Abstract of "Multilabel Classification over Category Taxonomies" by Lijuan Cai, Ph.D., Brown University, May 2008.

Multilabel classification is the task of assigning a pattern to one or more classes or categories from a pre-defined set of classes. It is a crucial tool in knowledge and content management. Standard machine learning techniques such as Support Vector Machines (SVMs) and Perceptron have been successfully applied to this task. However, many real-world classification problems involve large numbers of overlapping categories that are arranged in a hierarchy or taxonomy. This poses a challenge to learning algorithms as they ignore the class hierarchies thereby losing valuable information.

In this thesis, we propose to systematically incorporate prior knowledge on category taxonomy directly into the learning architecture. We present two methods, hierarchical SVM learning and hierarchical Perceptron learning. Both methods take a ranking view of the multilabel problem by focusing on ranking category relevances. In the hierarchical SVM, the hierarchical learning problem is expressed as a joint large margin formulation that simultaneously learns the discriminant functions of each class. As the resulting optimization problem can be prohibitively large, we also present a variable selection algorithm to efficiently solve it. In the hierarchical Perceptron method, the construction of weight vectors and the update rule are made to capture the category taxonomy. Both methods can leverage kernel techniques, work with arbitrary directed acyclic graph taxonomy, and be applied to general settings where categories can be characterized by attributes. We also present an automatic approach to learn a taxonomy if one isn't available. Our approach is adapted from the hierarchical agglomerative clustering algorithm. The learned hierarchy can then be used in existing hierarchical classification approaches. Extensive experiments demonstrate the performance advantage of our approaches.

Multilabel Classification over Category Taxonomies

by

Lijuan Cai
B. Eng., Computer Science and Engineering, Nanjing University of Aeronautics and Astronautics, 1997
M. Eng., Computer Science, Nanjing University, 2000
M. Sc., Computer Science, Brown University, 2003

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2008

This dissertation by Lijuan Cai is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date ______________ ______________________________
Thomas Hofmann, Director

Recommended to the Graduate Council

Date ______________ ______________________________
Chad Jenkins, Reader

Date ______________ ______________________________
Gregory Shakhnarovich, Reader

Approved by the Graduate Council

Date ______________ ______________________________
Sheila Bonde
Dean of the Graduate School

Acknowledgements

To begin, I would like to give my deepest gratitude to Thomas Hofmann, my advisor. He introduced me to the main topic of the dissertation and advised me on most of my research work. I have been constantly amazed by his vision, knowledge, and energy. His support over all these years in academics and research has been an invaluable resource for me.

I am profoundly indebted to Gregory Shakhnarovich, who has worked with me closely for over a year. He has provided me countless guidance and encouragement, both in research and in many aspects of life. I have been impressed by his intuition, insight, and depth of thinking.

I am very grateful to Chad Jenkins for valuable discussion and advice. He has been very approachable and motivating.

It has been a great pleasure for me to study in the Computer Science department of Brown University. I thank the faculty for offering an excellent and friendly intellectual environment. I would like to especially thank Tom Doeppner, Amy Greenwald, Meinolf Sellmann, John Hughes, Eli Upfal, Roberto Tamassia, and Andy van Dam. Tom and Amy have helped me a great deal in my job search. I also thank the technical and administrative staff for their help.

I wish to thank all my colleagues and friends, in particular Joel Young, Stuart Andrews, Massimiliano Ciaramita, Ioannis Tsochantaridis, Yasemin Altun, David Gondek, Ying Xing, Jue Yan, Ye Sun, Zheng Li, Olga Karpenko, Song Zhang, Ya Jin, Danfeng Yao, Micha Elsner, Xiaofei He, Lili Ma, and Mengqiao Xu.

Finally I want to specially thank my father Jinchang Cai, my mother Xingdi Yan, my sister Limin Cai, my husband Thomas Liang, and my son Dustin Liang. They have supported me immensely all the way along. To me, their love makes this world beautiful and warm. This dissertation is dedicated to my grandmother who deceased

two decades ago. Her unconditional love in my childhood had shaped me and has always inspired me.

Contents

List of Tables

List of Figures

List of Notation

$\mathbf{x}$	input pattern	8
$\mathcal{X}$	input space	8
$\mathcal{D}$	training set	13
n	number of trainig examples	13
$\mathcal{Y}$	set of all categories	13
q	the number of classes	13
$\mathrm{P}(\mathcal{Y})$	power set of $\mathcal{Y}$	13
y	a category	13
Y	set of classes	13
F	discriminant function	14
g	ranking function from $\mathcal{X}$ to S_q	14
S_q	the set of permutations of ranks 1 to q	14
$\mathcal{V}$	set of all nodes in a directed acyclic graph	14
$\boldsymbol{\Lambda}(y)$	attribute vector for class y	15
s	length of class attribute vectors. $\boldsymbol{\Lambda}(y) \in \mathbb{R}^s$	15
$\mathbf{w}_r$	weight vector associated with r-th attribute	15
$\mathbf{w}$	stacked weight vector	15
$\boldsymbol{\Phi}(\mathbf{x}, y)$	joint feature representation	15
$\mathrm{anc}(y)$	set of all ancestor nodes of y, including y itself	16
$\mathbf{W}_y$	functioning weight vector of class y	17
$\mathrm{anc}(Y)$	set of all ancestor nodes of classes in Y	18
$\triangle(Y, \hat{Y})$	loss between two sets of classes	18
$\triangle(y, y')$	loss between two classes	19
Y_i	the set of correct categories for instance i	21
$\bar{Y}_i$	the set of negative categories for instance i	21

$\delta\mathbf{\Phi}_i(y,\bar{y})$	$\mathbf{\Phi}(\mathbf{x}_i,y)-\mathbf{\Phi}(\mathbf{x}_i,\bar{y})$ 24
$\alpha_{iy\bar{y}}$	dual variable associated with instance i, a positive class y, and a negative class $\bar{y}$ 27
$\Theta(\boldsymbol{\alpha})$	dual objective 27
$\triangle^x$	maximal loss 28
$\triangle^t$	top loss 29
$H_{iy\bar{y}}$	an auxiliary variable used for variable selection 31
ψ_i	the quantity used for selecting subspace to optimize 32
$\mathcal{O}$	the set of interior nodes 35
l	the number of interior nodes 35
$\mathbf{A}$	a matrix that is determined by taxonomy and class attributes 36
$\mathbf{u}$	a vector function of class weight vectors 37

Chapter 1

Introduction

1.1 The problem

Pattern classification is the task of assigning *patterns*, or *instances*, to a predefined set of *classes*, or *categories*. It covers a broad range of applications such as document classification, optical character recognition, and image recognition. Relevant application domains include information retrieval, natural language processing, computer vision, and computational biology, where categories may represent concepts as diverse as document categories, word senses, visual object classes, or protein functions. Take document classification, a classical pattern classification problem, as an example. The classes are generally *topics* discussed in documents while patterns or instances refer to documents. Document classification is a crucial and well-proven instrument for organizing large volumes of textual information. Comprehensive classification systems have been developed and maintained by librarians since the 19th century and are in widespread use today. The advent of the Web and the enormous growth of digital content in intranets, databases, and archives, have further increased the demand for categorization. In the face of the pace and complexity of this process, manual categorization often lacks economic efficiency and automatic tools are indispensable to supplement human efforts.

In most cases, the use of statistical or machine learning techniques has been proven to be successful in this context, since it is typically more feasible to induce categorization rules based on example documents than to elicit such rules from domain experts.

The wide range of methods applied to this problem include nearest neighbor classifiers [72], neural networks [56, 44, 66], generative probabilistic classifiers [36, 31, 35], and – more recently – boosting [55] and Support Vector Machines (SVMs) [29], to name just a few. Extensive experimental comparisons (e.g. [29, 73, 6]) have evidenced that among the methods available today, SVMs are among the best in their classification accuracy and can therefore be considered the state-of-the art in pattern classification.

A potential drawback of traditional classification methods is that they treat the category structure as 'flat' and that they do not consider relationships between categories, which are commonly expressed in concept *hierarchies* or *taxonomies*. Such structures, however, are the preferred way in which concepts, subject headings, or categories are arranged in practice. Taxonomies offer clear advantages in supporting tasks like browsing, searching, or visualization. They are also easier to maintain and alleviate the manual annotation process. This is witnessed by the fact that many real world classification systems have complex hierarchical structures. This includes traditional systems like Dewey or Library of Congress subject headings, the International Patent Classification (IPC) scheme (approx. 69,000 patent groups) [68], the Medical Subject Headings (MeSH) [43] maintained by NIH, the Gene Ontology (approx. 17,000 terms to describe gene products) [13], as well as Web catalogs created by Yahoo! [70], the Open Directory Project (DMOZ) (approx. 590,000 categories for Web pages) [47] or LookSmart, to name some of the most important ones.

In many cases, since categories are often not mutually exclusive, instances are assigned to more than one category. This leads to large scale *multilabel* classification problems. The categories are typically organized in *hierarchies* or *taxonomies*, most commonly by introducing superordinate concepts and by relating categories via 'is-a' relationships. Multiply connected taxonomies are not uncommon in this context. An example of class hierarchies is given in Figure 1.1. It depicts a small part of the taxonomy over IPC categories. A patent document can belong to multiple classes. For example, a document with title "method and apparatus for continuous cross-channel interleaving" is assigned to both class `H 04 K 001` and class `H 04 L 001`.

We believe that taxonomies encode valuable domain knowledge, which learning methods should be able to capitalize on, in particular since the number of training

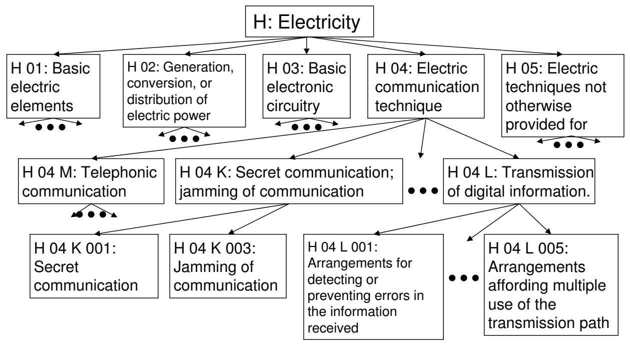

H: Electricity
H 01: Basic electric elements
H 02: Generation, conversion, or distribution of electric power
H 03: Basic electronic circuitry
H 04: Electric communication technique
H 05: Electric techniques not otherwise provided for
H 04 M: Telephonic communication
H 04 K: Secret communication; jamming of communication
H 04 L: Transmission of digital information.
H 04 K 001: Secret communication
H 04 K 003: Jamming of communication
H 04 L 001: Arrangements for detecting or preventing errors in the information received
H 04 L 005: Arrangements affording multiple use of the transmission path

Figure 1.1: Part of the IPC classification hierarchy.

examples for individual classes may be very small when dealing with tens of thousands or more classes. In many applications, the training data can be expensive to collect. This is sometimes due to the nature of applications (e.g. costly biological experiments or highly-trained experts are needed), and sometimes due to the general observation that the labeling process becomes harder as the number of classes grows and the classes become more and more specific. It is therefore important for automatic classification system to take advantage of all information available. Hierarchy is one valuable source of domain knowledge. The potential loss of useful information by ignoring class hierarchies has been pointed out before and has led to a number of approaches that employ different ways to exploit hierarchies [41, 65, 23].

One intuitive way to exploit hierarchy is to use a divide-and-conquer strategy [31, 23, 52, 10]. Usually one or more local classifiers are trained *independently* for each node in a taxonomy. During evaluation, instances are first classified on the top level and then successively lower levels in the taxonomy. The outputs of these local classifiers are combined to form predictions. This is a special case of output coding [21]. The main disadvantage of divide-and-conquer methods is that the local classifiers are combined in a particular way (e.g. greedy decision or Bayes-optimal label assignment) that is not reflected in the learning of the classifiers.

1.2 Our approach

In our work, we take the so-called big bang approach [59], in which a single classifier is learned with all its parameters fitted together. Thus the way our classifier is learned is consistent with the way they are used and a more accurate discrimination may be derived.

In this thesis, we present a new approach for systematically incorporating domain knowledge about the relationships between categories into the Perceptron learning and SVM classification architecture. The work has been published in [26], [4], and [5]. Our approach is based on two ways of encoding class taxonomy. First, we view the hierarchy as a way to naturally decompose the classifier parameters, which in our case are the *prototypes, or weight vectors* of each class. The prototype of a class is composed of contributions from all its ancestors in a taxonomy. Through this we tie the learning across categories. The motivation is that by pooling together instances of individual classes for parameter estimation at higher levels of a taxonomy, a more robust classifier will be obtained. This in spirit is very similar to the hierarchical shrinkage model [41] in which the parameters that are decomposed are class-conditional word probabilities. Second, we adapt the standard 0/1 loss function to weigh misclassification errors in accordance with the taxonomy structure. By considering the degree of "proximity" among categories, the hierarchical loss is more precise in capturing the user experience and hence the performance of classification systems. For example, given a document about `soccer game`, a misclassification into `volleyball game` is more tolerable than that into `Latin music`. In addition, if the automatic classification is to aid humans in labeling instances, a low hierarchical loss means that it is easier for humans to find the correct class.

We then incorporate the taxonomy encodings into two learning algorithms: Support Vector Machines (SVM) and Perceptron, yielding the two new algorithms of hierarchical SVM and hierarchical Perceptron. In multi-label classification, a pattern can be assigned to more than one category. We take a ranking view of the learning problem, i.e. the focus is to rank the category relevance correctly. In hierarchical SVM, the learning problem is formulated as a joint large margin problem that learns the discriminant functions for each class at the same time. Unlike in the flat SVM, these discriminant functions are coupled. The objective function also includes a term that upper bounds a taxonomy-based loss. The resulting dual program can be quite large. We thus propose a variable selection algorithm to efficiently solve it. To our best knowledge our work is the first contribution for SVM-based hierarchical categorization that is not based on a greedy, decision tree-like classification scheme, but rather optimizes a common objective jointly over all parameters. In Perceptron learning, the construction of weight vectors and the update rule are modified to reflect the taxonomy. Both the hierarchical SVM and the hierarchical Perceptron work with any taxonomy that can be represented as a Directed Acyclic Graph. Furthermore, they can both take advantage of the power of kernel functions. The experiments have shown the merits of our hierarchical approach.

It has been shown in our work and many others that exploiting given taxonomies can improve the classification accuracy. One question to then ask is whether flat classification is our only choice if no taxonomy is present. It is reasonable to assume there is still an underlying structure over the classes. We thus propose an algorithm to automatically generate a hierarchy from training data, which is then used in existing hierarchical classification methods. The hierarchy generation is modified from agglomerative clustering algorithm and produces a two-level hierarchy. Our experiments have shown that it improved classification performance for our hierarchical Perceptron and the Hieron algorithm [19].

In summary, our major contributions are

1. Two methods to represent taxonomy knowledge. One is to directly encode structure in the scoring function used to rank categories. The other is a taxonomy-based loss function between categories that is motivated by real applications.

2. Incorporating the taxonomy representations into two learning architectures, leading to our hierarchical SVM algorithm and hierarchical Perceptron algorithm. We developed an efficient variable selection algorithm for the optimization problem in the hierarchical SVM.

3. An algorithm to automatically learn a hierarchy that can then be used with our hierarchical algorithms or with many other hierarchical methods.

4. Empirical evaluation on several datasets with both standard metrics and a few hierarchical ones we proposed. We have also run hierarchical algorithms with

random hierarchies to investigate the impact of hierarchy and the stability of classification algorithms.

1.3 Thesis organization

The rest of the thesis is organized as follows. We outline the general background on SVM and Perceptron in Chapter 2. In Chapter 3 we formalize our learning problem and introduce the two ways of representing taxonomy knowledge. In Chapter 4 we formulate the hierarchical SVM learning problem in terms of a joint large margin problem, for which we derive an efficient training algorithm. In Chapter 5 we propose a hierarchical Perceptron algorithm that exploits taxonomies in a similar fashion. The algorithm for automatic hierarchy learning is presented in Chapter 6. In Chapter 7 we discuss the experimental results. The conclusions are in Chapter 8.

Chapter 2

Background

This chapter outlines the Perceptron and Support Vector Machine (SVM) algorithm that the thesis work is based on. The Perceptron algorithm goes back to Rosenblatt in 1950s and still is an active research area. SVMs, introduced in early 1990s, have been a significant advance in machine learning research. The Perceptron is known for its simplicity and speed while SVMs are known for their performance and theoretic guarantees. The majority of work on Perceptron and SVM has focused on *binary classification*, in which there are only two possible classes. A growing interest in *multiclass* and *multilabel* classification has been witnessed in recent years. We use multiclass classification to refer to the scenario in which there are multiple classes and each pattern is relevant with *exactly one* class. We use multilabel classification for the scenario in which there are multiple classes and each pattern can be relevant with *one or more* classes. Some literature uses different terms for these two scenarios, such as single-label multiclass classification and multi-label multiclass classification respectively. However, this thesis will use the terms of multiclass and multilabel classifications.

The Perceptron and SVM learning for both binary and multiclass classification will be described in this chapter. The (flat) multilabel algorithms will be introduced in Sections 4.1 and 5.1 where more context is provided. Note the multiclass algorithms are special cases of their respective multilabel algorithms.

2.1 Binary classification

2.1.1 Problem setting

We denote an instance, such as a document, by $\mathbf{x} \in \mathcal{X}$, where $\mathcal{X}$ is the input space. We employ the popular vector space model [53] and assume $\mathcal{X} \subseteq \mathbb{R}^d$. We denote the set of all classes by $\mathcal{Y}$ and a single class by y. In case of binary classification $\mathcal{Y} = \{-1, 1\}$. Pattern classification is one type of *supervised learning*, in which a labeled training set (or sample) is given. Let $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ denote the training set for binary classification. It is assumed that $(\mathbf{x}_i, y_i)$ pairs are drawn i.i.d. from an unknown but fixed distribution $p(\mathbf{x}, y)$ over $\mathcal{X} \times \mathcal{Y}$. The *discriminative learning* methods, which both Perceptron and SVMs are, aim at solving classification directly. That is, to learn a mapping from the input space to the output space, i.e. $f : \mathcal{X} \to \mathcal{Y}$. The mapping is called *classifier*. It is typical to define a classifier via a linear discriminant function $F : \mathcal{X} \to \mathbb{R}$ that is linear in $\mathbf{x}$. The score that a discriminant function predicts for an instance indicates the confidence of assigning the instance to class 1. The classification rule then is to assign an instance to class 1 if the discriminant function predicts a score that is greater than 0 and to class -1 otherwise. Both the binary Perceptron and SVMs use discriminant function and classification function defined as

$$F(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b \quad \text{and} \quad f(\mathbf{x}) = \text{sign}(F(\mathbf{x})) \tag{2.1}$$

respectively. The *hyperplane* defined by $\mathbf{w}$ and b, i.e. $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$, separates the input space into the positive halfspace and the negative halfspace.

2.1.2 Binary Perceptron

Rosenblatt's Perceptron [49] will be introduced in this section. The algorithm initializes $\mathbf{w}$ and b to be zero. It then cycles through all training instances repeatedly, one at a time, until all instances are correctly classified. At each iteration, for current instance class pair $(\mathbf{x}_i, y_i)$, the algorithm predicts its class with the current classifier:

$$\hat{y}_i = f(\mathbf{x}_i) = \text{sign}(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\,. \tag{2.2}$$

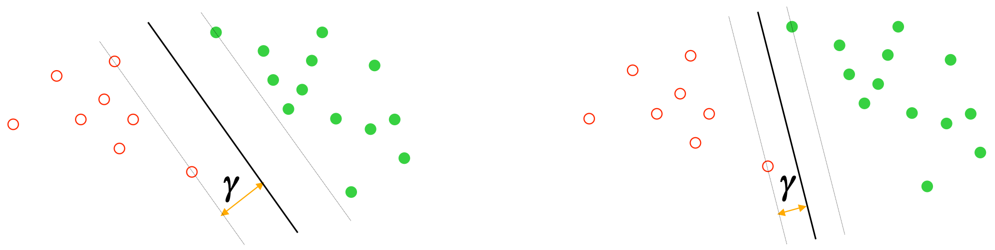


Figure 2.1: The solid and empty circles represent positive and negative instances respectively. The solid lines represent hyperplanes. The figure on left shows the maximum-margin separating hyperplane while the figure on right shows another separating hyperplane on the same data. The grey lines are used to emphasize the margin.

If $y_i \neq \hat{y}_i$, i.e. a mistake is made, then an update step is performed on the classifier:

$$\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i, \qquad b \leftarrow b + y_i \tag{2.3}$$

The step moves the discriminant function value of $\mathbf{x}_i$ towards the correct direction of y_i. A training set is called *linearly separable* if there exists a linear classifier that can classify all training instances correctly, i.e. $\exists\ \mathbf{w}^*$ s.t. $y_i(\langle \mathbf{w}^*, \mathbf{x} \rangle + b) > 0$ for $i = 1, \ldots, n$. The Perceptron algorithm is guaranteed to converge with every instance correctly classified if the training data is linearly separable [45].

2.1.3 Binary SVMs

In contrary to Perceptron, in which it suffices to find any hyperplane that separates the training data correctly, the goal of SVMs is to find a maximum-margin separating hyperplane. Figure 2.1 shows an example of maximum-margin separating hyperplane, which is the hyperplane that has the largest distance, or *margin*, to the training set. The margin of a hyperplane with respect to a data set is defined by

$$\gamma = \min_i \frac{y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)}{\|\mathbf{w}\|}, \tag{2.4}$$

where $\|\mathbf{w}\| = \sqrt{\mathbf{w}^T \mathbf{w}}$ is the two-norm of $\mathbf{w}$. Note scaling $\mathbf{w}$ and b with the same non-zero constant does not change the value of γ. Without the normalization term $\|\mathbf{w}\|$, the margin defined in Equation (2.4) could be made arbitrarily large. SVMs are derived from the principle of *risk minimization*. A small error (or *risk*) on the training data is no guarantee that the classifier will generalize well on unseen data.

There are a number of generalization bounds (e.g. in [17] and [64]) which show that assuming all other variables remain the same, with high probability the larger the margin is, the smaller risk bound that classifier will have on the whole population of data. Thus the goal of SVMs is to maximize γ in Equation (2.4). After several steps of equivalent conversions, this problem takes the standard norm minimization form of SVM.

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{2.5a}$$

$$\text{s.t. } y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \quad \forall i \tag{2.5b}$$

If the training data is not linearly separable, there are also soft margin SVMs to deal with it. Here we formulate the 1-norm soft margin SVM as below.

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \tag{2.6a}$$

$$\text{s.t. } y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i,\ \xi_i \geq 0 \quad \forall i \tag{2.6b}$$

The slack variables are denoted by ξ_i and C is a given constant controlling the trade-off between maximizing the margin and minimizing the training error. The margin maximization and kernel techniques (which will be briefly presented in Section 4.2.2) make SVMs one of the most powerful classification algorithms. Empirical results have shown leading performance of SVMs on a large number of tasks.

2.2 Multiclass classification

2.2.1 Problem setting

The goal of multiclass classification is to map an instance to a class from a set of q classes. Let $\mathcal{Y} = \{1, 2, \ldots, q\}$. There are essentially two types of philosophies in addressing the multiclass problem. One is to combine multiple binary learning algorithms into a multiclass classifier. The other is to directly generalize binary learning algorithms to deal with multiple classes. The algorithms introduced here are of the second type.

Let us introduce a weight vector $\mathbf{w}_y$ for each class $1 \leq y \leq q$. Following [14]

Algorithm 1 Multiclass Perceptron

```
Inputs: training data $\{\mathbf{x}_i, y_i\}_{i=1}^n$.
 1: Initialize $\mathbf{w}_y \leftarrow 0, \forall\, y$.
 2: repeat
 3:   for all training instance $i$ do
 4:     $\hat{y}_i = \mathrm{argmax}_{y=1}^q \langle \mathbf{w}_y, \mathbf{x} \rangle$
 5:     if $\hat{y}_i \neq y_i$ then
 6:       $\mathbf{w}_{y_i} \leftarrow \mathbf{w}_{y_i} + \mathbf{x}_i$.
 7:       $\mathbf{w}_{\hat{y}_i} \leftarrow \mathbf{w}_{\hat{y}_i} - \mathbf{x}_i$.
 8:     end if
 9:   end for
10: until all instances are correctly predicted
```

and [16], the discriminant function takes the form of $F_y(\mathbf{x}) = \langle \mathbf{w}_y, \mathbf{x} \rangle$ and the prediction is made using the following Winner-Take-All rule.

$$f(\mathbf{x}) = \operatorname*{argmax}_{y=1}^{q} \langle \mathbf{w}_y, \mathbf{x} \rangle \tag{2.7}$$

One can view $\mathbf{w}_y$ as the *prototype* of class y. Thus the inner product between $\mathbf{w}_y$ and an instance $\mathbf{x}$ measures the similarity between the class and the instance. The classification rule in Equation (2.7) then predicts the class that achieves the highest similarity score with an instance. If introducing a bias term b_y for each class, one could arrive at a more general form of linear discriminant: $F'_y(\mathbf{x}) = \langle \mathbf{w}_y, \mathbf{x} \rangle + b_y$. However, this makes solving the optimization problem in multiclass SVM considerably more difficult. Moreover, the bias terms can always be approximated by adding an additional feature to all instances with the feature value always being 1.

2.2.2 Multiclass Perceptron

In [16] a family of multiclass online algorithms that generalize binary Perceptron have been proposed. Here we introduce the most straightforward one, which is also the foundation of our hierarchical Perceptron algorithm.

The multiclass Perceptron algorithm is described in Algorithm 1. The key change from binary Perceptron is the update rule. In the multiclass Perceptron, if the predicted class is wrong, then the weight vector of the correct class is moved towards the current instance, while that of the predicted class is moved away from the current instance. One can show that the multiclass Perceptron in Algorithm 1 will converge if the training data can be separated by a linear classifier of the form in Equation (2.7).

2.2.3 Multiclass SVM

There are multiple formulations that generalize binary SVMs to handle the multiclass classification. Here we outline the one in [14]. The (functional) margin in the multiclass setting is defined as

$$\gamma \equiv \min_i \gamma_i \text{ where } \gamma_i \equiv \langle \mathbf{w}_{y_i}, \mathbf{x}_i \rangle - \max_{y \neq y_i} \langle \mathbf{w}_y, \mathbf{x}_i \rangle \tag{2.8}$$

The margin for each instance is defined as the difference between the score of the correct class and the maximal score of any incorrect class. The goal of maximizing the margin leads to

$$\min_{\{\mathbf{w}_y\}_{y=1}^q, \boldsymbol{\xi}} \frac{1}{2} \sum_{y=1}^{q} \|\mathbf{w}_y\|^2 + C \sum_i \xi_i \tag{2.9a}$$

$$\text{s.t. } \langle \mathbf{w}_{y_i}, \mathbf{x}_i \rangle - \max_{y \neq y_i} \langle \mathbf{w}_y, \mathbf{x}_i \rangle \geq 1 - \xi_i \quad \forall i \tag{2.9b}$$

$$\xi_i \geq 0 \quad \forall i \tag{2.9c}$$

The constraints in Equation (2.9b) can be rewritten as a set of linear constraints

$$\langle \mathbf{w}_{y_i}, \mathbf{x}_i \rangle - \langle \mathbf{w}_y, \mathbf{x}_i \rangle \geq 1 - \xi_i \quad \forall i, y \neq y_i \tag{2.10}$$

Therefore the multiclass SVM becomes a quadratic program.

Chapter 3

Utilizing known taxonomies

3.1 Problem of multilabel classification over hierarchies

This section formulates the problem of multilabel classification over taxonomies that we address. The same as the multiclass problem setting in Section 2.2.1, an instance is denoted by $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$, the set of all categories by $\mathcal{Y} = \{1, \ldots, q\}$, and a category by $y \in \mathcal{Y}$. Furthermore we denote a label set by $Y \subseteq \mathcal{Y}$ and the *power set* of $\mathcal{Y}$ by $\mathrm{P}(\mathcal{Y}) = \{Y | Y \subseteq \mathcal{Y}\}$. In multilabel classification, a pattern can be relevant with multiple categories. Therefore the training set is denoted by $\mathcal{D} = \{(\mathbf{x}_1, Y_1), (\mathbf{x}_2, Y_2), \ldots, (\mathbf{x}_n, Y_n)\} \subseteq \mathcal{X} \times \mathrm{P}(\mathcal{Y})$.

The majority of methods in literature specify a taxonomy to be a tree structure. In our work we define a taxonomy to be a *directed acyclic graph* (DAG) since DAGs are broader than trees and are commonly used in real life (e.g. Web catalogs created by Yahoo! [70] and by DMOZ [47]). The taxonomies in previous work usually appear in two types: only the leaf nodes (or minimal elements in DAGs) are classes (e.g. [40, 23, 19]); or any node in a taxonomy represents a class (e.g. [50, 10]). The examples of the former type of taxonomies include the IPC scheme [68] and the Enzyme Classification (EC) system [46] while those of the latter type include MeSH [43] and the Yahoo! [70] directory. In this work we specify the classes to be minimal nodes in a DAG. In some cases one wants to express that patterns belong to a super-category, but to none of the terminal categories. we suggest to model this by formally adding one terminal node to each inner node, representing a “miscellaneous” category; this avoids the problem of partial paths.

Formally, a *taxonomy* is a Directed Acyclic Graph $(\mathcal{V}, E)$ with nodes $\mathcal{V} \supseteq \mathcal{Y}$ such that the set of terminal nodes equals $\mathcal{Y}$, formally $\mathcal{Y} = \{y \in \mathcal{V} : \nexists v \in \mathcal{V}, (y, v) \in E\}$. Note that we do not assume that a taxonomy is singly connected (tree or forest), but allow for converging nodes. The directed edges indicate “is-a” relationships between nodes. For example, class `country music` is a type of `music`. So concepts in a taxonomy become increasingly specialized along the directed path from a root to a terminal node.

In multilabel learning, the aim is to find a mapping $f : \mathcal{X} \to \mathrm{P}(\mathcal{Y})$, based on a sample of training pairs in $\mathcal{D}$. A popular approach as suggested, for instance by [54], is to actually learn a ranking function over the categories for each pattern, $g : \mathcal{X} \to S_q$, where S_q is the set of permutations of ranks 1 to q. In order to get a unique subset of labels, one then needs to address the additional question on how to select the number of categories a pattern should be assigned to. This can be achieved, for example, by the set size prediction method presented in [24].

It is common to define the ranking function g implicitly via a scoring function $F : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, such that $g(\mathbf{x})(y) < g(\mathbf{x})(y')$ whenever $F(\mathbf{x}, y) > F(\mathbf{x}, y')$, i.e. categories with higher F-values appear earlier in the ranking (for ease of presentation we ignore ties). Notation $g(y)$ is used when clear from context.

3.2 Class attributes

The same as in the multiclass classification in Section 2.2, we introduce a weight vector $\mathbf{w}_y$ for every class. We will refer to the stacked vector of all weights by $\mathbf{w} = (\mathbf{w}_1^T, \ldots, \mathbf{w}_q^T)^T$. We can then define a linear discriminant function for flat classification

$$F(\mathbf{x}, y; \mathbf{w}) \equiv \langle \mathbf{w}_y, \mathbf{x} \rangle . \tag{3.1}$$

Following [16, 15, 14] we have not included bias terms for categories. One can also introduce explicit bias terms b_y for every class, but this would complicate the presentation and lead to further complications in the optimization algorithm. We thus restrict ourselves to this simpler setting. One way to approximate bias terms is to augment each instance with an extra feature whose value is always 1. The additional dimensions in weight vectors thus approximate bias terms. Another way to view the discriminant function is that $\mathbf{w}_y$ represents the prototype vector of class y and the inner product in Equation (3.1) measures the similarity between a class prototype and an instance. In the multiclass classification, an instance is then assigned to the class with the most similar class prototye (see the Winner-Take-All rule in Equation (2.7)).

We would like to extend Equation (3.1) to cases where classes are not just arbitrary numbers, but can be characterized by attribute vectors $\mathbf{\Lambda}(y) \equiv (\lambda_1(y), \ldots, \lambda_s(y))^T \in \mathbb{R}^s$. This should be carried out in a way that recovers the flat setting as a special case of an orthogonal attribute representation with $s = q$ and $\lambda_r(y) = \delta_{yr}$, i.e. a case where each class is interpreted as a binary attribute of its own. To that extent, we propose to redefine a more general version of the scoring functions F as

$$F(\mathbf{x}, y; \mathbf{w}) \equiv \langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x}, y) \rangle , \tag{3.2}$$

where $\mathbf{w}$ is a weight vector and $\mathbf{\Phi}(\mathbf{x}, y) = \mathbf{\Lambda}(y) \otimes \mathbf{x}$ is the joint feature representation. Here $\otimes$ is the Kronecker product, i.e. $\mathbf{\Phi}(\mathbf{x}, y) \in \mathbb{R}^{d \cdot s}$ is a vector containing all products of coefficients from the first and second vector argument. Writing out $\mathbf{\Phi}(\mathbf{x}, y)$, one gets

$$\mathbf{\Phi}(\mathbf{x}, y) = \begin{pmatrix} \lambda_1(y) \cdot \mathbf{x} \\ \lambda_2(y) \cdot \mathbf{x} \\ \cdots \\ \lambda_s(y) \cdot \mathbf{x} \end{pmatrix} . \tag{3.3}$$

One can interpret $\mathbf{w}$ in Equation (3.2) in terms of a stacked vector of individual weight vectors, i.e. $\mathbf{w} = (\mathbf{w}_1^T, \ldots, \mathbf{w}_s^T)^T$, where $\mathbf{w}_r \in \mathbb{R}^d$ is the weight vector associated with the r-th attribute.

