The Dissertation Committee for Kunal Vinod Kumar Punera
certifies that this is the approved version of the following dissertation:

Enhanced Classification through Exploitation of Hierarchical Structures

Committee:

Joydeep Ghosh, Supervisor

Ross Baldick

Sarfraz Khurshid

Raymond Mooney

Andrew Tomkins

Enhanced Classification through Exploitation of Hierarchical Structures

by

Kunal Vinod Kumar Punera, B.E.; M.S.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2007

Dedicated to my parents, Cdr. Vinod Punera and Shashi Punera.

Acknowledgments

My 27 year journey from birth in Goa, India, to obtaining the Ph.D. degree at the University of Texas at Austin, has been aided by numerous people in a myriad of ways. I feel that God has blessed me with more than my fair share of love, support, guidance, friendship, and of course, technical assistance. In these pages I would like to acknowledge and express my gratitude towards everyone who helped me on this journey; I do so in roughly the chronological order of meeting them.

While this dissertation would certainly not have been possible without the constant encouragement and support of my family, their key contributions were made much earlier. I thank Dad for instilling in me the principles of sportsmanship and a confidence in my abilities, and Mom for shaping my outlook in terms of moderation and adaptability. To both I am thankful for teaching me that hard work performed and knowledge gained is never wasted. In these principles, I might never quite be able to live up to the standards they set in their lives, but to do so continues to be my goal. Finally, I would like to give my love to my little sister, Tripti, and to tell her that I am proud of what she has achieved and am excited to see what else the future holds for her.

The most important contribution to this dissertation is that of my advisor, Prof. Joydeep Ghosh, who has, throughout my graduate studies, mixed in just the right amount of guidance along with copious amounts of freedom, all the while backed up with unshakable support. He was very encouraging when my

research digressed into areas not of interest to him and very patient while I was looking for a thesis topic. His input on my research, coming from considerable intellectual depth and experience, has been an invaluable learning resource for me. While he has left indelible marks on this dissertation, much more importantly, I thank him for his unique style of advising and accessible nature that have greatly shaped me as a researcher. I have been extremely lucky to have had him as my graduate advisor.

In addition, I thank Prof. Ross Baldick, Prof. Sarfraz Khurshid, Prof. Raymond Mooney, and Dr. Andrew Tomkins for serving on my Ph.D. dissertation committee, and for all their input on my research and thesis writeup.

I would like to thank all my colleagues at the IDEALab for creating an excellent work atmosphere and for their invaluable help with my research. In particular, I learnt a lot from my interactions with Sreangsu Acharyya, Arindan Banerjee, Meghana Deodhar, Gunjan Gupta, Goo Jun, Manish Katyal, Chase Krumpelman, Alex Liu, Srujana Merugu, and Suju Rajan, especially during our weekly group meetings. A special thanks to Srujana, Suju, and Meghana for the warm friendships, eager help, and home cooked food, both in and away from Austin. Finally, I thank Amy Levin and Melanie Gulick for all their help with successfully negotiating the administrative landscape of UT-Austin.

During the course of my stay in the US, I have been fortunate to make some very close friends with whom I have spent nearly all my time away from the lab. In Austin Siddarth Krishnan, Shobha Vasudevan, and Vinod Viswanathan have been a family to me. I thank them for the countless enjoyable hours spent talking, laughing, and playing. I thank Naresh Rajkumar, Anu Murthy, Srujana Merugu, and Suju Rajan for providing me a similar home in the San

Francisco bay area.

A significant part of my development as a researcher has happened during internships. I have spent many happy months working at Yahoo! Research and IBM Almaden Research Center, and I would like to thank everyone I interacted and collaborated with during those wonderful learning experiences. In particular, apart from all other things, I thank Andrew Tomkins for taking on the role of advisor while I was away from Austin, Andrei Broder for agreeing to mentor my internship at Almaden in the summer of 2005 after three successive mentors of mine moved to Yahoo, Ravi Kumar for patiently investing precious time in our joint work and getting me interested in theoretical research, and Deepayan Chakrabarti, Sandeep Pandey, and Aris Anagnostopoulos for insightful discussions over hyper-competitive games of foosball. I also spent a year as a research assistant to Prof. Soumen Chakrabarti and Prof. Rushikesh Joshi at Indian Institute of Technology - Bombay right after my undergraduate studies, and I would like to thank them for introducing me to high quality research that early in my career.

Lastly, and most importantly, I would like to express my affection, my respect, and my gratitude to my lovely wife, Darpana. In the last few months as I have spent more and more time in the abstract world of computer science she has often been my one anchor to the real-world, and has provided a much needed diversity in perspective. I thank her with all my heart for her love, patience, and understanding in the face of considerable time and monetary constraints. It is now time to start our much awaited "after the thesis" life.

Enhanced Classification through Exploitation of Hierarchical Structures

Publication No. _____

Kunal Vinod Kumar Punera, Ph.D.
The University of Texas at Austin, 2007

Supervisor: Joydeep Ghosh

Humans often organize information by encoding it in structures that link together entities such as concepts, objects, properties etc. Among the various structures possible, hierarchies are commonly used. For instance, taxonomies of categories commonly employ hierarchies to indicate that one category "is a" type of another. The Yahoo! Web Directory and the Open Directory Project are two examples of large taxonomies where topics are hierarchically arranged. Hierarchies are also used to recursively decompose composite objects into their constituent parts. Examples of this are webpages that can be parsed and then represented as DOM-trees, where the DOM nodes correspond to sections of the webpages.

In this thesis we argue that these hierarchical relationships between entities can be exploited to facilitate common data mining tasks defined upon them, like automated classification. Specifically, we show that the information encoded in these hierarchies can be reduced to constraints on class membership scores that can then be enforced as a post-processing step to enhance the

accuracy of classification. We demonstrate our ideas and algorithms on three real-world tasks.

First, we tackle the problem of classification into hierarchical taxonomies. We show how different taxonomy structures can be translated into constraints on the outputs of classifiers learned at the nodes of the hierarchy. In addition, we give algorithms to optimally enforce these constraints and show that this results in improved classification accuracy. In cases where the taxonomies are not available, we give an approach to automatically derive hierarchical relationships amongst a flat set of categories. Next, we work on the problem of detecting noisy (templated) parts of webpages. We give algorithms that rate each section of a webpage in terms of how templated it is. Then we show that smoothing the output of these template classifiers over the DOM-tree hierarchy improves the template detection performance of our system. Finally, we investigate the task of segmenting websites into topically cohesive regions. We define a framework and within it a set of measures that characterize good segmentations, and give an efficient algorithm to find the best segmentation within this framework.

We formalize the problem of enforcing constraints on the outputs of classifiers as regularized isotonic or unimodal regression on rooted trees; these are generalizations of the classic isotonic regression problem. The nature of the constraints as well as the cost functions is different in each of the applications mentioned above. For all these formulations we give efficient algorithms to optimally smooth the classifier outputs. These novel formulations and algorithms might be of interest independent of the applications in this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In philosophy, the term Ontology refers to the study of existence or being. Broadly speaking, a key aspect of an ontology is the categorization of entities into groups, and the study of relationships amongst these groups. More specifically, in the fields of computer and information sciences, the term ontology refers to a data model that represents concepts within a domain and the relationships between these concepts [Gru93].

The basic components that make up ontologies are the *individuals*, also referred to as objects or instances that are being studied and reasoned about. Examples of these are concrete objects such as people, automobiles, documents etc, or even abstract entities like numbers or vowels. These instances are described completely by a set of properties and their values; we call them *features*. For instance, automobiles might be described by features like number of axles, weight, horsepower, color etc. Concepts (also called *classes*) are often more abstract groups of instances and/or other classes. For example, in

our ontology of automobiles, Car, Truck, and Sedan might be three distinct classes. The class Car might group together the instances Honda-Civic and Ford-Mustang, and also the class Sedan. Finally, ontologies contain a set of *relations* that link classes to instances as well as link classes amongst themselves. An important type of relation is subsumption, which denotes that an entity *is-a* type of another entity[1]. This can be used to define which instances are members of which classes. When applied between classes, the subsumption relation results in hierarchies of classes; the child class *is-a* type of the parent class (in the above example, Sedan *is-a* Car). Such a system of classes, the relationships amongst them, and the rules of memberships of instances to classes, are often collectively referred to as a *taxonomy*.

Throughout our intellectual history, humans have sought to organize their knowledge of the world with the help of taxonomies. Around 300 BC, Aristotle devised a taxonomy of all possible things that can be referred to in sentences; he called this treatise "Categories" [1]. Some of the classes in this taxonomy were Substance, Quantity, Quality, Position, Action etc. Carolus Linnaeus, who is known as the "father of modern taxonomy", attempted to categorize all living things into a canonical biological taxonomy in the eighteenth century. The levels of the Linnaean taxonomy divided organisms into Kingdoms, then Classes, and then Orders etc. Even today, the use of taxonomies for organization spans the gamut of knowledge from objects as ephemeral as webpages [DMO] to concepts as eternal as human goals [CRW01].

Of the different taxonomical structures possible, those with subsumption relations linking classes have become ubiquitous as knowledge organization

[1]Another important relation is *meronymy*, which represents how entities combine together to form other composite entities [WCH87].

tools. In addition to some mentioned above, taxonomies such as the US Patent Office class codes, the Library of Congress catalog, the phylogenetic "Tree of Life" [Me04], and even the ACM Computing Classification System are hierarchical in nature. In general, taxonomies structured as hierarchies make it easier to navigate and access the data as well as to maintain and enrich it. This is especially true in the context of the World Wide Web where the amount of available information is overwhelming. Therefore, it is not surprising that many Internet topic directories such as Yahoo! [Yah] and DMOZ [DMO] are organized as hierarchies.

Apart from classes in hierarchical taxonomies, other types of objects are also often arranged in hierarchies. In many domains, the instances placed in taxonomies are not atomic objects, but themselves have an internal hierarchical structure (often formed of meronymic relations). For example, a webpage can be represented by a tree of DOM nodes, which represent parts of the webpage - the content and formatting instructions - and are connected via containment relations [HHW+04]. As another example, consider a set of webpages that are individual chapters, sections, and subsections of a thesis. In this case, the instances (webpages) are related to each other by the overall hierarchical organization of the thesis (chapters contain sections etc).

In this thesis we study domains and scenarios where classes, objects, or parts of objects are related to one another via hierarchical relationships. We propose ways in which this hierarchical information can be translated into gains in accuracy of data mining tasks such as classification. Finally, we evaluate these approaches and showcase their efficacy on real-world applications like text classification, template detection and website segmentation.

Automated Construction of Taxonomies.

Traditionally, taxonomy construction has been done manually by humans. When Linnaeus attempted to classify all of nature into a hierarchy, he manually compared the physical characteristics of organisms while deciding which Kingdom, Class, and Order, to place them in. Even as recently as a few years ago, both Yahoo! and DMOZ taxonomies were constructed and populated manually by employees and volunteers[2]. This manual process is, however, time-consuming, expensive, and, in the case of an changing and expanding corpus like the World Wide Web, inherently incomplete. Hence, there has been a lot of recent research on automating the taxonomy construction process [KGC02, PRG05, VD04, ST99, SKO01].

In Chapter 3 we tackle the problem of constructing a hierarchical taxonomy of classes automatically. We propose a framework that characterizes a "good" taxonomy, and also provide an algorithm to construct it [PRG06]. Our approach improves on prior work by avoiding unnecessary restrictions on the hierarchical structure learned (we allow n-ary trees) and not requiring any user-defined parameters to be specified. We compare our work to an existing approach that yields taxonomies structured as binary trees, and our empirical results show that n-ary tree based taxonomies constructed by our approach group classes in more "natural" ways.

Exploiting Hierarchical Taxonomies for Classification.

Given that a large portion of human knowledge is currently in the form of taxonomies, it is natural that there is considerable interest in being able to automatically place instances into appropriate classes. Consider a taxonomy

[2] `www.dmoz.org/about.html`

with the following set of classes: Car, Truck, and Motorcycle. We want to find a function that takes as input a vector of features-values - number of axles, weight, horsepower, color - and outputs the label of the vehicle - Car, Truck, or Motorcycle - which these attributes describe. The task we have just described is known as *classification* in machine learning literature. The standard model of classification of instances into a "flat" set of classes has been extensively studied and numerous techniques have been proposed to solve it [Hay99, MN98, Mit97, PR03, Qui, Vap95]. Under this model, each instance is classified based on only its features-values, and each class does little more than group similar instances together, separate from instances of other classes.

However, there is often some structure in the classification problem, and this can be exploited to improve accuracy. One form of structure is subsumption relationships between classes in the taxonomy. Suppose the taxonomy mentioned above contains an additional class Sedan that *is-a* a type of class Car. In such a taxonomy, instances can have multiple labels: the instance Honda-Civic belongs to both the classes Sedan and Car. However, the structure within the taxonomy places limits on these labellings: an instance cannot be labeled both a Sedan and a Motorcycle. More importantly, the structure helps in inference of class memberships: if we know an instance belongs to class Sedan, we can be sure that it also belongs to class Car. Contrapositively, our knowledge that an instance is not a Car, also informs us that it is not a Sedan. Using this additional structure in taxonomies to devise improved classification functions has also been studied [CDAR98, DC00, KS97, MRMN98, PRG06, TJHA05, Fal96].

In Chapter 4 we show how the information implicitly encoded in the hierarchical structure of taxonomies can be translated into constraints on the

class membership scores of instances. Furthermore, we argue that these constraints can be enforced as a post-processing step to correct errors introduced by classifiers and hence to improve classification accuracy. Alternate taxonomy structures and the different forms of constraints they generate are also discussed. We formalize the problems of enforcing constraints on membership scores as regularized generalizations of the classic isotonic and unimodal tree regression problems. We also provide efficient dynamic programming based algorithms to find optimal solutions to these problems. Empirical evaluation on real-world domains shows that enforcing constraints on membership scores derived from the hierarchical relationships between classes of a taxonomy results in improved accuracy of classification.

Exploiting Other Sources of Hierarchical Information.

