Learning to Rank from Structures in Hierarchical Text Classification

Qi Ju[1], Alessandro Moschitti[1], and Richard Johansson[2]

[1] DISI, University of Trento, Italy
[2] Department of Swedish, University of Gothenburg, Sweden
{qi,moschitti}@disi.unitn.it, richard.johansson@gu.se

Abstract. In this paper, we model learning to rank algorithms based on structural dependencies in hierarchical multi-label text categorization (TC). Our method uses the classification probability of the binary classifiers of a standard top-down approach to generate k-best hypotheses. The latter are generated according to their global probability while at the same time satisfy the structural constraints between father and children nodes. The rank is then refined using Support Vector Machines and tree kernels applied to a structural representation of hypotheses, i.e., a hierarchy tree in which the outcome of binary one-vs-all classifiers is directly marked in its nodes. Our extensive experiments on the whole Reuters Corpus Volume 1 show that our models significantly improve over the state of the art in TC, thanks to the use of structural dependecies.

1 Introduction

Hierarchical text categorization shows interesting real-world applications, e.g., Yahoo! Categories and Dmoz. These involve a large number of categories and documents, making traditional multi-label classification methods, e.g., one-versus-all, inadequate. To produce a *sufficient* classification accuracy in such conditions, the structure of the hierarchy must be taken into account. This is not straightforward as hierarchical classifiers often impose a number of simplifying restrictions on their models. In particular, category assignments are normally assumed to be conditionally independent. The probability of a document d belonging to a subcategory C_i of a category C is assumed to depend only on d and C, but not on other subcategories of C, or any other categories in the hierarchy. If other dependencies between categories are introduced, the maximization step becomes computationally intractable.

Previous work has tackled the problem by introducing dependencies between labels without exploiting hierarchical structures, e.g., SVM-struct in [26,9] optimized with respect to the output label subset and [10] learned meta-classifiers by exploiting dependency features (between category labels). In contrast, [23] exploited hierarchical dependencies (not just label dependencies) but the algorithm was computationally expensive. Indeed, they could only experiment with 34 categories out of 103 of the Reuters Corpus Volume 1 collection (RCV1) [15] and a small document subset of the entire corpus.

To our knowledge, our approach in [19] is the only one using a hierarchy structure on large data. We designed an efficient model based on a simple generator of hypotheses and a reranking algorithm based on hierarchy structure features from tree kernel spaces. This model improved the state of the art on the entire RCV1. However, the generation of hypotheses did not take into account the structure of the hierarchy. This means that many hypotheses could violate the hierarchy constraints, providing the reranker with many wrong hypotheses (this is a shortcoming as, in principle, such hypotheses can be filtered out). Additionally, we only measured the accuracy with standard Micro-and Macro-average F1. This does not guarantee that the model with the highest F1 produces the best hierarchical classifications. For example, such measure does not consider that mistaking a leaf node by a first level node is a much worse error than mistaking a node for one of its siblings.

In this paper, we propose an efficient and more accurate *reranker* than in [19]. The new reranker is based on our new algorithm for the generation of the top k category assignments. This satisfies the hierarchical constraints between different node classifications during the generation by providing a more precise global hypothesis probability. The better set of hypotheses is then reranked similarly to what proposed in [19], i.e., Support Vector Machines (SVMs) using tree kernels classify pairs of hypotheses represented with trees. The latter describe both hierarchy and classification labels. Our algorithm along with our fast tree kernel representation can be applied to large hierarchies. Note that there can be other ways to impose hierarchical constrained, e.g., Conditional Random Fields [14], but our reranker can be applied on top of any model: it can exploit any basic classifier that provides prediction scores, provided that the latter are converted into probabilities. We carried out experiments on the entire hierarchy (103 nodes organized on 5 levels) of the well-known RCV1. As a first step, we evaluated the accuracy of our reranker on a setting comparable with previous work in TC. Then, we explore the accuracy and the efficiency of several reranking models by showing that our rerankers consistently and significantly improve on the traditional approaches to TC up to about 5 absolute percent points. Finally, we also trained our reranker according to the Multi-label Graph-Induced Error (MGIE) [27], which is a standard hierarchal measure. The results show that our reranker outperforms our previous results and can be easily optimized to improve on any structural measure.