If $\lambda_r(y) = \delta_{ry}$ the joint feature representation simply reduces to

$$\mathbf{\Phi}(\mathbf{x}, y) = \begin{pmatrix} \vdots \\ \mathbf{0} \\ \mathbf{x} \\ \mathbf{0} \\ \vdots \end{pmatrix} \leftarrow \text{occupying the } (d(y-1)+1)\text{-th through } dy\text{-th positions,} \tag{3.4}$$

making

$$\langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x}, y) \rangle = \begin{pmatrix} \vdots \\ \mathbf{w}_{y-1} \\ \mathbf{w}_y \\ \mathbf{w}_{y+1} \\ \vdots \end{pmatrix}^T \begin{pmatrix} \vdots \\ \mathbf{0} \\ \mathbf{x} \\ \mathbf{0} \\ \vdots \end{pmatrix} = \langle \mathbf{w}_y, \mathbf{x} \rangle \,. \tag{3.5}$$

This is conceptually similar to the Kesler construction [22]. Notice when $\lambda_r(y) = \delta_{ry}$ Equation (3.2) indeed reduces to the formulation in Equation (3.1).

It is a straightforward consequence of the linearity of Equation (3.2) to show that one can re-write F as an additive decomposition as follows:

$$F(\mathbf{x}, y; \mathbf{w}) = \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_s \end{pmatrix}^T \begin{pmatrix} \lambda_1(y)\mathbf{x} \\ \vdots \\ \lambda_s(y)\mathbf{x} \end{pmatrix} = \sum_{r=1}^{s} \lambda_r(y) \langle \mathbf{w}_r, \mathbf{x} \rangle \,. \tag{3.6}$$

The general idea is that the notion of class attributes will allow generalization to take place across (similar) categories and not just across training examples belonging to the same category.

How are we going to translate the taxonomy information into attributes for categories? The idea is to treat the nodes in the taxonomy as properties that a subset of categories share. Formally, we define

$$\lambda_v(y) = \begin{cases} t_v(y), & \text{if } v \in \text{anc}(y) \\ 0, & \text{otherwise}\,, \end{cases} \tag{3.7}$$

where $t_v(y) \geq 0$ is the attribute value for category y with respect to node v. We denote by anc(y) the set of ancestor nodes of y in the taxonomy including y itself (for notational convenience). In the simplest case, $t_v(y)$ can be set to a constant, like 1, such that $t_v(y)$ becomes an indicator function. Other choices for $t_v(y)$ are, for example, setting all $t_v(y)$ equal to a constant for nodes v at the same depth in the taxonomy. Equation (3.7) leads to an intuitive decomposition of the scoring function F into contributions from all nodes along the paths from a root node to a specific terminal node. Hence Equation (3.2) becomes

$$F(\mathbf{x}, y; \mathbf{w}) = \sum_{z\colon z \in \text{anc}(y)} \lambda_z(y) \langle \mathbf{w}_z, \mathbf{x} \rangle, \tag{3.8}$$

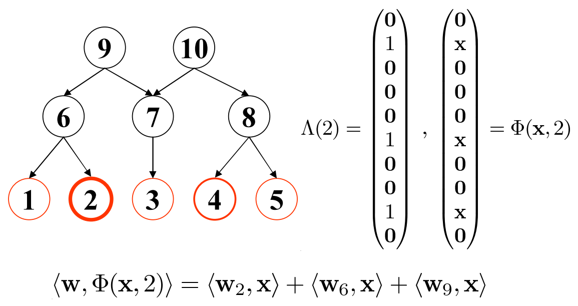

9
10
6
7
8
1
2
3
4
5
$\Lambda(2) = (0,1,0,0,0,1,0,0,1,0)^T$
$(0,x,0,0,0,x,0,0,x,0)^T = \Phi(\mathbf{x},2)$
$\langle \mathbf{w}, \Phi(\mathbf{x},2)\rangle = \langle \mathbf{w}_2,\mathbf{x}\rangle + \langle \mathbf{w}_6,\mathbf{x}\rangle + \langle \mathbf{w}_9,\mathbf{x}\rangle$

Figure 3.1: Taxonomy with 5 categories and a total of 10 nodes. The decomposition of the discriminant function for category 2 is depicted as an example.

An illustrating example is depicted in Figure 3.1.

If we define

$$\mathbf{W}_y = \sum_{z: z \in \mathrm{anc}(y)} \lambda_z(y)\mathbf{w}_z \,, \tag{3.9}$$

we can rewrite Equation (3.8) as

$$F(\mathbf{x}, y; \mathbf{w}) = \left\langle \sum_{z:\, z \in \mathrm{anc}(y)} \lambda_z(y)\mathbf{w}_z \;,\; \mathbf{x} \right\rangle = \langle \mathbf{W}_y, \mathbf{x}\rangle \,. \tag{3.10}$$

Note that $\mathbf{W}_y$ consists of a linear combination of weight vectors of ancestor nodes and functions as the weight vector or prototype associated with class y. We therefore name $\mathbf{w}$ the weight vector or the node weight vector and $\mathbf{W}$ the class weight vector. Owing to how we define taxonomy-based class attributes, the weight vectors of nearby classes in the taxonomy tend to be more similar since they share many common ancestors. Meanwhile the weight vectors of distant classes tend to be less similar. This is intuitive as we would expect the prototype of class `volleyball game` to be more similar to that of `soccer game` than to that of `country music`.

3.3 Loss functions

A standard loss function for the multi-label case is to use the *symmetric difference* between the predicted and the actual label set, i.e. to count the number of correct categories missed plus the number of incorrect categories that have been assigned, $\triangle_0(Y,\hat{Y}) \equiv |Y \ominus \hat{Y}|$.

Yet, in many applications, the actual loss of a predicted label set relative to the true set of category labels will depend on the relationship between the categories. As a motivation we consider the generic setting of routing items based on their membership at nodes in the taxonomy. For instance, in a news routing setting, readers may sign-up for specific topics by selecting an appropriate node, which can either be a terminal node in the taxonomy (e.g. the category "Soccer") or an inner node (e.g. the super-category "Sports"). Note that while we assume that all items can only be assigned to terminal nodes of the taxonomy, customers may prefer to sign-up for many categories *en bloc* by selecting an appropriate super-category.

We assume that there is some sign-up volume $s_v \geq 0$ (e.g. the number of users that subscribe to node v) for each node as well as costs c^- of missing a relevant item and c^+ for assigning an irrelevant item. For any label set Y, define $\text{anc}(Y) \equiv \{v \in \mathcal{V} : \exists y \in Y, v \in \text{anc}(y)\}$. Now we can quantify the loss in the following manner:

$$\triangle_{\text{H}}(Y,\hat{Y}) = c^{-}\sum_{v\in\text{anc}(Y)} s_v + c^{+}\sum_{v\in\text{anc}(\hat{Y})} s_v - (c^- + c^+)\sum_{\substack{v\in\text{anc}(Y)\\ \cap\text{anc}(\hat{Y})}} s_v \tag{3.11}$$

Note that only nodes in the symmetric difference $\text{anc}(Y) \ominus \text{anc}(\hat{Y})$ contribute to the loss. In the following we will simplify the presentation by assuming that $c^- = c^+ = 1$. Then, by further setting $s_v = 1$ $(\forall v \in \mathcal{V})$ one gets

$$\triangle_{\text{H}}(Y,\hat{Y}) = \left|\text{anc}(Y) \ominus \text{anc}(\hat{Y})\right|. \tag{3.12}$$

Intuitively, this means that one colors all nodes that are on a path to a node in Y with one color, say blue, and all nodes on paths to nodes in $\hat{Y}$ with another color, say yellow (see Figure 3.2). Nodes that have both colors (blue+yellow=green) are correct, blue nodes are the ones that have been missed and yellow nodes are the ones that have been incorrectly selected; both types of mistakes contribute to the loss proportional to their volume.

During training, this loss function is difficult to deal with directly, since it involves sets of labels. Rather, we would like to work with pairwise contributions, e.g. involving terms

$$\triangle_{\text{H}}(y,y') = \left|\text{anc}(y) \ominus \text{anc}(y')\right|. \tag{3.13}$$

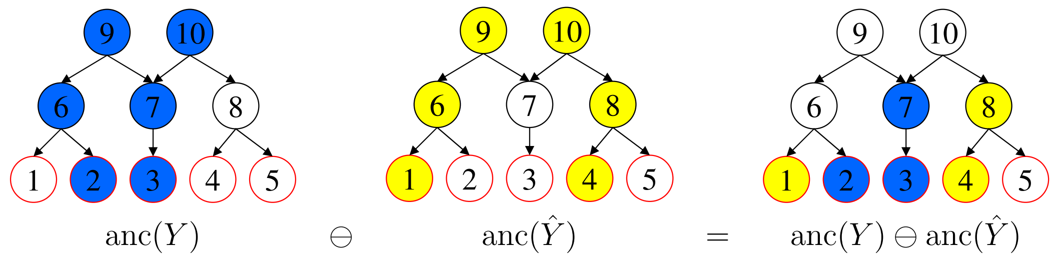


Figure 3.2: Hierarchical loss between two sets of categories. Let $Y = \{2,3\}$, $\hat{Y} = \{1,4\}$, and therefore $\text{anc}(Y) \ominus \text{anc}(\hat{Y}) = \{1,2,3,4,7,8\}$.

In singly connected taxonomies Equation (3.13) is equivalent to the length of the (undirected) shortest path connecting the nodes y and y', suggested by [65]. It is reasonable to assume that confusing classes that are "nearby" in the taxonomy is less costly or severe than predicting a class that is "far away" from the correct class. In order to relate the loss between label set pairs and the loss between label pairs, we state the following proposition:

Proposition 1. *For any $Y, \hat{Y} \subseteq \mathcal{Y}$ satisfying $Y \not\subseteq \hat{Y}$ and $\hat{Y} \not\subseteq Y$,*

$$|anc(Y) \ominus anc(\hat{Y})| \leq \sum_{\substack{y \in Y - \hat{Y} \\ \hat{y} \in \hat{Y} - Y}} |anc(y) \ominus anc(\hat{y})|$$

Proof. We first show

$$\text{anc}(Y) \ominus \text{anc}(\hat{Y}) \subseteq \bigcup_{\substack{y \in Y - \hat{Y} \\ \hat{y} \in \hat{Y} - Y}} \text{anc}(y) \ominus \text{anc}(\hat{y}) \tag{3.14}$$

Pick an arbitrary element, z, from set $\text{anc}(Y) - \text{anc}(\hat{Y})$. By definition, there exists an element $y \in Y$ such that $z \in \text{anc}(y)$ and for any element $\hat{r}$ in $\hat{Y}$, $z \notin \text{anc}(\hat{r})$. We thus have $y \in Y - \hat{Y}$. Since $\hat{Y} \not\subseteq Y$, there must exists at least one element $\hat{y}$ that is in the set $\hat{Y} - Y$. Therefore, we know

$$\exists y \in Y - \hat{Y}, \hat{y} \in \hat{Y} - Y, \text{ such that } z \in \text{anc}(y) - \text{anc}(\hat{y})\,.$$

As a consequence, $z \in \bigcup_{y \in Y - \hat{Y}, \hat{y} \in \hat{Y} - Y} \text{anc}(y) \ominus \text{anc}(\hat{y})$. We thus proved

$$\text{anc}(Y) - \text{anc}(\hat{Y}) \subseteq \bigcup_{\substack{y \in Y - \hat{Y} \\ \hat{y} \in \hat{Y} - Y}} \text{anc}(y) \ominus \text{anc}(\hat{y})$$

Similarly, we can prove $\mathrm{anc}(\hat{Y}) - \mathrm{anc}(Y) \subseteq \bigcup_{y \in Y - \hat{Y}, \hat{y} \in \hat{Y} - Y} \mathrm{anc}(y) \ominus \mathrm{anc}(\hat{y})$. Therefore Equation (3.14) holds. We thus have

$$\left| \mathrm{anc}(Y) \ominus \mathrm{anc}(\hat{Y}) \right| \leq \left| \bigcup_{\substack{y \in Y - \hat{Y} \\ \hat{y} \in \hat{Y} - Y}} \mathrm{anc}(y) \ominus \mathrm{anc}(\hat{y}) \right| \leq \sum_{\substack{y \in Y - \hat{Y} \\ \hat{y} \in \hat{Y} - Y}} \left| \mathrm{anc}(y) \ominus \mathrm{anc}(\hat{y}) \right|$$

□

For learning with ranking functions g, we translate this into

$$\triangle_{\mathrm{H}}(Y, g) = \sum_{\substack{y \in Y, \hat{y} \in \mathcal{Y} - Y \\ g(y) > g(\hat{y})}} \left| \mathrm{anc}(y) \ominus \mathrm{anc}(\hat{y}) \right| . \tag{3.15}$$

We look at every pair of categories where an incorrect category comes before a correct category in the order defined by g and count the symmetric difference of the respective ancestor sets as the corresponding loss.

Chapter 4

Hierarchical Support Vector Machines

In this chapter, we will first introduce the flat multilabel SVM, then propose our hierarchical SVM formulation and the optimization algorithm, rewrite our hierarchical multilabel SVM in another set of variables to analyze its taxonomy-dependent way of regularization, and examine related work.

4.1 Flat multilabel SVM

We generalize the multiclass SVM formulation in Section 2.2.3 to a multilabel formulation similar to that in [24]. For a given set of correct categories Y_i we denote the complement by $\bar{Y}_i = \mathcal{Y} - Y_i$. Following [24] we then approximate the separation margin of $\mathbf{w}$ with respect to the i-th example as

$$\gamma_i(\mathbf{w}) \equiv \min_{y \in Y_i, \bar{y} \in \bar{Y}_i} \langle \mathbf{w}_y, \mathbf{x}_i \rangle - \langle \mathbf{w}_{\bar{y}}, \mathbf{x}_i \rangle \,. \tag{4.1}$$

Note that the correct ranking of classes requires a positive margin. A correct ranking is one in which any relevant class is ranked higher than any irrelevant class. Figure 4.1 (a) and (b) show two scoring schemes that are both correct rankings. Assuming for now that the training data can indeed be correctly classified by some weight vector $\mathbf{w}$, we can apply the maximum margin principle to determine the weight vector $\mathbf{w}^*$

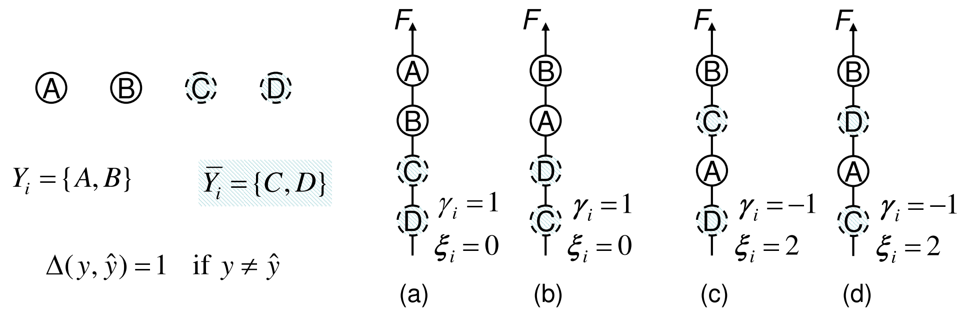


Figure 4.1: Illustration of margin and penalty for the flat multilabel SVM. The vertical axis represents the compatibility score F associated with the classes. For convenience, let the score difference of neighboring classes along F axis be 1. The class on top of the vertical line is the one that has the highest compatibility score. The relevant classes are class A and B while the irrelevant ones are class C and D. The leftmost column depicts the classes and corresponding loss. Figure (a) and (b) show two correct rankings while (c) and (d) show two incorrect rankings.

by achieving optimal separation

$$\mathbf{w}^* = \underset{\mathbf{w}:\|\mathbf{w}\|=1}{\operatorname{argmax}} \ \min_i \gamma_i(\mathbf{w})\,. \tag{4.2}$$

This is equivalent to minimizing the norm of the weight vector $\mathbf{w}$ while constraining all (functional) margins to be greater than or equal to 1. A multilabel soft-margin SVM formulation can be obtained by introducing slack variables ξ_i's. Thus the soft-margin multilabel flat SVM is defined as

$$\min_{\{\mathbf{w}_y\}_{y=1}^q, \boldsymbol{\xi}} \ \frac{1}{2}\sum_{y=1}^{q} \|\mathbf{w}_y\|^2 + C\sum_{i=1}^{n} \xi_i \tag{4.3a}$$

$$\text{s.t.} \quad \langle \mathbf{w}_y, \mathbf{x}_i \rangle - \langle \mathbf{w}_{\bar{y}}, \mathbf{x}_i \rangle \geq 1 - \xi_i\,, \quad (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i) \tag{4.3b}$$

$$\xi_i \geq 0, \quad (\forall i)\,. \tag{4.3c}$$

Figure (4.1) (c) and (d) show two scoring schemes in which slack variables become non-zero.

4.2 Hierarchical multilabel SVM

4.2.1 Primal program

A shortcoming of the flat multilabel SVM in Equation (4.3) is that the categories are treated as flat. We have proposed in Chapter 3 two methods of encoding a taxonomy.

One is to derive class attributes from a taxonomy and utilize them to tie learning across classes. The other is hierarchical loss functions that depend on the relation of classes. The problem that needs to be addressed next is how to modify the SVM formulation to take advantage of the taxonomy encodings.

Although we have derived specific hierarchical loss function $\triangle_{\mathrm{H}}$ in Section 3.3, we would like to work with general loss functions $\triangle : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$, where $\triangle(y, \hat{y})$ denotes the loss of predicting $\hat{y}$ when the true class is y. We only assume that $\triangle(y, y) = 0$ and that $\triangle(y, \hat{y}) > 0$ for $y \neq \hat{y}$.

Now with the introduction of class attributes, as in Equation (3.2), the discriminant function becomes $F(\mathbf{x}, y; \mathbf{w}) \equiv \langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x}, y) \rangle$. We can then define the margin to be

$$\gamma_i(\mathbf{w}) \equiv \min_{y \in Y_i, \bar{y} \in \bar{Y}_i} \langle \mathbf{\Phi}(\mathbf{x}_i, y) - \mathbf{\Phi}(\mathbf{x}_i, \bar{y}), \mathbf{w} \rangle \,. \tag{4.4}$$

Assuming the training data can indeed be correctly classified by some weight vector, the hard margin problem in Equation (4.2) now expands to

$$\max_{\mathbf{w}, \gamma} \quad \gamma \tag{4.5a}$$

$$\text{s.t.} \quad \|\mathbf{w}\| = 1 \,, \tag{4.5b}$$

$$\langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x}, y) - \mathbf{\Phi}(\mathbf{x}, \bar{y}) \rangle \geq \gamma \quad (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i) \,. \tag{4.5c}$$

The constraint in Equation (4.5c) can be rewritten as

$$\langle \frac{\mathbf{w}}{\gamma}, \mathbf{\Phi}(\mathbf{x}_i, y) - \mathbf{\Phi}(\mathbf{x}_i, \bar{y}) \rangle \geq 1 \,, (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i) \,. \tag{4.6}$$

Since $\|\mathbf{w}\| = 1$ in Equation (4.5b), the objective function in Equation (4.5a) can be transformed to

$$\gamma = \frac{\gamma}{\|\mathbf{w}\|} \|\mathbf{w}\| = \left\| \frac{\gamma}{\mathbf{w}} \right\| = \frac{1}{\| \frac{\mathbf{w}}{\gamma} \|} \,. \tag{4.7}$$

Now it becomes clear that Equation (4.5) is identical to

$$\min_{\mathbf{w}, \gamma} \quad \left\| \frac{\mathbf{w}}{\gamma} \right\| \tag{4.8a}$$

$$\text{s.t.} \quad \|\mathbf{w}\| = 1 \tag{4.8b}$$

$$\langle \frac{\mathbf{w}}{\gamma}, \mathbf{\Phi}(\mathbf{x}, y) - \mathbf{\Phi}(\mathbf{x}, \bar{y}) \rangle \geq 1 \,, \quad (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i) \,. \tag{4.8c}$$

Note that $\frac{\mathbf{w}}{\gamma}$ is also a weight vector and it defines the same classifier as $\mathbf{w}$. By using $\frac{\mathbf{w}}{\gamma}$ as the optimization variable, the constraint $\|\mathbf{w}\| = 1$ can be removed. Therefore we arrive at the following norm-minimization form of the hard margin SVM that incorporates joint feature representation:

$$\min_{\mathbf{w}} \quad \frac{1}{2}\|\mathbf{w}\|^2 \tag{4.9a}$$

$$\text{s.t.} \quad \langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x}_i, y) - \mathbf{\Phi}(\mathbf{x}_i, \bar{y}) \rangle \geq 1\,, \quad (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i)\,. \tag{4.9b}$$

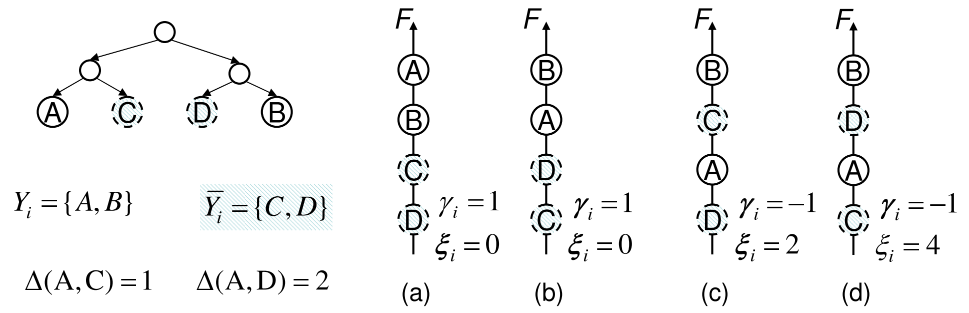
In our hierarchical SVM algorithm, we suggest the penalty ξ_i be scaled proportionally to the loss associated with the violation of the desired margin. Putting these ideas together yields our hierarchical multilabel SVM:

$$\begin{aligned} \min_{\mathbf{w},\boldsymbol{\xi}} \ & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t. } & \langle \mathbf{w}, \delta\mathbf{\Phi}_i(y,\bar{y}) \rangle \geq 1 - \frac{\xi_i}{\triangle(y,\bar{y})}\,, \quad (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i) \\ & \quad \xi_i \geq 0, \quad (\forall i)\,, \end{aligned} \tag{4.10}$$

where $\delta\mathbf{\Phi}_i(y,\bar{y}) \equiv \mathbf{\Phi}(\mathbf{x}_i, y) - \mathbf{\Phi}(\mathbf{x}_i, \bar{y})$..

This formulation generalizes the flat multilabel SVM. The same as in flat multilabel SVM, the efforts are put into correctly ordering each pair of positive and negative labels. Different from the flat SVM, the margin violations involving an incorrect class will be penalized in accordance with the associated loss. The higher the loss, the more severe the penalty will be. Figure 4.2 shows this distinction. We can also use a set size prediction mechanism such as the one in [24] to convert the category ranking into an actual multi-label assignment. The size prediction function is learned from training data to predict, for any instance, the number of correct labels or the threshold value to select positive classes.

Comparing our hierarchical SVM formulation in Equation (4.10) with the binary SVM in Equation (2.6), one finds that the two formulations appear similar although they address different learning problems. Indeed, we can view $\delta\mathbf{\Phi}_i(y,\bar{y})$ $(\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i)$ as a positive instance casted in the joint feature space. A binary discriminant in the joint space corresponds to a multilabel category-ranking classifier in the input space. The multilabel learning problem in the input space turns out to be almost a



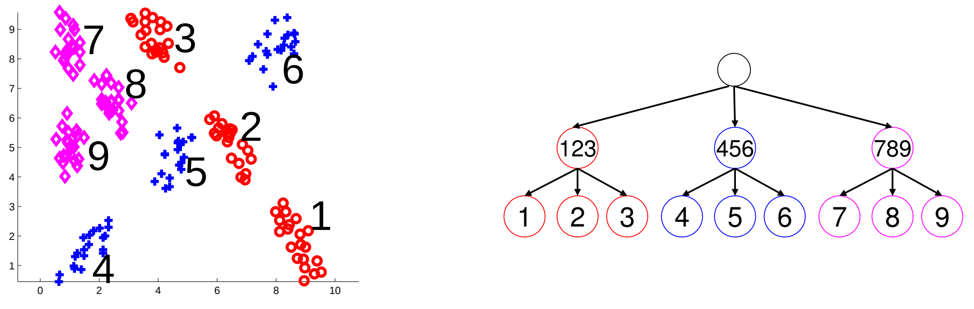
$Y_i = \{A, B\}$
$\bar{Y}_i = \{C, D\}$
$\Delta(A,C) = 1$
$\Delta(A,D) = 2$
F
$\gamma_i = 1$ $\xi_i = 0$
(a)
$\gamma_i = 1$ $\xi_i = 0$
(b)
$\gamma_i = -1$ $\xi_i = 2$
(c)
$\gamma_i = -1$ $\xi_i = 4$
(d)

Figure 4.2: Illustration of margin and penalty for the hierarchical multilabel SVM. The notations, classes, and ranking schemes are the same as those in Figure 4.1. The pairwise hierarchical loss used is half the distance between classes in the taxonomy. In particular, note how the ξ_i value in (d) differs from its counterpart in Figure 4.1 due to the involvement of the hierarchical loss.

binary classification problem in the joint input-output space, except $\delta\mathbf{\Phi}_i(y, \bar{y})$'s for the same instance $\mathbf{x}_i$ are tied by sharing a common slack variable ξ_i.

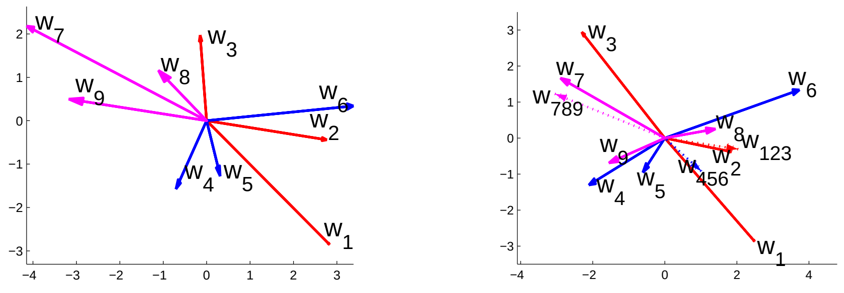
In the special case of multiclass learning with $q = 2$, the learning problem reduces to binary classification. If using 0/1 loss and indicator class attributes $\lambda_r(y) = \delta_{ry}$, then the constraints in Equation (4.10) only involve the difference vector $\mathbf{v} \equiv \mathbf{w}_1 - \mathbf{w}_2$. Moreover it is easy to see that the penalty on $\|\mathbf{w}\|$ implies that the optimal weight vector fulfills $\mathbf{w}_1 = \frac{1}{2}\mathbf{v}$ and $\mathbf{w}_2 = -\frac{1}{2}\mathbf{v}$. Hence, one can equivalently minimize the norm $\|\mathbf{v}\|$, showing that the $q = 2$ case can indeed reduce to the binary SVM classification.

Here we discuss a concrete example to enhance understanding of the hierarchical SVM learning. Figure 4.3 depicts an artificial data set together with its category taxonomy. The two-dimensional data are assigned to nine categories which are organized into a two-level hierarchy. The learned weight vectors for both the flat SVM and the hierarchical SVM, in the same coordinate system as the data, are shown in 4.4. In the hierarchical learning, we note the weight vector of a parent node gives a direction that the child nodes share and then the weight vectors of these child nodes articulate the difference among them. In the flat SVM whose solution is depicted in Figure 4.4(a), the weight vector associated with a class is used directly and solely in the scoring function of the class. In contrary, in the hierarchical SVM whose solution is depicted in Figure 4.4(b), each weight vector is associated with a node in the taxonomy; the weight vector of a node that corresponds to a class has to be combined with the weight vectors of all its ancestor nodes, in a form of weighted sum, to contribute to



(a) data (b) taxonomy

Figure 4.3: A 2-dimensional dataset with 9 classes in a 2-level tree taxonomy. Node 1, 2, and 3 share a common parent node numbered 123. The same numbering convention applies to the other two branches under the root.



(a) flat SVM solution (b) hierarchical SVM solution

Figure 4.4: Flat and hierarchical SVM solutions on training data in Figure 4.3. The flat SVM solution is in Figure (a) and the hierarchical in Figure (b).

the scoring function of the class. We did not provide the learned weight vector for the root node of the taxonomy because it would always be 0 as long as its attribute value is identical across all classes. In fact this is true for for any node which is an ancestor for all terminal nodes in a taxonomy. This phenomenon is determined by the optimization problem of the hierarchical SVM itself.

4.2.2 Dual quadratic program

In this section we derive the dual quadratic program (QP) of the above cost-sensitive formulation of large margin learning with class attributes. To that extent one first forms the Lagrangian function

$$\mathcal{L}(\mathbf{w},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\zeta}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\zeta_i\xi_i - \sum_{i=1}^{n}\sum_{y\in Y_i}\sum_{\bar{y}\in\bar{Y}_i}\alpha_{iy\bar{y}}\left(\langle\delta\boldsymbol{\Phi}_i(y,\bar{y}),\mathbf{w}\rangle - 1 + \frac{\xi_i}{\triangle(y,\bar{y})}\right) \tag{4.11}$$

Computing derivatives of $\mathcal{L}$ with respect to the primal variables results in

$$\nabla_{\mathbf{w}}\mathcal{L} = 0 \iff \mathbf{w}(\boldsymbol{\alpha}) \equiv \sum_{i=1}^{n}\sum_{y\in Y_i,\bar{y}\in\bar{Y}_i}\alpha_{iy\bar{y}}\delta\boldsymbol{\Phi}_i(y,\bar{y})\,, \tag{4.12}$$

$$\nabla_{\boldsymbol{\xi}}\mathcal{L} = 0 \iff \zeta_i = C - \sum_{y\in Y_i,\bar{y}\in\bar{Y}_i}\frac{\alpha_{iy\bar{y}}}{\triangle(y,\bar{y})}\,. \tag{4.13}$$

Since $\zeta_i \geq 0$, Equation (4.13) reduces to a loss function weighted box constraint. Plugging in the optimality equation for $\mathbf{w}$ and exploiting Equation (4.13), one arrives at the dual objective

$$\Theta(\boldsymbol{\alpha}) = \sum_{i=1}^{n}\sum_{\substack{y\in Y_i\\ \bar{y}\in\bar{Y}_i}}\alpha_{iy\bar{y}} - \frac{1}{2}\sum_{i,j}\sum_{\substack{y\in Y_i\\ \bar{y}\in\bar{Y}_i}}\sum_{\substack{r\in Y_j\\ \bar{r}\in\bar{Y}_j}}\alpha_{iy\bar{y}}\alpha_{jr\bar{r}}\langle\delta\boldsymbol{\Phi}_i(y,\bar{y}),\delta\boldsymbol{\Phi}_j(r,\bar{r})\rangle\,. \tag{4.14}$$

The solution to the dual QP is thus characterized by

$$\boldsymbol{\alpha}^* \equiv \underset{\boldsymbol{\alpha}}{\operatorname{argmax}}\ \Theta(\boldsymbol{\alpha}) \qquad \text{s.t. } \alpha_{iy\bar{y}} \geq 0\ (\forall i, y\in Y_i, \bar{y}\in\bar{Y}_i)\,,\ \sum_{\substack{y\in Y_i\\ \bar{y}\in\bar{Y}_i}}\frac{\alpha_{iy\bar{y}}}{\triangle(y,\bar{y})} \leq C\ (\forall i)\,. \tag{4.15}$$

Note that in Equation (4.14)

$$\langle\delta\boldsymbol{\Phi}_i(y,\bar{y}),\delta\boldsymbol{\Phi}_j(r,\bar{r})\rangle = \langle\boldsymbol{\Lambda}(y)-\boldsymbol{\Lambda}(\bar{y})\,,\,\boldsymbol{\Lambda}(r)-\boldsymbol{\Lambda}(\bar{r})\rangle\langle\mathbf{x}_i,\mathbf{x}_j\rangle\,. \tag{4.16}$$

Herein one can simply replace the inner products $\langle\mathbf{x}_i,\mathbf{x}_j\rangle$ by corresponding kernel functions like in the standard SVM classification. Thus data can be mapped to a space of very high dimension yet classification can be solved efficiently, with little extra efforts. Furthermore note the right hand side of Equation (4.16) can be further expanded into

$$\begin{aligned}&\langle\delta\boldsymbol{\Phi}_i(y,\bar{y}),\delta\boldsymbol{\Phi}_j(r,\bar{r})\rangle\\ &= (\,\langle\boldsymbol{\Lambda}(y)\,,\,\boldsymbol{\Lambda}(r)\rangle + \langle\boldsymbol{\Lambda}(\bar{y})\,,\,\boldsymbol{\Lambda}(\bar{r})\rangle - \langle\boldsymbol{\Lambda}(y)\,,\,\boldsymbol{\Lambda}(\bar{r})\rangle - \langle\boldsymbol{\Lambda}(\bar{y})\,,\,\boldsymbol{\Lambda}(r)\rangle\,)\,\langle\mathbf{x}_i,\mathbf{x}_j\rangle\,.\end{aligned} \tag{4.17}$$

The dual program depends on class attribute vectors only in terms of their inner products. Therefore one can also replace $\langle \mathbf{\Lambda}(y), \mathbf{\Lambda}(\hat{y}) \rangle$ with a a kernel function so as to leverage the power of kernel or to cope with cases where the class attributes are not given but the similarity between classes is available.

