Journal of Machine Learning Research 7 (2006) 1601–1626 Submitted 10/05; Published 7/06

Kernel-Based Learning of Hierarchical Multilabel Classification Models*

Juho Rousu JUHO.ROUSU@CS.HELSINKI.FI
Department of Computer Science
PO Box 68
FI-00014 University of Helsinki, Finland

Craig Saunders CJS@ECS.SOTON.AC.UK
Sandor Szedmak SS03V@ECS.SOTON.AC.UK
John Shawe-Taylor JST@ECS.SOTON.AC.UK
Electronics and Computer Science
University of Southampton
SO17 1BJ, United Kingdom

Editors: Kristin P. Bennett and Emilio Parrado-Hernández

Abstract

We present a kernel-based algorithm for hierarchical text classification where the documents are allowed to belong to more than one category at a time. The classification model is a variant of the Maximum Margin Markov Network framework, where the classification hierarchy is represented as a Markov tree equipped with an exponential family defined on the edges. We present an efficient optimization algorithm based on incremental conditional gradient ascent in single-example subspaces spanned by the marginal dual variables. The optimization is facilitated with a dynamic programming based algorithm that computes best update directions in the feasible set.

Experiments show that the algorithm can feasibly optimize training sets of thousands of examples and classification hierarchies consisting of hundreds of nodes. Training of the full hierarchical model is as efficient as training independent SVM-light classifiers for each node. The algorithm's predictive accuracy was found to be competitive with other recently introduced hierarchical multi-category or multilabel classification learning algorithms.

Keywords: kernel methods, hierarchical classification, text categorization, convex optimization, structured outputs

1. Introduction

In many application fields, taxonomies and hierarchies are natural ways to organize and classify objects, hence they are widely used for tasks such as text classification. In contrast, machine learning research has largely been focused on flat target prediction, where the output is a single binary or multivalued scalar variable. Naively encoding a large hierarchy either into a series of binary problems or a single multiclass problem with many possible class values suffers from the fact that dependencies between the classes cannot be represented well. For example, if a news article belongs to category MUSIC, it is very likely that the article belongs to category ENTERTAINMENT. The failure to represent these relationships leads to a steep decline of the predictive accuracy in the

*. A preliminary version of this paper appeared in Proceedings of 19th ICML, Bonn, Germany, 2005.

number of possible categories. In recent years, methods that utilize the hierarchy in learning the classification have been proposed by several authors (Koller and Sahami, 1997; McCallum et al., 1998; Dumais and Chen, 2000). Very recently, new hierarchical classification approaches utilizing kernel methods have been introduced (Hofmann et al., 2003; Cai and Hofmann, 2004; Dekel et al., 2004). The main idea behind these methods is to map the documents (or document–labeling pairs) into a potentially high-dimensional feature space where linear maximum margin separation of the documents becomes possible.

Most of the above mentioned methods assume that the object to be classified belongs to exactly one (leaf) node in the hierarchy. In this paper we consider the more general case where a single object can be classified into several categories in the hierarchy, to be specific, the multilabel is a *union of partial paths* in the hierarchy. For example, a news article about David and Victoria Beckham could belong to partial paths SPORT, FOOTBALL and ENTERTAINMENT, MUSIC but might not belong to any leaf categories such as CHAMPIONS LEAGUE. The problem of multiple partial paths was also considered in Cesa-Bianchi et al. (2004).

Recently Taskar et al. (2003) introduced a maximum margin technique which optimized an SVM-style objective function over structured outputs. This technique used a marginalization trick to obtain a polynomial sized quadratic program using marginal dual variables. This was an improvement over the exponentially-sized problem resulting from the dualization of the primal margin maximization problem, which only can be approximated with polynomial number of support vectors using a working set method (Altun et al., 2003; Tsochantaridis et al., 2004).

Even using marginal variables, however, the problem becomes infeasible for even medium sized data sets. Therefore, efficient optimization algorithms are needed. In this paper we present an algorithm for working with the marginal variables that is in the spirit of Taskar et al. (2003), however a reformulation of the objective allows a conditional-gradient method to be used which gains efficiency and also enables us to work with a richer class of loss functions.

The structure of this article is the following. In Section 2 we present the classification framework, review loss functions and derive a quadratic optimization problem for finding the maximum margin model parameters. In Section 3 we present an efficient learning algorithm relying a decomposition of the problem into single training example subproblems and conducting iterative conditional gradient ascent in marginal dual variable subspaces corresponding to single training examples. A dynamic programming algorithm is presented that used to efficiently find the best update directions. Extensions and variants are briefly discussed in Section 4. We compare the new algorithm in Section 5 to flat and hierarchical SVM learning approaches and the hierarchical regularized least squares algorithm recently proposed by Cesa-Bianchi et al. (2004). We conclude the article with discussion in Section 6.

2. Maximum Margin Hierarchical Multilabel Classification

We consider data from a domain $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ is a set and $\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_k$ is a Cartesian product of the sets $\mathcal{Y}_j = \{+1,-1\}, j = 1,\ldots,k$. A vector $\mathbf{y} = (y_1,\ldots,y_k) \in \mathcal{Y}$ is called the *multilabel* and the components y_j are called the *microlabels*.

We assume that a training set $\{(\mathbf{x}_i,\mathbf{y}_i)\}_{i=1}^m \subset \mathcal{X} \times \mathcal{Y}$ has been given, consisting of training examples $(\mathbf{x}_i,\mathbf{y}_i)$ of a training pattern $\mathbf{x}_i$ and multilabel $\mathbf{y}_i$. A pair $(\mathbf{x}_i,\mathbf{y})$ where $\mathbf{x}_i$ is a training pattern and $\mathbf{y} \in \mathcal{Y}$ is arbitrary, is called a *pseudo-example*, to denote the fact that the pair may or may not be generated by the distribution generating the training examples.

A HUMAN NECESSITIES
  A 01 AGRICULTURE; FORESTRY; ANIMAL HUSBANDRY; ...
    A 01 B SOIL WORKING IN AGRICULTURE OR FORESTRY
      A 01 B 1/02 Spades; Shovels
      A 01 B 9/00 Ploughs with rotary driven tools
D TEXTILES; PAPER
  D 21 PAPER-MAKING; PRODUCTION OF CELLULOSE
    D 21 F PAPER-MAKING MACHINES
      D 21 F 1/00 Wet end of machines for making continuous webs of paper

E.C.1 Oxidoreductases
  E.C.1.1. Acting on the CH-OH group of donors.
    E.C.1.1.1 With NAD(+) or NADP(+) as acceptor.
      E.C.1.1.1.1 Alcohol dehydrogenase.
E.C.6 Ligases
  E.C.6.1 Forming carbon-oxygen bonds.
    E.C.6.1.1 Ligases forming aminoacyl-tRNA and related compounds.
      E.C.6.1.1.1 Tyrosine–tRNA ligase.

Figure 1: Examples of classification hierarchies: An excerpt from the WIPO patent classification hierarchy (top) and an excerpt from the Enzyme Classification scheme (bottom).

As the model class we use the exponential family

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x},\mathbf{w})} \prod_{e \in E} \exp\left(\mathbf{w}_e^T \phi_e(\mathbf{x}, \mathbf{y}_e)\right) = \frac{1}{Z(\mathbf{x},\mathbf{w})} \exp\left(\mathbf{w}^T \phi(\mathbf{x},\mathbf{y})\right)$$

defined on the edges of a Markov tree $T = (V,E)$, where node $j \in V$ corresponds to the j'th component of the multilabel and the edges $e = (j,j') \in E$ correspond to the classification hierarchy given as input. Above, $Z(\mathbf{x},\mathbf{w}) = \sum_{\mathbf{y}} \exp\left(\mathbf{w}^T \phi(\mathbf{x},\mathbf{y})\right)$ is the normalizing factor also called the partition function. By $\mathbf{y}_e = (y_j, y_{j'})$ we denote the restriction of the multilabel $\mathbf{y} = (y_1, \ldots, y_k)$ to the edge $e = (j,j')$. By $\mathcal{Y}_e = \mathcal{Y}_j \times \mathcal{Y}_{j'}$ we denote the set of labelings of an edge $e = (j,j')$.

In this work, we assume that the Markov tree T is given a priori. This is a reasonable assumption, as hand-made hierarchies and taxonomies are frequent in applications. The ability to learn the structure from data is an important and challenging question, which is out of scope of this article (See Lafferty et al. (2004) for a study to that direction).

Figure 1 depicts examples of two hierarchical classification domains, patent classification according to the World International Patent Organization (WIPO) that is used to classify patent texts, and enzyme classification scheme (EC) used by biologists to classify amino acid sequences for enzymatic proteins.

2.1 Loss Functions for Hierarchical Multilabel Classification

There are many ways to define loss functions for multilabel classification setting, and it depends on the application which loss function is the most suitable. A few general guidelines can be set,

though. The loss function between two multilabel vectors $\mathbf{y}$ and $\mathbf{u}$ should obviously fulfill some basic conditions: $\ell(\mathbf{u},\mathbf{y}) = 0$ if and only if $\mathbf{u} = \mathbf{y}$, $\ell(\mathbf{u},\mathbf{y})$ is maximum when $\mathbf{u}_j \neq \mathbf{y}_j$ for every $1 \leq j \leq k$, and ℓ should be monotonically non-decreasing with respect to the sets of incorrect microlabels. These conditions are satisfied by, for example, *zero-one loss* $\ell_{0/1}(\mathbf{y},\mathbf{u}) = [\mathbf{y} \neq \mathbf{u}]$. However, it gives loss of 1 if the complete hierarchy is not labeled correctly, even if only a single microlabel was predicted incorrectly.

In multilabel classification, we would like the loss to increase smoothly so that we can make a difference between 'nearly correct' and 'clearly incorrect' multilabel predictions. *Symmetric difference loss*

$$\ell_\Delta(\mathbf{y},\mathbf{u}) = \sum_j [y_j \neq u_j],$$

has this property and is an obvious first choice as the loss function in structured classification tasks. However, the classification hierarchy is not reflected in any way in the loss. For *uni-category* hierarchical classification (Hofmann et al., 2003; Cai and Hofmann, 2004; Dekel et al., 2004), where exactly one of the microlabels has value 1, Dekel et al. (2004) use as a loss function the length of the path $(i_1,\cdots,i_k)$ between the the true and predicted nodes with positive microlabels $\ell_{PATH}(\mathbf{y},\mathbf{u}) = |path(i : y_i = 1, j : u_j = 1)|$. Cai and Hofmann (2004) defined a weighted version of the loss that can take into account factors such as subscription loads of nodes.