In the remainder of this paper, Section 2 introduces preliminaries for the hypothesis generation algorithm, which is then presented in Section 3. Section 4 illustrates our reranking approach based on tree kernels, Section 5 reports on our experiments, and finally Section 6 derives the conclusions.

2 Global Classification Hypotheses from Binary Decisions

The idea of the paper is to build efficient models for hierarchical classification using global dependencies. For this purpose, we use reranking models, which encode global information. These necessitate of a set of initial hypotheses, which are typically generated by *local classifiers*. In our study, we used n one-vs-all binary classifiers, associated with the n different nodes of the hierarchy. In the following sections, we show a *hierarchical* algorithm in which the structure imposes constrains on the feasibility of the hypotheses.

2.1 Structural Generation from a Hierarchy

The generation process becomes more complex when the hierarchy is taken into account. Indeed, if d belongs to a category C, then it also implicitly belongs to all supercategories of C, including the top category T. We consider tree-shaped hierarchies and leave the extension to general DAG-shaped category systems to future work. To take into consideration a tree structure, we base our model on the computation of two types of probabilities. Firstly, for a given document d, and a category C with subcategories $C_1, \ldots, C_n$, we define the *stop probability* as the probability of "stopping" at C, i.e., that d does not belong to any of the subcategories of C: $p_s(C) = P(d \notin C_1 \ldots d \notin C_n | d \in C)$. Secondly, in the case where we know that at least one subcategory has been selected, we can compute the probabilities of selecting a particular subcategory: $p_{C_i}(C) = P(d \in C_i | d \in C(d \in C_1 \vee \ldots \vee d \in C_n)), i \in \{1, \ldots, n\}$. At this stage, we assume conditional independence between the subcategories, so the probability will depend only on the document and the supercategory.

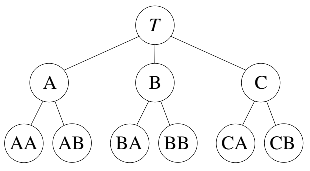

Fig. 1. Example of a hierarchy

These probabilities can be used to compute the probability of a complete assignment of categories to a document. To exemplify, consider the hierarchy in Figure 1. To compute the probability of a document d belonging to the categories AB and C (and then also implicitly to T and A) but not to AA, B, CA, or CB, we decompose the probability using the above-mentioned conditional probabilities: $(1-p_s(T)) \cdot p_A(T) \cdot (1-p_B(T)) \cdot p_C(T) \cdot (1-p_s(A)) \cdot (1-p_{AA}(A)) \cdot p_{AB}(A) \cdot p_s(C)$. The next section presents hypothesis generation exploiting this decomposition.

3 Generation the Top *k* Classifications in a Hierarchy

The number of category assignments is exponential in the number of categories, so for any nontrivial hierarchy a brute-force search to find the best hypothesis is not applicable. However, the independence assumptions ensure that the search space is decomposable so that the best assignment – and the k best assignments – can be found quickly. Similar to the fastest k-best algorithm for natural language parsing [11], our algorithm proceeds in two steps: We first find the best assignment, and then we construct the k-best list by incremental modifications.

3.1 Generation of the Top Hypothesis

We first describe the function Top1 that finds the category assignment having the highest probability. The algorithm works top-down, and due to the conditional independence assumptions, we can find optimal assignments in subtrees

Algorithm 1. Generation of the top hypothesis

```
function Top1($C$)
    // Returns the top hypothesis and its probability
    if $p_s(C) \geq 0.5$
        return $\langle \{C\}, p_s(C) \rangle$
    $\langle S, P \rangle \leftarrow$ MaxSubcats($C$)
    if $S = \emptyset$
        $\langle S, P \rangle \leftarrow$ MaxOneSubcat($C, P$)
    if $p_s(C) \geq P$
        return $\langle \{C\}, p_s(C) \rangle$
    else
        return $\langle \{C\} \cup S, P \rangle$

function MaxOneSubcat($C, P$)
    $q_{min} \leftarrow \infty$
    for each subcategory $C_i \in$ Sub($C$)
        $\langle S_i, P_i \rangle \leftarrow$ Top1($C_i$)
        $q_i \leftarrow (1 - p_{C_i}(C))/(P_i \cdot p_{C_i}(C))$
        if $q_i < q_{min}$
            $q_{min} \leftarrow q_i$, $S_{min} \leftarrow S_i$
    return $\langle S_{min}, P/q_{min} \rangle$

function MaxSubcats($C$)
    $S \leftarrow \emptyset$, $P \leftarrow 1 - p_s(C)$
    for each subcategory $C_i \in$ Sub($C$)
        if $p_{C_i}(C) \leq 0.5$
            $P \leftarrow P \cdot (1 - p_{C_i}(C))$
        else
            $\langle S_i, P_i \rangle \leftarrow$ Top1($C_i$)
            if $p_{C_i}(C) \cdot P_i > (1 - p_{C_i}(C))$
                $P \leftarrow P \cdot p_{C_i}(C) \cdot P_i$
                $S \leftarrow S \cup S_i$
            else
                $P \leftarrow P \cdot (1 - p_{C_i}(C))$
    return $\langle S, P \rangle$
```

