Hierarchical classification using SVM

J. Díez, J.J. del Coz

Machine Learning Group - Artificial Intelligence Center
University of Oviedo at Gijón
http://www.aic.uniovi.es/MLGroup
{jdiez, juanjo}@aic.uniovi.es

Abstract. In hierarchical classifications classes are arranged in a hierarchy represented by a tree forest, and each example is labeled with a set of classes located in paths from roots to leaves or internal nodes. In other words, both multiple and partial paths are allowed. A straightforward approach to learn these classifiers consists in learning one binary classifier per node of the hierarchy; the hierarchical classifier is then obtained using a top-down evaluation procedure. In this paper, we present a new approach where node classifiers are learned by binary SVMs weighted according to the hierarchy structure and the loss function used to measure the goodness of the classifiers. The result is a collection of modular algorithms that are competitive with state-of-the-art approaches. Moreover, the benefits of the modularity include the possibility of parallel implementations, and the use of all available and well-known techniques to tune binary classification SVMs.

1 Introduction

Many real-world domains require automatic systems to organize objects into known taxonomies. For instance, a news website, or a news service in general, needs to classify latest articles into sections and subsections of the site [1, 2, 3, 4, 5]. Although most applications deal with textual information, in [6, 7] it is described an algorithm to classify speech data into a hierarchy of phonemes.

This learning task is usually called *hierarchical classification* and differs from multiclass learning in: 1) the whole set of classes has a hierarchical structure usually defined by a tree, and 2) each object must be labeled with a set of classes consistent with the hierarchy: if an object belongs to a class, then it must belong to any of its ancestors.

The aim of hierarchical classification algorithms is to learn a model that can accurately predict a set of classes; notice that these subsets have in general more than one element, and they are endowed with a subtree structure. These subtrees may have more than one branch, in this case we say that there are *multipaths* in the labels, and subtrees may not end on a leaf, that is they include *partial paths*.

Roughly speaking, the algorithms available in the literature can be arranged in two groups: those that take a *local* point of view, and those that learn a model from a *global* perspective. Local algorithms learn a model for each node of the hierarchy using different approaches; a hierarchical classification of an object is then obtained by evaluating local classifiers in a top-down procedure until a model fails to include that node on its attached class. The algorithms presented in [1, 6, 7, 3] belong to this group. On the other hand, the hierarchical classification can be seen as a whole rather than a series of local learning tasks. The idea is to optimize the global performance all at once. This position is adopted in [2, 4, 5].

In this paper we will present a learning algorithm for hierarchical classification with multiple and partial paths. The algorithm adopts a local strategy; the motivations are the following. First, somehow hierarchical learning is similar to multiclass classification, and in this context the combination of binary (local in this case) classifiers obtain similar results that those attained by approaches that try to learn a model to classify at the same time all classes [8]. Second, it is not enough clear how the global performance of a hierarchical classifier can be benefited by a bad performance of the local classifier attached to a node of the hierarchy, specially in multiple partial path situations. Third, in several papers about hierarchical classifications a *simple* combination of SVM (frequently called HSVM) is used as baseline algorithm in comparisons; the results are not too unfavorable for this baseline.

In this paper we improve the vanilla version of HSVM in a couple of directions. We study the options available to decompose hierarchical classifications into a set of binary SVM, one for each node (class) of the hierarchy. And we add some weighting schemes that improve the performance of the classifiers to reach a very high level. In addition to the performance obtained, the advantages of local algorithms for hierarchical classification are derived from their modularity. They can be straightforwardly implemented in a parallel platform to obtain a very fast learning method. They are simple and can be built with one's favorite SVM, with some easy adaptations; therefore, we can improve the overall performance of the classifier using well-known techniques available to tune binary SVMs.

The paper is organized as follows. The next section introduces formally hierarchical learning and the notation used throughout the rest of the paper. The third section is devoted to explain in detail the learners devised from the decomposition strategies and weighting schemes. Finally, the last section reports some experiments with benchmark and artificial datasets conducted to compare the algorithms presented in this paper with other state-of-the-art algorithms.