In the union of partial paths model, where essentially we need to compare a predicted tree to the true one the concept of a path distance is not very natural. We would like to account for the incorrectly predicted subtrees—in the spirit of ℓ_Δ—but taking the hierarchy into account. Predicting the parent microlabel correctly is more important than predicting the child correctly, as the child may deal with some detailed concept that the user may not be interested in; for example whether a document was about CHAMPIONS LEAGUE football or not may not relevant to a person that is interested in FOOTBALL in general. Also, for the learners point of view, if the parent class was already predicted incorrectly, we don't want to penalize the mistake in the child. A loss function that has these properties was given by Cesa-Bianchi et al. (2004). It penalizes the first mistake along a path from root to a node

$$\ell_H(\mathbf{y},\mathbf{u}) = \sum_j c_j [y_j \neq u_j \ \ \& \ \ y_h = u_h \forall h \in anc(j)],$$

where $anc(j)$ denotes the set of ancestors of node j. The coefficients $0 \leq c_j \leq 1$ are used for down-scaling the loss when going deeper in the tree. These can be chosen in many ways. One can divide the maximum loss among the subtrees met along the path. This is done by defining

$$c_{root} = 1, c_j = c_{pa(j)}/|sibl(j)|,$$

where we denoted by $pa(j)$ the immediate parent and by $sibl(j)$ the set of siblings of node j (including j itself). Another possibility is to scale the loss by the proportion of the hierarchy that is in the subtree $T(j)$ rooted by j, that is, to define

$$c_j = |T(j)|/|T(root)|.$$

In our experiments we use both the sibling and subtree scaling to re-weight prediction errors on individual nodes, these are referred to as ℓ-*sibl* and ℓ-*subtree* respectively. If we just use a uniform weighting ($c_j = 1$) in conjunction with the hierarchical loss above this is denoted as ℓ-*unif*.

Using ℓ_H for learning a model has the drawback that it does not decompose very well: the labelings of the complete path are needed to compute the loss. Therefore, in this paper we consider a simplified version of ℓ_H, namely

$$\ell_{\tilde{H}}(\mathbf{y},\mathbf{u}) = \sum_j c_j [y_j \neq u_j \ \& \ y_{pa(j)} = u_{pa(j)}],$$

that penalizes a mistake in a child only if the label of the parent was correct. This choice leads the loss function to capture some of the hierarchical dependencies (between the parent and the child) but allows us define the loss in terms of edges, which is crucial for the efficiency of our learning algorithm.

Using the above, the per-microlabel loss is divided among the edges adjacent to the node. This is achieved by defining an *edge-loss* $\ell_e(\mathbf{y}_e,\mathbf{u}_e) = \ell_j(y_j,u_j)/\mathcal{N}(j) + \ell_{j'}(y_{j'},u_{j'})/\mathcal{N}(j')$ for each $e = (j,j')$, where ℓ_j is the term regarding microlabel j, $\mathbf{y}_e = (y_j,y_{j'})$ is a labeling of the edge e and $\mathcal{N}(j)$ denotes the neighbors of node j in the hierarchy (i.e. the children of a nodes and it's parent). Intuitively, the edges adjacent to node j 'share the blame' of the microlabel loss ℓ_j. The multilabel loss (ℓ_Δ or $\ell_{\tilde{H}}$) is then written as a sum over the edges: $\ell(\mathbf{y},\mathbf{u}) = \sum_{e\in E} \ell_e(\mathbf{y}_e,\mathbf{u}_e)$.

The above described loss functions do not represent an exhaustive list of the possible ones. With probabilistic models, it is common to employ KL-divergence or negative log likelihood as the loss function (Lafferty et al., 2004). In the max-margin learning framework these types of loss functions are not applicable, as they require estimating the underlying probability distribution, e.g. to compute the log-partition function. As our central theme is efficient computation of structured prediction models, we concentrate on the above simpler formulations of loss functions.

2.2 Feature Representations for Structured Inputs

When handling input data that already comes in vector form, there is no obligation to introduce a special kernel function. The inner product of the inputs $K(\mathbf{x},\mathbf{z}) = \mathbf{x}^T\mathbf{z}$, also called the 'linear kernel', can be used. However, when using structured data such as sequences, trees or graphs, one needs to convert the structured representation to a vector form. Feature representation for structured input data have been considered in many works already (c.f. Gartner (2003)), we will concentrate to the important case of hierarchical classification of text or, in general, sequence data.

For sequences the most common feature representation is to count or check the existence of subsequence occurrences, when the subsequences are taken from a fixed index set U. Different choices for the index set and accounting for occurrences give rise to a family of feature representations and kernels. Below we review the main forms of representation for sequences and the computation kernels for such representations.

Word spectrum (Bag-of-words) kernels. In the most widely used feature representation for strings in a natural language, informally called *bag-of-words* (BoW), the index set is taken as the set of words in the language, possibly excluding some frequently occurring stop words (Salton, 1989). The representation was brought to SVM learning by Joachims (1998).

In the case of a string s containing English text, for each English word u, we define the feature value

$$\phi_u(s) = |\{j | s_j \ldots s_{j+|u|-1} = u\}|,$$

as the number of times u occurs in some position j of s. For the example text $s =$ '`The cat was chased by the fat dog`' the BoW will contain the following non-zero entries: $\phi_{\texttt{the}}(s) = 2$,

$\phi_{\texttt{dog}}(s) = 1$, $\phi_{\texttt{was}}(s) = 1$, $\phi_{\texttt{chased}}(s) = 1$, $\phi_{\texttt{by}}(s) = 1$, $\phi_{\texttt{fat}}(s) = 1$, $\phi_{\texttt{cat}} = 1$. These occurrence counts can also be weighted, for example by scaling by the inverse document frequency as is done in TFIDF weighting (c.f. Salton (1989)):

$$\phi_u(s) = |\{j|s_j \dots s_{j+|u|-1} = u\}| \times \log_2 N/N_u,$$

where N_u is the number of documents where u occurs and N is the total number of documents in the collection.

Although the dimension of the feature space may be high, computation of the BoW kernel can be efficiently implemented by scanning the two strings, constructing lists $L(s)$ and $L(t)$ of pairs (u, c_u) of word u and occurrence count c_u ordered in the lexicographical order of the substrings u, and finally traversing the two lists to compute the dot product.

Substring spectrum kernels. For strings that do not encompass a crisply defined word-structure, for example, biological sequences, a different approach is more suitable. Given an alphabet Σ, a simple choice is to take $U = \Sigma^p$, the set of strings of length p. In some cases, using a range of substring lengths $q \leq l \leq p$ may be more appropriate than picking a single length. We can define

$$U = \Sigma^q \cup \Sigma^{q+1} \cdots \cup \Sigma^p \text{ for some } 1 \leq q \leq p.$$

The most efficient approaches, working in $O(p(|s| + |t|))$ time, to compute substring spectrum kernels are based on suffix trees (Leslie et al., 2002; Vishwanathan and Smola, 2002), although dynamic programming and approaches based on the *trie* data structure also can be used Shawe-Taylor and Cristianini (2004).

The substring kernels can be generalized in many ways, for example

- *Gapped substring spectrum kernels* allow gaps in the subsequence occurrences. *Gap-weighting* can be used to down-weight substring occurrences that contain many or long gaps (Lodhi et al., 2002; Rousu and Shawe-Taylor, 2005).
- *Word or syllable alphabets* can be used in place of characters (Saunders et al., 2002; Cancedda et al., 2003).

2.3 Feature Representations for Hierarchical Outputs

When the input features are used in hierarchical classification, they need to be associated with the labelings of the hierarchy. In our setting, this is done via constructing a joint feature map $\phi : \mathcal{X} \times \mathcal{Y} \mapsto \mathcal{F}_{xy}$. There are important design choices to be made in how the hierarchical structure should reflect in the feature representation.

There are two general types of features that can be distinguished:

Global features are given by the feature map $\phi^x : \mathcal{X} \mapsto \mathcal{F}_x$. They are not tied to a particular vertex or edge but represent the structured object as a whole. For example, the bag-of-words or the substring spectrum of a document is not tied to a single class of documents in a hierarchy, but a given word can relate to different classes with different importances.

Local features, are given by a feature map $\phi^x_j : \mathcal{X} \mapsto \mathcal{F}_{xj}$ tied to a particular vertex j or edge of the structure. For example, given a structured representation of a scientific article, we can make a difference between elements occurring within the title, abstract, article body and references, and construct local feature maps for each of the components.

Given the input features, there are two basic ways by which the joint feature vector can be constructed:

Orthogonal feature representation is defined as $\phi(\mathbf{x},\mathbf{y}) = (\phi_e(\mathbf{x},\mathbf{y}_e))_{e\in E}$, so that there is a block for each edge (or vertex), which, in turn, is divided into blocks for a specific edge-labeling pairs $(e,\mathbf{u}_e)$, i.e. $\phi_e(\mathbf{x},\mathbf{y}_e) = (\phi_e^{\mathbf{u}_e}(\mathbf{x},\mathbf{y}_e))_{\mathbf{u}_e\in\mathcal{Y}_e}$.

The vector ϕ_e^u should incorporate both the x-features relevant to the edge and encode the dependency on the labeling of the edge. A simple choice is to define

$$\phi_e^{\mathbf{u}_e}(\mathbf{x},\mathbf{y}_e) = [\mathbf{u}_e = \mathbf{y}_e]\left(\phi^x(\mathbf{x})^T, \phi_e^x(\mathbf{x})^T\right)^T$$

that incorporates both the global and local features if the edge is labeled $\mathbf{y}_e = \mathbf{u}_e$, and a zero vector otherwise. Intuitively, the features are turned 'on' only for the particular labeling of the edge that is consistent with $\mathbf{y}$.