Algorithm 2. Generation of the top k hypotheses

```
function TopK($C, k$)
    // Returns the top $k$ hypotheses and their
       probabilities
    $H \leftarrow \emptyset$
    $q \leftarrow$ empty priority queue
    Enqueue($q$, Top1($C$))
    while $|H| < k$ and $q$ is nonempty
        $\langle S, P \rangle \leftarrow$ Dequeue($q$)
        $H \leftarrow H \cup \{\langle S, P \rangle\}$
        if $|H| < k$
            for each $h \in$ Succs($C, P, S$)
                Enqueue($q, h$)
    return $H$

function Succs($C, P, S$)
    // Returns the set of modifications of
       the hypothesis $S$
    if $C$ has no subcategory
        return $\emptyset$
    $H \leftarrow \emptyset$
    if $S \neq \{C\}$
        Stop($C, P, S, H$)
        EnableEachSubcat($C, P, S, H$)
        DisableEachSubcat($C, P, S, H$)
        SubcatSuccs($C, P, S, H$)
    else
        Unstop($C, P, S, H$)
    return $H$
```

independently of each other. At each node, we check whether the stop probability is higher than the probability of enabling at least one subcategory; the probability of each subcategory is computed recursively. To cut the search space, the algorithm exploits the fact that if the stop probability p_s is greater than 0.5, the probability of entering any subcategory, $(1 - p_s) \cdot p_{C_i}$, is guaranteed to be less than 0.5.[1]

Algorithm 1 shows the pseudocode. Here, the function Sub returns the subclasses of a given class C. While the algorithm is straightforward; note that the optimal assignment is not necessarily what we would get by a greedy algorithm selecting the highest probability assignment at each choice point. In practice, the implementation will cache the probabilities and maximal assignments to avoid redundant recomputations. For brevity, we omit the caching from the pseudocode.

[1] The algorithm can be rewritten without this trick to generalize to non probabilistic scores.

Algorithm 3. One-step modifications of a hypothesis

```
procedure SubcatSuccs($C, P, S, H$)
  for each subcategory $C_i \in$ Sub($C$)
    if $C_i \in S$
      $P_i \leftarrow$ ProbSubcats($S, C_i$)
      $S_i \leftarrow S \cap$ Subtree($C_i$)
      for each $\langle S_s, P_s\rangle \in$ Succs($C_i, P_i, S_i$)
        $H \leftarrow H \cup \{\langle (S \setminus S_i) \cup S_s, P/P_i \cdot P_s\rangle\}$

procedure Stop($C, P, S, H$)
  $P' \leftarrow P \cdot p_s(C)/(1 - p_s(C))$
  for each subcategory $C_i \in$ Sub($C$)
    if $C_i \in S$
      $P' \leftarrow P'/p_{C_i}(C)$
      $P' \leftarrow P'/$ProbSubcats($S, C_i$)
    else
      $P' \leftarrow P'/(1 - p_{C_i}(C))$
  $H \leftarrow H \cup \{\langle\{C\}, P'\rangle\}$

procedure Unstop($C, P, S, H$)
  $\langle S_s, P_s\rangle \leftarrow$ MaxSubcats($C$)
  if $S_s = \emptyset$
    $\langle S_s, P_s\rangle \leftarrow$ MaxOneSubcat($C, P$)
  $P' \leftarrow P \cdot (1 - p_s(C)) \cdot P_s/p_s(C)$
  $H \leftarrow H \cup \{\langle S \cup S_s, P'\rangle\}$

procedure EnableEachSubcat($C, P, S, H$)
  for each subcategory $C_i \in$ Sub($C$)
    if $C_i \notin S$
      $\langle S_i, P_i\rangle \leftarrow$ Top1($C_i$)
      $P' \leftarrow P \cdot p_{C_i}(C) \cdot P_i/(1 - p_{C_i}(C))$
      $H \leftarrow H \cup \{\langle S \cup S_i, P'\rangle\}$

procedure DisableEachSubcat($C, P, S, H$)
  for each subcategory $C_i \in$ Sub($C$)
    if $C_i \in S$
      $P' \leftarrow P \cdot (1 - p_{C_i}(C))$
      $P' \leftarrow P'/p_{C_i}(C)/$ProbSubcats($S, C_i$)
      $S' \leftarrow S \setminus$ Subtree($C_i$)
      if $S' \neq \{C\}$
        $H \leftarrow H \cup \{\langle S', P'\rangle\}$
      else if $P' \geq P$
        EnableEachSubcat($C, P', S', H$)
```

