PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

Large-scale Structural Reranking for Hierarchical Text Categorization

Qi Ju

Advisor:

Prof. Alessandro Moschitti

Università degli Studi di Trento

Co-Advisor:

Prof. Fausto Giunchiglia

Università degli Studi di Trento

April 2013

Abstract

Current hierarchical text categorization (HTC) methods mainly fall into three directions: (1) Flat one-vs.-all approach, which flattens the hierarchy into independent nodes and trains a binary one-vs.-all classifier for each node. (2) Top-down method, which uses the hierarchical structure to decompose the entire problem into a set of smaller sub-problems, and deals with such sub-problems in top-down fashion along the hierarchy. (3) Big-bang approach, which learns a single (but generally complex) global model for the class hierarchy as a whole with a single run of the learning algorithm. These methods were shown to provide relatively high performance in previous evaluations. However, they still suffer from two main drawbacks: (1) relatively low accuracy as they disregard category dependencies, or (2) low computational efficiency when considering such dependencies.

In order to build an accurate and efficient model we adopted the following strategy: first, we design advanced global reranking models (GR) that exploit structural dependencies in hierarchical multi-label text classification (TC). They are based on two algorithms: (1) to generate the k-best classification of hypotheses based on decision probabilities of the flat one-vs.-all and top-down methods; and (2) to encode dependencies in the reranker by: (i) modeling hypotheses as trees derived by the hierarchy itself and (ii) applying tree kernels (TK) to them. Such TK-based reranker selects the best hierarchical test hypothesis, which is naturally represented as a labeled tree. Additionally, to better investigate the role of category relationships, we consider two interesting cases: (i) traditional schemes in which node-fathers include all the documents of their child-categories; and (ii) more general schemes, in which children can include documents not belonging to their fathers.

Second, we propose an efficient local incremental reranking model (LIR), which combines a top-down method with a local reranking model for each sub-problem. These local rerankers improve the accuracy by absorbing the local category dependencies of sub-problems, which alleviate the errors of top-down method in the higher levels of the hierarchy. The application of LIR recursively deals with the sub-problems by applying the corresponding local rerankers in top-down fashion, resulting in high efficiency.

In addition, we further optimize LIR by (i) improving the top-down method by creating local dictionaries for each sub-problem; (ii) using LIBLINEAR instead of LIBSVM; and (iii) adopting the compact representation of hypotheses for learning the local reranking model. This makes LIR applicable for large-scale hierarchical text categorization.

The experimentation on different hierarchical datasets has shown promising enhancements by exploiting the structural dependencies in large-scale hierarchical text categorization.

Keywords
Text Categorization, Hierarchical Dependencies, Support Vector Machines, Kernel Methods

Contributions and publications

This work has been developed in collaboration with Prof. Alessandro Moschitti, Prof. Fausto Giunchiglia, Richard Johansson, Feroz Farazi, Alexandre Kandalintsev.

This dissertation makes the following contributions:

1) A probabilistic LIBLINEAR is developed which can generate the decision probability in the same way as the probabilistic LIBSVM in both classification and regression. Most importantly, probabilistic LIBLINEAR provides high efficiency while preserving the same accuracy.

2) Evidence of the promising use of text categorization (TC) as a support for an interesting and high level human activity in the educational context.

3) A simple and efficient algorithm for producing k-best classification hypotheses. This algorithm is based on the decision probabilities of local binary classifiers on each node in the hierarchy.

4) A new and complex algorithm for generating k-best hypotheses that considers the structural dependencies of the hierarchy is proposed.

5) Global and compact kernel-based representations for representing the generated k-best hypotheses, which encode the nodes interdependencies in the tree-shaped hierarchy.

6) Several tree kernel-based rerankers with structural kernels, i.e., SK, STK and PTK, which are applied to pairs of hypotheses generated by a simple generation algorithm defined in Point 3.

7) Several hierarchical rerankers with different structural kernels based on hypotheses generated with the algorithm defined in Point 4.

8) Efficient and accurate local incremental reranking model for large scale hierarchical dataset.

Part of the material of this dissertation has been published in:

- [55] Qi Ju, Alessandro Moschitti, and Richard Johansson. *Learning to Rank from Structures in Hierarchical Text Classification.* In Proceedings of the 34th European Conference on Information Retrieval (ECIR'13). Moscow, Russia, 2013.
- [54] Qi Ju and Alessandro Moschitti. *Incremental Reranking for Hierarchical Text Classification.* In Proceedings of the 34th European Conference on Information Retrieval (ECIR'13). Moscow, Russia, 2013.
- [80] Alessandro Moschitti, Qi Ju, and Richard Johansson. *Modeling Topic Dependencies in Hierarchical Text Categorization.* In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL 2012), pp. 759-767. Jeju, Republic of Korea, 2012.
- [56] Qi Ju, Chiara Ravagni, Alessandro Moschitti, and Giampiero Vaschetto. *Hierarchical Text Classification for Supporting Educational Programs.* In Proceedings of the 3rd Italian Information Retrieval Workshop. Bari, Italy, January 26-27, 2012;
- [53] Qi Ju, Richard Johansson, and Alessandro Moschitti. *Towards Using Reranking in Hierarchical Classification.* In Proceedings of the Joint ECML/PKDDPASCAL Workshop on Large-Scale Hierarchical Classification. Athens, Greece, 2011.

Acknowledgments

Above all I would like to thank my advisor Prof. Alessandro Moschitti for his teachings over the course of these PhD years, especially in relation to how to conduct research and how to turn ideas into research papers. I am grateful for the innumerable days he has spent patiently teaching me how to organize, structure, and improve my writing (although, I still have a long path to cover in these directions). He introduced me to the emerging and compelling research field of hierarchical text categorization. Without his support, inspiration, and encouragement, it would have been impossible to finish this thesis.

I would also like to thank Prof. Fausto Giunchiglia for giving me the initial research objectives and co-advising me the first years of my PhD study. I am grateful for his continuous support and feedback, in particular during the qualifying exam.

I am thankful to Richard Johansson for helping me in defining the hypothesis generation model, and to Feroz Farazi for his continuous helps and feedbacks in Java coding. Meaningful discussions with them allowed me to finish such a large volume of work in a reasonable time.

I am also thankful to iKernels group members for all the discussions we have had on every Friday, particularly I thank Aliaksei Severyn and Alexandre Kandalintsev for their technical supports in the experimental work of this thesis.

Finally, I would like to express my deep gratitude to my wife Xue Xiaofei, who resigned her job as a college lecturer in China to accompany me and encourage me to go for doctoral studies. Without her spiritual support, I would not have been able to smoothly complete my PhD studies.

This dissertation work 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.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Hierarchical Organization of Data

As the number of categories grows, it becomes increasingly difficult for people to manage the related classification scheme, e.g., browsing or searching the desired information. To solve this problem, one effective and prevalently adopted way is to organize categories in category hierarchies. This has been proposed by G. W. Leibniz since 1704:

The art of ranking things in gender and species is of no small importance and very much assists our judgment as well as our memory. You know how much it matters in botany, not to mention animals and other substances, or again moral and notional entities as some call them. Order largely depends on it, and many good authors write in such a way that their whole account could be divided and subdivided according to a procedure related to genera and species. This helps one not merely to retain things, but also to find them. And those who have laid out all sorts of notions under certain headings or categories have done something very useful.

Gottfried Wilhelm Leibniz, New Essays on Human Understanding (1704).

As Leibniz pointed out, a hierarchical organization of entities or notions is very helpful for humans to organize their knowledge about the world. Therefore, it is not surprising that the large-scale datasets are stored in hierarchical classifications. Well-known hierarchical classifications include, e.g., Reuters Corpus Volume 1 (RCV1)[1], Enron email corpus (A New Dataset for Email Classification)[2], Directory-style Yahoo![3] and Dmoz catalogue[4]. Such classifications allow us to focus on a smaller and smaller subpart of the information

[1] http://jmlr.csail.mit.edu/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm
[2] http://www.cs.cmu.edu/ enron/
[3] http://dir.yahoo.com/
[4] http://www.dmoz.org/

in the whole hierarchy by ignoring their generalization. This is actually the nature and effective way for humans to manage their knowledge.

My goal in this dissertation is to contribute to research on the automatic categorization of documents to conceptual categories that are organized in a hierarchical classification. In particular I explore automatic categorization by using category dependencies in the hierarchical structure.

In the next, I will further introduce the hierarchical data from IR and semantic web perspectives by placing emphasis on the automatic classification.

1.1.1 Information Retrieval Perspective

In this section, we look at hierarchical classification from an information retrieval point of view. More specifically, we consider a scenario in which a user classifies a document into a topic hierarchy. This scenario gives rise to the problem of predicting one or more topics (multi-class or multi-label problems), that is, search for relevant pieces of topic information. The degrees of relevancy can be computed by most IR systems as a numeric score on how well each topic in the hierarchy matches the document. The categories then will be ranked according to this value and the top ranking categories are then shown to the user.

To get the categories ranking, one method is to compute the similarity (or relevancy) between each category and the document, such as vector space model. According to similarity ranking, we select the top k as the results. This method assumes that all categories are independent from each other, and it tends to suffer from low accuracy when the data (number of categories or documents) becomes huge.

On the other side, hierarchical structures can be used to efficiently search the needed documents starting from the root node in a top-down manner, i.e., they search in different branches of the hierarchy, level by level for finding a few most specific subcategories, in which the interesting documents are located.

It should be noted that the categorization schemes in IR usually assume that node-fathers include all the documents of their child-categories. Additionally, another scheme assumes that children can include documents not-belonging to their fathers.

In summary, text classification in IR is a tool for converting unstructured text collections into structured ones, in which storage and search get easier.

1.1.2 Semantic Web Perspective

Hierarchical taxonomy in semantic web refers to the study of the nature and relation between nodes. It defines an information structure for communicating knowledge. Com-

pared to the taxonomy in IR, it does not only organize categories in a hierarchy, but also provides exact semantics for these categories. The semantics is represented as specialization or generalization between categories by defining is-a or part-of relations. Semantics in text can be extracted under different forms, ranging from metadata to shallower information such as named entities, relationships between them, events and so on.

Since nodes may represent properties a father node may not contain objects, i.e., satisfy properties of some of its children. We call this kind of nodes child-free. The left hierarchy of Figure 1.1 is a standard IR hierarchy composed by child-full nodes. This hierarchy can be converted in child-free hierarchy on the right by adding two more nodes, A' and C', where $A' = (A \cap B) \cup (A \cap C)$ and $C' = (C \cap D) \cup (C \cap E)$.

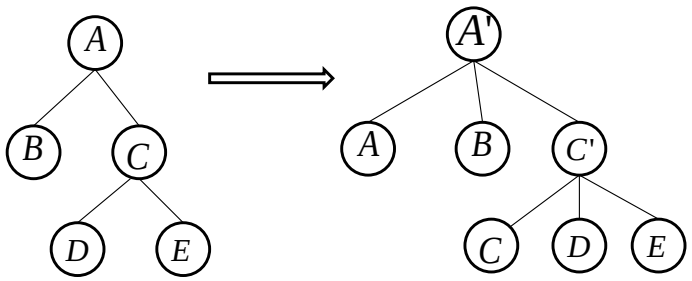

Figure 1.1: Semantics of child-free and child-full

Indeed, hierarchies not only can be used to organize information but at the same time provides semantic dependencies among their nodes. Although, the above fact is well known, there is a substantial lack of practical approaches for exploiting such information in automatic text categorization (TC). The major problem concerns with defining an effective semantic representation for learning algorithm with the exponential number of category relationships.

1.2 Machine Learning for Automated TC

Text categorization is the task of assigning a set of documents into a fixed number of predefined categories, and each document can be associated with multiple (multi-label case), exactly one (multi-class case), or no category at all.

Original classification methodologies relate to manually organizing objects into classification categories following a predefined system of rules. Instead of manually classifying documents or predefined automatic classification rules, statistical text categorization uses machine learning methods to learn such rules based on human-labeled training documents.

Next we will formalize the text categorization problem and describe some basic methods

that are commonly used for text categorization, including the hierarchical text categorization.

1.2.1 A Definition of Text Categorization

The classification problem can be modeled as follows. Given:

- $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{|\mathcal{X}|}\}$ denote the domain of instances, where $\forall i, \boldsymbol{x}_i \in R^n$, is the n-dimensional vector for the ith example.
- $\mathcal{C} = \{c_1, c_2, ..., c_{|\mathcal{C}|}\}$ be the finite set of predefined class labels,
- A set of training data based on $\mathcal{X}$ and $\mathcal{C}$: $(\boldsymbol{x}_1, C_1), (\boldsymbol{x}_2, C_2), ..., (\boldsymbol{x}_{|\mathcal{X}|}, C_{|\mathcal{X}|})$, where $\forall i, C_i \subseteq \mathcal{C}$, is the class set manually assigned to the ith example.

Our goal is to learn a model a classification hypothesis $\mathbb{H}$ such that $\mathbb{H}(\boldsymbol{x}_i) = \{C_i\}, 1 \leq i \leq |\mathcal{X}|$ for new unseen examples.

Some extensions of real-world text classification include:

- Binary classification: the task of classifying the instances into two classes (i.e., $|\mathcal{C}| = 2$), and one or the other is assigned for them (i.e., $|C_i| = 1$).
- Multi-class classification: the problem of classifying instances into more than two classes (i.e., $|\mathcal{C}| > 2$), and only the best one for each instance is returned as positive class (i.e., $|C_i| = 1$).
- Multi-label classification: the problem of classifying instances into more than two classes (i.e., $|\mathcal{C}| > 2$), and a set of classes for each instance are returned as positive classes (i.e., $C_i \subseteq \mathcal{C}$).

1.2.2 Methods of Text Categorization

In this section we described four well-known text categorization methods: Rocchio, k-Nearest Neighbor, Decision Trees, Naïve Bayes and support vector machine. All of these methods were published with relatively strong performance in previous evaluations.

- *Rocchio*, is based on a method of relevance feedback defined in IR systems. It can be divided into three steps: (i) use tf*idf weighted vectors to represent text documents; (ii) compute a centroid vector for each category by summing the vectors of the training documents in this category and; (iii) assign test documents to the category with the closest centroid vector based on cosine similarity.

- *Decision Trees.* Given a set of instances, each of them consists of a set of attributes and corresponding class label. The purpose is to build an accurate model for each class based on the set of attributes and then use the model to classify new unseen instances.

- Naïve Bayes: is a simple probabilistic classifier based on applying Bayes' theorem[5] with strong (Naïve) independence assumptions. Such assumption makes the computation of Naïve Bayes classifier far more efficient than the exponential complexity of the pure Bayes approach (i.e. the former does not use word combinations).

- k-Nearest Neighbor: k-NN classifies instances by (i) measuring their similarity to instances in the training data; and (ii) selecting the class(es) which the nearest documents belong to.

- Support vector machines (SVMs): refer to Chapter 2 for details.

1.2.3 Methods of Hierarchical Text Categorization

There are generally two directions of hierarchical classification approaches: the big-bang and the top-down methods. We will illustrate both of them in the following.

- *Big-bang approaches*: a single global (relatively complex) classification model is built from the training set, taking into account the class hierarchy as a whole during a single run of the classification algorithm. In the test phase, each test example is classified by the induced model: this assigns classes at any level of the hierarchy to the text example.

  One advantage of the big-bang approaches is that the size of the global model is typically considerably smaller than the total size of all the local models learned by the top-down approaches. In addition, semantic dependencies between different classes with respect to class membership (e.g. any example belonging to some nodes automatically belongs to the parent nodes) can be taken into account in a natural, straightforward way.

  Advantages sometimes can be also disadvantages for the big-bang methods for a large-scale real-world dataset:

  (i) They suffer from large computational complexity in the training phase, e.g., [71] and [133] have proved that it is infeasible to directly build a classifier for a large-scale hierarchy.
  (ii) The constructed classifiers may not be flexible enough to account for changes

[5] http://en.wikipedia.org/wiki/Bayes%27_theorem

of the category structure. The classifiers need to be retrained once the category structure is changed, which cost a lot of time.

In general, global classifiers in big-bang methods have two related broad characteristics. They consider the entire class hierarchy at once (as mentioned earlier) and they lack the kind of modularity for local training of the classifiers that is a core characteristic of the top-down approaches.

- *Top-down approaches* constitute another direction of hierarchical classification. They use the hierarchical structure to decompose the entire problem into a set of smaller sub-problems, which are constituted by one internal category and all its direct child nodes. This allows for high efficiency in both learning and prediction since each time a much smaller problem with corresponding feature set is addressed.

  The so-called pachinko-machine model [4] defines a subtree classifier for each node of the hierarchy, and one more local classifier built for each internal node suggested by [109]. If a document is assigned to the internal nodes, the multi-classifiers of their children are recursively activated. This way, the decisions are made from the root until the leaf nodes.

  It is worth noting that subtree classifier of a node decides whether an instance belongs to the subtree rooted at this node, once it is, the local classifier of this node decides if an instance belongs to this node itself rather than its descendants. Only if the instance were accepted by the subtree classifier, the local classifier will have the chance to further classify this instance.

  A typical problem of the top-down approaches in hierarchical classification is error propagation: if an upper classifier made a wrong decision for the document, then it would choose a wrong path to traverse the taxonomy and would no longer have a chance to find the correct category. The errors are unrecovered.

1.3 Accurate and Efficient Models for HTC

Besides the two hierarchical methods applied in HTC, the flat models (i.e., flatten the hierarchy into independent categories, each associated with a binary classifier) are also commonly used in HTC. They face the same problem as big-bang approaches: large computational complexity. Moreover, they do not consider the category dependencies existing in the hierarchy. In a word, the typical methods in HTC including flat models, top-down and big-bang approaches suffer either from low accuracy or low efficiency. In this section, we simply describe our reranking systems for improving both.

1.3.1 Reranking to Boost Accuracy

We design global rerankers (RR) to exploit structural dependencies in hierarchical multi-label text classification (TC). They are based on two algorithms:

- The first generates the k-best classification hypotheses according to (i) the classification probabilities of the flat binary classifiers associated with each node of the hierarchy and (ii) the classification probabilities of binary classifiers working in top-down way exploring the node structure of the hierarchy.
- The second encodes dependencies in the reranker by (i) modeling hypotheses as trees derived by the hierarchy itself and (ii) applying tree kernels to them.

We refer to the global reranker based on the former algorithm as flat reranker (FRR) and the latter as hierarchical reranker (HRR), as the probabilities for generating the hypotheses are from the flat models and the top-down methods.

Our extensive comparison with previous work on Reuters Corpus Volume 1 shows that our rerankers constantly and significantly outperform the state of the art in TC and can inject much more effective structural dependencies in multi-classifiers.

1.3.2 LIR to Boost Efficiency

The top-down methods are efficient since they decompose the entire problem into many sub-problems and deal with them in top-down way. Their main drawback is the error propagation from the higher to the lower nodes. To address this issue we propose an efficient incremental reranking model to improve classifier decisions of high-level nodes.

We build a reranking model for each of such sub-problems (we call it local reranker) as described in Section 1.3.1, which is used to rerank the generated classification hypotheses based on such sub-problem to select the best one. Our local rerankers exploit category dependencies of the sub-problems, which allow them to recover from the multi-classifier errors whereas their application in top-down fashion results in high efficiency.

The experimentation on Reuters Corpus Volume 1(RCV1) shows that our incremental reranking is as accurate as global rerankers but at least one order of magnitude faster.

1.3.3 Fast LIR for Large-scale HTC

LIR in Section 1.3.2 based on the top-down methods improve the accuracy of basic models while preserving the efficiency of the reranker. The reranker is based on the top-down methods, in which one or two basic multi-classifiers are built for each category. This suffers from low efficiency problem when the number and size of categories in the hierarchy become huge. To solve this problem, we suggested some optimizations of basic models

such as the use of LIBLINEAR model instead of LIBSVM, local dictionaries construction. In addition, we adopt a compact representation of the hypotheses used in learning RR and LIR to further improve the efficiency of LIR. We called this optimization of LIR fastLIR.

The experimentation of fastLIR on DMOZ is shown its advantages in large-scale hierarchical text categorization.

1.4 Thesis Outline

This thesis aims to study the dependencies between categories in the hierarchy for text categorization. The main contributions can be analyzed by measuring: (a) the improvement in accuracy that global reranker over the basic models in HTC by considering the dependencies and (b) the efficiency of our rerankers on large-scale hierarchical datasets. The thesis is organized as follows:

- Chapter 2 describes support vector machines in details including the optimal separating hyperplane, soft margin hyperplane, and kernel methods for solving the nonlinear separable problems, finally we enumerate several specific structural kernels and several popular pieces of SVM software.

- Chapter 3 first introduces the corpora the basic document pre-processing techniques. Then it explains the commonly used flat models and top-down models in HTC, applying such models on an Italian dataset and RCV1 reports the evaluations of the basic SVM models. Most importantly, we develop a piece of software: proLIBLINEAR based on LIBLINEAR and the Platt's sigmoid function, which matches the state-of-the-art one-vs.-all SVMs on RCV1 while preserving high efficiency.

- Chapter 4 designs advanced global reranking model to exploit structural dependencies in hierarchical multi-label text classification (TC). They are based on two algorithms: (1) generates the k-best classification hypotheses; and (2) encodes dependencies in the reranker by: (i) modeling hypotheses as trees derived by the hierarchy itself and (ii) applying tree kernels to them.

- Chapter 5 proposes an efficient local incremental reranking model: the combination of top-down methods with a reranking algorithm based on SVMs. The use of structural hierarchy features from tree kernel spaces in the latter allowed us to outperform state-of-the-art accuracy on the entire RCV1 while the use of the former ensured efficiency. Furthermore, we optimize the top-down methods to further improve the efficiency so that it can be applied in large-scale hierarchical text categorization.

Finally, the conclusions can be found in Chapter 6.

Chapter 2

Support Vector Machines

In this chapter we described support vector machines from the following perspectives:

- Principle: support vector machines (SVMs) are training algorithms for learning classification and regression ranking models from data. The model is the optimal decision boundary of two set of training examples in a vector space. This algorithm is motivated by statistical learning theory [85]. Its fundamental idea is very simple: locating the boundary to make the nearest datapoints from two sets have the largest margin.

- History: the original SVM algorithm (hard margin) was invented by Vladimir N. Vapnik and the current standard incarnation (soft margin) was proposed by Vapnik and Corinna Cortes in 1995 [26]. The simple idea is to find the optimal separating hyperplane which linearly divides one set from the other, which is commonly applied to a number of applications, ranging from face identification [25], particle identification [4, 21], and text categorization to engine knock detection [96], biological data processing for medical diagnosis [119], and database marketing [7].

- Extension of kernel methods: SVMs have attracted more and more attention because of the introduction of kernel methods, which transform the original vector space into much higher dimensional vector space so that locate a nonlinear boundary for the two sets. The transformation of vector spaces is dependent on the *kernel function*, which determines the relationships between the original vector space and transformed space.

Section 2.1 decribes the simple linear SVM based on the optimal separating hyperplane (i.e., Section 2.1.1) and its extension with soft margin that can tolerate some errors but pays penalties for them (i.e., Section 2.1.2). We introduce kernel methods for solving the nonlinear separatable problems in Section 2.2, and explain several specific stuctural kernels in Section 2.2.2. Section 2.3 lists several popular pieces of SVM software.

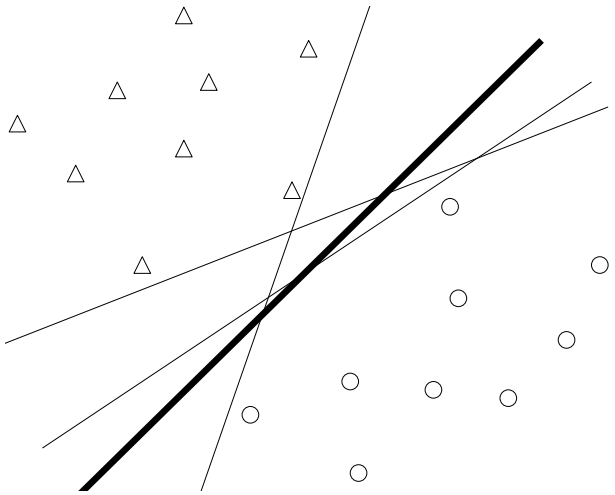

Figure 2.1: Optimal separating hyperplane

2.1 Optimal Separating Hyperplane in Linear SVM

Consider the example in Figure 2.1, it is a binary classification task with two sets of datapoints: triangles and circles. We can easily know from the figure that there are many possible linear classifiers that can completely separate one from the other, but there is only one which maximizes the distance (or margin) of the nearest data points from the two classes. This linear classifier is called the optimal separating hyperplane as specified in bold.

2.1.1 Hard Margin SVM

Suppose we are given a set of training points:

$$\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_l, y_l)\},\ \boldsymbol{x} \in R^n, y \in \{+1, -1\}$$

where $\boldsymbol{x}_i$ denotes the $n-$dimensional *real* vector instance, and y its $+1$ or -1 label. As shown from Figure 2.1, there are many boundary hyperplanes that can separate the two classes and one of them can be expressed as:

$$\mathbf{H}_0 : \quad \boldsymbol{w}^T \cdot \boldsymbol{x} + b = 0 \tag{2.1}$$

where:

- $\boldsymbol{w}$ is the normal vector to the hyperplane,
- b is a bias term,
- $\cdot$ is the doc product of the two vectors $\boldsymbol{w}$ and $\boldsymbol{x}$,

- $\frac{b}{\|\boldsymbol{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\boldsymbol{w}$.

The distance for a training vector $\boldsymbol{x}_i$ to this boundary, called *margin*, is expressed as follows:

$$margin = \frac{|\boldsymbol{w}^T \cdot \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|} \tag{2.2}$$

since $\boldsymbol{w}^T$ in Equation 2.1 is multiplied by training vectors (known as common constants) and b is also a constant term, we can impose a restriction on equation (2.1) as follows:

$$|\boldsymbol{w}^T \cdot \boldsymbol{x}_i + b| \geq 1, \forall \boldsymbol{x}_i \tag{2.3}$$

From equation (2.3) we can select two hyperplanes satisfying either:

$$\mathbf{H}_1 : \quad \boldsymbol{w}^T \cdot \boldsymbol{x} + b \geq 1, \quad y_i = +1 \tag{2.4}$$

or

$$\mathbf{H}_{-1} : \quad \boldsymbol{w}^T \cdot \boldsymbol{x} + b \leq -1, \quad y_i = -1 \tag{2.5}$$

in a way that they separate the data and there are no points between them. According to Equation 2.2, the *minimal distance* between these two hyperplanes is:

$$\frac{2}{\|\boldsymbol{w}\|} \tag{2.6}$$

Thus, we can maximize this distance. To maximize the equation (2.6), it is preferable to minimize:

$$\frac{1}{2}\|\boldsymbol{w}\|^2 \tag{2.7}$$

Consequently, the optimization problem finding the optimal separating hyperplane is formalized as:

$$\begin{aligned} &minimize \quad \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &subject\ to \quad y_i(\boldsymbol{w}^T \cdot \boldsymbol{x}_i + b) \geq 1, (i = 1, ..., l) \end{aligned} \tag{2.8}$$

where y_i equals 1 if $\boldsymbol{x}_i$ belongs to the *positive* example set, and -1 if $\boldsymbol{x}_i$ belongs to the *negative* set.

Equation 2.8 is a typical optimization problem subject to Karush-Kuhn-Tucker (KTT) conditions [62]. The Lagrange's method is based on the *Lagrangian function*:

$$\begin{aligned} \mathbf{L}(\boldsymbol{w}, b, \alpha_i) &= \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{l} \alpha_i [y_i(\boldsymbol{w}^T \cdot \boldsymbol{x}_i + b) - 1] \\ &= \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w}^T - \sum_{i=1}^{l} \alpha_i [y_i(\boldsymbol{w}^T \cdot \boldsymbol{x}_i + b) - 1] \end{aligned} \tag{2.9}$$

where α_i are KKT multipliers or Lagrange Multipliers and $\alpha_i \geq 0$.

To solve the Wolfe's problem [108] we compute the derivatives with respect to $\boldsymbol{w}$ and b, and we impose them to 0 to find minimal points.

$$\begin{aligned} \frac{\partial \mathbf{L}}{\partial \boldsymbol{w}} &= w - \sum_{i=1}^{l} \alpha_i y_i \boldsymbol{x}_i \\ \frac{\partial \mathbf{L}}{\partial b} &= -\sum_{i=1}^{l} \alpha_i y_i \end{aligned} \tag{2.10}$$

must be zero, which means that from Equation 2.10 we have:

$$\begin{aligned} \boldsymbol{w} &= \sum_{i=1}^{l} \alpha_i y_i \boldsymbol{x}_i \\ \sum_{i=1}^{l} \alpha_i y_i &= 0 \end{aligned} \tag{2.11}$$

Substituting Equation 3.5 to Equation 2.9, we get

$$\begin{aligned} \mathbf{L}(\boldsymbol{w}, b, \alpha_i) &= \frac{1}{2}(\sum_{i=1}^{l} \alpha_i y_i)(\sum_{j=1}^{l} \alpha_j y_j)^T - \sum_{i=1}^{l} \alpha_i [y_i((\sum_{j=1}^{l} \alpha_j y_j \boldsymbol{x}_j)^T \cdot \boldsymbol{x}_i + b) - 1] \\ &= -\frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_i)^T \cdot \boldsymbol{x}_j + \sum_{i=1}^{l} \alpha_i \end{aligned} \tag{2.12}$$

and we define the *Lagrange dual function*

$$\boldsymbol{w}(\alpha) = -\frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_i)^T \cdot \boldsymbol{x}_j + \sum_{i=1}^{l} \alpha_i \tag{2.13}$$

$\boldsymbol{w}(\alpha)$ can be regarded as the minimal value of $\mathbf{L}(\boldsymbol{w}, b, \alpha_i)$ based on optimal $\boldsymbol{w}$. Obtaining this minimal value is equivalent to maximize $\boldsymbol{w}(\alpha)$ based on α, which is reduced to a quadratic programming problem as follows:

$$\begin{aligned} \textit{maximize} \quad & -\frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_i)^T \cdot \boldsymbol{x}_j + \sum_{i=1}^{l} \alpha_i \\ \textit{subject to} \quad & \sum_{i=1}^{l} \alpha_i y_i = 0, \alpha_i \geq 0 \end{aligned} \tag{2.14}$$

Many algorithms for solving this quadratic programming problem are available: such as augmented Lagrangian [36], active set [83], interior point [82], Quasi-Newton methods [101]. Particularly, the Sequential minimal optimization (SMO) [86] suggested by John C. Platt for support vector machines is regarded as the fastest quadratic programming method.

2.1.2 Soft Margin SVM

The above discussion on hard margin linear SVMs is applicable to the case of two completely separatable sets only. If there is a small number of noisy data making the two sets nonlinearly separatable, a linear boundary that separates them will not be existing as explained in Section 2.1.1. Consider Figure 2.2, the triangles (in positive class +) and cirles (in negative class -). The red points violate the rule defined in Equation 2.4 and Equation 2.5.

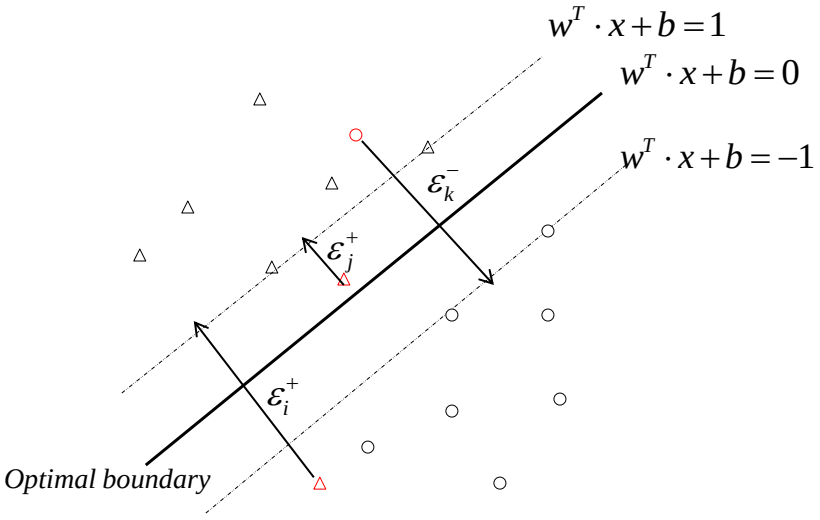

Figure 2.2: Slack margin for SVM

In order to extend the hard-margin SVM to handle data that is not fully linearly separable, we relax the constraints for Equation 2.4 and Equation 2.5 to indicate tolerances of those misclassified datapoints. This is done by introducing *positive slack variable* ξ_i:

$$\mathbf{H}_1 : \quad \boldsymbol{w}^T \cdot \boldsymbol{x}_i + b \geq 1 - \xi_i, \quad y_i = +1 \tag{2.15}$$

or

$$\mathbf{H}_{-1} : \quad \boldsymbol{w}^T \cdot \boldsymbol{x}_i + b \leq -1 + \xi_i, \quad y_i = -1 \tag{2.16}$$

where $i = 1,...,l$, l is the number of training datapoints, which can be integrated into:

$$y_i(\boldsymbol{w}^T \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i (i = 1, ..., l) \tag{2.17}$$

Thus, some training vectors are allowed to be located in a limited region in the wrong sides w.r.t. the hyperplane, as shown in Figure 2.2. One positive triangle point and one negative circle point are on the incorrect side of the margin boundary, and we assign the

slack variables ξ_i^+ and ξ_k^-, respectively. Several optimization boundaries are proposed for this case, for instance

$$\begin{aligned} &minimize \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}\xi_i \\ &subject\ to \quad y_i(\boldsymbol{w}^T \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, (i = 1, ..., l) \\ &\qquad\qquad \forall i, \xi_i \geq 0 \end{aligned} \tag{2.18}$$

We can see that the second term in Equation 2.18 is a penalty term for the misclassified datapoints, and the constant C is the weight of misclassifications.