Another source of structure, which can be exploited for improving accuracy in classification problems, is the hierarchical relationships amongst instances being classified. The way objects in some domains are linked to one another often constrains how they can be classified. Consider the template detection problem, where parts of webpages have to be labeled either templates (irrelevant content) or non-templates (relevant content). Webpages are represented by a rooted tree structure based on their HTML code (DOM trees [HHW+04]), and so the problem boils down to assigning "templateness" scores to nodes of a tree. However, the scores for nodes (sections of webpages) are constrained by the edges in the tree structure. For example, if we declare a node to be non-template, we cannot have its parent labeled as template; content of a page section (parent node) that contains relevant content (child node) cannot be called irrelevant. Recently, specialized classification algorithms have

been proposed to exploit this kind of structure amongst instances to improve accuracy in various domains [CDI98, DGMS01, PDG02]

In Chapter 5 we investigate the problem of template detection on the Web. We begin by reporting on two large-scale studies we performed to measure the volume and evolution of templates [GPT05]. The results from the studies show that more than half of all content on the web is within templates (is irrelevant). Having established the gravity of the problem, we introduce both local and global methods that assign "templateness" scores to sections of a webpage. These scores are then adjusted to conform to constraints derived from the tree structure within the webpage [CKP07]. We formulate this constraint enforcement problem as isotonic tree regression with step regularization, and give an efficient exact algorithm to solve it. Through large-scale experiments we show that our approach to enforcing constraints as a post-processing step accurately segments a webpage into template and non-template parts.

In Chapter 6 we tackle the problem of topically segmenting the hierarchical structure of a large website [KPT06]. As in the template detection problem, the class labels of a webpage (leaf in the URL directory tree of the website) are constrained based on labels of other webpages in the same directory (internal node in the URL tree). Unlike the template detection problem, however, only leaf nodes in the tree are assigned membership scores. The goal is to find segments of the URL tree such that content within them is topically cohesive. We develop a set of cost functions that can be used to measure the quality of a segmentation and provide an efficient algorithm to find the best segmentation in this framework. Through extensive experiments on human-labeled data we confirm the soundness of our framework and show that a judicious choice of cost functions allows the algorithm to perform surprisingly accurate topical

segmentations.

In summary, this thesis introduces ideas and algorithms through which information implicitly encoded in hierarchical relationships can be exploited in principled ways to improve accuracy of data mining tasks. Two very distinct sources of this information are considered; the subsumption relations between classes in a taxonomy and the hierarchical arrangement of instances being classified. Moreover, as we show, the existence of different types of taxonomies as well as the diverse ways in which instances link to each other lead to further variations in the type of information encoded in edges of a hierarchy. In our work we describe how all these different types of hierarchical relationships can be translated into constraints on class membership scores. In order to enforce some of these types of constraints we introduce regularized versions of the classic isotonic and unimodal tree regression problems and give algorithms to solve them exactly[3].

We motivate these abstract problems through diverse real-world applications such as hierarchical classification, template detection, and website segmentation. For each problem, we construct end-to-end systems in which our work on exploiting hierarchical relationships is an important sub-system. For example, in the case of hierarchical classification, we also give an approach to construct hierarchical taxonomies automatically. While, in the template detection work, we perform large-scale studies on the prevalence of noisy content on the Web, and construct a classifier that labels irrelevant sections of webpages as templates. These applications also help us showcase the improvements in performance due to our approach to enforcing constraints in hierarchical data.

[3]These novel problem formulations and the exact algorithms we give to solve them might of interest independent of applications discussed in this thesis

Chapter 2

Background and Related Research

2.1 Introduction

In broad terms, this thesis deals with work on automated taxonomy construction, hierarchical classification, isotonic regression, template detection, and topical website segmentation. Here we review some background information on the machine learning concepts used in the later chapters. Basic undergraduate level knowledge of algorithms [KaT05] and probability [Fel68] is assumed and not covered.

The problems covered in this thesis have received considerable attention in the data mining and machine learning research community in recent years. In this chapter we briefly describe some of this prior research. For detailed descriptions of these works readers are referred to the original publications.

2.2 Clustering and Classification

Machine learning problems are typically studied under the broad dichotomy of unsupervised and supervised learning. The goal of unsupervised learning is to discover "interesting" structure in a dataset $X = \{x_1, \ldots, x_n\}$ assumed to be have been drawn i.i.d. from a distribution $\mathcal{D}$. Some unsupervised learning approaches try to estimate the form and parameters of the distribution that likely produced dataset X. Clustering is one form of unsupervised learning.

Clustering. The goal in clustering is to partition a dataset into clusters (groups), such that pairs of data instances from within the same group are more "similar" to each other than pairs taken from across groups. Many different approaches to finding clusters have been proposed, and they can be broadly divided into two categories: *Partitional* and *Agglomerative*. Partitional methods try to divide the dataset into exhaustive and disjoint partitions such that each data instance is closest to the partition it belongs to. Differences in aspects such as representation of clusters and the metrics used to measure distances (or similarities) give rise to various partitional algorithms. One of the most popular partition-based approaches is k-Means [Mac67], which represents each cluster by the mean of the instances that comprise it and measures the distance between an instance and a cluster mean using the Euclidean metric. Variations of k-Means differ in the measures used to compute distance, such as cosine distance [DFG01], KL-distance [DMK03], and Bregman divergences [BMDG05]. Partitional clustering algorithms can also be model-based [DLR77, BDGS05], where each cluster is represented by a density function and goal of learning is to estimate the parameter values that most likely generated the observed data. Other partition-based approaches use ad-hoc notions of density of data [EKSX96, SEKX98, ABKS99] or partition the

similarity graph of instances [Cha04, SG02] to locate clusters.

Agglomerative clustering approaches [JD88] differ from partition-based ones in that they find partitions of the dataset at all levels of granularity, effectively constructing a hierarchy. These methods are discussed in detail in the next section.

CLASSIFICATION. In the supervised learning setting, along with the data X we also are given a vector y with a target value for each data instance. It is assumed that the pairs $\{x_i, y_i\}$ are sampled i.i.d. from a joint distribution $\mathcal{X} \times \mathcal{Y}$. The task is to learn the function $\tau : \mathcal{X} \rightarrow \mathcal{Y}$; when $y_i \in \mathcal{Y}$ are discrete labels, this task is called classification.

Depending on how they learn the function τ, classification algorithms are categorized as *generative* or *discriminative*. Generative approaches first try to model the conditional density functions $\mathrm{p}(x|y)$ for each class and then make predictions using Bayes rule. Linear Discriminant Analysis [DHS00], Bayesian Networks [PR03], and the popular Naive Bayes algorithm [MN98] are examples of generative learning methods. In contrast, discriminative learning approaches try to directly learn a function that predicts the posterior probability $p(y|x)$ or the class labels y. Examples of discriminative learning algorithms approach are Decision trees [Qui], Neural Networks [Hay99], Support Vector Machine [Vap95], and Logistic Regression [NJ02].

Within classification, the special case of $y_i \in \{0, 1\}$ - binary classification - has been very well studied. Most well known learning approaches, like C4.5 (decision tree) [Qui93], neural network algorithms such as Perceptron [Ros58] and Backpropogation [RHW86], and Support Vector Machine [Vap95] are best suited for two class problems. Even theoretical analysis of learning is better

established for binary classification [Val84, Vap95]. Hence, much research has focused on finding ways to solve multi-class problems by reducing them to multiple binary classification problems [DB95, ASS00, CS02, RK04].

While unsupervised and supervised learning represent two extremes of the machine learning landscape, recent work has explored problems that fall into the middle ground. Typically, in these problems two sets of data are provided, with ground truth available for only one set; hence, these problems are referred to as semi-supervised learning [CSZ06]. In one of the earliest such approaches, Nigam et al. [NMTM00] present an algorithm that improved classification accuracy by exploiting large amounts of unlabeled data. Other work has focused on improving clustering accuracy by using a small amount of labeled data in the form of constraints [BBM02, BBM04]. A related problem is that of transductive learning [Joa99], where the task is to learn a classification function for a specific test dataset, and not for the entire data space.

2.3 Taxonomy Construction via Hierarchical Clustering

Hierarchical taxonomies are used for organizing knowledge in many domains [Me04, CKKS02, DMO]. In Chapter 3, we present a novel approach to automatic construction of n-ary tree based taxonomies. In this section, we review existing work on hierarchical clustering and its use in taxonomy construction.

Semi-automatic Taxonomy Building. The taxonomy construction process involves the specification of a hierarchical system of classes as well as placing data into the nodes of this hierarchy. In the past, both these processes have typically been done by hand by humans. For example, both Yahoo [Yah]

and DMOZ [DMO] taxonomies were created manually by employees and volunteers[1]. However, as the size of taxonomies grow - DMOZ has over 500 thousand classes and millions of webpages - the manual process of creation and maintenance becomes expensive and time consuming. Moreover, the issues of scale are exacerbated by the dynamic and expanding nature of domains like the World Wide Web.

Gates et al. [GTC05] describe a system for semi-automatic construction of a large general purpose taxonomy for categorization of Web and intranet documents. They also present arguments in favor of automatic construction of taxonomies as opposed to manual labeling of documents. Other efforts on semi-automatically defining taxonomies and labeled data for text categorization systems include the InfoAnalyzer system by Zhang et al. [ZLPY04] and a Self-Organizing Maps based approach by Adami et al. [AAS03]. Finally, there has been some recent work on identifying hierarchical relationships between concepts via "folksonomies" [Kom05], which are organizations and categorizations developing on the web from user-generated tags and content.

Hierarchical Clustering. Hierarchical clustering algorithms group data instances together to produce nested partitions of all possible sizes. Hence, they try to arrange instances as leaves of a tree such that each anti-chain of nodes represents a good clustering of that size. These hierarchies can be formed in either a bottom-up or a top-down manner. The bottom-up or agglomerative approach starts with each data instance forming a cluster on its own. Then the clusters that are closest according to some distance measure are merged successively until the termination criterion is satisfied. The clas-

[1] `www.dmoz.org/about.html`

sic single-link and complete-link algorithms [JD88] are examples of bottom-up approaches. The top-down or divisive approach starts with all instances in the same cluster, and successively splits a clusters into smaller clusters until the stopping condition is fulfilled. Any partitional clustering algorithm, such as k-Means, can be fit into the divisive framework to generate hierarchies via repeated bisections. There are also some approaches which try to improve upon the greedy nature of agglomerative clustering by employing a hybrid search phase [PG05, ZRL96, ZK05], or by using complex splitting and merging criteria [GRS98, KHK99].

While the algorithms mentioned above are discriminative in nature, many generative model-based hierarchical clustering methods have also been proposed. Tishby et al. [TPB99] present a top-down method based on the Information Bottleneck principle that finds clusters using a deterministic annealing procedure. Other approaches that employ deterministic annealing to create hierarchies are given in [GGPC02, KGC02, Hof99]. Finally, there has been some recent work on learning mixture model hierarchies [GR04, SKO01, VD00b].

Automatic Taxonomy Construction. Many of the above mentioned algorithms can be used to construct taxonomies by applying them to the set of class means. Hierarchical agglomerative clustering [JD88] applied to class means works by considering each class as a separate cluster and then successively merging clusters that are closest to each other. This results in a topic hierarchy structured as a rooted binary tree with classes at the leaves. Similarly, divisive hierarchical clustering algorithms [TPB99, GGPC02] can also be used to automatically construct taxonomies.

Considerable work has also been done on algorithms dedicated to taxonomy generation. Kumar et al. [KGC02] propose a top-down approach (called

BHC) for construction of binary hierarchies of classes. The BHC approach recursively partitions a set of classes into two disjoint subsets of classes until only singleton sets of classes remain. The partitioning is achieved through a deterministic annealing process. Like BHC, Vural and Dy [VD04] and Punera et al. [PRG05] describe methods for creation of binary hierarchies in top-down fashion by successively splitting sets of classes using the k-Means algorithm. Tibshirani and Hastie [TH07] perform the same top-down construction using the optimal margin classifier at each split. In addition, Punera et al. [PRG05] consider splitting the classes themselves if their contents belong in multiple different parts of the taxonomy. Finally, Slonim and Tishby propose an agglomerative approach in [ST99] that produces binary trees by greedily merging clusters that minimize the loss in mutual information between the intermediate clustering and the category labels.

The methods mentioned so far are restricted to finding binary tree based taxonomies. There has also been some work on learning taxonomies that can be represented as n-ary trees [BGJT04, GR04, Hof99, SKO01, VL99].

2.4 Classification using Hierarchies

In this section we review past works that attempt to exploit hierarchical relationships to improve classification accuracy; this is the theme of our work in Chapters 3, 4, and 5. Some works explore scenarios where a set of classes are arranged in a hierarchical taxonomy, while others involve classifying instances that have a hierarchical relationships amongst themselves.

Classification with Hierarchical Taxonomies. Hierarchical classification systems typically create one multi-class classifier for each internal node

in the hierarchy, and classify a test instance once at each level so as to direct it to the classes at the leaves. This approach has been shown to have many advantages. Chakrabarti et al. [CDAR98] use hierarchical classifiers to segment large classification problems into more manageable units at the nodes of the hierarchy. Dumais and Chen [DC00] and Koller and Sahami [KS97] show that a smaller set of features suffices for each classifier when using well defined hierarchies. Finally, in the event of scarce labeled data, McCallum et al. [MRMN98] obtain more robust parameter estimates for classes via the parameter estimates of their parents in the hierarchy.

In a similar vein, hierarchies have also been used to decompose the output space of a multi-class problem into a series of binary problems that can be solved using popular binary classifiers like SVMs [KGC02, SKO01, VD04]. Also, there is some recent work on generalizing support vector learning to take into account relationships among classes mirrored in the class hierarchy [CH04, TH07, TJHA05].

Classification of Hierarchical Objects. There is a rich body of literature around classification of tree-structured objects such as semi-structured HTML or XML documents, website URL directories etc. Theobald et al. [TSW03] discuss classification of XML documents, using features that derive from the tree structure of the XML document. However, these features are extracted from simple types of path relationships, and are then processed by a traditional classifier. Diligenti et al. [DGMS01] consider classification of semi-structured documents using a "hidden tree Markov model", in which each subtree is generated by a particular Markov model. Piwowarski et al. [PDG02] and Denoyer and Gallinari [DG04b] consider a similar model in which document trees are modeled using Bayesian networks. Tian et al. [THG+03] model both the URL

tree structure within websites and the DOM tree structure within webpages using hidden tree Markov models to classify the pages in the website.

Hierarchical Probabilistic Models. Other researchers have employed hierarchical models, not to the problem of classifying hierarchical objects, but in order to consider flat objects at different levels of granularity. Hierarchical HMM, introduced by Fine et al. [FST98], is a hierarchical generalization of the widely used hidden Markov model. Hierarchical HMMs have been shown to be useful for unsupervised learning and modeling of complex multi-scale structures that occur in language, handwriting, and speech; in particular, they were used to construct hierarchical models of natural English text. Koller and Sahami [KS97] show a hierarchical generative model called tree-augmented naive Bayes. The goal is to capture local similarities in vocabulary within a document. Blei and Jordan [BJ03] consider the problem of modeling annotated data and derive hierarchical probabilistic mixture models to describe such data.