3.2 Hypothesis Expansion

The algorithm TopK to generate the k top hypotheses (Algorithm 2) relies on the fact that conditional independence between siblings ensure that the search space is monotonic. The hypothesis at position i in the list of hypotheses is then a one-step modification of one of the first $i - 1$ hypotheses. To generate k hypotheses, we thus start with the most probable one and put it into a priority queue ordered by probability. Until we have found k hypotheses, we pop the front item and put it into the output list. We then apply the function Succs to find all one-step modifications of the item, and we add them all back to the queue.

The Succs function applies the following one-step modification operations: SubcatSuccs, which recursively computes a one-step modification of every enabled subcategory; Stop, which changes an assignment with subcategories to a stop; Unstop, which enables at least one subcategory of an assignment without subcategories; EnableEachSubcat, which generates multiple hypotheses by enabling every disabled subcategory; and finally DisableEachSubcat, which conversely disables every enabled subcategory. The pseudocode for the modification operations is shown in Algorithm 3. The pseudocode uses two auxiliary functions: Subtree(C), which returns the set of categories that are subcategories of C, and ProbSubcats, which returns the (previously computed) probability of an assignment of a set of subcategories. Again, the pseudocode omits possible optimizations, such as ignoring assignments that have already been processed.

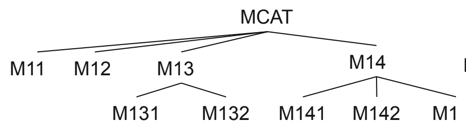

Fig. 2. A subhierarchy of Reuters

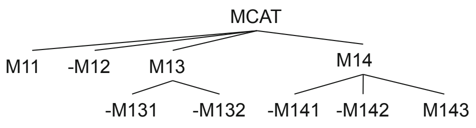

Fig. 3. A tree representing a category assignment hypothesis for the subhierarchy in Fig. 2

3.3 Efficiency of the Hypothesis Set Generation

The complexity of the algorithm is $O(ks \log(ks))$ where s is the maximal number of modified items generated by the SUCCS function, since the complexity of the ENQUEUE operation is logarithmic in a standard priority queue. A non-tight upper bound on s is $2N$, where N is the number of nodes in the hierarchy, but this is of limited interest: in practice, the number of modified items will be much smaller, and depends on parameters such as the shape of the hierarchy and the number of enabled subcategories in an assignment. However, it is clear that the algorithm is able to handle very large hierarchies even in the worst case.

The bottleneck in practice will typically be the call to the probability estimation procedure, and we note that the worst case – for 1-best as well as k-best generation – occurs when we have to estimate all probabilities in the hierarchy. The number of estimations in a hierarchy of N nodes is at most $N - 1$ stop probabilities and $N - 1$ subcategory probabilities; note that these two worst-case numbers do not occur at the same time. However, since we generate the probabilities only when we need them, the number of estimations will typically be much smaller in practice. How much of the hierarchy we actually need to explore will of course depend on the particular probabilities.