Additive feature representation is defined as

$$\phi(\mathbf{x},\mathbf{y}) = \sum_{e\in E}\sum_{\mathbf{u}_e\in\mathcal{Y}_e}[\mathbf{y}_e = \mathbf{u}_e]\phi_e^{\mathbf{u}_e}(\mathbf{x}),$$

where $\phi_e^{\mathbf{u}_e}$ contains features specific to the pair $(e,\mathbf{u}_e)$.

The orthogonal and additive feature representations differ from each other in several respects. In the orthogonal representation, global features get weighted in a context-dependent manner: some features may be more important in labeling one edge than another. Thus, the global features will be 'localized' by the learning algorithm. The size of the feature vectors grow linearly in the number of edges, which requires careful implementation if solving the primal optimization problem (1) instead of the dual. The kernel induced by the above feature map decomposes as

$$K(\mathbf{x},\mathbf{y};\mathbf{x}',\mathbf{y}') = \sum_{e\in E}\phi_e(\mathbf{x},\mathbf{y}_e)^T\phi_e(\mathbf{x}',\mathbf{y}'_e) = \sum_{e\in E}K_e(\mathbf{x},\mathbf{y}_e;\mathbf{x}',\mathbf{y}'_e),$$

which means that there is no crosstalk between the edges:

$$\phi_e(\mathbf{x},\mathbf{y}_e)^T\phi_{e'}(\mathbf{x},\mathbf{y}_{e'}) = 0$$

if $e \neq e'$, hence the name 'orthogonal'. The number of terms in the sum when calculating the kernel obviously scales linearly in the number of edges.

The dimension of the feature vector using the additive feature representation is independent of the size of the hierarchy, thus optimization in primal representation (1) is more feasible for large structures. Second, as there are no feature weights depending on a particular part of the structure, the existence of local features is mandatory, otherwise the output structure is not reflected in the feature vector. Third, the kernel

$$\begin{aligned} K(\mathbf{x},\mathbf{y};\mathbf{x}',\mathbf{y}') &= \left(\sum_e\phi_e(\mathbf{x},\mathbf{y})\right)^T\left(\sum_e\phi_e(\mathbf{x}',\mathbf{y}')\right) \\ &= \sum_{e,e'}\phi_e(\mathbf{x},\mathbf{y}_e)^T\phi_{e'}(\mathbf{x},\mathbf{y}'_{e'}) = \sum_{e,e'}K_{ee'}(\mathbf{x},\mathbf{y}_e;\mathbf{x}',\mathbf{y}'_{e'}) \end{aligned}$$

induced by this representation typically has non-zero blocks $\mathbf{K}_{ee'} \neq 0$, representing cross-talk between edges. There are two consequences of this fact. First, the kernel does not exhibit the sparsity that is implied by the hierarchy, thus it creates the possibility of overfitting. Second, the complexity of the kernel will grow quadratically in the size of the hierarchy rather than linearly as is the case with orthogonal features. This is another reason why a primal optimization approach for this representation might be more justified than a dual approach.

In the sequel, we describe a method that relies on the orthogonal feature representation which will give us a dual formulation with complexity growing linearly in the number of edges in E. The kernel defined by the feature vectors, denoted by

$$K^x(\mathbf{x},\mathbf{x}') = \phi^x(\mathbf{x})^T \phi^x(\mathbf{x}'),$$

is referred to as *x*-kernel while $K(\mathbf{x},\mathbf{y};\mathbf{x},\mathbf{y}')$ is referred to as the *joint kernel*.

2.4 Maximum Margin Learning

Typically in learning probabilistic models, one aims to learn maximum likelihood parameters, which in the exponential CRF amounts to solving

$$\text{argmax}_{\mathbf{w}} \log\left(\prod_{i=1}^{m} P(\mathbf{y}_i|\mathbf{x}_i;\mathbf{w})\right) = \text{argmax}_{\mathbf{w}} \sum_{i=1}^{m} \left[\mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i) - \log Z(\mathbf{x}_i,\mathbf{w})\right].$$

This estimation problem is hampered by the need to compute the (logarithm of the) partition function Z. For a general graph this problem is hard to solve. Approximation methods for its computation is a subject of active research (c.f. Wainwright and Jordan 2003). Also, in the absence of regularization the max-likelihood model is likely to suffer from overfitting

An alternative formulation (c.f Altun et al. 2003; Taskar et al. 2003), inspired by support vector machines, is to estimate parameters that in some sense maximize the ratio

$$\frac{P(\mathbf{y}_i|\mathbf{x}_i;\mathbf{w})}{P(\mathbf{y}|\mathbf{x}_i;\mathbf{w})}$$

between the probability of the correct labeling $\mathbf{y}_i$ and the worst competing labeling $\mathbf{y}$. With the exponential family, the problem translates to the problem of maximizing the minimum linear margin

$$\mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i) - \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y})$$

in the log-space.

Furthermore, we would like the margin γ to scale as a function of the loss so that grossly incorrect pseudo-examples are pushed farther from the correct labeling than only slightly incorrect ones. Using the canonical hyperplane representation (c.f. Cristianini and Shawe-Taylor (2000)) this can be stated as the following minimization problem:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}||\mathbf{w}||^2 \\ \text{s.t.} \quad & \mathbf{w}^T\Delta\phi(x_i,\mathbf{y}) \geq \ell(\mathbf{y}_i,\mathbf{y}), \forall i,\mathbf{y} \end{aligned}$$

where $\Delta\phi(x_i,\mathbf{y}) = \phi(x_i,\mathbf{y}_i) - \phi(x_i,\mathbf{y})$. As with SVMs, a model satisfying margin constraints exactly rarely exists, hence it is necessary to add slack variables ξ_i to allow examples to deviate from the margin boundary. Altogether, this results in the following optimization problem

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & \mathbf{w}^T\Delta\phi(x_i,\mathbf{y}) \geq \ell(\mathbf{y}_i,\mathbf{y}) - \xi_i, \forall i,\mathbf{y}. \end{aligned} \tag{1}$$

This optimization problem suffers from the possible high-dimensionality of the feature vectors, for example with string kernels, and from the exponential-sized constraint set (in the length of the multilabel vector). A dual problem

$$\max_{\alpha\geq 0} \alpha^T\ell - \frac{1}{2}\alpha^T\mathbf{K}\alpha, \text{ s.t.} \sum_{\mathbf{y}}\alpha(i,\mathbf{y}) \leq C, \forall i, \tag{2}$$

where $\mathbf{K} = \Delta\Phi^T\Delta\Phi$ is the *joint* kernel matrix for *pseudo-examples* $(x_i,\mathbf{y})$ and $\ell = (\ell(\mathbf{y}_i,\mathbf{y}))_{i,\mathbf{y}}$ is the loss vector, allows us to circumvent the problem with feature vectors. However, in the dual problem there are exponentially many dual variables $\alpha(i,\mathbf{y})$, one for each pseudo-example.

There are a few basic routes by which the exponential complexity can be circumvented:

- Dual working set methods where the constraint set is grown incrementally by adding the worst margin violator

  $$\text{argmin}_{i,\mathbf{y}}\mathbf{w}^T\Delta\phi(\mathbf{x}_i,\mathbf{y}) - \ell(\mathbf{y}_i,\mathbf{y})$$

  to the dual problem. One can guarantee an approximate solution with a polynomial number of support vectors by this approach (Altun et al., 2003; Tsochantaridis et al., 2004).

- Primal methods where the solution above inference problem is integrated to the primal optimization problem, rather than writing down the exponential-sized constraint set (Taskar et al., 2004).

- Marginal dual methods, where the problem is translated to a polynomial-sized form via considering the marginals of the dual variables (Taskar et al., 2003).

The methodology presented in this article belongs to the third category.

2.5 Marginalized Dual Problem

The feasible set of the dual problem (2) is a Cartesian product

$$\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_m \tag{3}$$

of identical closed polytopes

$$\mathcal{A}_i = \{\alpha_i \in \mathbb{R}^{|\mathcal{Y}|} \mid \alpha_i \geq 0, ||\alpha_i||_1 \leq C\}, \tag{4}$$

with a vertex set $\mathcal{V}_i = \{0, Ce_1, \ldots, Ce_{|\mathcal{Y}|}\} \subset \mathbb{R}^{|\mathcal{Y}|}$ consisting of the zero vector and the unit vectors of $\mathbb{R}^{|\mathcal{Y}|}$, scaled by C. The vertex set of $\mathcal{A}$ is the Cartesian product $\mathcal{V}_1 \times \cdots \times \mathcal{V}_m$.

The dimension of the set $\mathcal{A}$, $d_{\mathcal{A}} = m|\mathcal{Y}|$ is exponential in the length of the multilabel vectors. This means that optimizing directly over the the set $\mathcal{A}$ is not feasible. Fortunately by utilizing the structure of T, the set $\mathcal{A}$ can be mapped to a set $\mathcal{M}$ of polynomial dimension, called the marginal polytope of H, where optimization becomes more feasible (Taskar et al., 2003).

For an edge $e \in E$ of the Markov tree T, and an associated labeling $\mathbf{y}_e$, the marginal of $\alpha(i,\mathbf{y})$ for the pair $(e,\mathbf{y}_e)$ is given by

$$\mu_e(i,\mathbf{y}_e) = \sum_{\{\mathbf{u}\in\mathcal{Y}_i\}} [\mathbf{y}_e = \mathbf{u}_e]\alpha(i,\mathbf{u}) \tag{5}$$

where the sum picks up those dual variables $\alpha(i,\mathbf{y})$ that have equal value $\mathbf{u}_e = \mathbf{y}_e$ on the edge e. Single node marginals $\mu_j(i,y_j)$ are defined analogously.

For the hierarchy T, the vector containing the edge marginals of the example $\mathbf{x}_i$, the marginal dual vector, is given by

$$\mu_i = (\mu_e(i,\mathbf{u}_e))_{e\in E,\mathbf{u}_e\in\mathcal{Y}_e}.$$

The marginal vector of the whole training set is the concatenation of the single example marginal dual vectors $\mu = (\mu_i)_{i=1}^m$. The vector has dimension $d_{\mathcal{M}} = m\sum_{e\in E}|\mathcal{Y}_e| = O(m|E|\max_e|\mathcal{Y}_e|)$. Thus the dimension is linear in the number of the examples, edges and the maximum cardinality of set of labelings of a single edge.