It is straightforward to observe that $\frac{1}{n}\sum_i \xi_i$ yields an upper bound on the training loss of the resulting classifier measured by $\triangle$ in the following sense. We define the *maximum loss* as

$$\triangle^x(Y, g) \equiv \max_{y \in Y, \bar{y} \in \bar{Y} : g(y) \geq g(\bar{y})} \triangle(y, \bar{y})\,. \tag{4.18}$$

If there is no $(y, \bar{y})$ pair that is mistakenly ranked, the right hand side of Equation (4.18) takes the value 0. So maximal loss is the biggest pair-wise loss among mis-ranked positive label and negative label pairs. The *maximal loss* over a set of examples is defined as

$$\triangle^x(F, \{\mathbf{x}_i, Y_i\}_{i=1}^n) \equiv \frac{1}{n}\sum_{i=1}^n \triangle^x(Y_i, g(\mathbf{x}_i; F))\,. \tag{4.19}$$

Proposition 2. *Denote by* $(\hat{\mathbf{w}}, \hat{\boldsymbol{\xi}})$ *a feasible solution of the QP in* (4.10). *Then* $\frac{1}{n}\sum_{i=1}^n \hat{\xi}_i$ *is an upper bound on the empirical maximal loss* $\triangle^x(F(;\hat{\mathbf{w}}), \{\mathbf{x}_i, Y_i\}_{i=1}^n)$.

Proof. Since $\hat{\mathbf{w}}$ and $\hat{\boldsymbol{\xi}}$ satisfy constraints in Equation (4.10),

$$\hat{\xi}_i \geq \max\{0\ ,\ \max_{\substack{y \in Y_i \\ \bar{y} \in \bar{Y}_i}} \{\triangle(y, \bar{y})(1 - \langle \delta\mathbf{\Phi}_i(y, \bar{y}), \hat{\mathbf{w}} \rangle)\}\}\,. \tag{4.20}$$

Clearly, if there are no negative categories that have compatibility scores higher than or equal to those of any positive categories, then the maximal loss is 0 and $\hat{\xi}_i \geq 0$.

If there is any pair of positive label and negative label $(y, \bar{y})$ that satisfies $F(\mathbf{x}_i, y; \hat{\mathbf{w}}) \leq F(\mathbf{x}_i, \bar{y}; \hat{\mathbf{w}})$, then by Equation (4.20) $\xi_i \geq \triangle(y, \bar{y})$. This holds for any mis-ranked label pairs.

By definition of $\triangle^x$, we have $\xi_i \geq \triangle^x(F, \mathbf{x}_i, Y_i)$. Applying this inequality to every instance proves the proposition. □

Note that to minimize (an upper bound on) the loss in Equation (3.15), we could simply assign one slack variable $\xi_{iy\bar{y}}$ for every triplet of instance, positive label, and negative label. This leads to a dual program similar to Equation (4.15) except the second set of constraints become $\frac{\alpha_{iy\bar{y}}}{\triangle(y,\bar{y})} \leq C \;\; (\forall i, y \in Y, \bar{y} \in \bar{Y})$. We have not explored this variant.

In many cases, especially in the multiclass case, users are particularly interested in the quality of the category that is ranked the highest by the learned classifier. We therefore also define the *top loss* $\triangle^t$ as

$$f(\mathbf{x}) \equiv \operatorname*{argmax}_{y \in \mathcal{Y}} F(\mathbf{x}, y)\,, \tag{4.21}$$

$$\triangle^t(F, \mathbf{x}_i, Y_i) \equiv \begin{cases} 0 & \text{if } f(\mathbf{x}_i) \in Y_i\,, \\ \max_{y \in Y_i} \triangle(y, f(\mathbf{x}_i)) & \text{otherwise,} \end{cases} \tag{4.22}$$

$$\triangle^t(F, \{\mathbf{x}_i, Y_i\}_{i=1}^n) \equiv \frac{1}{n} \sum_{i=1}^n \triangle^t(F, \mathbf{x}_i, Y_i)\,. \tag{4.23}$$

The function $f(\mathbf{x})$ always predicts the category that achieves the top compatibility score. If $f(\mathbf{x}_i)$ is in the set of correct categories Y_i, the top loss will be 0. Otherwise, $\triangle^t$ takes the maximum loss between the predicted label and any positive label. In the special case of multiclass learning, the top loss becomes the loss between the correct class and the predicted class. If $\triangle$ is 0/1 loss, the top loss then becomes the *one error* [54], which is the percentage of predicted classes being wrong. It is clear from the definitions of the two losses that the maximal loss is always an upper bound of the top loss. Therefore the following proposition holds as well.

Corollary 1. *Denote by* $(\hat{\mathbf{w}}, \hat{\boldsymbol{\xi}})$ *a feasible solution of the QP in Equation* (4.10). *Then* $\frac{1}{n}\sum_{i=1}^n \hat{\xi}_i$ *is an upper bound on the empirical top loss* $\triangle^t(F(;\hat{\mathbf{w}}, \{\mathbf{x}_i, Y_i\}_{i=1}^n)$.

4.3 Optimization algorithm

4.3.1 General strategy

The dual QP in Equation (4.15) can become quite large in practice, since the number of α-variables equals $\sum_{i=1}^n |Y_i||\bar{Y}_i|$. In particular the scaling with q, the number of categories, is problematic when compared, for instance, to standard (binary) SVMs. Inspired by the SMO algorithm [48], we propose to exploit two properties of the dual problem in order to design a more efficient optimization algorithm.

First, note that the upper bound constraints in the dual problem Equation (4.15) factorize over the instance index. By this we mean that the second set of constraints in Equation (4.15) do not couple dual variables $\alpha_{iy\bar{y}}$ and $\alpha_{jr\bar{r}}$ belonging to different training instances i and j. This can be exploited in an optimization procedure which iteratively performs subspace optimization over all dual variables $\alpha_{iy\bar{y}}$ belonging to the same training instance. This will in general be a much smaller QP, since it freezes all $\alpha_{jr\bar{r}}$ with $j \neq i$ at their current values. This idea has also been successfully applied in the multiclass SVM optimization algorithm proposed in [14]. However, since class attribute vectors $\mathbf{\Lambda}(y)$ are in general not orthogonal, we can not use the fix point method proposed in [14].

Secondly, we expect the number of active constraints at the solution to be relatively small, since only a small fraction of category pairs will typically fail to achieve the required margin. As with SVMs, this is not a necessity, but for classification problems that can be solved with reasonable accuracy, this sparseness property can be observed empirically (cf. Chapter 7). We propose to exploit the expected sparseness by employing a variable selection strategy for the dual problem. Equivalently, this corresponds to a cutting plane algorithm for the primal QP. Intuitively, we will identify the most violated margin constraint with index $(i, y, \bar{y})$ and then add the variable $\alpha_{iy\bar{y}}$ to the optimization problem. This means that we start with extremely sparse (i.e. small) problems and only successively increase the number of variables in the active set. This general approach to deal with large linear or quadratic optimization problems is also known as column generation [20]. In practice, it is often not necessary to optimize until final convergence, which adds to the attractiveness of this approach.

4.3.2 Variable selection strategy

Since we use an iterative approach for optimization, which selects one dual variable at a time for inclusion in the sparsified optimization problem, it is important to develop a sensible strategy for selecting those variables. In particular, we would like to utilize a heuristic which focuses on those constraints that are most severely violated. In order to implement this idea, we need to quantify the extent to which constraints are violated. For that purpose, we generalize the approach of [14] for which we will present a simplified derivation in the sequel.

Let

$$H_{iy\bar{y}} \equiv \triangle(y,\bar{y})\left(1 - \langle \delta\Phi_i(y,\bar{y}), \mathbf{w}(\boldsymbol{\alpha})\rangle\right)\ . \tag{4.24}$$

Remember $\mathbf{w}(\boldsymbol{\alpha}) \equiv \sum_{i=1}^{n} \sum_{y\in Y_i,\bar{y}\in\bar{Y}_i} \alpha_{iy\bar{y}} \delta\boldsymbol{\Phi}_i(y,\bar{y})$ from Equation (4.12). Positive values of $H_{iy\bar{y}}$ correspond to violations of the requested (functional) margin of 1. Given a feasible solution $\boldsymbol{\alpha}$ of the dual QP problem in Equation (4.15), the necessary and sufficient conditions for $\boldsymbol{\alpha}$ to be an optimal solution (by Karush-Kuhn-Tucker theorem, e.g. in [17]) are

$$\xi_i \geq H_{iy\bar{y}}, \quad (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i) \tag{4.25a}$$

$$\xi_i \geq 0\,, \quad (\forall i) \tag{4.25b}$$

$$\alpha_{iy\bar{y}}\left\{H_{iy\bar{y}} - \xi_i\right\} = 0\,, \quad (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i) \tag{4.25c}$$

$$\zeta_i \xi_i = 0\,, \quad (\forall i)\,, \tag{4.25d}$$

where $\zeta_i = C - \sum_{y\in Y_i,\bar{y}\in\bar{Y}_i} \frac{\alpha_{iy\bar{y}}}{\triangle(y,\bar{y})}$ is given in Equation (4.13). Equation (4.25c) and (4.25d) are derived from KKT complimentary conditions.

Via Equation (4.25a) and (4.25b), we get

$$\xi_i \geq \max\left(\max_{y\in Y_i,\bar{y}\in\bar{Y}_i} H_{iy\bar{y}} \quad , \quad 0\right) \tag{4.26}$$

From Equation (4.25c) and (4.25d), we have that

$$\xi_i = \min_{\substack{y\in Y_i,\bar{y}\in\bar{Y}_i: \\ \alpha_{iy\bar{y}}>0}} H_{iy\bar{y}} \tag{4.27}$$

when $\zeta_i = 0$ and

$$\xi_i = \min\left(\min_{\substack{y\in Y_i,\bar{y}\in\bar{Y}_i: \\ \alpha_{iy\bar{y}}>0}} H_{iy\bar{y}} \ , \ 0\right) \tag{4.28}$$

when $\zeta_i > 0$. We let the quantity $\min_{y\in Y_i,\bar{y}\in\bar{Y}_i:\alpha_{iy\bar{y}}>0} H_{iy\bar{y}}$ be 0 if all dual variables related to $\mathbf{x}_i$ are zeros.

We now define the following quantities for every instance:

$$l_i \equiv \max\left(\max_{y\in Y_i,\bar{y}\in\bar{Y}_i} H_{iy\bar{y}} \quad , \quad 0\right) \tag{4.29}$$

$$u_i \equiv \begin{cases} \min_{\substack{y\in Y_i,\bar{y}\in\bar{Y}_i: \\ \alpha_{iy\bar{y}}>0}} H_{iy\bar{y}} & \text{if } \zeta_i = 0 \\ \min\left(\min_{\substack{y\in Y_i,\bar{y}\in\bar{Y}_i: \\ \alpha_{iy\bar{y}}>0}} H_{iy\bar{y}} \ , \ 0\right) & \text{if } \zeta_i > 0\,. \end{cases} \tag{4.30}$$

Therefore from Equation (4.25) we have

$$\xi_i = u_i \geq l_i, \quad (\forall i) \tag{4.31}$$

The following score, ψ_i, is hence used for selecting subspaces.

$$\psi_i \equiv l_i - u_i\,. \tag{4.32}$$

It can be shown that Equation (4.25) holds if and only if $\psi_i = 0\,(\forall i)$. Therefore, we have the proposition as below.

Proposition 3. *Given a feasible solution $\boldsymbol{\alpha}$ of Equation* (4.15)*, $\boldsymbol{\alpha}$ is an optimum if and only if $\psi_i = 0$ for all $i = 1,\ldots,n$.*

Proof. First we show that if $\boldsymbol{\alpha}$ is an optimum then $\psi_i = 0\,(\forall i)$. Since $\boldsymbol{\alpha}$ is an optimum, Equation (4.25) is true. From the deduction above, we arrive at Equation (4.31): $\xi_i = u_i \geq l_i\,(\forall i)$.

On the other hand, by definitions of u_i and l_i we have either
case I: if $\alpha_{iy\bar{y}} = 0\,(\forall y\in Y_i, \bar{y}\in\bar{Y}_i)$, then $l_i \geq 0$ and $u_i \leq 0$; or
case II: if there exists some $\hat{y}\in Y_i$ and $\hat{\bar{y}}\in\bar{Y}_i$ such that $\alpha_{i\hat{y}\hat{\bar{y}}} > 0$, then $l_i \geq H_{i\hat{y}\hat{\bar{y}}}$ and $u_i \leq H_{i\hat{y}\hat{\bar{y}}}$.
In either cases, $l_i \geq u_i$ $(\forall i)$.

Since both $u_i \geq l_i$ and $l_i \geq u_i$ have to be true, we conclude $l_i = u_i$ and therefore $\psi_i = 0\,(\forall i)$.

Next we show that if $\psi_i = 0$ $(\forall i)$ in a given feasible solution $\boldsymbol{\alpha}$, then $\boldsymbol{\alpha}$ is an optimal solution. To this purpose, we only need to show Equation (4.25) holds. Given that $\psi_i = 0$ for all instances, we let $\xi_i \equiv l_i = u_i$. Then we have

$$\xi_i = l_i \geq H_{iy\bar{y}} \qquad (\forall i, y\in Y_i, \bar{y}\in\bar{Y}_i)\,, \tag{4.33}$$

$$\xi_i = l_i \geq 0\,, \tag{4.34}$$

by definitions of l_i in Equation (4.29). For any $\alpha_{iy\bar{y}} > 0$, we know $H_{iy\bar{y}} \leq l_i = u_i \leq H_{iy\bar{y}}$ by definitions of l_i and u_i, and therefore $\xi_i = l_i = H_{iy\bar{y}}$. It is true then

$$\alpha_{iy\bar{y}}(H_{iy\bar{y}} - \xi_i) = 0 \qquad (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i)\,. \tag{4.35}$$

In order to prove Equation (4.25d), suppose $\zeta_i > 0$. Then by Equation (4.30), $u_i \leq 0$ if $\zeta_i > 0$. Because $0 \leq l_i = u_i \leq 0$, we get $\xi_i = l_i = 0$. Thus we have

$$\zeta_i \xi_i = 0 \qquad (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i) \tag{4.36}$$

Equation (4.33), (4.34), (4.35), and (4.36) show Equation (4.25) is true when $\psi_i = 0$ $(\forall i)$ for a feasible $\boldsymbol{\alpha}$. $\square$

Proposition 3 justifies a selection strategy that selects training pattern for which ψ_i is maximal. Intuitively ψ_i measures how strongly the margin constraint $\gamma_i(\mathbf{w}) \geq 1 - \xi_i$ is violated for the current solutions $\mathbf{w} = \mathbf{w}(\boldsymbol{\alpha})$. Once we have selected the i-th example we select the class pair $y \in Y_i, \bar{y} \in \bar{Y}_i$ for which $H_{iy\bar{y}}$ is maximal and add the variable $\alpha_{iy\bar{y}}$ to the optimization.

Moreover, the following proposition sheds some light on why it is reasonable to work on the reduced problem.

Proposition 4. *Given a set of selected variables $S \subseteq \{(i, y, \bar{y}) : i = 1, \ldots, n,\ y \in Y_i, \bar{y} \in \bar{Y}_i\}$ and an optimal solution $\boldsymbol{\alpha}^*$ of the dual QP over the reduced problem, i.e. the problem where implicitly $\alpha_{iy\bar{y}} = 0$ for all $(i, y, \bar{y}) \notin S$. Then $\boldsymbol{\alpha}^*$ is an optimal solution to the full QP in Equation* (4.15) *if and only if $\psi_i = 0$ for all $i = 1, \ldots, n$.*

Proof. Since $\boldsymbol{\alpha}^*$ is an optimal solution to the reduced problem over S, $\boldsymbol{\alpha}^*$ (with $\alpha_{iy\bar{y}} = 0$ for $(i, y, \bar{y}) \notin S$) is a feasible solution to the full problem. The proposition can then be proved by using results of Proposition 3. $\square$

More details on convergence and sparseness of a more general class of algorithms can be found in [62].

4.3.3 Implementation details

The previous discussion immediately leads to an optimization algorithm. Pseudo-code is shown in Algorithm 2. The sets S_i's keep track of the selected constraints

Algorithm 2 Hierarchical Multilabel SVM. The optimization algorithm uses variable selection and subspace optimization.

```
Inputs: training data $\{\mathbf{x}_i, Y_i\}_{i=1}^n$, tolerance $\epsilon \geq 0$
 1: initialize $S_i = \emptyset$, $\alpha_{iy\bar{y}} = 0$, $\forall\, i$, $y \in Y_i$, $\bar{y} \in \bar{Y}_i$.
 2: repeat
 3:     compute $H_{iy\bar{y}}$ from (4.24) and Equation $\psi_i$ from Equation (4.32)
 4:     select $\hat{i} = \text{argmax}_{i=1}^n \psi_i$
 5:     if $\psi_{\hat{i}} > \epsilon$ then
 6:         select $(\hat{y}, \hat{\bar{y}}) = \text{argmax}_{y \in Y_{\hat{i}}, \bar{y} \in \bar{Y}_{\hat{i}}} H_{\hat{i}y\bar{y}}$
 7:         expand working set: $S_{\hat{i}} = S_{\hat{i}} \cup \{(\hat{y}, \hat{\bar{y}})\}$
 8:         solve reduced QP over $\{\alpha_{\hat{i}y\bar{y}} : (y, \bar{y}) \in S_{\hat{i}}\}$ [8a]
            solve reduced QP over $\bigcup_{i=1}^n \{\alpha_{iy\bar{y}} : (y, \bar{y}) \in S_i\}$ [8b]
 9:         reduce working set: $S_{\hat{i}} = S_{\hat{i}} - \{(y, \bar{y}) : \alpha_{\hat{i}y\bar{y}} = 0\}$
10:     end if
11: until $\psi_{\hat{i}} \leq \epsilon$
```

for each training pattern. In step 4 and 6 the next constraint is selected. For the optimization in step 8 one can use any standard QP solver. In order to guarantee convergence in a finite number of steps, one would need to optimize over all selected variables in step (8b). However, in practice we propose to use a variant based on step (8a) which only optimizes over the subspace of the variables in $S_{\hat{i}}$, i.e. the active dual variables belonging to the selected training instance. In order to carry out step (8a), we have used the LOQO optimization package [63] in our experiments. The tolerance ϵ specifies the termination criterion based on the maximal optimization violation, although in practice one might use other heuristics to stop the training process, if one is only interested in an approximate solution.

Finally, note that one can keep track of the quantities $H_{iy\bar{y}}$ and incrementally update their values after each optimization step, since only the $\alpha_{iy\bar{y}}$ parameters for the selected i-th training instance change, while the other dual variables remain frozen at their current values. Introducing these auxiliary variables prevents the undue computational load of naively evaluating the variable selection criterion.

4.4 Taxonomy-derived regularizer

In the section, we will view the hierarchical SVM formulation from a different angle. We will transform the hierarchical SVM formula into one with the standard class weight vector notation, and show that indeed the hierarchical SVM employs a taxonomy-derived regularizer.

To remind the reader, the discriminant function for the hierarchical SVM is

$$F(\mathbf{x}, y; \mathbf{w}) = \sum_{z:z \in \mathrm{anc}(y)} \lambda_y(z) \langle \mathbf{w}_z, \mathbf{x} \rangle = \langle \mathbf{W}_y, \mathbf{x} \rangle \text{ where } \mathbf{W}_y = \sum_{z:z \in \mathrm{anc}(y)} \mathbf{w}_z \, .$$

We call $\mathbf{w}_v$ $(v \in \mathcal{V})$ a node weight vector and $\mathbf{W}_y$ $(y \in \mathcal{Y})$ a class weight vector. Although $\mathbf{w}_v$'s are sufficient to describe the hierarchical SVM formula, it is of conceptual importance to represent the hierarchical SVM in terms of classical class weight vectors since this is what is used in standard SVMs as well as in the flat multilabel SVMs.

We define the set of all interior nodes by $\mathcal{O} = \mathcal{V} - \mathcal{Y} = \{o_1, o_2, \cdots, o_l\}$. Thus the hard-margin hierarchical SVM in Equation (4.9) can be rewritten as

$$\min_{\mathbf{w}} \ \frac{1}{2} \left(\sum_{y \in \mathcal{Y}} \|\mathbf{w}_y\|^2 + \sum_{o \in \mathcal{O}} \|\mathbf{w}_o\|^2 \right) \tag{4.37a}$$

$$\text{s.t.} \quad \langle \mathbf{W}_y, \mathbf{x}_i \rangle - \langle \mathbf{W}_{\bar{y}}, \mathbf{x}_i \rangle) \geq 1 \, , \quad (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i) \, . \tag{4.37b}$$

Next, we will replace the node weight vector variables ($\mathbf{w}_y$'s and $\mathbf{w}_o$'s) in the objective function with class weight vectors ($\mathbf{W}_y$'s). Substituting

$$\mathbf{w}_y = \frac{\mathbf{W}_y - \sum_{o \in \mathcal{O} \cap \mathrm{anc}(y)} \lambda_o(y) \mathbf{w}_o}{\lambda_y(y)} \tag{4.38}$$

into the objective function, we have

$$
\begin{aligned}
G \equiv & \sum_{y\in\mathcal{Y}} \|\mathbf{w}_y\|^2 + \sum_{o\in\mathcal{O}} \|\mathbf{w}_o\|^2 \\
= & \sum_{y\in\mathcal{Y}} \left\| \frac{\mathbf{W}_y - \sum_{o\in\mathcal{O}\cap\mathrm{anc}(y)} \lambda_o(y)\mathbf{w}_o}{\lambda_y(y)} \right\|^2 + \sum_{o\in\mathcal{O}} \|\mathbf{w}_o\|^2 \\
= & \sum_{y\in\mathcal{Y}} \left(\frac{1}{\lambda_y(y)^2}\|\mathbf{W}_y\|^2 - 2\sum_{o\in\mathcal{O}\cap\mathrm{anc}(y)} \frac{\lambda_o(y)}{\lambda_y(y)^2}\langle \mathbf{W}_y, \mathbf{w}_o\rangle + \sum_{o,o':\substack{o\in\mathcal{O}\cap\mathrm{anc}(y)\\ o'\in\mathcal{O}\cap\mathrm{anc}(y)}} \frac{\lambda_o(y)\lambda_{o'}(y)}{\lambda_y(y)^2}\langle \mathbf{w}_o, \mathbf{w}_{o'}\rangle \right) \\
& + \sum_{o\in\mathcal{O}} \|\mathbf{w}_o\|^2 \\
= & \sum_{y\in\mathcal{Y}} \frac{1}{\lambda_y(y)^2}\|\mathbf{W}_y\|^2 - \sum_{o\in\mathcal{O}} \left\langle \sum_{y:\substack{o\in\mathrm{anc}(y)\\ y\in\mathcal{Y}}} 2\frac{\lambda_o(y)}{\lambda_y(y)^2}\mathbf{W}_y, \mathbf{w}_o \right\rangle \\
& + \sum_{o\in\mathcal{O}} \left(1 + \sum_{y:\substack{o\in\mathrm{anc}(y)\\ y\in\mathcal{Y}}} \frac{\lambda_o(y)^2}{\lambda_y(y)^2} \right) \|\mathbf{w}_o\|^2 \\
& + \sum_{o,o':\substack{o\neq o'\\ o\in\mathcal{O}\\ o'\in\mathcal{O}}} \left[\left(\sum_{\substack{o\in\mathrm{anc}(y)\\ y:o'\in\mathrm{anc}(y)\\ y\in\mathcal{Y}}} \frac{\lambda_o(y)\lambda_{o'}(y)}{\lambda_y(y)^2} \right) \langle \mathbf{w}_o, \mathbf{w}_{o'}\rangle \right]
\end{aligned}
\tag{4.39}
$$

Let $\mathbf{w}_{\mathcal{O}} = (\mathbf{w}_{o_1}^T, \ldots, \mathbf{w}_{o_l}^T)^T$ be the stacked weight vectors of all interior nodes. Let $\mathbf{A}$ be a matrix of l rows and l columns, with

$$
A_{kt} = \begin{cases} 1 + \sum_{y:\substack{o_k\in\mathrm{anc}(y)\\ y\in\mathcal{Y}}} \frac{\lambda_{o_k}(y)^2}{\lambda_y(y)^2} & \text{if } k = t \\ \sum_{\substack{o_k\in\mathrm{anc}(y)\\ y:o_t\in\mathrm{anc}(y)\\ y\in\mathcal{Y}}} \frac{\lambda_{o_k}(y)\lambda_{o_t}(y)}{\lambda_y(y)^2} & \text{if } k \neq t \end{cases}
\tag{4.40}
$$

The matrix $\mathbf{A}$ depends on the number of descendent classes that two interior nodes share. Let

$$
\tilde{\mathbf{A}} \equiv \mathbf{A} \otimes \mathbf{I}_{d\times d} \tag{4.41}
$$

be a matrix of size ld by ld, where $\mathbf{I}_{d\times d}$ is an identity matrix of size d by d. Note d is the dimension of feature space.

Let $\mathbf{u}(\mathbf{W}) = (\mathbf{u}_1^T, \ldots, \mathbf{u}_l^T)^T$ be a vector of length ld, where the component vector is defined as

$$\mathbf{u}_k = \sum_{y: o_k \in \operatorname{anc}(y),\, y \in \mathcal{Y}} \frac{\lambda_{o_k}(y)}{\lambda_y(y)^2} \mathbf{W}_y \tag{4.42}$$

Note $\mathbf{u}_k$ equals a linear combination of weight vectors of classes that are descendents of node o_k.

With $\tilde{\mathbf{A}}$ and $\mathbf{u}$ introduced, the quantity G in Equation (4.39) can be written as

$$G = \sum_{y \in \mathcal{Y}} \frac{1}{\lambda_y(y)^2} \|\mathbf{W}_y\|^2 - 2\mathbf{u}(\mathbf{W})^T \mathbf{w}_{\mathcal{O}} + \mathbf{w}_{\mathcal{O}}^T \tilde{\mathbf{A}} \mathbf{w}_{\mathcal{O}} \,. \tag{4.43}$$

For a given $\mathbf{W}$, equation (4.43) is an unconstrained quadratic convex function of $\mathbf{w}_{\mathcal{O}}$, with minimum value

$$G^* = \sum_{y \in \mathcal{Y}} \frac{1}{\lambda_y(y)^2} \|\mathbf{W}_y\|^2 - \mathbf{u}(\mathbf{W})^T \tilde{\mathbf{A}}^{-1} \mathbf{u}(\mathbf{W}) \,, \tag{4.44}$$

which is achieved at

$$\mathbf{w}_{\mathcal{O}}^* = \tilde{\mathbf{A}}^{-1} \mathbf{u}(\mathbf{W}) \tag{4.45}$$

By definition of $\tilde{\mathbf{A}}$, $\tilde{\mathbf{A}}^{-1} = \mathbf{A}^{-1} \otimes \mathbf{I}_{d \times d}$. Thus the hierarchical SVM in equation (4.37) can be rewritten in a form parameterized by $\mathbf{W}_y$'s. We therefore arrive at the following equivalent quadratic program.

$$\min_{\mathbf{W}} \frac{1}{2} \left(\sum_{y \in \mathcal{Y}} \frac{1}{\lambda_y(y)^2} \|\mathbf{W}_y\|^2 - \mathbf{u}(\mathbf{W})^T \tilde{\mathbf{A}}^{-1} \mathbf{u}(\mathbf{W}) \right) \tag{4.46a}$$

$$\text{s.t. } \langle \mathbf{W}_y - \mathbf{W}_{\bar{y}}, \mathbf{x}_i \rangle \geq 1 \qquad (\forall i, y \in \mathcal{Y}_i, \bar{y} \in \bar{Y}_i) \,. \tag{4.46b}$$

In Equation (4.46), matrix $\tilde{\mathbf{A}}$ only depends on the class attribute vectors and the taxonomy, so it is a given matrix for given optimization problem.

Compared to the flat SVM, the hierarchical SVM (Equation (4.46)) has an extra term $-\mathbf{u}(\mathbf{W})^T \tilde{\mathbf{A}}^{-1} \mathbf{u}(\mathbf{W})$ to minimize, which couples the weights of classes depending on how they are located in the taxonomy. Therefore, the hierarchical SVM encourages similarity or dissimilarity between class weight vectors based on the class locations in the given taxonomy.

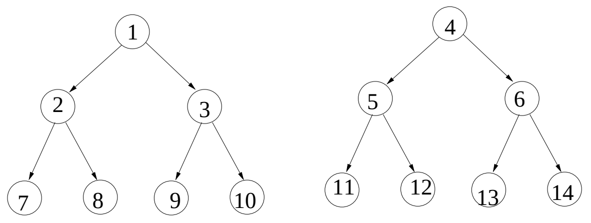


Figure 4.5: A taxonomy comprising two perfect trees of depth 2. Node $1, 2, 3, 4, 5, 6$ are interior nodes and $7, 8, 9, 10, 11, 12, 13, 14$ are leaf nodes

In the more general case, the class attributes can be arbitrarily defined. The above deduction still holds for cases where any class is not represented by any other class node, that is $\lambda_{y'}(y) = 0$ if $y \neq y'$. Equation (4.40) and (4.42) will be the same except that the sum is now over all y's instead of just the y's under particular interior nodes.

To get a clearer view of how hierarchical SVM ties the weight vectors of classes, here we give a concrete example for the taxonomy depicted in Figure 4.5. The hierarchy comprises two *perfect trees* in which all leaf nodes are at the same depth and all interior nodes have the same number of child nodes. To simplify the discussion, we assume

$$\lambda_z(y) = \begin{cases} \lambda & \text{if } z \in \mathrm{anc}(y) \\ 0 & \text{otherwise} \end{cases} .$$

Then

$$\begin{aligned} &\mathbf{u}(\mathbf{W})^T \tilde{\mathbf{A}}^{-1} \mathbf{u}(\mathbf{W}) \\ =&\frac{1}{\lambda^2}\Big(\frac{\|\mathbf{W}_7 + \mathbf{W}_8 + \mathbf{W}_9 + \mathbf{W}_{10}\|^2}{3 \times 7} + \frac{\|\mathbf{W}_7 + \mathbf{W}_8\|^2}{3} + \frac{\|\mathbf{W}_9 + \mathbf{W}_{10}\|^2}{3} \\ &+ \frac{\|\mathbf{W}_{11} + \mathbf{W}_{12} + \mathbf{W}_{13} + \mathbf{W}_{14}\|^2}{3 \times 7} + \frac{\|\mathbf{W}_{11} + \mathbf{W}_{12}\|^2}{3} + \frac{\|\mathbf{W}_{13} + \mathbf{W}_{14}\|^2}{3} \Big) . \end{aligned} \tag{4.47}$$

The details of computation for this hierarchy as well as for general perfect tree of depth 2 are in Appendix A. Note for any vectors of fixed norms, their summation has the largest norm if the vectors point to the same direction. In this example we observe the closer two classes are in the taxonomy, the harder the hierarchical SVM learning tries to make their class prototype vectors similar to each other.

By the same deduction, the soft-margin hierarchical SVM in Equation (4.10) can be rewritten as

$$\min_{\mathbf{W},\boldsymbol{\xi}} \frac{1}{2}\left(\sum_{y\in\mathcal{Y}} \frac{1}{\lambda_y(y)^2}\|\mathbf{W}_y\|^2 - \mathbf{u}(\mathbf{W})^T\tilde{\mathbf{A}}^{-1}\mathbf{u}(\mathbf{W})\right) + C\sum_{i=1}^{n}\xi_i \tag{4.48a}$$

$$\text{s.t. } \langle \mathbf{W}_y - \mathbf{W}_{\bar{y}}, \mathbf{x}_i\rangle \geq 1 - \frac{\xi_i}{\triangle(y,\bar{y})} \qquad (\forall i, y\in\mathcal{Y}_i, \bar{y}\in\bar{Y}_i)\,, \tag{4.48b}$$

$$\xi_i \geq 0, \quad (\forall i)\,. \tag{4.48c}$$

The discussion above comparing the flat SVM and the hierarchical SVM still holds.

4.5 Related work

A considerable number of approaches have been proposed on hierarchical classification [23, 31, 10], based on various classical learning methods such as neural networks, probabilistical models and SVMs. Many approaches for hierarchical classification use a divide-and-conquer strategy to first classify instances on a coarse level and then on successively finer levels. While this offers advantages in terms of modularity, the local optimization of (partial) classifiers at every inner node is unable to reflect a more global objective.