4 Structural Reranker for Hierarchical Classification

In this section we provide a representation from which the dependencies between the different nodes of the hierarchy can be learned. As an example let us consider the Reuters categorization scheme. Figure 2 shows a subhierarchy of the *Markets* (MCAT) category and its subcategories: *Equity Markets* (M11), *Bond Markets* (M12), *Money Markets* (M13) and *Commodity Markets* (M14). These also have subcategories: *Interbank Markets* (M131), *Forex Markets* (M132), *Soft Commodities* (M141), *Metals Trading* (M142) and *Energy Markets* (M143).

Representing such hierarchy and the dependencies between their nodes in a learning algorithm is not a trivial matter. Possible features are node subsets of the hierarchy but: (i) their exhaustive generation produces an exponential number of features, which is computationally infeasible; and (ii) the node order as well as ancestor and sibling relations are lost. Since, to our knowledge, no previous work has already addressed the TC hierarchy reranking, we may only

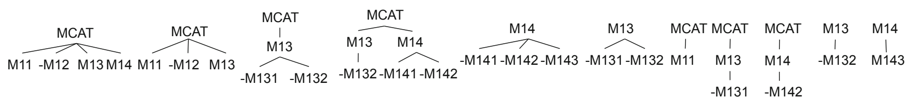

Fig. 4. Some tree fragments of the hypothesis in Fig. 3

start exploring some reasonable features provided for other structured output tasks. For example, trigrams and bigrams in parse-tree reranking [4].

However, even in such cases, we have too many options to explore. For example, which node pairs should the path be extracted from? Which nodes should be part of the n-grams? We found much simpler to employ tree kernels for automatically generating all possible features (hierarchy fragments) in a way similar to parse tree reranking [5]. In addition to a tree representation, the input of tree kernels must also take into consideration the categories assigned to a given document. For this purpose, we mark the *negative* assignments of the current hypothesis in the node labels with “-”, e.g., -M142 means that the document was not classified in *Metals Trading*. For example, Figure 3 shows the representation of a classification hypothesis consisting in assigning the target document to the categories MCAT, M11, M13, M14 and M143. By applying the partial tree kernel (PTK) [18] to such labeled tree all possible dependency features are generated. For example, Fig. 4 shows some of the tree fragments (features from the hypothesis of Fig. 3, encoding label dependencies).

5 Experiments

We show that several reranking models based on tree kernels can improve the state of the art in TC. For this purpose, we experimented with RCV1, Lewis et al.'s setting [15], by measuring the accuracy of different models according to it on the Lewis' split. We also optimized our ranker for MGIE and measured the models accuracy according to it.

5.1 Setup

We used the full hierarchy of Reuters Volume 1 (RCV1)[2] TC corpus. To compare with previous work we considered, the Lewis' split [15], which includes 23,149 news for training and 781,265 for testing. The training sets are used for learning the binary classifiers needed to build the multiclass-classifier (MCC). We used the state-of-the-art method used by [15] for RCV1, i.e.,: SVMs with the default parameters (trade-off and cost factor = 1), linear kernel, normalized vectors, stemmed bag-of-words representation, $log(TF + 1) \times IDF$ weighting scheme and stop list[3]. We used the LIBSVM[4] implementation, which provides

[2] `trec.nist.gov/data/reuters/reuters.html`

[3] We have just a small difference in the number of tokens, i.e., 51,002 vs. 47,219 but this is both not critical and rarely achievable (diverse stop lists or tokenizers).

[4] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

Table 1. Comparison between our rankers on the entire Topic hierarchy of RCV1 exactly using Lewis' split and data (Lewis' model maximizing Micro-F1 is reported).

F1	baseline			our Rerankers		
	Lewis, flat	Ours, flat	Ours, hier	SeqRR	FRR	HRR
Micro-F1	0.816	0.815	0.819	0.828	0.849	0.855
Macro-F1	0.567	0.566	0.578	0.590	0.615	0.634

a probabilistic output of the classification function. The classifiers are combined using the one-vs.-all approach, which is also state of the art as argued in [22]. Since the task requires to assign multiple labels, we simply collect the decisions of the n classifiers: this constitutes our MCC baseline.