The indicator functions in (5) can be collectively represented by the the matrix M_E, $M_E(e,\mathbf{u}_e;\mathbf{y}) = [\mathbf{u}_e = \mathbf{y}_e]$, and the relationship between a dual vector alpha and the corresponding marginal vector μ is given by the linear map $M_E\cdot\alpha_i = \mu_i$ and $\mu = (M_E\cdot\alpha_i)_{i=1}^m$. The image of the set $\mathcal{A}_i$, defined by

$$\mathcal{M}_i = \{\mu_i|\ \exists\alpha_i\in\mathcal{A}_i : M_E\alpha_i = \mu_i\}$$

is called the *marginal polytope* of α_i on T.

The following properties of the set $\mathcal{M}_i$ are immediate: Let $\mathcal{A}_i$ be the polytope of (4) and let $\mathcal{M}_i$ be the corresponding marginal polytope. Then

- the vertex set of $\mathcal{M}_i$ is the image of the vertex set of $\mathcal{A}_i$:

$$V_{\mu,i} = \{\mu_i|\ \exists\alpha_i\in V_i : M_E\alpha_i = \mu_i\}.$$

- As an image of a convex polytope $\mathcal{A}_i$ under the linear map M_E, $\mathcal{M}_i$ is a convex polytope.

These properties underlie the efficient solution of the dual problem on the marginal polytope.

The exponential size of the dual problem (2) can be tackled via the relationship between its feasible set $\mathcal{A} = \mathcal{A}_1\times\cdots\times\mathcal{A}_m$ and the marginal polytopes $\mathcal{M}_i$ of each $\mathcal{A}_i$.

Given a decomposable loss function

$$\ell(\mathbf{y}_i,\mathbf{y}) = \sum_{e\in E}\ell_e(i,\mathbf{y}_e)$$

the linear part of the objective satisfies

$$\begin{aligned}
\sum_{i=1}^m\sum_{\mathbf{y}\in\mathcal{Y}}\alpha(i,\mathbf{y})\ell(i,\mathbf{y}) &= \sum_{i=1}^m\sum_{\mathbf{y}}\alpha(i,\mathbf{y})\sum_e\ell_e(i,\mathbf{y}_e)\\
&= \sum_{i=1}^m\sum_{e\in E}\sum_{\mathbf{u}_e\in\mathcal{Y}_e}\sum_{\mathbf{y}:\mathbf{y}_e=\mathbf{u}_e}\alpha(i,\mathbf{y})\ell_e(i,\mathbf{u}_e) = \sum_{i=1}^m\sum_{e\in E}\sum_{\mathbf{u}_e\in\mathcal{Y}_e}\mu_e(i,\mathbf{u}_e)\ell_e(i,\mathbf{u}_e)\\
&= \sum_{i=1}^m\mu_i^T\ell_i = \mu^T\ell_E,
\end{aligned}$$

where $\ell_E = (\ell_i)_{i=1}^m = (\ell_e(i,\mathbf{u}_e))_{i=1,e\in E,\mathbf{u}_e\in\mathcal{Y}_e}^m$ is the marginal loss vector.

Given an orthogonal feature representation inducing a decomposable kernel $K(\mathbf{x},\mathbf{y};\mathbf{x}',\mathbf{y}') = \sum_{e\in E} K_e(\mathbf{x},\mathbf{y}_e;\mathbf{x}',\mathbf{y}'_e)$, the quadratic part of the objective becomes

$$\begin{aligned}
\alpha\mathbf{K}\alpha &= \sum_e \sum_{i,i'} \sum_{\mathbf{y},\mathbf{y}'} \alpha(i,\mathbf{y}) K_e(i,\mathbf{y}_e;i',\mathbf{y}'_e)\alpha(i',\mathbf{y}') \\
&= \sum_e \sum_{i,i'} \sum_{\mathbf{u}_e,\mathbf{u}'_e} K_e(i,\mathbf{u}_e;i',\mathbf{u}'_e) \sum_{\mathbf{y}:\mathbf{y}_e=\mathbf{u}_e} \sum_{\mathbf{y}':\mathbf{y}'_e=\mathbf{u}'_e} \alpha(i,\mathbf{y})\alpha(i',\mathbf{y}') \\
&= \sum_e \sum_{i,i'} \sum_{\mathbf{u}_e,\mathbf{u}'_e} \mu_e(i,\mathbf{u}_e) K_e(i,\mathbf{u}_e;i',\mathbf{u}'_e)\mu_e(i,\mathbf{u}'_e) \\
&= \mu^T \mathbf{K}_E \mu,
\end{aligned}$$

where $\mathbf{K}_E = \mathrm{diag}\,(\mathbf{K}_e, e\in E)$ is a block diagonal matrix with edge-specific kernel blocks $\mathbf{K}_e$.

The objective should be maximized with respect to μ whilst ensuring that there exist $\alpha\in\mathcal{A}$ satisfying $M\alpha_i = \mu_i$ for all i, so that the marginal dual solution represents a feasible solution of the original dual. By the properties outlined above, the feasible set of the marginalized problem is the marginal dual polytope, or to be exact the Cartesian product of the marginal polytopes of single examples (which are in fact equal):

$$\mathcal{M} = \mathcal{M}_1 \times \cdots \times \mathcal{M}_m$$

In summary, the marginalized optimization problem can be stated in implicit form as

$$\max_{\mu\in\mathcal{M}} \mu^T \ell_E - \frac{1}{2}\mu^T \mathbf{K}_E \mu$$

This problem is a quadratic programme with a linear number of variables in the number of training examples and in the number of edges.

For optimization algorithms, an explicit characterization of the feasible set is required. Characterizing the polytope $\mathcal{M}$ in terms of linear constraints defining the faces of the polytope, is for general graphs infeasible. Singly-connected graphs such as trees are an exception: for such graphs the marginal polytope is exactly reproduced by the box constraints

$$\sum_{\mathbf{u}_e} \mu_e(i,\mathbf{u}_e) \le C, \forall i, e\in E, \mu_e \ge 0 \tag{6}$$

and the local consistency constraints

$$\sum_{y_k} \mu_{kj}(i,y_k,y_j) = \mu_j(i,y_j); \sum_{y_j} \mu_{kj}(i,y_k,y_j) = \mu_k(i,y_k). \tag{7}$$

In this case the size of the resulting constraint set is linear in the number of vertices the graph. Thus for small hierarchies graphs it can be written down explicitly and the resulting optimization problem has linear size in both the number of examples and the size of the graph. Thus the approach can in principle be made to work, although not with off-the-shelf QP solvers (see sections 3 and 5).

For hierarchies, the consistency constraints (7), can be equivalently defined in terms of the edges: it suffices to pair up each edge with its parent which results in the set of edge pairs $E_2 =$

$\{(e,e') \in E \times E | e = (j',i), e' = (i,j)\}$. By introduction of these marginal consistency constraints the optimization problem gets the form

$$\begin{aligned} \max_{\mu \geq 0} & \sum_{e \in E} \mu_e^T \ell_e - \frac{1}{2} \sum_{e \in E} \mu_e^T \mathbf{K}_e \mu_e \\ \text{s.t} & \sum_{\mathbf{y}_e} \mu_e(i, \mathbf{y}_e) \leq C, \forall i, e \in E, \\ & \sum_{y'} \mu_e(i,(y',y)) = \sum_{y'} \mu_{e'}(i,(y,y')), \ \forall i, y, (e,e') \in E_2, \\ , & \end{aligned} \tag{8}$$

While the above formulation is closely related to that described in Taskar et al. (2003), there are a few differences to be pointed out. Firstly, as we assign the loss to the edges rather than the microlabels, we are able to use richer loss functions than the simple ℓ_Δ. Secondly, single-node marginal dual variables—the μ_j's in (7)—become redundant when the constraints are given in terms of the edges. Thirdly, we have utilized the fact that in our feature representation the 'cross-edge' values $\Delta\phi_e(x, \mathbf{y}_e)^T \Delta\phi_{e'}(x', \mathbf{y}'_{e'})$, where $e \neq e'$, do not contribute to the kernel, hence we have a block-diagonal kernel $\mathbf{K}_E = \text{diag}\,(\mathbf{K}_{e_1}, \ldots, \mathbf{K}_{e_{|E|}}), K_E(i,e,\mathbf{u}_e; j,e,\mathbf{v}_e) = K_e(i,\mathbf{u}_e; j,\mathbf{v}_e)$ with the number of non-zero entries thus scaling linearly rather than quadratically in the number of edges. Finally, we write the box constraint (6) as an inequality as we want the algorithm to be able to inactivate training examples (see Section 3.2).

Like that of Taskar et al. (2003), our approach can be generalized to non-tree structures. However, for a general graph, the feasible region in (8) will only approximate that of (2), which will give rise to a approximate solution to the primal. To arrive at an exact solution, one should construct the junction tree for the graph and to write down the corresponding constraints for the junction tree. As a caveat, one should note that for dense graphs, the junction tree may be significantly larger than the size of the original structure. Also, in tractable time, finding the maximum likelihood multilabel can only be approximated.

3. Efficient Optimization of the Marginalized Dual Problem

While the above quadratic program is polynomial-sized—and considerably smaller than that described in Taskar et al. (2003)—it is still easily too large in practice to fit in main memory or to solve by off-the-shelf QP solvers. To arrive at a more tractable problem, we notice from (3) and (4) that the constraint set decomposes by the examples: to satisfy a single box constraint (6) or a marginal consistency constraint (7) one only needs to change the marginal dual variables of a single example. Moreover, the structure of the feasible set only depends on the edge set E, not on the training example in question: we have $\mathcal{A}_1 = \cdots = \mathcal{A}_m$.

However, the kernel matrix only decomposes by the edges as most pairs of examples have non-positive kernel value between them. Thus there does not seem to be a straightforward way to decompose the quadratic programme.