It should be noted that we can use a simpler approach followed in Section 2.1.1 to obtain the optimal value of Equation 2.18.

2.2 Kernel Methods

The soft margin method is an extension of linear hard-margin SVM, which allows for including some misclassifications. *Kernel methods* are another approach for finding non-linear boundaries that fully separate the two sets of datapoints.

The fundamental idea of kernel methods is to transform the original vector space into a higher dimensional space. We consider the linearly non-separable examples shown on the left of Figure 2.3, which becomes fully linearly separatable on the right after space transformation by a mapping

$$\Phi : \boldsymbol{x}_i \rightarrow \Phi(\boldsymbol{x}_j).$$

so the Lagrange dual function in Equation 2.13 becomes:

$$\boldsymbol{w}(\alpha) = -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j (\Phi(\boldsymbol{x}_i))^T \cdot \Phi(\boldsymbol{x}_j) + \sum_{i=1}^{l}\alpha_i \tag{2.19}$$

The inner product of $\Phi(\boldsymbol{x}_i)^T \cdot \Phi(\boldsymbol{x}_j)$ in such high dimensional feature spaces solves the problem of expressing complex nonlinear functions, but it also results in the computational complexity problem because of the implicit computation of $\boldsymbol{w}$. However given a mapping Φ and two datapoints $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, the inner product of the transformed points in higher feature space can be evaluated by using the kernel function without even explicitly knowing the mapping, e.g.,

$$\Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_i) \equiv \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.20}$$

With a suitable choice of kernel $\mathcal{K}$ as above the data can become separable in feature space despite being non-separable in the original input space. The kernel trick here lies in working in a high dimensional space, without even explicitly transforming the original

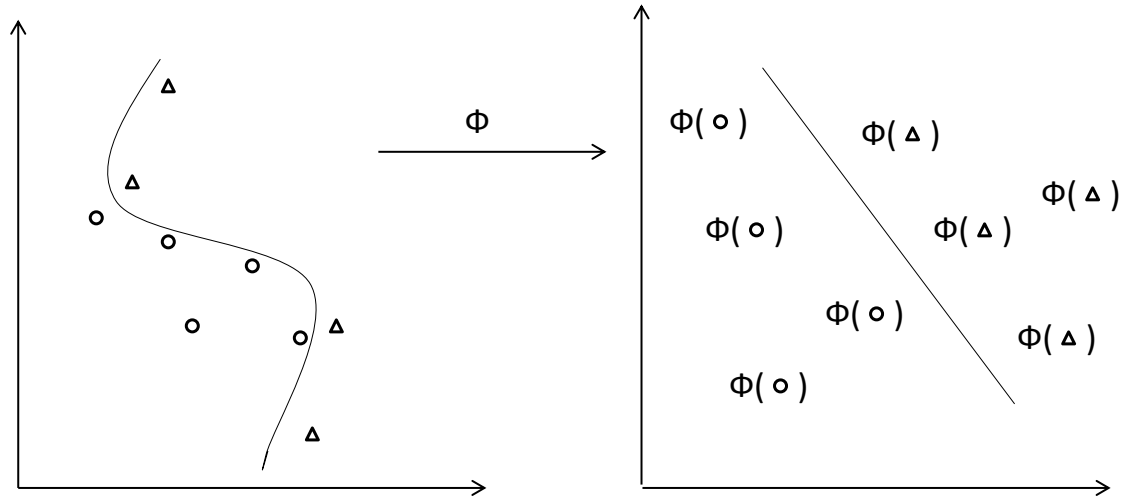

Figure 2.3: Vector space transformation

data points into that space. Instead we rely on algorithms that only need to compute inner products within that space, which are identical to $\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and can thus be cheaply computed in the original space using only multiplications. This means that the inner product in the richer feature space will not cause the computational problem.

Based on Equation 2.20 and Equation 2.19, the learning task for the binary classification problem with a given choice of kernel therefore involves maximization of the Lagrange dual function:

$$\boldsymbol{w}(\alpha) = -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) + \sum_{i=1}^{l}\alpha_i \tag{2.21}$$

A sufficient condition for satisfying the Equation 2.20 is that the kernel $\mathcal{K}$ is semi-positive definite. Several examples of such kernel functions are commonly used as follows:

- *RBF kernels*: $\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-(\boldsymbol{x}_i - \boldsymbol{x}_j)^2/2\sigma}$
- *Gaussian kernels*: $\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{\sigma^2})$
- *polynomial kernels*: $\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i \cdot \boldsymbol{x}_j + 1)^p$
- *tanh kernel*: $\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = tanh(\beta \boldsymbol{x}_i \cdot \boldsymbol{x}_j + b)$.

Suitable kernels must satisfy a mathematical condition called Mercers theorem [74], for example, the tanh kernel above only satisfies Mercers' conditions for certain values of β and b.

2.2.1 String Kernels

The String Kernels (SK) that we consider count the number of subsequences shared by two strings of symbols, s_1 and s_2. Some symbols during the matching process can be skipped. This modifies the weight associated with the target substrings as shown by the following SK equation:

$$\begin{aligned} SK(s_1, s_2) &= \sum_{u \in \Sigma^*} \phi_u(s_1) \cdot \phi_u(s_2) \\ &= \sum_{u \in \Sigma^*} \sum_{\vec{I}_1 : u = s_1[\vec{I}_1]} \sum_{\vec{I}_2 : u = s_2[\vec{I}_2]} \lambda^{d(\vec{I}_1)+d(\vec{I}_2)} \end{aligned} \tag{2.22}$$

where, $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$ is the set of all strings, $\vec{I}_1$ and $\vec{I}_2$ are two sequences of indexes $\vec{I} = (i_1, ..., i_{|u|})$, with $1 \leq i_1 < ... < i_{|u|} \leq |s|$, such that $u = s_{i_1}..s_{i_{|u|}}$, $d(\vec{I}) = i_{|u|} - i_1 + 1$ (distance between the first and last character) and $\lambda \in [0, 1]$ is a decay factor.

It is worth noting that: (a) longer subsequences receive lower weights; (b) some characters can be omitted, i.e. gaps; (c) gaps determine a weight since the exponent of λ is the number of characters and gaps between the first and last character; and (c) the complexity of the SK computation is $O(mnp)$ [105], where m and n are the lengths of the two strings, respectively and p is the length of the largest subsequence we want to consider.

In our case, given a hypothesis represented as a tree like in Figure 4.15, we can visit it and derive a linearization of the tree. SK applied to such a node sequence can derive useful dependencies between category nodes. For example, using the Breadth First Search on the compact representation, we get the sequence `[MCAT, M11, M13, M14, M143]`, which generates the subsequences, `[MCAT, M11]`, `[MCAT, M11, M13, M14]`, `[M11, M13, M143]`, `[M11, M13, M143]` and so on.

2.2.2 Structural Kernels

In kernel-based machines, both learning and classification algorithms only depend on the inner product between instances. In several cases, this can be efficiently and implicitly computed by kernel functions by exploiting the following dual formulation:

$$\sum_{i=1..l} y_i \alpha_i \phi(\boldsymbol{o}_i)\phi(\boldsymbol{o}) + b = 0 \tag{2.23}$$

where $\boldsymbol{o}_i$ and $\boldsymbol{o}$ are two objects, ϕ is a mapping from the objects to feature vectors $\boldsymbol{x}_i$ and

$$\phi(\boldsymbol{o}_i)\phi(\boldsymbol{o}) = \mathcal{K}(\boldsymbol{o}_i, \boldsymbol{o}) \tag{2.24}$$

is a kernel function implicitly defining such a mapping. In case of structural kernels, $\mathcal{K}$ determines the shape of the substructures describing the objects above. The most general kind of kernels used in NLP are string kernels, e.g. [105], the Syntactic Tree Kernels [24] and the Partial Tree Kernels [78].

The vast majority of tasks in natural language processing involve the processing of *structured objects*. Building classifiers for these objects is traditionally carried out by implementing rule-based extractors of features. However, the complexity of the structure prevents an exhaustive approach to feature generation since the use of all possible substructures produces an exponential number of features, and consequently the development of such systems is typically guided by heuristics rather than a systematic approach. For instance, [20] commented on the development of features for a parse tree reranker: "It is worth noting that developing feature schemata is much more an art than a science."

As a way to avoid the feature selection problem, learning methods that work directly with objects instead of feature vectors have been proposed. The generalization from linear classifiers (that apply to vectors) to *kernel-based classifiers* (that apply to objects) is straightforward. To derive the kernel-based decision function, we start from the decision function of a linear classifier:

$$f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i \cdot \boldsymbol{x} + b \tag{2.25}$$

where $\boldsymbol{x}$ is a classifying example and $\boldsymbol{w}$ and b are the separating hyperplane's *gradient* and its *bias*, respectively. The gradient is a linear combination of the training points $\boldsymbol{x}_i$, their labels y_i and their weights α_i. Applying the so-called *kernel trick* it is possible to replace the scalar product with a *kernel function* defined over pairs of *objects*:

$$f(o) = \sum_{i=1}^{n} \alpha_i y_i \mathcal{K}(\boldsymbol{o}_i, \boldsymbol{o}) + b \tag{2.26}$$

with the advantage that we do not need to provide an explicit mapping $\phi(\cdot)$ of our examples in a vector space; instead, the scalar product can be computed implicitly, which may be much more efficient. It is also easy to show that for kernels $\mathcal{K}_1$ and $\mathcal{K}_2$, we may form new kernels $\mathcal{K}_1 + \mathcal{K}_2$ and $\mathcal{K}_1 \cdot \mathcal{K}_2$, allowing for a modular decomposition. Kernel functions have proven very effective for natural language applications as suggested by the large body of related work, e.g. [24, 60, 30, 15, 29, 32, 113, 61, 111, 81, 37].

In the case where the objects we want to classify are trees, there exist efficient algorithms based on dynamic programming that compute kernel functions based on counting the shared substructures of the trees, i.e., *tree kernels*. These computations are efficient since they do not have to enumerate the whole fragment space explicitly.

Let $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$ be the set of tree fragments and $\chi_i(n)$ is an indicator function, which equals 1 if the target f_i is rooted at node n and equals 0 otherwise. A tree kernel function over T_1 and T_2 is defined as

$$TK(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) \tag{2.27}$$

where N_{T_1} and N_{T_2} are the sets of nodes in T_1 and T_2, respectively, and

$$\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} \chi_i(n_1)\chi_i(n_2) \tag{2.28}$$

The Δ function is equal to the number of common fragments rooted in nodes n_1 and n_2 and thus depends on the fragment type. Below, we report the algorithm to compute Δ for the syntactic tree kernels (STK) [24] and partial tree kernel (PTK) [78].

Syntactic Tree Kernel

A syntactic tree fragment (STF) is a set of nodes and edges from the original tree such that the fragment is still a tree, with the further constraint that any node must be expanded with either all or none of its children[1].

To compute the number of common STFs rooted in n_1 and n_2, the Syntactic Tree Kernel (STK) uses the following Δ function [24]:

1. if n_1 and n_2 or their children are different then

$$\Delta(n_1, n_2) = 0 \tag{2.29}$$

2. if n_1 and n_2 and their children are the same, and n_1 and n_2 have only leaf children (i.e. they are pre-terminal symbols) then

$$\Delta(n_1, n_2) = \lambda \tag{2.30}$$

3. if n_1 and n_2 or their children are the same, and n_1 and n_2 are not pre-terminals then

$$\Delta(n_1, n_2) = \lambda \prod_{j=1}^{l(n_1)} (1 + \Delta(c_{n_1}(j), c_{n_2}(j))) \tag{2.31}$$

where $l(n_1)$ is the number of children of n_1, $c_n(j)$ is the j-th child of node n and λ is a decay factor penalizing larger structures.

Figure 2.5 shows the five fragments of the hypothesis in Figure 2.4. Such fragments satisfy the constraint that each of their nodes includes all or none of its children. For

[1]STK has been originally defined in the context of parse-tree reranking. Accordingly, the property above is equivalent to state that the grammar production rules associated with an STF cannot be a partial rule.

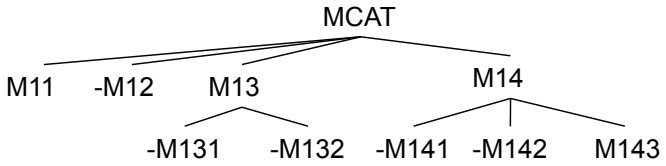

Figure 2.4: A tree representing a category assignment hypothesis for the subhierarchy MCAT of RCV1

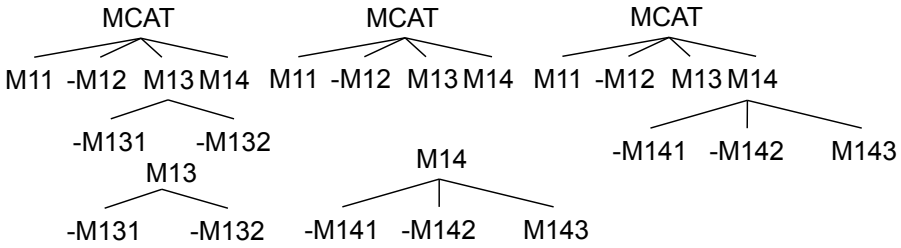

Figure 2.5: The tree fragments of the hypothesis in Figure 2.4 generated by STK

example, [M13 [-M131 -M132]] is an STF, which has two non-terminal symbols, -M131 and -M132, as leaves while [M13 [-M131]] is not an STF. The computational complexity of STK is $O(|N_{T_1}||N_{T_2}|)$, although it is shown in [78] that the average running time is linear in the number of tree nodes.

Partial Tree Kernel

The Δ function for PTK is as follows. Given two nodes n_1 and n_2, STK is applied to all possible child subsequences of the two nodes. For this purpose a string kernel [104] is used to asparaenum all child-subsequences whereas a tree kernel is applied to each subsequence. More formally:

1. if the node labels of n_1 and n_2 are different then

$$\Delta(n_1, n_2) = 0 \tag{2.32}$$

2. else

$$\Delta(n_1, n_2) = 1 + \sum_{\vec{I}_1, \vec{I}_2, l(\vec{I}_1)=l(\vec{I}_2)} \prod_{j=1}^{l(\vec{I}_1)} \Delta(c_{n_1}(\vec{I}_{1j}), c_{n_2}(\vec{I}_{2j})) \tag{2.33}$$

where $\vec{I}_1 = \langle h_1, h_2, h_3, .. \rangle$ and $\vec{I}_2 = \langle k_1, k_2, k_3, .. \rangle$ are index sequences associated with the ordered child sequences c_{n_1} of n_1 and c_{n_2} of n_2, respectively, $\vec{I}_{1j}$ and $\vec{I}_{2j}$ point to the j-th child in the corresponding sequence, and $l(\cdot)$ returns the sequence length, i.e. the number of children. Additionally, we add two decay factors: μ for the depth of the tree and λ for

the length of the child subsequences with respect to the original sequence, i.e. we account for gaps. It follows that:

$$\Delta(n_1, n_2) = \mu\Big(\lambda^2 + \sum_{\vec{I}_1, \vec{I}_2, l(\vec{I}_1)=l(\vec{I}_2)} \lambda^{d(\vec{I}_1)+d(\vec{I}_2)} \prod_{j=1}^{l(\vec{I}_1)} \Delta(c_{n_1}(\vec{I}_{1j}), c_{n_2}(\vec{I}_{2j}))\Big), \tag{2.34}$$

where $d(\vec{I}_1) = \vec{I}_{1l(\vec{I}_1)} - \vec{I}_{11}+1$ and $d(\vec{I}_2) = \vec{I}_{2l(\vec{I}_2)} - \vec{I}_{21}+1$. This way, we penalize both larger trees and child subsequences with gaps. An efficient algorithm for the computation of PTK is given in [78], whose worse-case complexity is $O(\rho^3|N_{T_1}||N_{T_2}|)$, where ρ is the largest branching factor. Again, the authors showed that the average running time for natural language trees is linear (more precisely $O(n^{1.2})$.

Given a target T, PTK can generate any subset of connected nodes of T, whose edges are in T. For example, Fig. 2.6 shows the tree fragments from the hypothesis of Fig. 2.4. Note that each fragment captures dependencies between different categories.

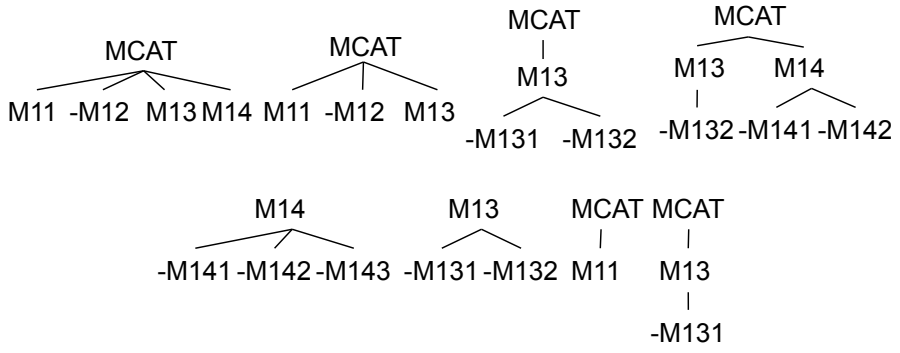

Figure 2.6: Some tree fragments of the hypothesis in Figure 2.4 generated by PTK

2.3 SVM Software

In this section, we introduced the four pieces of SVM software used in this thesis.

2.3.1 LIBSVM

LIBSVM[2] is a library for SVM, and currently becomes one of the most widely used SVM software. It has been downloaded more than 250,000 times since 2000, and successfully applied in many areas such as computer vision, natural language processing, Neuroimaging, and Bioinformatics [19]. Its main features include:

[2]http://www.csie.ntu.edu.tw/ cjlin/libsvm/

1) Integrations of support vector classification (C-SVC, nu-SVC), support vector regression (epsilon-SVR, nu-SVR) and distribution estimation (one-class SVM);

2) Cross validation for model selection;

3) Efficient multi-class classification: one-vs.-one;

4) Probability estimates;

5) Various kernels (including precomputed kernel matrix);

6) Weighted SVM for unbalanced data.

One more advantage for LIBSVM is that it extends SVM to give probability estimates, by implementing the Platt's sigmoid functions [87] into SVM. The sigmoid function takes the decision value as parameter as follows:

$$p_i = \frac{1}{1 + e^{Af_i + B}} \tag{2.35}$$

where f_i is the decision (or target) value of example $\boldsymbol{x}_i$, p_i is the class probability estimated for multi-class classification, and A and B are estimated by minimizing the negative log likelihood of training data (using their labels and decision values).

The fact that SVM+sigmoid yields probabilities provides one more criterion for predicting $\boldsymbol{x}_i$, using the probability to compare with a threshold of 0.5. This sometimes improves the decision-making as we do not need to directly judge the difference between decision value and 0 [87]. This mainly depends on the different datasets. For simplicity, we call this fact *Platt's theory*, which would be used many times in the experimental parts of this thesis.

2.3.2 LIBLINEAR

LIBLINEAR[3] is a linear classifier for data with *millions* of instances and features. Solving such large-scale classification problems is crucial in many applications such as text classification. Linear classification, comparing to classifications based on other kernels listed in Section 2.2, has become one of the most promising learning techniques for large sparse data with a huge number of instances and features. LIBLINEAR is a software that proved to be efficient and effective [41]. It supports L2-regularized logistic regression (LR), L2-loss and L1-loss linear support vector machines (LSVMs)

Given a set of instance-label pairs $(\boldsymbol{x}_i, y_i)$, $i = 1, ..., l$, $\boldsymbol{x}_i \in R^n$, $y_i \in \{-1, +1\}$, both LR and LSVMs solve the following unconstrained optimization with different loss functions $L(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$:

$$min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} L(\boldsymbol{w}; \boldsymbol{x}_i, y_i) \tag{2.36}$$

[3] http://www.csie.ntu.edu.tw/ cjlin/liblinear/

where $\boldsymbol{w}$ and C are the same as defined in Section 2.1. For LSVMs, the two commonly used loss functions L are:

- L1-SVM:
$$L(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = max(1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i, 0) \tag{2.37}$$
- L2-SVM:
$$L(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = max(1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i, 0)^2 \tag{2.38}$$

In some cases, a bias term b is attached at the end of instance $\boldsymbol{x}_i$, which is a constant specified by the users, usually we specify it for 0 or 1. The approach for LSVM is a coordinate descent method [46], and for L2-SVM, LIBLINEAR implements the trust region of the Newton method [51].

Its main features include (2), (3), (6) of LIBSVM in Section 2.3.1 and the following distinctive features:

1) Integrations of L2-regularized classifiers (L2-loss linear SVM, L1-loss linear SVM, and logistic regression), L1-regularized classifiers (L2-loss linear SVM and logistic regression) and L2-regularized support vector regression (L2-loss linear SVR and L1-loss linear SVR);

2) Efficient multi-class classification;

3) Multi-class classification: (1) one-vs.-rest, (2) Crammer & Singer;

4) Probability estimates (logistic regression only).

2.3.3 proLIBLINEAR

Modeling a classifier to produce a posterior probability is very useful for building joint models, where the decomposed small parts of an overall decision, and their probabilities can be combined for the overall decision. One typical example is the use of a Viterbi search or HMM to combine recognition results from phoneme into word recognition [10]. Another example of this combination in this thesis is the local incremental reranking system in Chapter 5, in which each local reranker is built for each decomposed sub-problem based on hierarchical probabilistic hypothesis generation.

We have pointed out that LIBLINEAR does not produce the posterior probability. It just produces the probability using logistic regression. Meanwhile, LIBSVM cannot be properly applied in large-scale dataset as LIBLINEAR, as it is much slower. In particular, LIBLINEAR stores $\boldsymbol{w}$ in the model, but LIBSVM stores all support vectors. So LIBLINEAR does not need to compute the kernel value as LIBSVM. Combining the advantages of LIBSVM and LIBLINEAR is surely necessary for processing big data.

In order to extend LIBLINEAR for producing probability in classification, we developed *proLIBLINEAR* by implementing the probability output mechanism (i.e., the sigmoid

function taking decision value as parameter) of LIBSVM into LIBLINEAR. We observed that the probabilities output by proLIBLINEAR are almost the same as LIBSVM in a 5-fold cross validation.

2.3.4 SVM-LIGHT-TK

SVM-LIGHT-TK[4], created by Alessandro Moschitti, is an extension of SVMlight[5], which encodes tree kernels for processing syntactic parse trees in NLP tasks. Syntactic tree features improve the learning models, for example, they encode hierarchical dependencies in the learning algorithms designed in this thesis for text categorization.

In the NLP area, convolution kernels (see Kernel philosophy[6]) are alternatives to the explicit feature design. They measure similarity between two syntactic trees in terms of their sub-structures [24] by applying the tree kernels described in Section 2.2.2. These approaches have given optimal results [77] when introducing syntactic information in the task of predicate argument classification (PAC), or injecting semantic dependencies among the hierarchy in hierarchical text categorization (HTC).

Their main features include:

1) Fast kernel computation for similarity of syntactic trees [79].

2) Vector sets, multiple feature vectors over multiple feature spaces can be specified in the input. This allows us to use different kernels with different feature subsets.

3) Tree forests, a set of trees over multiple feature spaces can be specified in the input. This allows us to use a set of different structured features, thus limiting the sparseness of the kernels applied to the whole tree.

4) Four types of tree kernels: SubSet Tree kernel (SST) [24, 79], Subtree kernel (ST) [120, 79], string kernel (SK) [72], and partial tree kernel (PTK) [78].

5) Embedded combinations of trees and vectors: (a) sequential summation, the kernels between corresponding pairs of trees and/or vectors in the input sequence are summed together; and (b) all vs. all summation, each tree and vector of the first object are evaluated against each tree and vector of the second object.

In this thesis, we applied structural kernels such as PTK and STK to labeled trees. This way, all possible dependency features are generated, and used by the preference reranking techniques to choose the best category hypothesis.

[4] http://disi.unitn.it/moschitti/Tree-Kernel.htm
[5] http://svmlight.joachims.org/
[6] http://disi.unitn.it/moschitti/Kernel_Group.htm

Chapter 3

Automated Text Categorization

This chapter describes the phases of automated text classification illustrated in Figure 3.1, which includes:

1) Document pre-processing. This takes the training, validation, and test documents as input, and outputs internal representations (e.g., binary real vectors) for them. These techniques are mainly from traditional IR, such as tokenization, stopword removal, stemming, inverted indexing, feature selection and word weighting.

2) Classifier learning. The internal representations of the training and validation sets are the input of learning algorithms. The fundamental classifiers for text categorization (TC) include Naïve Bayes, SVM, k-NN, Rocchio and Decision Trees, which are applied in flat or top-down manner in the classification phase.

3) Classification evaluation. This takes the results of the classification of the test set and output different accuracy measures.

The major contributions of this chapter are:

- A study on Point (2), which explores proLIBSVM and proLIBLINEAR parameterization to achieve the state of the art method on RCV1, measured by precision, recall, Micro- and Macro-Average F1 measures.
- ProLIBLINEAR shows its high efficiency while preserving high accuracy.
- Both of the pieces of software above generate accurate category probabilities, which are further utilized in reranking model construction (see Chapter 4).

Section 3.1 lists the corpora used in this chapter. Section 3.2 describes the document pre-processing steps in details. The construction of flat multi-class and multi-label classifiers is given in Section 3.3. Section 3.4 shows the two typical methods for hierarchical text classification. Section 3.5 defines some traditional evaluation measures and distance-based multi-label graph-induced error (MGIE). The experimental results are reported in Section 3.6. Finally, Section 3.7 derives the conclusions on applying the basic models for

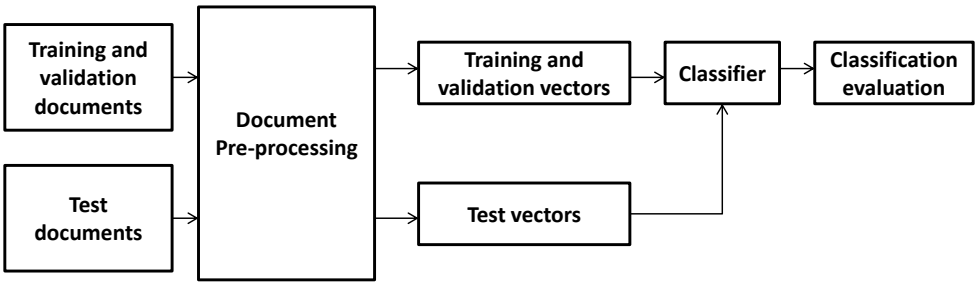

Figure 3.1: Architecture of a text classification system

TC.

3.1 Corpora

In this thesis 3 different collections have been considered and will be described in the following sections.

3.1.1 Reuters Corpus Volume I

Reuters Corpus Volume I (RCV1) is an archive containing 35 times (over 800,000 newswire stories) larger data than the popular Reuters-21578 collections and its variants [68, 33]. Such massive document set has been manually categorized with respect to three domains: *Topics*, *Industries* and *Regions*, which are recently made available by Reuters, Ltd. for research purpose. The first two category sets (i.e., Topics and Industries) are tree-structured hierarchies with 103 and 365 classes respectively, and the Regions contains 366 flatten classes.

Our goal is to improve the accuracy of classifying text on structural hierarchies, so in this thesis we will focus on the Topics and Industries hierarchical category sets. According to its generation process, RCV1 has two versions: the original raw data (we call it RCV1-v1) and the corrected version of RCV1-v1 (we called it RCV1-v2).

- RCV1-v1: the raw RCV1 data contains 806,791 documents that are split chronologically into 23,307 training documents (news published from August 20 to August 31, 1996) and 783,484 test documents set (news released from September 1, 1996 to August 19, 1997). The feature vector representing a document is from the concatenation of text in $\langle headline \rangle$ and $\langle text \rangle$ XML elements. The complete feature space of RCV1-v1 is 47,236 in the training set.

- RCV1-v2 is converted from RCV1-v1 by the following modifications:

1) Remove from RCV1-v1 the 13 documents that violate the minimum code policy (i.e., each story was required to have at least one topic code) due to missing all Region codes, and the 2,364 documents missing Topics codes.

2) Add all missing ancestors of the code for each Topic present in a document. This increases 25,402 Topic code assignments.

Then the RCV1-v2 data includes 804,414 documents, which are split chronologically into 23149 training documents (unique document IDs ranging from 2286 to 26150, news published from August 20 to August 31, 1996) and 781,265 test documents (unique document IDs from 26151 to 810596, news released from September 1, 1996 to August 19, 1997). This split is called by Lewis *LYRL2004 split*. 47,219 features (unique stemmed words) of 47,236 occurring in RCV1-v2 training and/or test set. So 47,219 is the size of the complete feature set for RCV1-v2. Of these 47,219 terms, only 47,152 have one or more occurences in the RCV1-v2 training set, and so were used for training the models.

RCV1-v2 collection is the version of RCV1 used in his experiments by Lewis [67], and he provides the results of different models such as k-NN, Rocchio and SVM for RCV1-v2.

Although Lewis released all processed RCV1-v2 data including the stemmed tokens, binary vectors for each documents and so on, he still encourages to process the CD-ROMs files for different research purpose. From this point of view, we did the same as Lewis by extracting the $\langle headline \rangle$ and $\langle text \rangle$ parts of the original XML news files, and finally obtained another version of RCV1, which we called RCV1-v3.

- RCV1-v3 conforms to the *LYRL2004 split*, but it has 22,424 unique documents and 51,095 features in 23,037 training documents of RCV1-v1. Of these, 51,002 features are available in RCV1-v3 for training classifiers.

  The difference in the number of training documents can be explained by the duplicated documents obtained by the concatenation of text in $\langle headline \rangle$ and $\langle text \rangle$ XML elements in CD-ROMs:

  1) Duplicates in consecutive document IDs in RCV1-v2, such as the pairs of 2288 and 2289, 2334 and 2335, 2339 and 2440, and so on. These pairs of documents have exactly the same textual content and the number of duplicated pairs is 59.

  2) Duplicates in non-consecutive document IDs, for example, 2336 and 2338, 2299 and 2495, 2492 and 2506, and so on. 666 duplicates of such type are detected in RCV1-v2 training documents.

We have also a small difference in the number of tokens, i.e., 51,002 vs. 47,219, which is dues to slightly different preprocessing, e.g., the use different steamers or stop list.

It is worth noting that the Topics hierarchy contains the entire documents because of the minimum code policy, and business-related Industries hierarchy just share a part of them. In the Industries hierarchy, 352,361 of 804,414 news are assigned to six-layer, tree-shaped hierarchy, and they are divided into 9,644 news for training and 342,117 for testing by the LYRL2004 split.

3.1.2 Italian Dataset

This dataset is from the collaboration with Erickson Research Centre under Italian Project e-Value, whose aims are the reorganization or combination of educational materials in different pedagogical contexts. This refers to the selection and definition of educational programs tailored on specific needs of pupils, who sometime require particular attention and actions to solve their learning problems. Text categorization (TC) in this context is exploited to automatically extract several aspects and properties from learning objects, i.e., didactic material, in terms of semantic labels. These can be used to organized the different pieces of material in specific didactic program, which can address specific deficiencies of pupils.

In the case of e-Vaule, we studied different hierarchical taxonomies to capture the most information of educational text, i.e., the system we design must contain the characteristics of the educational texts. For this purpose, we did the following steps:

1) Designed a new taxonomy that meets the organization needs of e-Value (see Appendix B);

2) Defined an annotation procedure and produced an initial datasets;

3) Implemented a multi-class classifier (MCC) and performed accurate hierarchical categorization.