Regarding the reranker, we divided the training set in two chunks of data: Train1 and Train2. The binary classifiers are trained on Train1 and tested on Train2 (and vice versa) to generate the hypotheses on Train2 (Train1). The union of the two sets constitutes the training data for the reranker. We implemented three rerankers: flat RR (**FRR**), using the simple flat hypothesis generation (see, e.g., [19]) and the representation type described in Fig. 3; hierarchical RR (**HRR**) using the hierarchical hypothesis generation (see Sec. 2) and the same representation of FRR; and **SeqRR**, which is a sequence kernel applied to the sequence of labels in a hypothesis, lexicographically ordered, e.g., we associated the hypothesis in Fig. 3 with the following sequence: ⟨M11, -M12, M13, -M131, -M132, M14, -M141, -M142, M143, MCAT⟩.

The rerankers are based on SVMs and the Preference Kernel (P_K) [12] built on top of PTK [18] or a sequence kernel. The latter is applied to the tree-structured hypotheses. We also add a linear kernel to P_K, which is applied to unidimensional vectors containing the probability of the hypothesis (computed as explained in Sec. 2). We trained the rerankers using SVM-light-TK[5], a structural kernel toolkit based on SVM-light [12], which allows for using PTK on pairs of trees and combining them with kernel-based vectors. Again we use default parameters to facilitate replicability and preserve generality. In all experiments, if not mentioned, always 8 hypotheses are used. All the performance figures are provided by means of Micro- and Macro-Average F1, evaluated from our test data over all 103 categories. Finally, we assessed the statistical significance of our results by using the model described in [29] and implemented in [20].

5.2 Classification Accuracy on Whole Reuters

In the first experiments, we used the Lewis' split. The results are reported in Table 1, whose columns have the following meaning: (i) Lewis' flat refers to the result achieved in Lewis et al. paper (the best Micro-F1 that they achieved); (ii) *Ours, flat* is our reimplementation of the Lewis et al. MCC, i.e., a one-vs-all multi-classifier using the same Lewis' setting; (iii) *Hier* goes beyond the

[5] `disi.unitn.it/moschitti/Tree-Kernel.htm`

flat model as it is a top down algorithm so already exploiting the classification hierarchy; (vi) FRR and HRR are our kernel-based reranking models applied to hypotheses generated with a flat or structural algorithm; and SeqRR is a sequence kernel reranking "flat generated" hypotheses.

Our flat MCC achieved a Micro-F1 of 81.5, which basically matches the 81.6 reported in [15]. The top down model slightly improves the flat models, i.e., 81.9-81.5=0.4. This is significant with $p=10^{-5}$ (please consider that the test set contains about 800k examples). When FRR is used on top of the baseline, we improved it by 3.4 absolute percent points (significant at $p=10^{-5}$), i.e., 84.9-81.5=3.4. The hierarchical generation of hypotheses seems to be beneficial as we obtain another statistical significant delta of 0.6 (significant at $p=10^{-5}$). The improvement on the Macro-average follows a similar pattern.

The SeqRR, only relying on label subset features, improves the baselines, i.e., flat and top down models, but it is outperformed by FRR, which exploits hierarchical structural dependencies, i.e., the above feature but within a structure.

Very interestingly, HRR generates better hypotheses as the reranker using the same features of FRR can achieve a slightly better accuracy (e.g., 0.855 - 0.849 = +0.6%, statistical significant result).

5.3 Discussion and Related Work

Ideally a comparison with other hierarchical models would be needed to better assess the benefit of our approach. This is not always simple as not all previous work follows the standard training/test split of RCV1. Moreover, previous models tend to be inefficient and this leads to experimentation with only Reuters subparts. For example, the work in [23] is very close to ours. They directly encoded global dependencies in a gradient descendent learning approach. Their approach is less efficient than ours so they could experiment with only CCAT subhierarchy of RCV1, which only contains 34 nodes, achieving lower accuracy than ours. Other relevant work such as [17] and [8] used a rather different dataset and a different idea of dependencies based on feature distributions over the linked categories. In particular, early work on automated hierarchical text categorization, e.g., [8,16,13], simply approached the problem in a top down fashion by recursively creating multi-classifiers for each individual node. This approach is one of the baselines we compare with. [1] defined an algorithm called Refined Experts, which propagates the lower-level category classification up through the hierarchy before applying top-down classification, which thus refines the first classification decisions. This model is obviously generalized by our reranker, which indeed refines the first pass classification of local classifiers, exploiting the classification of the entire structure. [6] used a Bayesian aggregator on the result of the individual binary classifiers, thus also this is generalized by our approach. [28] used a search engine to refine the set of category candidates. This approach works well for a huge number of categories but of course the pre-selection it applies introduces some noise.