A decomposition becomes possible when considering gradient-based approaches. Let us consider optimizing the dual variables $\mu_i = (\mu_e(i, \mathbf{y}_e))_{e \in E, \mathbf{y}_e \in \mathcal{Y}_e}$ of example x_i where ℓ_i denotes the corresponding loss vector and $\mathbf{K}_{ij} = \left(K_e(i,\mathbf{u}_e; j, \mathbf{v}_e)_{e \in E, \mathbf{u}_e, \mathbf{v}_e \in \mathcal{Y}_e}\right)$ denotes the block of kernel values between examples i and j, and by $\mathbf{K}_{i\cdot} = (\mathbf{K}_{ij})_{j \in \{1,\ldots,m\}}$ the columns of the kernel matrix $\mathbf{K}_E$ referring to example i.

Obtaining the gradient for the x_i-subspace requires computing the corresponding part of the gradient of the objective function in (8) which is $\mathbf{g}_i = \ell_i - \mathbf{K}_{i\cdot}\mu$ where $\ell_i = (\ell_e(i, \mathbf{u}_e))_{e\in E, \mathbf{u}_e\in\mathcal{Y}_e}$ is the corresponding loss vector for x_i. However, when updating μ_i only, evaluating the change in objective and updating the gradient can be done more cheaply: $\Delta\mathbf{g}_i = -\mathbf{K}_{ii}\Delta\mu_i$ and $\Delta obj = \mathbf{g}_i^T\Delta\mu_i - 1/2\Delta\mu_i\mathbf{K}_{ii}\Delta\mu_i$. Thus local optimization in a subspace of a single training example can be done without consulting the other training examples. On the other hand, we do not want to spend too much time in optimizing a single example: When the dual variables of the other examples are non-optimal, so is the initial gradient $\mathbf{g}_i$. Thus the optimum we would arrive at would not be the global optimum of the quadratic objective. It makes more sense to optimize all examples more or less in tandem so that the full gradient approaches its optimum as quickly as possible.

Before presenting the pseudocode of our method some notations have to be introduced. The function $f()$ denotes the objective function and $\mathcal{F}$ stands for the set of the feasible solutions in (8). The feasibility domain for μ_i when all other components in μ are fixed is denoted by $\mathcal{F}_i$.

In our approach, we have chosen to conduct a few optimization steps for each training example using a conditional gradient ascent (see Algorithm 2) before moving on to the next example. The iteration limit for each example is set by using the Karush-Kuhn-Tucker(KKT) conditions as a guideline (see Section 3.2).

The pseudocode of our algorithm is given in Algorithm 1. It takes as input the training data, the edge set of the hierarchy, the loss vector $\ell = (\ell_i)_{i=1}^m$ and the constraints defining the feasible region. The algorithm chooses a chunk of examples as the working set, computes the kernel for each x_i and makes an optimization pass over the chunk. After one pass, the gradient, slacks and the duality gap are computed and a new chunk is picked. The process is iterated until the duality gap gets below given threshold.

Note in particular, that the joint kernel is not explicitly computed, although evaluating the gradient requires computing the product $\mathbf{K}_E\mu$. However, we are able to take advantage of the special structure of the feature vectors, repeating the same feature vector in different contexts, see the definition of the edge marginal dual variables (5) and the explanation after, to facilitate the computation using the x-kernel $K^x(i,j) = \Delta\phi(x_i)^T\Delta\phi(x_j)$ and the dual variables only.

3.1 Conditional Subspace Gradient Ascent

The optimization algorithm used for a single example is a variant of conditional gradient ascent (or descent) algorithms (Bertsekas, 1999). The algorithms in this family solve a constrained quadratic problem by iteratively stepping to the best feasible direction with respect to the current gradient. It exploits the fact if μ^* is an optimum solution of a maximization problem with objective function f above the feasibility domain $\mathcal{F}_i$ then it has to satisfy the first order optimality condition, i.e., the inequality

$$\nabla f(\mu_i)(\mu_i - \mu^*) \geq 0 \tag{9}$$

has to hold for any feasible μ_i chosen from $\mathcal{F}_i$.

The pseudocode of our variant CSGA is given in Algorithm 2. The algorithm takes as input the current dual variables, gradient, constraints and the kernel block for the example x_i, and an iteration limit. It outputs new values for the dual variables μ_i and the change in objective value. As discussed above, the iteration limit is set very tight so that only a few iterations will be typically conducted.

Algorithm 1 Maximum margin optimization algorithm for the H-M^3 hierarchical classification model.

H-M^3($S, E, \ell, \mathcal{F}$)

Require: Training data $S = ((x_i, \mathbf{y}_i))_{i=1}^m$, edge set E of the hierarchy, a loss vector ℓ, and the feasibility domain $\mathcal{F}$.

Ensure: Dual variable vector μ and objective value $f(\mu)$.

```
Initialize g = ℓ, ξ = ℓ, dg = ∞ and OBJ = 0.
while dg > dg_min & iter < max_iter do
    [WS, Freq] = UpdateWorkingSet(μ, g, ξ);
    Compute x-kernel values K_{X,WS} with respect to the working set;
    for i ∈ WS do
        Compute joint kernel block K_ii and subspace gradient g_i;
        [μ_i, Δobj] = CSGA(μ_i, g_i, K_ii, F_i, Freq_i);
    end for
    Compute gradient g, slacks ξ and duality gap dg;
end while
```

First we need to find a feasible μ^* which maximizes the first order feasibility condition (9) at a fixed μ_i. It gives a direction potentially increasing the value of objective function f. Then we have to choose a step length, τ that gives the optimal feasible solution as a stationary point along the line segment $\mu_i(\tau) = \mu_i + \tau\Delta\mu,\ \tau \in (0,1]$, where $\Delta\mu = \mu^* - \mu_i$, starting on the known feasible solution μ_i.

The stationary point is found by solving the equation

$$\frac{\mathbf{d}}{\mathbf{d}\tau}\left[\ell_i^T \mu_i(\tau) - 1/2\mu_i(\tau)^T \mathbf{K}_{ii}\mu_i(\tau)\right] = 0, \tag{10}$$

expressing the optimality condition with respect to τ. If $\tau > 1$, the stationary point is infeasible and the feasible maximum is obtained at $\tau = 1$. In our experience, the time taken to compute the stationary point was typically significantly smaller than time taken to find μ_i^*, depending on the dataset characteristics and the actual algorithm (see Section 3.3) that was used to find μ_i^*.

3.2 Working Set Maintenance

We wish to maintain the working set so that the most promising examples to be updated are contained there at all times to minimize the amount of computation used for unsuccessful updates. Our working set update is based on the Karush-Kuhn-Tucker(KKT) conditions which at the optimum hold for all x_i:

1. $(C - \sum_{e,\mathbf{y}_e} \mu_e(i,\mathbf{y}_e))\xi_i = 0$, and
2. $\alpha(i,\mathbf{y})(\mathbf{w}^T\phi(x_i,\mathbf{y}) - \ell(\mathbf{y}_i,\mathbf{y}) + \xi_i) = 0$.

The first condition states that, at optimum, only examples that saturate the box constraint can have positive slack, and consequently a pseudo-example that has a negative margin. The second condition states that pseudo-examples with non-zero dual variables are those that have the minimum margin, that is, need the full slack ξ_i. Consequently, if all pseudo-examples of x_i have positive margin, all dual variables satisfy $\alpha(i,\mathbf{y}) = 0$. This observation leads to the following heuristics for the working set update:

Algorithm 2 Conditional subspace gradient ascent optimization step.

CSGA($\mu_i, \mathbf{g}_i, \mathbf{K}_{ii}, \mathcal{F}_i, maxiter_i$)

Require: Initial dual variable vector μ_i, gradient $\mathbf{g}_i$, constraints of the feasible region $\mathcal{F}_i$, a joint kernel block $\mathbf{K}_{ii}$ for the subspace, and an iteration limit $maxiter_i$.

Ensure: New values for dual variables μ_i and change in objective Δobj.

$\Delta obj = 0; iter = 0;$
while $iter < maxiter$ **do**
  % find highest feasible point given $\mathbf{g}_i$
  $\mu^* = \text{argmax}_{v \in \mathcal{F}_i} \mathbf{g}_i^T v;$
  $\Delta\mu = \mu^* - \mu_i;$
  $q = \mathbf{g}_i^T \Delta\mu,\ r = \Delta\mu^T \mathbf{K}_{ii} \Delta\mu;$ % taken from the solution of (10)
  $\tau = \min(q/r, 1);$ % clip to remain feasible
  if $\tau \leq 0$ **then**
    break; % no progress, stop
  else
    $\mu_i = \mu_i + \tau\Delta\mu;$ % update
    $\mathbf{g}_i = \mathbf{g}_i - \tau\mathbf{K}_{ii}\Delta\mu;$
    $\Delta obj = \Delta obj + \tau q - \tau^2 r/2;$
  end if
  $iter = iter + 1;$
end while

- Non-saturated ($\sum_{e,\mathbf{y}_e} \mu_e(i,\mathbf{y}_e) < C$) examples are given priority as they certainly will need to be updated to reach the optimum.

- Saturated examples ($\sum_{e,\mathbf{y}_e} \mu_e(i,\mathbf{y}_e) = C$) are added if there are not enough non-saturated ones. The rationale is that the even though an example is saturated, the individual dual variable values may still be suboptimal being equal to 0.

- Inactive ($\sum_{e,\mathbf{y}_e} \mu_e(i,\mathbf{y}_e) = 0$) non-violators ($\xi_i = 0$) are removed from the working set, as they do not constrain the objective.

Another heuristic technique to concentrate computational effort to most promising examples is to favor examples with a large duality gap

$$\Delta obj(\mu, \xi) = \sum_i C\xi_i + \mu_i^T \mathbf{g}_i.$$

As feasible primal solutions always are least as large as feasible dual solutions, the duality gap gives an upper bound to the distance from the dual solution to the optimum. We use the quantity $\Delta_i = C\xi_i + \mu_i^T \mathbf{g}_i$ as a heuristic measure of the work needed for that particular example in order to reach the optimum. Examples are then chosen to the chunk to be updated with probability proportional to $p_i \propto \Delta_i - \min_j \Delta_j$. An example that is drawn more than once will be set a higher iteration limit for the next optimization step.