Regarding 1), we have defined a new taxonomy as well as the annotation procedure and initial datasets to meet the need of the e-Value project. Our partial hierarchical categorization scheme is shown in Figure 3.2, whose more descriptive labels are reported in Appendix B. The materials have to be classified according to four macro-categories, and then divided into a structure of sub-categories of 4 levels. Each category is meaningful for a correct description of the materials, from both administrative perspective (e.g., in which educational context should be applied) and subject/cognitive process viewpoint (e.g. Mathematics Number, Lexical and semantic processes instead of Mathematics Basic processes of calculus, Numerical facts). The Macro-categories are:

- C1 – School and class (referring to the ages 5 – 14);

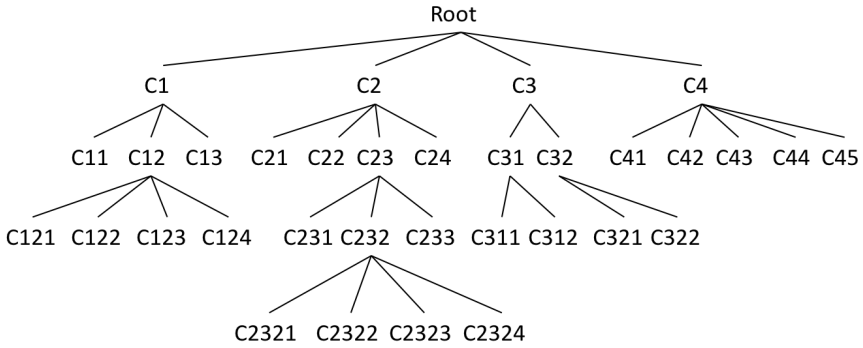

Figure 3.2: Partial hierarchical categorization scheme of e-Value

- C2 – Subject/cognitive process (referring to the subjects of mathematics, linguistics, phonetics, reading-writing abilities);

- C3 – Pupils situation (for the cases of special needs or particular situations); and

- C4 – Type of material (or the normal didactic usage in the class, or for pupils with special situation or greater difficulties in the subject).

Regarding step (2) we manually annotate 122 documents (the documents are repeated in the hierarchy), organized in a 112-category hierarchy (categories without symbol "-" ahead of label description in Appendix B) with 28120 unique features. A new updated version of category hierarchy contains 50 categories (symbolized by "*" after the label description of Appendix B) and 126 unique documents with 29103 features.

Concerning step (3), We carried out the TC experiments with state-of-the-art algorithms on this Italian educational dataset and the updated version in Section 3.6.5. Such automatic classification could improve the manual categorization costs, in terms of both time and human resource. Each piece of educational material, being part of a book, article or best practice, needs to be read and evaluated by experts, before being assigned to the proper categories, and this process takes a huge amount of time. Therefore, the use of an automatic classifier could significantly reduce the time required to read and evaluate the materials. Of course, experts will need to read part of the material in any case to refine and validate the output of the classifier. However, the materials pertaining to a certain subject can be directly routed to the experts of such field, thus improving the categorization accuracy.

3.1.3 Large-scale DMOZ Datasets

The DMOZ dataset is from the challenge on large scale hierarchical text classification (LSHTC) [1], which has been constructed by crawling Web pages that are found in the Open Directory Project (ODP) [2]. The information was translated into feature vectors (content vectors) and split into 300,000 training and 94,756 test documents.

DMOZ consists of 35,448 categories in a five-layer structural hierarchy, and 27,875 of 35,448 are leaf categories. For each level of DMOZ hierarchy, we compute the numbers of categories for each level, and they are 11, 343, 3,670, 13,255 and 18,169 for levels from 1 to 5.

A document of DMOZ may belong to more than one category but this phenomenon is rare, and the average leaf category per document is 1.0239. The hierarchy of this dataset is a *tree*, i.e. each node has only one parent.

3.2 Document Preprocessing

Real world textual dataset consists of large volume of documents, which are collected from heterogeneous sources. Due to this heterogeneity, such documents tend to be inconsistent and noisy. If they are inconsistent, it will be likely to lead to confusions for the mining process and then result in inaccurate performances. In order to extract correct and consistent data, document pre-processing is necessary to be applied, and the objective is to enhance the text quality by mining the key features or key terms from online news text documents so that they can be used properly in many applications, such as text clustering and text categorization.

The goal behind document pre-processing is to represent each document as a feature vector, i.e., to separate the text into a list of significant keywords that carry the meaning. The activity of choosing the most important words is known as feature selection. This is a fundamental preprocessing step necessary before weight word computation and document indexing.

The importance of pre-processing becomes more and more critical as the quantity of training data in supervised learning grows exponentially with the dimension of the input space. It has already been shown that the time spent on pre-processing can take from 50% up to 80% of the entire classification process [76] in text categorization. This clearly shows that the preprocessing in text classification is necessary and significant.

This section discusses the various preprocessing techniques used in the present research work to convert a raw textual document into an indexed weighted feature vector. The

[1] http://lshtc.iit.demokritos.gr/
[2] http://www.dmoz.org/

document processing workflow includes: Section 3.2.1, which describes the processes of parsing, tokenization, stopword removal and stemming to get the meaningful words. Section 3.2.2, which explains how to create inverted indexes for the words in Section 3.2.1 and Section 3.2.4, which introduces the most well known weighting scheme for the indexed words.

3.2.1 Parsing, Tokenization, Stopwords removal and Stemming

The first thing for document pre-processing is to parse the information stored in diverse formats, such as HyperText Markup Language (HTML), Adobe Portable Document format (PDF), Extensible Markup Language (XML),Microsoft Word (DOC) and so on, into a *canonical* format, i.e., plain text and then tokenize the text. Typical implementations first build a *Tokenizer*, which breaks the stream of characters from the document into raw *Tokens*. One or more token filters may then be applied to the output of the *Tokenizer*, such as *lowercase filter* converting all words into lowercase and *split filter* recognizing word boundaries. The main use of tokenization is to explore the meaningful keywords in a sentence. This process is language-specific, which means that it is non-trivial for some language like Chinese, Japanese.

Once we have tokenized our document, we could discover if there is a certain number of most frequently used words in English, actually many of them are useless in Information Retrieval (IR) and text mining. These words are called *stop words*, which are language-specific functional words, carrying no information, i.e., prepositions, pronouns, conjunctions and articles. In English language, there are about 400–500 stop words. Examples of such words include “a”, “to”, “of”, “and”. Three drawbacks of stop words are:

- Increase the indexing file size since the stop words account for 20-30% of total word counts;
- Decrease the efficiency and effectiveness since they are not useful for practical application;
- Confuse the retrieval system.

To delete these stop words, we can apply the appropriate filter to remove the tokens from the token stream containing a stop word and get the results. Most work uses the SMART stop word list [98], which has shown to be very important [130].

As a supplement step, it is to erase infrequent words, for instance, to erase words with a frequency less than or equals 2 in a massive volume of dataset, to avoid very big feature space resulting in low efficiency in vector similarity computation.

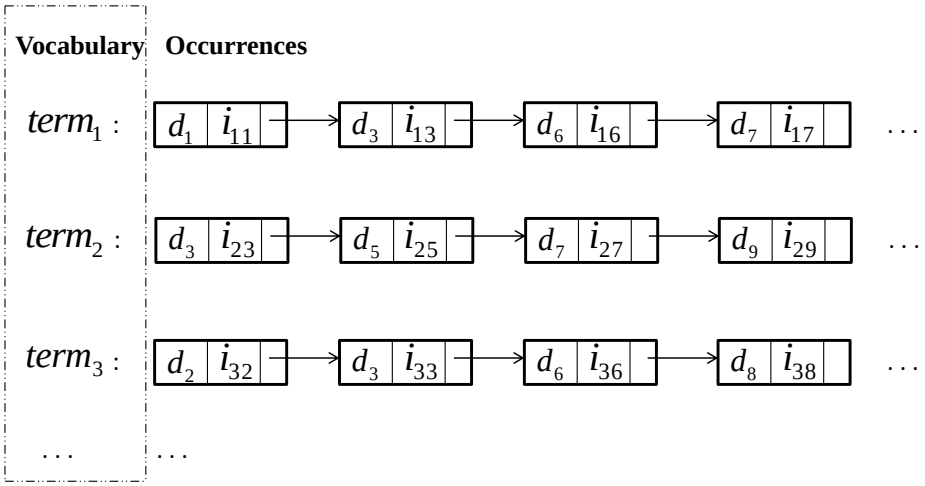

Figure 3.3: Simple illustration of an inverted index

After removing the stop words from tokenized words, the remaining one are reduced to their base form or *root*, e.g., user, users, used, and using belong to use. Stemming or lemmatization is a technique for deriving the root/stem of a word. Basic stemming methods use a set of rules, e.g., Porter Stemmer [89]. The English (Porter2) stemming algorithm[3] is an improved version of its predecessor Porter Algorithm[4] and is subject to continuous improvements. This algorithm uses rules which are divided into five different stages and these rules are applied to all the words one by one.

The advantages of stemming lie in:

- Improving effectiveness of TC, IR and text mining by matching similar words.
- Reducing indexing size by combing words with same root, this may reduce indexing size as much as 40-50% [117].

3.2.2 Inverted Indexing

Inverted index is a word-oriented mechanism for indexing a text collection to compute the word weight as $TF \times IDF$, and to speed up the searching task in IR. The inverted index structure is basically composed of two elements: the vocabulary and the occurrences. The former is the set of all different terms in the text and the latter are postings lists, one associated with each term appearing in the vocabulary. The structure of an inverted index is illustrated in Figure 3.3, a posting list is a set of linked postings, each of which is composed of three parts:

[3] http://snowball.tartarus.org/algorithms/english/stemmer.html
[4] http://snowball.tartarus.org/algorithms/porter/stemmer.html

1) *Document ID*, records the document ID that contains the corresponding term. Generally, postings of one term are sorted by document ID, although other orders are possible as well. Note that the document IDs have no inherent semantic meanings as they are assignments of numeric ids to documents.

2) *Term information*, keeps the information about occurrences of the term in the document specified by document ID. This information can be nothing for simple boolean retrieval. The most common information, however, is term frequency tf, or the number of occurrences the term appears in the document. Also it can be the positions of every occurrence of the term in the document, properties of the term or even anchor text information in web pages associated with hyperlinks.

3) *posting link*, a link is used to connect the documents that include the term.

In the example shown in Figure 3.3, we see that $term_1$ occurs in document set $D_1 = \{d_1, d_3, d_6, d_7, ...\}$, and there are different information for d_1, d_3, d_6, d_7, such as $i_{11}, i_{13}, i_{16}, i_{17}$. This is the same as the posting lists of $term_2$ and $term_3$ by defining i_{mn} as the information of $term_m$ occurs in document d_n.

3.2.3 Feature Selection

After pre-processing and indexing steps, the feature set dimensions of text articles still can be in the order of tens of thousands. The computational complexity with such size prevents the applicability and efficiency of many learning algorithms if we simply take it as dimensionality of the feature space [131], so feature selection methods, besides stopword removal and stemming, are often necessary to further reduce the size of the feature space (as long as they do not significantly impact the classification performance). This can be guaranteed by choosing a high quality subset of features, i.e., keeping more relevant words while eliminating irrelevant ones from the original feature set. This improves efficiencies in model construction and may produce higher classification accuracy in text classification.

Simplest approach for getting such subset is an exhaustive search: first select a criterion, and then evaluate all possible subsets. This, however, is computationally prohibitive, so automatic feature selection models are developed for the removal of non-informative terms according to corpus statistics and the construction of new (extracted) feature space. Each of these methods uses a term-goodness criterion threshold to filter the terms from the vocabulary of a document corpus. Commonly used five criterions include information gain (IG) [69], a χ^2 statistics (CHI) [102, 127], mutual information (MI) [127, 102], term strength (TS) [122, 132], and document frequency (DF) [131]. As pointed out in [131], DF, IG and χ^2 work best as selectors to reduce the feature set dimensionality and produce an increase in text classifier performances.

Recently, the previous techniques have been introduced even for selecting the relevant

n-grams [16] to add informative features. It was proved that such extended features bring extra information and often they help to increase the performance of simple features. The problem is that n-grams impact on the ranking of other features. This will cause information loss when selection is applied in only a limited number of (i.e. top ranked) features.

3.2.4 Document Weighting

After the application of feature reduction algorithms one can expect to bring the size of a typical feature vector down from hundreds of thousands of dimensions to a few thousands of dimensions [75]. The classification problem becomes much less complex than before. Documents now can be treated as a bag of words or terms. Then, we need to model our documents as vectors by weighting schemes. Given a vocabulary set of terms $T = \{t_1, t_2, ..., t_{|T|}\}$ extracted from *training set* $D = \{d_1, d_2, ..., d_{|D|}\}$, where $|T|$ is the size of vocabulary T and $|D|$ indicates the number of documents in D, the most well known weighting scheme is **TF** $\times$ **IDF** used in *SMART* [97], where:

- **TF**: *term frequency*, is the weight of a term t_i in document d_j represented as the number of times that t_i appears in d_j, denoted by f_{ij}. Normalization with maximum is commonly applied for each document and the normalized **TF** denoted by tf_{ij}:

$$tf_{ij} = \frac{f_{ij}}{max\{f_{1j}, f_{2j}, ..., f_{|T_j|j}\}} \tag{3.1}$$

  where $|T_j|$ is the number of terms in d_j, and $j = 1, ..., |D|$

- **IDF**: *inverse document frequency*, if the document frequency of a document d_j in the corpus D is df_i, the inverse document frequency denoted by idf_i can be expressed as:

$$idf_i = log_e \frac{|D|}{df_i} \tag{3.2}$$

Based on Equation 3.1 and Equation 3.2 the term weight w_{ij} of term t_i in document d_j is:

$$w_{ij} = tf_{ij} * idf_i = tf_{ij} * log_e \frac{|D|}{df_i} \tag{3.3}$$

A second weighting scheme used in [50, 67] is $log_e(TF) \times IDF$, It uses the logarithm of tf_{ij} as follow:

$$tf'_{ij} = \begin{cases} 0 & \text{if} \quad tf_{ij} = 0; \quad \text{(3.4a)} \\ log_e(tf_{ij}) + 1 & \text{otherwise.} \quad \text{(3.4b)} \end{cases}$$

Accordingly, the document weights w_{ij} is:

$$w_{ij} = tf'_{ij} * idf_i = tf'_{ij} * log_e \frac{|D|}{df_i} \tag{3.5}$$

The third weighting scheme is referred to as **TF** $\times$ **IWF** [5] (Inverse Word Frequency) is defined as:

$$iwf_i = log_e \frac{|T|}{tf_i} \tag{3.6}$$

where iwf_i is the inverse word frequency of term t_i. Consequently, the document weights w_{ij} is:

$$w_{ij} = tf_{ij} * iwf_i = tf_{ij} * log_e \frac{|T|}{tf_i} \tag{3.7}$$

In order to balance the **IWF**'s biased contribution, its square is thus preferred in [5].

We use the second weighting scheme for the basic model construction in this thesis, same as in [67].

3.3 Multi-class and Multi-label Classification

3.3.1 From Binary to Multi-class Classification

Supervised multi-class classification algorithms aim at assigning a class label for each input example. The difference with the binary classification problem is that the former selects from a label subset of a size larger than 2, whereas the latter chooses the class label from a subset of exactly two elements.

Let $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_l\}$ denote the domain of instances, $\mathcal{Y} = \{y_1, y_2, ..., y_p\}$ be the finite set of class labels, and a set of training data:

$$\{(\boldsymbol{x}_1, Y_1), (\boldsymbol{x}_2, Y_2), ..., (\boldsymbol{x}_l, Y_m)\},\ \boldsymbol{x}_i \in R^n, Y_i \subseteq \mathcal{Y}$$

where $\boldsymbol{x}_i$ denotes the n-dimensional *real* vector for the ith example, y_i is the class label and Y_j the class set of $\boldsymbol{x}_j$. In multi-class classification, $p > 2$ is the size of class labels, and $\forall i, |Y_i| = 1$. Our goal is to learn a model $\mathbb{H}$ such that $\mathbb{H}(\boldsymbol{x}_i) = y_i$ for new unseen examples.

Several algorithms have been proposed to solve this problem in the two class case, which means $|C| = 2$, some of which can be naturally extended to the multi-class case ($|C| > 2$), which considers all classes at once, such as k-Nearest Neighbor [6], naïve Bayes classifiers [93], decision trees [13, 91], neural networks [118] and extended SVMs [12, 28, 66, 126]. [27] is another multi-class classification method, which encodes the class label as output. All these methods construct one multi-class classifier on the training data, which inevitably brings computational complexity problems. In particular, methods solving multi-class

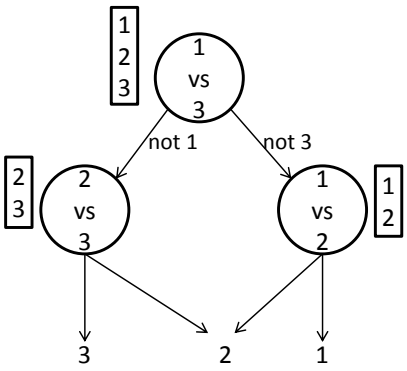

Figure 3.4: The decision DAG for three classes

SVM in one step requires a much larger optimization problem. So up to now experiments of building one multi-class classifier are limited to small datasets.

Another typical direction of multi-class classification algorithms focuses on decomposition implementations. They decompose the multi-class classification problem into binary classification tasks that can be solved efficiently using binary classifiers. Several methods have been proposed for such decomposition and have been investigated for extending SVMs for handing multi-class problem, e.g.:

- All-vs.-All (AVA) or One-vs.-One (OVO): this approach constructs $N(N-1)/2$ classifiers, each of them discriminate between pair of classes. In the test phase, it adopts max-win strategy and finally the class with the maximum number of votes wins. This method was first introduced in [57], and the first use of this strategy on SVM was in [59].

- One-vs.-All (OVA) or One-vs.-Rest (OVR): n classifiers can be constructed, one for each class, the ith classifier differentiates class i from the remaining m-1 other classes. A voting mechanism like AVA across the n classifiers or some other measures can be applied to categorize the new unseen examples. This method is also implemented in LIBSVMs and commonly used until now [67].

- Directed acyclic graph SVM (DAGSVM): it is proposed by [88]. In the training phase, it builds $N(N-1)/2$ binary classifiers as all-vs.-all method. In the test phase, it uses a rooted binary directed acyclic graph composed of the $N(N-1)/2$ classifiers as internal nodes and N classes as leaves. For example, Figure 3.4 is a DAG built by three binary classifiers (expressed by cycles). When they test an example, they start from the root nodes, and $N-1$ decision nodes will be evaluated in top-down way to derive a final class as the answer.

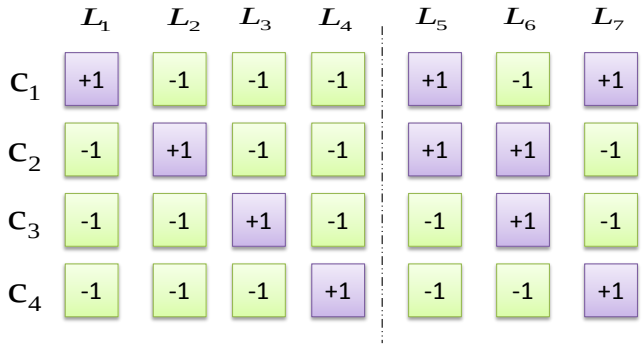

Figure 3.5: A codebook

- Hierarchical SVM (HSVM): this method suggested in [22] solves a series of max-cut problems: an undirected class graph with nonnegative edge weights is cut into two subgroups, and the cuts between these two subgroups receive the maximum weight. Then a binary SVM will be applied for solving the two-group problem. This approach is recursively applied to the two decomposed subgroups, until pure leaf nodes that have only one class label, are obtained. It has been shown that HSVM uses distance measures to weigh and exploit the natural class groupings. The hierarchical graph structure results in a fast and intuitive SVM training process that requires little running time and gives high classification accuracy and good generalization [22].

- Error correcting output codes (ECOC): it was proposed by [38]. It works by training N binary classifiers to distinguish between the K different classes. Each class receives a codeword of length L ($L = 10logK$ is suggested by [2]) according to a codebook M, randomly generated or by BCH codes method [9]. Each row of M corresponds to a certain class. Figure 3.5 shows an example for $N = 5$ classes and $L = 7$ bit codewords. Each class is given a row of the codebook and each column is used to train a discriminative binary classifier. If L=4, the first four columns in Figure 3.5 correspond to the OVA method. We could still construct the codebook by bringing code "0" to reach the OVO method. When testing an unseen example, the output codeword from the N classifiers is compared to the given L codewords, and the one with the minimum hamming distance is considered to be the class label for that example. Results in[2] show that this approach is in general better than the OVA and OVO approaches.

Other work includes binary hierarchical classifier (BHS) [63] and Divide-By-2 (DB2) [121] and so on. Different methods provide accuracy on different datasets and parameters thus it is difficult to rank them. Those using SVMs and OVA or OVO are the most successful

Ex.	M11	M12	M13	M14
1	X		X	
2	X			X
3		X		
4		X	X	X

Table 3.1: Example of a multi-label data set.

Ex.	M12	M11 ∧ M13	M11 ∧ M14	M12 ∧ M13 ∧ M14
1		X		
2			X	
3	X			
4				X

Table 3.2: Transformed data by using TR_1.

and widely used whereas the most well-known benchmark of classification problems is provided [67].

3.3.2 Multi-label Classification

Multi-label classification algorithms like multi-class classification are applied for problems with class labels larger than 2, but the difference is that it can return more than one labels instead of only one in the latter for an unseen example, i.e., the number of learning model $\mathbb{H}$ applied in an example $\boldsymbol{x}_i : |\mathbb{H}(\boldsymbol{x}_i)| > 2$.

The existing methods for multi-label classification mainly fall into two categories [116]:

- Problem transformation methods: those methods that transform the multi-label classification problem either into one or more single-label classification problems. To exemplify these typical methods we will use the data set of Table 3.1. It consists of four subcategories of the Markets (MCAT) category: Equity Markets (M11), Bond Markets (M12), Money Markets (M13) and Commodity Markets (M14). In order to avoid loss of the data information, there are two kinds of transformations commonly used in previous work.

  One transformation in Table 3.2 considers each different set of labels that exist in the multi-label data set as a single label, and constructs one binary classifier (i.e., OVA for SVM) for each of such classes [11, 39]. The other in Table 3.3 transforms the original data set into $|\mathcal{Y}|$ single-label data sets. Each of which contains all examples in the original data set, labeled as y_i if the labels of the original example contained y_i and as $\neg y_i$ otherwise. A binary classifier is also learned for the single-label dataset. The second is the most popular transformation in current research [65, 43, 11] and we use it in this thesis as well.

- Algorithm adaptation methods: they are adaptations of a specific learning approach, or directly extend some specific algorithms to solve the multi-label classification. Representative work includes adaptation of C4.5 algorithm in [23], two extensions of AdaBoost (Adaboost.MH and Adaboost.MR) in [100], an adaptation of the k-NN lazy learning algorithm (ML-k-NN) in [70] and so on.

Ex.	M11	¬M11	Ex.	M12	¬M12	Ex.	M13	¬M13	Ex.	M14	¬M14
1	X		1		X	1	X		1		X
2	X		2		X	2		X	2	X	
3		X	3	X		3		X	3		X
4		X	4	X		4	X		4	X	

Table 3.3: Transformed single-label data for M11, M12, M13, M14.

3.4 Hierarchical Text Classification

Hierarchical text categorization shows the feasibility and efficiency in real-world applications, e.g., Yahoo! Categories and Dmoz. These involve a large number of categories and documents, making the conventional multi-label classification methods such as SVM, k-NN and Rocchio in hierarchical categorization like [67] inadequate. To produce a sufficient hierarchical model under the scalability condition, the structure of the hierarchy must be exploited. There are basically two existing hierarchical methods, namely the *big bang* approach and the *top-down* approach, taking the hierarchy into account. In the following sections we described these two approaches and introduced some related work.

3.4.1 Big-bang Method

Big-bang approach implicitly uses the class hierarchy to learn a single (but generally complex) hierarchical classification model during training, and assigns classes of the hierarchy to test examples by this model.

At present, there exist a line of the big-bang algorithms (see [99, 124, 125, 123, 64] for example). A good survey of the big-bang approaches in the hierarchical classification up to year 2001 was provided in [73]. In this survey, the author applied a statistical technique called shrinkage, a particular form of smoothing to derive improved estimates of parameters for the class conditional distributions. The hierarchy is thus used to overcome sparseness problems in parameter estimation and not in a divide-and-conquer manner. By employing a simple form of shrinkage, the author created new parameter estimates in a child by a linear interpolation of all hierarchy categories and finally improved the accuracy using the naive Bayes text classifier.

[112] extended the work in [73] by presenting a hierarchical mixture model and developing an improved Expectation Maximization algorithm. The hierarchy of topics was used to provide estimates for class conditional term probabilities and to obtain a differentiation of words in the hierarchy according to their level of generality/specificity. The inner nodes of the hierarchy represent abstraction levels with their corresponding specific vocabulary. Each word in a document is assumed to be generated from abstraction level on the path

from the document class node to the root. They evaluated their model on two virtual trees constructed from 20 Newsgroups and Reuters-21578 respectively. Compared to the Hierarchical Shrinkage model proposed by [73], the hierarchical mixture model achieved better results on Newsgroups when limited number of training documents are given.

The more recent work [14] proposed a novel hierarchical classification approach to generalize support vector machine learning. This method considers the relationships among categories in different layers and designs discriminative functions structured according to the hierarchy. All parameters were learned jointly by optimizing a common objective function corresponding to a regularized upper bound on the empirical loss. Finally the author adopted the more competitive classifier SVMs on the WIPO-alpha patent collection to show the performances and confirmed the advantages and competitiveness. According to [67], it was possible to minimize upper bounds on arbitrary loss functions, in particular ones that quantifies the seriousness of incorrect classification based on the category hierarchy.

From the previous works we can see the big-bang methods only have a single classifier trained on the whole hierarchy, which has at least two drawbacks:

- Time complexity in the training phase for large-scale real-world dataset: [71, 133] have proved that it is infeasible to directly build a classifier for a large-scale hierarchy.
- The classifiers in the big-bang approaches constructed may not be flexible enough to dynamically account for changes to the category structure. The classifiers need to be retrained once the category structure is changed.

3.4.2 Top-down Method

The most commonly used top-down approach explores the hierarchical structure to decompose the entire problem into a set of smaller sub-problems and deals with such sub-problems in top-down way along the hierarchy. This allows for high efficiency in both learning and prediction since each time a much smaller problem with corresponding feature set is addressed.

Each *sub-problem* is constituted by one internal category and all its direct child nodes, for example, we can divide the sub-hierarchy of MCAT from RCV1 in Figure 3.6 into three sub-problems. Let's name them by the labels of each root category, i.e., MCAT, M13, M14. In the classification phase, we apply the multi-classifiers in top-down way to choose sub-problems for the next task. Look back to the Figure 3.6, the multi-classifiers of the first sub-problem of MCAT decides the category M11 and M14 as the candidates for a given instance. Since M14 is non-leaf node, we need to further deal with the sub-problem of M14. On the contrary, the sub-problem of M13 is disregarded since M13 is

not predicted as candidate. Furthermore, we will never consider the subtree rooted at M13. This ensures the efficiency in classification phase.

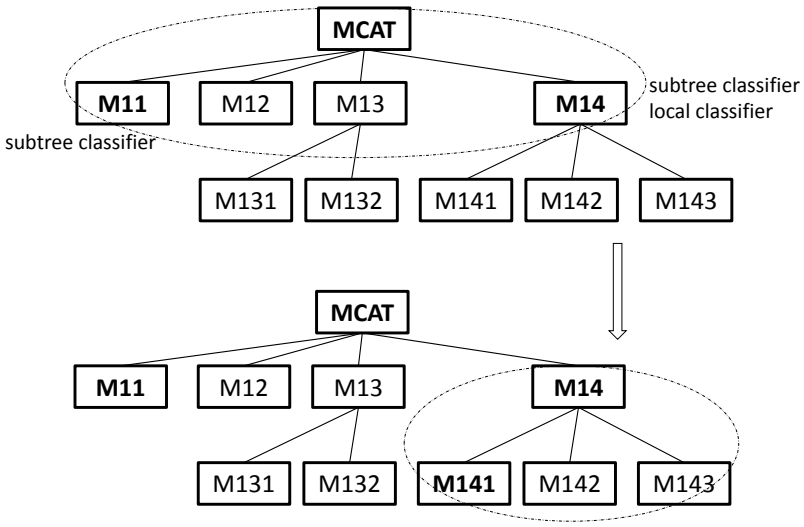

Figure 3.6: Classification in top-down manner

One case of the document distribution in the hierarchy is that only the leaf nodes have the documents (i.e., the non-leaf nodes are unassignable). This refers to an early top-down method commonly known as the pachinko-machine search [71], which greedily makes decisions from root node until the leaf nodes by applying the *subtree classifier* of each node. A more usual case is that the internal categories can also have some documents. [109] introduced one more *local classifier* for each non-leaf node to stop the greedy decision process and make internal nodes assignable.

It is worth noting that subtree classifier of a node decides whether an instance belongs to the subtree rooted at this node, once it is, the local classifier of this node decides if an instance belongs to this node itself rather than its descendants. Only if the instance were accepted by the subtree classifier, the local classifier will have the chance to further classify this instance.

In this top-down method, both subtree classifier and local classifier are usually learned with one-vs.-all strategy in SVMs. To build such classifiers, special consideration to the dependencies of categories in the hierarchy should be given by the feature selection of the training documents for each classifier. Before introducing the positive and negative training set for each category in the induced classifier, we define two concepts:

- *child-free*: if the category C_i is child-free in a hierarchy, denoted by $CFree(C_i)$, it will only contain the documents belong to C_i itself.
- *child-full*: if the category C_i is child-full, denoted by $CFull(C_i)$, it will include all

the documents of the subtree rooted at C_i.

Then the positive and negative examples for constructing the subtree classifier and local classifier of node C_i can be represented as:

- *subtree classifier*: positive examples: $CFull(C_i)$ and negative examples: $\forall d \in D$ such that $d \in CFull(C_{parent})$ and $d \notin CFull(C_i)$, where C_{parent} is the father node of C_i.
- *local classifier*: positive examples: $CFree(C_i)$ and negative examples: $\forall d \in D$ such that $d \in CFull(C_{parent})$ and $d \notin CFree(C_i)$, where C_{parent} is the father node of C_i.

Algorithm 1 Hierarchical classification in top-down manner.

```
S ← ∅ // Returns S: the positive category set of prediction
function PREDICT(Root, x)
  // Root is the root node in the hierarchy, and x is the test instance
  v ← LOCAL_PREDICT(Root, x)
  if v = 1
    S ← S ∪ {Root}
    return S
  for each subcategory C_i of Root
    v ← SUBTREE_PREDICT(C_i, x)
    if v = 1
        S ← S ∪ {C_i}
        PREDICT(C_i, x)
  return S
```

Once we have the two kinds of classifiers, in the classification phase, we apply them in top-down way as described in Algorithm 1, where:

- PREDICT($Root, \boldsymbol{x}$): classifies the instance $\boldsymbol{x}$ on the hierarchy rooted at node $Root$.
- SUBTREE_PREDICT($C_i, \boldsymbol{x}$): applies the subtree classifier of node C_i for $\boldsymbol{x}$,
- LOCAL_PREDICT($C_i, \boldsymbol{x}$): applies the local classifier of node C_i for $\boldsymbol{x}$.

A large number of researches, such as [109, 58, 95, 31], adopted this top-down strategy in hierarchical classification but such methods still suffer some potential problems as follows:

1) *Ineffective learning.* The problems of data sparseness and skewed distribution induce ineffective learning. [22] has pointed out that when working with imbalanced data, SVMs will produce a less effective classification boundary.

2) *Error propagation.* Error propagation is a typical problem in hierarchical classification. If an upper classifier made a wrong decision for the document, then it would choose a wrong path to traverse the taxonomy and would no longer have a chance to find the correct category.
Other top-down methods include:

- Instead of introducing a local classifier to avoid the severe disadvantage of the greedy decision process from the root to bottom, Dumais and Chen [40, 45] proposed a scoring rule that further took advantage of the hierarchy by considering only second-level categories that exceeded a threshold at the top level . The author explored two general ways: a) the logic and relationship between the probabilities from first layer and the second layer ($P(L1)\&\&P(L2)$); b) a product of probabilities from the two layers ($P(L1) * P(L2)$). The author further defined the threshold to be exceeded to pass a test document down to descendant categories. The binary SVM classifiers were used on the top-two levels of the LookSmart categories with 163 categories in total, and found small advantages in accuracy for hierarchical models over flat models.

- More recent work [71] extended the work in [40, 45] by taking into account the scalability of the hierarchical classification. The author developed a scalable system for large-scale text categorization, and theoretically analyzed and experimentally evaluated the SVMs in hierarchical and non-hierarchical settings of classification over the full taxonomy of the Yahoo! directory. This was the first work using SVMs for such large-scale Yahoo! directory in hierarchical classification. Like [40, 45], an investigation of threshold tuning algorithms with respect to time complexity and their effect on the classification accuracy of hierarchical SVMs in one-vs.-all mode was conducted.