Exploiting Constraints from Object Relationships. There are many approaches to classification of objects where the outputs of classifiers are modified to conform to various types of constraints. Chakrabarti et al. [CDI98] present Hyperclass, a hyperlink-aware classifier operating upon webpages in a graph. In this work the label of a webpage is related to the labels of neighboring webpages via a technique based on Markov Blankets. In their research on natural language tasks, Roth and Yi [Rot02, RtY04] propose a *inference with classifiers* framework, which separates learning of classifiers and the maintenance of task-specific constraints. The constraints are enforced by formulating and solving integer programming problems. Agarwal et al. [ACA06] formulate and solve optimization problems to enforce user preference constraints while

ranking networked entities. Finally, in [BVZ01] Boykov et al. give a fast approximate algorithm for segmentation of entities related by general graphs. This algorithm also can be used to update the classification of a node with respect to labels of neighbouring nodes.

2.5 Isotonic and Unimodal Regression

In Chapters 4 and 5 we introduce regularized versions of the classic isotonic and unimodal regression problems. Algorithms for these classic problems have been used when the domain dictates that a function being estimated is constrained to be monotonically increasing, or maybe unimodal, but the data doesn't exhibit this behavior because of noise. Hence, in [MICAM02] isotonic regression is used to smooth the probability of heart attack as a function of cholesterol level, and in [SAM97] the credit-worthiness as a function of income. Other applications include epidemiology [MJDP$^+$00], microarray data analysis [AHKW06], and calibration of classifiers [ZE02].

A lot of research has concentrated on finding efficient algorithms for isotonic regression under different cost functions. For complete orders, Stout [Sto00] gives algorithms to find the optimal solutions for L_1 norm in $O(N \log N)$ time and for L_2 norm in $O(N)$ time. Boyarshinov and Magdon-Ismail [BMI06] give a linear time algorithm under the L_1 cost for cases where the number of output levels is bounded. For isotonic regression on rooted trees the best known algorithms work in $O(N \log N)$ time for L_2 [PX99] and $O(N^2 \log N)$ time for L_1 [AHKW06] metrics. Schell and Bahadur [SS97] propose a method for regularizing isotonic regression outputs by merging adjacent level-sets whose values are not significantly different.

Unimodal regression can be solved by multiple calls to isotonic regression

but this simple approach leads to expensive algorithms. Stout [Sto00] introduces prefix isotonic regression, whereby the regression on all initial segments is computed. The prefix approach utilizes the solution for one initial segment to aid in the solution of the next, which considerably reduces the total time required, giving $O(N \log N)$ and $O(N)$ time algorithms over complete orders for the L_1 and L_2 norms respectively. We do not know of any work that considers unimodal regression over trees.

2.6 Webpage Template Detection

Webpages contain a combination of unique content and *template material*, which is often present across multiple pages and used primarily for formatting, navigation, and branding. These template structures pollute the content by digressing from the main topic of discourse of the webpage. Furthermore, they can cripple the performance of many search engine modules, including the index, ranking function, summarization, duplicate detection, etc. Consequently, a lot of research work has focused on automatically detecting templates on webpages. Techniques proposed for the problem fall into two families. *Global* techniques consider a family of pages together (often from the same website) and exploit the property that templates occur many times. *Local* techniques, on the other hand, detect templates on an individual page using only information local to the webpage.

Site-level Template Detection. The problem of extracting templates from web pages was first introduced by Bar-Yossef and Rajagopalan [BYR02]. They propose a technique based on segmentation of the DOM tree and selection of certain key nodes using properties of the content of the node (such as

the number of links within the node) as candidate templates. Yi et al. [YLL03] and Yi and Liu [YL03] study template extraction in order to improve data mining results by removing noisy features due to templates. They present a data structure called the *style tree* which takes into account certain metadata about each node of the DOM tree, rather than the particular content of the node. Vieira et al. [VSP+06] frame the template detection problem as a problem of mapping identical nodes and subtrees in the DOM trees of two different webpages. They propose performing the expensive task of template detection on a small number of pages, and then removing all instances of these templates from the entire site using a much cheaper approach.

Page-level Template Detection. Some page-level algorithms have also been proposed recently. Kao et al. [KHC05] segment a given webpage using a greedy algorithm operating on features derived from the page. However, their method is not completely page-level; they use some site-level features such as the number of links between pages on a website. Debnath et al. [DMPG05] propose a page-level algorithm ("L-Extractor") that applies a classifier to DOM nodes, but only certain nodes are chosen for classification, based on a predefined set of tags. Kao et al. [KCLH02] propose a scheme based on information entropy to focus on the links and pages that are most information-rich, reducing the weights of template material as a by-product. Song et al. [SLWM04] use visual layout features of the webpage to segment it into blocks which are then judged on their salience and quality. Other local algorithms based on machine learning have been proposed to remove certain types of template material. Davison [Dav00] uses decision tree learning to detect and remove "nepotistic" links, and Kushmerick [Kus99] develops a browsing assistant that learns to automatically remove banner advertisements from pages. Finally,

some related research focuses on segmentation of webpages for the purpose of displaying them on small mobile device screens [Bal06, CXMZ05, YL04].

2.7 Topical Segmentation of Websites

In Chapter 6 we tackle the problem of segmenting a website into topically cohesive regions. We propose an algorithm that operates upon a tree-structured object with a class distribution at each node. The goal is to segment the tree structure into connected components that are cohesive in terms of the class distributions. While we have not been able to find previous work that directly studies such objects, we list below a number of related ones.

Site-level Classification. There has been some prior work on treating websites or groups of webpages as the basic unit of analysis [Pie00, THA99, TW04]. Kriegel et al. [KS04, EKS02] present various website classification schemes based on features extracted from the individual webpages. Some of their schemes consider topics at individual webpages but they use these as features for the site level classifier. They have no notion of segmenting a website into sub-parts, and they learn models for websites as a whole. More recent work by Tian et al. [THG+03] uses "hidden Markov trees" to model both the site directory trees as well as the DOM trees of the webpages. They then employ a two-phase system through a fine-to-coarse recursion to classify the site. Sun et al. [SL03] propose a technique to partition websites into "Web Units", which are collections of webpages. These fragments are created using heuristics based on intra-site linkages and the topical structure within the website is not considered.

Hierarchical Partitioning. Chakrabarti et al. [CJT01] propose a method

motivated from information theory for segmenting the DOM tree of a webpage into pagelets that are topically cohesive. Moving beyond the purview of machine learning, there are some approaches to partitioning hierarchical objects. Fagin et al. [FGK+05, FKK+05] consider a general notion of partitioning hierarchical structures based on a variety of different quality measures.

Our website segmentation work is closely related to the problem of facility location on trees. While the problem on general graphs is NP-hard [KH79], lot of work has been done to obtain fast exact algorithms for the facility location problem on trees with n nodes; the goal is to find k facilities. Tamir [Tam96] obtained a dynamic programming algorithm that runs in time $O(kn^2)$; this was an improvement over the $O(k^2n^2)$ algorithm of Kariv and Hakimi [KH79] and $O(kn^3)$ algorithm of Hsu [Hsu82]. For some important special case distance functions, the best running time bound of $O(n \log n)$ is due to Shah and Farach-Colton [SFC02].

Chapter 3

Automated Construction of Taxonomies

3.1 Introduction

In Chapter 1 we discussed the ubiquitous use of taxonomies as knowledge management tools. In additional, various information retrieval, data mining, and machine learning approaches make use of data arranged in hierarchies. Categories from taxonomies such as the Yahoo Web Directory are returned as search results for queries that map to them. Searchers can even provide context to their queries by searching documents within a certain category. The Scatter Gather system of Cutting et al. [CKT92, HKP95] makes use of hierarchical clustering techniques to provide an intuitive paradigm for presentation and exploration of retrieved results to users. Hierarchies have also been used to decompose the output space for the purposes of classification in diverse domains such as text mining [CDAR98, DC00, KS97], hyper-spectral analysis [KGC02], and image classification [HKZ98]. In machine learning, hierarchies of classes have been used for smoothing parameter estimates, as in

Shrinkage [MRMN98]. Hierarchical taxonomies are also a core component of ontologies for the Semantic Web [BLHL01], and their construction and maintenance of is the subject of much current research [DMDH02, FFR97, NM00].

Automatic Taxonomy Construction.

In this chapter we tackle the problem of arranging a given set of classes into a hierarchy, specifically as leaves of a **rooted n-ary tree**. Moreover, given the high cost and unscalable nature of manual intervention, we seek to do this completely automatically (parameter-free). Several approaches that seek to solve parts of this problem have been proposed. Kumar et al. [KGC02], Vural and Dy [VD04], and Punera et al. [PRG05] propose top-down approaches, while Slonim and Tishby [ST99] describe an agglomerative approach, for the construction of binary hierarchies of classes. Apart from the fact that these approaches perform greedy operations, the binary restriction on the branching factor of nodes creates artificial groupings of classes, especially at the top levels of the taxonomy. There has also been some work on learning n-ary tree structured hierarchies [GR04, SKO01] but these methods require significant user input on the structure of the tree. Specifying these parameter settings is very difficult without significant insight into the structure of the data. In contrast to these approaches, our algorithm is parameter-free and learns the structure of the taxonomy from the given data in an entirely automatic manner.

Our Contribution.

(1) In Section 3.2 we present an approach that constructs taxonomies of categories in a completely automated fashion. We introduce a novel constraint on the relationships between categories, and this helps our algorithm learn "good" taxonomies with no user-defined parameters.

(2) Our approach doesn't place any restrictions on the branching factor of the tree being learned, effectively constructing n-ary trees. This avoids the problem of arbitrary groupings of categories at the top levels of the tree.

(3) In our approach, some greedy decisions made early in the taxonomy construction process are re-evaluated in more specific contexts. This makes our approach significantly less greedy than some previous methods in literature that we review in Section 3.3.

(4) Through experiments (Section 3.4) on datasets from a variety of domains, we show that taxonomies modeled as n-ary trees are more "natural" and result in better hierarchical classification accuracies than those modeled as binary trees. To the best of our knowledge this is the first study of this kind.

3.2 Automatic Taxonomy Generator (ATG)

We begin with a detailed description of the problem of automatic taxonomy construction and then propose our solution to it. Henceforth in this chapter, we use the terms "class" and "category" interchangeably. Moreover, since we model taxonomies as hierarchies, and as trees to be specific, we use the terms "taxonomy", "hierarchy", and "tree" interchangeably too.

3.2.1 The Automatic Taxonomy Construction Problem

We broadly define the problem of automatic taxonomy construction as finding an arrangement of classes in a hierarchy. In this chapter we tackle the problem of learning the structure of a rooted n-ary tree with the classes placed at the leaves. The desiderata of a solution are as follows:

1. "Similar" classes should be placed close to each other in the learned taxonomy. Since our taxonomy is a rooted tree, similar classes should be closer to each other than different ones in terms of, say, the number of undirected edges in the path connecting them. For example, in the creation of a Shopping taxonomy, the lowest common ancestor of classes Car and SUV should be farther from the root than that of classes Car and Power-Tool. This is a standard requirement of any taxonomy employed for classification.

2. The internal nodes should have an interpretation based on their content. In other words, the content of each internal node should be as homogeneous as possible. Sometimes, taxonomies organize classes by grouping them arbitrarily. This often happens at the top levels of taxonomies which are modeled as binary trees. For example, consider a Shopping taxonomy where the root is associated with the set of classes {Electronics, Computers, Home & Garden, Clothing & Accessories}. These classes are all very different from each other and should all be placed in separate nodes at the next level. But if the taxonomy is modeled as a binary tree it might be forced to partition the set of classes into {Electronics,Computers} and {Home & Garden, Clothing & Accessories}. We desire that the taxonomy construction process partition the set of classes at each internal node into as many parts as needed to maintain the homogeneity of the children nodes.

3. The taxonomy creation approach should work automatically without any user-defined parameters. Many existing approaches require the user to specify, for instance, the number of internal nodes in the tree or at each

level. These parameters are very difficult to set manually without intimate knowledge of the structure of data. We want our approach to avoid any such parameters.

NOTATION.

We need a few definitions to make the ideas expressed above and the subsequent solution to the problem precise. Let X be the set of data-points, such that each data-point x_i has an associated class label l_i from a set of k classes C. Thus, each class c_j has a set of data-points X_{c_j} associated with it using which its prior π_{c_j} and class-conditional probability density functions $p_{c_j} = p_X(x|c_j)$ can be estimated.

We want an arrangement of the classes C into a taxonomy. Let the taxonomy be represented by a rooted n-ary tree T with k leaves. Let leaf(T) and root(T) represent the set of leaves and the root of the tree T respectively. Each class is placed at exactly one leaf of T so that leaf(T) = C. Let w be an internal node of T, and let T_w denote the subtree rooted at w. Each such internal node w is then associated with a set of classes $C_w = \text{leaf}(T_w)$. Let X_{C_w} represent the data obtained by putting together the data belonging to all classes in C_w. Using X_{C_w}, each set of classes C_w, and thereby each internal node w, has an associated prior $\pi(C_w)$ and a probability density function p_{C_w}. Note that more sophisticated models for p_{C_w} can be used, such as a mixture model of pdfs associated with classes in C_w. Finally, we note that while describing our approach a collection of sets of classes such as $\{C_{v_i} : 1 \leq i \leq m\}$ is sometimes shortened to $\{C_{v_i}\}_{i=1}^{m}$.

3.2.2 Proposed Solution (ATG)

In this section we first describe a generic algorithm for the construction of a hierarchy and then justify the design choices made in our work.

A Generic Top-down Algorithm.

We adopt a top-down approach to learning the tree structure. We start with T as a single node. Then root(T) is associated with the set of given classes C and the variable root(T).tosplit is set to $true$. At any time during the algorithm's run there are a set of leaves of the tree T that have their tosplit variable set to $true$. We pick one such leaf w for splitting. Let w be associated with the set of classes C_w. We then need to find the m disjoint subsets $\{C_{v_i}\}_{i=1}^{m}$ into which C_w must be partitioned. This involves both finding the value of m and the subsets themselves. This partitioning is computed by a procedure called `findPartition`, which is described later in this section. Once the C_{v_i} are obtained, we create m new nodes v_i that are assigned as immediate children of w. Each C_{v_i} is then associated with the corresponding leaf v_i. The tosplit variable of each v_i whose associated C_{v_i} has more than one class is set to $true$; w.tosplit is set to $false$. In this fashion we proceed with splitting leaves until all leaves have tosplit set to $false$. In other words, internal nodes are split until the leaves have only one class each. The pseudo-code for this algorithm is shown in Figure 3.1.