2 Hierarchical classification

In hierarchical classification we have a set of classes arranged according to a known taxonomy. Formally, we have a tree $\mathcal{T}$ with r nodes (one for each class). In fact, we could start from a forest of trees $\mathcal{F}$, but then we would add an artificial root node to joint in a tree the whole set of classes; therefore, in the following we will consider that our hierarchy is represented by a tree $\mathcal{T}$. In this context, training tasks are defined by a training set $\mathcal{S} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$, where each example is described by an entry represented by a vector $\boldsymbol{x}_i$ of an input space $\mathcal{X}$, and a vector $\boldsymbol{y}_i$ of an output space $\mathcal{Y} \subset \{-1, +1\}^r$. We will interpret each output $\boldsymbol{y}_i$ as a subset of the set of classes $\{1, \ldots, r\}$: $y_{ij} = +1$ if and only if the i^{th} example belongs to the j^{th} class. In the following, we will use the symbol $\boldsymbol{y}_i$ both as a vector and as a subset when no confusion can arise. We will assume that all sets of classes of $\mathcal{Y}$ respect the underlying hierarchy:

$$\forall \boldsymbol{y}_i \in \mathcal{Y}, \quad y_{ij} = +1 \Rightarrow \forall k \in anc(j), \quad y_{ik} = +1; \tag{1}$$

where $anc(j)$ stands for the set of *ancestors* of node or class j including j.

A straightforward approach to learn these kind of tasks may consists in learning a family of models $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_r\}$ one for each node (class) of $\mathcal{T}$. Then, an entry $\boldsymbol{x}$ will be assigned to all classes j such that $(+1 = \text{sign}(\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle))$[1]. But this procedure may lead to inconsistent predictions with respect to $\mathcal{T}$. To avoid them, as in [3], a top-down prediction procedure can be used. So, an entry can only be assigned to a class j if previously it was classified into its class parent, $par(j)$; therefore, an entry not assigned to a class, *automatically*, will not be assigned to any of its descendants.

2.1 Loss functions

To complete the specification of the hierarchical learning task, we need to decide which loss function will be used to measure the goodness of the hypothesis learned. A first option may be to employ the zero-one loss:

$$l_{0/1}(\boldsymbol{y}, \boldsymbol{y}') = [\boldsymbol{y} \neq \boldsymbol{y}']. \tag{2}$$

The problem is that using this loss function it is not possible to capture any difference between very wrong predictions and nearly correct ones. Thus, in a real-world application hopefully, an expert in the field could provide us with a antisymmetric square loss matrix L where each component L_{jk} expresses the cost of classifying an object of class j as an object of class k. In practice, at most the available information could be reduced to have, for each class j, the costs of having false positives $(fp(j))$ and false negatives $(fn(j))$ [2]. Thus, we can define

$$l(\boldsymbol{y}, \boldsymbol{y}') = \sum_{j \in \boldsymbol{y} - \boldsymbol{y}'} fn(j) + \sum_{j \in \boldsymbol{y}' - \boldsymbol{y}} fp(j). \tag{3}$$

Nevertheless, in the experiments reported below, as in [4, 5, 2] we always assume that all costs have value 1, and so we obtain a loss function that only reflects the cardinal of the symmetric difference of a pair of subsets of classes: the number of different elements. In symbols:

$$l_{\Delta}(\boldsymbol{y}, \boldsymbol{y}') = \sum_{j=1}^{r} [y_j \neq y'_j] = |(\boldsymbol{y}' - \boldsymbol{y}) \cup (\boldsymbol{y} - \boldsymbol{y}')| = |\boldsymbol{y}' \ominus \boldsymbol{y}|. \tag{4}$$

[1] For ease of reading, we omit the threshold terms b_j; in any case, they can be easily included by adding an additional feature of constant value to each $\boldsymbol{x}_i$

In [4], Rousu et al. have proposed other loss functions, weighting the classes according to the proportion of hierarchy that is in the subtree $T(j)$ rooted by j, or sharing the relevance of each node between its siblings starting with 1 for the root. It is easy to see that these loss functions are particular cases of the framework presented in this section.

3 HSVM: Hierarchical SVM

In this section we are going to present an approach to learn hierarchical classifications based on the use of weighted binary classifications. Following the notation of the previous section, we assume that we have available a learning task specified by a training set $\mathcal{S}$ and real functions fp and fn to compute the costs of false positives and negatives of classes respectively. First we will discuss possible strategies to decompose the learning task into *local* binary classifications: one for each class. Then we will detail two ways to specify weighted versions according to the structure of the hierarchy and the loss function.