- To better utilize the hierarchical structures, [8] adopted a cross-validation strategy to take the output of the top-level classifiers as features for the second-level classifiers. A bayesian aggregation on the result of the individual binary classifiers was proposed in [34]. [129] proposed a pruning technique for the large-scale hierarchy based on the test instance and re-training on the smaller hierarchy.

- Large-margin discriminative methods proposed by [115] and [14]. The former proposed to appropriately generalize the well-known notion of a separation margin and derive a corresponding maximum-margin formulation for dealing with more complex output, and the latter used discriminative functions structured in a way that mirrors the class hierarchy, and the parameters are jointly learned to minimize a global loss

over the hierarchy. Similar online variants of such methods have been proposed in [35, 18].

- In [128], 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. Hierarchical shrinkage approaches such as isotonic smoothing in [90], smoothing the estimated parameters in Naive Bayes classifiers along the path from root to the leaf node in [73] have also been tried. [103] proposed Multinomial logistic models that take sum of contributions along the path with bayesian priors on the variances. For a more thorough survey and comparison of hierarchical classification methods please refer to [107].

3.5 Categorization Measurements

3.5.1 Precision, Recall and F1

To measure the accuracy of the classifier traditional and widely used methods have been used. These refer to Precision and Recall, whose definition is reported below:

- Given a set of document T
- Precision = #Correct Retrieved Document/#Retrieved Documents
- Recall = #Correct Retrieved Document/#Correct Documents

Precision and Recall are usually combined in a single measure to have a more compact parameter, which is called F1-measure. It is the harmonic mean between Precision and Recall, i.e.:

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.8}$$

Since we measure the performance of a set of classifiers, i.e., one for each node of the hierarchy, we need to combine the Precision, Recall and F1 for all classifiers. This is usually done as follows. Given the following quantities calculated for each classifier i:

- a_i, number of corrects (i.e., documents in the intersection of the correct documents set and retrieved documents set);
- b_i, number of mistakes (i.e., documents in the correct documents set, but not in the intersection of the correct documents set and retrieved documents set);
- c_i, number of not retrieved (i.e., documents in the retrieved documents set, and not in the intersection of the correct documents set and retrieved documents set).

The *Precison* and *Recall* are defined by the above counts:

$$Precision_i = \frac{a_i}{a_i + b_i} \tag{3.9}$$

$$Recall_i = \frac{a_i}{a_i + c_i} \tag{3.10}$$

and F1 measure is:

$$F_1^i = \frac{2 * Precision_i * Recall_i}{Precision_i + Recall_i} \tag{3.11}$$

The global performance of a set of classifiers is measure with

- the Micro-average (defined below) and
- the Macro-average (average over all individual classifiers)

The Micro-Average is less intuitive than the arithmetic average (i.e., the Macro-average). The formula is:

$$Micro\_Precision = \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n}(a_i + b_i)} \tag{3.12}$$

$$Micro\_Recall = \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n}(a_i + c_i)} \tag{3.13}$$

$$Micro\_F1 = \frac{2 * Micro\_Precision * Micro\_Recall}{Micro\_Precision + Micro\_Recall} \tag{3.14}$$

and the Macro-F1 is:

$$Macro\_Precision = \frac{\sum_{i=1}^{n} Precision_i}{n} \tag{3.15}$$

$$Macro\_Recall = \frac{\sum_{i=1}^{n} Recall_i}{n} \tag{3.16}$$

$$Macro\_F1 = \frac{\sum_{i=1}^{n} F_1^i}{n} \tag{3.17}$$

3.5.2 Multi-label Graph-induced Error

We also aim at demonstrating that our approach is effective for optimizing hierarchical classification. For this purpose, a hierarchical measure is needed, 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 farer node of the hierarchy. The Multi-label Graph-induced Error (MGIE) (suggested by ECML/PKDD 2012 Discovery Challenge) takes the distances between true positives and false positives by also over-penalizing the false negatives. It is computed as follows:

i) Find the smaller set between the true and the predicted classes of each document;

ii) Compute the minimum graph distance between each class of the smaller set and the closest class of the other set, in such a way that minimizes the sum of distances;

iii) Set all classes in excess equal to the maximum distance; and

iv) Add all the distances and divide them by the number of the classification tasks, where such number is equal to the sum of the true categories of each document.
In our experiments in Section 4.5.6 we define the maximum distance as five (and seven), so all distances above five (and seven) are treated the same.

3.6 Experiments and Evaluations

To implement the baseline model, we applied the state-of-the-art method used by [67] 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[5]. We used the LIBSVM[6] implementation, which provides a probabilistic outcome for the classification function. We also used the LIBLINEAR[7] by implementing the Platt's sigmoid function inside LIBLINEAR to generate a probabilistic output for the classification. Moreover, we compared the efficiencies of such two kinds of software in flat and top-down modes for both learning and classification phases.

Topics hierarchy of RCV1 covers all documents, and is mostly used dataset in literatures of machine learning area. In this thesis RCV1-v2 (or RCV1-v3) in the experiments represents the Topics hierarchy unless we give a specification for the other category hierarchy, i.e., Industries.

Additionally, we train an MCC based on SVMs for an Italian corpus, annotated according to a hierarchical taxonomy in the context of the educational framework. The classification performance obtained by applying such SVMs is promising for improving the production cycle of educational systems.

3.6.1 LIBSVM on RCV1

Lewis [67] released the benchmarks of different basic models such Rocchio, k-NN and SVM on RCV1, among which the SVM performed best using the software of SVM^{light} [8]. Additionally he also released the binary training/test vectors of RCV1, namely RCV1-v2 in Section 3.1.1. Since our reranking system in Chapter 4 is based on the basic SVM probability output, in this thesis we used another version of SVM software, i.e., LIBSVM,

[5] 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 because of the diverse stop lists or tokenizers.

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

[7] http://www.csie.ntu.edu.tw/ cjlin/liblinear/

[8] http://svmlight.joachims.org/

which can generate the classification probability by implementing the sigmoid function (let's call it proLIBSVM). For more details please refer to Section 2.3. In order to make the comparison between our reranking system and the state-of-the-art one-vs.-rest SVM meaningful, we first have to prove that our basic proLIBSVM matches SVM^{light} accuracy as Lewis did, by means of Micro- and Macro-Average F1 on the same data, i.e., RCV1-v2. Note that we used the exact same parameters (e.g., the penalty factor C=1) for proLIBSVM and SVM^{light} in learning the models.

For the purpose of comparison, we apply LIBSVM (without probability output) on RCV1-v2, and derived the corresponding performances in the fourth column of Table 3.4. These are much lower than those obtained by proLIBSVM in the third column. The reason is that Platt probabilities [87] with threshold 0.5 can make the decision simpler than just considering the decision value.

F1	Lewis' SVM^{light}	proLIBSVM	LIBSVM
Micro-F1	0.816	0.815	0.798
Macro-F1	0.567	0.565	0.517

Table 3.4: Classification performances of basic SVM models on RCV1-v2.

From Table 3.4, we could conclude that our proLIBSVM is completely equivalent to SVM^{light} on RCV1-v2. Note that the Macro-F1 in Lewis' SVM^{light} is 0.567 rather than 0.607 [67], since Lewis optimized the Micro-F1 and we followed his setting.

proLIBSVM on RCV1-v3, gets the same performances in Table 3.5. This explains that the difference (i.e., numbers of features and documents) between RCV1-v2 and RCV1-v3 does not really matter, and RCV1-v3 is reliable for the subsequent experiments. For comparison, we also try LIBSVM on RCV1-v3, and we verified again Platt's model [87].

F1	proLIBSVM	LIBSVM
Micro-F1	0.815	0.801
Macro-F1	0.565	0.515

Table 3.5: Classification performances of basic SVM models on RCV1-v3.

We also applied proLIBSVM on the Industries category hierarchy and reported the accuracy in Table 3.6.

We can clearly see from Table 3.6 that the baseline of proLIBSVM is 5 points and 2.6 points higher than Lewis' in Micro- and Macro-F1, respectively. This could be explained by the fact that the learning ability of the former is better than the latter on a smaller training set of RCV1 (9,644 vs 23,307 for Industries and Topics). In particular, under the fact that SVM^{light} equals to proLIBSVM on the whole Topics datasets (RCV1-v2 or

F1	Lewis' SVM^{light}	proLIBSVM
Micro-F1	0.512	0.562
Macro-F1	0.263	0.289

Table 3.6: Classification performances of basic SVM models on Industries of RCV1.

RCV1-v3).

3.6.2 LIBLINEAR on RCV1

LIBSVM (or proLIBSVM) on RCV1 in Section 3.6.1 costs much time for learning the 104 models and classifying over 800,000 instances. To make it more efficient, we explored the software of LIBLINEAR, which has been developed for large datasets. In order to make LIBLINEAR generate decision probability like proLIBSVM for our post processing, we implemented Platt's sigmoid function into LIBLINEAR as described in Section 2.3.4. We called this model proLIBLINEAR. Based on LIBLINEAR package, there are four main parameter settings under the recommendation of bias item b=1: L2-Regulation L1-Loss Dual, L2-Regulation L2-Loss Dual (default in LIBLINEAR), L2-Regulation L2-Loss Primal, and L1-Regulation L2-Loss Primal. To find the best parameters for proLIBLINEAR, we applied proLIBLINEAR on RCV1-v3 from both efficiency and accuracy aspects. The results are reported in Table 3.7.

	L2R_L1Loss_Dual	L2R_L2Loss_Dual	L2R_L2Loss_Primal	L1R_L2Loss_Primal
Train Time(s)	265.69	273.34	494.58	761.84
Test Time(s)	726.68	752.88	746.45	780.17
Micro-F1	0.815	0.811	0.812	0.806
Macro-F1	0.564	0.560	0.561	0.553

Table 3.7: Classification performances of LIBLINEAR on RCV1-v3.

We can see from Table 3.7 that proLIBLINEAR with L2-Regulation L1-Loss Dual works best among others. Most importantly, proLIBLINEAR matches the proLIBSVM on RCV1-v3 in Micro- and Macro-Average F1, which means the former can generate the same accuracy while consuming far less time than the latter. We will present the detailed efficiency computations in Section 3.6.4.

To check again Platt's theory, we applied LIBLINEAR on RCV1-v3, and found consistent results, i.e., lower Micro- and Macro-F1 than those of using proLIBLINEAR.

3.6.3 Top-down Models on RCV1

Section 3.6.1 and Section 3.6.2 described the two flat classification systems (one-vs.-rest probabilistic model for each category) that match the state-of-the-art method in [67] on RCV1, but it completely ignores the tree-shaped hierarchy. Another commonly used top-down method proposed by [109] is described in Section 3.4.2. Since the probabilistic models (learned by proLIBSVM or proLIBLINEAR) perform better than the non-probabilistic models in flat case, we applied the probabilistic models in an efficient way from root to low-level nodes of hierarchy in top-down manner.

In the experiment, we applied the models from proLIBSVM and proLIBLINEAR software in top-down manner (let's call them hier_proLIBSVM and hier_proLIBLIENAR, respectively) on the RCV1-v3: the results are showed in Table 3.8.

F1	hier_proLIBSVM	hier_proLIBLINEAR	hier_LIBLINEAR
Micro-F1	0.819	0.819	0.807
Macro-F1	0.578	0.575	0.525

Table 3.8: Classification performances of top-down basic models on RCV1-v3.

We can clearly see the following points from Table 3.8:

1) hier_proLIBLINEAR performs equivalently with hier_proLIBSVM, the same as they are in the flat mode;

2) the top-down method with probabilistic models works slightly better than the flat probabilistic models (i.e., 0.05 and 1.0 points for Micro- and Macro-F1, respectively. More details on F1s of MCC are presented in Appendix D).

3) based on 1) and 2), using of hier_proLIBLINEAR instead of hier_proLIBSVM not only has the same accuracy, but also further improves the efficiency of learning and classification. We reported the efficiencies in Section 3.6.4.

Additionally, the comparison of hier_proLIBLINEAR and hier_LIBLINEAR in top-down way is consistent with results in flat mode.

3.6.4 Running Times

In previous sections we implemented several probabilistic baseline models that matched the state-of-the-art method used by [67] for RCV1. These models are based on the two pieces of SVM software: LIBSVM and LIBLINEAR, each of them is associate with two models according to the classification strategy (i.e., flat mode or top-down manner). The difference of such models in learning and classification impacts efficiency. In this section, we reported a detailed comparison in Table 3.9 for the four baseline models: proLIBSVM,

proLIBLINEAR, hier_proLIBSVM, and hier_proLIBLINEAR. The first two in the flat mode, and the other two in top-down way. We can see from Table 3.9 that:

	flat		top-down	
	proLIBSVM	proLIBLINEAR	hier_proLIBSVM	hier_proLIBLINEAR
Training	150.38	2.11	79.58	1.45
Test	2604.00	18.75	1393.78	3.29

Table 3.9: time cost (minutes) of basic models on RCV1-v3 with nr_fold=5.

1) proLIBLINEAR is remarkably fast compared to proLIBSVM (i.e., 2min vs 80min in model training and 19min vs 1300min in classification), independently of whether proLIBSVM works in flat or top-down mode.

2) hier_proLIBLINEAR is the most efficient model, since it is based on proLIBLINEAR and explores the category hierarchy to work in top-down manner.

3.6.5 Experiments on Italian Dataset

The aim of our evaluation is to demonstrate that state-of-the-art TC methods can be applied to learn hierarchical classifiers for our e-Value taxonomy. This task is made complex by two different aspects:

i) in addition to topic labels such as, *Euclidean Geometry*, *Problem Solving* or *Geometric Transformation*, the taxonomy also contains semantic characterization such as *Story Development* or *Story Understanding*, whose characterization using simple terms seems harder;

ii) given the novelty of the taxonomy, we could only produce a small dataset, which makes the learning of classification functions more difficult.

To deal with and analyze such problems, we experimented with hierarchy subsets, defined according to the levels if hierarchy, ranging from 1 to 4 (the maximum depth of our hierarchy). The deeper the level, the more difficult TC is.

One major drawback of machine learning and thus of TC based on it is the need of training data, i.e., a set of documents manually classified into the referring taxonomy. This data is difficult to find and/or to produce, as it requires human labor. Given the novelty of our taxonomy defined in Appendix B, no previous data was available. Thus, we set an annotation procedure (with only one annotator) of the didactic material available in the Ericksons database. We randomly selected 60 documents and we classified each of them according to all the 112 nodes of the taxonomy. This led to a dataset of 122 documents (repetitions are considered). As for the updated version of dataset, we select 91 of 126 unique documents as training set for the 50-category hierarchy.

We randomly divided the above data in training and test set by taking care that for each document all its repetitions were all put either in the training or in the test set. The training data was used to learn the set of 112 (and 50 for the updated data) binary classifiers, one for each category, following the one-vs.-all schema. The output of the multi-class classifier is the merged set of the individual binary classifier decisions. Although simple, this is considered a state-of-the-art approach [92, 67]. We used default SVM parameters as the small training data prevented to apply any reasonable parameterization approach. We used a bag-of-term representation (string separated by space and punctuation) without applying any feature selection, stop list or lemmatization. Although, we are confident that the latter may relevantly improves our models. We used the classical log(TF)*IDF weighting scheme and normalized vectors.

The performance is provided by means of Micro- and Macro-Average Precision, Recall, and F1, evaluated from our test data over all categories, additionally, the F1s of all the binary classifiers. For measuring the performance of different hierarchical levels, only the nodes up to the target level are considered, e.g., for the first level, we only measure the Micro/Macro F1 of C1, C2, C3 and C4 (without C4 in the updated dataset).

Table 3.10 reports the performances on different category levels (more details on F1s of MCC are presented in Appendix C).

In the first level, we note that for each category there are a few documents for training. These seem to be enough as the accuracy of the individual categories as well as the overall Micro/Macro F1 is exceptionally high. This is not completely surprising as most documents are repeated in the above four/three categories.

We note that in the second level when the training documents are more than half in the first level, very good results can be achieved. Low performance is shown for C11 and C13, which are trained with less than 7 documents. Additionally, they have only one test document, this means that their accuracy cannot really be estimated. The situation of C31 is even worse as it has no test documents. In this case, we do not report any accuracy in the related row. It should also be noted that, since we use one-vs.-all schema, the accuracy of C1,..,C4 is the same as before. Thus, from now on, we will not report the accuracy of previously reported binary classifiers.

On levels 3 and 4, again the few training documents available for the classifiers prevent to achieve a reasonable F1. There are some good cases such as C124 and C322 but also bad cases such as C122 and C123. The latter two refer to Primaria Classe II and Primaria Classe III, respectively, which have large overlap with the other classes, i.e., I, IV and V. For separating such categories, the simple bag-of-words may not be enough.

	Level_1	Level_2	Level_3	Level_4
Micro-Precision	0.920	0.916	0.855	0.843
Macro-Precision	0.924	0.641	0.288	0.139
Micro-Recall	0.958	0.916	0.725	0.640
Macro-Recall	0.955	0.637	0.269	0.129
Micro-F1	0.939	0.916	0.785	0.727
Macro-F1	0.938	0.633	0.263	0.115

Table 3.10: Performance for the Italian dataset (112 categories)

	Level_1	Level_2	Level_3
Micro-Precision	0.887	0.825	0.761
Macro-Precision	0.885	0.531	0.330
Micro-Recall	0.909	0.862	0.665
Macro-Recall	0.990	0.558	0.403
Micro-F1	0.935	0.843	0.709
Macro-F1	0.933	0.535	0.390

Table 3.11: Performance for the updated version of Italian dataset (50 categories)

3.7 Conclusions

3.7.1 Discussion on RCV1

In this chapter we have presented the use of LIBSVM and LIBLINEAR in flat and top-down methods for text categorization on RCV1, which match the state-of-the-art method used in [67].

First, we applied probabilistic LIBSVM (proLIBSVM) 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 to match Lewis' SVM^{light} model on RCV1 [9].

Second, we have developed the proLIBLINEAR based on LIBLINEAR[10] by implementing the Platt's sigmoid function inside it to generate a probablisitic output, which is used for making the classification decision (instead of simply using the decision value from LIBLINEAR). The results show that proLIBLINEAR reaches the the state-of-the-art method for RCV1, most importantly, it has very high efficiency.

Third, we applied proLIBSVM and proLIBLINEAR in top-down manner on RCV1, and found that they perform a little bit better than when they work in flat case. This can

[9] We have just a small difference in the number of tokens in two versions of RCV1, but this is both not critical and rarely achievable because of the diverse stop lists or tokenizers.

[10] http://www.csie.ntu.edu.tw/ cjlin/liblinear/

be explained by the fact that the top-down method explores the hierarchical father-child dependencies. This top-down classification way of course further improves the efficiencies of the flat case. In particular, the proLIBLINEAR in top-down manner works very fast.

It is worth noticing that the probabilistic models (i.e., proLIBSVM and proLIBLINEAR) outperform the default non-probabilistic models such as LIBSVM and LIBLINEAR for RCV1. What we care more about is the output of probabilities, which we can use for learning the reranking of global classification hypotheses.

3.7.2 Discussion on e-Value

We have described an interesting and new semantic classification problem in the context of the educational framework of the e-Value project. We have defined a new hierarchical taxonomy, which is promising for improving the production cycle of educational systems. To test the feasibility of the approach, we have also built a corpus annotated according to the above taxonomy. Such data was used for training an MCC based on SVMs. The results show that when there is a reasonable amount of training documents the classifiers can deploy remarkably high accuracy. On the other hand, the F1 of lower level categories is highly affected by data scarceness. Some categories would probably require the definition of more expressive features to better model their separation.

Chapter 4

Structural Reranking for Hierarchical Text Classification

Chapter 3 described our reimplementations of two baseline models:

(a) Lewis' flat MCC, i.e., a one-vs.-all multi-classifier for each category, and the classifiers learned from two pieces of software: proLIBSVM in Section 3.6.1 and proLIBLINEAR in Section 3.6.2. Application of such models in the flat way constitutes the state-of-the-art method on RCV1 in [67], which is also argued in [92].

(b) Sun's hierarchical model [109] goes beyond the flat model as it is a top down algorithm so already exploiting the classification hierarchy by applying the hier_proLIBSVM and hier_proLIBLIENAR models (see Section 3.6.3).

However, both the flat and hierarchical models for hierarchical text categorization (HTC) lose the inherited semantic relations existing in the hierarchical taxonomy. Thus, efficiently deriving the interdependencies of categories in the hierarchy, is difficult but necessary for improving hierarchical text classification.

In this chapter advanced reranking systems were designed to exploit structural dependencies in hierarchical multi-label text classification (TC). They are based on two algorithms:

1) Generate the k-best classification hypotheses according to: (i) the classification probability of the flat MCC state-of-the-art model (i.e., (a) above); (ii) the more complicated probability of top-down method exploiting the node structure in the hierarchy (i.e., (b) above).

2) Encode dependencies in the reranker by: (i) modeling hypotheses as trees derived by the hierarchy itself and (ii) applying tree kernels to them.

Section 4.1 describes two algorithms of the k-best classification hypothesis generation, i.e., the flat hypothesis generation in Section 4.1.1 and hierarchical hypothesis generation in Section 4.1.2. Section 4.2 shows two ways of representing the hypotheses with

the hierarchy itself. Section 4.3 introduces the structural reranking method based on reranker learning in Section 4.3.2 and reranking system for classification in Section 4.3.3. Section 4.4 reports on experiments for the flat rerankers and Section 4.5 reports on the experiments for the hierarchical rerankers. Related work is analyzed in Section 4.6. Finally, Section 4.7 derives the conclusions on using the rerankers for HTC.

4.1 Hypothesis Generation

As claimed in Chapter 3, the conventional flat and top-down models used in hierarchical text classification ignore the correlations and dependencies existing inside the category hierarchy. The big-bang approaches take them into account but result in a high computational complexity in learning and prediction. Developing efficient models for hierarchical classification using global dependencies becomes necessary. For this purpose, we use reranking models to rank a set of initial hypotheses, which are typically generated by *local classifiers*, encoding the underlying pre-defined structural information. Meanwhile, the different kinds of compact representations of these hypotheses ensure the reranking efficiency.

In the following sections, we show two different frameworks for hypothesis generation: *flat*, i.e., we ignore the structural organization of the categories; and *hierarchical* in which the structure imposes constrains on the feasibility of the hypotheses.

4.1.1 Flat Hypothesis Generation

The flat hypotheses generation process is based on categories output and corresponding probabilities of the flat one-vs.-all models. Given a category hierarchy with n categories, $C_1, \ldots, C_n$, we can define:

- $p^1_{C_i}(d)$, i.e., the probability that the classifier i (associated with category C_i) assigns the document d to C_i, and
- $p^0_{C_i}(d)$, i.e., the probability that the classifier i does *not* assign the document d to C_i.

$p^1_{C_i}(d)$ can be computed by taking the SVM classification output (i.e., the decision value) as input of Platt's sigmoid function as described in Section 2.3.1. According to such function, the positive decision value corresponds to

$$p^1_{C_i}(d) > 0.5 \tag{4.1}$$

which means d belongs to C_i with the probability of $p^1_{C_i}(d)$. The latter increases proportionally to the increment of the decision score. On the contrary, the decision value less

than 0 corresponds to

$$p^1_{C_i}(d) < 0.5 \tag{4.2}$$

which means d does not belong to C_i with the probability of $p^0_{C_i}(d)$.

Since the multi-classifiers we used in flat case are independent, for any mapping of $< d, C_i >$ we have:

$$p^0_{C_i}(d) + p^1_{C_i}(d) = 1.0 \tag{4.3}$$

From Equation 4.2 and Equation 4.3 we can derive:

$$p^0_{C_i}(d) > 0.5 \tag{4.4}$$

if C_i is not assigned to document d.

After determining the $p^1_{C_i}(d)$ and $p^0_{C_i}(d)$ for each of the mappings such as $< d, C_i >$, we define the joint probability $\mathcal{P}$ as the scalar product of all individual decision probability to represent the probability of a complete assignment of a label set to a document:

$$\mathcal{P}(\mathbf{h}) = \prod_{i=1}^{n} \Big(p^{h_i}_{C_i}(d)\Big), h_i \in \{0, 1\} \tag{4.5}$$

where $h_i = 0$, if $d \notin C_i$ and $h_i = 1$, if $d \in C_i$.

Let us indicate with $\mathbf{h} = \{h_1, .., h_n\} \in \{0, 1\}^n$ a classification hypothesis, i.e., the set of n binary decisions for the document d. If we assume independence between the SVM scores, the most probable classification hypothesis on d is:

$$\mathcal{P}(\tilde{\mathbf{h}}) = \underset{\mathbf{h}\in\{0,1\}^n}{\operatorname{argmax}} \prod_{i=1}^{n} p^{\mathbf{h}_i}_{C_i}(d) = \Big(\underset{h\in\{0,1\}}{\operatorname{argmax}}(p^h_i(d))\Big)^n_{i=1}, \tag{4.6}$$

where the last equality works because we assume a flat hierarchy. From Equation 4.6 we can derive the most probable hypothesis, which is the one with the largest joint probability.

Let $\tilde{\mathbf{h}_1}$ be the hypothesis associated with the max joint probability, the second best hypothesis $\tilde{\mathbf{h}_2}$ can be obtained by changing some category decision as well as corresponding probability, for example, $p^0_{C_i}(d) = 0.501$ represents $d \notin C_i$ with probability equal to 0.501, if we change it for $d \in C_i$, the probability is $p^1_{C_i}(d)$ = 1- $p^0_{C_i}(d) = 0.499$. Note that we have marked C_i as positive for d instead of negative after changing the decision of d on C_i.

This change generates a second best joint probability. Thus in general, a second best hypothesis $\tilde{\mathbf{h}_2}$ must satisfy:

$$\mathcal{P}(\tilde{\mathbf{h}_2}) = \underset{\mathbf{h}\in\{0,1\}^n - \tilde{\mathbf{h}_1}}{\operatorname{argmax}} \prod_{i=1}^{n} p^{h_i}_{C_i}(d) \quad and \quad \mathcal{P}(\tilde{\mathbf{h}_2}) < \mathcal{P}(\tilde{\mathbf{h}_1}) \tag{4.7}$$

By storing the joint probabilities of the previous $k-1$ most probable configurations, the k-th best hypothesis $\tilde{\mathbf{h}_k}$ can be efficiently generated with the following conditions:

$$\mathcal{P}(\tilde{\mathbf{h}_k}) = \underset{\mathbf{h}\in\{0,1\}^n - \{\tilde{\mathbf{h}_1},..,\tilde{\mathbf{h}_{k-1}}\}}{\operatorname{argmax}} \prod_{i=1}^{n} p_{C_i}^{h_i}(d) \quad and \quad \mathcal{P}(\tilde{\mathbf{h}_k}) = \underset{i=1,..,k}{\operatorname{argmin}} \quad \mathcal{P}(\tilde{\mathbf{h}_i}) \tag{4.8}$$

To implement Equation 4.8 for getting the k best hypotheses, we first compute the top hypothesis as in Algorithm 1, then developed the Algorithm 2 for generating the next $k-1$ hypotheses.

Algorithm 2 Generation of the top flat hypothesis.

function TOP1(*Root*)
  // Returns the top hypothesis and its probability
  // $p(C_i)$ is the probability of (C_i) assigns to test instance
  // S is the positive category set of hypothesis
  // P is the joint probability of hypothesis
  $S \leftarrow \emptyset$, $P \leftarrow 1$
  for each subcategory C_i of *Root*
    if $p(C_i) > 0.5$
      $P \leftarrow P \cdot p(C_i)$
      $S \leftarrow S \cup \{C_i\}$
    else
      $P \leftarrow P \cdot (1 - p(C_i))$
  return $\langle S, P \rangle$

To make the flat hypothesis generation clearer, we illustrate the process by Figure 4.1–4.4 for the example $\boldsymbol{x}_i$ on MCAT subhierarchy, whose annotated labels are M131, M132, (no links between categories indicate that the flat generation algorithm does not consider the dependencies among categories). We set $k = 4$ in order to limit the exhaustive search.

- $\tilde{\mathbf{h}_1}$: Figure 4.1 is the basic SVM output with only M13 predicted as positive for $\boldsymbol{x}_i$, the probability marked *Prob* represents the probability that $\boldsymbol{x}_i$ belongs to that category. According to Equation 4.6, it has the biggest joint probability:
  $\mathcal{P}(\tilde{\mathbf{h}_1}) = (1-0.003)*(1-0.006)*(1-0.453)*(1-0.006)*(1-0.009)*(1-0.023)*$ $\mathbf{0.779}*(1-0.001)*(1-0.004)*(1-0.001) = 0.40397$.

- $\tilde{\mathbf{h}_2}$: Among all possible changes on $\tilde{\mathbf{h}_1}$ in Figure 4.1, Figure 4.2 changes the M12 decision, which has the least confidence among all decisions of basic SVM category outputs. This makes the second hypothesis $\tilde{\mathbf{h}_2}$ with the second largest joint probability:

Algorithm 3 Generation of the top k hypotheses.

```
function TopK(Root, k)
  // Returns the top k hypotheses
  // and their probabilities
  H ← ∅
  q ← empty priority queue
  Enqueue(q, Top1(Root))
  while |H| < k and q is nonempty
    repeat
      ⟨S, P⟩ ← Dequeue(q)
    until ⟨S, P⟩ ∉ H
    H ← H ∪ {⟨S, P⟩}
    if |H| < k
      for each h ∈ Succs(Root, P, S)
        Enqueue(q, h)
  return H

function Succs(Root, P, S)
  // Returns the set of modifications
  // of the hypothesis S
  H ← ∅
  for each subcategory C_i of Root
    if C_i ∈ S
        S' ← S \ {C_i}
        P' ← P · (1 − p(C_i))/p(C_i)
    else
        S' ← S ∪ {C_i}
        P' ← P · p(C_i)/(1 − p(C_i))
    H ← H ∪ {⟨S', P'⟩}
    return H
```

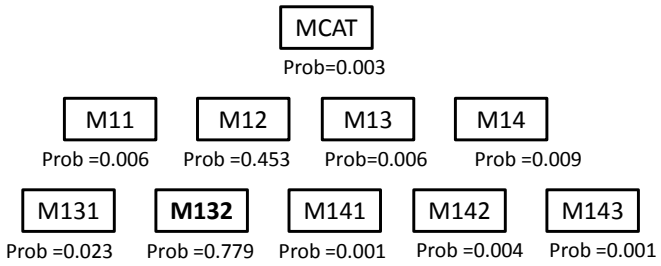

Figure 4.1: Flat hypothesis $\tilde{\mathbf{h}}_1$

$\mathcal{P}(\tilde{\mathbf{h}}_2) = (1-0.003)*(1-0.006)*\mathbf{0.453}*(1-0.006)*(1-0.009)*(1-0.023)*\mathbf{0.779}*(1-0.001)*(1-0.004)*(1-0.001) = 0.33455.$

- $\tilde{\mathbf{h}}_3$: Based on $\tilde{\mathbf{h}}_1$ and $\tilde{\mathbf{h}}_2$, we get the hypothesis $\tilde{\mathbf{h}}_3$ by selecting M12 as the only positive category, who has the third largest joint probability:
  $\mathcal{P}(\tilde{\mathbf{h}}_3) = (1-0.003)*(1-0.006)*\mathbf{0.453}*(1-0.006)*(1-0.009)*(1-0.023)*(1-0.779)*(1-0.001)*(1-0.004)*(1-0.001) = 0.0949.$

- $\tilde{\mathbf{h}}_4$: In the same way, we obtain the fourth hypothesis $\tilde{\mathbf{h}}_4$ in Figure 4.4 based on the previous three hypotheses. Of course, $\tilde{\mathbf{h}}_4$ has the fourth largest probability among all changed hypotheses:

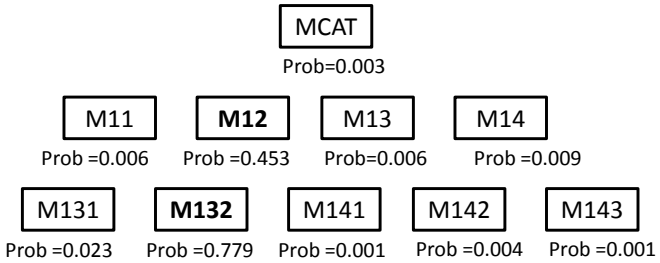

Figure 4.2: Flat hypothesis $\tilde{\mathbf{h}}_2$

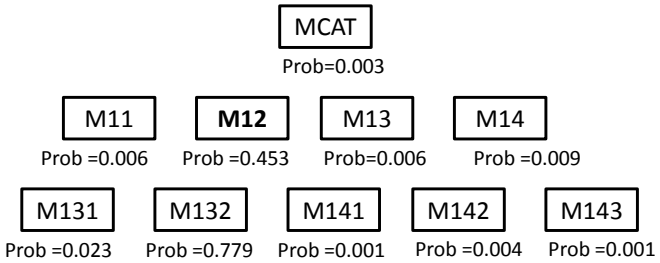

Figure 4.3: Flat hypothesis $\tilde{\mathbf{h}}_3$

$\mathcal{P}(\tilde{\mathbf{h}}_4) = (1 - 0.003) * (1 - 0.006) * (1 - 0.453) * (1 - 0.006) * (1 - 0.009) * \mathbf{0.023} * \mathbf{0.779} * (1 - 0.001) * (1 - 0.004) * (1 - 0.001) = 0.00951.$

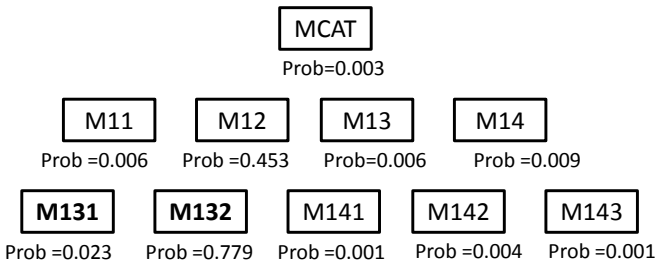

Figure 4.4: Flat hypothesis $\tilde{\mathbf{h}}_4$

It is worth noting that the fourth hypothesis $\tilde{\mathbf{h}}_4$ is actually the gold standard annotation of the example $\boldsymbol{x}_i$. This further shows that the basic SVM output is not always correct, and our flat hypothesis generation algorithm can produce better classification outputs (actually how to choose the best one as the final output is the key point of our thesis).