The Partitioning Criterion.

As mentioned above, at each internal node w of the tree we need to find a partitioning of the set of classes C_w into an appropriate number of subsets. But before we describe our choice of partitioning criterion we need a notion of distance between sets of classes.

We compute the distance between sets of classes using the *Jensen-Shannon (JS) divergence* [Lin91]. The distance between two sets of classes C_1 and C_2 is defined in terms of their associated pdfs p_{C_1} and p_{C_2}, and priors $\pi_i = \pi(C_i)$

$$\mathrm{JS}_\pi(\{C_1, C_2\}) = \pi_1 \mathrm{KL}(p_{C_1}, \pi_1 p_{C_1} + \pi_2 p_{C_2}) + \pi_2 \mathrm{KL}(p_{C_2}, \pi_1 p_{C_1} + \pi_2 p_{C_2})$$

where $\pi_1 + \pi_2 = 1$, $\pi_i \geq 0$, and KL is the Kullback-Leibler divergence [KL51]. The JS divergence measures how "far" the classes are from their weighted combination, where the π_i assign the contribution of the two distributions. The JS measure is always non-negative, symmetric in its arguments, and, unlike the KL divergence, is bounded. Moreover, it can be generalized to more than 2 sets of classes/distributions. The JS divergence between k sets of classes C_i is defined as

$$\mathrm{JS}_\pi(\{C_i : 1 \leq i \leq k\}) = \sum_{i=1}^{k} \pi_i \mathrm{KL}(p_i, p_m) \tag{3.1}$$

where $\sum_i \pi_i = 1, \pi_i \geq 0$, and p_m is the weighted mean probability distribution $p_m = \sum_i \pi_i p_i$ [DMK02]. In this work we use $\mathrm{JS}(\{c_j : c_j \in C_{v_i}\})$ to refer to the JS divergence between the set of distributions p_{c_j}; and $\mathrm{JS}(\{C_{v_i}, C_{v_j}\})$ to refer to the JS divergence between the distributions $p_{C_{v_i}}$ and $p_{C_{v_j}}$, though they are sets of classes.

Using this definition of distance we can define a criterion of partitioning C_w. We would like to partition C_w into m disjoint subsets $\{C_{v_i} : 1 \leq i \leq m\}$ so as to minimize

$$\sum_{i=1}^{m} \pi(C_{v_i}) \mathrm{JS}_{\pi'}(\{c_j : c_j \in C_{v_i}\}) \tag{3.2}$$

where $\pi(C_{v_i}) = \sum_{c_j \in C_{v_i}} \pi_{c_j}$, and $\pi'_{c_j} = \pi_{c_j}/\pi(C_{v_i})$, under the constraint that $\forall i, j \neq i$

$$\mathrm{JS}_{\pi''}\left(\{C_{v_i}, C_{v_j}\}\right) > min\{\mathrm{JS}_{\pi''}\left(\{C_{v_i}, C_w\}\right), \mathrm{JS}_{\pi''}\left(\{C_{v_j}, C_w\}\right)\} \tag{3.3}$$

where $\pi'' = \{\frac{1}{2}, \frac{1}{2}\}$.

The objective function in Equation (3.2) computes the similarity of all the classes to the subset that they end up in. Minimizing this function gives us subsets that are very homogeneous. We would like to minimize this objective function over all possible m sized partitionings of C_w, where m ranges from $2...\|C_w\|$. Since the function in Equation 3.2 is trivially minimized if C_w is partitioned into $\|C_w\|$ singleton subsets, we need to constrain the solution.

The constraint in Equation (3.3) ensures that none of the m subsets are closer to each other than to the parent C_w. In other words any solution in which there exist at least one pair of subsets that are closer to each other than to the parent is considered invalid. This constraint enforces a distance between sibling nodes, and is natural in the context of a taxonomy. If two sets of classes C_{v_1} and C_{v_2} are closer to each other than each is to parent C_w, then it can be argued that they should be placed in the same subset, and be separated lower in the tree. Setting uniform priors π'' in Equation (3.3) gives equal importance to all distributions, and prevents larger classes from biasing the mean distribution towards themselves.

An attractive feature of this constraint is that the threshold on the distance between subsets is defined by the distances of the subsets from the parent set, and from each other. Hence, a solution with C_{v_1} and C_{v_2} very close to each other will still be considered valid if either one of them is still closer to C_w. On the other hand, another solution in which C_{v_3} and C_{v_4} are far from each other might not be considered valid if both are even further away from the parent C_w. Since the partitioning criterion depends on the distance relationships between classes, the structure of the taxonomy is learned automatically and no parameters need to be set by the user.

As mentioned above, we want to minimize the objective in Equation (3.2) over all possible partitionings of C_w into m (where $2 \leq m \leq \|C_w\|$) subsets that satisfy the constraint in Equation (3.3). The optimal solution can be obtained by enumerating all possible solutions which satisfies the constraint, and picking the one which minimizes the objective. The time complexity of this procedure will be exponential in the number of classes in the parent. Hence, we need an algorithm that computes a "good" solution efficiently at the expense of optimality guarantees.

A Greedy Algorithm to Find Partitionings.

In order to find a solution efficiently we devise a greedy agglomerative approach. This is implemented as the procedure `findPartition` in the pseudo-code in Figure 3.1.

Let the current node w being partitioned have a set of n classes C_w associated with it. We seek to find the partitioning by agglomeratively clustering the set of classes. We begin with each class as a separate cluster, $\{C_{v_i}\}_{i=1}^{n}$. We then obtain pair-wise distances (as defined by the JS divergence) between each pair of clusters, and also between each cluster and C_w. We define a *candidate-pair* for merging as a pair of clusters C_{v_i} and C_{v_j}, such that they violate the constraint in Equation (3.3). From all such candidate-pairs we pick the one that has the smallest value for $(\pi(C_{v_i}) + \pi(C_{v_j}))\mathrm{JS}_\pi(\{C_{v_i}, C_{v_j}\})$ and merge its constituents. The process of merging clusters C_{v_i} and C_{v_j} involves replacing them by another cluster C_{v_k} that includes both their classes ($C_{v_k} = C_{v_i} \cup C_{v_j}$), and then recalculating the pair-wise distances and finding the new set of candidate-pairs. We repeat this process of merging until no candidate-pairs remain. The final set of clusters (each a set of classes) define the partitioning of C_w.

Algorithm `constructTaxonomy`
Input: C is the set of all classes
p_{cj} are class-conditional density functions
Output: T is the rooted n-ary tree with leaf$(T) = C$
1. Initialize T as a single node. Set root(T).classes $= C$
and root(T).tosplit $= true$
2. `while` (w.tosplit $== true$), for some node w
3. $\{C_{v_i}\}_{i=1}^{m}$ = `findPartition`(w.classes)
4. Create m new nodes v_i, set v_i.tosplit $= false$
5. `for-each` v_i
6. set v_i as a child of w
7. v_i.classes $= C_{v_i}$
8. `if` ($|C_{v_i}| > 1$) `then` set v_i.tosplit $= true$
9. `end-for`
10. `end-while`

Algorithm `findPartition`
Input: C_w is the set of n classes to partition
Output: $\{C_{v_i}\}_{i=1}^{m}$ form the partition of C
1. Let each class in C_w be a cluster $\{C_{v_i}\}_{i=1}^{n}$
2. Get JS divergence among all pairs from C_{v_i},
and also between each C_{v_i} and C_w
3. Find all pairs $P_k = (C_{v_i}, C_{v_j})$ that violate the
constraint in Equation (3.3)
4. From P, select the pair (C_{v_i}, C_{v_j}) which has the
lowest value for the expression in Equation (3.4).
5. `while` (there exists a pair (C_{v_i}, C_{v_j}))
6. Replace C_{v_i} and C_{v_j} with $C_{v_k} = C_{v_i} \cup C_{v_j}$
7. Recompute pairwise JS divergence as in step 2
8. Pick the pair (C_{v_i}, C_{v_j}) as in step 3 and 4
9. `end-while`

Figure 3.1: Pseudo-code for our proposed approach for automatic construction of a taxonomy (ATG).

In our algorithm, we start with a presumably *invalid* solution with the lowest possible objective function value; each class forms a singleton cluster $\{C_{v_i}\}_{i=1}^{n}$. We then successively merge clusters (and incur an increase in objective function value) until a valid solution is obtained. The pair of clusters for merging are chosen from the set of clusters that violate the constraint, in such a way so as to minimize the increase in objective function value. After a valid solution has been obtained we stop merging since further merging cannot improve the value of the objective function.

Proposition 3.1. *At any step in the* `findPartition` *algorithm, merging the candidate-pair* (C_{v_i}, C_{v_j}) *with lowest value of* $\big(\pi(C_{v_i}) + \pi(C_{v_j})\big)\,\mathrm{JS}_{\pi}\left(\{C_{v_i}, C_{v_j}\}\right)$ *results in the least increase in the objective function value*

Corollary 3.2. *Merging clusters will always result in an increase in the value of the objective function.*

Proposition 3.1 can be proved by noting that the change in objective value due to merging sets C_{v_i} and C_{v_j} is

$$\begin{aligned}\delta &= \big(\pi(C_{v_i}) + \pi(C_{v_j})\big)\,\mathrm{JS}_{\pi'}\left(\{c : c \in C_{v_i} \cup C_{v_j}\}\right) \\ &\quad - \pi\left(C_{v_i}\right)\mathrm{JS}_{\pi'}\left(\{c : c \in C_{v_i}\}\right) \\ &\quad - \pi\left(C_{v_j}\right)\mathrm{JS}_{\pi'}\left(\{c : c \in C_{v_j}\}\right) \\ &= (\pi(C_{v_i}) + \pi(C_{v_j}))\mathrm{JS}_{\pi}(\{C_{v_i}, C_{v_j}\}) \qquad (3.4)\end{aligned}$$

where $\pi' = \pi_c/\pi(C_v)$ are the class priors normalized within each cluster. Equation (3.4) can be obtained by using Theorem 4 in [DMK02]. Then Corollary 3.2 follows from the non-negativity of Jensen-Shannon divergence.

Proposition 3.3. *The* `findPartition` *algorithm in Figure 3.1 will terminate with at least two clusters.*

This proposition states that when only two clusters are left, each cluster will be closer to the parent than to the other cluster. In other words, a solution with only two clusters is always valid. This follows from the fact that the Jensen-Shannon divergence $\mathrm{JS}_\pi(\{p_i\})$ is convex in p_i for a fixed π. This property of our algorithm ensures that the procedure `constructTaxonomy` in Figure 3.1 always terminates by outputting a tree with classes placed at the leaves.

The merging process in our algorithm is greedy and no guarantee can be given that the objective function will be optimally minimized. However, we note that some of these merges will be re-evaluated when children nodes are further partitioned lower in the tree. The parent node during these new partitions will be different and more specific. We claim that this ameliorates some of the effects of greedy merges and helps our algorithm find better hierarchies.

3.3 Comparison with Related Work

In this section we briefly review existing research on creation of taxonomies and compare it to our work. We concentrate on those aspects of existing algorithms that distinguish our approach from them. Readers are referred to the original publications for details of the works mentioned.

Comparison with Agglomerative Information Bottleneck.

Agglomerative Information Bottleneck (AIB) was proposed by Slonim and Tishby in [ST99] where it was used to hierarchically cluster words in a given dataset. However, this technique can also be applied to classes in order to construct a hierarchical structure over them. The method is initialized with each class as a separate cluster. The algorithm then produces a binary tree

by greedily merging clusters that minimize the loss in mutual information between the intermediate clustering and the category labels. In this respect, this method resembles the way we partition the set of classes at each node in function `findPartition` in Figure 3.1. We refer the readers to the original paper by Slonim and Tishby [ST99] for more details about the AIB algorithm.

In spite of having the same cost function as our approach (ATG), AIB differs in two significant ways. First, while ATG yields n-ary tree based taxonomies, AIB only constructs binary trees. Second, ATG constructs the taxonomy in a top-down fashion while AIB is an agglomerative procedure. Essentially, our approach performs operations partially resembling AIB (the `findPartition` function) in order to partition the classes at each internal node. This top-down nature lets ATG reconsider some of the merge decisions made by the `findPartition` procedure. AIB does not enjoy this benefit making it more greedy than ATG.

In the next section, we will empirically compare the n-ary taxonomies generated by ATG with the binary taxonomies generated by AIB. Since both approaches use Jensen Shannon divergence based cost functions, this is an objective evaluation of whether n-ary tree based taxonomies are better than binary tree based ones. We will show that n-ary taxonomies are more natural than binary taxonomies, and classifiers learned over them perform better. Moreover, we will show that ATG makes less greedy merges than AIB.

Comparison with Other Existing Research.

The classic agglomerative clustering method (HAC) [JD88] can be easily adapted for construction of a taxonomy in much the same way as the AIB algorithm above. However, it suffers from the same shortcomings as AIB.

Tishby et al. [TPB99] present a method for obtaining a hierarchical clustering based on the Information Bottleneck principal that finds clusters using a deterministic annealing like procedure. The "detection" of clusters in this process is, however, a very subtle task, and requires tuning of parameters which is non-trivial without prior knowledge about the data [Slo03]. Furthermore, this taxonomy creation approach is non-deterministic in the solution it produces. By contrast, our approach is parameter-free and completely deterministic in nature. Other approaches that employ deterministic annealing to create hierarchies are given in [GGPC02, KGC02, Hof99].

Kumar et al. [KGC02] and Tibshirani and Hastie [TH07] propose top-down approaches - called BHC and Margin Trees respectively - for the construction of binary hierarchies of classes specifically with hierarchical classification as the application. Both approaches recursively partition a set of classes into two disjoint subsets until only singleton sets of classes remain. The partitioning is achieved through a deterministic annealing process in BHC and through maximum margin methods in Margin Trees. Similarly, Vural and Dy [VD04] and Punera et al. [PRG05] describe methods for creation of binary hierarchies in top-down fashion by successively splitting nodes using the k-Means algorithm. All these approaches, however, restrict the taxonomy structure to only binary trees. This results in many arbitrary groupings of classes, especially at top levels of the hierarchy, where the approaches are constrained to place all classes in one of two children. By modeling the taxonomy as a n-ary tree our approach avoids the occurrence of arbitrary groupings of classes. We showcase this property of our algorithm later in Section 3.4.3.