3.1 Training datasets for binary classification tasks of nodes

If the models $\boldsymbol{w}_j$ are going to be learned from binary classification tasks, we must specify the set of training examples that must be used. Basically, when we try to learn a model for class j we have three different options:

- **All** entries of $\mathcal{S}$ will be considered and, like in multiclass learning when we are using the strategy one-vs-rest, the subset of positive (*POS*) and negative (*NEG*) examples will be given by
$$POS = \{\boldsymbol{x}_i : y_{ij} = +1\} \qquad NEG = \{\boldsymbol{x}_i : i = 1, \dots, n\} - POS. \tag{5}$$
- We learn to distinguish between those examples that belongs to j or to any of its **siblings**. Formally,
$$POS = \{\boldsymbol{x}_i : y_{ij} = +1\} \qquad NEG = \{\boldsymbol{x}_i : y_{i,par(j)} = +1\} - POS. \tag{6}$$
- The same idea of the preceding option, but we will consider only those examples that, following the top-down **prediction** strategy, have been assigned to the parent of j. We assume that all examples belong to the root class in any case. Thus, in symbols
$$\begin{aligned} POS &= \{\boldsymbol{x}_i : y_{ij} = +1 \wedge \langle \boldsymbol{w}_{par(j)}, \boldsymbol{x}_i \rangle > 0\} \\ NEG &= \{\boldsymbol{x}_i : \langle \boldsymbol{w}_{par(j)}, \boldsymbol{x}_i \rangle > 0\} - POS. \end{aligned} \tag{7}$$

Apparently, the third choice could be more adequate than the other two, since it follows a similar procedure during learning and prediction stages. On the other hand, it produces a slower learning phase than that of the other options given that they can compute all necessary models in parallel, while the third option must wait for parent models to fix the datasets. Notice that, the fastest choice is the second one because training datasets are smaller and permit the biggest degree of parallelism.

3.2 Costs of training examples for binary classifications

For each node or class j different from the root of the hierarchy, we have defined a binary classification task in section 3.1. Despite the strategy followed to define the subset of positive and negative entries, now we are going to define a cost s_i for each example $\boldsymbol{x}_i$ of $POS \cup NEG$. The aim is to learn a discrimination model $\boldsymbol{w}_j$ where the impact of each entry will be proportional to that cost. In other words, we solve the optimization problem (see, for instance, [9])

$$\begin{aligned} \min \ & \frac{1}{2}\langle \boldsymbol{w}_j, \boldsymbol{w}_j \rangle + C \sum_{\boldsymbol{x}_i \in POS \cup NEG} s_i \cdot \xi_{ij}, \\ \text{s.t. } & y_{ij}\langle \boldsymbol{w}_j, \boldsymbol{x}_i \rangle \geq 1 - \xi_{ij}, \quad \xi_{ij} \geq 0, \quad \forall i : \boldsymbol{x}_i \in POS \cup NEG. \end{aligned} \tag{8}$$

The costs will be different for positive and negative entries depending of the class of the example and the target class j. So, if $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{S}$ is such that $\boldsymbol{x}_i$ is in POS for class j, we define

$$s_i = \sum_{k \in T(j) \cap \boldsymbol{y}_i} fn(k), \tag{9}$$

where $T(j)$ is the subtree of the hierarchy rooted by j. The idea is to highlight the number of classes of $\boldsymbol{x}_i$ that conceptually belong to class j, that is, the classes of $\boldsymbol{y}_i$ placed at or below j. On the other hand, for negative entries, we define

$$s_i = \sum_{k \in anc(j) - \boldsymbol{y}_i} fp(k). \tag{10}$$

Given that the members of class j must belong to all ancestors of j, the idea of Eq. 10 is to penalize misclassifications of entries that are not only negatives for class j, but also for some of its ancestors.

The graphs of Figure 1 illustrate with an example the assignment of costs detailed above.