Koller and Sahami [31] employed this strategy in conjunction with probabilistic classifier (naive Bayes [35] and slightly less naive versions thereof), which are trained at each split node in the hierarchy. The classification decision is thus decomposed into a number of local routing or refinement decisions in the taxonomy. In addition, [31] proposes feature selection at every refinement level and they show that locally only a small number of discriminative features may be sufficient to achieve reasonable classification accuracy. Follow-up work on the feature selection aspect has been performed by Mladenik and Grobelnik [42]. The downside of this approach is the use of a less competitive classifier, naive Bayes, together with an independent training process for each refinement classifier. The latter may lead to suboptimal discrimination, since the classifiers are finally operated in a specific architecture that combines their outputs. Even more severe are the disadvantages of the greedy decision process, which does not allow to recover from incorrect routing decision made at higher levels of the hierarchy. Improved non-greedy techniques have been investigated by Charkabarti et al. [12].

Divide-and-conquer strategies in conjunction with SVM classifiers have been proposed by Dumais and Chen [23] and by Sun and Lim [58]. Again, classifiers are trained independently and their outputs are combined by integrating scores along each path. In [23] a sigmoidal transformation is applied to derive estimates of posterior probabilities. These probabilities are then either thresholded independently or combined multiplicatively and then thresholded. In [58] a heuristic is developed to select training instances for training each refinement classifier. [23] also use a feature selection strategy similar to [31].

Hierarchical neural networks architectures have been utilized by Ruiz and Srinivasan [52] as well as by Weigend, Wiener, and Pedersen [66]. In both approaches a quite aggressive feature selection and/or dimension reduction step is necessary in order to reduce the number of input weights in the neural network. The training of networks at different levels is again performed independently using back-propagation, leading to the same problems that were mentioned above.

In the context of naive Bayes classification, McCallum et al. [41] have proposed the use of 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. Toutanova et al. [61] have developed an improved Expectation Maximization algorithm that refines the technique of [41]. The main downside of this line of work is that naive Bayes classifiers are often not competitive and the gain from using the hierarchy is often less than the loss in accuracy suffered relative to more competitive methods like SVMs.

[9] introduces a loss function, called the H-loss, specifically designed to deal with the case of partial and overlapping paths in tree-structured taxonomies. [10] has proposed B-SVM, which also uses the H-loss and uses a decoding scheme that explicitly computes the Bayes-optimal label assignment based on the H-loss and certain conditional independence assumptions about label paths. The loss function we proposed exploits the taxonomy in a different way from H-loss, partly because we always convert partial path categories to complete path categories. Our loss function is inspired by real applications like routing and subscription to a taxonomy. Thus misclassifications are penalized along all ancestors that miss relevant patterns or include irrelevant

ones. In H-loss, however, if punishment already occurs to a node, its descendents are not penalized again. In addition, our loss function works with arbitrary taxonomy, not just trees.

[19] presents an online algorithm for multiclass hierarchical classification. It decomposes the weight vectors into contributions of taxonomy nodes in the same way that we do. However, it only deals with trees. Their method employs a conservative update rule: when a pattern is wrongly classified, then the weight vectors associated with the correct class and the wrong class and some of their ancestors will be updated; the update coefficient is determined by solving a quadratic optimization problem where a hierarchical loss is imposed.

[51] applies the Maximum-Margin Markov Networks [60] to hierarchical classification where the taxonomy is regarded as Markov Networks. They propose a simplified version of H-loss that decomposes into contributions of edges so as to marginalize the exponential-sized problem into a polynomial one. In our methods, learning occurs on taxonomy nodes instead of edges. We view the taxonomy as a dependency graph of “is-a” relation.

Finally, we would like to point out that our method is a natural generalization of the multiclass SVM formulation proposed in [14, 67]. We employ the category ranking approach proposed in [54] to deal with the additional challenge posed by overlapping categories, i.e. the multi-label problem. Our work is a particularly interesting special case of a more general learning architecture presented in [62].

Chapter 5

Hierarchical Perceptron

The Perceptron algorithm and Support Vector Machines possess different strengths. As seen in Chapter 4, training a Support Vector Machine requires solving a large and complex quadratic optimization problem. Although SVMs generalize better than Perceptron, it is usually slower than the Perceptron algorithm. The Perceptron algorithm also naturally fits into the *online* learning paradigm in which training examples are encountered one at a time and correction to the discriminant function is made before receiving the next example. The Perceptron algorithm is therefore memory efficient since it processes one example at a time. Another advantage is that the Perceptron algorithm is very easy to understand and implement.

In this chapter, we first describe the flat multilabel Perceptron, then generalize it to our online hierarchical Perceptron, present two batch variants of the online hierarchical Perceptron, and give related work.

5.1 Flat multilabel Perceptron

We extend the multiclass Perceptron in Algorithm 1 to Algorithm 3 to deal with multilabel learning. In multilabel classification, a pattern can be relevant with multiple classes. In step 4, $(\hat{y}, \hat{\bar{y}})$ is selected to be the pair of positive and negative classes that achieves the multilabel margin $\gamma_i(\mathbf{w})$ for the i-th instance, as defined in Equation (4.1). If the margin is not greater than 0, it indicates that there are errors in the ranking. The current discriminant function is then updated by adding the current

Algorithm 3 Flat Multilabel Perceptron

```
Inputs: training data $\{\mathbf{x}_i, Y_i\}_{i=1}^n$; maximal number of epochs $\kappa$.
 1: Initialize $\mathbf{w}_y \leftarrow \mathbf{0}$, $\forall y$.
 2: for $num = 1$ to $\kappa$ do
 3:   for all training instance $\mathbf{x}_i$ do
 4:     $(\hat{y}, \hat{\bar{y}}) = \mathrm{argmin}_{y \in Y_i, \bar{y} \in \bar{Y}_i} \langle \mathbf{w}_y, \mathbf{x}_i \rangle - \langle \mathbf{w}_{\bar{y}}, \mathbf{x}_i \rangle$
 5:     if $\langle \mathbf{w}_{\hat{y}}, \mathbf{x}_i \rangle - \langle \mathbf{w}_{\hat{\bar{y}}}, \mathbf{x}_i \rangle \leq 0$ then
 6:       $\mathbf{w}_{\hat{y}} \leftarrow \mathbf{w}_{\hat{y}} + \mathbf{x}_i$.
 7:       $\mathbf{w}_{\hat{\bar{y}}} \leftarrow \mathbf{w}_{\hat{\bar{y}}} - \mathbf{x}_i$.
 8:     end if
 9:   end for
10:   if no update is performed in current epoch then
11:     terminate with a satisfactory solution
12:   end if
13: end for
```

instance to $\mathbf{w}_{\hat{y}}$ and subtracting the current instance from $\mathbf{w}_{\hat{\bar{y}}}$.

In Perceptron learning, an update strategy is called *conservative* if an update to the discriminant function can only be made if the current hypothesis makes a mistake on the current instance. An update strategy for multilabel Perceptron is called *ultraconservative* [15] if it is not only conservative, but it also leaves the weight vectors of classes that are not in the error set of the current instance intact. The *error set* of instance $\mathbf{x}_i$ is defined as [15]

$$\begin{aligned} E^i =& \{y \mid y \in Y_i \wedge \exists \bar{y} \text{ s.t. } \langle \mathbf{w}_y, \mathbf{x}_i \rangle - \langle \mathbf{w}_{\bar{y}}, \mathbf{x}_i \rangle \leq 0\} \\ & \cup \{\bar{y} \mid \bar{y} \in \bar{Y}_i \wedge \exists y \text{ s.t. } \langle \mathbf{w}_y, \mathbf{x}_i \rangle - \langle \mathbf{w}_{\bar{y}}, \mathbf{x}_i \rangle \leq 0\} \,. \end{aligned} \tag{5.1}$$

It means that E^i contains all classes that are mis-ranked for instance $\mathbf{x}_i$ by the current classifier. The flat multilabel Perceptron in Algorithm 3 is an ultraconservative algorithm. In fact, it falls into a family of ultraconservative online algorithms presented in [15].

When a set of training data is given in advance, the traditional way that the Perceptron algorithm uses it is by making a number of passes over the whole training set. Each pass is called an *epoch*. The maximal number of epochs is denoted by κ in Algorithm 3.

Algorithm 4 Online hierarchical multilabel Perceptron algorithm in primal form

Inputs: training data $\{\mathbf{x}_i, Y_i\}_{i=1}^n$, maximal number of epochs κ.

```
1: Initialize $\mathbf{w} \leftarrow \mathbf{0}$.
2: for $num = 1$ to $\kappa$ do
3:   for all training instance $\mathbf{x}_i$ do
4:     $(\hat{y}, \hat{\bar{y}}) = \mathrm{argmin}_{y \in Y_i, \bar{y} \in \bar{Y}_i} \langle \mathbf{w}, \delta\mathbf{\Phi}_i(y, \bar{y}) \rangle$
5:     if $\langle \mathbf{w}, \delta\mathbf{\Phi}_i(\hat{y}, \hat{\bar{y}}) \rangle \le 0$ then
6:       $\mathbf{w} \leftarrow \mathbf{w} + \triangle(\hat{y}, \hat{\bar{y}})\delta\mathbf{\Phi}_i(\hat{y}, \hat{\bar{y}})$
7:     end if
8:   end for
9:   if no update is performed in current epoch then
10:    terminate with a satisfactory solution
11:  end if
12: end for
```

5.2 Online hierarchical multilabel Perceptron

5.2.1 Primal form

To remind the reader, we encode a taxonomy in two ways. One is taxonomy-derived class attributes, which are manifested in the structure of the stacked weight vector $\mathbf{w}$ and in the joint feature representation $\mathbf{\Phi}(\mathbf{x}, y)$. The other is the taxonomy-based loss function. The goal of online hierarchical multilabel Perceptron is to extend the flat algorithm to incorporate the two encodings. We propose to do so by Algorithm 4.

To remind the readers, $\delta\mathbf{\Phi}_i(y, \bar{y}) \equiv \mathbf{\Phi}(\mathbf{x}_i, y) - \mathbf{\Phi}(\mathbf{x}_i, \bar{y})$, where $\mathbf{\Phi}(\mathbf{x}_i, y)$ is the joint feature representation of instance $\mathbf{x}_i$ and class y, and $\langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x}_i, y) \rangle$ is the scoring function of class y. In Algorithm 4, the selection of $(\hat{y}, \hat{\bar{y}})$ pair is modified according to the multilabel margin defined in Equation (4.4), which uses class attributes. The pair is composed of the relevant class that has the lowest compatibility score and the irrelevant class that has the highest score. A positive margin for a pattern indicates a correct ranking of all categories while a non-positive one indicts errors. The class attributes are also used in the update rule in step 6. In addition, the amount of update is scaled by the loss involved. So the more severe the incurred loss is, the more aggressive the update to the current classifier is. This strategy is also employed in [15]. In the special case of indicator class attributes and 0/1 loss, step 6 in the online hierarchical Perceptron algorithm can be verified to reduce to the update step in the flat multilabel Perceptron.

To better understand the update rule, let us expand the update vector to

$$\begin{aligned}&\triangle(\hat{y},\hat{\hat{y}})\delta\mathbf{\Phi}_i(\hat{y},\hat{\hat{y}})\\ =&\triangle(\hat{y},\hat{\hat{y}})\left(\mathbf{\Lambda}(\hat{y})\otimes\mathbf{x}_i-\mathbf{\Lambda}(\hat{\hat{y}})\otimes\mathbf{x}_i\right)\\ =&\triangle(\hat{y},\hat{\hat{y}})\left(\,(\lambda_1(\hat{y})-\lambda_1(\hat{\hat{y}}))\mathbf{x}_i^T,(\lambda_2(\hat{y})-\lambda_2(\hat{\hat{y}}))\mathbf{x}_i^T,\cdots,(\lambda_s(\hat{y})-\lambda_s(\hat{\hat{y}}))\mathbf{x}_i^T\,\right)^T\,.\end{aligned}\tag{5.2}$$

Therefore the update to the stacked weight vector can be decomposed to a set of updates to individual weight vectors as

$$\mathbf{w}_z\leftarrow\mathbf{w}_z+\triangle(\hat{y},\hat{\hat{y}})\left(\lambda_z(\hat{y})-\lambda_z(\hat{\hat{y}})\right)\mathbf{x}_i\quad(\forall z=1,\cdots,s)\,.\tag{5.3}$$

To simplify discussion, suppose the following hierarchical attribute vector is used.

$$\lambda_z(y)=\begin{cases}1 & \text{if } z\in\text{anc}(y)\\ 0 & \text{otherwise}\end{cases}\qquad(\forall\, y\in\mathcal{Y},\, z\in\mathcal{V})\tag{5.4}$$

Then Equation (5.3) becomes

$$\mathbf{w}_z\leftarrow\begin{cases}\mathbf{w}_z+\triangle(\hat{y},\hat{\hat{y}})\mathbf{x}_i & \text{if } z\in\text{anc}(\hat{y})\setminus\text{anc}(\hat{\hat{y}})\\ \mathbf{w}_z-\triangle(\hat{y},\hat{\hat{y}})\mathbf{x}_i & \text{if } z\in\text{anc}(\hat{\hat{y}})\setminus\text{anc}(\hat{y})\\ \mathbf{w}_z & \text{otherwise}\end{cases}\,.\tag{5.5}$$

Only the weight vectors of those nodes that are predecessors of $\hat{y}$ or $\hat{\hat{y}}$ but not both will be updated. Other nodes are left intact. This strategy is also used in [19] for online multiclass classification. The more severe the loss is incurred, the more dramatic the update will be. Although our algorithm uses the mere loss function as the update coefficient, the experiments show that it performs competitively with the considerably more complicated technique employed in [19]. Moreover, step 6 of Algorithm 4 not only updates the scoring functions of the two classes in question, but also spreads the impact to other classes sharing affected ancestors with them. Therefore, the hierarchical Perceptron is not ultraconservative.

5.2.2 Dual form

In the online hierarchical Perceptron algorithm, the stacked weight vector is initialized to $\mathbf{0}$. The update rule changes the current weight vector only by adding

Algorithm 5 Online hierarchical multilabel Perceptron algorithm in dual form

Inputs: training data $\{\mathbf{x}_i, Y_i\}_{i=1}^n$, maximal number of epochs κ.

```
1:  Initialize $\alpha_{iy\bar{y}} = 0$, $\forall\, i,\ y \in Y_i,\ \bar{y} \in \bar{Y}_i$.
2:  for $num = 1$ to $\kappa$ do
3:      for all training instance $\mathbf{x}_i$ do
4:          $(\hat{y}, \hat{\bar{y}}) = \mathrm{argmin}_{y \in Y_i, \bar{y} \in \bar{Y}_i} \langle \mathbf{w}(\boldsymbol{\alpha}), \delta\boldsymbol{\Phi}_i(y, \bar{y}) \rangle$
5:          if $\langle \mathbf{w}(\boldsymbol{\alpha}), \delta\boldsymbol{\Phi}_i(\hat{y}, \hat{\bar{y}}) \rangle \leq 0$ then
6:              $\alpha_{i\hat{y}\hat{\bar{y}}} \leftarrow \alpha_{i\hat{y}\hat{\bar{y}}} + \triangle(\hat{y}, \hat{\bar{y}})$
7:          end if
8:      end for
9:      if no update is performed in current epoch then
10:         terminate with a satisfactory solution
11:     end if
12: end for
```

$\triangle(\hat{y}, \hat{\bar{y}})\delta\boldsymbol{\Phi}_i(\hat{y}, \hat{\bar{y}})$ whenever the margin of $\mathbf{x}_i$ (achieved at $(\hat{y}, \hat{\bar{y}})$) is not greater than 0, i.e. the category ranking for instance $\mathbf{x}_i$ is wrong. Hence the weight vector can be written as

$$\mathbf{w}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \sum_{y \in Y_i} \sum_{\bar{y} \in \bar{Y}_i} \alpha_{iy\bar{y}} \delta\boldsymbol{\Phi}_i(y, \bar{y}) \qquad (\alpha_{iy\bar{y}} \geq 0\ \forall i, y \in Y_i, \bar{y} \in \bar{Y}_i)\,, \tag{5.6}$$

where $\frac{\alpha_{iy\bar{y}}}{\triangle(y,\bar{y})}$ equals the number of times that the category ranking of instance $\mathbf{x}_i$ is predicted incorrectly, with $(y, \bar{y})$ being the most wrongly ordered relevant class and irrelevant class pair. Note $\sum_{i=1}^{n} \sum_{y \in Y_i} \sum_{\bar{y} \in \bar{Y}_i} \frac{\alpha_{iy\bar{y}}}{\triangle(y,\bar{y})}$ is the total number of updates performed. With the dual parameterization, we can replace all occurrences of $\mathbf{w}$, the primal variable, with $\boldsymbol{\alpha}$, the dual variable in Algorithm 4. This leads to the dual form of online hierarchical Perceptron in Algorithm 5. Since in Algorithm 5,

$$\begin{aligned}
\mathbf{w}(\boldsymbol{\alpha}), \delta\boldsymbol{\Phi}_i(y, \bar{y})\rangle &= \sum_{j=1}^{n} \sum_{r \in Y_j} \sum_{\bar{r} \in \bar{Y}_j} \langle \delta\boldsymbol{\Phi}_j(r, \bar{r}), \delta\boldsymbol{\Phi}_i(y, \bar{y}) \rangle \\
&= \sum_{j=1}^{n} \sum_{r \in Y_j} \sum_{\bar{r} \in \bar{Y}_j} \langle\, \boldsymbol{\Lambda}(r) - \boldsymbol{\Lambda}(\bar{r}), \boldsymbol{\Lambda}(y) - \boldsymbol{\Lambda}(\bar{y}) \,\rangle\, \langle \mathbf{x}_i, \mathbf{x}_j \rangle\,,
\end{aligned} \tag{5.7}$$

the training data are only used in inner product form. This means that we can again utilize kernel functions in place of the inner products to learn non-linear discriminant functions efficiently.

5.2.3 Convergence

The hierarchical Perceptron in Algorithm 4 will stop if all instances are predicted correctly. This means that the algorithm converges to a correct solution. Proposition 5 below shows that the online hierarchical Perceptron algorithm converges in a finite number of steps as long as a correct solution exists. The proposition also gives an upper bound on the number of update steps that are needed before convergence.

Proposition 5. *Let $\{\mathbf{x}_i, Y_i\}_{i=1}^n$ be a training set. Set $R \equiv \max_i \|\mathbf{x}_i\|$, $\triangle_{\min} \equiv \min_{y \neq \hat{y}} \triangle_(y, \hat{y}) > 0$, $\triangle_{\max} \equiv \max_{y \neq \hat{y}} \triangle_(y, \hat{y})$, $P \equiv \max_y \|\mathbf{\Lambda}(y)\|$. Assume that a weight vector $\mathbf{w}^*$ exists with $\|\mathbf{w}^*\| = 1$ that predicts correct rankings on all training data with a margin*

$$\gamma = \min_i \left(\min_{y \in Y_i, \bar{y} \in \bar{Y}_i} \langle \mathbf{w}^*, \delta\mathbf{\Phi}_i(y, \bar{y}) \rangle \right) > 0 \,.$$

Then the online hierarchical Perceptron algorithm will not make more than $\left(\frac{2\triangle_{\max} RP}{\triangle_{\min}\gamma}\right)^2$ update steps.

Proof. Denote the weight vector after the t-th update by $\mathbf{w}^{(t)}$ and let $\mathbf{w}^{(0)} = 0$. At the t-th update step, the instance under investigation is denoted by $\mathbf{x}^{(t-1)}$ and the pair that achieves the margin for the current instance by $(y^{(t-1)}, \bar{y}^{(t-1)})$. Since an update is performed, we know

$$\langle \mathbf{w}^{(t-1)} \ , \ \mathbf{\Phi}(\mathbf{x}^{(t-1)}, y^{(t-1)}) - \mathbf{\Phi}(\mathbf{x}^{(t-1)}, \bar{y}^{(t-1)}) \rangle \leq 0 \,. \tag{5.8}$$

We prove the proposition by bounding $\|\mathbf{w}^{(t)}\|$ from above and below. This technique is due to Novikoff [45] for binary Perceptron convergence proof. First we compute the lower bound. Applying the update rule,

$$\begin{aligned}\langle \mathbf{w}^{(t)}, \mathbf{w}^* \rangle =& \left\langle \mathbf{w}^{(t-1)} + \triangle(y^{(t-1)}, \bar{y}^{(t-1)}) \left(\mathbf{\Phi}(\mathbf{x}^{(t-1)}, y^{(t-1)}) - \mathbf{\Phi}(\mathbf{x}^{(t-1)}, \bar{y}^{(t-1)}) \right) \ , \ \mathbf{w}^* \right\rangle \\ =& \langle \mathbf{w}^{(t-1)}, \mathbf{w}^* \rangle + \triangle(y^{(t-1)}, \bar{y}^{(t-1)}) \left\langle \mathbf{w}^* \ , \ \left(\mathbf{\Phi}(\mathbf{x}^{(t-1)}, y^{(t-1)}) - \mathbf{\Phi}(\mathbf{x}^{(t-1)}, \bar{y}^{(t-1)}) \right) \right\rangle\end{aligned}$$

Since $\mathbf{w}^*$ obtains a margin of γ on the entire training set,

$$\langle \mathbf{w}^{(t)}, \mathbf{w}^* \rangle \ \geq \ \langle \mathbf{w}^{(t-1)}, \mathbf{w}^* \rangle + \triangle(y^{(t-1)}, \bar{y}^{(t-1)})\gamma \ \geq \ \langle \mathbf{w}^{(t-1)}, \mathbf{w}^* \rangle + \triangle_{\min}\gamma \,.$$

By induction,

$$\langle \mathbf{w}^{(t)}, \mathbf{w}^* \rangle \geq \langle \mathbf{w}^{(0)}, \mathbf{w}^* \rangle + t\triangle_{\min}\gamma = t\triangle_{\min}\gamma\,.$$

Because $\|\mathbf{w}^*\| = 1$,

$$\|\mathbf{w}^{(t)}\| \geq \langle \mathbf{w}^{(t)}, \mathbf{w}^* \rangle \geq t\triangle_{\min}\gamma \tag{5.9}$$

In the next step, we compute the upper bound of $\|\mathbf{w}^{(t)}\|$.

$$\begin{aligned}
\|\mathbf{w}^{(t)}\|^2 = &\left\|\mathbf{w}^{(t-1)} + \triangle(y^{(t-1)}, \bar{y}^{(t-1)}) \left((\mathbf{\Phi}(\mathbf{x}^{(t-1)}, y^{(t-1)}) - \mathbf{\Phi}(\mathbf{x}^{(t-1)}, \bar{y}^{(t-1)})\right)\right\|^2 \\
=&\|\mathbf{w}^{(t-1)}\|^2 + \|\triangle(y^{(t-1)}, \bar{y}^{(t-1)})\|^2 \|\mathbf{\Phi}(\mathbf{x}^{(t-1)}, y^{(t-1)}) - \mathbf{\Phi}(\mathbf{x}^{(t-1)}, \bar{y}^{(t-1)})\|^2 \\
&+ 2\triangle(y^{(t-1)}, \bar{y}^{(t-1)}) \left\langle \ \mathbf{w}^{(t-1)}\ ,\ \mathbf{\Phi}(\mathbf{x}^{(t-1)}, y^{(t-1)}) - \mathbf{\Phi}(\mathbf{x}^{(t-1)}, \bar{y}^{(t-1)}) \ \right\rangle \\
\leq&\|\mathbf{w}^{(t-1)}\|^2 + 4\triangle_{\max}^2 R^2 P^2
\end{aligned}$$

The last inequality is due to Equation (5.8) and

$$\begin{aligned}
&\|\mathbf{\Phi}(\mathbf{x}^{(t-1)}, y^{(t-1)}) \ - \ \mathbf{\Phi}(\mathbf{x}^{(t-1)}, \bar{y}^{(t-1)})\|^2 \\
=&\|\mathbf{x}^{(t-1)}\|^2 \, \|\mathbf{\Lambda}(y^{(t-1)}) - \mathbf{\Lambda}(\bar{y}^{(t-1)})\|^2 \\
\leq& R^2 \left(\|\mathbf{\Lambda}(y^{(t-1)})\|^2 + \|\mathbf{\Lambda}(\bar{y}^{(t-1)})\|^2 - 2\langle \mathbf{\Lambda}(y^{(t-1)}), \mathbf{\Lambda}(\bar{y}^{(t-1)})\rangle \right) \\
\leq& R^2 \left(\|\mathbf{\Lambda}(y^{(t-1)})\|^2 + \|\mathbf{\Lambda}(\bar{y}^{(t-1)})\|^2 + 2\|\mathbf{\Lambda}(y^{(t-1)})\|\|\mathbf{\Lambda}(\bar{y}^{(t-1)})\| \right) \\
\leq& 4R^2P^2\,.
\end{aligned}$$

Again by induction, we get the upper bound

$$\|\mathbf{w}^{(t)}\|^2 \leq 4t\triangle_{\max}^2 R^2 P^2\,. \tag{5.10}$$

Finally combining Equations (5.9) and (5.10) we arrive at

$$t^2\triangle_{\min}^2\gamma^2 \leq \|\mathbf{w}^{(t)}\|^2 \leq 4t\triangle_{\max}^2 R^2 P^2\,. \tag{5.11}$$

Hence

$$t \leq \left(\frac{2\triangle_{\max}RP}{\triangle_{\min}\gamma}\right)^2\,. \tag{5.12}$$

$\square$

Algorithm 6 Averaged hierarchical Perceptron algorithm

```
Inputs: training data $\{\mathbf{x}_i, Y_i\}_{i=1}^n$, maximal number of epochs $\kappa$.
 1: Initialize $\alpha^0_{iy\bar{y}} = 0, \forall\, i,\, y \in Y_i,\, \bar{y} \in \bar{Y}_i$
 2: $l \leftarrow 0$
 3: for $num = 1$ to $\kappa$ do
 4:     for all training instance $\mathbf{x}_i$ do
 5:         $l \leftarrow l + 1$
 6:         $(\hat{y}, \hat{\bar{y}}) = \mathrm{argmin}_{y \in Y_i, \bar{y} \in \bar{Y}_i} \langle \mathbf{w}(\boldsymbol{\alpha}^{l-1}), \delta\boldsymbol{\Phi}_i(y, \bar{y}) \rangle$
 7:         $\boldsymbol{\alpha}^l \leftarrow \boldsymbol{\alpha}^{l-1}$
 8:         if $\langle \mathbf{w}(\boldsymbol{\alpha}^{l-1}), \delta\boldsymbol{\Phi}_i(\hat{y}, \hat{\bar{y}}) \rangle \leq 0$ then
 9:             $\alpha^l_{i\hat{y}\hat{\bar{y}}} \leftarrow \alpha^l_{i\hat{y}\hat{\bar{y}}} + \triangle(\hat{y}, \hat{\bar{y}})$
10:         end if
11:     end for
12:     if no update is performed in current epoch then
13:         break the loop.
14:     end if
15: end for
16: $L \leftarrow l$
17: output: $\boldsymbol{\alpha} \leftarrow \frac{1}{L} \sum_{k=1}^{L} \boldsymbol{\alpha}^k$
```

5.3 Batch hierarchical Perceptron algorithms

In the previous section, we presented a hierarchical Perceptron for online learning. For batch learning, in which the entire training set is given in advance, we could use the online algorithm directly. However, with some modifications, better-performing batch algorithms can be obtained. In this section, we present two batch hierarchical Perceptron algorithms: the averaged hierarchical Perceptron algorithm and the Minover hierarchical algorithm.

5.3.1 Averaged hierarchical Perceptron

The averaged hierarchical Perceptron differs from the online algorithm only in how the final classifier is constructed. The online hierarchical Perceptron takes the last weight vector while the averaged hierarchical Perceptron takes the average of all weight vectors generated by the training process. Discussion of hypothesis averaging can be found in [8].

The dual form of averaged hierarchical Perceptron is given in Algorithm 6. Notice the weight vector produced at the l-th step can be recovered by

$$\mathbf{w}^l = \sum_{i=1}^{n} \sum_{y \in Y_i} \sum_{\bar{y} \in \bar{Y}_i} \alpha_{iy\bar{y}}^l \delta \mathbf{\Phi}_i(y, \bar{y}) \tag{5.13}$$

and thus step 17 implies taking average of weight vectors of all steps.

In implementation, directly applying Algorithm 6 would be impractical, since $\boldsymbol{\alpha}$ generally is a vector of large size and L such vectors have to be kept. Instead, we only keep track of the updates and build the final classifier from these updates alone. This is based on the observation that successive $\boldsymbol{\alpha}^l$'s are very similar to each other. In fact they only differ by at most one vector component in Algorithm 6.

In detail, at each step l, we keep track of the instance examined and the label pairs that are selected. We refer to them by i^l and $(\hat{y}^l, \hat{\bar{y}}^l)$ respectively. We also keep track of the update that is performed. Define

$$\tau^l = \begin{cases} \triangle(\hat{y}^l, \hat{\bar{y}}^l) & \text{if } \langle \mathbf{w}(\boldsymbol{\alpha}^{(l-1)}) \, , \, \delta \mathbf{\Phi}_{i^l}(\hat{y}^l, \hat{\bar{y}}^l) \rangle \leq 0 \\ 0 & \text{otherwise} \end{cases} . \tag{5.14}$$

Therefore, the final averaged weight vector can be computed by

$$\begin{aligned} \mathbf{w} =& \frac{1}{L} \sum_{l=1}^{L} \mathbf{w}^l \\ =& \frac{1}{L} \sum_{l=1}^{L} \sum_{k=1}^{l} \tau^k \delta \mathbf{\Phi}_{i^k}(\hat{y}^k, \hat{\bar{y}}^k) \\ =& \frac{1}{L} \sum_{l=1}^{L} (L + 1 - l) \tau^l \delta \mathbf{\Phi}_{i^l}(\hat{y}^l, \hat{\bar{y}}^l) \, . \end{aligned} \tag{5.15}$$

With this method we reduce the memory cost from $\Omega(Lnq)$ (each $\boldsymbol{\alpha}$ takes $\Omega(nq)$) to $O(L)$.

5.3.2 Minover hierarchical Perceptron

The Minover hierarchical Perceptron takes advantage of the batch setting by using a greedy search for updates that is the most "beneficial". The algorithm is called Minover because it uses minimum-overlap rule [32]. The Minover Perceptron uses the instance that most violates the desired margin to update the separating hyperplane.

Algorithm 7 Minover hierarchical Perceptron algorithm

Inputs: training data $\{\mathbf{x}_i, Y_i\}_{i=1}^n$, maximal number of epochs κ.

```
1: Initialize $\alpha_{iy\bar{y}} = 0, \forall\, i,\ y \in Y_i,\ \bar{y} \in \bar{Y}_i$
2: for $num = 1$ to $\kappa n$ do
3:     $(\hat{i}, \hat{y}, \hat{\bar{y}}) = \text{argmin}_{i, y \in Y_i, \bar{y} \in \bar{Y}_i} \langle \mathbf{w}(\boldsymbol{\alpha}), \delta\boldsymbol{\Phi}_i(y, \bar{y}) \rangle$
4:     if $\langle \mathbf{w}(\boldsymbol{\alpha}), \delta\boldsymbol{\Phi}_{\hat{i}}(\hat{y}, \hat{\bar{y}}) \rangle > 0$ then
5:         terminate with a satisfactory solution
6:     else
7:         $\alpha_{\hat{i}\hat{y}\hat{\bar{y}}} \leftarrow \alpha_{\hat{i}\hat{y}\hat{\bar{y}}} + \triangle(\hat{y}, \hat{\bar{y}})$
8:     end if
9: end for
```

Using the minimum overlap selection rule effectively speeds up the convergence and yields sparser solutions.

The algorithm is given in Algorithm 7. Observe that selecting a worst triplet in step 3 requires calculating compatibility scores for all instances with all classes, and this step is performed for each iteration. Since the update at step 7 only affects a single dual variable, we suggest to use the same technique as in Section 4.3.3 to reduce computational cost. Essentially, we introduce auxiliary variable

$$T_{iy\bar{y}} \equiv \langle \mathbf{w}(\boldsymbol{\alpha}), \delta\boldsymbol{\Phi}_i(y, \bar{y}) \rangle \qquad (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i)\,. \tag{5.16}$$

The variables $T_{iy\bar{y}}$'s are initialized to 0 and incrementally updated whenever step 7 is performed.

$$\begin{aligned}
T_{iy\bar{y}}^{(\text{new})} &= \sum_{j=1}^{n} \sum_{r \in Y_j} \sum_{\bar{r} \in \bar{Y}_j} \alpha_{jr\bar{r}}^{(\text{new})} \langle \delta\boldsymbol{\Phi}_j(r, \bar{r}), \delta\boldsymbol{\Phi}_i(y, \bar{y}) \rangle \\
&= \sum_{j=1}^{n} \sum_{r \in Y_j} \sum_{\bar{r} \in \bar{Y}_j} \alpha_{jr\bar{r}}^{(\text{old})} \langle \delta\boldsymbol{\Phi}_j(r, \bar{r}), \delta\boldsymbol{\Phi}_i(y, \bar{y}) \rangle + \triangle(\hat{y}, \hat{\bar{y}}) \langle \delta\boldsymbol{\Phi}_{\hat{i}}(\hat{y}, \hat{\bar{y}}), \delta\boldsymbol{\Phi}_i(y, \bar{y}) \rangle \\
&= T_{iy\bar{y}}^{(\text{old})} + \triangle(\hat{y}, \hat{\bar{y}}) \langle \delta\boldsymbol{\Phi}_{\hat{i}}(\hat{y}, \hat{\bar{y}}), \delta\boldsymbol{\Phi}_i(y, \bar{y}) \rangle
\end{aligned} \tag{5.17}$$

5.4 Related work

Our hierarchical Perceptron is a big bang approach that learns all parameters simultaneously. In nature, it is different from the divide and conquer hierarchical approaches.