There has been some work on learning taxonomies that can be represented as n-ary trees [BGJT04, GR04, Hof99, SKO01, VL99]. But all these

algorithms require the user to specify parameters that define the structure of the taxonomy to be learned. For instance, in the Cluster-Abstraction Model of Hofmann et al. [Hof99] the structure of hierarchy is fixed and provided to the algorithm, in Probabilistic Abstraction Hierarchies of Segal et al. [SKO01] and "nested Chinese restaurant process" by Blei et al. [BGJT04] the number of internal nodes is a user-defined parameter, while [VL99] and [GR04] are agglomerative approaches where the user has to specify the number of nodes for each new level. This type of information is not easy to provide without significant domain knowledge about the structure of the data. In contrast to these approaches, our algorithm is parameter-free and learns the structure of the taxonomy from the given data in an entirely automatic manner. We claim that this is an extremely desirable characteristic, and along with the ability of arrange topics as nodes of a n-ary tree sets our approach apart from existing approaches in literature.

3.4 Experiments

In this section we report on our empirical comparison of taxonomies generated by our approach (ATG) and Agglomerative Information Bottleneck (AIB) [Slo03]. First we describe the datasets as well as the experimental setup. Next, we show that the n-ary tree based taxonomies created by ATG group classes in more natural ways than the binary tree based taxonomies generated by AIB. This part of our evaluation involves a subjective comparison of the taxonomies generated by the two methods. Finally, for a more objective analysis we report on our experiments with using these taxonomies to learn hierarchical classifiers.

3.4.1 Datasets

We experiment on standard datasets in which classes are related to each other by a possible hierarchical structure. The datasets used in this chapter are available from `http://www.ideal.ece.utexas.edu/~kunal/thesis/`.

20-NEWSGROUPS. This standard text dataset[1] [Lan95] consists of around 1000 documents from each of 20 different newsgroups. While the human defined names suggest a hierarchical organization of the newsgroups, some of the newsgroups such as "talk.religion.misc" and "soc.religion.christianity" have many cross-postings and share similar vocabularies. On the other hand, newsgroups such as "sci.crypt" and "sci.space" both fall under the science sub-category but have very different vocabularies and no cross-postings. The existence of such relationships between the newsgroups makes this dataset an ideal candidate for testing the different content-based hierarchy generators. The 20-newsgroups dataset has been extensively used for evaluating text categorization techniques (see, for example, [RR01, RK04]).

REMOTE SENSING DATASETS. Remote sensing data is a hyper-spectral image wherein each pixel has a spectral signature associated with it. Each pixel is assigned a class, which typically refers to a geographical feature, like forest, grassland, etc. Similarity between the spectral signatures of the different classes enables one to define inter-class relationships, and subsequently hierarchies of classes. In this chapter we use remote sensing data [2] obtained from two sites, the NASA's John F. Kennedy Space Center (KSC) [Mor02], Florida and the Okavango Delta, Botswana [HCCG05]. The KSC dataset has

[1] `http://people.csail.mit.edu/jrennie/20Newsgroups/`
[2] `http://www.csr.utexas.edu/hyperspectral/codes.html`

10 landcover types which can be broadly classified into the upland and wetland classes. Classes 1, 3, 4, 5 and 6 are all trees that grow in the uplands. Classes 2 and 7 are also trees that grow in heavily inundated soil. Class 8 is a type of marsh grass and Class 9 is the transitional area between land and water. The Botswana dataset has 14 different land cover types consisting of seasonal swamps, occasional swamps, and drier woodlands located in the distal portion of the delta. In this dataset (Figure 3.4(a)), Classes 3 and 4 are grasslands that grow in regions that can get flooded. Classes 5 and 6 are vegetation found along streams. Class 7 represents burnt vegetation. Classes 10, 11, 12, and 13 represent grasslands with mopane and acacia tree. The rest of the class labels are self-explanatory.

Glass. The instances in this dataset – samples of glass used for different purposes – are described by real-valued features corresponding to chemical and optical properties. This dataset has a well-defined hierarchy associated with it. The 6 different glass types are broadly categorized into window and non-window glass at the highest level of granularity. The window glasses are then subdivided into the float processed and the non-float processed categories. It would be interesting to see if our approach can retrieve this hierarchical structure.

Pendigits. The Pendigits dataset consists of 250 handwriting samples from 44 writers. The handwriting samples were collected using a pressure sensitive tablet which sent the (x, y) co-ordinates of the pen as inputs at fixed time intervals. Therefore, the similarity between classes in this dataset is defined not as much by the shape of the numbers but by how they are typically written. This is a particularly difficult dataset to define inter-class relations on as writing styles and speeds vary widely among subjects. However, visualizing the

average digit for each class [DMS99] is helpful in interpreting the taxonomies obtained by our approach.

VOWEL. In the Vowel dataset the different classes correspond to 11 different vowel sounds. Each word corresponding to a vowel sound was uttered by 15 different speakers. It would be interesting in this case to see if similar sounding vowel classes actually end up closer to each other in the generated tree.

3.4.2 Implementation Details

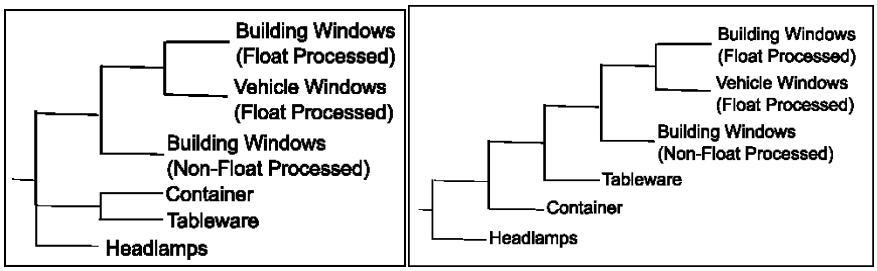
The 20-newsgroups dataset was preprocessed to remove posting headers, stop words and words that occur less than 5 times leaving us with a vocabulary of 50736 words. While partitioning an internal node, a vocabulary specific to that internal node was generated using the Fisher index criterion [CDAR98]. The class-conditional pdfs at the leaf and the internal nodes were then estimated by assuming an independent, multinomial distribution of the words.

For real-valued datasets, a multi-variate Gaussian distribution was used to model the class-conditional distributions of each of the nodes in the taxonomy. A node-specific feature space was created prior to partitioning the classes at that node by using the Fisher discriminant. Since the Fisher discriminant technique yields a feature space that maximizes the discrimination between the classes, one could interpret the closeness of the classes projected in this space as a very strong indicator of the inter-class similarity. For the remote sensing data, since there is a high degree of correlation between the different features we made use of a domain-specific feature reduction technique, called the best-bases algorithm [KGC01], prior to applying the Fisher discriminant.

3.4.3 N-ary Taxonomies are More Natural

In this section, we discuss the taxonomies returned by the two approaches on the different datasets. While analyzing the generated taxonomies, one has to keep in mind the fact that both the ATG and AIB methods generate 'content-based' hierarchies as opposed to 'concept-based' ones. A content-based hierarchy generator allows the data to guide the taxonomy generation process, and requires minimal human intervention, whereas the 'concept-based' hierarchies require an understanding of the underlying data at a more abstract level. For instance, in the 20-newsgroups dataset, a concept-based hierarchy would have grouped the two science classes (space and crypt) together whereas the content of the classes themselves suggest otherwise. Hence our approaches separate the science classes fairly early in the taxonomy.

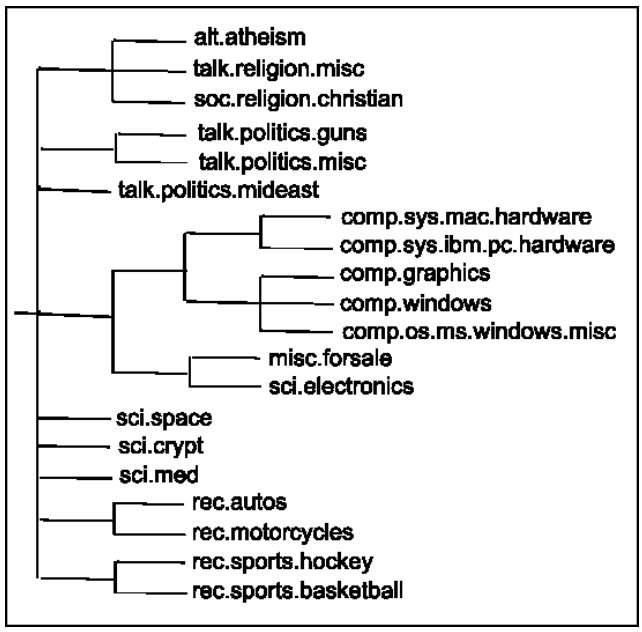
From the trees in Figures 3.3(a), 3.4(a), 3.2(a) and 3.5, one can see that the ATG method yields n-ary taxonomies that reflect the underlying class affinities well. In the case of the 20-newsgroups data, the first level of the ATG taxonomy (Figure 3.3) shows the different clusters of classes that exist in the dataset. The taxonomies constructed for the rest of the datasets also clearly reflect the number of clusters in the classes and the inter-class relationships between the different classes. For the taxonomy generated for the Pendigits dataset (Figure 3.5(b)), one has to look at the average digits representation, as illustrated in [DMS99], to interpret the hierarchy better. Note that the average figures have shown the extrapolated trajectories between the pen locations at different time intervals. A careful observation of the scaled average digits reveals that the pen locations for the clusters identified by the ATG are similar. Finally, for the Glass dataset, we recover the exact hierarchical structure specified in the UCI-ML description of the dataset (Figure 3.2(a)).



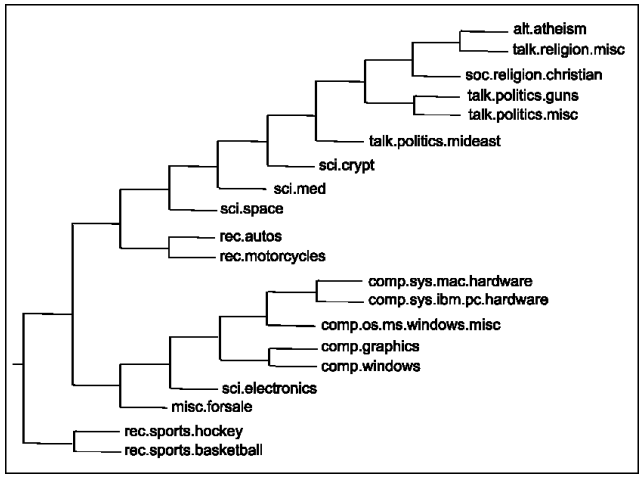
(a) Dataset: Glass, Algorithm: ATG (b) Dataset: Glass, Algorithm: AIB

Figure 3.2: Taxonomies constructed for the Glass dataset by the ATG and AIB algorithms. ATG recovers the exact hierarchical structure specified in the UCI-ML description of the Glass dataset.

While the taxonomies generated by the AIB (Figures 3.3(b) and 3.4(b)) also eventually group similar classes together, the meta-classes generated higher up in the tree are a mix of fairly well-separated classes, such as the "autos" and "motorcycles" group with that of "politics" and "science". This behavior is all the more striking for the Botswana dataset (Figure 3.4(b)) as it has a wider mix of landcover types. The greedy nature of the AIB approach ensures that merge decisions that are once made cannot be revisited, whereas in the ATG technique reevaluating the similarities in a node-specific feature space better reveals the inter-class affinities. For instance, in the 20-newsgroups dataset while both the ATG and the AIB technique correctly identify the Electronics/Computer meta-class, the ATG technique by virtue of reconsidering this cluster in a more specialized space is able to correctly group the "comp.os.ms.windows.misc" with the "comp.windows" and "comp.graphics" classes, unlike the AIB that clumps the "comp.os.ms.windows.misc" class with the hardware classes during the initial stages of hierarchy creation. Similar observations can be drawn from the taxonomies generated for the remaining datasets.

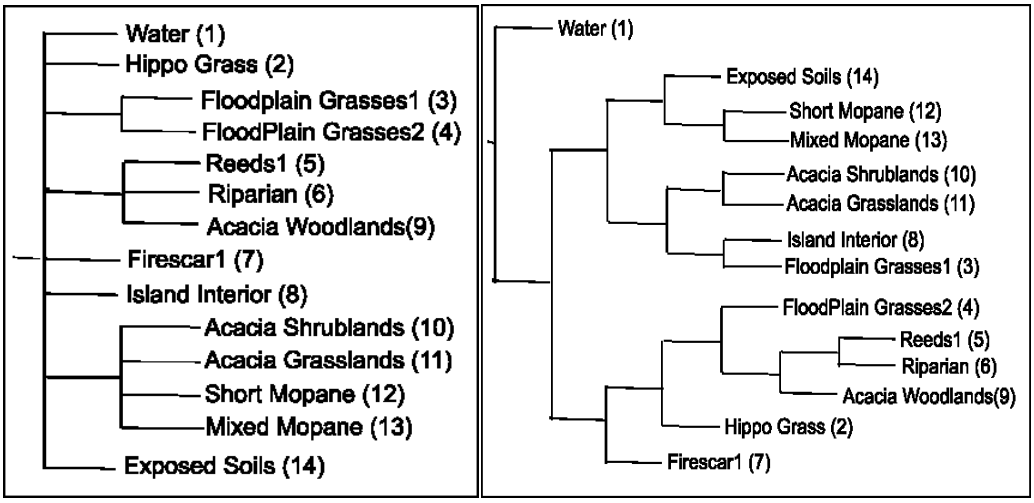


(a) Dataset: 20-newsgroups, Algorithm: ATG



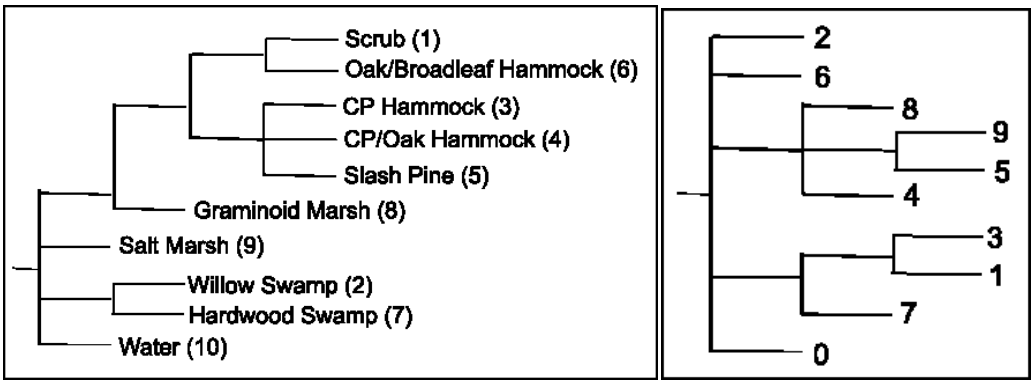
(b) Dataset: 20-newsgroups, Algorithm: AIB

Figure 3.3: Taxonomies constructed for the 20-newsgroups dataset by the ATG and AIB algorithms.