3.3 Balanced learning

In the optimization problem Eq. 8, POS and NEG sets may be very unbalanced, for instance, when we are using the first decomposition strategy of section 3.1, or if the number of siblings and their descendants is high. Thus, to adjust the weight of false positives and false negatives, following [10] we set a new optimization problem

$$\begin{aligned} \min \ & \frac{1}{2}\langle \boldsymbol{w}_j, \boldsymbol{w}_j \rangle + C^+ \sum_{i:\boldsymbol{x}_i \in POS} s_i \cdot \xi_{ij} + C^- \sum_{i:\boldsymbol{x}_i \in NEG} s_i \cdot \xi_{ij}, \\ \text{s.t. } & y_{ij}\langle \boldsymbol{w}_j, \boldsymbol{x}_i \rangle \geq 1 - \xi_{ij}, \quad \xi_{ij} \geq 0, \quad \forall i : \boldsymbol{x}_i \in POS \cup NEG \end{aligned} \tag{11}$$

where, using Eq. 9 and Eq. 10,

$$C^+ = \sum_{i:\boldsymbol{x}_i \in NEG} s_i, \qquad C^- = \sum_{i:\boldsymbol{x}_i \in POS} s_i. \tag{12}$$

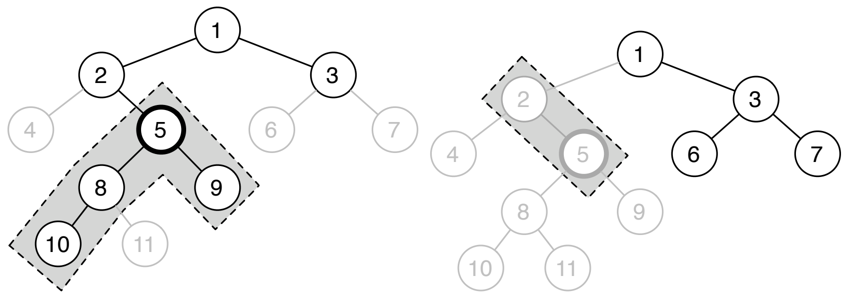

Fig. 1. If we are trying to learn a model for class 5 ($\boldsymbol{w}_5$), the cost of a *positive* example $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ with $\boldsymbol{y}_i = \{1, 2, 3, 5, 8, 9, 10\}$ (see graph (a)) is the sum of penalties for making false negatives classes $\{5, 8, 9, 10\}$. On the other hand, the cost of a *negative* $(\boldsymbol{x}_i, \{1, 3, 6, 7\})$, see graph (b), is the sum penalization (false positives) due to $\{2, 5\}$

The idea is to balance the potential total cost of false positives and false negatives. Then the ratio of the regularization parameters C^+ and C^- must be inverse to the ratio of class sizes, where the *size* of the examples is its cost.

Additionally, we investigated if the possible benefits of balancing (as stated in Eq. 12) were due to the presence of costs or they were a consequence of the balancing itself. Thus we will use, in the experiments reported in the next section, a version with all costs $s_i = 1$ for all i.

4 Experimental results

For testing the hierarchical SVMs presented in this paper, we have carried out some experiments in order to compare their performance with two state-of-the-art learners that use different approaches. The algorithms used were H-M^3-l_Δ of Rousu et al. [4], and HRLS of Cesa-Bianchi et al. [3]; while the first one searches for a global optimum loss (in this case the loss function l_Δ of Eq. 4), HRLS takes a local point of view and it incrementally learns a linear-threshold for each node of the hierarchy of classes.

The comparison was done with a couple of benchmark information retrieval (IR) datasets. Thus, in addition to the loss functions $l_{0/1}$ (Eq. 2), l_Δ (Eq. 4), we employed the performance measures: *precision*, *recall*, and *F1*.

The implementation of H-M^3-l_Δ was provided by the author, and the scores of HRLS were taken from [4], where it is reported that the algorithm was implemented according to published descriptions of its authors. The implementation of the versions of HSVM were done modifying slightly Joachims' SVM^{perf} [11]; this SVM implementation provided us with an excellent base due to its linear complexity. In all cases, when it was needed, we set the regularization parameter $C = 1$ and we used a linear kernel.

