994. He became Associate Professor at UAM in 2000 and left on leave of absence to his current Research Scientist position at the University of California San Diego. His research interests are at the intersection of Aritificial Intelligence, Physics and Biology with more than 80 journal articles in these three fields.