(a) Dataset: Botswana, Algorithm: ATG (b) Dataset: Botswana, Algorithm: AIB

Figure 3.4: Taxonomies constructed for the Botswana dataset by the ATG and AIB algorithms.



(a) Dataset: KSC, Algorithm: ATG (b) Dataset: Pendigits, Algorithm: ATG

Figure 3.5: Taxonomies constructed for the KSC and Pendigits datasets by the ATG algorithm.

3.4.4 Comparison over Classification Accuracy

In this section we evaluate the utility of taxonomies generated by ATG (our approach) and AIB [Slo03] for classification.

We used the taxonomies generated by each algorithm to build hierarchical classifiers. Experiments were performed using one-vs-all SVM classifiers [RR01] at each level of the hierarchy. Linear kernel SVMs and Gaussian kernel SVMs were used for text and numeric data respectively. 5% of the training data was used as the validation set to tune both the upper bound on the support vector coefficients (varied over 0.125, 0.25, 0.5, 1, and 2) as well as the kernel width of the Gaussians (varied over 0.0625, 0.125, 0.25, 0.5, 1, and 2) for the numeric datasets.

The SVMs were trained using the original feature space as preliminary experiments showed that feature selection did not improve SVM classification accuracies. However, a useful property of learning hierarchical classifiers is that of exploiting feature spaces that are specialized to each sub-problem. Hence, experiments were also performed using a Bayesian classifier at the internal nodes. The Fisher discriminant as detailed in Section 3.3 was used to reduce the dimensionality of the input space prior to classification. The data was then projected into the feature space associated with that node prior to classification. In the case of text data we made the usual independence assumption, whereas for the real-valued datasets we used the full-covariance matrix.

For each dataset, 80% of the data was used as the training set and the remaining 20% was used as the test set. All the classification accuracies reported here were obtained by averaging the results over five different samplings of the

training and test set. Two sets of experiments were performed as detailed below.

In the first set of experiments, the entire training set was used to first construct the taxonomy. Once the ATG and the AIB trees were obtained, fractions of the training data (5%, 10%, 20%, 40%, 60%, and 80%) were used to obtain both the new node-specific feature spaces as well as to train the internal SVM or Bayesian classifiers. It was expected that since the n-ary splits are more natural than binary, classifiers built on those hierarchies should have better classification accuracies than similar classifiers built on binary trees.

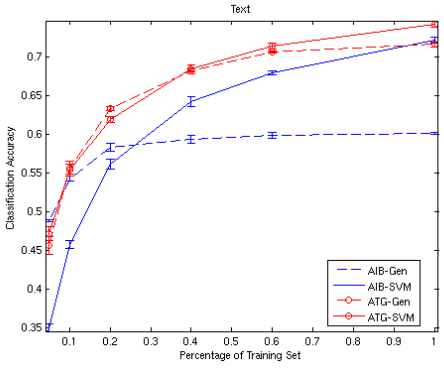
The learning rates for the different datasets are shown in Figure 3.6. The superior classification accuracies of the ATG-Bayesian classifiers, under the limited data conditions, validate our belief about the utility of using a more "natural" tree to learn a hierarchical classifier. Figure 3.6(a) shows that for text datasets like 20-newsgroups, even "powerful" classifiers such as SVMs benefit from the n-ary splits in terms of the classification accuracies. One might also expect that the easier decision boundaries of the n-ary trees speeds up the training times for SVMs. Similar results were obtained for the remaining datasets.

While in the previous set of experiments the hierarchy was constructed using the entire training set, we also investigated the effect of limited data on hierarchy creation and the classification accuracy of the resulting classifiers. Fractions of the training data (5%, 10%, 20%, 40%, 60%, and 80%) were used to construct the AIB and ATG hierarchies, the corresponding feature spaces, and train the internal node classifiers. The results of these experiments are shown in Figure 3.7. It can be seen that for all datasets using the ATG hierar-

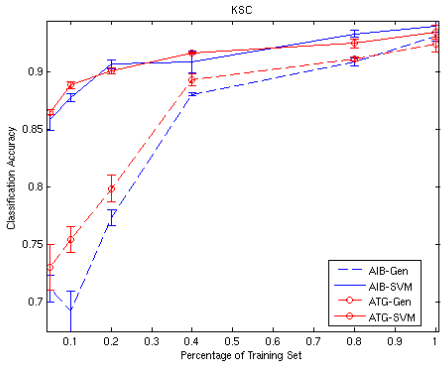
chy with internal Bayesian classifiers outperforms the AIB based hierarchical classifiers. Using SVM-based ATG classifiers offers comparable, if not better, classification accuracies than using SVMs with the AIB hierarchy. The results show that the proposed ATG method can not only be used to generate meaningful hierarchies, but can also be used as an alternative classifier especially for low data conditions. Note that while the hierarchies are used to obtain an output space decomposition we did not take advantage of parent-child relationships in the hierarchy. Evaluating the different methods while using shrinkage techniques to estimate class parameter estimates of child-nodes is part of our future work.

3.5 Conclusions

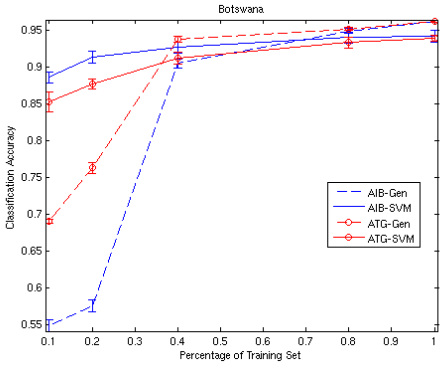
In this chapter, we presented a framework to learn hierarchies, modeled as rooted n-ary trees over a set of categories, in a completely automated manner. Our experimental evaluation over multiple datasets, from diverse domains, showed that our approach produces more "natural" taxonomies than the binary taxonomies outputted by Agglomerative Information Bottleneck [ST99]. Finally, we also showed that hierarchical classification using the taxonomies modeled as n-ary trees learned by our approach resulted in higher accuracies than taxonomies modeled as binary trees, especially under limited data conditions.



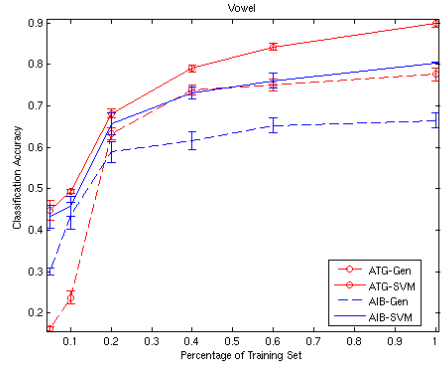
(a) Dataset: 20-newsgroups



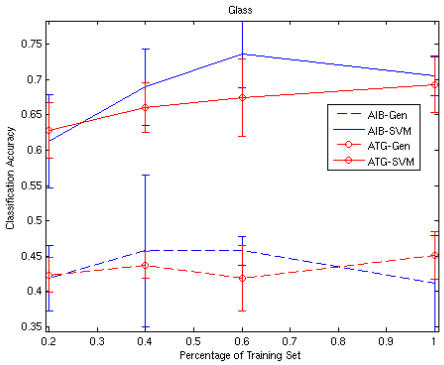
(b) Dataset: KSC



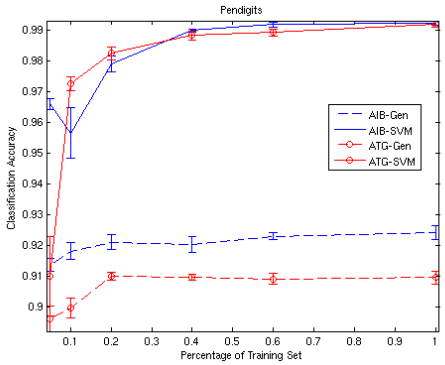
(c) Dataset: Botswana



(d) Dataset: Vowel

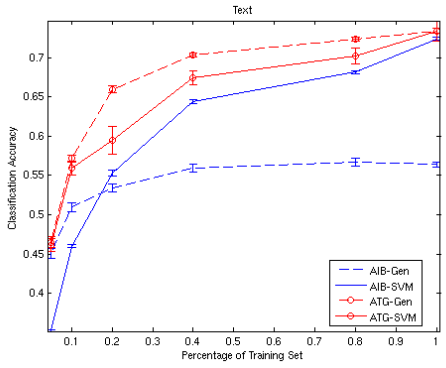


(e) Dataset: Glass

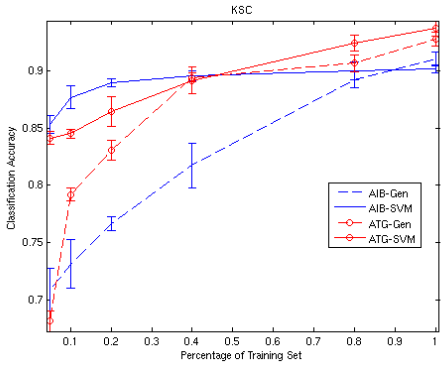


(f) Dataset: Pendigits

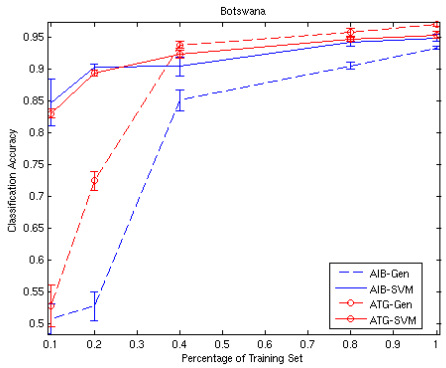
Figure 3.6: Learning rates when the taxonomies are pre-constructed.



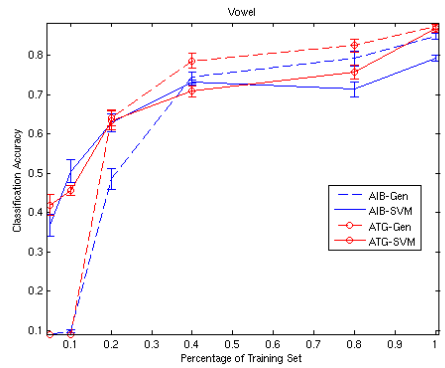
(a) Dataset: 20-newsgroups



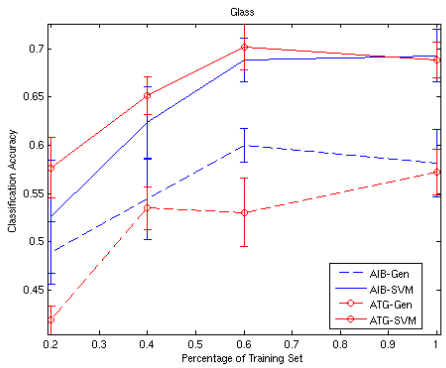
(b) Dataset: KSC



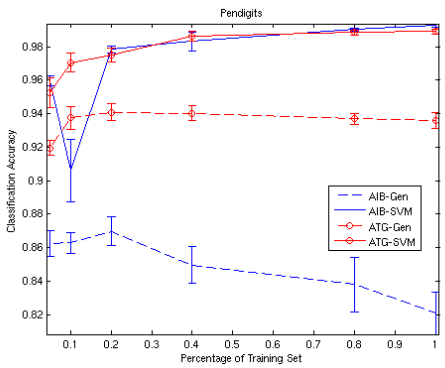
(c) Dataset: Botswana



(d) Dataset: Vowel



(e) Dataset: Glass



(f) Dataset: Pendigits

Figure 3.7: Learning rates when taxonomies are built from limited data.

Chapter 4

Enhanced Hierarchical Classification via Smoothing

4.1 Introduction

Given the widespread use of hierarchical taxonomies as tools for knowledge management, there has been considerable interest in automated methods for hierarchical classification, and numerous techniques have been proposed [DC00, CDAR98, KS97, TJHA05]. In Chapter 3 we proposed an algorithm for automated construction of n-ary tree based taxonomies. We also showed that classifiers learned over well-structured taxonomies achieved higher accuracy than those learned over badly organized ones. In this chapter, we will introduce ideas as well as algorithms that serve to further improve the accuracy of hierarchical classification systems.

While studying systems that classify instances into a taxonomy (hierarchical arrangement) of classes we have to consider many different configurations.

Some of these variations are a result of characteristics of the taxonomy itself. For instance, there exist many different relationships that can connect classes to each other; of these a principal one is subsumption. Under this relation, a class "is-a" type of its parent class. We can interpret this to mean that all data that belongs to a class also belongs to its parent class. Many hierarchies of classes we deal with in this chapter have this property. Taxonomies also vary in whether an internal node class is allowed to have data of its own or if it can only contain data belonging to its leaf-level sub-classes. Our work in this chapter will explicitly allow for hierarchical taxonomies of both these types. A related issue is that of instances belonging to multiple classes that are not related by a subsumption relation (for example, an instance belonging to two leaf-level nodes)? Once again, our algorithms from this chapter are applicable to both situations.