We tested two types of HSVMs. On the first hand we compared each of the three options (detailed in section 3.1) to decompose hierarchical classifications into a series of binary classifications tasks. So we used the subscript a for the option that uses *a*ll entries of the training set in all cases; s stands for the option that distinguishes between *s*iblings; and finally p means the third option, where only are considered examples *p*redicted to belong to parent classes. The second dimension of versions was the kind of weighting scheme used. Thus, we built versions without any weighting at all, or using only costs (c) (Eq. 8), or using costs balancing (cb) (Eq. 11); we finally tested a weighted scheme that uses balancing without costs, that is, with all costs $s_i = 1$ (b).

Additionally, in order to test the performance of HSVMs in the presence of an increasing percentage of examples with multipath, we built a collection of artificial datasets specially devised for that purpose.

4.1 Results on IR datasets

We used two well-known datasets in information retrieval[2]. The documents were represented as *bag-of-words* and the features were scaled with their *inverse document frequency*, in fact with the variant TFIFD [12, 13].

From the first dataset, REUTERS Corpus Volume 1 (RCV1) [14], following [4], 7500 documents were collected and 5000 were separated for testing purposes. The 'CCAT' family of categories (Corporate/Industrial news articles) was used as the label hierarchy. This hierarchy represents a tree with maximum depth 3 and with a total of 34 nodes. The tree is quite unbalanced: there are 18 nodes residing in depth 1, 14 nodes in depth 2 and one node in depth 3. In this dataset, approximately 8% of examples have multiple partial paths; that is, these examples were classified into classes that are not in the same path from the root.

The WIPO-alpha was published by the World Intellectual Property Organization [15], and it is the second dataset used in these experiments. We used D section of the hierarchy. This section contains 1730 documents, and 358 of them were separated as test set. There are 188 nodes in the tree organized as follows: 7 in depth 1, 20 in depth 2, and 160 in depth 3. In this dataset there are no examples classified into more than one path.

The left hand side of Table 1 depicts the scores obtained in RCV1 in the conditions described above. The key loss is l_Δ, since this is the optimization target; on the other hand, from an IR point of view *F1* is the most relevant measure. In both measures HRLS outperforms H-M^3-l_Δ . In the field of HSVM variants, s and p versions achieve better results than the one-vs-rest (a) variants; however, HSVM$_{a,b}$ reaches the scores of H-M^3-l_Δ in l_Δ and slightly better in *F1*. With respect to weighting schemes, balances (b) seem a bit better than costs (c), although with similar scores. In l_Δ both HSVM$_s$ and HSVM$_p$ outperform H-M^3-l_Δ , even without any weighting; to reach the score of HRLS we need the balanced releases. In *F1* any weighting outperform both HRLS and H-M^3-l_Δ .

[2] These datasets can be downloaded from http://users.ecs.soton.ac.uk/cjs/downloads

Table 1. Prediction losses $l_{0/1}$, l_Δ, precision P, recall R and *F1* on REUTERS and WIPO datasets. All scores are given as percentages, but the values of the column labeled by l_Δ that are the average of Eq. 4 across the test set

	REUTERS Corpus Vol. 1					WIPO-alpha				
	$l_{0/1}$	l_Δ	P	R	$F1$	$l_{0/1}$	l_Δ	P	R	$F1$
HSVM$_a$	32.9	0.612	94.7	58.3	72.2	86.0	1.830	93.2	58.5	71.9
HSVM$_{a,c}$	34.0	0.599	92.9	60.5	73.3	85.2	1.682	91.8	63.6	75.2
HSVM$_{a,b}$	30.2	0.574	90.5	64.6	75.4	71.5	1.606	88.6	68.7	77.4
HSVM$_{a,cb}$	30.5	0.581	89.9	64.5	75.1	73.2	1.615	88.5	68.6	77.3
HSVM$_s$	29.8	0.571	92.2	63.4	75.1	76.3	1.743	90.5	63.1	74.3
HSVM$_{s,c}$	29.8	0.555	89.6	66.9	76.6	71.2	1.592	88.1	69.6	77.8
HSVM$_{s,b}$	27.8	0.553	88.4	68.3	77.0	66.8	1.637	85.4	71.2	77.7
HSVM$_{s,cb}$	28.0	0.563	87.7	68.1	76.7	66.8	1.634	85.3	71.4	77.8
HSVM$_p$	29.9	0.573	92.5	63.0	74.9	76.8	1.749	90.5	62.8	74.2
HSVM$_{p,c}$	30.0	0.556	90.3	66.2	76.4	71.0	1.578	88.7	69.3	77.9
HSVM$_{p,b}$	27.8	0.550	88.9	68.0	77.1	66.8	1.634	85.8	70.9	77.6
HSVM$_{p,cb}$	28.1	0.560	88.1	68.0	76.7	66.5	1.634	85.5	71.2	77.7
H-M^3-l_Δ	27.1	0.574	91.0	64.1	75.2	70.9	1.670	90.3	65.3	75.8
HRLS	28.1	0.550	91.5	65.4	76.3	72.1	1.690	88.5	66.4	75.9