3.3 Finding Update Directions Efficiently

The optimization algorithm described above relies on efficient computation of update directions μ_i^* in the single example subspaces, that is, to solve the constrained linear program

$$\text{argmax}_{\mathbf{v}\in\mathcal{F}_i}\ \mathbf{g}_i^T\mathbf{v}. \tag{11}$$

A straightforward approach would be to use a linear programming solver, such as the LIPSOL interior point solver. However, a such black-box approach does not utilize the special structure of the problem in any way.

In order to solve this problem efficiently, we first notice two things:

1. A vertex of the feasible set is always among the optimal solutions.

2. Vertices correspond to consistent labelings of the hierarchy. This can be seen from the fact that at the vertex, for each edge $\mu_e(i,\mathbf{y}_e) = C$ for exactly one $\mathbf{y}_e$ and $\mu_e(i,\mathbf{u}_e) = 0$ for $\mathbf{u}_e \neq \mathbf{y}_e$, and that the marginal consistency constraints require that for two adjacent edges $e' = (j', j), e'' = (j, j'')$ we have $\mu_{e'}(i,\mathbf{y}'_e) = C = \mu_{e''}(i,\mathbf{y}''_e)$ with matching edge-labelings $\mathbf{y}'_e = (y_{j'}, y_j)$ and $\mathbf{y}''_e = (y_j, y_{j''})$.

Thus instead of solving (11) directly, we can search for the labeling $\mathbf{y}_*$ of the hierarchy corresponding to an optimal vertex
$vmu(\mathbf{y}_*)$ of the feasible set:

$$\text{argmax}_{\mathbf{y}\in\mathcal{Y}}\ \mathbf{g}_i^T\mu(\mathbf{y}) \tag{12}$$

This problem can be solved efficiently using a dynamic programming inference algorithm, reviewed in the next section.

3.4 Solving the Inference Problem in Linear Time

When dealing with structured output models, one needs to solve the inference problem

$$\text{argmax}_{\mathbf{y}\in\mathcal{Y}}\ \mathbf{g}_i^T\mu(\mathbf{y}) \tag{13}$$

to find a multilabel $\mathbf{y}$ maximizing the inner product between some (gradient) vector h and the marginal dual variables $\mu(\mathbf{y})$ corresponding to $\mathbf{y}$. In our learning scheme this problem is found in two situations,

- when predicting multilabels given a learned model, and

- to find update directions (12).

The algorithm described below can be used for both problems, the only quantity that changes is the gradient $\mathbf{g}_i$.

Inference algorithms solving problems of the above form have been well-studied in the literature of probabilistic models, under the names of belief propagation and generalized distributive law (Aji and McEliece, 2000; Kschischang et al., 2001; Wainwright and Jordan, 2003). It is known that, for general graphs, solving (13) is not any easier than solving (11). However, for a hierarchical model dynamic programming can be used: starting from the leaves of the hierarchy we compute bottom-up

for each subtree the optimal labeling of the subtree, conditioned on fixing the label of the subtree root to $+1$ or -1.

We denote by $T_j = (V_j, E_j)$ the subtree of T rooted at node j. We need to maintain two quantities during the bottom-up pass:

- The best objective value that can be obtained for the example i in the subtree rooted at node j when the label y_j has been fixed. We denote this value by $S_{y_j}(i, j)$.
- The best objective value that can be obtained for the subtree rooted by the edge $e = (j, j')$ when the root node j is fixed to y_j. We denote this value by $G_{y_j}(i, e)$.

The two quantities are computed from the recurrences

$$S_{y_j}(i,j) = \begin{cases} \sum_{e=(j,j')\in E_j} G_{y_j}(i,e), & \text{if } E_j \neq \emptyset, \text{and} \\ 0, & \text{otherwise ,} \end{cases}$$

and

$$G_{y_j}(i,e) = \max_{y_{j'}} g_e(i,y_j,y_{j'})\mu_e(i,y_j,y_{j'}) + S_{y_{j'}}(i,j')$$

At the root node of the hierarchy, $\max_y S_y(i, root)$ finally gives the optimum. The corresponding vertex $v(\mathbf{y}_*)$ is found in making a top-down pass over the hierarchy: one looks for best label for a child of a node given the parent has been fixed. It should be noted that although in principle the best conditional labeling—how to label a subtree when the root is fixed to one of the possible labels—could be computed already during the bottom-up pass, the two pass algorithm, where the labeling is worked out only after the label of the global root of the hierarchy has been found out, is much easier to implement and works just as fast.

The dynamic programming scheme can be implemented in vectorized form so that all examples and all nodes on a level of the hierarchy are handled at the same time, thus eliminating the need for loops going over examples and nodes, which in MATLAB implementation are to be avoided.

All in all, the above described inference algorithm works in linear time in the number of dual variables, which can be seen from the fact that each example is processed once, each edge is visited twice (once in the bottom-up pass, once in the top-down pass) and the max operations are taken over the dual variables belonging to the current edge.

3.5 Computing Stationary Points in Linear Time

The conditional gradient ascent requires us to iteratively solve (10) for τ, which gives $\tau = \Delta\mu / \Delta\mu \mathbf{K}_{ii} \Delta\mu$. The potentially expensive part is evaluating the matrix-vector product $\mathbf{K}_{ii}\Delta\mu = \mathbf{K}_{ii}\mu^* - \mathbf{K}_{ii}\mu_i$, which trivially could take quadratic time in the number of variables. However, we can keep in memory the vector $\mathbf{K}_{ii}\mu_i$ during the computation, thus it remains to compute $\mathbf{K}_{ii}\mu^*$. Firstly, we notice that for a normalized x-kernel, the entries of the joint kernel are given as sums of indicators $K_{ii}(e, \mathbf{u}_e; e, \mathbf{u}'_e) = 1 - [\mathbf{y}_{ie} = \mathbf{u}'_e] - [\mathbf{y}'_{ie} = \mathbf{u}_e] + [\mathbf{u}_e = \mathbf{u}'_e]$. Secondly, since μ^* is an extreme point of the feasible set, $\mu^*(e, \mathbf{u}_e) = C$ for exactly one of the components $\mathbf{u}_e \in \mathcal{Y}_e$. By these facts and some arithmetic manipulation we obtain $\mathbf{K}_{ii}\mu^* = [\mathbf{1} - \mathbf{y}_i]C - \mathbf{y}_i \cdot \mu^* + \mu^*$. Thus, instead of matrix-vector product we only need to compute a single vector-vector product and a sum of three vectors. Finally, the update for $\mathbf{K}_{ii}\mu_i$ is given as a convex combination of vectors $\mathbf{K}_{ii}\mu_i^{new} = \tau\mathbf{K}_{ii}\mu^* + (1-\tau)\mathbf{K}_{ii}\mu_i$. The total number of operations to compute the stationary point remains linear in the number of variables.

4. Extensions and Variants

There are several variations of the multilabel classification models described above.

Slack variables were defined as non-negative and a single variable was allocated per example. Allowing negative slack (c.f.Taskar et al. (2003); Tsochantaridis et al. (2004)) results in the dual equality constraint $\sum_{\mathbf{y}_e} \mu_e(i,\mathbf{y}_e) = C$ instead of the box constraint. This results in non-sparse models as training points are very likely to have non-zero slack.

Allotting a separate slack variable for each edge is a possibility when the data for some edges can be considered less reliable than the data for others; in such case the unreliable edge can consume required slack without affecting the other edges. From an optimization point of view, edge-based slack variables make the model decompose into separate edge-based quadratic programs and may allow larger models to be optimized.

Partial paths could be used as the basis of the classification model instead of the edges. For each partial path $p = (j_1,\ldots,j_d)$ one defines a feature vector $\phi_p(i,\mathbf{y}) = [\mathbf{y}_p = \mathbf{1}_{|p|}]\phi(x)$, where $\mathbf{y}_p = (y_{j_1},\ldots,y_{j_d})$ is the restriction of the multilabel to the partial path. As the number of partial paths in the hierarchy equals the number of nodes, the resulting feature vectors are actually smaller than the ones defined by edge-labelings. The marginalization of the model by the partial paths works in an analogous way to the edge-marginalization and the same optimization algorithms can be used. The price of the more compact feature representation comes in the form of slightly more complicated consistency constraints and inference: For consistency one needs to ensure that if a partial path p has non-zero path-marginal $\mu_p(i,\mathbf{y}_p)$, no prefix p' of p has non-zero marginal $\mu_{p'}(i,\mathbf{y}_p)$. Correspondingly, the inference algorithms need to make comparisons between a partial path and its prefixes.

Non-hierarchical models can also be tackled with the above described framework, with a few caveats. First, ensuring global consistency of the marginalized dual is more involved as local consistency of edge-marginals does not guarantee existence of a dual variable $\alpha(i,\mathbf{y})$ with those marginals. If the graph is not too dense this problem can be circumvented by computing the clique tree of the graph and making the clique tree locally consistent, and the conditional gradient optimization will work unmodified. However, inference for general graphs is NP-hard so both computing predictions of the model and finding the update directions in the optimization becomes hard. Several schemes to find approximate solutions exist, including loopy belief propagation, semi-definite relaxations and tree-based approximations (Wainwright and Jordan (2003); Wainwright et al. (2003)). Depending on the application, also considering the model in a decomposed form via definition of edge-slack variables (see above) may be justified.

5. Experiments

We tested the presented learning approach on three datasets that have an associated classification hierarchy:

- REUTERS Corpus Volume 1, RCV1 (Lewis et al., 2004). 2500 documents were used for training and 5000 for testing. As the label hierarchy we used the 'CCAT' family of categories (Corporate/Industrial news articles), which had a total of 34 nodes, organized in a tree with maximum depth 3. The tree is quite unbalanced, half of the nodes residing in depth 1, and very few nodes in depth 3.

- WIPO-alpha patent dataset (WIPO, 2001). The dataset consisted of the 1372 training and 358 testing document comprising the D section of the hierarchy. The number of nodes in the hierarchy was 188, with maximum depth 3.

- ENZYME classification dataset. The training data consisted of 7700 protein sequences with hierarchical classification given by the Enzyme Classification (EC) system. The hierarchy consisted of 236 nodes organized into a tree of depth three. Test data consisted of 1755 sequences.