Perhaps what is the most similar to our hierarchical Perceptron is the Hieron algorithm proposed in [19]. Moreover, small modifications to the Hieron algorithm have been propsed in [38] to make Hieron more practical. The key difference between our hierarchical Perceptron and the Hieron algorithm lies in the update rule. The Hieron algorithm solves a quadratic program at each update step to determine the update quantity. The quadratic program aims at minimizing the distance between new and old weight vectors while constraining a desired margin to be achieved. This optimization problem has an analytic solution that facilitates learning. Our update rule, however, is quite simple and straightforward. Empirical results have shown that our hierarchical Perceptron algorithm performs competitively with the Hieron algorithm. In addition, the Hieron algorithm is proposed only for multiclass classification that uses a tree taxonomy. Our work provides a framework to work with multilabel classification and arbitrary DAG taxonomy. Our hierarchical Perceptron can also incorporate any class attributes and loss functions.

Chapter 6

Automatic hierarchy learning

In Chapter 4 and 5, we have presented two algorithms that can take advantage of given category hierarchies. A natural question that arises is whether the flat algorithms are the only choices when no category hierarchies are given. To use hierarchical algorithms, both ours and what have been proposed in other works, hierarchies over classes need to be provided. In this chapter, we propose an approach that automatically generates class hierarchies. The approach is based on agglomerative clustering and the hierarchy generated takes shape of a two-level tree. The generated hierarchy can then be used by existing hierarchical classification methods.

6.1 Introduction

Learning a class hierarchy from a set of instance and label set pairs is an unsupervised task. One may instantly link this task with the common task of data clustering, in particular hierarchical clustering in which a binary tree is constructed. The goal of clustering is to partition a data set into subsets, so that the data in each subset is compact. Data clustering is a common technique used in areas like machine learning, data mining, and bioinformatics.

Data clustering algorithms are generally divided into two paradigms: *partitional* or *hierarchical*. Partitional clustering algorithms such as K-means finds all clusters at once. The hierarchical clustering algorithms, on the other hand, iteratively produce new clusters based on previously determined clusters. The hierarchical algorithms therefore establish clustering solutions in the form of trees called *dendrograms*. Hierarchical clustering algorithms are thus of more interest to us, since they provide a view of data points in different levels of granularity, a characteristic desired in an automatic hierarchy learning method.

The hierarchical clustering algorithms usually fall into two types: *agglomerative* or *divisive*. Agglomerative clustering algorithms start by assigning each object to an individual cluster and then successively merge pairs of clusters until at the end there is only one cluster encompassing all objects. Divisive algorithms do the reverse by starting from a cluster of all objects and dividing established clusters into smaller pieces successively. There are also algorithms such as *constrained agglomerative clustering* that combine the agglomerative and divisive approaches [74]. The traditional view is that divisive algorithms are inferior to agglomerative algorithms with respect to cluster quality. Recent research, however, has demonstrated that divisive clustering algorithms can be as effective as agglomerative algorithms [74]. In our work, the agglomerative clustering algorithm is used since it is more readily available.

6.2 Hierarchy generation algorithm

To cluster a set of q objects, the hierarchical agglomerative clustering starts by assigning each object to a cluster of its own. At each step, a pair of clusters (which are often cluster pairs with lowest inter-cluster distance or highest inter-cluster similarity) is selected and merged into one cluster. Thus at each step, the number of clusters decreases by one. After $q-1$ steps, all objects are merged into one cluster. The results can be represented as a dendrogram in which a link indicates merging of two clusters [22].

To use the hierarchical agglomerative clustering algorithm to our purpose, several choices need to be made.

Objects to be clustered One option is to regard each class as an object to be clustered. The class can, for example, be characterized by its centroid. Another option is to view each instance as an object. So the starting point of hierarchical agglomerative clustering in our task is an intermediate clustering solution, in which each cluster corresponds to a category and the instances assigned to a cluster are those that are relevant with the corresponding category. We take the first option, since it significantly reduces the computational cost of agglomerative clustering, in terms of both space and time.

Distance metric between objects We dictate the merging policy to pick the pair of clusters that has the shortest distance among all pairs of current clusters. To determine the distance between two clusters of objects, one generally needs to first have a distance metric over object pairs. In our case, the object is centroid vector. We use the Euclidean distance, which is also used in [37] and [71]. The Euclidean distance between $c \in \mathbb{R}^d$ and $c' \in \mathbb{R}^d$ is defined as

$$\begin{aligned} d(c, c') &= \sqrt{(c - c')^T (c - c')} \\ &= \sqrt{\langle c, c \rangle + \langle c', c' \rangle - 2\langle c, c' \rangle} \,. \end{aligned} \tag{6.1}$$

From Equation (6.1), one observes that the Euclidean distance only depends on the inner products of object vectors. Thus the distance metric can be kernelized and a clustering in an implicit space of higher dimension can be computed efficiently.

Distance metric between clusters There are several techniques to determine the distance between clusters of objects, based on the distance between objects. The most common ones are *single linkage*, *complete linkage*, and *average linkage*. The single linkage measure determines the distance between two clusters to be the minimum distance between all element pairs, with one element from one cluster and the other from the other cluster. In contrary, the complete linkage measure takes the maximum distance. The average linkage is a compromise between the single linkage and the complete linkage. It takes the average of distances between corresponding element pairs. For two cluster $\mathcal{C}$ and $\mathcal{C}'$ the average link distance is defined as

$$d(\mathcal{C}, \mathcal{C}') = \frac{1}{|\mathcal{C}||\mathcal{C}'|} \sum_{c \in \mathcal{C}} \sum_{c' \in \mathcal{C}'} d(c, c') \,. \tag{6.2}$$

Our algorithm used average linkage since it is generally the most robust among the three [22].

With the decisions made, we arrive at the hierarchical agglomerative clustering procedure as described in Algorithm 8. The link at level t is denoted by ln_t and the merging distance of a link ln_t is denoted by $m(ln_t)$, which is the distance between the two clusters that ln_t merges.

Algorithm 8 Automatic Hierarchy Learning.

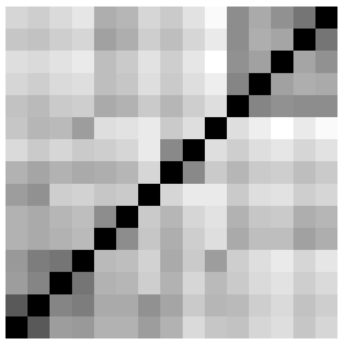
```
Inputs: $\mathcal{D} = \{\mathbf{x}_i, Y_i\}_{i=1}^n$, in which $\mathbf{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$ and $Y_i \subseteq \mathcal{Y} = \{1, \cdots, q\}$
1:  Compute class centroids $\mathbf{c}_k \leftarrow \frac{\sum_{i:k\in Y_i} \mathbf{x}_i}{\sum_{i:k\in Y_i} 1}$ for $k = 1, \cdots, q$
2:  {Hierarchical agglomerative clustering}
3:  Initialize $q$ clusters such that $\mathcal{C}_k \leftarrow \{\mathbf{c}_k\}$ for $k = 1, \cdots, q$.
4:  $B \leftarrow \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_q\}$
5:  for $t = 1$ to $q - 1$ do
6:     $(\hat{\mathcal{C}}, \hat{\mathcal{C}}') \leftarrow \text{argmin}_{\mathcal{C},\mathcal{C}':\mathcal{C}\in B,\mathcal{C}'\in B,\mathcal{C}\neq\mathcal{C}'} \; d(\mathcal{C}, \mathcal{C}')$
7:     merge $\hat{\mathcal{C}}$ and $\hat{\mathcal{C}}'$ into a new cluster $\mathcal{C}_{q+t}$.
8:     $m(ln_t) \leftarrow d(\hat{\mathcal{C}}, \hat{\mathcal{C}}')$
9:     $B \leftarrow B \setminus \{\hat{\mathcal{C}}, \hat{\mathcal{C}}'\}$
10:    $B \leftarrow B \cup \{\mathcal{C}_{q+t}\}$
11: end for
12: {Calculate inconsistency coefficients}
13: for $t = 1$ to $q - 1$ do
14:    $S \leftarrow \{m(ln_t)\}$
15:    for each cluster $\mathcal{C}$ that link $ln_t$ merges do
16:       if $\mathcal{C}$ is not a leaf cluster then
17:          $ln \leftarrow$ the link that $\mathcal{C}$ corresponds to
18:          $S \leftarrow S \cup \{m(ln)\}$
19:       end if
20:       $\mu(S) \leftarrow$ sample mean of $S$
21:       $\sigma(S) \leftarrow$ sample standard deviation of $S$
22:       $IC(ln_t) = \frac{m(ln_t) - \mu(S)}{\sigma(S)}$
23:    end for
24: end for
25: {Form a hierarchy}
26: $m \leftarrow \frac{\sum_{t=1:IC(ln_t)\neq 0}^{q-1} IC(ln_t)}{\sum_{t=1:IC(ln_t)\neq 0}^{q-1} 1}$
27: Determine a flat clustering by cutting off links with inconsistency coefficient below
    $m$
28: Build a hierarchy with the clustering outcome as the first level and individual
    classes as the second level.
```

The dendrogram output by the hierarchical agglomerative clustering is a binary tree that is generally very tall. It is complicated yet lacks structure. Ideally we prefer a balanced tree. We propose to use *inconsistency coefficient* [28, 27] to cut off some links and derive a two-level hierarchy. The inconsistency coefficient, as computed in Algorithm 8, characterizes a link by comparing its distance to those of neighboring links. If the merge is consistent, then the link will likely have a merging distance that is close to those of the two clusters that the link fuses, and thus the link will have a low inconsistency coefficient. If the inconsistency coefficient is high, then it indicates that the merging distance varies considerably from those of its constituent clusters, and a natural separation of clusters is likely to occur at this point.

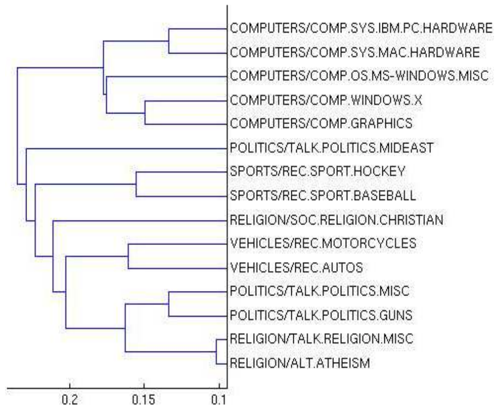
We use the mean of non-zero inconsistency coefficients in a dendrogram as the threshold to cut links off the dendrogram (or one may imagine removing the links from the dendrogram). Any link whose inconsistency coefficient is above the threshold will be cut off the dendrogram. If a link is cut, any links that build on top of it is also cut. A concrete example will be given below in Figure 6.1. After this stage, the dendrogram is partitioned into a set of continuous dendrogram pieces. All classes that are still connected form a flat cluster. The inconsistency coefficient is thus used to determine the number of clusters. Data points within each cluster are expected to be consistent.

We can hence build a two-level hierarchy with the classes as leaf nodes and the flat clusters produced after the dendrogram cut-off as the first level nodes. An edge in the hierarchy indicates that a class belongs to a flat cluster. Indeed our algorithm is a special case and one can generalize it to employ other flat clustering algorithms to determine a two-level hierarchy.

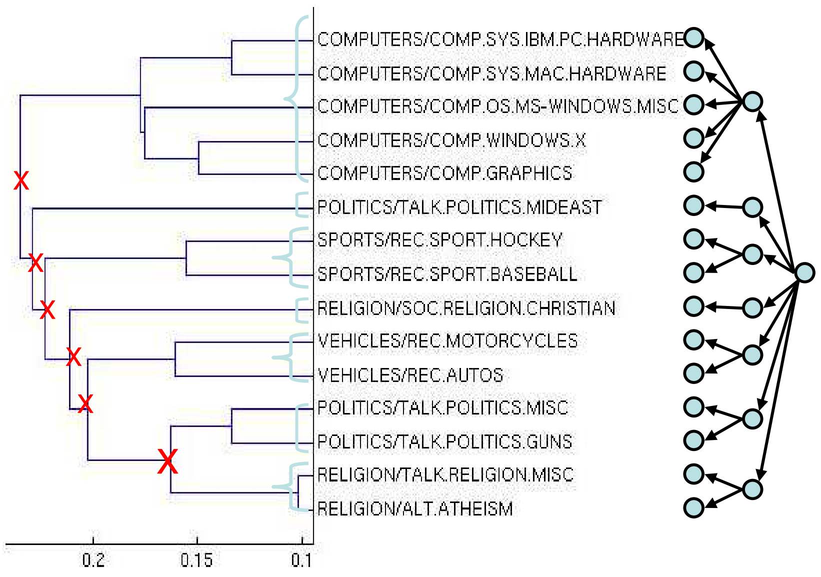
Figure 6.1 depicts a hierarchy produced by Algorithm 8 from a 90% subset of the Newsgroup data. It is a meaningful structure and is not far from the actual hierarchy in Figure 7.2. Actually the automatic hierarchy differs from the actual one only in that it replaces the `RELIGION` node with two nodes and the `POLITICS` node with two. The cluster separation (corresponding to rectangles in Figure 6.1(b)) appear consistent with the distance matrix in Figure 6.1(a). To use existing hierarchical classification algorithms, an unsupervised learning stage can first be performed to establish a class hierarchy and then hierarchical classification can be carried out.



(a)



(b)


COMPUTERS/COMP.SYS.IBM.PC.HARDWARE
COMPUTERS/COMP.SYS.MAC.HARDWARE
COMPUTERS/COMP.OS.MS-WINDOWS.MISC
COMPUTERS/COMP.WINDOWS.X
COMPUTERS/COMP.GRAPHICS
POLITICS/TALK.POLITICS.MIDEAST
SPORTS/REC.SPORT.HOCKEY
SPORTS/REC.SPORT.BASEBALL
RELIGION/SOC.RELIGION.CHRISTIAN
VEHICLES/REC.MOTORCYCLES
VEHICLES/REC.AUTOS
POLITICS/TALK.POLITICS.MISC
POLITICS/TALK.POLITICS.GUNS
RELIGION/TALK.RELIGION.MISC
RELIGION/ALT.ATHEISM
0.2
0.15
0.1

(c)

Figure 6.1: The distance matrix, along with the dendrogram and hierarchy, generated by Algorithm 8 on a 90% subset of Newsgroup corpus. Figure (a) illustrates the distance matrix in which the darker the color is, the shorter the distance is between two corresponding classes. The classes in Figure (a), from left to right, or from bottom to top, are the same sequence of classes from bottom to up in Figure (b). Figure (b) depicts the dendrogram that is generated. The horizontal axis denotes the merging distance. Figure (c) illustrates the generated hierarchy. The crosses mark the links that are cut, among which the single biggest cross marks the link whose inconsistency coefficient is above the threshold. The 7 flat clusters are indicated by braces and brackets. The generated hierarchy has 7 nodes in the first level and 15 nodes in the second level.

6.3 Related work

Applying data clustering techniques to pattern classification (mainly document classification) has been previously studied. Some works are concerned with clustering words so that a considerably more compact feature representation is derived, with only a minor loss in classification accuracy. For instance, [1] employs distributional clustering, a probabilistic soft clustering method, to effectively cluster words into groups. This way the number of features is aggressively reduced while the decrease in accuracy is only a few percentage points. This methods enables scalability of classification models to larger data sets. Similarly, [57] applies the information bottleneck method to cluster words in a way that preserves information about classes. By using the word clusters as features in place of words themselves, the dimension of features is significantly reduced. [57] reports that while the training data is small, the accuracy is actually improved. In contrast, some other works are concerned with clustering documents to boost classification performance. For instance, [3] uses the probabilistic latent semantic indexing to estimate cluster-dependent document probabilities. By incorporating these probabilities as additional features and using Boosting algorithm to select the most relevant features, classification performance is improved. A similar approach is taken in [33]. A hard clustering algorithm is applied to both labeled and unlabeled data and cluster assignments are used to augment feature representation. Our task of automatically generating class hierarchies based on clustering techniques, however, has been rarely studied. The work on existing hierarchical classification algorithms almost exclusively uses hierarchies that are provided with corpora.

There have been many works on organizing a collection of documents into a hierarchy [2, 71]. Different from our work, the goal is often to visualize data and facilitate navigation. A major emphasis on these works is labeling the clusters. Our work, on the other hand, focuses on organizing classes into a balanced hierarchy that summarizes class proximity. Our goal is to improve the classification performance with the automatically learned hierarchy rather than the hierarchy itself. Examples of producing hierarchies over documents include [2], in which a dendrogram is first built through a divisive hierarchical clustering and then the dendrogram is converted to a tree with a specified branching factor. To split a cluster, a linear discriminant function is used that calculates the projection of centered documents to the direction of maximal variance. This discriminant function is also employed to find significant features to describe clusters. To establish a tree with the specified branching factor, a sequence of splits is fused into one split step that divides one cluster into multiple child clusters. A different approach is taken in [71]. First flat clusters of documents are constructed through the self organizing map method. The hierarchy is generated by successively finding clusters with current largest supporting class similarity and then including their neighboring clusters as child clusters. So varying values of branching factors naturally appear in the established hierarchy.

[37] also presents a method to generate a class hierarchy based on the hierarchical agglomerative clustering algorithm. Our work differs from [37] mainly in the methods that transfer a dendrogram to a shorter hierarchy with branching factors that are not exclusively 2. We use inconsistency coefficients to cut off links while [37] uses a heuristic idea based on sudden increase of merging distance. Experiments are conducted in [37] with the divide-and-conquer hierarchical SVM while our experiments (see Chapter 7) are with our hierarchical Perceptron and the Hieron algorithm (both are big bang methods). A linear discriminant projection step is proposed in [37] that appears beneficial to the empirical success of the method in [37]. In this step, the Fisher discriminant analysis is performed so that all documents are transformed into a space with fewer dimensions than the original space. The distance metric used for agglomerative clustering is then computed in this new space. In our experiments, adding this step yields neither improvements in hierarchy quality nor better classification performance.

Chapter 7

Experiments

In this chapter we describe the experiments that we have performed to verify the empirical advantages of the hierarchical methods proposed in previous chapters. To remind the reader, we presented the hierarchical SVM in Chapter 4 with its primal formulation in Equation (4.10) and dual formulation in Equation (4.15). Several variants of the hierarchical Perceptron are described in Chapter 5. The automatic hierarchy learning algorithm is proposed in Chapter 6 with the algorithm summarized in Algorithm 8. In addition to comparing these approaches with their flat counterparts, we have also evaluated them against two other hierarchical approaches. We start the chapter with a description of experimental methodology and setups. Then we present experimental results. Finally we summarize our findings.

7.1 Data sets

The five corpora described below are used in our experiments. The synthetic data are generated artificially and it employs a perfect tree taxonomy. The WIPO-alpha collection, Newsgroup set, and OHSUMED corpus are all composed of text documents, but of different natures: patent documents, newsgroup posts, and medical documents respectively. The first two data sets use tree taxonomies with classes denoted by leaf nodes all at the same level. The OHSUMED corpus, however, uses a complicated structure in which branches have various depths and classes can be represented by partial paths. ENZYME is a biological data set with protein sequences as instances.

The taxonomy is a balanced tree and the categories for classification only appear at the leaf level.

7.1.1 Synthetic data

A first set of experiments has been conducted on a simple synthetic data set. We start with a tree structure and a given number of features. Random weight vectors are generated for each node, using the same multivariate normal distribution for nodes at the same depth. The covariance matrices of the normal distributions are diagonal and the variances decrease with depth. The weight vector for each category equals the sum of the weight vectors along the path from the root to the category. Data points are randomly generated according to a multivariate normal. A data point $\mathbf{x}$ is assigned to those categories y for which $F(\mathbf{x}, y; \mathbf{w}) > 0$. The data point is only accepted if the number of relevant categories is less than $\sqrt{q}$. The way the artificial data is generated assures that weight vectors of nearby categories are closer than that of distant categories, simulating a property that we expect to be relevant for real-world taxonomies.

7.1.2 WIPO-alpha set

The World Intellectual Property Organization (WIPO) published the WIPO-alpha collection in 2002 [69]. The patent documents in the collection are classified according to a standard taxonomy known as International Patent Classification (IPC) [68]. IPC categories are organized in a four-level hierarchy, i.e. sections, classes, subclasses and groups (main groups and subgroups). Part of section D is illustrated in Figure 7.1 for concreteness. Another example showing part of section H is given in Figure 1.1.

The categories in our experiments refer to main groups that are all leaves at the same depth in the hierarchy. Each document is labeled with one primary category as well as any number of secondary categories. Predicting the primary categories is therefore a multiclass categorization problem while predicting both types of categories forms a multi-label corpus. We have performed independent experiments on taxonomies under the 8 top-level sections. Document parsing was performed with the Lemur toolkit [7]. Stop words were removed. Stemming was not performed. Word

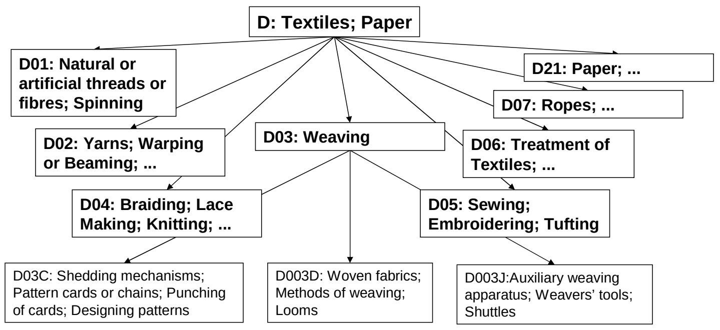

D: Textiles; Paper
D01: Natural or artificial threads or fibres; Spinning
D21: Paper; ...
D07: Ropes; ...
D02: Yarns; Warping or Beaming; ...
D03: Weaving
D06: Treatment of Textiles; ...
D04: Braiding; Lace Making; Knitting; ...
D05: Sewing; Embroidering; Tufting
D03C: Shedding mechanisms; Pattern cards or chains; Punching of cards; Designing patterns
D003D: Woven fabrics; Methods of weaving; Looms
D003J:Auxiliary weaving apparatus; Weavers' tools; Shuttles

Figure 7.1: Part of the IPC classification hierarchy rooted at section D which contains a total of 160 main groups. Only classes and subclasses for D03 are shown.

section	Number of classes	Number of instances	Number of features	Number of nodes
A	694	10,962	73,188	781
B	1,172	14,690	51,470	1,320
C	852	16,245	178,202	9,26
D	160	1,710	18,077	188
E	230	3,027	17,802	264
F	675	6,685	26,914	758
G	470	10,302	55,663	537
H	403	11,629	46,210	455

Table 7.1: Summary of the WIPO-alpha multiclass corpus. Properties of sub-collections under each top level node (called section) are given.

counts from title and claim fields are used as document features.

Table 7.1 and 7.2 summarize the properties of the sub-collections under each top-level section, for the multiclass and the multilabel WIPO-alpha corpus respectively. A document is counted for classification under a section if the document has a relevant category located in the subtree rooted at the node representing this section.

To investigate the effect of the training size, we have furthermore subsampled the data. In particular, three samples (or all available documents, if the category possesses less than three training documents) are randomly picked for each category. If an instance is assigned multiple classes, then a random correct class is used for sampling purpose. We refer to this as subsampled WIPO-alpha corpus. Table 7.3 summarizes the subsampled multiclass and multilabel corpora.

section	No. of classes	No. of instances	No. of classes per instance	No. of features	No. of nodes
A	846	15662	1.44	147,757	948
B	1514	17626	1.48	65,975	1,723
C	1152	14841	2.11	204,486	1,264
D	237	2194	1.53	24,056	286
E	267	3586	1.34	21,577	306
F	862	8011	1.45	33,605	980
G	565	12944	1.29	81,862	649
H	462	13178	1.35	55,188	520

Table 7.2: Summary of the WIPO-alpha multilabel corpus. Note the data subset under a given section is larger than that in the multiclass context since secondary classes are now also taken into account to determine if an instance is included in the subset. The number of classes per instance is macro-averaged number across corresponding instances.

section	No. of instances in multiclass corpus	No. of instances in multilabel corpus
A, sampled	1,781	1,935
B, sampled	3,033	3,355
C, sampled	2,212	2,491
D, sampled	391	464
E, sampled	600	634
F, sampled	1,729	1,936
G, sampled	1,228	1,340
H, sampled	1,084	1,149

Table 7.3: Summary of the subsampled WIPO-alpha corpus.

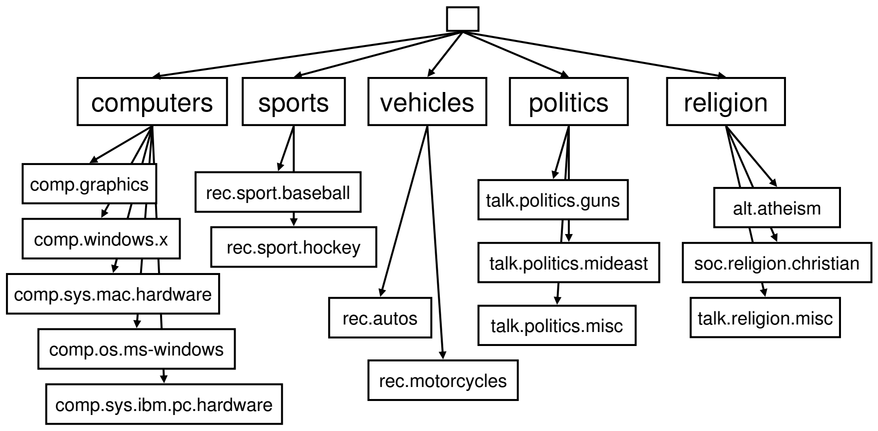


Figure 7.2: Class taxonomy of Newsgroup corpus.

7.1.3 Newsgroup

The Newsgroup collection is composed of Usenet articles drawn from 20 discussion groups [30], with 1,000 articles per group. We construct a two level hierarchy from 15 groups as suggested by [41]. Compared to IPC, this is a very small category tree with 5 top level nodes and 15 terminal nodes. The 5 top nodes together with their children are illustrated in Figure 7.2. We used the Lemur toolkit [7] for indexing and removing stop words. No stemming of words was performed. After removing words that occur only once, the data set consists of 2.37 million word occurrences with a vocabulary size of 63,315.

7.1.4 OHSUMED

The OHSUMED corpus is the 1987 portion of the OHSUMED collection [25] that has also been used for the TREC9 filtering track. The collection consists of short documents with titles and abstracts from MEDLINE, a national bibliographic database covering medicine and related fields. Each document has been manually indexed with MeSH (Medical Subject Headings) terms [43]. This corpus involves 4,904 MeSH terms. After removing stop words and words that occurred in fewer than 10 documents, we are left with 54,708 documents with a vocabulary of 19,066 words.

The MeSH terms are arranged from the most general to most specific ones in an 11-level hierarchy. The articles are indexed with the most specific MeSH terms that

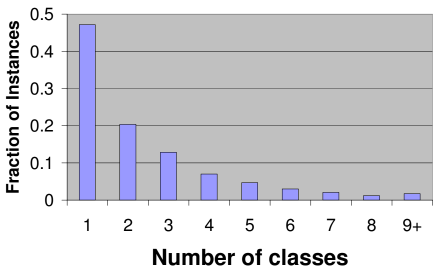
subtree	Number of classes	Number of instances	Number of classes per instance
A	719	15617	2.34
B	208	4797	1.68
C	2523	30237	3.52
D	2471	24769	4.38
E	964	23939	2.29
F	266	5207	1.94
G	1363	20743	1.87
H	154	6248	1.35
I	194	4263	1.74
J	67	1485	1.25
K	24	618	1.48
L	80	1996	1.37
M	63	1752	1.47
N	488	8627	2.37
Z	35	735	1.12

Table 7.4: Summary of the OHSUMED corpus.

are available. In the experiments, we used the MeSH tree of 2003, which organizes $22,568$ MeSH headings. The MeSH headings used in the OHSUMED corpus covers only a subset of all available headings. Those MeSH terms that do not appear in the OHSUMED corpus are simply ignored. The OHSUMED articles are labeled with the MeSH terms. However, a term can correspond to multiple positions in the hierarchy. Our strategy is to regard the tree locations as categories and include all the locations that match a term as labels for a document with that term.

This set differs from WIPO-alpha and Newsgroup set not only in terms of the document nature but also of the taxonomy complexity. The branches of the tree have various depths, the number of child nodes of an internal node can be very different, and the categories can virtually be at any level of the hierarchy.

Due to the large size of this data set, we have divided the hierarchy into sub-hierarchy, each being a subtree rooted at a top-level node. There are 15 top-level nodes representing concepts such as `anatomic terms` for node A, `diseases` for node C, and `drugs and chemicals` for node D. Table 7.4 summarizes the properties of the subtrees and the data subset corresponding to these subtrees, which are all instances



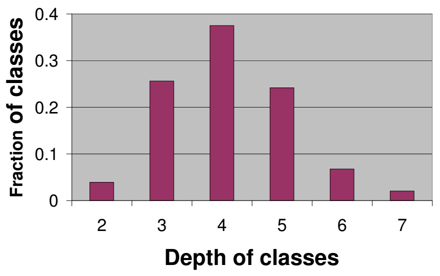


Figure 7.3: Distribution of the number of positive classes per document (left), and distribution of the depths of classes (right) for N subtree of OHSUMED corpus.

with at least one relevant label located in a specified subtree. We observe that the number of classes per document varies from 1.12 to 4.38. To take a closer look, we pick subtree N and depict in Figure 7.3 the distribution of the number of classes per document and the distribution of the depths of classes in this subtree. We use the N subtree as an example since it is in the middle range among all subtrees in terms of both the total number of categories and the average number of categories per document. Figure 7.3 shows more than half of the documents in the sub-collection have two or more relevant classes. The number of positive classes for instances can vary considerably. The distribution of depths of classes shows classes are located at a variety of levels in the sub-taxonomy. Depth 4, which is the middle level of the sub-taxonomy, accommodates the greatest number of classes.

7.1.5 ENZYME

The ENZYME corpus was compiled and processed by Rousu et. al. [50]. The instances are protein sequences and the categories are the reactions that the proteins catalyze. The category taxonomy is a subset of the Enzyme Classification system [46]. The taxonomy consists of 236 nodes (including two nodes that are ancestors of every class) organized in a three-level hierarchy tree. The classes are the 172 leaves of the hierarchy. The data set was divided into 7700 protein sequences for training and 1755 sequences for testing [50].

7.2 Experimental setup

In synthetic data, WIPO-alpha set, Newsgroup, and ENZYME corpus, the taxonomies are trees with categories at the same depth of the tree. Taxonomy-derived attributes are employed in our hierarchical approaches. For comparison purpose, $t_v(y)$ in Eq. (3.7) is set to 0 for the root node (that is an ancestor of every class) and otherwise $1/\sqrt{depth}$ where $depth$ is the depth of the tree and is defined as $depth \equiv \max_y |anc(y)| - 1$. This scaling is used for mathematical convenience, since it implies that $\langle \mathbf{\Lambda}(y), \mathbf{\Lambda}(y) \rangle = 1$ and we therefore get that

$$\langle \Phi(\mathbf{x}, y), \Phi(\mathbf{x}, y) \rangle = \langle \mathbf{\Lambda}(y), \mathbf{\Lambda}(y) \rangle \langle \mathbf{x}, \mathbf{x} \rangle = \langle \mathbf{x}, \mathbf{x} \rangle . \tag{7.1}$$

This is also to facilitate fair comparison between the flat and hierarchical SVM classification since, in the former case, $\mathbf{\Phi}(\mathbf{x}, y)$ can be viewed as in Equation (3.4) and hence $\langle \Phi(\mathbf{x}, y), \Phi(\mathbf{x}, y) \rangle$ equals $\langle \mathbf{x}, \mathbf{x} \rangle$ as well. The category taxonomy of OHSUMED collection is a quite unbalanced tree. To deal with it, we let $t_v(y) = 0$ if y is the root and otherwise $t_v(y) = 1/\sqrt{depth(y)}$, where $depth(y) = |anc(y)| - 1$ is the depth of class y. Thus $\langle \mathbf{\Lambda}(y), \mathbf{\Lambda}(y) \rangle = 1$ still holds for all classes.