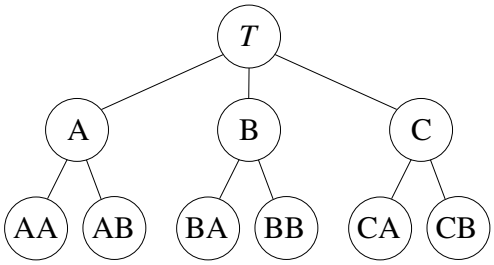

Figure 4.5: Example of a hierarchy.

The flat hypotheses generation algorithm ignores the correlations of categories, resulting in the violation of *hierarchical constraint*: once an example belongs to some categories, it should belong to all its ancestor categories. This is why in Figure 4.4 for $\tilde{\mathbf{h}}_4$, M131, M132 are marked as positive while M13 and MCAT are marked as negative, since in the flat case, we just explore the local information for each independent category. This will be solved by representing these hypotheses with a hierarchy, encoding the global dependencies (see Section 4.2).

4.1.2 Hierarchical Hypothesis Generation

Flat hypothesis generation is based on the flat one-vs.-all binary models by ignoring the correlations and interdependencies provided by the hierarchy. It lacks the inherit information inside the hierarchical structure. To make up this defect, we show a *hierarchical* algorithm in which the structure imposes constrains during the generation of the hypotheses.

The generation process becomes more complex when taking into account the *hierarchical constraint:* 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 ultilize the conventional top-down method [109] on hierarchical text classification. Besides a local classifier for each category, there is one more subtree classifier for the non-leaf categories. The former decides whether the classification showed a "stop" at current category while the latter chooses which sub-hierarchy to go through. Based on these two kinds of classifiers, we make 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:

$$p_s(C) = P(d \notin C_1 \ldots d \notin C_n | d \in C) \tag{4.9}$$

Secondly, in the case where we know that at least one subcategory has been selected, we can compute the *sub-probabilities* of selecting a particular subcategory:

$$p_{C_i}(C) = P(d \in C_i | d \in C(d \in C_1 \vee \ldots \vee d \in C_n)), i \in \{1, \ldots, n\} \tag{4.10}$$

At this stage, we assume conditional independence between the subcategories, so the probability will depend only on the document and the supercategory. 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 4.5. 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:

$$(1 - p_s(T)) \cdot p_A(T) \cdot (1 - p_B(T)) \cdot p_C(T) \cdot (1 - p_s(A)) \cdot (1 - p_{AA}(A)) \cdot p_{AB}(A) \cdot p_s(C) \tag{4.11}$$

.

The next section details our algorithm for hypothesis generation, which exploits this decomposition.

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 presented in [49], our algorithm proceeds in two steps: We first find the best assignment, and then we construct the k-best list by incremental modifications.

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 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 4 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

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

Algorithm 4 Generation of the top hierarchical hypothesis.

```
function Top1(C)
  // Returns the top hypothesis
  // and its probability
  if p_s(C) ≥ 0.5
    return ⟨{C}, p_s(C)⟩
  ⟨S, P⟩ ← MaxSubcats(C)
  if S = ∅
    ⟨S, P⟩ ← MaxOneSubcat(C, P)
  if p_s(C) ≥ P
    return ⟨{C}, p_s(C)⟩
  else
    return ⟨{C} ∪ S, P⟩

function MaxOneSubcat(C, P)
  q_min ← ∞
  for each subcategory C_i ∈ Sub(C)
    ⟨S_i, P_i⟩ ← Top1(C_i)
    q_i ← (1 − p_{C_i}(C))/(P_i · p_{C_i}(C))
    if q_i < q_min
      q_min ← q_i, S_min ← S_i
  return ⟨S_min, P/q_min⟩

function MaxSubcats(C)
  S ← ∅, P ← 1 − p_s(C)
  for each subcategory C_i ∈ Sub(C)
    if p_{C_i}(C) ≤ 0.5
      P ← P · (1 − p_{C_i}(C))
    else
      ⟨S_i, P_i⟩ ← Top1(C_i)
      if p_{C_i}(C) · P_i > (1 − p_{C_i}(C))
        P ← P · p_{C_i}(C) · P_i
        S ← S ∪ S_i
      else
        P ← P · (1 − p_{C_i}(C))
  return ⟨S, P⟩
```

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.

The algorithm TopK to generate the k top hypotheses (Algorithm 5) 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;

Algorithm 5 Generation of the top k hierarchical hypotheses.

```
function TopK(C, k)
  // Returns the top k hypotheses
  // and their probabilities
  H ← ∅
  q ← empty priority queue
  Enqueue(q, Top1(C))
  while |H| < k and q is nonempty
    ⟨S, P⟩ ← Dequeue(q)
    H ← H ∪ {⟨S, P⟩}
    if |H| < k
      for each h ∈ 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 ∅
  H ← ∅
  if S ≠ {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
```

- 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 6. 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.

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.

Algorithm 6 One-step modifications of a hypothesis.

procedure SUBCATSUCCS(C, P, S, H)
  for each subcategory $C_i \in \text{SUB}(C)$
   if $C_i \in S$
    $P_i \leftarrow \text{PROBSUBCATS}(S, C_i)$
    $S_i \leftarrow S \cap \text{SUBTREE}(C_i)$
    for each $\langle S_s, P_s \rangle \in \text{SUCCS}(C_i, P_i, S_i)$
     $H \leftarrow H \cup \{\langle (S \setminus S_i) \cup S_s, P/P_i \cdot P_s \rangle\}$

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

procedure UNSTOP(C, P, S, H)
  $\langle S_s, P_s \rangle \leftarrow \text{MAXSUBCATS}(C)$
  if $S_s = \emptyset$
   $\langle S_s, P_s \rangle \leftarrow \text{MAXONESUBCAT}(C, P)$
  $P' \leftarrow P \cdot (1 - p_s(C)) \cdot P_s/p_s(C)$
  $H \leftarrow H \cup \{\langle S \cup S_s, P' \rangle\}$

procedure ENABLEEACHSUBCAT(C, P, S, H)
 for each subcategory $C_i \in \text{SUB}(C)$
  if $C_i \notin S$
   $\langle S_i, P_i \rangle \leftarrow \text{TOP1}(C_i)$
   $P' \leftarrow P \cdot p_{C_i}(C) \cdot P_i/(1 - p_{C_i}(C))$
   $H \leftarrow H \cup \{\langle S \cup S_i, P' \rangle\}$

procedure DISABLEEACHSUBCAT(C, P, S, H)
 for each subcategory $C_i \in \text{SUB}(C)$
  if $C_i \in S$
   $P' \leftarrow P \cdot (1 - p_{C_i}(C))$
   $P' \leftarrow P'/p_{C_i}(C)/\text{PROBSUBCATS}(S, C_i)$
   $S' \leftarrow S \setminus \text{SUBTREE}(C_i)$
   if $S' \neq \{C\}$
    $H \leftarrow H \cup \{\langle S', P' \rangle\}$
   else if $P' \geq P$
    ENABLEEACHSUBCAT(C, P', S', H)

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.

The hierarchical hypothesis generation algorithm is indeed a little bit more complicated. It considers the category interdependencies in the hierarchy. To make it clear how it works, we use some figures, which illustrate the generation process. For the example we use as an example, $\boldsymbol{x}_i$, which has the manual label annotation of MCAT and M14 on

MCAT subhierarchy. We describe the hypothesis generations with $k = 8$ as follows:

- $\tilde{\mathbf{h}_1}$: Figure 4.6 is the basic SVM output with MCAT, M14, and M143 predicted positive for $\boldsymbol{x}_i$, the *Sub_Pro* for a category means probability that $\boldsymbol{x}_i$ belongs to the tree rooted by this category and *Stop_Pro* is the probability that $\boldsymbol{x}_i$ belongs to the current category. According to algorithm 4 and Equation 4.11, $\mathbf{h}_1$ has the largest joint probability:
$\mathcal{P}(\tilde{\mathbf{h}_1}) = \frac{1.0*(1-0.011)}{MCAT} * \frac{(1-0.0036)}{M11} * \frac{(1-0.0037)}{M12} * \frac{(1-0.0)}{M13} * \frac{0.999*(1-0.015)}{M14} * \frac{0.995}{M141} * \frac{(1-0.0)}{M142} * \frac{(1-0.012)}{M143} = 0.9497$, where the upper part for each fraction element represent the decision probability of the category labeled by the lower part.

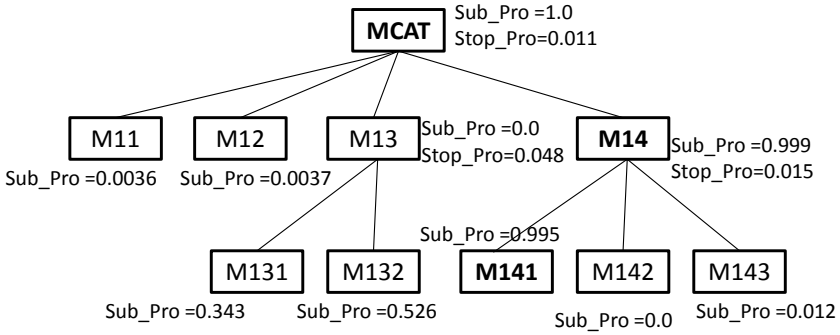

Figure 4.6: Hierarchical hypothesis $\tilde{\mathbf{h}_1}$

- $\tilde{\mathbf{h}_2}$: following one-step modification operations STOP to stop at subcategory M14, and get the second hierarchical hypothesis $\tilde{\mathbf{h}_2}$ in Figure 4.7 with the second joint probability:
$\mathcal{P}(\tilde{\mathbf{h}_2}) = \frac{1.0*(1-0.011)}{MCAT} * \frac{(1-0.0036)}{M11} * \frac{(1-0.0037)}{M12} * \frac{(1-0.0)}{M13} * \frac{0.999*0.015}{M14} = 0.0147.$

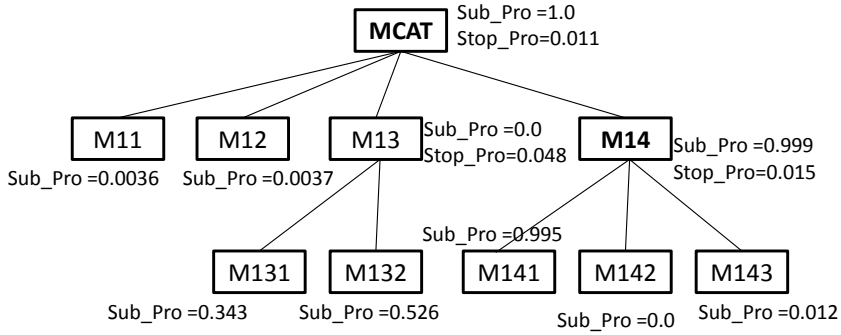

Figure 4.7: Hierarchical hypothesis $\tilde{\mathbf{h}_2}$

- $\tilde{\mathbf{h}_3}$: following one-step modification operations ENABLEEACHSUBCAT to change the assignment with subcategory M143 based on the first hypothesis $\tilde{\mathbf{h}_1}$, and get the

third hierarchical hypothesis $\tilde{\mathbf{h}_3}$ in Figure 4.8 with the second joint probability: $\mathcal{P}(\tilde{\mathbf{h}_3}) = \frac{1.0*(1-0.011)}{MCAT} * \frac{(1-0.0036)}{M11} * \frac{(1-0.0037)}{M12} * \frac{(1-0.0)}{M13} * \frac{0.999*(1-0.015)}{M14} * \frac{0.995}{M141} * \frac{(1-0.0)}{M142} * \frac{0.012}{M143} = 0.0111$.

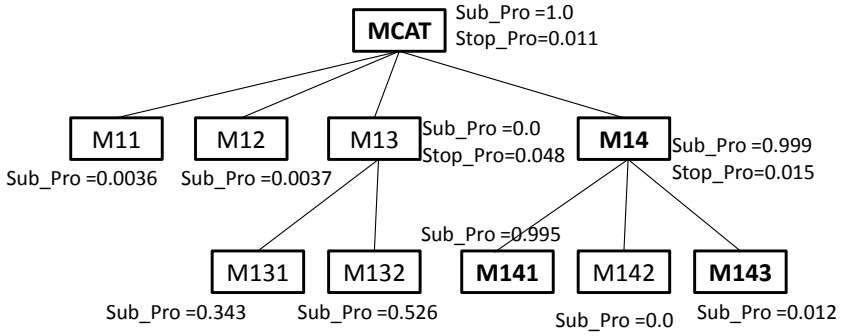

Figure 4.8: Hierarchical hypothesis $\tilde{\mathbf{h}_3}$

- $\tilde{\mathbf{h}_4}$: following one-step modification operations STOP to change the assignment with subcategory M14 based on the second hypothesis $\tilde{\mathbf{h}_2}$, and get the fourth hierarchical hypothesis $\tilde{\mathbf{h}_4}$ in Figure 4.9 with the second joint probability: $\mathcal{P}(\tilde{\mathbf{h}_4}) = \frac{1.0*0.011}{MCAT} = 0.011$.

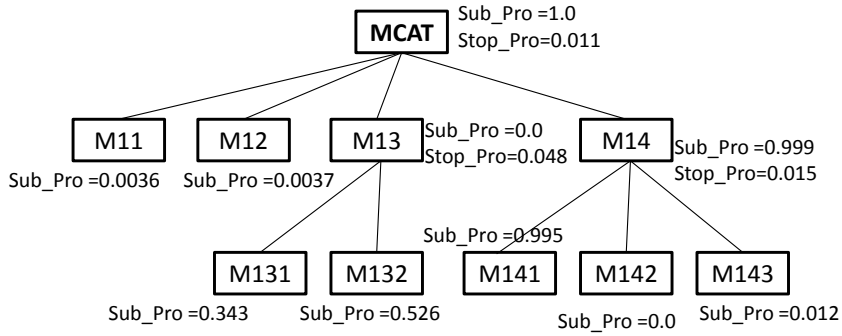

Figure 4.9: Hierarchical hypothesis $\tilde{\mathbf{h}_4}$

- $\tilde{\mathbf{h}_5}$: following one-step modification operations ENABLEEACHSUBCAT to change the assignment with subcategory M12 based on the first hypothesis $\tilde{\mathbf{h}_1}$, and get the fifth hierarchical hypothesis $\tilde{\mathbf{h}_5}$ in Figure 4.10 with the second joint probability: $\mathcal{P}(\tilde{\mathbf{h}_5}) = \frac{1.0*(1-0.011)}{MCAT} * \frac{(1-0.0036)}{M11} * \frac{0.0037}{M12} * \frac{(1-0.0)}{M13} * \frac{0.999*(1-0.015)}{M14} * \frac{0.995}{M141} * \frac{(1-0.0)}{M142} * \frac{(1-0.012)}{M143} = 0.00353$.

- $\tilde{\mathbf{h}_6}$: following one-step modification operations ENABLEEACHSUBCAT to change the assignment with subcategory M11 based on the first hypothesis $\tilde{\mathbf{h}_1}$, and get the sixth

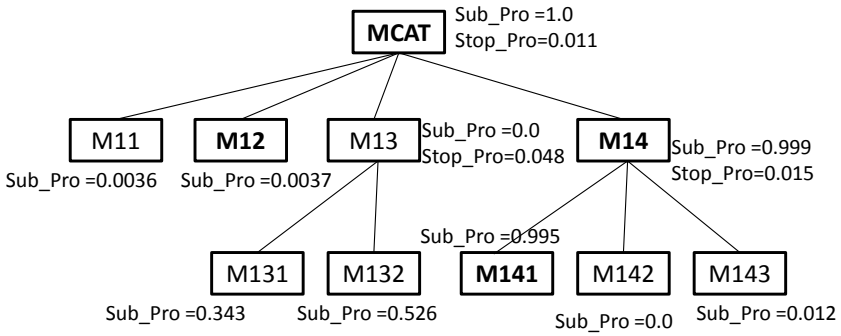

Figure 4.10: Hierarchical hypothesis $\tilde{\mathbf{h}_5}$

hierarchical hypothesis $\tilde{\mathbf{h}_6}$ in Figure 4.11 with the second joint probability: $\mathcal{P}(\tilde{\mathbf{h}_6}) = \frac{1.0*(1-0.011)}{MCAT} * \frac{0.0036}{M11} * \frac{(1-0.0037)}{M12} * \frac{(1-0.0)}{M13} * \frac{0.999*(1-0.015)}{M14} * \frac{0.995}{M141} * \frac{(1-0.0)}{M142} * \frac{(1-0.012)}{M143} = 0.00343$.

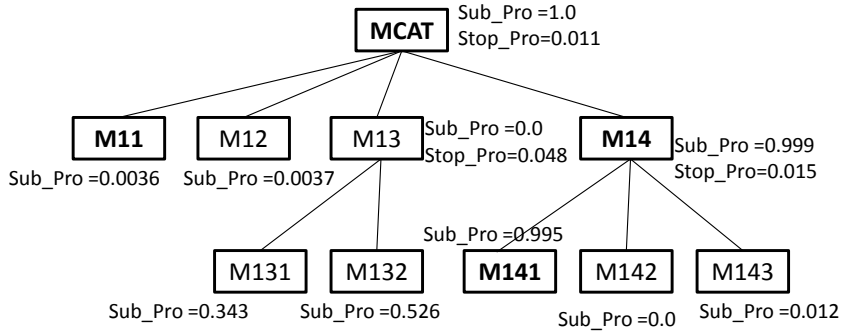

Figure 4.11: Hierarchical hypothesis $\tilde{\mathbf{h}_6}$

- $\tilde{\mathbf{h}_7}$: following one-step modification operations SubcatSuccs to change the assignment with subcategory M143 based on the second hypothesis $\tilde{\mathbf{h}_2}$, and get the seventh hierarchical hypothesis $\tilde{\mathbf{h}_7}$ in Figure 4.12 with the second joint probability: $\mathcal{P}(\tilde{\mathbf{h}_7}) = \frac{1.0*(1-0.011)}{MCAT} * \frac{(1-0.0036)}{M11} * \frac{(1-0.0037)}{M12} * \frac{(1-0.0)}{M13} * \frac{0.999*(1-0.015)}{M14} * \frac{(1-0.995)}{M141} * \frac{(1-0.0)}{M142} * \frac{0.012}{M143} = 0.000058$.

- $\tilde{\mathbf{h}_8}$: following one-step modification operations Stop to change the assignment with subcategory M14 based on the fifth hypothesis $\tilde{\mathbf{h}_5}$, and get the eighth hierarchical hypothesis $\tilde{\mathbf{h}_8}$ in Figure 4.13 with the eighth joint probability: $\mathcal{P}(\tilde{\mathbf{h}_8}) = \frac{1.0*(1-0.011)}{MCAT} * \frac{(1-0.0036)}{M11} * \frac{0.0037}{M12} * \frac{(1-0.0)}{M13} * \frac{0.999*0.015}{M14} = 0.00005469$.

We can clearly see from the generated hypotheses that some of them have better results than the original classifier output, e.g., $\tilde{\mathbf{h}_2}$ is exactly the manual annotation for $\boldsymbol{x}_i$.

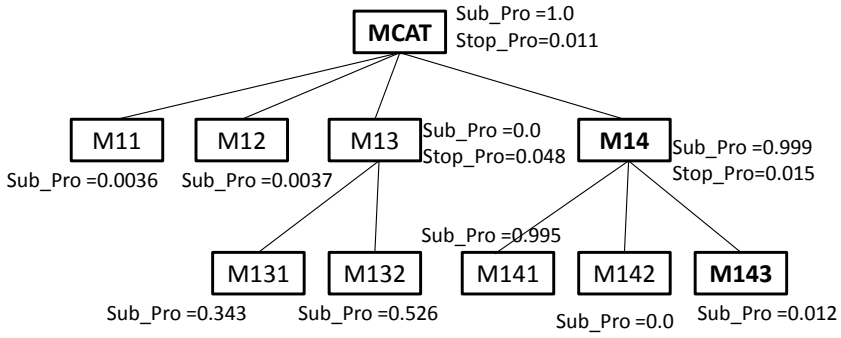

Figure 4.12: Hierarchical hypothesis $\tilde{\mathbf{h}}_7$

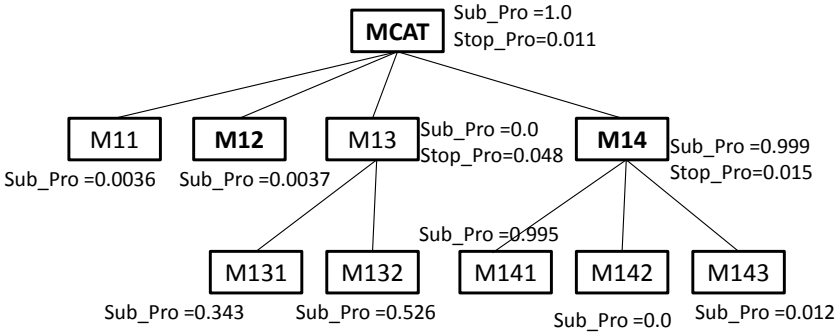

Figure 4.13: Hierarchical hypothesis $\tilde{\mathbf{h}}_8$

4.2 Encoding Hypotheses in a Tree

4.2.1 Hypotheses in Global Tree

Once hypotheses are generated, we need a representation from which the dependencies between the different nodes of the hierarchy can be learned. Since we do not know in advance which are the important dependencies and not even the scope of the interaction between the different structure subparts, we rely on automatic feature engineering via structural kernels. For this paper, we consider tree-shaped hierarchies so that tree kernels, e.g. [24], can be applied. As an example let us consider the Reuters categorization scheme. Figure 4.14 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 a hierarchy and the dependencies between its 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

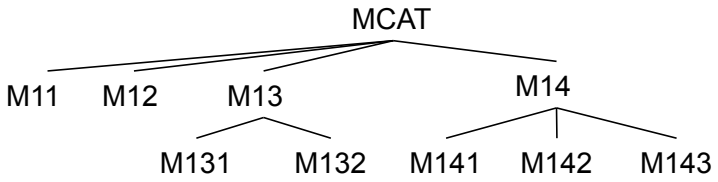

Figure 4.14: A subhierarchy of Reuters.

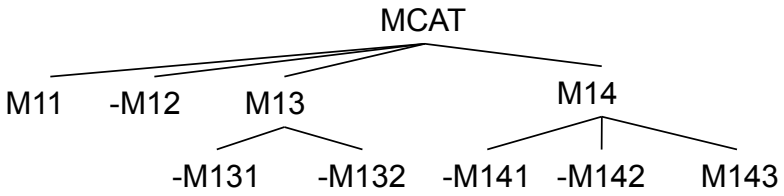

Figure 4.15: A tree representing a category assignment hypothesis for the subhierarchy in Figure 4.14.

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 start exploring some reasonable features provided for other structured output tasks. For example, the path between nodes in semantic role labeling [17, 110] or trigrams and bigrams in parse-tree reranking [20].

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). Indeed, tree kernels have been successfully used in many other NLP tasks, e.g., [60, 30, 15, 29, 32, 113, 61, 111, 81, 37], although no previous work has used them to address hierarchy reranking (parse tree reranking is a rather different task [24]).

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 4.15 shows the representation of a classification hypothesis consisting in assigning the target document to the categories MCAT, M11, M13, M14 and M143.

4.2.2 Hypotheses in Compact Tree

Another more compact representation is the hierarchy tree from which all the nodes associated with a negative classification decision are removed. As only a small subset of nodes of the full hierarchy will be positively classified, the tree will be much smaller.

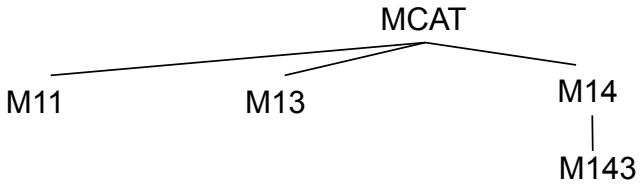

Figure 4.16: A compact representation of the hypothesis in Figure 4.15.

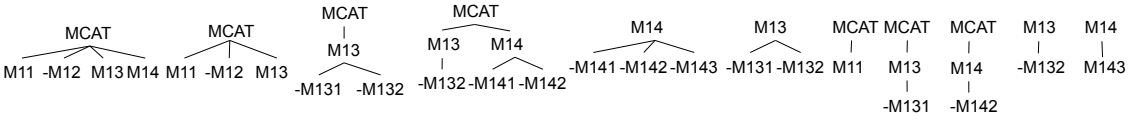

Figure 4.17: Some tree fragments of the hypothesis in Figure 4.15

Figure 4.16 shows the compact representation of the hypothesis in Fig. 4.15. By applying the partial tree kernel (PTK) [78] to such labeled tree all possible dependency features are generated. For example, Fig. 4.17 shows some of the tree fragments from the hypothesis of Fig. 4.15.

4.3 Structural Reranking

4.3.1 Preference Reranker

Up to now the k-best hierarchical hypotheses encode the global dependencies by the hierarchy itself as well as local features of the basic binary model. This poses the problem of how to use such information in a discriminative machine learning algorithm for reranking the k hypotheses. The reranking machine learning problem in this thesis consists in learning to select the best candidate hypothesis from a given candidate set generated from basic SVM models.

In order to be able to apply machine learning methods for binary classifiers such as support vector learning, we applied the reduction known as the Preference Kernel method [106]. The development of reduction methods from ranking tasks to binary classification is an active research area; see for instance [3] and [1].

In the Preference Kernel approach, the reranking problem – learning to pick the best candidate $\tilde{\mathbf{h}_1}$ with the highest Micro-F1 (e.g., $\tilde{\mathbf{h}_4}$ in flat hypotheses generation example and $\tilde{\mathbf{h}_2}$ in hierarchical hypotheses generation example) from a candidate set $\{\tilde{\mathbf{h}_1}, \ldots, \tilde{\mathbf{h}_k}\}$ – is reduced to a binary classification problem by creating *pairs*: positive training instances $\langle \tilde{\mathbf{h}_1}, \tilde{\mathbf{h}_2} \rangle, \ldots, \langle \tilde{\mathbf{h}_1}, \tilde{\mathbf{h}_k} \rangle$ and negative instances $\langle \tilde{\mathbf{h}_2}, \tilde{\mathbf{h}_1} \rangle, \ldots, \langle \tilde{\mathbf{h}_k}, \tilde{\mathbf{h}_1} \rangle$. This training set can then be used to train a binary classifier. At classification time, pairs are not formed (since

the correct candidate is not known); instead, the standard one-versus-all binarization method is still applied.

The kernels are then engineered to implicitly represent the *differences* between the objects in the pairs. If we have a valid kernel $\mathcal{K}$ over the candidate space T, we can construct a function $D_{\mathcal{K}}$ over the space of pairs $T \times T$ as follows:

$$\begin{aligned} D_{\mathcal{K}}(\boldsymbol{x}, \boldsymbol{y}) &= D_{\mathcal{K}}(\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle, \langle \boldsymbol{y}_1, \boldsymbol{y}_2 \rangle) \\ &= \mathcal{K}(\boldsymbol{x}_1, \boldsymbol{y}_1) + \mathcal{K}(\boldsymbol{x}_2, \boldsymbol{y}_2) \\ &- \mathcal{K}(\boldsymbol{x}_1, \boldsymbol{y}_2) - \mathcal{K}(\boldsymbol{x}_2, \boldsymbol{y}_1). \end{aligned}$$

It is easy to show [106] that $D_{\mathcal{K}}$ is also a valid Mercer kernel. This makes it possible to use kernel methods to train the reranker.

In this thesis, each candidate $\tilde{\mathbf{h}_i}$ can be described by a structural hypothesis and a vector of linear feature vector $\boldsymbol{v}_i$ representing information that cannot be capture by $\tilde{\mathbf{h}_i}$, e.g., the joint probability of hypothesis, or the individual category probability for each category in the hypothesis. As a whole, each classifier example $\boldsymbol{e}_i$ is expressed by a combination of structural hypothesis pairs $\langle \tilde{\mathbf{h}_i}^1, \tilde{\mathbf{h}_i}^2 \rangle$ and feature vector pairs $\langle \boldsymbol{x}_i^1, \boldsymbol{x}_i^2 \rangle$ associated with the hypotheses. Then we can define the following kernels:

$$\mathcal{K}_{tr}(\boldsymbol{e}_i, \boldsymbol{e}_j) = \mathcal{K}_t(\tilde{\mathbf{h}_i}^1, \tilde{\mathbf{h}_j}^1) + \mathcal{K}_t(\tilde{\mathbf{h}_i}^2, \tilde{\mathbf{h}_j}^2) - \mathcal{K}_t(\tilde{\mathbf{h}_i}^1, \tilde{\mathbf{h}_j}^2) - \mathcal{K}_t(\tilde{\mathbf{h}_i}^2, \tilde{\mathbf{h}_j}^1) \tag{4.12}$$

$$\mathcal{K}_{pr}(\boldsymbol{e}_i, \boldsymbol{e}_j) = \mathcal{K}_p(\boldsymbol{x}_i^1, \boldsymbol{x}_j^1) + \mathcal{K}_p(\boldsymbol{x}_i^2, \boldsymbol{x}_j^2) - \mathcal{K}_p(\boldsymbol{x}_i^1, \boldsymbol{x}_j^2) - \mathcal{K}_p(\boldsymbol{x}_i^2, \boldsymbol{x}_j^1) \tag{4.13}$$

where $\mathcal{K}_{tr}$ and $\mathcal{K}_{pr}$ are kernels like $D_{\mathcal{K}}$, and $\mathcal{K}_t$ is a tree kernel function such as ST, SST, and PTK kernels described in Section 2.2.2, and $\mathcal{K}_p$ is a polynomial kernel applied to the feature vectors. The final kernel that we use for re-ranking is the following:

$$\mathcal{K}(\boldsymbol{e}_i, \boldsymbol{e}_j) = \mathcal{K}_{tr}(\boldsymbol{e}_i, \boldsymbol{e}_j) + \mathcal{K}_{pr}(\boldsymbol{e}_i, \boldsymbol{e}_j) \tag{4.14}$$

We explore innovative kernels $\mathcal{K}$ to be used in Equation 4.14:

$$\mathcal{K}_J = S + p(x_i) \times p(x_j) \tag{4.15}$$

where $p(\cdot)$ is the global joint probability of a target hypothesis and S is a structural kernel, i.e., SK, STK and PTK.

$$\mathcal{K}_P = S + \boldsymbol{p}_i \cdot \boldsymbol{p}_j \tag{4.16}$$

where $\boldsymbol{p}_i^l$ and $\boldsymbol{p}_j^l$ are the classification probabilities of the node (category) l in the trees (hypotheses) $\tilde{\mathbf{h}_i}$ and $\tilde{\mathbf{h}_j}$, respectively and S is again a structural kernel, i.e., SK, STK and PTK.

4.3.2 Reranking Models

We implemented the following five rerankers:

- Flat Reranker (FRR): generates the hypotheses with the flat hypothesis generation algorithm explained in Section 4.1.1, and uses the representation of hypotheses with the global tree described in Section 4.2.1.
- Fast Flat Reranker (FFRR): generates the hypotheses with the flat hypothesis generation algorithm explained in Section 4.1.1, and uses the representation of hypotheses with the compact tree described in Section 4.2.2.
- Hierarchical Reranker (HRR): generates the hypotheses with the hierarchical hypothesis generation algorithm explained in Section 4.1.2, and uses the representation of hypotheses with the compact tree described in Section 4.2.2.
- Fast Hierarchical Reranker (FHRR): generates the hypotheses with the hierarchical hypothesis generation algorithm explained in Section 4.1.2, and uses the representation of hypotheses with the compact tree described in Section 4.2.2.
- Sequence Reranker (SeqRR): generates the hypotheses with the flat hypotheses generation algorithm explained in Section 4.1.1, and uses the representation of hypotheses that all categories are put in one line according to breadth-first strategy on the hierarchy.