In addition to the characteristics of taxonomies, another major source of variation in hierarchical classification systems is the mechanism by which data is classified. One method, commonly applied in literature, is to learn a classifier at each node of the hierarchy to distinguish between the node's children. Then, starting from the root, a data instance can be progressively classified amongst the children of each node to determine the final class. This is the method that we used for classification experiments in Chapter 3. Another mechanism, and the one that we use in this chapter, involves learning a classifier at each node to distinguish the node's content from data not belonging to the node. In this approach, all learned classifiers are applied to a test data instance to obtain the true set of classes (labels). The subtle difference between the two approaches is that the former approach learns classifiers in the context of the parent class. In the latter approach, however, each classifier is learned on

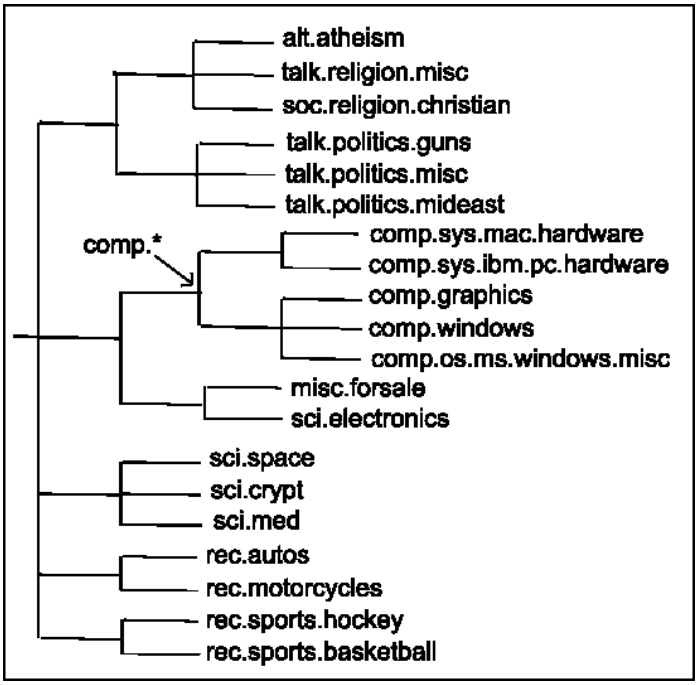


Figure 4.1: A taxonomy of classes from the 20-newsgroups dataset.

the entire training set distinguishing data in the corresponding node from all other data not in the node. As is evident from the description, the latter set of classifiers encodes some redundancies and in this chapter we introduce algorithms that exploit this fact to improve classification accuracy.

Finally, classification systems differ based on how they handle hierarchies that allow data to belong to internal nodes: whether the classifiers at each node learn a pattern for the data belonging to just the node or for the data belonging to all nodes in the subtree. In order to make these and other variation mentioned above concrete, lets consider a few real-world scenarios. We will refer to these scenarios throughout the chapter to place the applicability of our approaches in context. A taxonomy constructed on the 20-newsgroups dataset (Figure 4.1) is used as a running example below.

- **Scenario I**: In the 20-newsgroups dataset all instances belong to leaf-level classes of the taxonomy (Figure 4.1). Suppose while learning the classifiers at each node, the positive (negative) set of instances are from the leaf nodes within (outside) the subtree rooted at the node. For example, for the classifier at the node comp.*, the positive document set comes from all computer related leaf-level classes, and the negative document set from all other classes. In this scenario, when a test document is classified as belonging to a class (say, comp.graphics) all internal classes on the path to the root must also be predicted as true. Similarly, if an internal node is predicted as a true label for a document at least one of the sub-classes under it must also be predicted true.

- **Scenario II**: There exist taxonomies in which instances sometimes belong to classes that are internal nodes and not to any leaf-level classes. Once again consider the taxonomy for the 20-newsgroups dataset. We can imagine that the internal node comp.* might contain documents that discuss computers in general, and not software or hardware (MS or Mac) in particular. Now, if the classifiers are learned as in Scenario I - distinguishing the documents within the subtree from those outside - we can see that, if a node is predicted true for a test document then all internal classes on the path to the root must also be predicted as true labels. However, since there is no restriction that the document be associated to any leaf-level class, it is not necessary that at least one child of the node also be predicted true.

- **Scenario III**: As above, consider taxonomies where instances sometimes belong to internal classes. Consider a situation where the classifier for each node is trained to distinguish instances belonging to that class from

instances belonging to all other classes (even its own subclasses). For example, for the classifier at internal node comp.* documents belonging to it (on general computer topics) are used as the positive training set and documents belonging to all other nodes (including specialized hardware/software documents) are used as the negative training set. In this scenario, for any test document only one of the classifiers in the entire taxonomy can output true, making this system very different from those in Scenario I and Scenario II.

In addition to these three scenarios several other variations in the structure of the taxonomy and construction of classification problems can be imagined. A common aspect, however, is our mechanism for classifying new data instances into the taxonomy. For this purpose each learned classifier is applied to the new data instance to obtain a membership score for the corresponding node. The membership scores output may be binary, or they can be thresholded to determine the classes that are predicted as true labels.

However, from our discussion of different hierarchical classification scenarios we know that in many cases these membership scores are related across nodes. For example, in case of the 20-newsgroups dataset under scenario I, if class comp.graphics is predicted as a true label for a new data instance then so too must the comp.* class. The relationship between between classifier outputs can also involve inverse correlation, such as under scenario III, where if comp.* is predicted the true class then no other class can be true. Therefore, conditioned on the characteristics of taxonomies and specifics of classifier training, relationships between classes in a taxonomy lead to relationships between outputs of their classifiers. Furthermore, these properties

can also be devised from knowledge of the application domain or the behavior of classification algorithms.

Our Contribution.

The central idea in our work is that once we have identified the exact property that the outputs of classifiers in a taxonomy must satisfy, we can post-process the classification scores to enforce these constraints. Whenever classifier scores violate these constraints we will replace them with consistent scores that are as close as possible to the original ones. Since only a few classifiers are likely to make mistakes on any one instance it is hoped that the outputs of the incorrect ones will be modified appropriately.

In this chapter, we formulate the problem of enforcing constraints on classifier outputs under Scenarios I and II as *regularized isotonic median regression* problems. Classification score smoothing under Scenario III is modeled as a *regularized unimodal median regression* problem. These are generalizations of the classic isotonic regression problem. We will give algorithms to find the optimal solutions to these more general optimization problems in $O(n^2 \log n)$ time; this is equal to the best known algorithms for the classic problem. We will present empirical analysis from multiple real-world domains to show that post-processing of classifier scores results in improved classification accuracy.

NOTATION.

Before we describe the formulations and algorithms we establish some notation that will be used throughout this chapter.

Let $\mathcal{C}$ be a set of n classes in a taxonomy that have a one to one mapping with the nodes of a rooted tree T. Let leaf(T) and root(T) represent the set of leaves and the root of the tree T respectively. Let v be an node of T (written as $v \in T$) and let T_v denote the subtree rooted at v. We refer to a node in the tree T sometimes as a class. We use parent(v) to denote the parent of v in T, child(v) to denote the set of all immediate children of v in T. Let $\mathcal{D}$ denote the set of all data instances. These instances belong to one of the classes in $\mathcal{C}$; this mapping is denoted by function τ. Depending on the scenario τ may map instances to only a subset of classes in $\mathcal{C}$.

Each class $v \in T$ has associated with it a function $c_v : \mathcal{D} \rightarrow [0, 1]$, where $c_v(d)$ is the degree of membership of instance d in class v. Depending on the application setting this value can be a posterior probability; using an appropriate threshold it can be rounded to a boolean value. An instance d can be represented by a function x_d where the $x_d(v)$ represents the value $c_v(d)$. Since each class corresponds to a unique node in the taxonomy T, we can think of $x(.)$ as being scores assigned to nodes of T. From now onwards we will use T, v, and $x(v)$ as metaphors for the taxonomy, a class or a node, and the original classifier score or node value respectively. In our application settings we distinguish between an instance *belonging* to a class (implying that $\tau(d) = v$) and *associating* with a class (implying that because of the way classifiers were trained we expect $x_d(v) = 1$). However, when it is clear from context, we will use the two terms interchangeably.

4.2 Regularized Isotonic Regression

In this section we will formulate the problem of enforcing constraints on classifier outputs as an isotonic regression problem. We will introduce a regularized version of the problem which generalizes existing cost functions and provide an efficient algorithm to solve it.

4.2.1 Formulation

Consider the Scenario I described above. The data instances under this setting always belong to one of the leaf-level classes[1]; the range of τ is leaf(T). Moreover, for an internal node $v \in T$ the positive set of instances for training classifier c_v is the union of instances that belong to nodes in leaf(T_v). Consequently, this means that the true labeling of any instance is a leaf-level class v and all its parents on the path to the root(T). This property can be succinctly stated as follows:

Property 4.1 (Strict Classification Monotonicity)**.** *An instance belongs to a class in the taxonomy if and only if it belongs to at least one of its children classes.*

This means that we expect $x(v) = 1 \iff \exists u \in \text{child}(v) : x(u) = 1$.

However, as each classifier c_v processes the instances independently of other classifiers, they miss this intuitive relationship amongst classifier outputs; $x(\cdot)$ need not satisfy Property 4.1. Hence, we need to transform $x(\cdot)$ into smoothed classifier scores $y(\cdot)$ such that elements in $y(\cdot)$ satisfy the monotonicity property. Moreover, we assume that the individual classifiers have

[1]Our formulations and algorithms also hold for setting where an instance belongs to more than one leaf-level nodes.

reasonable accuracy and so we want to obtain $y(\cdot)$ that is as close as possible to the original scores $x(\cdot)$ while satisfying the monotonicity property.

Before we can state the problem, however, we need a more general monotonicity property that can handle real-valued classifier scores; currently the monotonicity property is defined over boolean classification values. We consider a natural generalization. Smoothed classifier scores $y(\cdot)$ satisfy the *generalized strict classification monotonicity* property if for every internal node v, with children $u_1, \ldots, u_\ell$, $y(v) = max\{y(u_1), \ldots, y(u_\ell)\}$, i.e., the smoothed score of an internal node is the equal to the maximum of its children's smoothed scores. Note that generalized strict monotonicity ensures that whenever an instance is associated with a non-root class v, first, it is also associated with parent(v), and second, it is also associated with at least one child of v, if any.

Now we are ready to state the problem:

Problem 4.2. *Given classifier scores $x(\cdot)$, find smoothed scores $y(\cdot)$ that minimize*

$$\sum_{v \in T} |x(v) - y(v)| \tag{4.1}$$

while satisfying the generalized version of Property 4.1.

Here we compute the distance between $x(\cdot)$ and $y(\cdot)$ via the L_1 distance. While any other distance measure could also have been used, we chose the L_1 metric because of its robustness to noise.

Now consider the Scenario II mentioned above. In this situation there exist some instances that belong only to internal nodes, and not to any of the leaf nodes. Moreover, the positive set of instances used for training classifier c_v for a node $v \in T$ is the union of instances that belong to all nodes in the

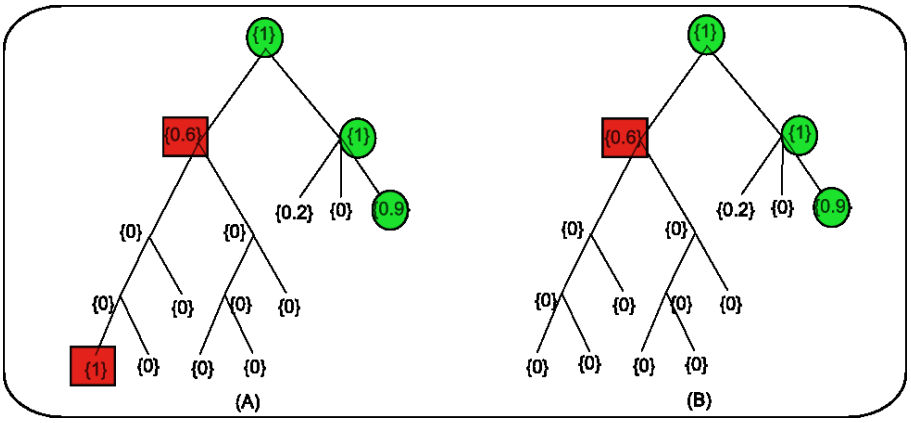


Figure 4.2: Examples of hierarchies with scores on nodes. The green circles highlight correct scores and the red squares erroneous ones.

subtree T_v. This implies that the true labels for an instance will be a node $v \in T$ and all its parents on the path to root(T). However, unlike Scenario I, its no longer necessary for at least one child of v to also be included in the true labels.

As is evident, this relationship between classifiers scores is not covered by Property 4.1, which enforces that every instance must belong to at least one leaf node. In order to incorporate situations such as Scenario II we state the relaxed monotonicity property.

Property 4.3 (Relaxed Classification Monotonicity)**.** *The classifier score of a node is always greater than or equal to the classifier scores of its children.*

Smoothed classifier scores $y(\cdot)$ satisfy relaxed monotonicity if for every internal node v, with children $u_1, \ldots, u_\ell$, $y(v) \geq max\{y(u_1), \ldots, y(u_\ell)\}$. Finding such a $y(\cdot)$ while minimizing Equation (4.1) will help us correct some of the errors introduced by the classifiers.

For example, consider the classification scores on the tree A in Figure 4.2. The nodes that are shaded by red squares represent the errors in the classification. The leaf-node with score 1 clearly violates monotonicity constraints since its ancestors' scores are lower than its own. This error will be corrected since, by Equation (4.1), it is more expensive to increase all the ancestors' scores than it is to reduce the erroneous node's score.

However, the relaxed monotonicity property will not correct certain other types of errors that might occur frequently. For example, consider the node with the erroneous score in tree B of Figure 4.2. This node doesn't violate the relaxed monotonicity property since its parent's score is higher than its own. However, this error node's score would have been corrected by the strict monotonicity property, which would have required at least one child of the error node to have the same score. It would have cost less (in terms of Equation (4.1)) to reduce 0.6 to 0, than to increase a whole series of 0s to 0.6.

In order to correct the latter type of errors we introduce an additional regularization term in our objective function, which penalizes violations of strict monotonicity. Hence, while we will accept $y(\cdot)$ that satisfy the relaxed monotonicity property as valid solutions, they will be charged for all the violations of the strict monotonicity constraints. In the case of the tree B in Figure 4.2, if the penalty is high enough, it will cost less to reduce the error node's value to 0 than to leave the scores as it is, thus correcting the false positive error. Regularization also corrects false negative errors when it is cheaper to increase a child node value than to pay the penalty.

Taking into account this regularization we state a new problem:

Problem 4.4 (Regularized Tree Isotonic Regression)**.** *Given classifier scores $x(\cdot)$, find smoothed scores $y(\cdot)$ that minimize*

$$\sum_{v \in T} w_v \cdot |x(v) - y(v)| + \sum_{v \in T, \{u_i\} = \text{child}(v)} \gamma_v \cdot (y(v) - \max\{y(u_i)\}) \tag{4.2}$$

while satisfying Property 4.3, where w_v and γ_v are node-specific weights and penalties, respectively.