Table 2. Oracle performance according to the number of hypotheses

k	Flat Generation		Hierarchical Generation	
	Micro-F_1	Macro-F_1	Micro-F_1	Macro-F_1
1	0.640	0.408	0.640	0.408
2	0.758	0.504	0.771	0.538
4	0.821	0.566	0.835	0.603
8	0.858	0.610	0.869	0.620
16	0.898	0.658	0.917	0.710

Table 3. Hierarchical TC models measured by the Multi-label Graph-Induced Error, using max distant equal to 5 and 7

F1	RCV1-v2				
	baseline	flatSVM	HierSVM	FRR	HRR
max = 5	4.462	1.343	1.322	1.036	0.974
max = 7	5.538	1.824	1.794	1.360	1.234

The work on SVM-struct [26,9] and meta-classifier [10] does not exploit hierarchical dependencies but it can be interesting for a comparison. For this purpose, we implemented the sequence kernel model (SeqRR), which completely subsumes the model in [10] since it generates a superset of the meta-features used in such work. It also approximates [26] as it uses the same subset features of SVM-struct but of course the search space of the latter is far larger than the best hypotheses we generate. Anyhow, according to Table 1, SeqRR improves on the baseline but it is also outperformed by our hierarchal rerankers. [2] used discriminant functions to encode dependencies and to jointly learn a global loss over the hierarchy. Similar online methods were proposed in [7,3]. Again, our reranker approach produces better features and it is in general more efficient.

On a different research line, hierarchical shrinkage in [21,17] estimates parameters in Naïve Bayes classifiers considering the path from the root to the leaf node. A similar idea is presented in [24], where the path above is encoded in multinomial logistic models accounting for Bayesian priors. These methods are generalized by all possible substructures generated by our approach. In [30], the authors enforced each node of the hierarchy to be orthogonal to its ancestors as much as possible in additional to minimizing the loss at individual nodes. [25] presents a survey of hierarchical classification methods.

Finally, our approach has high potential as: (i) it is very efficient since the reranker is constituted by only one binary classifier using efficient tree kernels. For lack of space we do not report our running time study, which shows that thousands of hypotheses can be classified in few seconds. (ii) There is a large margin of improvement for our rerankers as shown in Table 2. It reports the oracle performance with respect to the increasing number of hypotheses (using a RCV1 subset). Oracle accuracy corresponds to the result we would get if we were able to always select the best hypothesis with our reranker. The results also show that the quality of the hierarchically generated hypotheses is better than those generated by the flat method.

5.4 Multi-label Graph-Induced Error

We also demonstrate that our approach is effective for optimizing hierarchical classification by using a hierarchical measure, i.e., a measure that takes into account the different degrees of mistakes. For example, assigning a category to a document, which is sibling of the correct one is less critical than assigning a much more distant node of the hierarchy. The Multi-label Graph-Induced Error

(MGIE) [27] considers the distances between true positives, false positives and false negatives by also limiting this with a maximum distance.

In our experiments, we set the max distance to five and seven. The results are shown in Table 3. The baseline is computed by assigning categories according to their occurrence probability. We note that flatSVM (one-vs-all) is slightly improved by using a top-down approach. The flat reranker, FRR, improves on the previous models and the HRR model exploiting better initial structural hypotheses improves on FRR, suggesting that our rerankers can be tuned up on any measure, especially the hierarchical ones.

6 Conclusions

In this paper, we have described several models for reranking the output of an MCC. We have defined an algorithm for structural hypothesis generation along with a reranker based on structural kernels. Our models can learn to reorder a set of ranked hypotheses based on complex statistical dependencies. It should be noted that this algorithm is based on a simple binary classifier that can efficiently select the best hypothesis. We have seen a consistent improvement over state-of-the-art TC models. Most importantly, our approach (i) is rather general, (ii) can be applied to several other problems or domains and (iii) can be optimized according to several measure, e.g., MGIE. Finally, in short-term future, we would like to compare with other machine learning models and more interestingly to experiment with large-scale corpora, e.g., Dmoz.