In all datasets, the membership of examples in the nodes of the hierarchy is indicated by binary vectors $\mathbf{y} \in \{+1,-1\}^k$. Multiple paths were actually present in one of the datasets, REUTERS, approximately 8 percent of examples were classified into more than one category.

The two first datasets were processed into bag-of-words representation with TFIDF weighting. No word stemming or stop-word removal was performed. For the ENZYME sequences a length-4 subsequence kernel was used.

We compared the performance of the presented learning approach—below denoted by H-M^3—to three algorithms: SVM denotes an SVM trained for each microlabel separately, H-SVM denotes the case where the SVM for a microlabel is trained only with examples for which the ancestor labels are positive.

The SVM and H-SVM were run using the SVM-light package. After pre-computation of the kernel these algorithms are as fast as one could expect, as they just involve solving an SVM for each node in the graph (with the full training set for SVM and usually a much smaller subset for H-SVM).

H-RLS is a batch version of the hierarchical least squares algorithm described in Cesa-Bianchi et al. (2004). It essentially solves for each node i a least squares style problem $\mathbf{w}_i = (I + S_i S_i^T + \mathbf{x}\mathbf{x}^T)^{-1} S_i \mathbf{y}_i$, where S_i is a matrix consisting of all training examples for which the parent of node i was classified as positive, $\mathbf{y}_i$ is a microlabel vector for node i of those examples and I is the identity matrix. Predictions for a node i for a new example x is -1 if the parent of the node was classified negatively and $\text{sign}(\mathbf{w}_i^T \mathbf{x})$ otherwise.

H-RLS requires a matrix inversion for each prediction of each example, at each node along a path for which errors have not already been made. No optimization of the algorithm was done, except to use extension approaches to efficiently compute the matrix inverse (for each example an inverted matrix needs to be extended by one row/column, so a straightforward application of the Sherman-Morrison formula to efficiently update the inverse can be used).

The H-RLS and H-M^3 algorithms were implemented in MATLAB. The tests were run on a high-end PC. For SVM,H-SVM and H-M^3, the regularization parameter value $C = 1$ was used in all experiments.

Obtaining consistent labelings. As the learning algorithms compared here all decompose the hierarchy for learning, the multilabel composed of naively combining the microlabel predictions may be inconsistent, that is, they may predict a document as part of the child but not as part of the parent. For SVM and H-SVM consistent labelings were produced by post-processing the predicted labelings as follows: start at the root and traverse the tree in a breadth-first fashion. If the label of a node is predicted as -1 then all descendants of that node are also labeled negatively. This post-processing turned out to be crucial to obtain good accuracy, thus we only report results with the postprocessed labelings. Note that H-RLS performs essentially the same procedure (see above). For H-M^3 models, we computed by dynamic programming the consistent multilabel with maximum

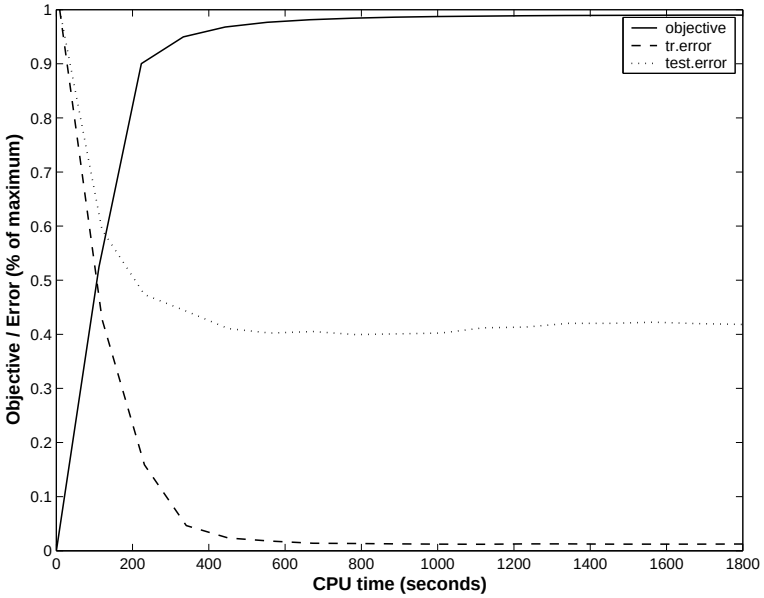

Figure 2: The objective function (% of optimum) and ℓ_Δ losses for H-M^3 on training and test sets (WIPO-alpha)

likelihood

$$\mathbf{u}(\mathbf{x}) = \text{argmax}_{\mathbf{y}\in\mathcal{Y}_T} P(\mathbf{y}|x) = \text{argmax}_{\mathbf{y}} \mathbf{w}^T \phi(\mathbf{x},\mathbf{y}),$$

where $\mathcal{Y}_T$ is the set of multilabels that correspond to unions of partial paths in T. The algorithm is otherwise the same as the one in 3.4, but the inconsistent edge-labelings are not taken into account in the maximization.

Efficiency of optimization. To give an indication of the efficiency of the H-M^3 algorithm, Figure 2 shows an example learning curve on WIPO-alpha dataset. The number of dual variables for this training set is just over one million with a joint kernel matrix with approx 5 billion entries. Note that the solutions for this optimization are not sparse, typically less than 25% of the marginal dual variables are zero. Training and test losses (ℓ_Δ) are all close to their optima within 10 minutes of starting the training, and the objective is within 2 percent of the optimum in 30 minutes.

To put these results in perspective, for the WIPO data set SVM (SVM-light) takes approximately 50 seconds per node, resulting in a total running time of about 2.5 hours, which makes it significantly slower than H-M^3, in these tests. It is possible that using early stopping for SVM the training time could be pushed down to the level of H-M^3, however, we have not explored this question. We also suspect that early stopping for SVM may be more costly than for H-M^3, due to the fact that the latter predicts whole labelings for the trees where the weight of a single microlabel is small, and in fact the predicted multilabels may contain microlabels that are not locally optimal. In other words, the inference procedure for multilabels may correct poor microlabel predictions.

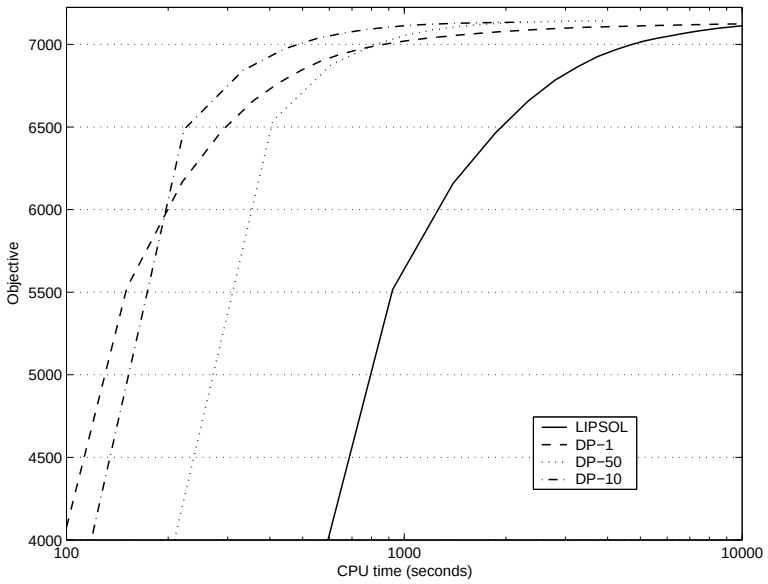

Figure 3: Learning curves for H-M^3 using LIPSOL and dynamic programming (DP) to compute update directions (WIPO-alpha). Curves with iteration limits 1,10 and 50 are shown for DP. The LIPSOL curve is computed with iteration limit set to 1.

The running time of H-RLS was slower than the other methods, however this could be due to our unoptimized implementation. It is our expectation that it would be very close to the time taken by H-SVM if coded more efficiently.

Therefore, the methods presented in this paper are very competitive from a computational efficiency point of view to other methods which do not operate in the large feature/output spaces of H-M^3.

Figure 3 shows on WIPO-alpha the efficiency of the dynamic programming (DP) based computation of update directions as compared to solving the update directions with MATLAB's linear interior point solver LIPSOL. The DP based updates result in an order of magnitude faster optimization than using LIPSOL.

In addition for DP the effect of the iteration limit for optimization speed is depicted. Setting the iteration limit too low (1) or too high (50) slows down the optimization, for different reasons. A too tight iteration limit makes the overhead in moving from one example to the other dominate the running time. A too high iteration limit makes the the algorithm spend too much time optimizing the dual variables of a single example. Unfortunately, it is not straightforward to suggest a iteration limit that would be universally the best, as the optimal value depends on the dataset.

Effect of choice of the loss function. In order to show the effect of training the H-M^3 algorithm using the different loss functions described in Section 2.1, we studied the performance of the algorithm on Reuters and WIPO data sets. The results can be seen in Table 5. The WIPO dataset gives an indication that using a hierarchical loss function during training (e.g. either $\ell_{\tilde{H}}$-sibl. or

	Test loss				
	$\ell_{0/1}$	ℓ_Δ	$\ell_{\tilde{H}}$ +scaling		
Tr. loss	%		unif	sibl.	subtree
ℓ_Δ	27.1	0.574	0.344	0.114	0.118
$\ell_{\tilde{H}}$-unif	26.8	0.590	0.338	0.118	0.122
$\ell_{\tilde{H}}$-sibl.	28.2	0.608	0.381	0.109	0.114
$\ell_{\tilde{H}}$-subtree	27.9	0.588	0.373	0.109	0.109
	$\ell_{0/1}$	ℓ_Δ	$\ell_{\tilde{H}}$ +scaling		
Tr. loss	%		unif	sibl.	subtree
ℓ_Δ	70.9	1.670	0.891	0.050	0.070
$\ell_{\tilde{H}}$-unif.	70.1	1.721	0.888	0.052	0.074
$\ell_{\tilde{H}}$-sibl.	64.8	1.729	0.927	0.048	0.071
$\ell_{\tilde{H}}$-subtree	65.0	1.709	0.919	0.048	0.072

Table 1: Prediction losses obtained using different training losses on Reuter's (top) and WIPO-alpha data (bottom). The loss $\ell_{0/1}$ is given as a percentage, the other losses as averages per-example.