The particular hierarchical loss function that is used in the experiments on all data sets except OHSUMED is $\triangle_h(y, \hat{y}) = \frac{1}{2}|\mathrm{anc}(y) \ominus \mathrm{anc}(\hat{y})|$, which equals half the loss function defined in Equation (3.13). We use the $\frac{1}{2}$ coefficient due to historical reason. Note in case of a tree structure $\triangle_h$ is half the (undirected) distance from one class to another. On the OHSUMED set, the depth of categories can vary considerably. This phenomenon could be because some categories represent broader concepts while some represent narrower ones, or because some sub-structures are articulated in more detail and thus occupy more levels than others. For this reason, we use a normalized loss function for hierarchical learning on OHSUMED set.

$$\triangle_{\bar{h}}(y, \hat{y}) = \frac{1}{2} \left(\frac{|\mathrm{anc}(y) \setminus \mathrm{anc}(\hat{y})|}{|\mathrm{anc}(y)| - 1} + \frac{|\mathrm{anc}(\hat{y}) \setminus \mathrm{anc}(y)|}{|\mathrm{anc}(\hat{y})| - 1} \right) \tag{7.2}$$

This loss function can be understood as half of the weighted distance between two categories. The edges on the path from y ($\hat{y}$) to the deepest common ancestor of y and $\hat{y}$ are weighted by 1 over the depth of y ($\hat{y}$). By summing these edge weights up, the normalized hierarchical loss is obtained. Note in Equation (7.2) $|anc(y)|-1$ equals the depth of y. If all classes are at the same depth, then the normalized hierarchical loss

$\triangle_{\bar{h}}$ reduces to $\triangle_h$. The normalized hierarchical loss is used in hierarchical learning on OHSUMED set. In the evaluation step, however, the hierarchical loss $\triangle_h$ is still used since it is easier to visualize.

In all experiments, except those on ENZYME corpus, a linear kernel is used, since it has computational advantages during optimization while achieving competitive performance compared to other more complicated kernels such as polynomial kernel and RBF kernel in the context of text categorization [29]. A length-4 subsequence kernel was used on ENZYME corpus [51].

Moreover, document vectors are normalized to be of unit length, $\|\mathbf{x}_i\|_2 = 1$, as a preprocessing step. The test performance was often computed using cross-validation and macro-averaging. The tolerance parameter ϵ in Algorithm 2 is set to 0.01 or 0.1. [14] points out that the choice of $\epsilon = 0.1$ achieves a good tradeoff between running time and generalization performance. Following the heuristics utilized in SVMlight [29], C is set to 1 if not otherwise specified (remember that input vectors are normalized to unit length). Our experiments show this is a good choice.

7.3 Evaluation measures

The measures we used include *one-accuracy*, *average precision*, *ranking loss*, *maximal loss*, *top loss* and *parent one-accuracy*. The first three are standard metrics for (flat) multilabel classification problem [54, 24]. In a hierarchical system, it is often also desirable to take into account the label relations encoded in the taxonomy. To accommodate this we also develop the other three metrics.

The output of our learning algorithms is a weight vector $\mathbf{w}$. The corresponding compatibility function is $F(\mathbf{x}, y; \mathbf{w}) = \langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x}, y) \rangle$. Let f be the classifier that always predicts the highest ranked label, i.e. $f(\mathbf{x}; \mathbf{w}) = \operatorname{argmax}_y F(\mathbf{x}, y; \mathbf{w})$. If documents are only allowed a single label, the classification system will naturally predict the one in which it has the greatest confidence. The test is performed on $S = \{\mathbf{x}_i, Y_i\}_{i=1}^n$.

One-accuracy (acc) measures the empirical probability of the top-ranked label being relevant to the document. Let [.] be 1 if the predicate inside holds and 0 if not.

$$\operatorname{acc}(f, S) = \frac{1}{n} \sum_{i=1}^{n} [f(\mathbf{x}_i) \in Y_i] \,. \tag{7.3}$$

Average precision (prec) measures the quality of label rankings. Precision is calculated at each position where a positive label occurred, as if all the labels ranked higher than it, including itself, are predicted as relevant. These precision values are then averaged to obtain average precision. Formally,

$$\text{prec}(F, S) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|Y_i|}\sum_{y\in Y_i}\frac{|\{r : r \in Y_i, F(\mathbf{x}_i, r) \geq F(\mathbf{x}_i, y)\}|}{|\{r : r \in \mathcal{Y}, F(\mathbf{x}_i, r) \geq F(\mathbf{x}_i, y)\}|}\,. \tag{7.4}$$

Ranking loss is defined in [54] as

$$\text{rloss}(F, S) = \frac{1}{n}\sum_{i=1}^{n}\frac{|\{(y \in Y_i, \bar{y} \in \bar{Y}_i) : F(\mathbf{x}_i, y) \leq F(\mathbf{x}_i, \bar{y})\}|}{|Y_i||\bar{Y}_i|}\,. \tag{7.5}$$

Ranking loss measures the average fraction of positive label and negative label pairs that are misordered.

Maximal loss, denoted by $\triangle^x$, and *top loss*, denoted by $\triangle^t$ were introduced in Chapter 4.2.2. Maximal loss takes the maximum from the losses incurred by all misranked positive and negative labels. Top loss is the loss suffered from the top-ranked label. Both of them are upper bounded by the averaged slack variable value that is part of the optimization objective in the dual formulation of the hierarchical SVM. In the experiments these two metrics are measured by the hierarchical loss function $\triangle_h$ defined in Section 7.2. In the context of multiclass classification only one label is needed from the classifier. The users are generally more concerned with the loss caused by this single label than by any other misordered labels. In the experiments, therefore, we generally evaluate top loss for the multiclass cases and maximal loss for the multilabel cases.

A misclassification can be weighed differently, depending on how far away the predicted label is from the correct labels. A misclassification to a sibling category is presumably less problematic than a misclassification to a very different category. For this reason we also evaluate *parent one-accuracy* (pacc), which measures the one-accuracy at the category's parent node level. Let $\mathcal{P}(y)$ denote the immediate parent node of y. Then

$$\text{pacc}(f, S) = \frac{1}{n}\sum_{i=1}^{n}[\exists y \in Y_i \text{ s.t. } \mathcal{P}(f(\mathbf{x}_i)) = \mathcal{P}(y)]\,. \tag{7.6}$$

fan-out	dep-th	acc (%)		prec (%)		$\triangle^x$-loss		rloss (%)		pacc (%)	
		F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM
3	3	88.5	**91.4**	86.4	**89.2**	1.63	**1.39**	3.30	**2.24**	91.7	**94.3**
6	2	**93.7**	93.4	87.0	**88.8**	1.32	**1.22**	3.09	**2.31**	95.3	**96.6**

Table 7.5: 10-fold cross-validation results on synthetic data. "F-SVM" is abbreviation of the flat SVM while "H-SVM" is abbreviation of the hierarchical SVM. "Fan-out" refers to the fan out factor of each interior concept, "depth" refers to the depth of the tree, "acc" refers to the accuracy, "prec" refers to the average precision, "$\triangle^x$-loss" refers to the taxonomy-based maximal loss, "rloss" refers to the ranking loss, and "pacc" refers to the parent accuracy. There are 20 features and 200 examples. $C = 10,000$. Bold face is used to mark better performance.

Assume two algorithms have similar accuracy. If a misclassification occurs, the algorithm with higher parent accuracy is then more likely to assign an instance to the true category's siblings, than to assign it to a class that is farther away from the correct class. An algorithm with a higher parent accuracy is thus favored, for instance, when used as an automatic categorization tool to assist human experts.

7.4 Results with hierarchical SVM

This section summarizes our experiments of comparing the flat SVM, the hierarchical SVM, and a divide-and-conquer SVM that exploits a hierarchy. We have also compared the latter two methods when a random hierarchy is used.

7.4.1 Hierarchical SVM versus flat SVM

In this section we compare the hierarchical SVM with the flat SVM on all five corpora.

Synthetic data

Table 7.5 summarizes the performance of 10-fold cross-validation. It is close to hard-margin separation since C is set to 10^4 here. We observe that the hierarchical SVM consistently excels on all runs with respect to almost all considered measures except one-accuracy. This also holds for various other C's that we used. The performance

section	acc (%)		prec (%)		$\triangle^t$-loss		pacc (%)	
	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM
A	42.3	**42.9**	51.7	**53.2**	1.24	**1.15**	61.5	**65.0**
B	33.2	**33.8**	41.5	**43.1**	1.54	**1.41**	57.3	**62.2**
C	**35.5**	35.1	**44.8**	44.6	1.32	**1.23**	61.5	**65.6**
D	41.8	**42.8**	52.3	**54.4**	1.20	**1.08**	65.4	**69.1**
E	**34.7**	34.3	44.8	**46.3**	1.38	**1.30**	62.7	**64.2**
F	31.2	**32.4**	40.6	**42.9**	1.47	**1.33**	57.6	**63.3**
G	41.0	**41.2**	50.3	**51.1**	1.32	**1.26**	60.6	**63.0**
H	43.0	**43.1**	54.2	**55.2**	1.12	**1.07**	63.3	**66.2**

Table 7.6: Performance comparison of flat SVM (F-SVM) and hierarchical SVM (H-SVM) on multi-class WIPO-alpha subtrees rooted at various section codes. Notations are the same as in Table 7.5. $\epsilon = 0.1$ for runs on section A, B, C, F, G, and H and $\epsilon = 0.01$ for experiments on sections D and E which are smaller.

gains are not huge but the performance advantage of using hierarchical SVM is clear and steady.

Multiclass WIPO-alpha set

Table 7.6 compares the performance of the flat SVM and the hierarchical SVM with respect to all 8 sections. When dealing with a specific section, only documents with their main category in the section in question are taken into account. Three-fold cross-validation has been performed for all runs. We observe that hierarchical SVM outperforms the flat SVM in terms of $\triangle^t$-loss in all cases. This can be attributed to the fact that it explicitly optimizes an upper bound on the $\triangle^t$-loss of a training set as well as to the specific hierarchical form of the discriminant function. Moreover, hierarchical SVM in most cases also produces higher accuracy, precision, and parent accuracy.

In Table 7.6, the relative improvements on the taxonomy-derived loss $\triangle^t$ are generally much higher than those on the one-accuracy measure. Figure 7.4 gives an example of misclassification counts related to each hierarchical loss value. In addition to making larger amount of correct predications, the hierarchical SVM makes more low-cost misclassifications and less high-cost misclassification. This way the overall hierarchical loss is effectively reduced.

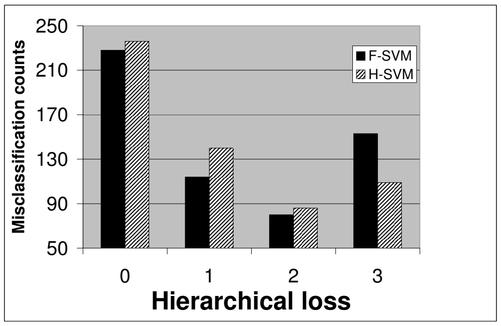


Figure 7.4: Histogram of the hierarchical loss on D section of the multiclass WIPO-alpha corpus. The solid column corresponds to the flat SVM while the shaded column corresponds to the hierarchical SVM. The height of a column indicates the number of misclassifications that incur a specific hierarchical loss. When loss equals 0, the columns correspond to the number of correct classification. The numbers are from one run of three-fold cross-validation.

As mentioned in Section 7.2, the trade-off coefficient C in the hierarchical SVM is set to 1 by default. Our experiments show this is a good choice. Figure 7.5 depicts the one-accuracy and top loss of the investigated algorithms for varying values of C. It appears that $C = 1$ leads to a decent performance and that the hierarchical SVM always achieves a lower top loss.

Figure 7.6 is an example of how the optimization process of the hierarchical SVM evolves over time as more and more variables are selected. For that purpose we define the active dual variable ratio as

$$\frac{\left|\{\alpha_{iy\bar{y}} | y \in Y_i \wedge \bar{y} \in \bar{Y}_i \wedge \alpha_{iy\bar{y}} > 0\}\right|}{\sum_i |Y_i||\bar{Y}_i|}. \tag{7.7}$$

We observe that in the beginning iterations, the variable selection strategy almost always adds a new dual variable in each iteration. When the active set reaches a reasonable size, the growth of the active set sizes slows down and more efforts are focused on optimizing the variables that are already in the set. In all our experiments on WIPO-alpha, the learned solutions were very sparse, usually with an active ratio in the range of $[0.005, 0.1]$.

Our method of adding one or zero dual variable each time into optimization also leads to sparser solutions and faster convergence, when compared to strategies such

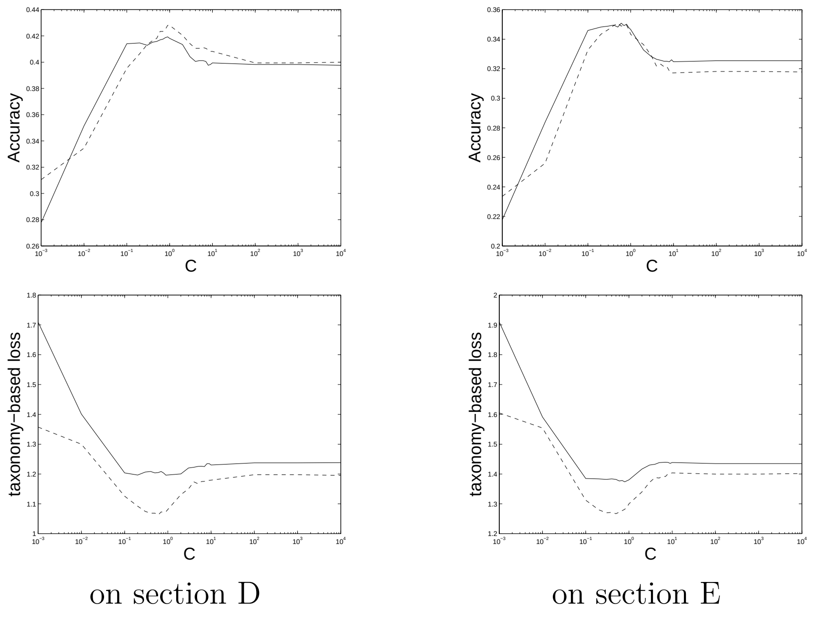


Figure 7.5: Performance on two sections of the multiclass WIPO-alpha with varying trade-off parameter C. Solid lines correspond to the flat SVM and dashed lines correspond to the hierarchical SVM. Here the "taxonomy-based loss" refers to the top loss.

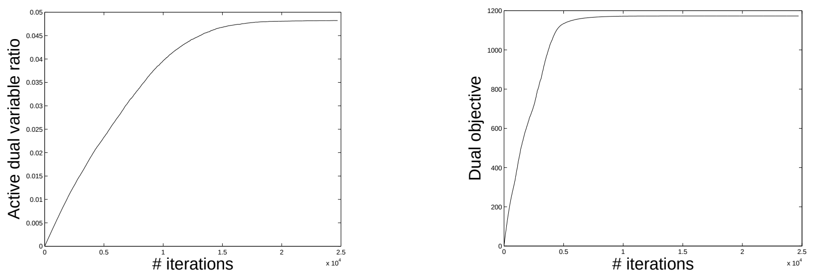


Figure 7.6: Optimization process of the hierarchical SVM on D section of the multiclass WIPO corpus. The objective of the dual problem is defined in Equation (4.14).

as the one in [14] that consider all variables belonging to the same instance in the subspace optimization. Since the active set increases slowly in our case and since we restrict optimization to the variables in the active set, our method needs more subspace optimizations, but needs to solve significantly smaller QPs in every iteration. To show how the computational complexity works out in a realistic example, we have trained the hierarchical SVM on the D section of WIPO-alpha with both optimization strategies. Our approach takes about 2,200 seconds with a final fraction of 4.6% non-zero dual variables. Without maintaining an active set, the learning has taken about 42,200 seconds with a larger active ratio of 4.9%. The sparser solution can be explained by the conservative manner of constructing the active set in our approach.

data	acc (%)		prec (%)		$\triangle^t$-loss		pacc (%)	
	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM
A, sampled	10.6	**11.7**	17.3	**20.5**	2.12	**1.87**	34.9	**43.2**
B, sampled	9.56	**11.3**	14.7	**18.9**	2.25	**1.99**	36.5	**45.6**
C, sampled	12.1	**13.3**	18.1	**20.7**	1.90	**1.69**	45.4	**53.0**
D, sampled	19.7	**20.5**	27.2	**30.9**	1.71	**1.54**	48.9	**57.3**
E, sampled	10.2	**11.4**	17.3	**20.6**	2.01	**1.82**	40.5	**48.3**
F, sampled	13.1	**14.5**	19.4	**22.8**	2.02	**1.75**	40.8	**50.5**
G, sampled	12.4	**13.6**	18.9	**22.4**	2.09	**1.87**	35.2	**43.5**
H, sampled	14.8	**15.7**	22.6	**25.0**	1.81	**1.66**	42.0	**48.0**

Table 7.7: Performance comparison of the flat SVM and the hierarchical SVM on the multi-class WIPO-alpha corpus with subsampling. The notations are the same as those in Table 7.5. The parameter ϵ was set to 0.1 for 'A, sample 3', 'B, sample 3', and 'C, sample 3', and 0.01 otherwise.

To examine the effect of the training set size, three-fold cross-validation has been performed on subsampled data. Table 7.7 shows that the hierarchical SVM outperforms the flat SVM. Moreover, the performance gains are more significant in the scenario with fewer training documents. Note that all measures on all runs have been improved. This demonstrates that the hierarchical formulation, which couples categories through the weight vectors of common ancestors, is particularly useful when operated on small training sets, since it allows more reliable estimates for weight vectors associated with higher-level nodes by effectively pooling observation in a manner similar to [41].

section	acc (%)		prec (%)		$\triangle^x$-loss		rloss (%)		pacc (%)	
	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM
A	53.8	**54.2**	56.0	**57.3**	1.66	**1.34**	9.09	**3.85**	70.3	**73.5**
B	37.3	**37.8**	40.8	**42.5**	2.08	**1.76**	12.6	**5.38**	59.9	**65.0**
C	45.3	**45.4**	45.5	**45.9**	2.10	**1.68**	10.1	**4.95**	67.8	**73.3**
D	47.3	**48.6**	52.7	**55.0**	1.82	**1.45**	11.7	**7.35**	67.7	**71.6**
E	38.7	**38.7**	44.9	**46.5**	1.99	**1.63**	12.7	**7.44**	63.7	**66.2**
F	36.6	**37.6**	41.3	**43.4**	2.07	**1.69**	11.6	**5.13**	59.7	**65.0**
G	**47.2**	47.2	52.3	**52.8**	1.73	**1.50**	10.5	**5.46**	64.9	**67.0**
H	48.7	**49.2**	55.1	**56.2**	1.63	**1.34**	8.25	**4.15**	66.5	**69.6**

Table 7.8: SVM experiments on the multilabel WIPO-alpha corpus. Each row is on categories under the specified top level node (i.e. section). The results are from random 3-fold cross-validation. Better performance is marked in bold face. Notations are the same as in Table 7.5.

Multilabel WIPO-alpha set

Table 7.8 summarizes the SVM performance on the multilabel WIPO-alpha corpus across the sections. The hierarchical SVM significantly outperforms the flat SVM in terms of hierarchical maximal loss, ranking loss, and parent accuracy in each individual setting. Moreover, the hierarchical SVM often produces higher classification accuracy and average precision with gains being more moderate. To see if the improvement is statistically significant, we conducted 10-fold cross-validation on section E and then paired permutation test. The achieved level of significance is less than 0.08 for one-accuracy, and less than 0.005 for the other four measures.

An important assumption of the hierarchical SVM is that the closer two classes are in the taxonomy, the more similar their class prototypes are. To see if this assumption is consistent with the data, we measure on the D section the cosine values of the angles between class centroids, which can be viewed as a rough estimate of class prototypes. We find that the closer two classes are in the given taxonomy, the smaller the angle between them tend to be. The average cosine values among class pairs with distance 2, 4, and 6 in the taxonomy are 0.21, 0.18, and 0.16 respectively. This agrees with our assumption.

In addition, in the subsampled corpus we randomly sampled 3 documents from

section	acc (%)		prec (%)		$\triangle^x$-loss		pacc (%)	
	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM
A, sampled	14.8	**16.3**	16.8	**20.5**	2.71	**2.28**	39.3	**47.3**
B, sampled	12.4	**14.0**	14.0	**17.9**	2.78	**2.39**	40.1	**48.6**
C, sampled	14.8	**16.3**	15.6	**18.2**	2.79	**2.23**	46.6	**58.1**
D, sampled	19.4	**21.3**	24.0	**27.3**	2.58	**2.06**	49.3	**56.3**
E, sampled	14.4	**15.2**	19.9	**22.4**	2.66	**2.19**	45.2	**50.4**
F, sampled	13.4	**14.9**	16.6	**20.2**	2.74	**2.25**	41.5	**51.1**
G, sampled	11.4	**12.4**	14.9	**18.4**	2.75	**2.40**	35.1	**41.9**
H, sampled	15.1	**16.2**	19.2	**22.6**	2.67	**2.12**	40.2	**48.4**

Table 7.9: SVM experiments on the multilabel WIPO-alpha corpus with subsampling. Three documents or less are sampled for each category.

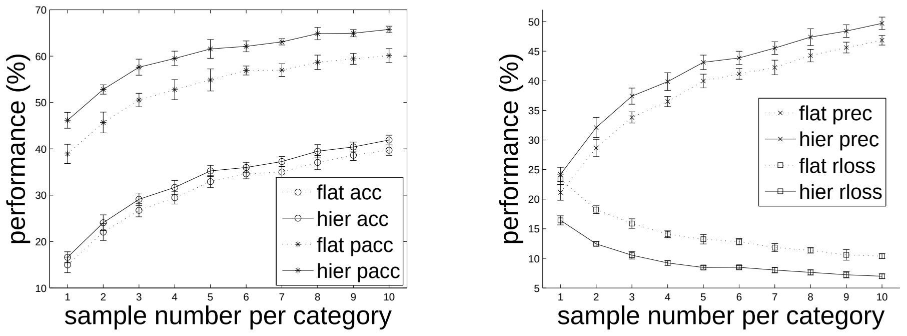


Figure 7.7: Flat and hierarchical SVM on section D of the multilabel WIPO-alpha data, with varying training set size. A small number of documents are sampled from each category for training purpose. The learned classifiers are tested on all remaining documents. This is repeated 10 times for each sampling number. The bars depict sample standard deviation. In this figure "flat" refers to the flat SVM while "hier" refers to the hierarchical SVM.

t	acc (%)		prec (%)		$\triangle^t$-loss		pacc (%)	
	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM
1	**17.1**	16.8	34.6	**34.8**	1.43	1.43	40.0	**40.1**
5	**38.1**	38.0	54.7	**55.3**	0.955	**0.942**	66.5	**68.0**
10	**49.8**	49.5	64.6	**65.1**	0.750	**0.743**	75.1	**76.2**
15	**54.1**	53.6	68.1	**68.4**	0.669	**0.668**	78.9	**79.5**
20	**58.7**	58.1	71.6	**71.7**	**0.591**	0.594	82.1	**82.6**
all	**84.8**	84.5	**90.7**	90.6	**0.204**	0.206	94.8	**94.9**

Table 7.10: Performance comparison of the flat and hierarchical SVMs on Newsgroup collection. t is the number of documents sampled from each class for training. The last row is by 10-fold cross-validation with all data.

each category to simulate the situation where data are only available in small quantities. The results in Table 7.9 show that the hierarchical SVM outperforms the flat SVM in all cases. The relative gains are somewhat higher than for the complete training set. Figure 7.7 demonstrates how the performance gains vary with the size of training data. We observe that hierarchical SVM excels in all runs. The gains appear to be slightly larger when the training set is smaller, except for one-accuracy.

Newsgroup set

We conduct 10-fold cross-validation on this set and find the performance of the hierarchical SVM is about the same as the performance of the flat SVM. Hierarchical SVM couples the weight vectors of nodes through their common ancestors. The closer two categories are in the taxonomy, the harder the hierarchical approach pushes their weight vectors to be similar. We hypothesize that this can be especially useful when the training data are sparse. In the newsgroup data, each category has $1,000$ documents. Therefore we perform subsampling. We randomly sample t documents from each category and evaluate the learned classifier on all remaining documents. The process was repeated 10 times and the measures were macro-averaged. Table 7.10 shows the results.

We observe that the performance difference between the two approaches is mild. However, the tendency is relatively stable. The hierarchical SVM always excels in parent accuracy. When the data are sparse, the hierarchical loss is usually also lower.

subtree	acc (%)		prec (%)		$\triangle^x$-loss		rloss (%)		pacc (%)	
	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM
H	**48.2**	47.0	**56.3**	55.4	2.26	**2.16**	11.3	**8.8**	**54.1**	53.2

Table 7.11: Performance comparison of the flat and hierarchical SVM on the H subtree of OHSUMED corpus. $C = 1$ and $\epsilon = 0.1$.

An interesting phenomenon is that the hierarchical model almost always produces higher average precision while the flat one always achieves a higher accuracy. This indicates that although the hierarchical SVM is not as good in predicting the correct label, it does tend to rank the correct category higher.

OHSUMED corpus

Experiments have been conducted on taxonomies formed by each top level node along with all nodes below it and the corresponding instances. The results show that the hierarchical SVM almost always beats the flat SVM in terms of ranking loss and the hierarchical maximal loss. However, in most cases, the hierarchical SVM produces lower one-accuracy, average precision, and parent accuracy. As an example, table 7.11 shows the 3-fold cross-validation outcome on H sub-taxonomy, which is indicative of the overall tendency.

The subsampling experiments, however, show different results. We randomly sampled two documents (or all available documents if the category possesses less than two documents) from each category as training data. Performance measures are evaluated on all the remaining documents. The performance measures over 10 repetitions are then averaged. The results are summarized in Table 7.12. We note that the hierarchical SVM outperforms the flat SVM in most cases, when the training data are scarce. The average precision, hierarchical maximal loss, and ranking loss are improved in all settings, with the improvement of ranking loss particularly significant.

ENZYME data

This is a multiclass set in which each instance is assigned to exactly one class. The results in Table 7.13 show that the hierarchical loss, a measure particularly suited to capture the performance of a hierarchical classification system, is improved by

subtree	acc (%)		prec (%)		$\triangle^x$-loss		rloss (%)		pacc (%)	
	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM
A, sampled	30.5	**34.2**	35.7	**38.1**	4.80	**4.37**	15.7	**9.6**	36.1	**40.7**
B, sampled	32.2	**33.8**	39.6	**41.5**	4.71	**4.22**	18.6	**11.7**	37.6	**39.1**
C, sampled	41.2	**45.0**	41.1	**43.4**	4.48	**4.26**	13.8	**7.6**	46.6	**50.7**
E, sampled	25.5	**28.7**	28.8	**30.8**	4.75	**4.43**	19.2	**13.0**	29.9	**33.6**
F, sampled	26.0	**27.6**	28.7	**29.9**	4.24	**4.03**	25.6	**19.5**	29.8	**31.6**
G, sampled	21.0	**24.1**	25.1	**25.2**	4.76	**4.40**	21.1	**12.7**	26.2	**29.9**
H, sampled	17.3	**18.1**	25.1	**25.9**	3.65	**3.37**	27.7	**21.6**	22.4	**24.0**
I, sampled	19.0	**19.8**	24.4	**25.1**	4.36	**4.02**	30.2	**24.0**	24.1	**25.3**
J, sampled	30.3	**30.5**	38.7	**39.8**	2.88	**2.60**	28.7	**21.9**	**39.5**	39.0
K, sampled	23.4	**26.2**	33.5	**36.0**	2.47	**2.35**	45.1	**41.1**	30.8	**31.2**
L, sampled	16.3	**16.4**	24.5	**25.0**	3.67	**3.46**	36.5	**29.2**	22.5	**23.0**
M, sampled	25.0	**25.6**	35.3	**36.1**	2.72	**2.36**	28.9	**23.0**	36.3	**37.0**
N, sampled	16.8	**18.4**	19.1	**20.1**	5.21	**5.00**	29.7	**21.5**	20.9	**22.8**
Z, sampled	**16.4**	16.2	26.3	**26.9**	3.19	**2.93**	41.8	**34.9**	**30.7**	29.5

Table 7.12: SVM experiments on OHSUMED with subsampling. Each row is on categories under the specified top level node. Two or less than two documents are randomly drawn from each category for training. Performance is evaluated on all remaining documents. The results are averaged over 10 such repetitions. $C = 1$ and $\epsilon = 0.1$. Experiments on sub-taxonomies rooted at D top-level nodes are too large to finish.

acc (%)		prec (%)		$\triangle^x$-loss		rloss (%)		pacc (%)	
F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM	F-SVM	H-SVM
34.2	33.6	**44.3**	43.7	1.94	**1.84**	13.2	**11.6**	38.3	**40.4**

Table 7.13: Performance comparison of the flat and hierarchical SVMs on ENZYME corpus. $\epsilon = 0.01$. Notations are the same as those in Table 7.5.

the hierarchical SVM. Compared the flat SVM, the hierarchical SVM also reduces the ranking loss and increases the one-accuracy at parent level. However, at the same time the hierarchical SVM achieves lower one-accuracy and average precision. So unlike the WIPO-alpha corpus in which the hierarchical SVM outperforms the flat SVM on almost all measures, the ENZYME set steadily favors the hierarchical SVM with respect to the hierarchical measures and ranking loss while favoring the flat SVM with respect to one-accuracy and average precision. The performance difference, compared to that on WIPO-alpha data, is also smaller.

In [50], the best 0/1 accuracy is 14.5% while ours is 34.2%. But this performance difference is likely due to the fact that they consider partial paths and multiple labels, although this in fact is a multiclass corpus with classes only at terminal levels of the tree taxonomy.

7.4.2 Comparison with Divide-and-Conquer SVM

In this section, we compare the hierarchical SVM with an intuitive and common construction of hierarchical classifiers that use divide-and-conquer techniques [31]. The latter builds a classifier at each interior node of a given taxonomy that learns to distinguish among its child nodes. To make a prediction, the method starts from the root and the classifier at a node routes the further decision to one of its child nodes; the terminal node that is reached at the end is the predicted class. Since we use the flat SVMs as the local classifiers, we call this method Divide-and-Conquer SVM (DNC-SVM).

The training process of DNC-SVM follows the procedure below.

1. For each interior node in the taxonomy, gather all training data that have at least one relevant label that is a descendent of the node in question. This set of data is then used as training data for the node.

2. The categories of the classifier at each interior node are its child nodes. If an instance has a category that is a descendent of a child node, then it is assigned to the child node. In multilabel data set, it is possible that an instance belongs to multiple children of a node.

3. Train a flat SVM at each node with the training data constructed as above.

section	acc (%)			pacc (%)		
	F-SVM	DNC-SVM	H-SVM	F-SVM	DNC-SVM	H-SVM
D	41.8	38.5	**42.8**	65.4	67.1	**69.1**
E	**34.7**	30.6	34.3	62.7	61.5	**64.2**
H	43.0	40.9	**43.1**	63.3	65.2	**66.2**

Table 7.14: Comparison of the flat SVM, Divide-and-Conquer SVM, and hierarchical SVM on some sections of the multiclass WIPO-alpha corpus.

t	acc (%)			pacc (%)		
	F-SVM	DNC-SVM	H-SVM	F-SVM	DNC-SVM	H-SVM
1	**17.1**	16.3	16.8	40.0	**44.2**	40.7
5	**38.1**	32.9	37.8	66.5	67.3	**68.0**
10	**49.8**	44.9	49.5	75.1	75.4	**76.2**
15	**54.1**	49.8	53.6	78.9	79.5	**79.5**
20	**58.7**	55.0	58.1	82.14	82.16	**82.6**
all	**84.8**	84.1	84.5	94.8	**95.0**	94.9

Table 7.15: Comparison of the flat SVM, Divide-and-Conquer SVM, and hierarchical SVM on Newsgroup data with different subsampling. t is the number of documents sampled from each category for training, with evaluation being performed on all remaining documents. Performance over 10 repetitions is macro-averaged and given in the table. The last row corresponds to 10-fold cross-validation.

subtree	acc (%)			pacc (%)		
	F-SVM	DNC-SVM	H-SVM	F-SVM	DNC-SVM	H-SVM
H, sampled	17.3	13.0	**18.1**	22.4	18.8	**24.0**
I, sampled	19.0	14.9	**19.8**	24.1	22.8	**25.3**
J, sampled	30.3	26.8	**30.5**	**39.5**	35.1	39.0
K, sampled	23.4	**27.0**	26.2	30.8	**33.4**	31.2
L, sampled	16.3	14.8	**16.4**	22.5	22.4	**23.0**
M, sampled	25.0	21.8	**25.6**	36.3	35.9	**37.0**
Z, sampled	**16.4**	8.7	16.2	30.7	**43.6**	29.5

Table 7.16: Performance comparison of the flat SVM, Divide-and-Conquer SVM, and hierarchical SVM on subsampled OHSUMED set. Each row is on categories under the specified top level node. Two or less than two documents are randomly drawn from each category for training. Performance is evaluated on all remaining documents. The results are averaged over 10 such repetitions. $C = 1$ and $\epsilon = 0.1$.

Since DNC-SVM does not produce a ranking of categories, measures such as average precision and ranking loss are not applicable. In empirical evaluation we have used one-accuracy and parent one-accuracy. We have conducted experiments on Newsgroup data and a few small to medium-sized subtrees of WIPO-alpha and OHSUMED collection. The results are summarized in Tables 7.14, 7.15, and 7.16. The results show that on WIPO-alpha and Newsgroup data, DNC-SVM leads to a higher parent one-accuracy than the flat SVM. This could be explained by the fact that DNC-SVM directly optimizes the classification accuracy at interior node. This is especially clear in Newsgroup corpus since it has only two levels of hierarchy. However, on the same corpora, DNC-SVM gives significantly lower one-accuracy than the flat SVM (often about 10% worse). It suggests that when the parent node is classified correctly, DNC-SVM has a lower rate of correctly predicting the leaf node than the flat SVM. Different from the WIPO-alpha and Newsgroup corpora, the OHSUMED set uses a taxonomy in which branches have various lengths and classes could be partial paths. On the subsampled OHSUMED set, the DNC-SVM not only almost always achieves worse one-accuracy than the flat SVM as baseline, but also frequently reaches lower parent one-accuracy than the flat SVM.