Acknowledgments. We would like to thank the anonymous reviewers for their valuable and committed work. This research has been partially supported by the EC's Seventh Framework Programme (FP7/2007-2013) under the grants #247758: EternalS – Trustworthy Eternal Systems via Evolving Software, Data and Knowledge, and #288024: LiMoSINe – Linguistically Motivated Semantic aggregation engiNes.

References

1. Bennett, P.N., Nguyen, N.: Refined experts: improving classification in large taxonomies. In: SIGIR (2009)
2. Cai, L., Hofmann, T.: Hierarchical document categorization with support vector machines. In: CIKM (2004)
3. Cesa-Bianchi, N., Gentile, C., Zaniboni, L.: Incremental algorithms for hierarchical classification. JMLR (2006)
4. Charniak, E., Johnson, M.: Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In: ACL (2005)
5. Collins, M., Duffy, N.: New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In: ACL (2002)
6. DeCoro, C., Barutcuoglu, Z., Fiebrink, R.: Bayesian aggregation for hierarchical genre classification. In: International Symposium on Information Retrieval (2007)

7. Dekel, O., Keshet, J., Singer, Y.: Large margin hierarchical classification. In: ICML (2004)
8. Dumais, S.T., Chen, H.: Hierarchical classification of web content. In: SIGIR (2000)
9. Finley, T., Joachims, T.: Parameter learning for loopy markov random fields with structural support vector machines. In: ICML Workshop (2007)
10. Gopal, S., Yang, Y.: Multilabel classification with meta-level features. In: SIGIR (2010)
11. Huang, L., Chiang, D.: Better k-best parsing. In: IWPT Workshop (2005)
12. Joachims, T.: Making large-scale SVM learning practical. Advances in Kernel Methods – Support Vector Learning (1999)
13. Koller, D., Sahami, M.: Hierarchically classifying documents using very few words. In: ICML (1997)
14. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: ICML (2001)
15. Lewis, D.D., Yang, Y., Rose, T., Li, F.: Rcv1: A new benchmark collection for text categorization research. JMLR (2004)
16. Liu, T.Y., Yang, Y., Wan, H., Zeng, H.J., Chen, Z., Ma, W.Y.: Support vector machines classification with a very large-scale taxonomy. SIGKDD Explorations (2005)
17. McCallum, A., Rosenfeld, R., Mitchell, T.M., Ng, A.Y.: Improving text classification by shrinkage in a hierarchy of classes. In: ICML (1998)
18. Moschitti, A.: Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 318–329. Springer, Heidelberg (2006)
19. Moschitti, A., Ju, Q., Johansson, R.: Modeling topic dependencies in hierarchical text categorization. In: ACL (2012)
20. Padó, S.: User's guide to `sigf`: Significance testing by approximate randomisation (2006)
21. Punera, K., Ghosh, J.: Enhanced hierarchical classification via isotonic smoothing. In: WWW (2008)
22. Rifkin, R., Klautau, A.: In defense of one-vs-all classification. JMLR (2004)
23. Rousu, J., Saunders, C., Szedmak, S., Shawe-Taylor, J.: Kernel-based learning of hierarchical multilabel classification models. JMLR (2006)
24. Shahbaba, B., Neal, R.M.: Improving classification when a class hierarchy is available using a hierarchy-based prior. Tech. rep., Bayesian Analysis (2005)
25. Silla Jr., C.N., Freitas, A.A.: A survey of hierarchical classification across different application domains. In: DMKD (2011)
26. Tsochantaridis, I., Hofmann, T., Joachims, T., Altun, Y.: Support vector machine learning for interdependent and structured output spaces. In: ICML (2004)
27. Tsoumakas, G., Katakis, I., Vlahavas, I.: Random k-labelsets for multi-label classification. In: TKDE (2011)
28. Xue, G.R., Xing, D., Yang, Q., Yu, Y.: Deep classification in large-scale text hierarchies. In: SIGIR (2008)
29. Yeh, A.S.: More accurate tests for the statistical significance of result differences. In: COLING (2000)
30. Zhou, D., Xiao, L., Wu, M.: Hierarchical classification via orthogonal transfer. In: ICML (2011)