For comparative purposes, we also use for S a linear kernel over the bag-of-label[2] (BOL). This is supposed to capture non-structural dependencies between the category labels.

4.3.3 Reranking System

Figure 4.18 illustrates the whole reranking system, which comprises two parts: reranker learning and hypothesis reranking. In the learning phase, we divided the training set in two chunks of data: Train1 and Train2. The binary SVM 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.

In the classification phase, the binary classifiers are built on the training set to generate the hypotheses on test set. These hypotheses then will be taken as the test set of the reranker, and finally output the reranker performance. By making the comparison with the baseline, we can derive the improvement produced by the structural dependencies of the hierarchy.

[2]It simply uses the categorized labels as features. It is used in the KJ or KP schemes.

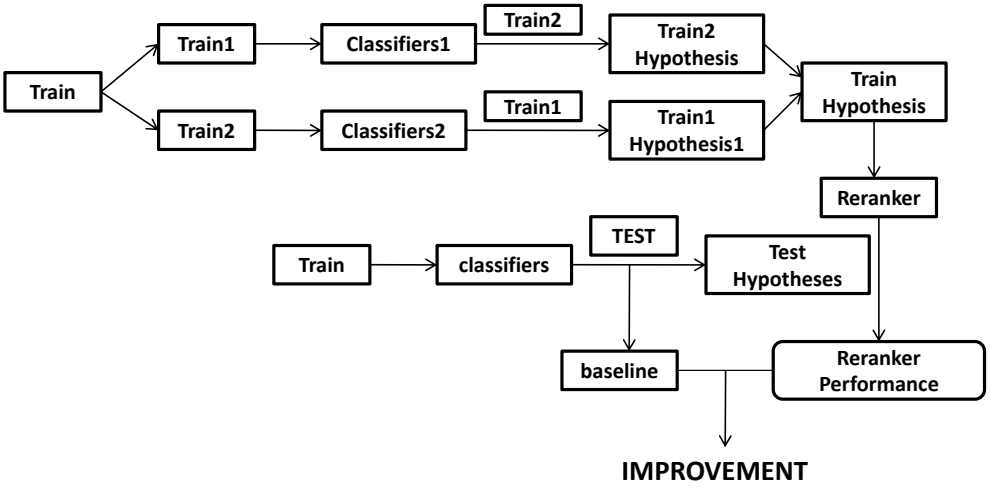

Figure 4.18: Framework of Reranking Model

4.4 Experiments on Flat Reranker

The aim of the experiments is to demonstrate that our reranking approach can introduce semantic dependencies in the hierarchical classification model, which can improve accuracy. For this purpose, we show that several reranking models based on tree kernels improve the classification based on the *flat* one-vs.-all approach. Then, we analyze the efficiency of our models, demonstrating their applicability.

4.4.1 Setup

We used two full hierarchies, *Topics* and *Industries* of Reuters Corpus Volume 1 (RCV1)[3] TC corpus and carried out some experiments on them with the entire documents from RCV1.

For most experiments, we randomly selected two subsets of 10k and 5k of documents for training and testing from the total 804,414 Reuters news from *Topics* by still using all the 103 categories organized in 5 levels (hereafter SAM). The distribution of the data instances of some of the different categories in such samples can be observed in Table 4.1. Note that the most populated categories are at the top, the medium sized ones follow and the smallest ones are at the bottom. There are some differences between the child-free and child-full setting since for the former, from each node, we removed all the documents from its children. The training set is used for learning the binary classifiers needed to build the multi-class classifier (MCC). We used the datasets with two different settings:

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

we removed from the node-fathers all the documents belonging to their children. We call *child-free* this modality to contrast the normal setting that we call *child-full*.

Category	Child-free				Child-full			
	Train	Train1	Train2	TEST	Train	Train1	Train2	TEST
C152	837	370	467	438	837	370	467	438
GPOL	723	357	366	380	723	357	366	380
M11	604	309	205	311	604	309	205	311
..	..	..	..	..	..	..	..	..
C31	313	163	150	179	531	274	257	284
E41	191	89	95	102	223	121	102	118
GCAT	345	177	168	173	3293	1687	1506	1600
..	..	..	..	..	..	..	..	..
E31	11	4	7	6	32	21	11	19
M14	96	49	47	58	1175	594	581	604
G15	5	4	1	0	290	137	153	146
Total: 103	10,000	5,000	5,000	5,000	10,000	5,000	5,000	5,000

Table 4.1: Instance distributions of RCV1.

We implemented the baseline model in Chapter 3, and the classifiers are combined using the one-vs.-all approach, which is also state-of-the-art as argued in [92]. 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 implemented two rerankers: FRR and FFRR described in Section 4.3.2.

The rerankers are based on SVMs (proLIBSVM) and the Preference Kernel (P_K) described in Section 4.3.1 built on top of SK, STK or PTK (see Section 2.2.2). These are applied to the tree-structured hypotheses. We trained the rerankers using SVM-light-TK[4], a tree-kernel-enabled version of SVM-light [52], which allows for using structural kernel on pairs of trees and combining them with vector-based kernels. Again we use default parameters to facilitate replicability and preserve generality. The rerankers always use 8 best hypotheses.

All the performances are provided by means of Micro- and Macro-Average F1, evaluated on test data over all categories (103 or 363). Additionally, the F1 of some binary classifiers is reported.

Finally, we assessed the statistical significance of our results by using the model described in [134] and implemented in [84].

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

4.4.2 Classification Accuracy

To definitely assess the benefit of our rerankers we first tested them on the Lewis' split of two different datasets of RCV1, i.e., *Topics* and *Industries*. Table 4.2 shows the comparison between flat rerankers (FRR) using STK or BOL (when indicated) with K_J and K_P schema, and 32,000 examples are used for training the rerankers. BL (Lewis) and BL (Ours) in Table 4.2 mean the baseline of Lewis' model on RCV1-v2 and our baseline on RCV1-v3. Table 4.2 gives impressive results, e.g., for *Topic*, there is 3.3 percent points more than the state-of-the-art flat TC models, and for INDUSTRY, the improvement is up to 5.2 percent points. We carried out statistical significance tests, which certified the significance at 99%. This was expected as the size of Lewis' test sets in the order of several hundred thousands.

F1	Topics				Industries			
	BL (Lewis)	BL (Ours)	K_J	K_P	BL (Lewis)	BL (Ours)	K_J	K_P
Micro-F1	0.816	0.816	0.827	0.849	0.512	0.562	0.576	0.628
Macro-F1	0.567	0.566	0.590	0.615	0.263	0.289	0.314	0.341

Table 4.2: Comparison between flat rerankers using STK.

Next we compared the different kernels using the K_J combination (which exploits the joint hypothesis probability, see Section 4.1.1 and Section 4.1.2) on SAM. Table 4.3 shows that the baseline (state-of-the-art flat model) is largely improved by all rerankers trained by only 8k training examples. However, the reranker based on BOL (it cannot capture the same dependencies as the structural kernels) seems to provide the same performance as the baseline. In contrast, when we remove the dependencies generated by shared documents between a node and its descendants (child-free setting), Table 4.4 shows tree kernels (STK and PTK) significantly improve on BOL only with 8k training examples. SK delivers an improvement similar to BOL, suggesting that sequential features are not enough.

$F1$	BL	BOL	SK	STK	PTK
Micro-F_1	0.769	0.787	0.786	0.790	0.790
Macro-F_1	0.539	0.541	0.546	0.547	0.560

Table 4.3: Comparison of rerankers using different kernels with child-full setting.

$F1$	BL	BOL	SK	STK	PTK
Micro-F_1	0.640	0.657	0.653	0.677	0.682
Macro-F_1	0.408	0.436	0.431	0.447	0.447

Table 4.4: Comparison of rerankers using different kernels with child-free setting.

To study how much data is needed for the reranker, Figures 4.19 and 4.20 report the Micro-and Macro-Average F1 of our rerankers over 103 categories, according to different sets of increasing training data under the child-free setting. This time, K_J is applied to only STK. We note that (i) few thousands of training examples are enough to deliver most of the RR improvement; and (ii) the FFRR produces similar results to standard FRR. This is very interesting since, as it will be shown in the next section, the compact representation produces much faster models.

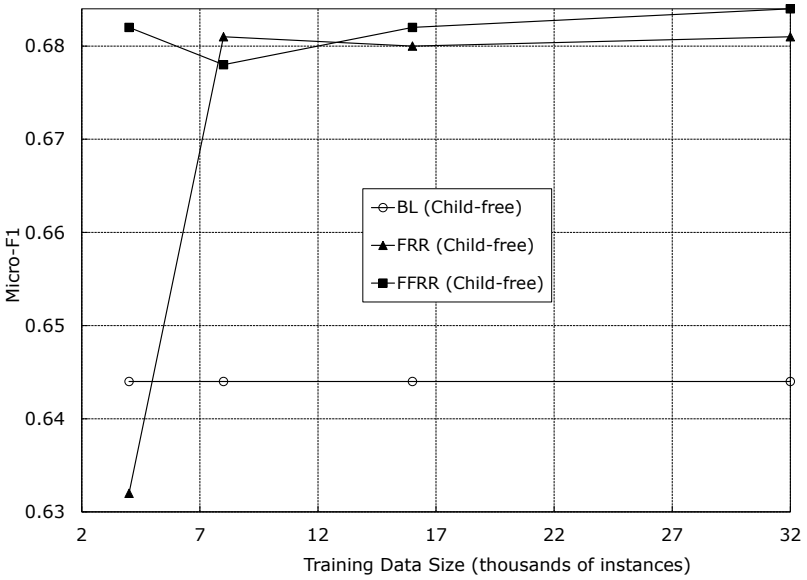

Figure 4.19: Learning curves of the reranking models using STK in terms of MicroAverage-F1.

Table 4.5 reports the F1 of some individual categories as well as global performance over all 103 categories of RCV1, 8 hypotheses and 32k of training data for rerankers using STK. In these experiments we used STK in K_J and K_P. We note that K_P highly improves on the baseline on child-free setting by about 7.1 and 9.9 absolute percent points in Micro-and Macro-F1, respectively. Also the improvement on child-full is meaningful, i.e., 4.6 percent points. This is rather interesting as BOL (not reported in the table) achieved a Micro-average of 80.4% and a Macro-average of 57.2% when used in K_P, i.e., up to 2 points below STK. This means that the use of probability vectors, in combination with structural kernels, is a very promising direction for reranker design.

To better understand the potential of reranking, Table 4.6 shows the oracle performance with respect to the increasing number of hypotheses. The outcome clearly demonstrates that there is large margin of improvement for the rerankers.

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

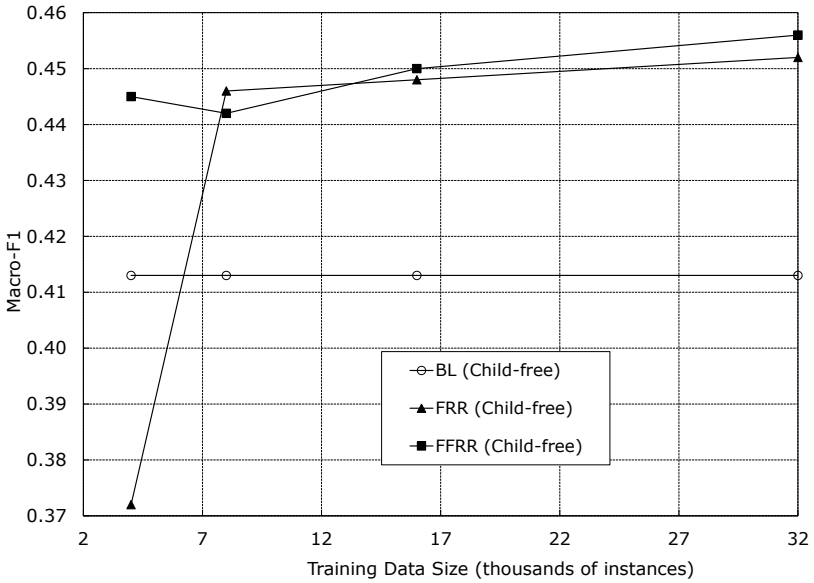

Figure 4.20: Learning curves of the reranking models using STK in terms of MacroAverage-F1.

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 4.6. 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.

4.4.3 Running Time

To study the applicability of our rerankers, we have analyzed both the training and classification time. Figure 4.21 shows the minutes required to train the different models as well as to classify the test set according to data of increasing size.

It can be noted that the models using the compact hypothesis representation are much faster than those using the complete hierarchy as representation, i.e., up to five times in training and eight times in testing. This is not surprising as, in the latter case, each kernel evaluation requires to perform tree kernel evaluation on trees of 103 nodes. When using the compact representation the number of nodes is upper-bounded by the maximum number of labels per documents, i.e., 6, times the depth of the hierarchy, i.e., 5 (the positive classification on the leaves is the worst case). Thus, the largest tree would contain 30 nodes. However, we only have 1.82 labels per document on average, therefore the trees have an average size of only about 9 nodes.

Cat.	Child-free			Child-full		
	BL	K_J	K_P	BL	K_J	K_P
C152	0.671	0.700	0.771	0.671	0.729	0.745
GPOL	0.660	0.695	0.743	0.660	0.680	0.734
M11	0.851	0.891	0.901	0.851	0.886	0.898
..	..	..	..	..	..	..
C31	0.225	0.311	0.446	0.356	0.421	0.526
E41	0.643	0.714	0.719	0.776	0.791	0.806
GCAT	0.896	0.908	0.917	0.908	0.916	0.926
..	..	..	..	..	..	..
E31	0.444	0.600	0.600	0.667	0.765	0.688
M14	0.591	0.600	0.575	0.887	0.897	0.904
G15	0.250	0.222	0.250	0.823	0.806	0.826
103 cat.						
Mi-F1	0.640	0.677	0.731	0.769	0.794	0.815
Ma-F1	0.408	0.447	0.507	0.539	0.567	0.590

Table 4.5: F1 of some binary classifiers along with the Micro and Macro-Average F1.

k	Micro-F_1	Macro-F_1
1	0.640	0.408
2	0.758	0.504
4	0.821	0.566
8	0.858	0.610
16	0.898	0.658

Table 4.6: Oracle performance according to the number of hypotheses (child-free setting).

4.5 Experiments on Hierarchical Reranker

In these experiments, we show that several hierarchical reranking models based on tree kernels can improve the state-of-the-art in TC. Then, we analyze the efficiency of our models, demonstrating their applicability. For this purpose, we used three different datasets: (1) the entire RCV1 (same Lewis' split [67]), which provides a concrete assessment with respect to the state-of-the-art in TC; (2) the same subset of RCV1 of 15k examples as in Section 4.4.1 for carrying out extensive performance analysis, e.g., deriving learning curves; and (3) the same setting of Zhou et al. [135] to compare with state-of-the-art reranking in machine learning.

Finally, we optimized our reranker for MGIE (in Section 3.5.2) and measure the accuracy of different models according to it on the Lewis' split.

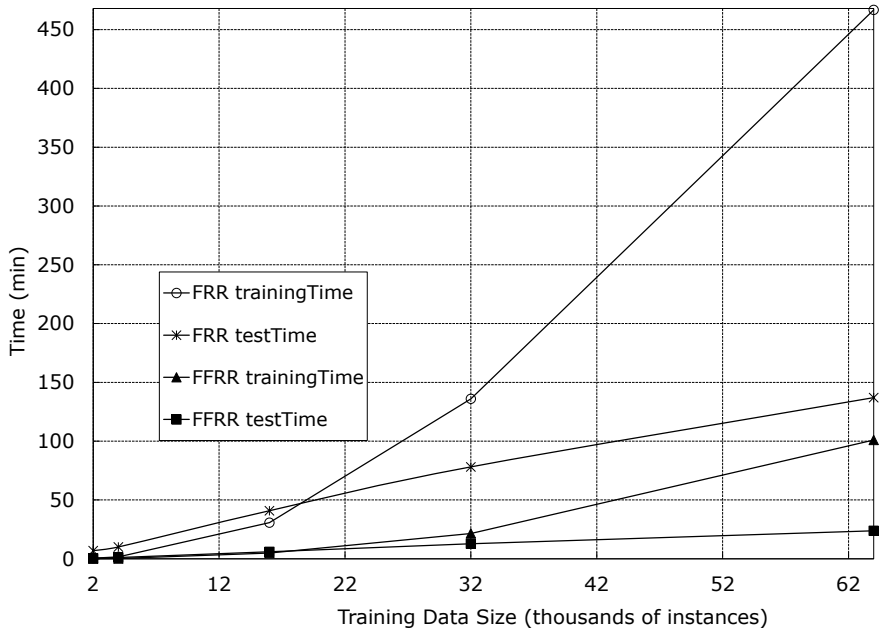

Figure 4.21: Training and test time of the rerankers trained on data of increasing size.

4.5.1 Setup

We also used the full hierarchy of Reuters Volume 1 (RCV1)[5] TC corpus to assess our developed hieararchical rerankers (HRR).

To compare with previous work we considered the Lewis' split [67], which includes 23,149 news for training and 781,265 for testing. We also used the same subsets of 10k training documents and 5k test documents as in Section 4.4.1, and the distribution of the data instances of some of the different categories in the 15k samples can be observed in Table 4.1.

We implemented the top-down model described in Chapter 3, and the subtree classifiers and local classifiers are combined using the one-vs.-all approach, which is also state-of-the-art as argued in [92]. 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 implemented two rerankers: HRR and FHRR described in Section 4.3.2.

The training sets are used for learning the binary local classifiers for the internal categories in the hierarchy and subtree classifiers for all categories. We used the proLIBSVM to build the classifiers with the one-vs.-all strategy. In the classification phase, the classifiers are applied in top-down manner as described in Section 3.4.2, which constitutes our top-down baseline.

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

The rerankers are based on proLIBSVM and the Preference Kernel (P_K) described in Section 4.3.1 built on top of PTK [78]. 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 Section 4.1). We trained the rerankers using SVM-light-TK[6], a structural kernel toolkit based on SVM-light [52], 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. Additionally, the F1 of some binary classifiers is reported.

Finally, we assessed the statistical significance of our results by using the model described in [134] and implemented in [84].

4.5.2 Classification Accuracy on Whole Reuters

In the first experiments, we used the Lewis' split. The results are reported in Table 4.7, whose columns have the following meaning: (i) Lewis' flat refers to the result achieved in Lewis et al. paper; (ii) *Ours, flat* is our reimplementation of the Lewis et al. MCC, i.e., a one-vs.-all multi-classifier; (iii) *Hier* goes beyond the flat model as it is a top down algorithm so already exploiting the classification hierarchy; and (vi) FRR and HRR are our kernel-based reranking models applied to hypotheses generated with a flat or structural algorithm.

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

Table 4.7: Comparison between our rankers on the entire Topic hierarchy of RCV1 exactly using Lewis' split and data.

Our flat MCC achieved a Micro-F1 of 81.5, which basically matches the 81.6 reported in [67]. 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

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

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 model is basically a reranker that only uses label subset features, i.e., dependencies between hierarchy nodes without using their structure. It is interesting to note that it improves the baselines, i.e., flat and top down models, but it is outperformed by FRR, which exploits hierarchical structural dependencies.

Very interestingly, HRR generates better hypotheses than the reranker using the same features of FRR, achieving slightly better accuracy (e.g., 0.855 - 0.849 = +0.6%, statistically significant result).

4.5.3 Classification Accuracy on the Reuters Subset

We used our sampled dataset to more easily compute learning and running time curves. To make such experiments more interesting, we removed the documents of children from their fathers (child-free). This enables a better assessment of our results concerning the encoding of dependencies with RRs. Indeed, such setting at least removes the hierarchical dependencies between a father and its children induced by their shared documents (i.e., their shared semantics).

Figures 4.22 and 4.23 report on the Micro-and Macro-Average F1 of our rerankers over 103 categories, according to different sets of training data, whereas Table 4.8 reports the F1 of some individual categories with different rerankers over all 103 categories, using the small subset of Reuters described in Table 4.1. The data uses the child-free setting.

We note that: first, the child-free setting is much harder than before (e.g., the baseline Micro-F1 for child-free is 0.640 whereas in an experiment using the same data and the standard child-full setting we obtained 0.769). As a consequence, the rerankers highly improve on the baseline by almost 5 absolute percent points in both Micro and Macro measures. Very interestingly, they only need a few thousands of training data for releasing most of their benefit.

Second, the models using the compact representation, i.e., FFRR and FHRR, produce the same accuracy as their respective counterpart, i.e., FRR and HRR. This is very interesting since, as shown in the next section, the compact representation produces much faster models. The differences between FFRR and FHRR, in this case, is statistically insignificant as the smaller test set does not allow to statistically assess close results. The only statistical significant differences are between the baseline and the reranking models (p=10^{-3}).

Third, the results on individual categories prevent to establish a clear winner between the rerankers, although, they always outperform the baseline.

Finally, to better understand the potential of hierarchical generation, Table 4.9 reports

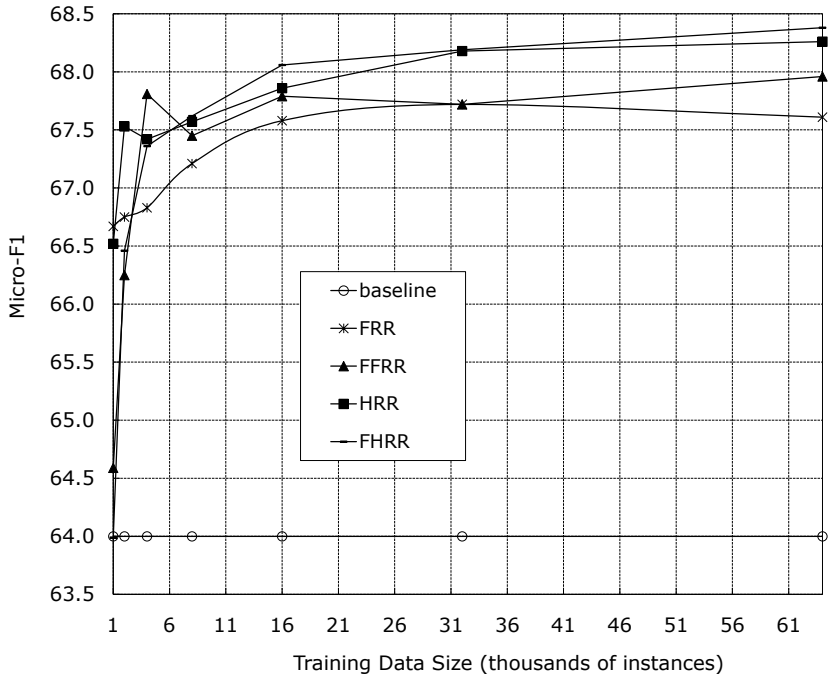

Figure 4.22: Learning curve of reranking models in terms of MicroAverage-F1 and reranking data.

the oracle performance with respect to the increasing number of hypotheses. This is the accuracy we would get if we were able to always select the best hypothesis with our reranker. The results clearly show that there is large margin of improvement for the rerankers and that the quality of the hierarchically generated hypotheses is better than the one of flat generated hypotheses.

4.5.4 Running Time on Reuter Subset

To study the applicability of our rerankers, we have analyzed both the training and classification time. Figure 4.24 shows the minutes required to train the different models on data of increasing size. Figure 4.25 reports the testing time required to classify 5k documents using rerankers learned on training sets of increasing size. It can be noted that the models using the compact hypothesis representation are much faster than those using the complete hierarchy as representation, i.e., up to five times in training and eight time in testing. This is not surprising as, in the latter case, each kernel evaluation requires to perform tree kernel evaluation on trees of 103 nodes. When using the compact representation the number of nodes is much lower.

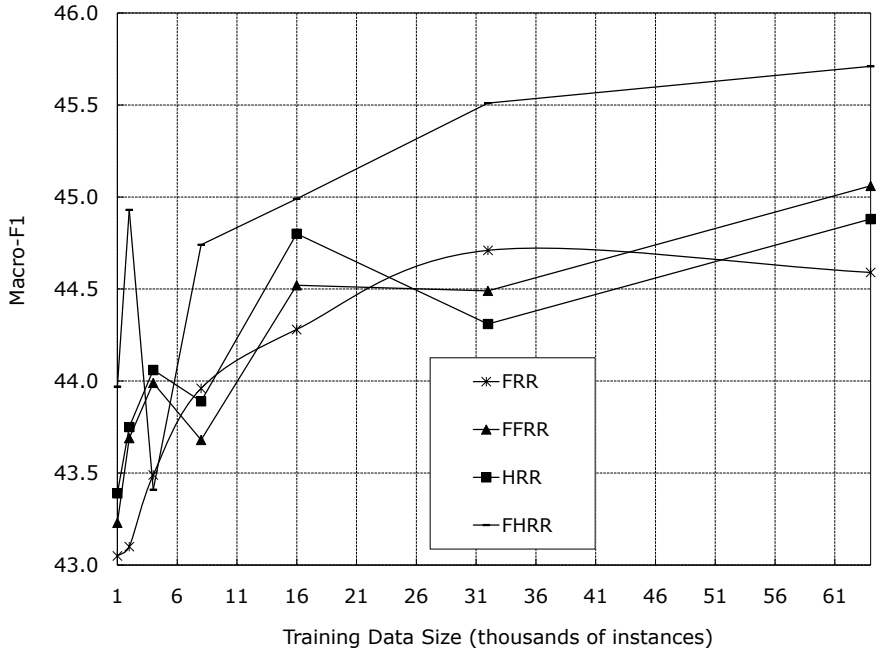

Figure 4.23: Learning curve of reranking models in terms of MacroAverage-F1 and reranking data.

4.5.5 Comparison with Zhou et al. (2011)

In addition, we compared to the work by Zhou et al. [128], which proposes a number of SVM models that improve over traditional flat and hierarchical classifiers. In order to ensure experimental compatibility, we prepared the data in a similar fashion and removed documents belonging to multiple siblings from the training and test sets. However, we obtained some differences in our dataset as shown in Table 4.10.

We experimented with three subhierarchies of the RCV collection: MCAT, CCAT, and ECAT, as in [128]. Table 4.11 shows our results obtained by our FRR and HRR rerankers and compare them with those reported in [128]. Additionally, considering that the Zhou et al. setting reduces our structural reranking to sequences we designed a reranker based on sequence kernels (*seq* column). The results show that: first, our baseline MCC already achieves higher accuracy than all the Zhou et al.'s models. This is probably due to the fact that (i) our baseline is very accurate as it exactly replicates the Lewis et al.'s MCC and (ii) we have some differences in the document set.

Second, our rerankers FRR and HRR highly improve on both our baselines as well as the best results obtained by Zhou et al. using their reranking approach (based on orthogonal transfer) for all three subhierarchies.

Third, FRR and HRR show similar accuracy in this setting (their difference is again statistical significant): this is reasonable as removing the sibling dependencies basically

Category	BL	SeqRR	FRR	FFRR	HRR	FHRR
C152	0.677	0.683	0.700	0.701	0.749	0.740
GPOL	0.660	0.588	0.695	0.706	0.674	0.684
M11	0.851	0.885	0.891	0.891	0.884	0.853
..	..	..	..	..	..	..
C183	0.507	0.546	0.588	0.612	0.588	0.575
G154	0.784	0.871	0.800	0.785	0.810	0.838
M142	0.678	0.618	0.721	0.706	0.722	0.718
..	..	..	..	..	..	..
C16	0.400	0.567	0.600	0.632	0.533	0.429
E121	0.316	0.433	0.546	0.546	0.563	0.483
G153	0.313	0.417	0.353	0.303	0.541	0.571
103 cat.						
Micro-F1	0.640	0.653	0.677	0.680	0.682	0.684
Macro-F1	0.408	0.431	0.447	0.451	0.443	0.457

Table 4.8: F1 of some different rerankers along with the Micro and Macro-Average F1 with child-free setting.

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 4.9: Oracle performance according to the number of hypotheses.

degenerates our structural hypothesis generation algorithm to an almost flat one.

Finally, to confirm the above claim, we experimented with a reranker based on a sequence kernel. This is applied to the path from the root to the only leaf derived from tree representation used in the tree kernels-based rerankers. Again, in the Zhou et al. setting, this is a tree with a single path. The accuracy of the model, *seq* column, equivalent to FRR and HRR, confirms that little structural information can be captured in this setting.

4.5.6 Multi-label Graph-induced Error

We also demonstrate that our approach is really effective for optimizing hierarchical classification. For this purpose, a hierarchical measure is needed, i.e., a measure that takes into

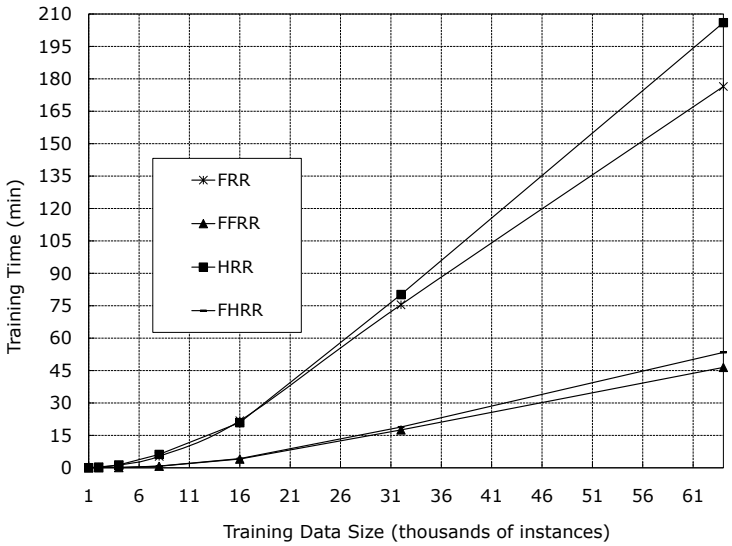

Figure 4.24: Training time of the rerankers trained on data of increasing size.

	$\|Y\|$		$\|L\|$		TOTAL		TRAIN		TEST	
	Zhou	Ours	Zhou	Ours	Zhou	Ours	Zhou	Ours	Zhou	Ours
CCAT	31	30	26	26	209,133	261,262	5,810	7,570	203,323	253,692
MCAT	9	9	7	7	189,211	180,438	5,438	5,205	183,773	175,233
ECAT	23	23	18	18	71,356	84,303	2,196	2,592	69,160	81,711

Table 4.10: Zhou et al.'s setting and differences with the achieved one. Y is the set of categories whereas L is the set of leaf categories.

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 farer node of the hierarchy. The Multi-label Graph-induced Error (MGIE, suggested by ECML/PKDD 2012 Discovery Challenge) takes the distances between true positives and false positives by also over-penalizing the false negatives. It is computed as follows: (i) find the smaller set between the true and the predicted classes of each document; (ii) compute the minimum graph distance between each class of the smaller set and the closest class of the other set, in such a way that minimizes the sum of distances; (iii) set all classes in excess equal to the maximum distance; and (iv) add all the distances and divide them by the number of the classification tasks, where such number is equal to the sum of the true categories of each document.

In our experiments we set the maximum distance to five (and seven), thus all distances above five (seven) are treated the same. The results are shown in Table 4.12. 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

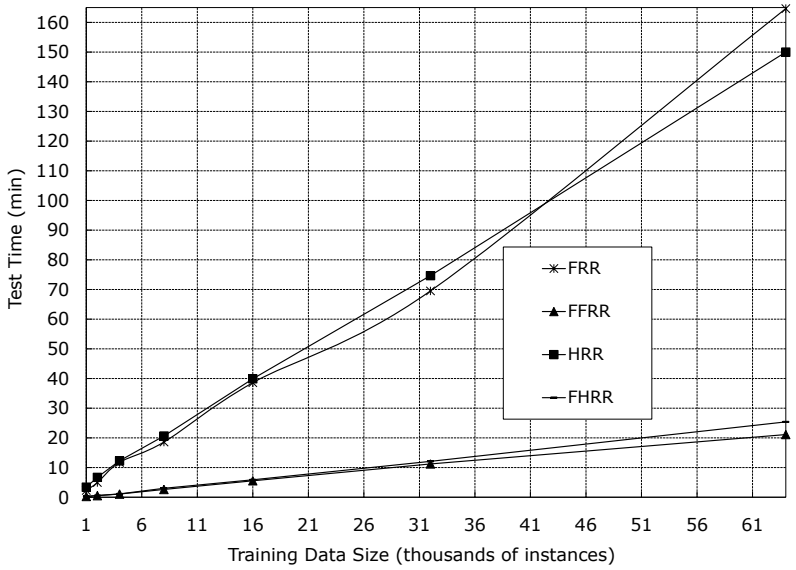

Figure 4.25: Testing time of the rerankers trained on data of increasing size.