Table 7.14, 7.15, and 7.16 also show that the hierarchical SVM generally outperforms the DNC-SVM in terms of one-accuracy. In many cases, the relative improvements are on the scale of 10%. The hierarchical SVM also often outperforms the DNC-SVM in terms of parent one-accuracy. Although the DNC-SVM occasionally can yield accuracy that is significantly better than the flat SVM (for instance, see the parent one-accuracy in the $t = 1$ row of Table 7.15 and that in the last row of Table 7.16), so can it yield significantly worse accuracies. Meanwhile, although the accuracy gains of the hierarchical SVM over the flat SVM are relatively modest, the tendency that the hierarchical SVM outperforms the flat SVM is stable.

7.4.3 Using random hierarchy

In this section, we empirically evaluate the effect of using a random hierarchy. This is designed to verify the value of a true hierarchy and how the hierarchical SVM and DNC-SVM react to erroneous or noise-abundant class hierarchy.

The experiments are carried out on WIPO-alpha and Newsgroup sets. We let

method	acc(%)	prec(%)	pacc(%)
F-SVM	41.82	52.27	65.45
H-SVM	**42.81**	**54.42**	**69.13**
H-SVM random	39.72± 0.52	49.2± 0.42	63.85± 0.53

Table 7.17: Performance of the hierarchical SVM with random hierarchies on the D section of the multiclass WIPO-alpha corpus. "H-SVM" refers to the hierarchical SVM with the actual true hierarchy. The last line gives sample mean and standard deviation of performance of the hierarchical SVM with 30 random hierarchies. The number of nodes at successive tree depths (in increasing order) is 1, 7, 20 and 160.

method	$t = 20$			all		
	acc(%)	prec(%)	pacc(%)	acc(%)	prec(%)	pacc(%)
F-SVM	**58.74**	71.58	82.14	**84.82**	**90.73**	94.79
H-SVM	58.07	**71.74**	**82.57**	84.45	90.63	**94.92**
H-SVM rand1	57.59	70.80	81.47	84.43	90.43	94.58
H-SVM rand2	57.85	70.92	81.27	84.25	90.18	94.58
H-SVM rand3	57.70	70.86	81.24	84.46	90.48	94.58
H-SVM rand4	57.65	70.78	81.34	84.19	90.26	94.37
H-SVM rand5	57.82	71.12	81.52	84.65	90.63	94.64

Table 7.18: Performance of the hierarchical SVM with random hierarchies on Newsgroup data. Notations and experimental settings are the same as in Table 7.15. Each of the last five rows gives results of the hierarchical SVM using a random hierarchy.

a random hierarchy have the same depth and the same number of nodes at each tree level as the true hierarchy. But the edges between nodes at adjacent depths are uniformly randomly generated and each interior node is made sure to have at least one child node.

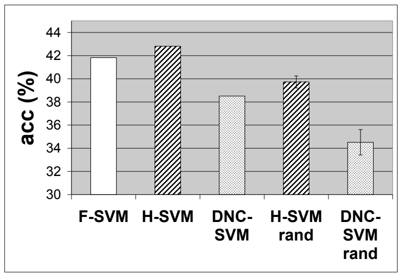
Table 7.17 gives the performance average of the hierarchical SVM over 30 random hierarchies on an exemplary section of WIPO-alpha corpus. It shows that using random hierarchy in the hierarchical SVM leads to worse performance than both the flat SVM and the hierarchical SVM with the actual hierarchy. The small sample standard deviation demonstrates that the actual hierarchy is an unlikely point in the random hierarchy distribution. The results suggest that the actual hierarchy contains useful information and our hierarchical SVM is able to take advantage of the information.

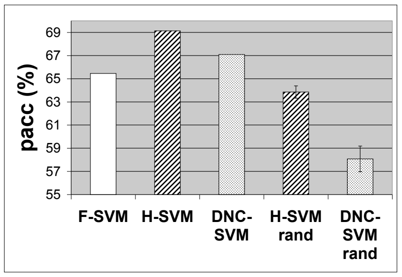
method	acc(%)	pacc(%)
F-SVM	**58.74**	82.14
DNC-SVM	54.96	**82.16**
DNC-SVM rand1	52.56	77.57
DNC-SVM rand2	52.87	77.08
DNC-SVM rand3	55.13	78.69
DNC-SVM rand4	52.34	76.71
DNC-SVM rand5	52.88	77.90

Table 7.19: Experiments of Divide-and-Conquer SVM using random hierarchies on Newsgroup data, with $t = 20$. The last five rows employ the same 5 random hierarchies as in Table 7.18.

Table 7.18 gives the results on Newsgroup set with five random hierarchies. We again observe the loss of classification performance by using random hierarchies, compared to the flat SVM or the hierarchical SVM with true hierarchies. The difference, however, is small. Moreover, we find that the performance decrease is smaller when more training data are available. This suggests that the influence imposed by a wrong hierarchy can be offset by using a larger set of training data. In the hierarchical SVM, when training data are sparse, more reliable estimates of class weight vectors are expected by pooling together instances from classes according to a class hierarchy. When the hierarchy itself is wrong, the procedure is misled and wrong smoothing occurs.

In addition, we have conducted experiments of the DNC-SVM with the same 5 random hierarchies on Newsgroup data. The results are summarized in Table 7.19. Figure 7.8 depicts the performance change of the hierarchical SVM and DNC-SVM when using random hierarchies on the D section of WIPO-alpha corpus. We observe that on both datasets the degradation of classification performance in DNC-SVM via using random hierarchies is considerably more severe than that in the hierarchical SVM. Indeed the DNC-SVM algorithm itself relies on the class taxonomy to a greater extent than the hierarchical SVM. In DNC-SVM, a greedy decision process is employed, which means a misclassification that happens at a higher-level node can not be recovered by classifications at lower-level nodes. When classes are randomly assigned to super-nodes, it is likely that super-nodes become heterogenous within





(a) (b)

Figure 7.8: Performance comparision of the hierarchical SVM and Divide-and-Conquer SVM with random hierarchies on D section of the multiclass WIPO-alpha corpus. We use "H-SVM" to indicate the hierarchical SVM with the true hierarchy, "DNC-SVM" to indicate the Divide-and-Conquer SVM with the true hierarchy, "H-SVM rand" to indicate the hierarchical SVM with random hierarchies, and "DNC-SVM" to indicate the Divide-and-Conquer SVM with random hierarchies. The performance sample mean and standard deviation of "H-SVM rand" and "DNC-rand" are computed over the same 30 random hierarchies that are used in Table 7.17.

themselves while they present no clear decision boundary between them. Thus classification at higher-level becomes more difficult, yet any misclassification becomes final due to DNC-SVM's greedy decision rule. The hierarchical SVM, on the other hand, is an all-together approach in which all parameters are learned simultaneously. Although a random taxonomy misleads learning, the weight vector of an erroenous interior node is not used by itself, but rather is always used together with other weight vectors that are along a same path from a class to a root. Therefore the hierarchical SVM is less susceptible to wrong or noisy taxonomies. The experiments demonstrate the robustness advantage of the hierarchical SVM.

7.5 Results with hierarchical Perceptron

In Chapter 5 three slightly different versions of hierarchical Perceptron algorithms have been proposed. The algorithms employ the same weight decomposition and update strategies, but differ in which variables are picked for update and how the final classifier is constructed. In this section, the hierarchical Perceptron algorithms

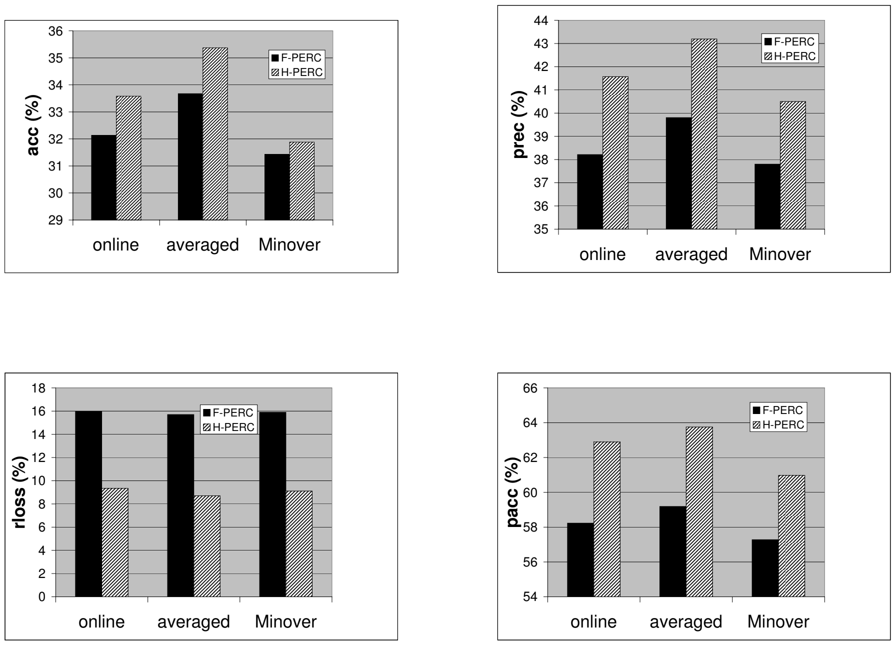


Figure 7.9: The flat and hierarchical Perceptron algorithms on the E section of the multilabel WIPO-alpha corpus. Three-fold cross-validation has been performed. The algorithms are allowed to run till convergence. We use "H-PERC" to indicate the hierarchical Perceptron algorithms and "F-PERC" to indicate the flat Perceptron algorithms.

will first be compared with their flat counterparts and among themselves. They will then be compared with the Hieron algorithm [19] which is also a hierarchical Perceptron-like algorithm.

7.5.1 Hierarchical Perceptron algorithms

Figure 7.9 gives an example of the performance of the three hierarchical Perceptron algorithms on the multilabel WIPO-alpha corpus. Figure 7.10 shows results on an

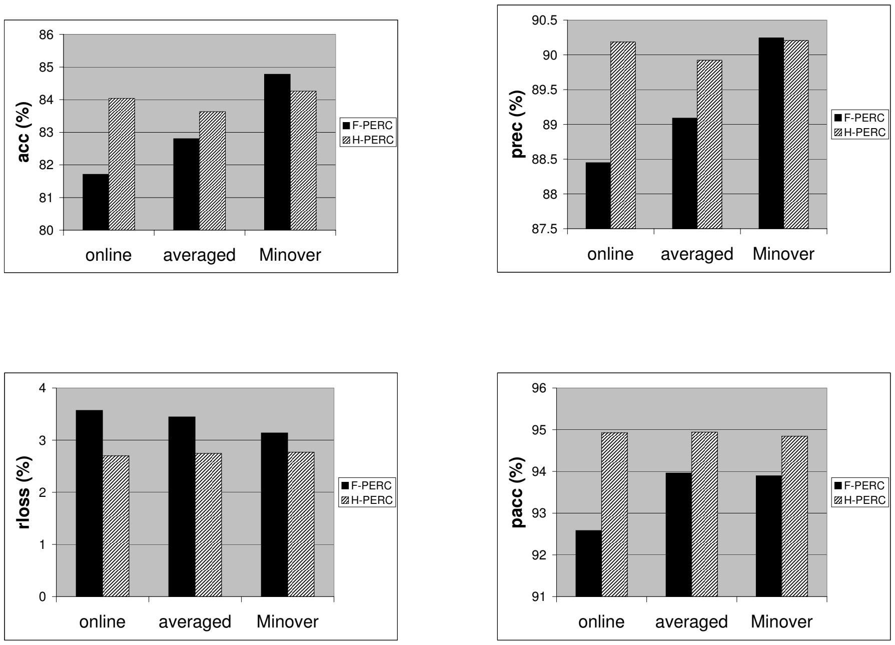


Figure 7.10: The flat and hierarchical Perceptron algorithms on the abridged newsgroup set, with 10-fold cross-validation. Under the condition that $\kappa = 100$, the Minover Perceptrons converged but not the online Perceptrons or averaged Perceptrons.

abridged version of the Newsgroup set.[1] All three variants of the hierarchical Perceptron outperform their corresponding flat algorithms with respect to the ranking loss and parent one-accuracy. The one-accuracy and average precision measures are improved in most cases as well. In addition, the Minover Perceptron algorithms usually achieve similar performance as the other two variants with lower active dual variable ratio. For instance, in experiments depicted in Figure 7.10, on average the solution of the Minover hierarchical Perceptron has 3.0% non-zero dual varaibles while the solution of the online hierarchical Perceptron has 3.8%.

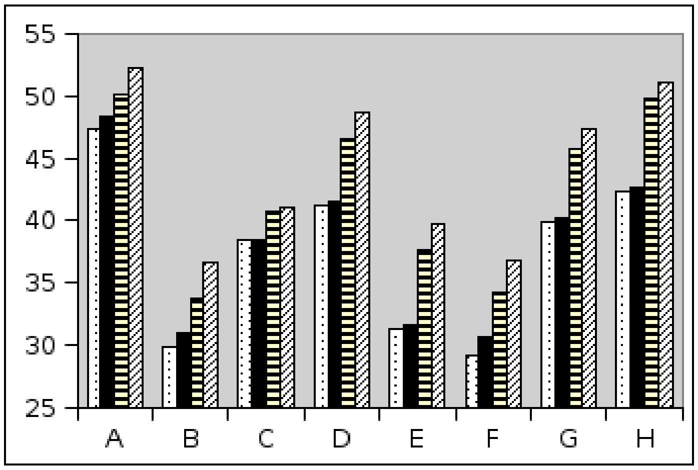


Figure 7.11: Results of the Minover flat and hierarchical Perceptron learning on the multilabel WIPO-alpha corpus. The four columns, from left to right, depict one-accuracy for the Minover flat and hierarchical Perceptron, and average precision for the Minover flat and hierarchical Perceptron.

Figure 7.11 depicts the performance of the Minover hierarchical Perceptron algorithm on the multilabel WIPO-alpha corpus. We allow the Perceptron to run until convergence. It takes significantly less time than SVM but reaches lower performance. We observe that the Minover hierarchical Perceptron performs better than the Minover flat Perceptron in all runs.

[1]The abridged set keeps all documents from the original data set except empty documents and documents that are identical but are assigned different classes (in this case only one class is randomly picked and kept). A small fraction of the Newsgroup articles are actually posted to multiple newsgroups. This is how the latter situation arises. If duplicate instances that are assigned to different classes are not removed, the Minover Perceptron algorithm would then cycle through the duplicate instances forever without making any progress.

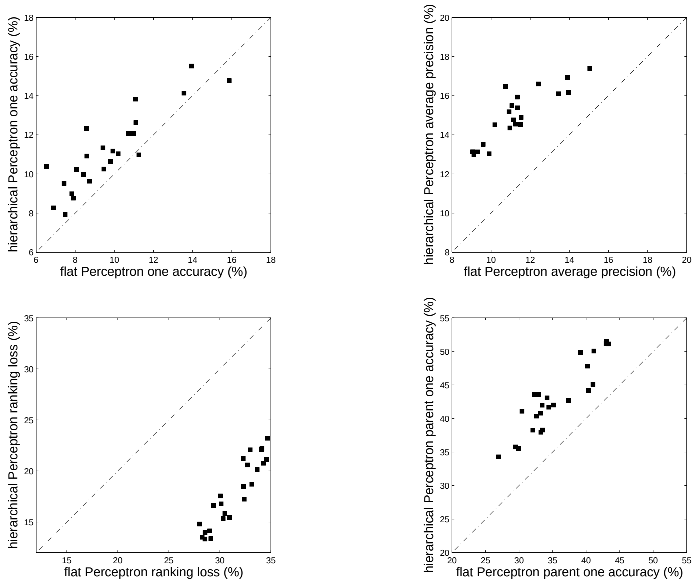


Figure 7.12: The Minover flat and hierarchical Perceptron on subsampled collection of the multilabel WIPO-alpha corpus.

Figure 7.12 compares the flat and hierarchical Minover Perceptron on the subsampled multilabel WIPO-alpha corpus. Each three-fold cross-validation on a random subset of documents under one section constitutes one sample in the figure, with each section contributing 3 samples. We observe the hierarchical approach helps with one-accuracy most times. It always significantly improves average precision, ranking loss and parent accuracy, with the gains on the first two metrics particularly large.

7.5.2 Hierarchical Perceptron versus Hieron

The Hieron algorithm is proposed for multiclass classification where classes are organized in trees [19]. The major difference between Hieron and our hierarchical Perceptron algorithms lies in the update rule. In Hieron, an update step is performed to minimize a distance between the new weight vector and the current weight vector while making sure a hierarchical margin is satisfied for the current instance. Let $\mathbf{x}_i$ be the current instance, y_i be the correct class of $\mathbf{x}_i$, and $\hat{y}_i$ be the predicted class for

$\mathbf{x}_i$. In our notation, the analytic solution of Hieron's optimization problem is

$$\tau \equiv \frac{\max\left(0\ ,\ \langle \mathbf{w}(\boldsymbol{\alpha}), \boldsymbol{\Phi}_i(y_i, \hat{y}_i)\rangle + \sqrt{\triangle(y_i, \hat{y}_i)}\right)}{|\mathrm{anc}(y_i) \ominus \mathrm{anc}(\hat{y}_i)|\ \|\mathbf{x}_i\|^2} \tag{7.8}$$

and the update to the current classifier is

$$\mathbf{w}^{(\mathrm{new})} = \mathbf{w}^{(\mathrm{old})} + \tau \boldsymbol{\Phi}_i(y_i, \hat{y}_i)\,. \tag{7.9}$$

A batch Hieron algorithm is also proposed in [19]. It not only takes the average of all generated discriminant functions as the final learned classifier in the same way as our averaged hierarchical Perceptron does, but also uses a hinge loss defined in [19] to pick variables to update. Due to the former, we call the algorithm the *averaged Hieron* algorithm for convenience.

Since our multiclass data sets (WIPO-alpha, Newsgroup, and ENZYME) all naturally have their categories in leaf nodes, we modify Hieron algorithms' classification rule so that it predicts only among leaf nodes. Note in our work, we have used half the distance between categories as the hierarchical loss. The Hieron algorithm has been modified to use our hierarchical loss, instead of the distance between categories proposed in [19]. This change only scales Hieron's solution weight vectors by $1/\sqrt{2}$, whether it is averaged Hieron or online Hieron. Therefore, this minor modification does not change the learned classifiers of Hieron.

We have first conducted a set of experiments on the Newsgroup set with different subsampling ratio. Table 7.20 summarizes the performance of the averaged flat and hierarchical Perceptron as well as that of the averaged flat and hierarchical Hieron. Table 7.21 compares the online Hieron and the online Perceptron on the Newsgroup corpus with 10-fold cross validation. We made the following observations.

1. As demonstrated in [19], the averaged algorithm usually performs significantly better than the online algorithm. This applies to both Hieron and Perceptron algorithms.

2. Our hierarchical Perceptron can often improve one-accuracy by exploiting the hierarchy. But the hierarchical Hieron often leads to lower one-accuracy than the flat Hieron.

t	Measure	F$^{\underline{a}}$Hieron	H$^{\underline{a}}$Hieron	F$^{\underline{a}}$PERC	H$^{\underline{a}}$PERC
10	acc(%)	**35.2**	34.9	36.2	**38.2**
	prec(%)	50.3	**52.7**	51.4	**54.9**
	$\triangle^t$-loss	1.08	**0.98**	1.05	**0.96**
	pacc(%)	57.0	**67.5**	59.2	**65.4**
20	acc(%)	**46.8**	44.5	46.1	**47.7**
	prec(%)	60.7	**61.5**	59.9	**63.2**
	$\triangle^t$-loss	0.83	**0.79**	0.85	**78.7**
	pacc(%)	70.0	**76.3**	69.3	**73.6**
all	acc(%)	**63.4**	62.5	64.9	**67.2**
	prec(%)	74.7	**75.9**	76.0	**78.7**
	$\triangle^t$-loss	0.52	**0.50**	0.49	**0.46**
	pacc(%)	84.9	**87.7**	85.7	**87.0**

Table 7.20: Performance comparison of the averaged Hieron and averaged Perceptron on the Newsgroup data. $\kappa = 20$. We use "F$^{\underline{a}}$Hieron" to indicate the averaged flat Hieron, "H$^{\underline{a}}$Hieron" to indicate the averaged hierarchical Hieron, "F$^{\underline{a}}$PERC" to indicate the averaged flat Perceptron, and "H$^{\underline{a}}$PERC" to indicate the averaged hierarchical Perceptron.

	Measure	F$^{\underline{o}}$Hieron	H$^{\underline{o}}$Hieron	F$^{\underline{o}}$PERC	H$^{\underline{o}}$PERC
all	acc(%)	29.5	**36.4**	34.7	**39.9**
	prec(%)	54.4	**58.7**	58.2	**62.4**
	$\triangle^t$-loss	1.24	**1.06**	1.13	**1.01**
	pacc(%)	46.7	**57.1**	52.0	**58.9**

Table 7.21: Performance comparison of the online flat and hierarchical Hieron and Perceptron on the Newsgroup data. We use "F$^{\underline{o}}$Hieron" to indicate the online flat Hieron, "H$^{\underline{o}}$Hieron" to indicate the online hierarchical Hieron, "F$^{\underline{o}}$PERC" to indicate the online flat Perceptron, and "H$^{\underline{o}}$PERC" to indicate the online hierarchical Perceptron.

section	Measure	$F^{\underline{a}}$Hieron	$H^{\underline{a}}$Hieron	$F^{\underline{a}}$PERC	$H^{\underline{a}}$PERC
D	acc(%)	**35.0**	33.4	34.6	**37.3**
	prec(%)	45.2	**45.4**	45.3	**49.1**
	$\triangle^t$-loss	1.39	**1.25**	1.39	**1.19**
	pacc(%)	59.0	**63.3**	59.1	**66.4**
E	acc(%)	**27.4**	26.5	27.4	**29.2**
	prec(%)	36.8	**37.4**	36.9	**39.6**
	$\triangle^t$-loss	1.62	**1.46**	1.62	**1.42**
	pacc(%)	55.4	**59.3**	55.0	**60.6**

Table 7.22: Comparison of the averaged Hieron and averaged Perceptron on the multiclass WIPO-alpha data. $\kappa = 20$. Refer to Table 7.20 for abbreviations.

3. Our hierarchical Perceptron often achieves better one-accuracy, average precision, and hierarchy-induced loss than the hierarchical Hieron, although a much simpler update rule is employed in the hierarchical Perceptron. The performance difference can be significant.

4. The hierarchical Hieron algorithm does not appear to achieve more performance gains against the flat Hieron than our hierarchical Perceptron algorithm against the flat Perceptron algorithm.

5. The hierarchical SVM (results in Table 7.10) significantly outperforms the hierarchical Perceptron and Hieron algorithms, although the performance gains of the hierarchical SVM over its flat counterpart are less significant than those of the hierarchical Perceptron or Hieron.

The observations made on the Newsgroup corpus also apply to WIPO-alpha data. Table 7.22 shows results on two smallest sub-taxonomies of the WIPO-alpha corpus.

Through experiments on the Newsgroup and WIPO-alpha data, we have seen that our hierarchical Perceptron algorithms are quite competitive with respect to Hieron, a considerably more complicated Perceptron-based hierarchical method. Our hierarchical Perceptron algorithm is able to take advantage of hierarchies to achieve performance gains that are on the same scale as Hieron. Our hierarchical Perceptron often outperforms the hierarchical Hieron.

section	acc (%)		prec (%)		$\triangle^t$-loss		pacc (%)	
	F$^{\underline{a}}$PERC	H$^{\underline{a}}$PERC$_a$	F$^{\underline{a}}$PERC	H$^{\underline{a}}$PERC$_a$	F$^{\underline{a}}$PERC	H$^{\underline{a}}$PERC$_a$	F$^{\underline{a}}$PERC	H$^{\underline{a}}$PERC$_a$
D	34.6	**34.9**	45.3	**47.3**	1.39	**1.36**	59.1	**59.2**
E	27.4	**27.6**	36.9	**38.1**	1.62	**1.57**	55.0	**55.8**
G	**34.2**	33.9	43.4	**44.4**	1.51	**1.50**	54.5	**55.1**
H	33.8	**34.4**	45.5	**47.6**	1.33	**1.29**	55.9	**57.8**

Table 7.23: Performance comparison of the averaged flat Perceptron (F$^{\underline{a}}$PERC) and the hierarchical Perceptron with the automatically generated hierarchy (H$^{\underline{a}}$PERC$_a$) on WIPO-alpha collection. In all experiments, $\kappa = 20$.

7.6 Results with automatic hierarchy learning

Our experiments show that when used in the hierarchical SVM, the hierarchy automatically learned from the training data actually causes slight performance decrease, compared to the flat SVM. The results on WIPO-alpha and Newsgroup show that the top loss and average precision can occasionally be slightly improved. But in most situations, the hierarchical SVM with an automatic hierarchy loses to the flat SVM with respect to most measures. The relative performance decrease is usually less than 1%.

However, the experiments with the hierarchical Perceptron and Hieron demonstrate that using the automatically generated hierarchy often boosts performance over the flat algorithms. Tables 7.23 and 7.24 summarize the results on the WIPO-alpha and Newsgroup collections of the hierarchical Perceptron with the automatically generated hierarchy. Experiments with the Hieron algorithm show similar performance comparison results and are not detailed here. In Table 7.23 and 7.24 the average precision, taxonomy-based loss, and parent one-accuracy are improved in all runs. The relative improvements of these metrics are usually in the range of one to five percentage point. This shows that a better ranking of classes is obtained by using automatic hierarchies. More often than not, the one-accuracy is also improved by using an automatically generated hierarchy. Table 7.24 also shows that when more data are available, the performance improvements of the hierarchical Perceptron with the automatically generated hierarchy over the flat Perceptron become smaller.

When compared to the hierarchical Perceptron with true taxonomies (see Tables 7.20 and 7.22 for experimental results), the hierarchical Perceptron with the

t	acc (%)		prec (%)		$\triangle^t$-loss		pacc (%)	
	F$^{\underline{a}}$PERC	H$^{\underline{a}}$PERC$_a$	F$^{\underline{a}}$PERC	H$^{\underline{a}}$PERC$_a$	F$^{\underline{a}}$PERC	H$^{\underline{a}}$PERC$_a$	F$^{\underline{a}}$PERC	H$^{\underline{a}}$PERC$_a$
10	36.2	**36.9**	51.4	**53.4**	1.05	**1.01**	59.2	**62.2**
20	46.1	**47.7**	59.9	**62.7**	0.85	**0.79**	69.3	**72.9**
all	64.9	**65.3**	76.0	**77.2**	0.49	**0.49**	85.7	**85.3**

Table 7.24: Performance comparison of the averaged flat Perceptron (F$^{\underline{a}}$PERC) and averaged hierarchical Perceptron with the automatically generated hierarchy (H$^{\underline{a}}$PERC$_a$) on Newsgroup collection. In all experiments, $\kappa = 20$.

automatically generated hierarchy always achieves worse taxonomy-based loss and parent one-accuracy. This is because these two metrics are evaluated according to the true taxonomies. In some cases, using the automatically generated hierarchy can lead to better one-accuracy and average precision than using the actual hierarchy.

We believe the empirical success of H-PERC$_a$ is because the automatically generated hierarchy captures some of the relations among classes in the true hierarchy. The automatically generated hierarchy usually contain more interior nodes than the actual hierarchy. For example, the number of nodes at each level of the D subtree of IPC hierarchy is 1, 7, 20, and 160, respectively. The automatically generated two-level hierarchy, however, contains 115 nodes at the first level and 160 nodes at the second level. So our hierarchy generation algorithm tends to be more conservative when pairing classes under a common ancestor. Figure 7.13 illustrates a taxonomy learned on 1% of the Newsgroup collection. We find that among the classes that our algorithm do place as siblings, for example in Figure 7.13 and Figure 6.1, it is more frequently that they are actually close in the true taxonomy than that they are distant.

7.7 Summary of results

We have empirically evaluated the algorithms proposed in the thesis and two other hierarchical algorithms on five data sets with respect to multiple classification metrics. The experimental results are summarized as follows. First, our hierarchical algorithms generally improve the hierarchical metrics (i.e. hierarchical loss and parent one-accuracy) and ranking loss against the flat algorithms. Experiments show that the

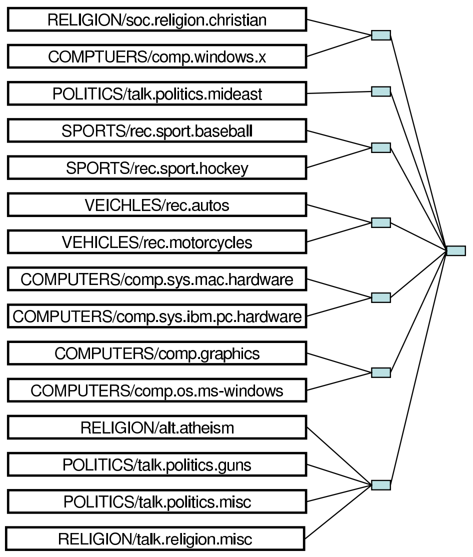

RELIGION/soc.religion.christian
COMPTUERS/comp.windows.x
POLITICS/talk.politics.mideast
SPORTS/rec.sport.baseball
SPORTS/rec.sport.hockey
VEICHLES/rec.autos
VEHICLES/rec.motorcycles
COMPUTERS/comp.sys.mac.hardware
COMPUTERS/comp.sys.ibm.pc.hardware
COMPUTERS/comp.graphics
COMPUTERS/comp.os.ms-windows
RELIGION/alt.atheism
POLITICS/talk.politics.guns
POLITICS/talk.politics.misc
RELIGION/talk.religion.misc

Figure 7.13: A taxonomy automatically generated on one percent of the Newsgroup corpus. The name of classes are prefixed with the name of its ancestor in the true taxonomy.

improvements are statistically significant. Whether the one-accuracy and average precision are boosted appear to depend on the category taxonomy and the data. Our hierarchical algorithms are particularly suited to cases in which the training data are sparse. Secondly, the performance gains of the hierarchical Perceptron over the flat Perceptron are higher than those of the hierarchical SVM over the flat SVM, while the SVMs usually achieve considerably better performance than the Perceptrons. Thirdly, compared to two other hierarchical classification approaches, our approaches either perform in the scale or perform better, with respect to classification metrics. When used with random hierarchies, our hierarchical approaches also perform more stably than a greedy divide-and-conquer method. Finally, our hierarchy learning algorithm, when used with the hierarchical Perceptron algorithms, is able to improve its classification performance with respect to most metrics over the flat Perceptron. It is also true with the Hieron algorithm.

We have used five benchmark data sets in the experiments. The data sets contain synthetic vectors, documents of various nature, and protein sequences. The category taxonomies range from perfect trees with classes all at leaf level to unbalanced trees with classes at any level of the taxonomies. Multiple metrics are employed to evaluate various aspects of a classifier.

Experiments on five data sets show that, with few exceptions, the hierarchical SVM leads to better hierarchical loss and ranking loss than the flat SVM. Parent one-accuracy is also improved on all data sets except on OHSUMED. Moreover, average precision is usually boosted by the hierarchical SVM on the synthetic data, WIPO-alpha, Newsgroup, and subsampled OHSUMED corpora. One-accuracy is the measure that is hardest to improve. We argue that it is because the flat SVM directly optimizes an upper bound on classification accuracy. Still, on the WIPO-alpha and subsampled OHSUMED sets, the hierarchical SVM algorithm usually achieves higher one-accuracy than the flat SVM. The experiments also show that the training set size matters. This is especially evidenced on the OHSUMED set. Experiments on the subsampled OHSUMED set show that the hierarchical SVM excels in almost all cases. However, on the complete OHSUMED data, only ranking loss and hierarchical loss are usually improved by the hierarchical SVM. Furthermore, subsampling experiments on WIPO-alpha and Newsgroup show greater performance gains when training data are scarcer.

The comparison with DNC-SVM shows that although DNC-SVM can occasionally produce much greater performance gains, the hierarchical SVM is considerably more stable and improves classification performance consistently. Furthermore, we observe that the hierarchical SVM is more resistant to errors in the class hierarchy than DNC-SVM via experiments using random hierarchies. The clear performance difference between the hierarchical SVM with the actual hierarchy and the hierarchical SVM with random hierarchies underscores the value of the true hierarchy.

We have presented three versions of the hierarchical Perceptron algorithm. Empirical results show that they usually beat their flat counterparts, by larger margin than the hierarchical SVM does. Comparison with Hieron shows that the performance of the hierarchical Hieron and the hierarchical Perceptron are on the same scale, with the hierarchical Perceptron often performing modestly better than the hierarchical Hieron. The performance gains of these two algorithms over their flat counterparts are on the same scale as well.