$\ell_{\tilde{H}}$-subtree) may lead to a reduced $0/1$ loss on the test set. On Reuters dataset this effect is not observed, however this is due to the fact that the label tree of the Reuters data set is very shallow.

Comparison of predictive accuracies of different algorithms. In our final test we compare the predictive accuracy of H-M^3 to other learning methods. For H-M^3 we include the results for training with ℓ_Δ and $\ell_{\tilde{H}}$-*subtree* losses. For training SVM and H-SVM, these losses produce the same learned model.

Table 2 depicts the different test losses, as well as the standard information retrieval statistics precision (P), recall (R) and F1 statistic ($F1 = 2PR/(P+R)$). Precision and recall were computed over all microlabel predictions in the test set. Flat SVM is expectedly inferior to the competing algorithms with respect to most statistics, as it cannot utilize the dependencies between the microlabels in any way. The two variants of H-M^3 are the most efficient in getting the complete tree correct as shown by the lower zero-one loss. With respect to other statistics, the hierarchical methods are quite evenly matched overall.

Finally, to highlight the differences between the predicted labelings, we computed level-wise precision and recall values, that is, the set of predictions contained all test instances and microlabels on a given level of the tree (Table 3). On all datasets, recall of all methods, especially with SVM and H-SVM, diminishes when going farther from the root. H-M^3 is the most efficient method in fighting the recall decline, and is still able to obtain reasonable precision on REUTERS and WIPO-alpha, especially when trained with the hierarchical loss.

The results on ENZYME data are generally not good for any of the methods, this is most probably due to the subsequence kernel used not being able to pick out the subsequences corresponding to the active centers of the enzymes. Nevertheless, the effect of H-M^3 obtaining better recall in deep nodes than the competition can be observed.

REUTERS	$\ell_{0/1}$	ℓ_Δ	P	R	F1
SVM	32.9	0.61	94.6	58.4	72.2
H-SVM	29.8	0.57	92.3	63.4	75.1
H-RLS	28.1	**0.55**	91.5	65.4	**76.3**
H-M^3-ℓ_Δ	**27.1**	0.58	91.0	64.1	75.2
H-M^3-$\ell_{\tilde{H}}$	27.9	0.59	85.4	68.3	75.9
WIPO-alpha	$\ell_{0/1}$	ℓ_Δ	P	R	F1
SVM	87.2	1.84	93.1	58.2	71.6
H-SVM	76.2	1.74	90.3	63.3	74.4
H-RLS	72.1	1.69	88.5	66.4	75.9
H-M^3-ℓ_Δ	70.9	**1.67**	90.3	65.3	75.8
H-M^3-$\ell_{\tilde{H}}$	**65.0**	1.73	84.1	70.6	**76.7**
ENZYME	$\ell_{0/1}$	ℓ_Δ	P	R	F1
SVM	99.7	1.3	99.6	41.1	58.2
H-SVM	98.5	**1.2**	98.9	41.7	58.7
H-RLS	95.6	2.0	51.9	54.7	53.3
H-M^3-ℓ_Δ	95.7	**1.2**	87.0	49.8	**63.3**
H-M^3-$\ell_{\tilde{H}}$	**85.5**	2.5	44.5	66.7	53.4

Table 2: Prediction losses $\ell_{0/1}$ and ℓ_Δ, precision, recall and F1 values obtained using different learning algorithms. All figures are given as percentages. Precision and recall are computed in terms of totals of microlabel predictions in the test set.

6. Conclusions and Future Work

In this paper we have proposed a new method for training variants of the Maximum Margin Markov Network framework for hierarchical multi-category text classification models.

Our method relies on a decomposition of the problem into single-example sub problems and conditional gradient ascent for optimisation of the subproblems. The method scales well to medium-sized datasets with label matrix (examples $\times$ microlabels) size upto hundreds of thousands, and via kernelization, very large feature vectors for the examples can be used. Experimental results on three classification tasks show that using the hierarchical structure of multi-category labelings leads to improved performance over the more traditional approach of combining individual binary classifiers.

Our future work includes generalization of the approach to general graph structures and looking for ways to scale up the method further.

REUTERS	Level 0	Level 1	Level 2	Level 3
SVM	92.4/89.4/90.9	96.8/38.7/55.3	98.1/49.3/65.6	81.8/46.2/59.0
H-SVM	92.4/89.4/90.9	93.7/43.6/59.5	91.1/61.5/73,4	72.0/46.2/56,3
H-RLS	93.2/89.1/**91.1**	90.9/46.8/61.8	89.7/64.8/**75.2**	76.0/48.7/59.4
H-M^3-ℓ_Δ	94.1/83.0/88.2	87.3/48.9/62.7	91.1/63.2/74.6	79.4/69.2/73.9
H-M^3-$\ell_{\tilde{H}}$	91.1/87.8/89.4	79.2/53.1/**63.6**	85.4/66.6/74.8	77.9/76.9/**77.4**
WIPO-alpha	Level 0	Level 1	Level 2	Level 3
SVM	100/100/100	92.1/77.7/84.3	84.4/42.5/56.5	82.1/12.8/22.1
H-SVM	100/100/100	92.1/77.7/84.3	79.6/51.1/62.2	77.0/24.3/36.9
H-RLS	100/100/100	91.3/79.1/84.8	78.2/57.0/65.9	72.6/29.6/42.1
H-M^3-ℓ_Δ	100/100/100	90.8/80.2/85.2	86.1/50.0/63.3	72.1/31.0/43.4
H-M^3-$\ell_{\tilde{H}}$	100/100/100	90.9/80.4/**85.3**	76.4/62.3/**68.6**	60.4/39.7/**47.9**
ENZYME	Level 0	Level 1	Level 2	Level 3
SVM	100/100/100	84.3/4.9/9.3	100/0.4/0.8	100/0.3/0.6
H-SVM	100/100/100	84.3/4.9/9.3	72.3/1.9/3.7	67.5/1.5/2.9
H-RLS	100/97.4/98.7	33.0/39.3/35.9	22.4/22.6/22.5	15.2/17.0/16.0
H-M^3-ℓ_Δ	100/100/100	61.2/30.8/41.0	49.8/13.3/21.0	52.9/4.7/8.6
H-M^3-$\ell_{\tilde{H}}$	100/100/100	49.3/56.0/**52.4**	21.5/42.5/**28.6**	14.7/35.2/**20.7**

Table 3: Precision/Recall/F1 statistics for each level of the hierarchy for different algorithms on Reuters RCV1 (top), WIPO-alpha (middle), and ENZYME datasets (bottom).

Acknowledgments

The authors gratefully acknowledge the insightful comments by the anonymous referees. We also wish to thank Esa Pitkänen for his help in preparing the datasets. This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. Juho Rousu has been supported by the European Union Marie Curie Fellowship grant HPMF-CT-2002-02110 and the work was partly conducted when he was visiting Royal Holloway University of London.

References

S. M. Aji and R. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden markov support vector machines. In *International Conference of Machine Learning*, 2003.

D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

L. Cai and T. Hofmann. Hierarchical document categorization with support vector machines. In *13 ACM CIKM*, 2004.

N. Cancedda, E. Gaussier, C. Goutte, and J.-M. Renders. Word-sequence kernels. *Journal of Machine Learning Research*, 3:1059–1082, 2003.

N. Cesa-Bianchi, C. Gentile, A. Tironi, and L. Zaniboni. Incremental algorithms for hierarchical classification. In *Neural Information Processing Systems*, 2004.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

O. Dekel, J. Keshet, and Y. Singer. Large margin hierarchical classification. In *ICML'04*, pages 209–216, 2004.

S. T. Dumais and H. Chen. Hierarchical classification of web content. In *SIGIR'00*, pages 256–263, 2000.

T. Gartner. A survey of kernels for structured data. *ACM SIGKDD Explorations*, pages 49–58, 2003.

T. Hofmann, L. Cai., and M. Ciaramita. Learning with taxonomies: Classifying documents and words. In *NIPS Workshop on Syntax, Semantics, and Statistics*, 2003.

T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning*, pages 137 – 142, Berlin, 1998. Springer.

D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *ICML'97*, pages 170–178, 1997.

F. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498–519, 2001.

J. Lafferty, X. Zhu, and Y. Liu. Kernel conditional random fields: representation and clique selection. In *Proc. 21th International Conference on Machine Learning*, pages 504–511, 2004.

C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 564 – 575, 2002.

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

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, February 2002.

A. McCallum, R. Rosenfeld, T. Mitchell, and A. Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *ICML'98*, pages 359–367, 1998.

J. Rousu and J. Shawe-Taylor. Efficient computation of gapped substring kernels on large alphabets. *JMLR*, 6:1323–1344, 2005.

G. Salton. *Automatic Text Processing*. Addison-Wesley, Massachusetts, 1989.

C. Saunders, H. Tschach, and J. Shawe-Taylor. Syllables and other string kernel extensions. In *ICML'02*, pages 530–537, 2002.

J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Neural Information Processing Systems 2003*, 2003.

B. Taskar, V. Chatalbashev, and D. Koller. Learning associative markov networks. In *Proc. 21th International Conference on Machine Learning*, pages 807–814, 2004.

I. Tsochantaridis, T. Hofmann, T. Joachims, and Y.n Altun. Support vector machine learning for interdependent and structured output spaces. In *Proc. 21th International Conference on Machine Learning*, pages 823–830, 2004.

S. V. N. Vishwanathan and A. J. Smola. Fast kernels on strings and trees. In *Proceedings of Neural Information Processing Systems 2002*, 2002. URL `http://users.rsise.anu.edu.au/ vishy/papers/VisSmo02.pdf`.

M. Wainwright and M. Jordan. Graphical models, exponential families, and variational inference. Technical Report 649, Department of Statistics, University of California, Berkeley, 2003.

M. Wainwright, T. Jaakkola, and A. Willsky. Tree-based reparameterization framework for analysis of sum-product and related algorithms. *IEEE Transactions on information theory*, 49:1120–1146, May 2003.

WIPO. *World Intellectual Property Organization. http://www.wipo.int/classifications/en*. 2001.