Micro F1	baseline				improved			
	Zhou, flat	Ours, flat	Zhou, hier	Ours, hier	Zhou, ortho	FRR	HRR	Seq
MCAT	0.930	0.946	0.926	0.942	0.934	0.970	0.967	0.970
CCAT	0.731	0.787	0.738	0.760	0.764	0.825	0.821	0.829
ECAT	0.836	0.857	0.840	0.846	0.838	0.884	0.881	0.883

Table 4.11: Comparison with Zhou et al. [128].

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.

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

Table 4.12: Multi-label Graph-induced Error.

4.6 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 [94] 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 [73] and [40] used a rather different dataset and a different idea of dependencies based on feature distributions over the linked categories.

Heuristics for reducing error propagation and methods for dealing with large-scale problems were also proposed in [8, 129]: although they may be interesting from an application viewpoint, they do not help to compare against our model. In particular, early work on automated hierarchical text categorization, e.g., [40, 71, 58], 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. [8] 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. [34] used a Bayesian aggregator on the result of the individual binary classifiers, thus also this is generalized by our approach. [129] 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. [115] propose large-margin discriminative methods for generating complex output such as category label subsets. Our approach allow to generate hierarchies of labels thus improving it. However, we can only rely on a few hypotheses, which limits the search space but at the same time makes our approach scalable to larger categorization schemes.

The work on SVM-struct [114, 42] and meta-classifier [44] 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 [44] since it generates a superset of the meta-features used in such work. It also approximates [114] 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 4.7, SeqRR improves on the baseline but it is also outperformed by our hierarchal rerankers. [14] used discriminative functions to encode dependencies and to jointly learn a global loss over the hierarchy. Similar online methods were proposed in [35, 18]. Again, our reranker approach produces better features and it is in general more efficient.

On a different research line, hierarchical shrinkage in [90, 73] estimates parameters in Naïve Bayes classifiers considering the path from the root to the leaf node. A similar idea

is presented in [103], 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 [135], 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. [107] presents a survey of hierarchical classification methods.

4.7 Conclusions

After the extensive experimentation carried out in this chapter some almost definitive conclusions can be derived about the use of reranking model for improving HTC accuracy. We have divided our conclusions in two parts: (a) The use of FRR, i.e. the model based on the simple flat hypothesis generation and (b) the uses of more complicated hierarchical rerankers based on the hierarchical hypothesis generation by exploiting the category hierarchy.

4.7.1 Flat Reranker in Hierarchical Text Categorization

In this chapter an extensive evaluation of two flat reranking models (i.e., FRR and FFRR) has been reported. Real data (Reuter Corpus Volume I), as well as known benchmarking corpora have been used for comparative analysis. The results of such experiments allowed to systematically examine the following design choices for HTC:

1) the flat hypothesis generation algorithm gives a set of classification results, some of which are better than the local models: this has been measured with the oracle performance.

2) two kinds of representation of the classification results encoding the interdependencies of categories in hierarchy.

3) use of preference kernel based on structural kernels helps to learn the dependencies for ranking the hypotheses and output the best one as a final output.

Additionally, to better investigate the role of topical relationships, we consider two interesting cases:

- traditional categorization schemes in which node fathers include all the documents of their child categories; and
- more general schemes, in which children can include documents not-belonging to their fathers.

Data analysis has shown that all rerankers based on different kernels using the K_J (or K_P) combination (which exploits the joint hypothesis probability, or individual category probability) on SAM largely improved the baseline (state-of-the-art flat model).

In order to demonstrate that this is due to structural dependencies, we used:

1) the reranker based on BOL (it cannot capture the same dependencies as the structural kernels), which seems to provide the same performance as the baseline.

2) the reranker based on SK, which delivers an improvement similar to BOL, suggesting that sequential features are not enough.

3) the preference kernel based on tree kernels (STK and PTK), which significantly improve on BOL and SK.

More interestingly, tree kernels (STK and PTK) based on reranker significantly improve on BOL and SK when we remove the dependencies generated by shared documents between a node and its descendants (child-free setting). This further shows that structural dependencies rather than the documents dependencies play an important role in HTC.

Regarding the applicability of our rerankers, we have analyzed both training and classification time. It can be easily concluded that the models using the compact hypothesis representation are much faster than those using the complete hierarchy as representation, since in the latter case, each kernel evaluation requires to perform tree kernel evaluation on the entire hierarchy.

4.7.2 Hierarchical Reranker in Hierarchical Text Categorization

The study of the impact of hierarchical reranking on HTC allows to derive these main conclusions:

1) a more complicated hypothesis generation algorithm considers the hierarchy constraint and produces better hypotheses than the flat simple algorithm;

2) based on the high-quality hypotheses in 1), the HRR remarkably improves the baseline, and more importantly slightly improves the performance of FRR;

3) compact hypothesis representation based on FHRR is efficient while almost keeping the accuracy as HRR;

4) a hierarchical distance measure confirms that HRR and FHRR reduce errors computed as the graph distances between target label set and predicted label set.

Chapter 5

Local Incremental Reranking for Hierarchical Text Classification

Chapter 4 introduces two kinds of global reranking model to improve the state-of-the-art by learning the interdependencies among categories in the hierarchy. However, when the number of categories becomes huge (such as thousands of categories in DMOZ or wikipedia datasets), the global rerankers suffer from the inefficiency of the big bang approach.

In this chapter we propose:

1) an efficient local incremental reranking model (LIR) based on a top-down approach to improve the accuracy by absorbing the local category dependencies. The LIR consists of a set of *local rerankers*, and each is for one *sub-problem* decomposed from the entire hierarchy in Section 3.4.2. According to the top-down method, LIR recursively deals with the sub-problems instantiated by the hierarchy by applying the corresponding local rerankers.

2) fast LIR model combining the optimization of basic generative models and 1) for large-scale hierarchical text classification.

Section 5.1 simply reviews the top-down method in HTC and the global reranker to provide the preliminaries for LIR, which is described in details in Section 5.2. We make a comparative complexity analysis for GR and LIR in Section 5.2.3. Section 5.4 shows the experimentation, i.e., a comparison of GR and LIR on RCV1 in Section 5.4.1, and performances of fast LIR on the large-scale DMOZ dataset presented in Section 5.4.2. Finally, we derive the conclusions in Section 5.5.

5.1 Preliminaries

In this secton, we simply review the hierarchical models in HTC and the global reranking model proposed in Chapter 4 to provide background knowledge for the local incremental

reranker in the next section.

5.1.1 Hierarchcial Models in Hierarchical Text Categorization

Chapter 3 describes two very well-known methods for the design of hierarchical text classifiers (HTC): the *big bang* and the *top-down* approaches. The former learns a single (but generally complex) hierarchical classification model, which is inadequate for large hierarchies, e.g., Yahoo! Categories and Dmoz as it is too slow. The latter uses the hierarchical structure to decompose the entire problem into a set of smaller sub-problems. Then it proceeds in top-down fashion along the hierarchy, achieving high efficiency in both learning and prediction phase, since each time a much smaller problem with corresponding feature set is addressed. However, it suffers from the unrecovered errors from the higher to the lower nodes.

5.1.2 Global Structural Reranker

The approach of the global reranker proposed in [80] mainly consists of three different steps:

1) the application of the one-vs.-all approach to build a multi-classifier over all hierarchy categories;

2) the use of the classification probability of the binary node classifiers to generate k global classification hypotheses, i.e., the set of categories the target document belong to; and

3) reranking them by means of an SVM using tree kernels applied to the hierarchy tree, i.e., each hypothesis is represented by the tree associated with the hierarchy[1], where the classification decisions are marked in the node themselves.

It should be noted that: in Step (i) no information about the hierarchy is used. Step (ii) generates global classification hypotheses by also deriving their joint probability, which is used for preliminary ranking them. Step (iii) uses a reranker that exploits structural features. This includes co-occurrences, e.g., given three categories, C_1, C_2 and C_3, it encodes their subsets $\{C_1, C_2, C_3\}$, $\{C_1, C_3\}$, $\{C_1, C_2, C_3\}$ as features. Additionally, it also encodes their structures, e.g., C_1 is father of C_2 which is father of C_3 as features. More details about hypothesis representation and their use for training the rerankers are given in [80].

[1]The approach can also be extended to hierarchies having a DAG shape.

5.2 Local Incremental Reranker

5.2.1 LIR Learning

We could see from Section 5.1 that the top-down approach provides high efficiency for HTC but suffers from the error propagation, which is caused by wrong predictions in the upper levels of hierarchy. Meanwhile, the global reranker could improve the prediction by learning the category dependencies, as one of the big bang approaches, it is subject to the low efficiency in learning and classification phases. An intuition is whether we could find some model that overcomes the drawbacks above by combining the efficient top-down manner and GR's accurate prediction, particularly for the nodes in upper levels. The answer is positive and the local incremental reranking model in this section is such a model.

The local incremental reranking model consists of a set of *local rerankers*, each corresponds to one sub-problem in top-down method. To build the local reranker we need to:

1) Obtain the individual decision probabilities output by the top-down one-vs.-all classifiers, e.g., the local probability of each local classifier and subtree probability of each subtree classifier;

2) Generate the top k hypotheses based on the joint probabilities above for each subproblem;

3) Represent the hypotheses with the hierarchy and organize the top k hypotheses into two kinds of pairs: positive with the Micro-F1 of the first hypothesis better than the second, and negative with the first hypothesis worse than the second.

4) Learn a local reranker by using the tree kernels (e.g., PTK in the perference kernel K_P in [80]) applied to the hypothesis representation. The latter is just a tree constituted by a node and its children (obviously such classifier also labels internal nodes).

5.2.2 LIR Reranking

In the classification phase, we apply the node multi-classifiers (i.e., local and subtree classifiers) in top-down way and we rerank their decisions with the local rerankers. Of course, we progress to the children of a node only after the reranking step of the multi-classifier associated with its father is terminated. This way, LIR exploits the efficient top-down algorithm but at the same time allows for capturing dependencies between father and its children. These dependencies are then propagated in a top-down fashion.

5.2.3 Computational Complexity Analysis

The focus of our paper is to improve the efficiency of the global reranker. Thus, we will analyze the computational complexity of GR vs. the one of LIR. There are two sources of complexity in SVM using tree kernels: (i) the learning algorithm working in dual space; and (ii) the computation of the tree kernel function. Let us define:

- m the number of hierarchy nodes,
- μ the number of the internal nodes and
- n the size of training data.

The worst case complexity of global reranker is given by the SVM learning, i.e., $O(n^c)$, where $2 < c < 3$, multiplied by the tree kernel times, which is quadratic in the number of tree nodes, i.e., $O(m^2)$. Thus global reranker runs in $\boldsymbol{t_{GR}}$ and:

$$\boldsymbol{t_{GR}} = O(n^c) * O(m^2) = O(n^c m^2) \tag{5.1}$$

The worst case complexity of local incremental reranking happens when the hierarchy is flat ($m = 1$) but this is not an interesting case. Thus, let us consider, a non trivial hierarchy with $m >> 1$. We also consider the average case in which the training data is distributed uniformly between the categories[2]. With these assumptions, we have μ multi-classifiers, each with n/μ training examples. It follows that their learning complexity is $\mathrm{O}(\mu(n/\mu)^c)$ multiplied by the tree kernel complexity. This, considering that the local classifiers have on average $m/\mu + 1$ nodes, is $(m/\mu + 1)^2$. As a result, LIR shows a complexity:

$$\boldsymbol{t_{LIR}} = O(\mu(n/\mu)^c * (m/\mu)^2) = O((n/\mu)^c m^2/\mu) \tag{5.2}$$

under the condition $2 < c < 3$, we can get:

$$\boldsymbol{t_{LIR}} < O(n^c m^2/\mu^3) \tag{5.3}$$

which is lower than the $\boldsymbol{t}_{GR}$ of global reranker.

Furthermore, if we used the fact that $O(\mu^3) > O(m^2)$, we could see:

$$\boldsymbol{t_{LIR}} < O(n^c) \tag{5.4}$$

The classification analysis is similar as there is (i) a quadratic term $O(n^2)$ wrt the number of support vectors (lower but proportional to n) and (ii) the usual $O(m^2)$ term for the tree kernel evaluation.

[2]The usual case of the node father containing all the documents of the children, clearly violates such assumption. More complex equations taking into account this assumption can be defined but this is beyond the purpose of this paper.

5.3 Fast Local Incremental Reranker for Large-scale Hierarchical Text Categorization

The reranking system in this thesis actually can be divided into two parts: the basic generative model and the reranking model. The former we used in chapter 3 is the flat or top-down one-vs.-all proLIBSVM classifiers. The latter is based on the former, which is the RR and efficient fast RR based on the compact hypotheses representation. In order to overcome the low efficiency of RR, we further concentrated on improving the latter by proposing LIR in Section 5.2. This has made the reranking model efficient enough. However, when the number of categories becomes huge (thousands of categories) in a huge hierarchy tree such as the DMOZ, Yahoo! directories[3], the former (i.e., flat or top-down one-vs.-all proLIBSVM classifiers) become inadequate in both learning and classification.

In order to improving the efficiency of basic generative model on large-scale hierarchical dataset, we suggested:

- using proLIBLINEAR instead of proLIBSVM model. The experimentation in Section 3.6.2 has proven that the former is equivalent with the latter in accuracy but far more efficient, particularly for the learning on a large amount of training set in one-vs.-rest strategy.

- applying the proLIBLINEAR models in top-down manner as described in Section 3.4.2.

- creating the *local dictionary* for each sub-problem. Generally we establish one global dictionary (or vocabulary of features) when pre-processing the training documents, this results in the sparse coding of $\boldsymbol{w}$ value stored in proLIBLINEAR models (i.e., contains too many zeros in the $\boldsymbol{w}$), which makes two negative effects: (i) each proLIBLINEAR takes a big memory size and a large number of them consume too much memory of PC during classification phase; (2) it decreases the computational efficiency.

  In order to solve the sparse coding of $\boldsymbol{w}$, we create the a set of local dictionaries, each associating with a sub-problem in top-down method, thus $\boldsymbol{w}$ of models within such sub-problem are based on the corresponding local dictionary, which is low-dimension and non-sparse as the $\boldsymbol{w}$ based on the global dictionary.

  It is worth noting that the introduction of local dictionary does not affect the classification results since in top-down method because: (a) the multi-classifiers (i.e., local classifiers and subtree classifiers) are locally learned, which we can see from the construction of positive and negative examples for each classifier in Section 3.4.2; (b)

[3] http://www.dmoz.org/,http://dir.yahoo.com/

in classification phase, once we choose the sub-problem as the further step, the classification based on such sub-problem is also local, i.e., independent from its sibling sub-problems.

In a word, the optimizaition of basic generative models and the fast local rerankers applied in LIR form the software (let's call it fastLIR) for large-scale hierarchical text categorization. The experimentation of fastLIR on DMOZ is shown in Section 5.4.2.

5.4 Experiments and Evaluations

To our knowledge the only approach using the entire RCV1 hierarchy structure with the same setting as [67], and significantly outperforms the benchmark is the structural reranking model (RR) proposed in Chapter 4. This method, however, is computationally expensive. The aim of the experiments is to demonstrate that our LIR remarkably improves the efficiency with little loss of accuracy comparing to RR in [80].

In order to extend our LIR to the large-scale dataset, we suggested the fast LIR. The experiments on DMOZ show the efficiency of the fast LIR model on large-scale hierarchical text categorization.

5.4.1 GR and LIR comparison on the Whole RCV1

We compare GR against LIR with respect to accuracy and running time. We used Reuters Volume 1 (RCV1) with Lewis' split [67], which includes 23,149 news for training and 781,265 news for testing. We implement the top-down classifiers with SVMs using the default parameters (trade-off and cost factor = 1), linear kernel, normalized vectors (using the Euclidean norm), stemmed bag-of-words representation, $log(TF+1) \times IDF$ weighting scheme and a common stop list. All the performance figures are provided by means of Micro/Macro-Average F1, evaluated from our test data over all 103 categories.

Table 5.1 reports the accuracy whereas Table 5.2 illustrates the learning and classification time. The table columns have the following meanings:

- *flat* refers to the results achieved in [67] and [80], respectively;
- *top-down* is our reimplementation of the conventional top-down method;
- GR represents the best accuracy of kernel-based reranking models applied to the hypotheses made on all hierarchy classification; and
- LIR refers to our local incremental reranking model.

F1	baseline		GR	LIR
	flat	top down		
Micro-F1	0.816	0.819	0.849	0.841
Macro-F1	0.567	0.578	0.615	0.611

Table 5.1: Micro/Macro-F1 of different models on RCV1.

time cost	GR	LIR
Training (s)	9023.24	508.75
Test (h)	43.40	4.31

Table 5.2: Classification and training time of GR and LIR on RCV1.

We can clearly see from Table 5.1 that the top-down model slightly improves the flat models reported in [67], i.e., by $81.9 - 81.6 = 0.3$. This is significant with $p = 10^{-5}$, according to our significance test using approximate randomization (please consider that the test set contains about 800k examples). When LIR is applied to the top-down baseline, the latter improves by 2.2 absolute percent points (significant at $p = 10^{-5}$) in Micro-F1; similarly the baseline of flat model improves by 2.5 points (in Micro-average).

Most importantly, LIR remarkably outperforms the reranking model proposed in [80] in efficiency, i.e., 9023.24/508.75= 17.8 times in learning and 43.40/4.31= 10 times in testing. In contrast, it loses 0.8 points in Micro-F1 (more details on F1s of categories are presented in Appendix E).

5.4.2 Fast LIR on DMOZ

The DMOZ dataset is provided by LSHTC workshop[4] as document vectors in the following format:

label, label, label ... feat:value ... feat:value

label is an integer and corresponds to a category in which the vector belongs. Each vector may belong to more than one category. The pair feat:value corresponds to a non-zero feature with index feat and value, feat is an integer and value is a double that corresponds to the term's count.

We processed the document vector by computing the $log(TF + 1) \times IDF$ weighting scheme, and implemented the top-down classifiers with proLIBLINEAR using the default parameters (trade-off and cost factor = 1), linear kernel, normalized vectors (using the Euclidean norm). We used 4 cores of CPU for computing the training and test time. The performance figures are provided by means of Micro/Macro-Average F1, evaluated

[4] http://lshtc.iit.demokritos.gr/LSHTC2_datasets

from our test data over all 35,448 categories. Additionally, we give some computations of efficiencies in both learning and classification phases. Table 5.3 below reports all the figures.

	training time (min)	test time (min)	Micro-F1	Macro-F1
topDown_proLIBLINEAR	60.180	19.747	0.629	0.202
fast LIR	81.262	39.831	0.693	0.336

Table 5.3: Performances and efficiency of fast LIR on DMOZ.

We can clearly see from Table 5.3 that (1) proLIBLINEAR in top-down method makes the basic generative model on large-scale text categorization possible, i.e., only 60.18 minutes (four cores) for 35,448-category hierarchy with 300,000 training documents in learning phase, and 19.75 minutes cost for classifying 94,756 documents; (2) the training and test time of fast LIR is not much more than that of proLIBLINEAR, since the average leaf category per document is 1.0239, which makes the compact representation of hypotheses very small, and the learning of local rerankers fast; (3) fast LIR remarkably improves the top-down proLIBLINEAR models, i.e., by $69.3 - 62.9 = 6.4$ points in Micro-F1 and $33.6 - 20.2 = 13.4$ points in Macro-F1. This is significant with $p = 10^{-5}$, according to our significance test using approximate randomization (please consider that the test set contains about 90k examples).

Additionally, we roughly compared the memory occupied for top-down proLIBLIENAR models on the global dictionary and local dictionaries, and they are 90.56 vs 4.80 gigabytes. Obviously the former is hard to achieve with the general experimentation and use of a PC with 4.8 gigabytes is more practical.

5.5 Conclusion

In this chapter, we have described the local incremental reranking model based on the conventional top-down approach. This model not only helps to improve the top-down baseline by using local rerankers, which consider the category dependencies to solve the unrecovered errors in internal node categorization, but also ensures the efficiency in the top-down working way. We have seen a consistent improvement over state-of-the-art flat and top-down models. Most importantly, our LIR (i) is very efficient while keeping high accuracy (ii) can be applied to several other problems or domains.

We also suggested the fast LIR model based on LIR by improving the efficiency of basic generative models to make it applicable in large-scale hierarchical text categorization. This fast LIR model significantly improves the state-of-the-art top-down models.

Chapter 6

Conclusions and Future Work

In this thesis the complex interdependencies between categories in the hierarchy has been studied to develop an accurate and efficient model for HTC. This can be divided into the following phases.

First, a study on improving the efficiency of the basic multi-classifiers (e.g., SVM) has been carried out. We reimplemented (i) the flat one-vs.-all probabilistic models with LIBSVM, which not only matched the state-of-the-art method (i.e., flat SVM^{light} models) on RCV1 in [67], but also output the useful decision probability for each node. (ii) the top-down baseline model in [109] on RCV1, which reached almost the same performances as (i) with much higher efficiency because of the top-down classification manner. These two conventional methods both suffer from the low efficiency when there is a large number of categories or instances. In order to solve this problem, we developed the probabilistic one-vs.-all and top-down models with LIBLINEAR by implementing the Platt's algorithm [87] in original LIBLINEAR, which does not produce the decision probabilities. To match (i) and (ii), we used parameterization techniques. The results on RCV1 have shown that the probabilistic LIBLINEAR remarkably improves the efficiency of LIBSVM in both flat and top-down methods, while preserving the equivalent accuracy.

Second, the impact of category dependencies on HTC accuracy has been studied. We have described several models (GR) for reranking the output of an MCC based on SVMs and structural kernels, i.e., SK, STK and PTK. We have proposed two algorithms for hypothesis generation and their kernel-based representations. The latter are exploited by SVMs using preference kernels to automatically derive features from the hypotheses. When using tree kernels such features are tree fragments, which can encode complex semantic dependencies between categories. 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 the state-of-the-art TC models. Most importantly, our approach (i) is rather general, (ii) can be applied to several other problems or do-

mains and (iii) can be optimized according to several measures, e.g., MGIE. We tested our rerankers on two hierarchical schemes of the well-known RCV1. The results show impressive improvement on the state-of-the-art of flat TC models.

Finally, a efficient model (LIR) based on GR has been proposed. This model is based on the conventional top-down approach and allows for efficiently using structural dependencies provided by tree kernels in HTC. Additionally, we extend LIR (fastLIR) for the large-scale HTC by two steps: (1) adopting the compact representation of hypothesis for building the local rerankers and (2) optimizing the MCC by (a) using probabilistic LIBLINEAR instead of LIBSVM; (b) establishing local dictionary for each of sub-problems. The comparative experiments with the state-of-the-art model, GR, show that LIR is much more efficient while showing almost the same accuracy. The results of fastLIR on DMOZ proves that iis applicable in large-scale HTC.

Future research includes:

- Extending our reranking models that exploit the more complicated node dependencies in hierarchical structures, e.g., graph. It should be noted that the DAG tree structure can be dealt with by LIR or fastLIR;
- Constructing other approximate innoference strategies for generating k-best hypotheses, For instance, the k-best search algorithm for natural language parsing presented in [47] was later used as the main building block in the forest reranking method for approximate inference in complex discriminative parsing models [48];
- Studying how to fully exploit the set of best hypotheses as the oracle accuracy reveals that there is still a large room of improvement;
- Applying our methods to other hierarchical natural language classification tasks, e.g., ontology population.

Bibliography

[1] Nir Ailon and Mehryar Mohri. Preference-based learning to rank. *Machine Learning*, 2010.

[2] Erin L. Allwein, Robert E. Schapire, Yoram Singer, and Pack Kaelbling. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[3] Maria-Florina Balcan, Nikhil Bansal, Alina Beygelzimer, Don Coppersmith, John Langford, and Gregory B. Sorkin. Robust reductions from ranking to classification. *Machine Learning*, 72(1-2):139–153, 2008.

[4] N. Barabino, M. Pallavicini, A. Petrolini, M. Pontil, and A. Verri. Support vector machines vs multi-layer perceptrons in particle identification. In *In Proceedings of the European Symposium on Artifical Neural Networks '99 (D-Facto*, pages 257–262. Press, 1999.

[5] Roberto Basili, Alessandro Moschitti, and Maria Teresa Pazienza. A text classifier based on linguistic processing, 1999.

[6] Stephen D. Bay. Combining nearest neighbor classifiers through multiple feature subsets. In *Proc. 15th International Conf. on Machine Learning*, pages 37–45. Morgan Kaufmann, San Francisco, CA, 1998.

[7] Kristin P. Bennett, Leonardo Auslender, Donghui Wu, and Sagamore Ave. On support vector decision trees for database marketing. Technical report, Department of Mathematical Sciences Math Report No. 98-100, Rensselaer Polytechnic Institute, 1998.

[8] Paul N. Bennett and Nam Nguyen. Refined experts: improving classification in large taxonomies. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 11–18, New York, NY, USA, 2009. ACM.

[9] R.C. Bose and D.K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68 – 79, 1960.

[10] Hervé Bourlard and Nelson Morgan. A continuous speech recognition system embedding mlp intohmm. In *NIPS*, pages 186–193, 1989.

[11] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, 2004.

[12] Erin J. Bredensteiner and Kristin P. Bennett. Multicategory classification by support vector machines. *Computational Optimizations and Applications*, 12:53–79, 1999.

[13] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[14] Lijuan Cai and Thomas Hofmann. Hierarchical document categorization with support vector machines. In *CIKM*, pages 78–87, 2004.

[15] Nicola Cancedda, Eric Gaussier, Cyril Goutte, and Jean Michel Renders. Word sequence kernels. *Journal of Machine Learning Research*, 3:1059–1082, 2003.

[16] Maria Fernanda Caropreso, Maria Fernandacnandad, Stan Matwin, and Fabrizio Sebastiani. A learner-independent evaluation of the usefulness of statistical phrases for automated text categorization, 2001.

[17] Xavier Carreras and Lluís Màrquez. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 152–164, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[18] Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Incremental algorithms for hierarchical classification. *J. Mach. Learn. Res.*, 7:31–54, dec 2006.

[19] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[20] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of ACL*, pages 173–180, Ann Arbor, United States, 2005.

[21] Xi Chen and Jing Han. A novel classification approach based on support vector machine and adaptive particle swarm optimization algorithm. In *Knowledge Acquisition and Modeling, 2008. KAM '08. International Symposium on*, pages 703 –707, dec. 2008.

[22] Yangchi Chen, M.M. Crawford, and J. Ghosh. Integrating support vector machines in a hierarchical output space decomposition framework. In *Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004 IEEE International*, volume 2, pages 949–952 vol.2, sept. 2004.

[23] Amanda Clare and Ross D. King. Knowledge discovery in multi-label phenotype data. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '01, pages 42–53, London, UK, UK, 2001. Springer-Verlag.

[24] Michael Collins and Nigel Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of ACL'02*, pages 263–270, 2002.

[25] Anastasios Tefas Constantine, Constantine Kotropoulos, and Ioannis Pitas. Enhancing the performance of elastic graph matching for face authentication by using support vector machines. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1999.

[26] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[27] Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. In *In Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 35–46, 2000.

[28] Koby Crammer, Yoram Singer, Nello Cristianini, John Shawe-taylor, and Bob Williamson. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:2001, 2001.

[29] Aron Culotta and Jeffrey Sorensen. Dependency Tree Kernels for Relation Extraction. In *Proceedings of ACL'04*, 2004.

[30] Chad Cumby and Dan Roth. On kernel methods for relational learning. In *Proceedings of ICML 2003*, 2003.

[31] Stephen D'Alessio, Keitha Murray, Robert Schiaffino, and Aaron Kershenbaum. The effect of using hierarchical classifiers in text categorization. In *In Proc. of 6th*

International Conference Recherche dInformation Assistee par Ordinateur (RIAO-00, pages 302–313, 2000.

[32] Hal Daumé III and Daniel Marcu. NP bracketing by maximum entropy tagging and SVM reranking. In *Proceedings of EMNLP'04*, 2004.

[33] D.D.Lewis. Reuters-21578 text categorization test collection. *Distribution 1.0. README file (version 1.2),Manuscript*, September 1997.

[34] Christopher DeCoro, Zafer Barutcuoglu, and Rebecca Fiebrink. Bayesian aggregation for hierarchical genre classification. In *International Symposium on Music Information Retrieval 2007*, sep 2007.

[35] Ofer Dekel, Joseph Keshet, and Yoram Singer. Large margin hierarchical classification. In *In Proceedings of the Twenty-First International Conference on Machine Learning*, pages 209–216, 2004.

[36] Frédéric Delbos and Jean Charles Gilbert. Global linear convergence of an augmented Lagrangian algorithm for solving convex quadratic optimization problems. Rapport de recherche RR-5028, INRIA, 2003.

[37] Mona Diab, Alessandro Moschitti, and Daniele Pighin. Semantic role labeling systems for Arabic using kernel methods. In *Proceedings of ACL-08: HLT*, pages 798–806, 2008.

[38] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[39] Sotiris Diplaris, Grigorios Tsoumakas, Pericles A. Mitkas, and Ioannis Vlahavas. Protein classification with multiple algorithms. In *10 th Panhelllenic Conference on Informatics (PCI 2005), P. Bozanis and E.N. Houstis (Eds.), Spring-Verlag, LNCS 3746*, pages 448–456, 2005.

[40] Susan T. Dumais and Hao Chen. Hierarchical classification of web content. In Nicholas J. Belkin, Peter Ingwersen, and Mun-Kew Leong, editors, *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval*, pages 256–263, Athens, GR, 2000. ACM Press, New York, US.

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

[42] T. Finley and T. Joachims. Parameter learning for loopy markov random fields with structural support vector machines. In *ICML Workshop on Constrained Optimization and Structured Output Spaces*, 2007.

[43] T. Gonçalves and P. Quaresma. *A Preliminary Approach to the Multilabel Classification Problem of Portuguese Juridical Documents*, volume 2902/2003 of *Lecture Notes in Computer Science*. Springer, 2003.

[44] Siddharth Gopal and Yiming Yang. Multilabel classification with meta-level features. In *SIGIR*, 2010.

[45] Chen H. and Dumais S. Bringing order to the web: Automatically categorizing search results. In *In Proc. of CHI*, pages 145–152. ACM Press, 2000.

[46] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 408–415, New York, NY, USA, 2008. ACM.

[47] Liang Huang. Better k-best parsing. In *Proceedings of the 9th Interna- tional Workshop on Parsing Technologies (IWPT 2005)*, pages 53–64, Vancouver, Canada, 2005.

[48] Liang Huang. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08: HLT*, pages 586–594, Columbus, United States, 2008.

[49] Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT 2005)*, pages 53–64, Vancouver, Canada, 2005.

[50] David J. Ittner, David D. Lewis Y, and David D. Ahn Z. Text categorization of low quality images. In *In Proceedings of SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval*, pages 301–315, 1995.

[51] Chih jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust region newton method for large-scale logistic regression. In *In Proceedings of the 24th International Conference on Machine Learning (ICML*, 2007.

[52] Thorsten Joachims. Making large-scale SVM learning practical. *Advances in Kernel Methods – Support Vector Learning*, 13, 1999.

[53] Qi Ju, Richard Johansson, and Alessandro Moschitti. Towards using reranking in hierarchical classification. In *Proceedings of the Joint ECML/PKDD–PASCAL Workshop on Large-Scale Hierarchical Classification*, Athens, Greece, 2011.

[54] Qi Ju and Alessandro Moschitti. Incremental reranking for hierarchical text classification. In *Proceedings of the 34th European Conference on Information Retrieval (ECIR'13)*, Moscow, Russia, 2013.

[55] Qi Ju, Alessandro Moschitti, and Richard Johansson. Learning to rank from structures in hierarchical text classification. In *Proceedings of the 34th European Conference on Information Retrieval (ECIR'13)*, Moscow, Russia, 2013.

[56] Qi Ju, Chiara Ravagni, Alessandro Moschitti, and Giampiero Vaschetto. Hierarchical text classification for supporting educational programs. In *IIR'12*, pages 18–25, 2012.

[57] S. Knerr, L. Personnaz, and G. Dreyfuss. Single-layer learning revisited: a stepwise procedure for building and training a neural network. 1990.

[58] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 170–178, Nashville, US, Nov. 1997.

[59] Ulrich H. G. Kressel. Pairwise classification and support vector machines. pages 255–268, 1999.

[60] Taku Kudo and Yuji Matsumoto. Fast methods for kernel-based text analysis. In *Proceedings of ACL'03*, 2003.

[61] Taku Kudo, Jun Suzuki, and Hideki Isozaki. Boosting-based parse reranking with subtree features. In *Proceedings of ACL'05*, 2005.

[62] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In Jerzy Neyman, editor, *Proceedings of the 2nd Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. University of California Press, Berkeley, CA, USA, 1950.

[63] Shailesh Kumar, Joydeep Ghosh, and Melba M. Crawford. Hierarchical fusion of multiple classifiers for hyperspectral data analysis. *Pattern Anal. Appl.*, 5(2):210–220, 2002.