The experiments show that the automatic hierarchy learning algorithm we proposed work well with the hierarchical Perceptron and the hierarchical Hieron. The two learning algorithms with the automatically generated hierarchy usually beat their flat counterparts, although the improvements are usually less significant than those using the true hierarchies.

Chapter 8

Conclusions

Multilabel classification is the task to assign instances to a predefined set of classes, in which one instance can be assigned to one or more than one classes. Multilabel classification is a crucial instrument for organizing information and has been studied for decades. A potential drawback of traditional multilabel classification methods is that they do not take into account of the relationship between classes. However, many real world classification systems employ taxonomies to organize categories. The goal of this thesis is to take advantage of valuable domain information that is represented in category taxonomies. When predefined category taxonomies are given, we proposed two algorithms, the hierarchical SVM and the hierarchical Perceptron, to exploit the taxonomies. When no taxonomies are given, we proposed an algorithm to learn taxonomies automatically from the training data, and then the learned taxonomies can be used with existing hierarchical classification algorithms. This way hierarchical classification algorithms can be applied to multilabel classification tasks in general.

In the rest of this chapter, we summarize our contributions and then discuss some directions for future work.

8.1 Contributions

In this thesis, we have proposed two approaches to encode taxonomy knowledge, as described in Chapter 3. One is to directly encode taxonomy structure in the scoring function used to rank categories. We introduced hierarchical class attributes to achieve this. The goal of this approach is to “smooth” weight vector estimates by tying learning across categories according to the class taxonomy. The other approach is a novel taxonomy-based loss function between overlapping categories that is motivated by real applications. We argue that the hierarchical loss is well suited in capturing performance of a hierarchical classification system.

We have then incorporated the taxonomy representation into two learning architecture, yielding the hierarchical SVM (in Chapter 4) and the hierarchical Perceptron (in Chapter 5). The hierarchical SVM is a large margin architecture for hierarchical multilabel categorization. It extends the strengths of Support Vector Machine classification to take advantage of information about class relationships encoded in a taxonomy. The parameters of the model are fitted by optimizing a joint objective. A variable selection algorithm has been presented to efficiently deal with the resulting large quadratic program. The hierarchical Perceptron algorithm couples the discriminant functions according to the given hierarchy and employs the hierarchical loss to scale its update. Our methods work with arbitrary, not necessarily singly connected taxonomies, and can be applied more generally in settings where categories are characterized by attributes and relations that are not necessarily induced by a taxonomy.

When no class taxonomy is provided, we proposed a two-stage approach to first learn a class taxonomy automatically from the training data and then apply existing hierarchical classification algorithms. To this end, we presented a hierarchy learning algorithm that is modified from the hierarchical agglomerative clustering algorithm (in Chapter 6). The outcome of the hierarchy learning algorithm is a coarse and conservative two-level tree structure.

We draw the following conclusions from extensive empirical evaluation.

- *The hierarchical SVM is a state-of-the-art hierarchical classification algorithm.* SVMs are generally regarded as among the most competing classification algorithms. The flat SVM that the hierarchical SVM is based on and compared to also shows leading performance in empirical evaluation [14]. Comparison with the Maximum Margin Markov Network hierarchical classification (H-M^3) [50] (in Section 7.4.1), although preliminary, shows that the hierarchical SVM can perform comparably with other state-of-the-art hierarchical classification algorithms. Our hierarchical SVM runs on medium-sized data (with up to thousands of classes and tens of thousands of instances), which is also on the same scale as H-M^3 can handle. Although Hieron [19], another hierarchical classification algorithm, usually achieves higher performance gains by exploiting category taxonomy (in Section 7.4.1 and 7.5.2), the hierarchical SVM always significantly outperforms Hieron.

- *The hierarchical SVM is a better choice than the flat SVM if one cares more about hierarchical measures than about the flat measures.* Experiments in Section 7.4.1 demonstrates that the hierarchical SVM almost always improves the hierarchical measures and ranking loss over the flat SVM. This can be attributed to the fact that the hierarchical SVM explicitly optimizes an upper bound on the taxonomy-induced loss of the training data as well as the hierarchical form of the discriminant function. In many cases, the hierarchical SVM also produces higher one-accuracy and average precision. It might appear disappointing at first sight. However, be reminded that the flat SVM has a natural advantage in terms of one-accuracy since it directly optimizes an bound on the one-accuracy of training data. The observation that the hierarchical SVM is as competitive as the flat SVM in terms of one-accuracy and average precision is already a strength of the hierarchical SVM.

- *The hierarchical SVM is more stable than DNC-SVM, a divide-and-conquer hierarchical classification algorithm.* Section 7.4.2 shows that although DNC-SVM occasionally can yield accuracies that are significantly better than the flat SVM, so can it frequently yields significantly worse accuracies. In contrary, the hierarchical SVM consistently improves over the flat SVM. Moreover, we investigate the impact of random class hierarchies on the learning algorithms (in Section 7.4.3). When using random hierarchies, classification accuracies of both the hierarchical SVM and DNC-SVM decrease, since learning was misled by wrong hierarchies. However, the performance decrease is considerably more severe in DNC-SVM than in the hierarchical SVM. We think this is because in DNC-SVM, a erroneous hierarchy makes classification much harder at higher-level nodes and any misclassification at any taxonomy level is final in

DNC-SVM due to its greedy decision rule. On the other hand, the hierarchical SVM is a big-bang method that learns all parameters together. In the hierarchical SVM, classification only happens at terminal nodes and weight vectors of interior nodes are always used with other weight vectors due to our weight decomposition method. Thus misrepresented interior node poses smaller influence to the hierarchical SVM. Therefore we conclude that the hierarchical SVM is less sensitive than DNC-SVM to noises in taxonomies. This property makes the hierarchical SVM more amenable to automatically learned taxonomies, which are naturally going to be noisy relative to the true but unknown taxonomy.

- *The hierarchical Perceptron is a highly competitive online hierarchical learning algorithm.* On one hand, the hierarchical Perceptron performs on the same scale as Hieron [19], a more sophisticated hierarchical online algorithm (in Section 7.5.2. Often the hierarchical Perceptron performs better than Hieron. On the other hand, the hierarchical Perceptron usually improves all classification measures over the flat Perceptron (in Section 7.5.1). That the hierarchical Perceptron achieves more performance gains than the hierarchical SVM might be due to the fact that Perceptron is a less powerful learning algorithm than SVM and therefore it is easier to improve the decision boundary produced by Perceptron than an already optimized one produced by SVM. Therefore, in the scenario of online hierarchical classification, the hierarchical Perceptron is a strong candidate.

- *Our algorithms are especially suited to cases where training data are scarce.* We have conducted sub-sampling experiments in which models are trained on a randomly sampled subset of all training data, for the hierarchical SVM (in Section 7.4.1), for the hierarchical Perceptron (in Section 7.5.1), and the hierarchical Perceptron with the automatically learned taxonomy (in Section 7.6). Results show that the performance gains of our algorithms are more significant in the scenario with fewer training data. This demonstrates that our approaches, which couples categories through the weight vectors of common ancestors, is particularly useful when operated on small training sets, since it allows more reliable estimates for weight vectors associated with higher-level nodes by effectively pooling instances of their descendent classes. Hence, our hierarchical algorithms are particularly attractive to classification scenarios where training data are difficult or costly to collect.

- *In cases where class taxonomies are unknown, classification performance can be boosted by exploiting automatically learned taxonomies.* Our work is among the pioneering research on hierarchical classification with automatically learned taxonomies. Empirical evaluation (in Section 7.6) demonstrates that the hierarchical Perceptron and Hieron benefit from the automatic hierarchy, in the sense that they outperform their corresponding flat algorithms. We think this is because the learned taxonomy captures some of the relations among classes in the true taxonomy. This is also because the taxonomy learned via our algorithm is conservative, by only pairing classes that show strong coherence, and thus makes less mistakes in predicting siblings. With an automatic hierarchy generation algorithm, hierarchical learning algorithms can then be applied to flat classification tasks as well. This leads to improved classification performance when combining our hierarchy learning algorithm with the hierarchical Perceptron or Hieron algorithm.

In this dissertation, we have presented two highly competitive hierarchical classification algorithms and demonstrated the value of using automatically learned taxonomy in classification. The study on hierarchical learning is still in its early stage and is a promising area.

8.2 Future directions

In this section, we list several future directions that one could go based on our work.

Generalizing our models We proposed the hierarchical loss between label sets in Equation (3.12). However, this loss is hard to deal with directly since it involves sets of classes. Thus we approximated it with pairwise hierarchical loss. One direction of future work is to reformulate the hierarchical SVM and the hierarchical Perceptron to directly work with losses between label sets.

In this work, we employed a less general form of linear discriminants (for example, see Section 2.2.1). We restrict the linear discriminant to always pass origin by not allowing direct modelling of bias terms. This is due to several reasons, with the major one being that it is more difficult to efficiently solve the resulting large optimization problem in the hierarchical SVM if bias terms are introduced, since it couples the dual variables associated with different instances in a data-dependent fashion. One future direction is to generalize the hierarchical SVM to directly incorporate bias terms and solve it efficiently.

In some classification applications short descriptions of categories are available. These descriptions can serve as attributes of the classes. Class similarities thus derived can be plugged into the hierarchical SVM or Perceptron algorithm, yielding a new method of exploiting category description in classification task.

Encoding taxonomy In this dissertation, we proposed two methods to encode class taxonomy knowledge: the taxonomy-derived class attributes and hierarchical loss. Another method to examine is adjusting the input kernel (kernel between instances) in accordance with class taxonomy. We believe that two classes that are very close in the class hierarchy tend to have their data also close in the inherent feature space, while two classes that are distant in the class hierarchy tend to have their data also far away in the inherent feature space. The inherent feature space, summarized by the input kernel in this space, can be learned by adjusting the input kernel with the class taxonomy. For example, one can employ the kernel alignment technique [18] to find a linear combination of candidate input kernels that optimally aligns the class kernel induced by the combined input kernels and the class kernel induced by the taxonomy. Then the learned input kernel can be used with any kernelized algorithm such as SVM and Perceptron.

Using hierarchical classification when class taxonomy is unknown When class taxonomy is unknown, one could take a two-stage approach to first construct a taxonomy and then apply hierarchical classification. In Chapter 6, we proposed an algorithm to learn class taxonomy automatically. It is based on hierarchical agglomerative clustering. One direction to go is to modify other clustering algorithms to learn class hierarchy. Currently our learned taxonomy only contains two levels of nodes. More complicated taxonomy can be learned by exploring more sophisticated approaches.

Another direction is to explore one-stage approaches to learn class taxonomy and hierarchical classifiers at the same time. For example, we can modify the hierarchical SVM to achieve this purpose. The idea is to pick the taxonomy that maximizes the achieved margin. If not considering hierarchical loss, the hierarchical SVM utilizes the taxonomy information only through the class kernel: $L(c, c') = \langle \mathbf{\Lambda}(c), \mathbf{\Lambda}(c') \rangle$. A class kernel is good in the sense that it increases the separation margin. So following similar steps as those in [34], the learning problem becomes

$$\begin{aligned}
\min_{\mathbf{L}} \max_{\boldsymbol{\alpha}} \Theta(\boldsymbol{\alpha}, L) = &\sum_{i=1}^{n} \sum_{y \in Y_i, \bar{y} \in \bar{Y}_i} \alpha_{iy\bar{y}} \\
&- \frac{1}{2} \sum_{i,j} \sum_{\substack{y \in Y_i \\ \bar{y} \in \bar{Y}_i}} \sum_{\substack{r \in Y_j \\ \bar{r} \in \bar{Y}_j}} (\alpha_{iy\bar{y}} \alpha_{jr\bar{r}} \langle \mathbf{x}_i, \mathbf{x}_j \rangle \left[L(y, r) + L(\bar{y}, \bar{r}) - L(y, \bar{r}) - L(\bar{y}, r) \right]),
\end{aligned}$$

$$\text{s.t. } \alpha_{iy\bar{y}} \geq 0 \ (\forall i, y \in Y_i, \bar{y} \in \bar{Y}_i) \tag{8.1a}$$

$$\sum_{y \in Y_i, \bar{y} \in \bar{Y}_i} \alpha_{iy\bar{y}} \leq C \ (\forall i) \tag{8.1b}$$

$$\text{s.t.} \mathbf{L} \succeq 0 \tag{8.1c}$$

$$\mathbf{L} \in \mathcal{L} \tag{8.1d}$$

It is not sufficient to simply maximize the separation margin, which may lead to degenerate or poor solutions. Instead different criteria have been proposed in the literature. The two most popular ones involve maximizing the margin subject to (i) a constraint on the trace of the kernel matrix [34] or (ii) a constraint on the radius of the smallest enclosing sphere, the so-called radius-margin bound [11]. These constraints characterize the set $\mathcal{L}$ in Equation (8.1d).

Equation (8.1) can be converted to a Semi-definite Program (SDP) and thus solved by a SDP software package. The steps are similar to [34]. Due to the complexity of SDP, the cost can be prohibitive for even a small data set. Thus efficient optimization or approximation is desired.

Appendix A

Exemplary calculation of additional regularizer

In this appendix we calculate $\mathbf{u}^T\tilde{\mathbf{A}}^{-1}\mathbf{u}$, the additional regularizer in Section 4.4, for two example taxonomies. One is the taxonomy in Figure 4.5. The other is general perfect tree of depth 2. To simplify the discussion, we assume

$$\lambda_z(y) = \begin{cases} \lambda & \text{if } z \in \text{anc}(y) \\ 0 & \text{otherwise} \end{cases}$$

A.1 For taxonomy in Figure 4.5

Figure 4.5 is replicated here in Figure A.1 for readers' convenience.

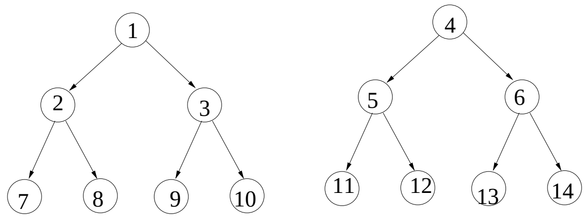


Figure A.1: A taxonomy comprising two perfect trees of depth 2. Node $1, 2, 3, 4, 5, 6$ are interior nodes and $7, 8, 9, 10, 11, 12, 13, 14$ are leaf nodes

Matrix $\mathbf{A}$ defined in Equation (4.40) takes value

$$\mathbf{A} = \begin{pmatrix} 5 & 2 & 2 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 2 & 2 \\ 0 & 0 & 0 & 2 & 3 & 0 \\ 0 & 0 & 0 & 2 & 0 & 3 \end{pmatrix}.$$

Hence

$$\mathbf{A}^{-1} = \begin{pmatrix} 3/7 & -2/7 & -2/7 & 0 & 0 & 0 \\ -2/7 & 11/21 & 4/21 & 0 & 0 & 0 \\ -2/7 & 4/21 & 11/21 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3/7 & -2/7 & -2/7 \\ 0 & 0 & 0 & -2/7 & 11/21 & 4/21 \\ 0 & 0 & 0 & -2/7 & 4/21 & 11/21 \end{pmatrix}.$$

$$\mathbf{u} = \frac{1}{\lambda} \begin{pmatrix} \mathbf{W}_7 + \mathbf{W}_8 + \mathbf{W}_9 + \mathbf{W}_{10} \\ \mathbf{W}_7 + \mathbf{W}_8 \\ \mathbf{W}_9 + \mathbf{W}_{10} \\ \mathbf{W}_{11} + \mathbf{W}_{12} + \mathbf{W}_{13} + \mathbf{W}_{14} \\ \mathbf{W}_{11} + \mathbf{W}_{12} \\ \mathbf{W}_{13} + \mathbf{W}_{14} \end{pmatrix}$$

By LU decomposition,

$$\mathbf{A}^{-1} = LU = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -2/3 & 1 & 0 & 0 & 0 & 0 \\ -2/3 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -2/3 & 1 & 0 \\ 0 & 0 & 0 & -2/3 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 3/7 & -2/7 & -2/7 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3/7 & -2/7 & -2/7 \\ 0 & 0 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/3 \end{pmatrix}$$

Therefore

$$\begin{aligned}\mathbf{u}^T\tilde{\mathbf{A}}^{-1}\mathbf{u} &= \mathbf{u}^T(\mathbf{A}^{-1}\otimes \mathbf{I}_{d\times d})\mathbf{u} = \mathbf{u}^T\left((LU)\otimes \mathbf{I}_{d\times d}\right)\mathbf{u}\\ &= \mathbf{u}^T(L\otimes \mathbf{I}_{d\times d})(U\otimes \mathbf{I}_{d\times d})\mathbf{u} = \left((L^T\otimes \mathbf{I}_{d\times d})\mathbf{u}\right)^T\left((U\otimes \mathbf{I}_{d\times d})\mathbf{u}\right)\end{aligned}$$

We obtain

$$(L^T\otimes \mathbf{I}_{d\times d})\mathbf{u} = \frac{1}{\lambda}\begin{pmatrix} \frac{1}{3}(\mathbf{W}_7+\mathbf{W}_8+\mathbf{W}_9+\mathbf{W}_{10})\\ \mathbf{W}_7+\mathbf{W}_8\\ \mathbf{W}_9+\mathbf{W}_{10}\\ \frac{1}{3}(\mathbf{W}_{11}+\mathbf{W}_{12}+\mathbf{W}_{13}+\mathbf{W}_{14})\\ \mathbf{W}_{11}+\mathbf{W}_{12}\\ \mathbf{W}_{13}+\mathbf{W}_{14}\end{pmatrix}$$

$$(U\otimes \mathbf{I}_{d\times d})\mathbf{u} = \frac{1}{\lambda}\begin{pmatrix} \frac{1}{7}(\mathbf{W}_7+\mathbf{W}_8+\mathbf{W}_9+\mathbf{W}_{10})\\ \frac{1}{3}(\mathbf{W}_7+\mathbf{W}_8)\\ \frac{1}{3}(\mathbf{W}_9+\mathbf{W}_{10})\\ \frac{1}{7}(\mathbf{W}_{11}+\mathbf{W}_{12}+\mathbf{W}_{13}+\mathbf{W}_{14})\\ \frac{1}{3}(\mathbf{W}_{11}+\mathbf{W}_{12})\\ \frac{1}{3}(\mathbf{W}_{13}+\mathbf{W}_{14})\end{pmatrix}$$

Finally

$$\begin{aligned}\mathbf{u}'H^{-1}\mathbf{u} =& \frac{1}{\lambda^2}\Big(\frac{\|\mathbf{W}_7+\mathbf{W}_8+\mathbf{W}_9+\mathbf{W}_{10}\|^2}{3\times 7}+\frac{\|\mathbf{W}_7+\mathbf{W}_8\|^2}{3}+\frac{\|\mathbf{W}_9+\mathbf{W}_{10}\|^2}{3}\\ &+\frac{\|\mathbf{W}_{11}+\mathbf{W}_{12}+\mathbf{W}_{13}+\mathbf{W}_{14}\|^2}{3\times 7}+\frac{\|\mathbf{W}_{11}+\mathbf{W}_{12}\|^2}{3}+\frac{\|\mathbf{W}_{13}+\mathbf{W}_{14}\|^2}{3}\Big).\end{aligned}$$

A.2 For general perfect trees of depth 2

The following proof is on a single perfect tree with depth 2. As seen in the previous section, trees in a forest do not overlap or interact with one another in the computation. So $\mathbf{u}'\tilde{\mathbf{A}}^{-1}\mathbf{u}$ over a forest is the juxtaposition of $\mathbf{u}^T\tilde{\mathbf{A}}^{-1}\mathbf{u}$ values over individual trees.

For a perfect tree with depth 2 such as any tree in Figure 4.5, number the nodes from root to leaves by increasing positive integer. Denote the outgoing degrees of any interior node by ρ. Thus

$$\mathbf{A} = \begin{pmatrix} \rho^2+1 & \rho & \rho & \rho & \dots & \rho \\ \rho & \rho+1 & 0 & 0 & \dots & 0 \\ \rho & 0 & \rho+1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \rho & 0 & 0 & 0 & \dots & \rho+1 \end{pmatrix} \tag{A.1}$$

is a $\rho+1$ by $\rho+1$ matrix.

A result on inverse of partitioned matrix is that given $A(m\times m), B(m\times n), C(n\times m), D(n\times n)$ and that D and $(A-BD^{-1}C)$ are nonsingular, then [39]

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} (A-BD^{-1}C)^{-1} & -(A-BD^{-1}C)^{-1}BD^{-1} \\ -D^{-1}C(A-BD^{-1}C)^{-1} & D^{-1}+D^{-1}C(A-BD^{-1}C)^{-1}BD^{-1} \end{pmatrix}$$

Using this rule, we have

$$\mathbf{A}^{-1} = \begin{pmatrix} \frac{\rho+1}{\rho^2+\rho+1} & -\frac{\rho}{\rho^2+\rho+1}\mathbf{e}' \\ \frac{\rho}{\rho^2+\rho+1}\mathbf{e} & \frac{\rho^2\mathbf{e}\mathbf{e}'}{(\rho^2+\rho+1)(\rho+1)} + \frac{\mathbf{I}}{\rho+1}, \end{pmatrix} \tag{A.2}$$

where $\mathbf{e}$ is $\rho\times 1$ matrix of all ones, and $\mathbf{I}$ is identity matrix of $\rho\times\rho$. LU decomposition gives

$$\begin{aligned} \mathbf{A}^{-1} &= LU \\ &= \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ -\frac{\rho}{\rho+1} & 1 & 0 & \dots & 0 \\ -\frac{\rho}{\rho+1} & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ -\frac{\rho}{\rho+1} & 0 & 0 & \dots & 1 \end{pmatrix} \times \begin{pmatrix} \frac{\rho+1}{\rho^2+\rho+1} & -\frac{\rho}{\rho^2+\rho+1} & -\frac{\rho}{\rho^2+\rho+1} & \dots & -\frac{\rho}{\rho^2+\rho+1} \\ 0 & \frac{1}{\rho+1} & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{\rho+1} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \frac{1}{\rho+1} \end{pmatrix} \end{aligned} \tag{A.3}$$

Define $\sum \mathbf{W}_o \equiv \sum_{y:y\in\mathcal{Y},o\in\text{anc}(y)} \mathbf{W}_y$. Therefore

$$\mathbf{u} = \frac{1}{\lambda}\left((\sum \mathbf{W}_1)^T, (\sum \mathbf{W}_2)^T, \dots, (\sum \mathbf{W}_{\rho+1})^T\right)^T$$

Remember node 1 is the root with node $2,\dots,\rho+1$ being its child nodes.

$$(L^T\otimes \mathbf{I}_{d\times d})\mathbf{u} = \frac{1}{\lambda}\left(\frac{1}{\rho+1}(\sum \mathbf{W}_1)^T, (\sum \mathbf{W}_2)^T, \dots, (\sum \mathbf{W}_{\rho+1})^T\right)^T \tag{A.4}$$

$$(U\otimes \mathbf{I}_{d\times d})\mathbf{u} = \frac{1}{\lambda}\left(\frac{1}{\rho^2+\rho+1}(\sum \mathbf{W}_1)^T, \frac{1}{\rho+1}(\sum \mathbf{W}_2)^T, \dots, \frac{1}{\rho+1}(\sum \mathbf{W}_{\rho+1})^T\right)^T \tag{A.5}$$

Therefore

$$\mathbf{u}'\tilde{\mathbf{A}}^{-1}\mathbf{u} = \frac{1}{\lambda^2}\left(\frac{1}{(\rho+1)(\rho^2+\rho+1)}\|\sum W_1\|^2 + \sum_{o=2}^{\rho+1}\frac{1}{\rho+1}\|\sum \mathbf{W}_o\|^2\right) \tag{A.6}$$

Bibliography

[1] L. Douglas Baker and Andrew K. McCallum. Distributional clustering of words for text classification. In W. Bruce Croft, Alistair Moffat, Cornelis J. van Rijsbergen, Ross Wilkinson, and Justin Zobel, editors, *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval*, pages 96–103, Melbourne, AU, 1998. ACM Press, New York, US.

[2] D. Boley. Hierarchical taxonomies using divisive partitioning. Technical Report TR-98-012, Department of Computer Science, University of Minnesota, Minneapolis, 1998.

[3] L. Cai and T. Hofmann. Text categorization by boosting automatically extracted concepts. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 182–189, 2003.

[4] Lijuan Cai and Thomas Hofmann. Hierarchical document categorization with support vector machines. In *Proceedings of the Conference on Information and Knowledge Management (CIKM)*, 2004.

[5] Lijuan Cai and Thomas Hofmann. Exploiting known taxonomies in learning overlapping concepts. In *20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[6] A. Cardoso-Cachopo and A. L. Oliveira. An empirical comparison of text categorization methods. In *Proceedings of the 10th International Symposium on String Processing and Information Retrieval (SPIRE)*, number 2857 in Lecture Notes in Computer Science, pages 183–196. Springer Verlag, 2003.

[7] Carnegie Mellon University Center for Intelligent Information Retrieval. The lemur toolkit for language modeling and information retrieval. URL, 2004. http://www-2.cs.cmu.edu/ lemur/.

[8] N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004.

[9] N. Cesa-Bianchi, C. Gentile, A. Tironi, and L. Zaniboni. Incremental algorithms for hierarchical classification. In *Advances in Neural Information Processing Systems*, 2005.

[10] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Hierarchical classification: Combining bayes with svm. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[11] Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1–3):131–159, 2002.

[12] S. Charkabarti, B. Dom, R. Agrawal, and P. Raghavan. Unsing taxonomy, discriminants, and signatures for navigating in text databases. In *Proceedings of the 23rd Conference on Very Large Databases (VLDB)*, pages 560–573, 1997.

[13] The Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. *Nat Genet*, 25:25–29, 2000.

[14] K. Crammer and Y. Singer. On the algorithmic implementation of multi-class kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[15] Koby Crammer and Yoram Singer. A family of additive online algorithms for category ranking. *Journal of Machine Learning Research*, 3:1025–1058, 2003.

[16] Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, 2003.

[17] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, March 2000.

[18] Nello Cristianini, John Shawe-Taylor, André Elisseeff, and Jaz Kandola. On kernel-target alignment, 2001.

[19] Ofer Dekel, Joseph Keshet, and Yoram Singer. Large margin hierarchical classification. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, 2004.

[20] Ayhan Demiriz, Kristin P. Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1–3):225–254, 2002.

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

[22] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2000.

[23] S. T. Dumais and H. Chen. Hierarchical classification of Web content. In *Proceedings of the 23rd ACM International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 256–263, 2000.

[24] André Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In *Proceedings of the Neural Information Processing Systems conference (NIPS)*, pages 681–687, 2001.

[25] W. R. Hersh, C. Buckley, T. J. Leone, and D. H. Hickam. Ohsumed: An interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 192–201, 1994.

[26] Thomas Hofmann, Lijuan Cai, and Massimiliano Ciaramita. Learning with taxonomies: Classifying documents and words. In *Workshop on Syntax, Semantics, and Statistics, Neural Information Processing (NIPS)*, 2003.

[27] R. V. Hogg and J. Ledolter. *Engineering Statistics*. MacMillan, 1987.

[28] The Math Works Inc. Statistics toolbox user's guide, 2007.

[29] T. Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of the 10th European Conference on Machine Learning (ECML)*, number 1398, pages 137–142. Springer Verlag, 1998.

[30] Thorsten Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In Douglas H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 143–151, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[31] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *Proceedings of the 14th International Conference on Machine Learning (ICML)*, 1997.

[32] Werner Krauth and Marc Mézard. Learning algorithms with optimal stability in neural networks. *Journal of Physics A.*, 20(745):L745–L752, 1987.

[33] A. Kyriakopoulou and T. Kalamboukis. Text classification using clustering. In *Proceedings of the ECML-PKDD Discovery Challenge Workshop*, 2006.

[34] Gert Lanckriet, Nello Cristianini, Peter Bartlett, Laurent Ghaoui, and Michael Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[35] D. D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. In *Proceedings of the 10th European Conference on Machine Learning (ECML)*, pages 4–15, 1998.

[36] D. D. Lewis and M. Ringuette. A comparison of two learning algorithms for text classification. In *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval*, pages 81–93, 1994.

[37] Tao Li, Shenghuo Zhu, and Mitsunori Ogihara. Hierarchical document classification using automatically generated hierarchy. *Journal of Intelligent Information Systems.* http://springerlink.metapress.com/content/wu5lr58j1547787j/fulltext.pdf, to appear as a regular paper.

[38] Yaoyong Li and Kalina Bontcheva. Hierarchical, perceptron-like learning for ontology-based information extraction. In *Proceedings of the 16th international conference on World Wide Web*, pages 777–786, 2007.

[39] Helmut Lutkepohl. *Handbook of matrices.* John Wiley and Sons, 1996.

[40] A. McCallum, R. Rosenfeld, T. Mitchell, and A. Ng. Improving text clasification by shrinkage in a hierarchy of classes. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 359–367, 1998.

[41] Andrew K. McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In Jude W. Shavlik, editor, *Proceedings of ICML-98, 15th International Conference on Machine Learning*, pages 359–367, Madison, US, 1998. Morgan Kaufmann Publishers, San Francisco, US.

[42] D. Mladenić and M. Grobelnik. Feature selection for classification based on text hierarchy. In *Proceedings of the Conference on Automated Learning and Discovery*, 1998.

[43] National Library of Medicine. Medical subject headings. URL, 2004. http://www.nlm.nih.gov/mesh/meshhome.html.

[44] H. T. Ng, W. B. Goh, and K. Low. Feature selection, perception learning and a usability case study. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 67–73, 1997.

[45] A. Novikoff. On convergence proofs for perceptrons. In *Proceeding of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1963.

[46] Nomenclature Committee of the International Union of Biochemistry and In consultation with the IUPAC-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) Molecular Biology (NC-IUBMB). Enzyme nomenclature – recommendations of the nomenclature committee of the international union of biochemistry and molecular biology on the nomenclature and classification of enzymes by the reactions they catalyse. URL, 2006. http://www.chem.qmul.ac.uk/iubmb/enzyme/.

[47] Open Directory Project. dmoz. URL, 2004. http://www.dmoz.org.

[48] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*, pages 185–208. MIT Press, 1999.

[49] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[50] Juho Rousu, Craig Saunders, Sandor Szedmak, and John Shawe-Taylor. Kernel-based learning of hierarchical multilabel classification models. *Journal of Machine Learning Research*, 7:1601–1626, 2005.

[51] Juho Rousu, Craig Saunders, Sandor Szedmak, and John Shawe-Taylor. Learning hierarchical multi-category text classification models. In *22nd International Conference on Machine Learning (ICML)*, 2005.

[52] M. E. Ruiz and P. Srinivasan. Hierarchical text categorization using neural networks. *Information Retrieval*, 5(1):87–118, 2002.

[53] G. Salton. Developments in automatic text retrieval. *Science*, 253:974–980, 1991.

[54] R. E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

[55] R. E. Schapire, Y. Singer, and A. Singhal. Boosting and Rocchio applied to text filtering. In *Proceedings of 21th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 215–223, 1998.

[56] H. Schütze, D. A. Hull, and J. O. Pedersen. A comparison of classifiers and document representations for the routing problem. In *Proceedings of 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 229–237, 1995.

[57] N. Slonim and N. Tishby. The power of word clusters for text classification. In *23rd European Colloquium on Information Retrieval Research*, 2001.

[58] A. Sun and E.-P. Lim. Hierarchical text classification and evaluation. In *International Conference on Data Mining (ICDM)*, pages 521–528, 2001.

[59] Aixin Sun and Ee-Peng Lim. Hierarchical text classification and evaluation. In *ICDM*, pages 521–528, 2001.

[60] Benjamin Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *Proceedings of Neural Information Processing Systems conference (NIPS)*, 2003.

[61] K. Toutanova, F. Chen, K. Popat, and T. Hofmann. Text classification in a hierarchical mixture model for small training sets. In *Proceedings of the Tenth International ACM Conference on Information and Knowledge Management (CIKM)*, 2001.

[62] I. Tsochantardis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, 2004.

[63] R. J. Vanderbei. LOQO: An interior point code for quadratic programming. *Optimization Methods and Software*, 11:451–484, 1999.

[64] Vladimir N. Vapnik. *Statistical Learning Theory.* Wiley-Interscience, September 1998.

[65] K. Wang, S. Zhou, and S. C. Liew. Building hierarchical classifiers using class proximity. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *Proceedings of VLDB-99, 25th International Conference on Very Large Data Bases*, pages 363–374. Morgan Kaufmann Publishers, San Francisco, US, 1999.

[66] A. S. Weigend, E. D. Wiener, and J. O. Pedersen. Exploiting hierarchy in text categorization. *Information Retrieval*, 1(3):193–216, 1999.

[67] J. Weston and C. Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, 1998.

[68] World Intellectual Property Organization. International patent classification. URL, 2001. http://www.wipo.int/classifications/en/.

[69] World Intellectual Property Organization. Wipo-alpha sataset. URL, 2003. http://www.wipo.int/ibis/datasets.

[70] Yahoo! Corporation. Yahoo! web site directory. URL, 2004. http://www.yahoo.com.

[71] Hsin-Chang Yang and Chung-Hong Lee. A text mining approach on automatic generation of web directories and hierarchies. *Expert Systems with Applications*, 27(4):645–663, 2004.

[72] Y. Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *Proceedings of 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 13–22, 1994.

[73] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 42–49, 1999.

[74] Ying Zhao and George Karypis. Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 10(2):141–168, 2005.