The scores in WIPO-alpha dataset are shown in the right hand side of Table 1. We can see bigger differences in this dataset than in RCV1. Again, versions of HSVM that use costs and/or balances attain better scores than those without any weighting, and better than H-M^3-l_Δ and HRLS . In contrast with RCV1 results, in this case, the best result is found in HSVM$_{p,c}$, not in HSVM$_{p,b}$.

In general we see that the scores of HSVM$_s$ and HSVM$_p$ are quite similar, although the later seems that is able to slightly correct some mistakes of HSVM$_s$. The strategy one-vs-rest (a) usually performs worse than s and p. With respect to weighting schemes, they reveal very useful both in costs or balances.

4.2 Artificial datasets

In order to test the ability of our approach to handle multiple paths, we devised an artificial dataset following a tree hierarchy. The tree has 44 nodes organized as follows: 3 in depth 1, 7 in depth 2, 9 in depth 3, 12 in depth 4, and 12 in depth 5. There are leaves at any depth, but in depth 0 and 1. All non-leaf-nodes have 2 or 3 children. We intended to simulate, in the artificial examples, a representation of the type *bag-of-words*. Therefore, we associated a number ρ_j of features to each node j; and the values in features were generated following a uniform distribution between -1 and $+1$, but negative values were eliminated to achieve a sparse dataset. Examples were assigned to a node when the sum of the features attached to it exceed a given threshold μ_j. We force the positive

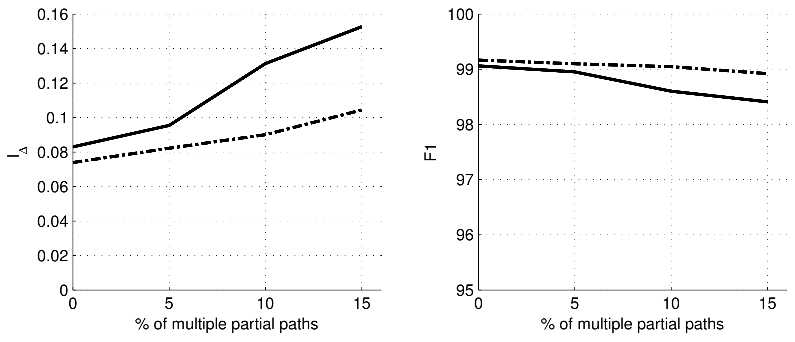

Fig. 2. Performance of the HSVM$_{a,cb}$ (solid line) and HSVM$_{s,cb}$ (dash-dot line) when the percentage of multiple partial paths in examples is increased. HSVM$_{p,cb}$ and HSVM$_{s,cb}$ achieved exactly the same results

attribute values in order to ensure that we have 40 examples per node for training set and 20 per node for test set. We created artificial datasets with 0, 5, 10 and 15 pecentage of examples with multiple partial paths[3]. The parameters used in the experiment were: $\rho_j = 50$ and $\mu_j = \rho_j/0.7$ for all j.

Figure 2 shows the performance of HSVM$_{a,cb}$, HSVM$_{s,cb}$, and HSVM$_{p,cb}$, when the percentage of multiple partial paths is increased. HSVM$_{s,cb}$, and HSVM$_{p,cb}$ reached exactly the same scores, and as it could be expected, they had better performance than HSVM$_{a,cb}$. It is important to notice that HSVM$_{s,cb}$ (and HSVM$_{p,cb}$) increased its l_Δ in only 0.0305 classes failed per example, while the percentage of multiple partial paths had been increased from 0% to 15%. HSVM$_{a,cb}$ had also a good performance, since its l_Δ only was incremented in 0.0697. On the other hand, the behavior in $F1$ was quite parallel: HSVM$_{s,cb}$ and HSVM$_{p,cb}$ outperform HSVM$_{a,cb}$, but all had good resistance to the increase in the percentage of multipaths.