[64] Yannis Labrou and Tim Finin. Yahoo! as an ontology: using yahoo! categories to describe documents. In *Proceedings of the eighth international conference on Information and knowledge management*, CIKM '99, pages 180–187, New York, NY, USA, 1999. ACM.

[65] Boris Lauser and Andreas Hotho. Automatic multi-label subject indexing in a multilingual environment. In Traugott Koch and IngeborgTorvik Slvberg, editors,

Research and Advanced Technology for Digital Libraries, volume 2769 of *Lecture Notes in Computer Science*, pages 140–151. Springer Berlin Heidelberg, 2003.

[66] Yoonkyung Lee, Yi Lin, and Grace Wahba. Multicategory support vector machines, theory, and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, 99:67–81, 2004.

[67] D. D. Lewis, Y. Yang, T. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *JMLR*, 2004.

[68] David D. Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '92, pages 37–50, New York, NY, USA, 1992. ACM.

[69] David D. Lewis and Marc Ringuette. A comparison of two learning algorithms for text categorization. In *In Third Annual Symposium on Document Analysis and Information Retrieval*, pages 81–93, 1994.

[70] Min ling Zhang and Zhi hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *PATTERN RECOGNITION*, 40:2007, 2007.

[71] Tie-Yan Liu, Yiming Yang, Hao Wan, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explorations*, 7(1):36–43, 2005.

[72] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, March 2002.

[73] Andrew McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *ICML*, 1998.

[74] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 209:pp. 415–446, 1909.

[75] Hahn ming Lee, Chih ming Chen, and Cheng wei Hwang. A hierarchical neural network document classifier with linguistic feature selection. *Applied Intelligence*, 23:277–294, 2005.

[76] Katharina Morik and Martin Scholz. The miningmart approach to knowledge discovery in databases. In *In Ning Zhong and Jiming Liu, editors, Intelligent Technologies for Information Analysis*, pages 47–65. Springer, 2003.

[77] Alessandro Moschitti. A study on convolution kernels for shallow semantic parsing. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[78] Alessandro Moschitti. Efficient convolution kernels for dependency and constituent syntactic trees. In *Proceedings of ECML'06*, 2006.

[79] Alessandro Moschitti. Making tree kernels practical for natural language learning. In *Proccedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 113–120, Trento, Italy, 2006.

[80] Alessandro Moschitti, Qi Ju, and Richard Johansson. Modeling topic dependencies in hierarchical text categorization. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL 2012)*, pages 759–767, Jeju, Republic of Korea, 2012.

[81] Alessandro Moschitti, Daniele Pighin, and Roberto Basili. Tree kernels for semantic role labeling. *Computational Linguistics*, 34(2):193–224, 2008.

[82] Katta G. Murty. A new practically efficient interior point method for lp. *Algorithmic Operations Research*, pages 1–3, 2006.

[83] K.G. Murty. *Linear complementarity, linear and nonlinear programming.* Sigma series in applied mathematics. Heldermann, 1988.

[84] Sebastian Padó. *User's guide to* `sigf`*: Significance testing by approximate randomisation*, 2006.

[85] Edwin P.D. Pednault and Edwin P. D. Pednault. Statistical learning theory. In *MIT Encyclopedia of the Cognitive Sciences*, pages 798–800. MIT Press, 1998.

[86] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING, 1998.

[87] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *ADVANCES IN LARGE MARGIN CLASSIFIERS*, pages 61–74. MIT Press, 1999.

[88] John C. Platt, Nello Cristianini, and John Shawe-taylor. Large margin dags for multiclass classification. In *Advances in Neural Information Processing Systems*, pages 547–553. MIT Press, 2000.

[89] M. F. Porter. Readings in information retrieval. pages 313–316, 1997.

[90] Kunal Punera and Joydeep Ghosh. Enhanced hierarchical classification via isotonic smoothing. In *WWW*, 2008.

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

[92] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *J. Mach. Learn. Res.*, 5:101–141, December 2004.

[93] Irina Rish. An empirical study of the naive Bayes classifier. In *IJCAI-01 workshop on "Empirical Methods in AI"*, Oct. 2005.

[94] Juho Rousu, Craig Saunders, Sandor Szedmak, and John Shawe-Taylor. Kernel-based learning of hierarchical multilabel classification models. *JMLR*, 2006.

[95] Miguel E. Ruiz and Padmini Srinivasan. Hierarchical text categorization using neural networks. *Information Retrieval*, 5:87–118, 2002.

[96] M. Rychetsky, S. Ortmann, and M. Glesner. Support vector approaches for engine knock detection. In *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, volume 2, pages 969 –974 vol.2, jul 1999.

[97] G. Salton. Development in automatic text retrieval. *Science*, 253:974–980, 1991.

[98] Gerard Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[99] Minoru Sasaki and Kenji Kita. Rule-based text categorization using hierarchical categories. In *in Proc. of the IEEE Int*, pages 2827–2830, 1998.

[100] Robert E. Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. In *Machine Learning*, pages 135–168, 2000.

[101] Nicol N. Schraudolph, Jin Yu, and Simon Gnter. A stochastic quasi-newton method for online convex optimization. In *In Proceedings of 11th International Conference on Artificial Intelligence and Statistics*, 2007.

[102] Hinrich Schütze, David A. Hull, and Jan O. Pedersen. A comparison of classifiers and document representations for the routing problem. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '95, pages 229–237, New York, NY, USA, 1995. ACM.

[103] Babak Shahbaba and Radford M. Neal. Improving classification when a class hierarchy is available using a hierarchy-based prior. Technical report, Bayesian Analysis, 2005.

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

[105] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[106] Libin Shen and Aravind K. Joshi. An SVM-based voting algorithm with application to parse reranking. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 9–16, 2003.

[107] Carlos N. Silla, Jr. and Alex A. Freitas. A survey of hierarchical classification across different application domains. *Data Min. Knowl. Discov.*, Jan. 2011.

[108] S. M. Sinha. A duality theorem for nonlinear programming. *Management Science*, 12(5):pp. 385–390, 1966.

[109] Aixin Sun and Ee-Peng Lim. Hierarchical text classification and evaluation. In *ICDM*, 2001.

[110] Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. The conll 2008 shared task on joint parsing of syntactic and semantic dependencies. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 159–177, Manchester, England, August 2008. Coling 2008 Organizing Committee.

[111] Ivan Titov and James Henderson. Porting statistical parsers with data-defined kernels. In *Proceedings of CoNLL-X*, 2006.

[112] Kristina Toutanova and Francine Chen. Text classification in a hierarchical mixture model for small training sets. In *In CIKM 01: Proceedings of the tenth international conference on Information and knowledge management*, pages 105–113. ACM Press, 2001.

[113] Kristina Toutanova, Penka Markova, and Christopher Manning. The leaf path projection view of parse trees: Exploring string kernels for HPSG parse selection. In *Proceedings of EMNLP 2004*, 2004.

[114] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*, 2004.

[115] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484, December 2005.

[116] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis P. Vlahavas. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook*, pages 667–685. 2010.

[117] R. Anitha V. Srividhya. Evaluating preprocessing techniques in text categorization. *International Journal of Computer Science and Application*, pages 49–51, 2010.

[118] H. Van Dyke Parunak. Book review: Neural networks for pattern recognition by christopher m. bishop (clarendon press, 1995). *SIGART Bull.*, 9(1):41–43, jun 1998.

[119] K. Veropoulos, N. Cristianini, and C. Campbell. The application of support vector machines to medical decision support: A case study, 1999.

[120] S. V. N. Vishwanathan and Alexander J. Smola. Fast kernels for string and tree matching. In *NIPS*, pages 569–576, 2002.

[121] Volkan Vural and Jennifer G. Dy. A hierarchical method for multi-class support vector machines. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 105–, New York, NY, USA, 2004. ACM.

[122] Karl Sirotkin W. John Wilbur. The automatic identification of stop words. *Journal of Information Science*, 18(1):45–55, 1992.

[123] Ke Wang and Senqiang Zhou. Hierarchical classification of real life documents. In *In Proceedings of the 1st SIAM International Conference on Data Mining*, 2001.

[124] Ke Wang, Senqiang Zhou, and Shiang Chen Liew. Building hierarchical classifiers using class proximity, 1999.

[125] Andreas S. Weigend, Erik D. Wiener, and Jan O. Pedersen. Exploiting hierarchy in text categorization. *Inf. Retr.*, 1(3):193–216, October 1999.

[126] Jason Weston and Chris Watkins. Multi-class support vector machines, 1998.

[127] Erik Wiener, Jan O. Pedersen, and Andreas S. Weigend. A neural network approach to topic spotting, 1995.

[128] Lin Xiao, Dengyong Zhou, and Mingrui Wu. Hierarchical classification via orthogonal transfer. In *ICML*, pages 801–808, 2011.

[129] Gui-Rong Xue, Dikan Xing, Qiang Yang, and Yong Yu. Deep classification in large-scale text hierarchies. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 619–626, New York, NY, USA, 2008. ACM.

[130] Xiao-Bing Xue and Zhi-Hua Zhou. Distributional features for text categorization. *Knowledge and Data Engineering, IEEE Transactions on*, 21(3):428 –442, march 2009.

[131] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[132] Yiming Yang and John Wilbur. Using corpus statistics to remove redundant words in text categorization. *Journal of the American Society for Information Science*, 47(5):357–369, 1996.

[133] Yiming Yang, Jian Zhang, and Bryan Kisiel. A scalability analysis of classifiers in text categorization. In *in Proceedings of SIGIR-03, 26th ACM International Conference on Research and Development in Information Retrieval, ACM*, pages 96–103. Press, 2003.

[134] Alexander S. Yeh. More accurate tests for the statistical significance of result differences. In *COLING*, pages 947–953, 2000.

[135] Dengyong Zhou, Lin Xiao, and Mingrui Wu. Hierarchical classification via orthogonal transfer. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pages 801–808, 2011.

Appendix A

Notations

Below is the notation used in this dissertation.

d	a document
n_i	the node i in a hierarchy
c_{n_i}	the ordered child sequences of node i
$c_n(j)$	the jth child of node c_n
$\boldsymbol{x}_i$	the ith instance (example)
$\mathcal{X}$	domain of instances (examples)
y_i	class label for the ith instance (example)
Y_i	class label set for the ith instance (example)
$\mathcal{Y}$	finite set of class labels
$\mathcal{K}$	kernel function
$\mathcal{C}$	the finite set of predefined class labels
$\|\mathcal{C}\|$	the size of predefined class set
C_i	the positive class set of the instance $\boldsymbol{x}_i$
$\|C_i\|$	the size of positive class set of the instance $\boldsymbol{x}_i$
c_i	the ith category
T	training set
$\mathbb{H}$	learning model (classifier)
$\mathcal{H}$	boundary hyperplane in SVMs
$\boldsymbol{w}$	normal vector to the hyperplane
$\|\boldsymbol{w}\|$	the offset of the hyperplane from the origin along the $\boldsymbol{w}$
ξ_i	positive slack variable
Φ	a mapping for space transformation in SVMs
$\boldsymbol{o}_i$	structural object
f_i	the ith tree fragment

$\mathcal{F}$ set of tree fragments
$\tilde{\mathbf{h}}_i$ the ith generated hypothesis
$\mathcal{P}(\tilde{\mathbf{h}}_i)$ joint probability of the ith hypothesis
Δ function of computing the number of common fragments rooted in two nodes
TK a tree kernel function

Appendix B

e-Value Taxonomy

parent	child	child-description
None	Root	No Description
Root	C1	Categorizzazione Contesto Didattico*
C1	C11	Scuola Dell'infanzia*
C1	C12	Scuola Primaria*
C12	C121	Primaria Classe I*
C12	C122	Primaria Classe II*
C12	C123	Primaria Classe III*
C12	C124	Primaria Classe IV*
C12	C125	Primaria Classe V*
C1	C13	Scuola Secondaria di 1 Grado*
C1	C14	– Scuola Secondaria di 2 Grado*
Root	C2	Categorizzazione Materia*
C2	C21	Letto Scrittura*
C21	C211	Prerequisiti*
C21	C212	Decodifica*
C212	C2121	Lettere
C212	C2122	Sillable
C212	C2123	Parole
C212	C2124	Non parole
C212	C2125	Frasi-Brano
C21	C213	Comprensione*
C213	C2131	Parole
C213	C2132	Frasi
C213	C2133	Brano

C21	C214	Compitazione*
C214	C2141	Clettere
C214	C2142	Sillabe non Ortografiche
C214	C2143	Parole non Ortografiche
C214	C2144	Non Parole
C21	C215	Ortografia*
C215	C2151	Oparole
C215	C2152	Ofrasi
C215	C2153	Obrano
C21	C216	Stesura Testo*
C216	C2161	Pianificazione
C216	C2162	Trascrizione
C216	C2163	Revisione
C2	C22	Metafonologia*
C22	C221	Globale*
C221	C2211	Rima
C221	C2212	Sillaba
C22	C222	Profonda*
C222	C2221	Fonema
C2	C23	Abilità Linguistiche*
C23	C231	Lessico*
C231	C2311	Denominazione
C231	C2312	Categorizzazione
C231	C2313	Identificazione
C231	C2314	Definizione
C231	C2315	Polisemia
C231	C2316	Arricchimento Lessicale
C23	C232	Morfo Sintassi*
C232	C2321	Concordanze
C232	C2322	Struttura Della Frase
C232	C2323	Analisi Grammaticale
C232	C2324	Analisi Logica
C23	C233	Narrazione*
C233	C2331	Comprensione Racconto
C233	C2332	Produzione Racconto
C2	C24	Matematica*
C24	C241	Numero*

C241	C2411	Processi Semantici
C241	C2412	Conteggio
C241	C2413	Processi Pre-sintattici
C241	C2414	Processi Lessicali e Sintattici
C24	C242	Calcolo Processi di Base*
C242	C2421	Segni delle Operazioni
C242	C2422	Fatti Numerici
C242	C2423	Tabelline
C242	C2424	Calcolo a Mente
C24	C243	Calcolo Numeri Naturali*
C243	C2431	Algoritmi di Calcolo Scritto
C243	C2432	Calcolo Incolonnamento di Numeri
C243	C2433	Multipli e Divisori
C243	C2434	Minimo Comun Multiplo e Massimo Comun Denominatore
C243	C2435	Espressioni
C243	C2436	Potenze
C243	C2437	Radici Quadrate
C24	C244	Calcolo-Numeri Razionali
C24	C244	⁻Calcolo-Numeri Razionali Q*
C24	C245	⁻Calcolo-Numeri Relativi Z*
C24	C245	Calcolo-Numeri Relativi
C24	C246	Calcolo-Rapporti e Proporzioni*
C24	C247	Calcolo-Calcolo Letterale*
C24	C248	Problem Solving*
C24	C249	Capacità di Orientarsi Nello Spazio*
C24	C24_10	Costruire Sistemi di Riferimento Convenzionali*
C24	C24_11	Geometria Euclidea (Piana)*
C24	C24_12	⁻Misura di Grandezze*
C24	C24_12	Misura di Grandezze Geometriche
C24	C24_13	Misura di Grandezze Fisiche
C24	C24_13	⁻Calcolo-Misura*
C24	C24_14	Le Trasformazioni Geometriche*
C2	C25	Altro
Root	C3	Categorizzazione Situazione Alunni*
C3	C31	BES*
C31	C311	Autismo
C31	C312	Udito

C31	C313	Vista
C31	C314	Psicomotricità
C31	C315	Sindrome di Down
C31	C316	Altro
C3	C32	DSA*
C32	C321	Iperattività*
C32	C322	Dislessia*
C32	C323	Disgrafia*
C32	C324	Discalculia*
C32	C325	Combinazione di DSA Diversi-Altro*
C32	C326	Nessun DSA
C3	C33	¯Assenza di DSA e BES*
Root	C4	Categorizzazione Dipo di Intervento
C4	C41	Potenziamento
C4	C42	Recupero
C4	C43	Didattica Insegnamento
C4	C44	Intervento Logopedico
C4	C45	Intervento Psicologico

Appendix C

Performances of Italian dataset

	Train_No	Test_No	Precision	Recall	F1
C1*	38/69	16/27	0.842/0.794	1.000/1.000	0.914/0.885
C2*	40/70	20/34	0.905/0.917	0.950/0.971	0.927/0.943
C3*	41/76	19/34	0.950/0.944	1.000/1.000	0.974/0.971
C4	39	17	1.000	0.882	0.938
Micro			0.920/0.887	0.958/0.909	0.939/0.935
Macro			0.924/0.885	0.955/0.990	0.938/0.933
C11*	5/10	1/1	0.000	0.000	
C12*	36/62	25/24	0.933/0.706	0.933/1.000	0.933/0.828
C13*	7/14	1/6	0.000	0.000	0.000
−C14*	2	0			
C21*	12/23	5/12	1.000/0.857	0.800/0.500	0.889/0.632
C22*	10/14	3/3	0.400/0.250	0.667/0.333	0.500/0.286
C23*	4/7	1/7	0.941/	0.941/	0.941/
C24*	20/37	11/20	1.000/0.864	1.000/0.950	1.000/0.905
C25	0	1	0.000	0.000	0.000
C31*	2/64	0/28	/0.824	/1.000	/0.903
C32*	39/3	19/1	0.950/	1.000/	0.974/
−C33*	13	6	0.750	0.500	0.600
C41	23	11	1.000	0.909	0.952
C42	31	12	0.857	1.000	0.923
C43	25	8	0.889	1.000	0.941
C44	10	6	1.000	0.667	0.800
C45	0	1			
Micro			0.916/0.825	0.916/0.862	0.916/0.843

Macro			0.641/0.531	0.637/0.558	0.633/0.535
C121*	23/35	10/10	0.667/0.545	0.600/0.600	0.632/0.571
C122*	20/32	8/10	/0.750	/0.600	/0.667
C123*	10/21	8/11			
C124*	10/21	5/11	1.000/	0.600/	0.750/
C125*	9/20	3/10	0.333/0.500	0.333/0.100	0.333/0.167
C211*	/3	/0			
C212*	5/5	1/6	/0.667	/0.333	/0.444
C213*	/10	/5	/1.000	/0.200	/0.333
C214*	0/2	1/4			
C215*	0/4	2/2			
C216*	2/3	1/4			
C221*	9/10	2/3	0.400/	1.000/	0.571/
C222*	9/13	3/1	0.667/	0.667/	0.667/
C231*	/4	/5	/1.000	/0.200	/0.333
C232*	2/2	1/3			
C233*	/3	/0			
C241*	1/6	2/2			
C242*	12/20	10/9	1.000/0.667	0.400/0.889	0.571/0.762
C243*	1/3	4/5			
C244					
−C244*	2	0			
−C245*	3	3	1.000	0.667	0.800
C245					
C246*	/2	/0			
C247*	/4	/0			
C248*	/12	/2			
C249*	/1	/2			
C24_10*	/4	/1	/0.333	/1.000	/0.500
C24_11*	/3	/0			
−C24_12*	6	1			
C24_12					
C24_13					
−C24_13*	4	1			
C24_14*	/2	/1	/1.000	/1.000	1.000
C311*	/3	/1	1.000	1.000	1.000
C312*	/31	/16	/0.875	/0.875	/0.875

C313*	/10	/1	/0.500	/1.000	/0.667
C314*	/23	/11	/0.833	/0.909	/0.870
C315*	/15	/5	/0.500	/0.400	/0.444
C316					
C321	0	3			
C322	20	9	0.700	0.778	0.737
C323	1	6			
C324	16	4	1.000	1.000	1.000
C325	5	1			
C326					
Micro			0.855/0.761	0.725/0.665	0.785/0.709
Macro			0.288/0.330	0.269/0.403	0.263/0.390
C2121	2	1			
C2122	3	1			
C2123	3	1			
C2124					
C2125					
C2141					
C2142	0	1			
C2143					
C2144					
C2151	0	2			
C2152	0	1			
C2153	0	1			
C2161	2	1			
C2162					
C2163					
C2211	4	1	1.000	1.000	1.000
C2212	9	2	0.400	1.000	0.571
C2221					
C2311					
C2312					
C2313					
C2314					
C2315					
C2316					
C2321					

C2322	0	1			
C2323	2	1			
C2324	1	1			
C2331					
C2332					
C2411	1	2			
C2412					
C2413					
C2414					
C2421	5	3			
C2422	4	5			
C2423	1	2			
C2424	3	6	1.000	0.167	0.286
C2431	1	4			
C2432	1	1			
C2433	0	2			
C2434					
C2435					
C2436					
C2437					
Micro			0.843/0.761	0.640/0.665	0.727/0.709
Macro			0.139/0.330	0.129/0.403	0.115/0.390

This appendix provides all F1s of MCC on both Italian dataset. Each item in the table is split by "/", and the first is the value of 112-category dataset and the latter is for updated 50-category dataset. For simplicity, we ignore the items with value "0".

Appendix D

Performances of proLIBLINEAR on RCV1-v3 in top-down manner

Topic	#Total	#Correct	#Wrong	Precision	Recall	F1
CCAT	370541	342270	23962	0.934	0.923	0.929
C11	23651	6718	4797	0.583	0.284	0.382
C12	11563	6429	1372	0.824	0.555	0.664
C13	36463	14879	9306	0.615	0.408	0.490
C14	7250	3510	1983	0.638	0.484	0.550
C15	147606	130299	9205	0.934	0.882	0.907
C151	79524	68933	6216	0.917	0.866	0.891
C1511	22812	10642	3228	0.767	0.466	0.580
C152	71162	55053	12195	0.818	0.773	0.795
C16	1871	488	104	0.824	0.260	0.396
C17	40983	28872	6818	0.808	0.704	0.753
C171	17876	11799	3416	0.775	0.660	0.713
C172	11202	8513	1945	0.814	0.759	0.786
C173	2560	1272	559	0.694	0.496	0.579
C174	5625	4815	576	0.893	0.856	0.874
C18	51355	38447	7218	0.841	0.748	0.792
C181	42169	31601	8972	0.778	0.749	0.763
C182	4529	1088	657	0.623	0.240	0.346
C183	7204	3798	1125	0.771	0.527	0.626
C21	24610	9279	3531	0.724	0.377	0.495
C22	5929	1521	1175	0.564	0.256	0.352
C23	2563	866	699	0.553	0.337	0.419

C24	31231	14500	5605	0.721	0.464	0.564
C31	39451	17733	8305	0.681	0.449	0.541
C311	4133	2059	1666	0.552	0.498	0.524
C312	6452	2564	2558	0.500	0.397	0.443
C313	1074	7	14	0.333	0.006	0.012
C32	2041	222	61	0.784	0.108	0.191
C33	14889	7089	3118	0.694	0.476	0.564
C331	1179	619	383	0.617	0.525	0.567
C34	4715	1237	300	0.804	0.262	0.395
C41	11043	7953	981	0.890	0.720	0.796
C411	9986	7800	1108	0.875	0.781	0.825
C42	11535	7345	1822	0.801	0.636	0.709
ECAT	116471	86750	13576	0.864	0.744	0.800
E11	8289	3925	1530	0.719	0.473	0.571
E12	26421	13555	6780	0.666	0.513	0.579
E121	2088	1397	403	0.776	0.669	0.718
E13	6416	4687	980	0.827	0.730	0.775
E131	5492	3906	1060	0.786	0.711	0.746
E132	922	518	98	0.840	0.561	0.673
E14	2112	970	361	0.728	0.459	0.563
E141	364	72	21	0.774	0.197	0.315
E142	192	38	33	0.535	0.197	0.288
E143	1172	706	253	0.736	0.602	0.662
E21	41875	30221	5095	0.855	0.721	0.783
E211	15361	9333	3366	0.734	0.607	0.665
E212	26552	20025	2809	0.876	0.754	0.810
E31	2349	1172	158	0.881	0.498	0.637
E311	1658	950	164	0.852	0.572	0.685
E312	52	0	0	0.0	0.0	0.0
E313	108	2	11	0.153	0.018	0.033
E41	16586	10245	2287	0.817	0.617	0.703
E411	2096	752	330	0.695	0.358	0.473
E51	20639	10515	4455	0.702	0.509	0.590
E511	2831	1147	749	0.604	0.405	0.485
E512	12234	6567	3010	0.685	0.536	0.602
E513	2236	1232	165	0.881	0.550	0.678
E61	376	92	96	0.489	0.244	0.326

E71	5102	4715	73	0.984	0.924	0.953
GCAT	232297	212977	13843	0.938	0.916	0.927
G15	20309	16124	2195	0.880	0.793	0.834
G151	3258	4	2	0.666	0.001	0.002
G152	2072	0	0	0.0	0.0	0.0
G153	2301	1446	684	0.678	0.628	0.652
G154	8266	6076	1168	0.838	0.735	0.783
G155	2086	7	118	0.056	0.003	0.006
G156	258	0	0	0.0	0.0	0.0
G157	1991	212	23	0.902	0.106	0.190
G158	4248	2417	1060	0.695	0.568	0.625
G159	38	0	0	0.0	0.0	0.0
GCRIM	31086	22335	4262	0.839	0.718	0.774
GDEF	8609	3536	993	0.780	0.410	0.538
GDIP	36735	23245	5560	0.806	0.632	0.709
GDIS	8364	5533	999	0.847	0.661	0.742
GENT	3695	1575	371	0.809	0.426	0.558
GENV	6089	2308	688	0.770	0.379	0.508
GFAS	307	2	1	0.666	0.006	0.012
GHEA	5833	3331	1500	0.689	0.571	0.624
GJOB	16770	11326	2038	0.847	0.675	0.751
GMIL	5	0	0	0.0	0.0	0.0
GOBIT	831	8	18	0.307	0.009	0.018
GODD	2712	126	101	0.555	0.046	0.085
GPOL	55231	37641	11676	0.763	0.681	0.720
GPRO	5332	681	228	0.749	0.127	0.218
GREL	2757	651	38	0.944	0.236	0.377
GSCI	2373	1436	340	0.808	0.605	0.692
GSPO	34404	33301	600	0.982	0.967	0.975
GTOUR	657	281	107	0.724	0.427	0.537
GVIO	31500	19661	3719	0.840	0.624	0.716
GVOTE	11186	5111	1192	0.810	0.456	0.584
GWEA	3743	1762	247	0.877	0.470	0.612
GWELF	1818	369	102	0.783	0.202	0.322
MCAT	198938	179045	11076	0.941	0.900	0.920
M11	47402	41844	4219	0.908	0.882	0.895
M12	25304	19822	3123	0.863	0.783	0.821

M13	52038	43634	4242	0.911	0.838	0.873
M131	27242	22852	3289	0.874	0.838	0.856
M132	26053	20121	3080	0.867	0.772	0.817
M14	82899	74278	4525	0.942	0.896	0.918
M141	46200	43104	3346	0.927	0.932	0.930
M142	11819	9342	899	0.912	0.790	0.846
M143	21351	18294	963	0.949	0.856	0.901
Micro				0.876	0.770	0.819
Macro				0.722	0.512	0.575

Appendix E

Performances for LIR on RCV1-v3

Topic	#Total	#Correct	#Wrong	Precision	Recall	F1
CCAT	370541	339780	18601	0.948	0.916	0.932
C11	23651	7745	5200	0.598	0.327	0.423
C12	11563	6494	820	0.887	0.561	0.688
C13	36463	15396	6538	0.701	0.422	0.527
C14	7250	3571	1686	0.679	0.492	0.571
C15	147606	130802	6950	0.949	0.886	0.916
C151	79524	68495	2991	0.958	0.861	0.907
C1511	22812	11799	3793	0.756	0.517	0.614
C152	71162	54543	4823	0.918	0.766	0.835
C16	1871	624	250	0.713	0.333	0.454
C17	40983	29286	3956	0.880	0.714	0.789
C171	17876	11757	476	0.961	0.657	0.780
C172	11202	8928	353	0.961	0.797	0.871
C173	2560	1438	301	0.826	0.561	0.668
C174	5625	4880	164	0.967	0.867	0.914
C18	51355	40019	7312	0.845	0.779	0.811
C181	42169	32314	2318	0.933	0.766	0.841
C182	4529	1190	411	0.743	0.262	0.388
C183	7204	3831	436	0.897	0.531	0.667
C21	24610	9288	2636	0.778	0.377	0.508
C22	5929	2006	1702	0.540	0.338	0.416
C23	2563	845	345	0.710	0.329	0.450
C24	31231	16407	5786	0.739	0.525	0.614
C31	39451	19326	6005	0.762	0.489	0.596

C311	4133	2023	733	0.734	0.489	0.587
C312	6452	2388	701	0.773	0.370	0.500
C313	1074	30	48	0.384	0.027	0.052
C32	2041	320	69	0.822	0.156	0.263
C33	14889	7447	2764	0.729	0.500	0.593
C331	1179	670	309	0.684	0.568	0.620
C34	4715	1248	296	0.808	0.264	0.398
C41	11043	8280	972	0.894	0.749	0.815
C411	9986	8071	199	0.975	0.808	0.884
C42	11535	7567	1000	0.883	0.656	0.752
ECAT	116471	87230	11635	0.882	0.748	0.810
E11	8289	3842	997	0.793	0.463	0.585
E12	26421	13294	2915	0.820	0.503	0.623
E121	2088	1471	89	0.942	0.704	0.806
E13	6416	4600	533	0.896	0.716	0.796
E131	5492	3766	194	0.951	0.685	0.796
E132	922	570	63	0.900	0.618	0.733
E14	2112	1017	238	0.810	0.481	0.604
E141	364	85	16	0.841	0.233	0.365
E142	192	48	13	0.786	0.25	0.379
E143	1172	767	22	0.972	0.654	0.782
E21	41875	30195	1635	0.948	0.721	0.819
E211	15361	9269	695	0.930	0.603	0.732
E212	26552	19880	385	0.981	0.748	0.849
E31	2349	1286	132	0.906	0.547	0.682
E311	1658	1017	91	0.917	0.613	0.735
E312	52	0	0	0.0	0.0	0.0
E313	108	0	2	0.0	0.0	0.0
E41	16586	10565	912	0.920	0.636	0.752
E411	2096	551	241	0.695	0.262	0.381
E51	20639	10533	1421	0.881	0.510	0.646
E511	2831	1179	222	0.841	0.416	0.557
E512	12234	6614	521	0.926	0.540	0.682
E513	2236	1307	18	0.986	0.584	0.734
E61	376	112	29	0.794	0.297	0.433
E71	5102	4746	66	0.986	0.930	0.957
GCAT	232297	212185	12507	0.944	0.913	0.928

G15	20309	16059	936	0.944	0.790	0.860
G151	3258	32	46	0.410	0.009	0.019
G152	2072	120	120	0.5	0.057	0.103
G153	2301	1435	246	0.853	0.623	0.720
G154	8266	6241	385	0.941	0.755	0.838
G155	2086	179	138	0.564	0.085	0.148
G156	258	0	0	0.0	0.0	0.0
G157	1991	815	178	0.820	0.409	0.546
G158	4248	2727	1924	0.586	0.641	0.612
G159	38	0	0	0.0	0.0	0.0
GCRIM	31086	21977	3447	0.864	0.706	0.777
GDEF	8609	3321	692	0.827	0.385	0.526
GDIP	36735	24082	4705	0.836	0.655	0.735
GDIS	8364	5919	1067	0.847	0.707	0.771
GENT	3695	1808	501	0.783	0.489	0.602
GENV	6089	2462	682	0.783	0.404	0.533
GFAS	307	1	0	1.0	0.003	0.006
GHEA	5833	3033	563	0.843	0.519	0.643
GJOB	16770	11132	817	0.931	0.663	0.775
GMIL	5	0	0	0.0	0.0	0.0
GOBIT	831	7	12	0.368	0.008	0.016
GODD	2712	302	283	0.516	0.111	0.183
GPOL	55231	38108	8590	0.816	0.689	0.747
GPRO	5332	1235	633	0.661	0.231	0.343
GREL	2757	733	59	0.925	0.265	0.413
GSCI	2373	1325	169	0.886	0.558	0.685
GSPO	34404	33547	577	0.983	0.975	0.979
GTOUR	657	303	81	0.789	0.461	0.582
GVIO	31500	20639	4324	0.826	0.655	0.731
GVOTE	11186	5670	1608	0.779	0.506	0.614
GWEA	3743	2278	216	0.913	0.608	0.730
GWELF	1818	304	41	0.881	0.167	0.281
MCAT	198938	179725	10422	0.945	0.903	0.923
M11	47402	42052	649	0.984	0.887	0.933
M12	25304	20147	1380	0.935	0.796	0.860
M13	52038	43797	1493	0.967	0.841	0.899
M131	27242	22671	825	0.964	0.832	0.893

M132	26053	19924	652	0.968	0.764	0.854
M14	82899	74910	598	0.992	0.903	0.945
M141	46200	43211	631	0.985	0.935	0.959
M142	11819	9681	91	0.990	0.819	0.896
M143	21351	18353	150	0.991	0.859	0.921
Micro				0.918	0.776	0.841
Macro				0.796	0.532	0.616