5 Conclusions

We have presented a learning method for hierarchical classifications tasks based on the use of learning *local* binary classifications: one for each node of the hierarchy. This approach is well known and it is acknowledged to achieve competitive results with other approaches [1, 3]. In the experiments reported in section 4, we also confirm the good scores of this *naïve* approach. The novelty of this paper is the use for hierarchical classification of weighting schemas in binary classifiers.

We have studied two aspects: the decomposition of hierarchical tasks into binary classifications, and the definitions of weights suggested by both the hierarchy and the loss function. The conclusion is that it is possible to obtain a

[3] A Matlab function for generating artificial datasets in this way can be downloaded from (http://www.aic.uniovi.es/MLGroup/DataSets/GenerateArtificial.m)

very competitive learning method, comparable with state-of-the-art algorithms [3, 4].

Additionally, in general the decomposition strategies have the advantage of modularity. So, it is possible to have fast parallel implementations for hierarchical classifications. Besides, each binary classification could be learned using well-known SVM implementations, and one could tune the regularization parameters or the kernels employed. Notice that if the application field is different from textual information, the choice of the right kernel may be a crucial point in the performance of any classification task.

Acknowledgments The research reported in this paper has been supported in part under Spanish Ministerio de Educación y Ciencia (MEC) grant TIN2008-06247. The authors would like to thank Juho Rousu for the source code of his algorithm and Thorsten Joachims for the source code of SVM$_{light}$ and SVM_{Perf}.

References

[1] Dumais, S.T., Chen, H.: Hierarchical classification of web content. In: Proceedings of the 23rd SIGIR, NY, USA, ACM Press (2000) 256–263

[2] Cai, L., Hofmann, T.: Hierarchical document categorization with support vector machines. In: Proceedings of the 13th CIKM, NY, USA, ACM Press (2004) 78–87

[3] Cesa-Bianchi, N., Gentile, C., Zaniboni, L.: Incremental algorithms for hierarchical classification. JMLR **7** (2006) 31–54

[4] Rousu, J., Saunders, C., Szedmak, S., Shawe-Taylor, J.: Kernel-based learning of hierarchical multilabel classification models. JMLR **7** (2006) 1601–1626

[5] Cai, L., Hofmann, T.: Exploiting known taxonomies in learning overlapping concepts. In: Proceedings of the 20th IJCAI. (2007) 708–713

[6] Dekel, O., Keshet, J., Singer, Y.: Large margin hierarchical classification. In: Proceedings of the 21st ICML. (2004) 209–216

[7] Dekel, O., Keshet, J., Singer, Y.: An efficient online algorithm for hierarchical phoneme classification. In: Proceedings of 1st International Workshop on Machine Learning for Multimodal Interaction. (2005) 146–158

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

[9] Fan, H., Ramamohanarao, K.: A weighting scheme based on emerging patterns for weighted support vector machines. In: Proceedings of IEEE International Conference on Granular Computing. (2005) 435– 440

[10] Morik, K., Brockhausen, P., Joachims, T.: Combining statistical learning with a knowledge-based approach - a case study in intensive care monitoring. In: Proceedings of the ICML. (1999) 268–277

[11] Joachims, T.: Training linear svms in linear time. In: Proceedings of the 12th SIGKDD, NY, USA, ACM Press (2006) 217–226

[12] Joachims, T.: Text categorization with support vector machines: learning with many relevant features. In: Proceedings of 10th ECML, (1998) 137–142

[13] Salton, G., Buckley, C.: Term weighting approaches in automatic text retrieval. Information Processing and Management **24**(5) (1988) 513–523

[14] Lewis, D.D., Yang, Y., Rose, T.G., Li, F.: Rcv1: A new benchmark collection for text categorization research. JMLR **5** (2004) 361–397

[15] WIPO: http://www.wipo.int/classifications/en (2001)