In Equation (4.2), w_v are the node-specific weights that control the amount each classifier's score contributes to the total cost. We can set these weights to reflect our belief in the classifier's accuracy. The γ_v values are weights that control the extent to which violations of strict monotonicity constraints are prohibited: we overload the term penalty to also refer to these weights. These penalty values can be set in a node-specific fashion based on our domain knowledge of whether parent and child values can differ.

Problem 4.4 is a regularized version of the classic isotonic regression problem on trees which has been widely studied [AHKW06, PX99, Sto00]. It reduces to the standard isotonic regression when all the penalties are set to 0. Moreover, we can also enforce strict monotonicity (Property 4.1) by setting $\gamma_v = \infty$.

4.2.2 Algorithm for the case $\gamma = \infty$

We first present an algorithm for the special case of penalty $\gamma = \infty$; in fact, we'll solve Problem 4.2. The dynamic program presented in this section is similar in structure to the more general algorithm presented in the next section, and serves as a good starting point in introducing the ideas behind the approach.

Before we describe the algorithm we will discuss a crucial detail of the structure of the problem. We show that the optimal smoothed scores in $y(\cdot)$ can only come from the classifier scores $x(\cdot)$.

Lemma 4.5. *For Problem 4.2 there exists an optimal solution, $y(\cdot)$, where, for all $i \in T$ there is a $j \in T$ such that $y(i) = x(j)$.*

Proof. Consider the maximal connected subtree T' of nodes in T such that (1) $i \in T'$, and (2) for all $j \in T'$, $y(j) = y(i)$. If $y(i)$ is not the median of the set of scores $\{x(j) \mid j \in T'\}$, then we can push $y(i)$ closer to the median by a small amount and decrease the cost of the solution given by Equation (4.1); this follows since the median is the minimizer for L_1 distance. □

This result shows that in the optimal solution the smoothed score for each node will come from a finite set of values. Note that this result also holds for the case of weighted L_1 distance. In this case each weighted node in the tree can be considered as multiple nodes, which number proportional to the weight, and which always have the same score. In this case too, the minimizer remains the median of this expanded graph. And hence the smoothed score values come from the finite set of original values.

Algorithm BUILDERRORSTRICT $(v, x, \hat{x})$

1. `if` (v is a leaf) `then`
2. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
3. $\text{err}(v, i) = w_v \cdot |x(v) - \hat{x}(i)|$
4. `else`
5. `for` child u of node v
6. BUILDERRORSTRICT$(u, x, \hat{x})$
7. `for` $i = 1 : |\hat{x}|$ /* all values child u can take */
8. $\text{errheap}(i) = \text{err}(u, i)$
9. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
10. $val^* = \text{argmin}_{j \in \{1 \ldots |\hat{x}|\}, \hat{x}(j) \leq \hat{x}(i)}\ \text{errheap}(j)$
11. $\text{val}(u, i) = val^*$
12. $\text{err}'(i) += \text{err}(u, val^*)$
13. `if` $((\text{err}(u, i) - \text{err}(u, val^*)) < \text{minchilderr}(i)$) `then`
14. $\text{minchilderr}(i) = \text{err}(u, i) - \text{err}(u, val^*)$; $\text{minchild}(i) = u$
15. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
16. $\text{val}(\text{minchild}(i), i) = i$
17. $\text{err}(v, i) = \text{err}'(i) + \text{minchilderr}(i) + w_v \cdot |x(v) - \hat{x}(i)|$

Algorithm ISOTONESMOOTH $(\text{err}, \text{val}, \hat{x})$

1. $val^* = \text{argmin}_{i \in \{1 \ldots |\hat{x}|\}}\ \text{err}(\text{root}(T), i)$
2. $p(\text{root}(T)) = val^*$; $y(\text{root}(T)) = \hat{x}(val^*)$
3. `for` v in a breadth-first search order of T
4. $p(v) = \text{val}(v, p(\text{parent}(v)))$; $y(v) = \hat{x}(p(v))$

Figure 4.3: Algorithm to solve Problem 4.2. Array x contains the original classifier scores and $\hat{x}$ is the set of unique values in x. w_v denote the node-specific weights. BUILDERRORSTRICT constructs functions $\text{err}(\cdot, \cdot)$ and $\text{val}(\cdot, \cdot)$ which are then used by ISOTONESMOOTH to find the smoothed scores $y(\cdot)$.

To solve Problem 4.2 optimally we construct a dynamic program (pseudo-code in Figure 4.3). The program consists of two main algorithms, (1) BuildErrorStrict, which builds up the index function val and error function err, and (2) IsotoneSmooth, which uses the val function to compute the optimal values for each node in the tree. Let $\hat{x}$ be the set of unique values in $x(\cdot)$, and let i be an index into this set. Then index function $val(v, i)$ holds the index of the value that node v should take in the optimal solution when its parent takes the value $\hat{x}(i)$. In other words, when $y(\text{parent}(v)) = \hat{x}(i)$ then $y(v) = \hat{x}(\text{val}(v, i))$. In order to compute the function val, BuildErrorStrict computes for each node v the function $\text{err}(v, i)$, which holds the total cost of the optimal smoothed scores in the subtree T_v when $y(v) = \hat{x}(i)$.

Initially BuildErrorStrict is invoked with root(T) as a parameter. The function then recursively calls itself (step 6) on the nodes of T in a depth-first order. While processing a node v, for each possible value $\hat{x}(i)$ that v can set itself to, BuildErrorStrict finds the best values for its children that are less than or equal to $\hat{x}(i)$ (step 10). All children of v are assumed to be set to their best possible values (step 11) and their costs (errors) are added up (step 12). Also, since in the final solution one of the children's value has to be equal to value of v, BuildErrorStrict maintains information about the child that would cost the least (steps 13 & 14) to move to $\hat{x}(i)$ (step 16). At the end the cost of all children and the additional cost of the "minchild" that is moved are added (step 17) to obtain the cost for the current node.

To demonstrate the correctness of this algorithm, we show that the restriction of the optimal solution to a subtree is also the optimal solution for the subtree under the monotonicity constraint imposed by its parent.

Consider the subtree rooted at any non-root node $v \in T$. Now suppose the

smoothed score $y(\text{parent}(v))$ is specified. Then, let $z(\cdot)$ be the smoothed scores of the optimal solution to the regularized tree isotonic regression problem for this subtree, under the additional constraint that $z(v) \leq y(\text{parent}(v))$. Note that if v is chosen as "minchild" in algorithm BuildErrorStrict above, the constraint is $z(v) = y(\text{parent}(v))$.

Lemma 4.6. *For all nodes i in the subtree of v, $y(i) = z(i)$.*

Proof. Consider a smoothed solution $w(\cdot)$ where $w(i) = z(i)$ for all nodes i in the subtree of v, and $w(i) = y(i)$ otherwise. It is clear that since $z(\cdot)$ obeys the monotonicity property and $z(v) \leq y(\text{parent}(v))$, the solution $w(\cdot)$ obeys the monotonicity property. Now, the cost $c(w)$ is the sum of the cost for the smoothed scores $z(i)$ in the subtree of v and the cost for the scores $y(k)$ for all other nodes. Thus, the difference between $c(w)$ and $c(y)$ is just the difference in costs for $z(i)$ and $y(i)$ in the subtree of v, for which we know that $z(\cdot)$ is the optimal. The lemma follows. □

Theorem 4.7. *Algorithm* IsotoneSmooth *in Figure 4.3 solves Problem 4.2 exactly.*

Proof. The algorithm computes up the optimal smoothed scores for each subtree, i.e., the $\text{err}(\cdot, \cdot)$ arrays, while maintaining Property 4.3 for every possible smoothed score of the parent. Further, the child that costs the least to move from its optimal position to the parent value is moved. This causes the least increase in the cost in Equation (4.1). Hence, the solution computed for each possible smoothed value of the parent is optimal. By Lemma 4.5, the parent can take only finitely many smoothed scores in the optimal solution, and by Lemma 4.6, combining the optimal smoothed scores for subtrees yields the optimal smoothed scores for the entire tree. □

COMPLEXITY. Let $|T| = n$, and so $|\hat{x}|$ can at most be n. The dynamic programming table takes $O(n)$ space per node, and so the total space required is $O(n^2)$. Next, we consider the running time of the algorithm. In the algorithm BUILDERRORSTRICT-I, step 2 takes $O(n^2)$ time, step 7 takes $O(n^2)$ time amortized over all calls (this loop is called for each node only once), and the loop in step 9 can be done in $O(n^2 \log n)$ time by storing errheap values in a heap and then running over the values $i \in \{1 \dots |\hat{x}|\}$ in ascending order of $\hat{x}(i)$. Hence, the total running time is $O(n^2 \log n)$. Note that this is same as the best time complexity of previously known algorithms for the non-regularized forms of tree isotonic regression [AHKW06].

4.2.3 Algorithm for Regularized Tree Isotonic Regression

In the previous section we presented an algorithm to solve Problem 4.2. In this section we give an algorithm that solves the more general regularized isotonic regression problem exactly. The main difference between the two problems is that in the latter case the hard constraint of a parent's value being equal to at least one of its children's value is enforced via soft penalties. These violations of the strict monotonicity rule are charged for in the objective function, so that if the cost of incurring the penalties is lower than the improvement in L_1 error, the optimal solution will contain violations. This makes the current problem much tougher to solve. When constraints were strict each child only had two ways it could set its own value: equal to the best possible value less than the parent's value or equal to the parent's value. In the current problem because penalties are soft, the child has more options of values it can set itself to. However, we show that this set of possible values is still finite.

Here we prove that the central result (Lemma 4.5) which facilitated the algorithm in Figure 4.3 also holds for the modified objective function Equation (4.2).

Preliminary Facts.

Consider the maximal connected subtree T' of nodes in T such that (1) $i \in T'$, and (2) for all $j \in T'$, $y(j) = y(i)$. Let m be the median of the set of original scores $S_{T'} = \{x(j) \mid j \in T'\}$. The cost incurred by T' through the first term of Equation (4.2) (L_1 distance between x and y) is minimized when $y(i) = m$. As we raise or lower $y(i)$ the increase in this cost is piecewise-linear, with the discontinuities at the values in the set $S_{T'}$. In other words, in between any two adjacent values in $S_{T'}$ the rate of change in the L_1 cost is constant (we denote this rate by a function $r_m(\cdot)$).

Now lets consider the cost due to penalties. Let u be the "maximal" node in T', such that parent$(u) \ni T'$. Note that, as T' is connected and is a tree, there is a unique such node. Also, let $\{v_i\} \in T'$ be a set of nodes such that at least one child of each v_i is not in T'. Some elements in the set $P_{T'} = \{\text{parent}(u), \text{child}(\{v_i\})\}$ are involved in penalties for having values different from $y(i)$. This cost from penalties also changes in a piecewise-linear fashion with possible discontinuities at the values in the set $P_{T'}$. Let us denote the rate of change of penalty-based cost as $r_p(\cdot)$.

Lemma 4.8. *For Problem 4.4 there exists an optimal solution, $y(\cdot)$, where, for all $i \in T$ there is a $j \in T$ such that $y(i) = x(j)$.*

Proof. Let the cost of the optimal solution $y(\cdot)$ be $c(y)$. We will prove the above lemma for a solution $y(\cdot)$ that has the fewest distinct score values of all

solutions that have cost $c(y)$. If this is not the case, then we'll show how $y(\cdot)$ can be converted to $y'(\cdot)$ that has fewer distinct score values.

Consider the maximal connected subtree T' of nodes in T such that (1) $i \in T'$, (2) for all $j \in T'$, $y(j) = y(i)$, and (3) there does not exist any $j \in T$ such that $y(i) = x(j)$. As mentioned above, since $y(i) \neq m$ (median of $S_{T'} = \{x(j) \mid j \in T'\}$), we can decrease the cost due to L_1 error at the rate of $r_m(y(i))$ by moving $y(i)$ towards m. Also, the cost due to penalties changes at the rate of $r_p(y(i))$ when we move $y(i)$. If the values $r_m(y(i)) \neq -r_p(y(i))$ then we can move $y(i)$ very slightly to decrease the overall cost.

Hence, we consider the case where $r_m(y(i)) = -r_p(y(i))$. Let $m_1 \in S_{T'}$ be the closest value to $y(i)$ in between it and m. Since the two rates of cost change counterbalance each other, small changes in $y(i)$ result in solutions with the exact same cost. In fact, we can move $y(i)$ to m_1 without any change in the overall cost, hence producing an optimal solution with satisfies the lemma. To see why this happens, consider that as we move $y(i)$ to m_1, $r_m(y(i))$ will not change as explained above. The quantity $r_p(y(i))$ will change if we encounter an element from the set $P_{T'}$ in between $y(i)$ and m_1. But this means that we can obtain a solution with cost $c(y)$ that has fewer distinct values, which violates our assumption. Another way in which $r_p(y(i))$ can change is if the maximal node u stops/starts having the highest value of all its siblings (stops/starts getting penalized) as we move $y(i)$. However, it can be easily verified that this can only reduce the cost of the new solution further. $\square$

Algorithm BUILDERRORRELAX $(v, x, \hat{x})$

1. `if` (v is a leaf) `then`
2. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
3. $\text{err}(v, i) = w_v \cdot |x(v) - \hat{x}(i)|$
4. `else`
5. `for` child u of node v
6. BUILDERRORRELAX$(u, x, \hat{x})$
7. `for` $i = 1 : |\hat{x}|$ /* all values child u can take */
8. $\text{errheap}(i) = \text{err}(u, i)$
9. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
10. $val^* = \text{argmin}_{j \in \{1 \ldots |\hat{x}|\}, \hat{x}(j) \leq \hat{x}(i)}$ errheap(j)
11. $\text{val}(u, i) = val^*$
12. $\text{err}'(i) += \text{err}(u, val^*)$
13. $\text{errchildren}(i, u) = \text{err}(u, i) - \text{err}(u, val^*) - \gamma_v \cdot \hat{x}(i)$
14. `if` $((\hat{x}(val^*) > \text{maxchildval}(i))$ `then`
15. $\text{maxchildval}(i) = \hat{x}(val^*)$
16. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
17. $(val^*, u) = \text{argmin}_{j \in \{1 \ldots |\hat{x}|\}, k \in child(v), \text{maxchildval}(i) \leq \hat{x}(j) \leq \hat{x}(i)}$ errchildren(j, k)
18. $\text{val}(u, i) = val^*$
19. $\text{err}'(i) += \text{errchildren}(val^*, u) + \gamma_v \cdot \hat{x}(val^*) + \gamma_v \cdot |\hat{x}(i) - \hat{x}(val^*)|$
20. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
21. $\text{err}(v, i) = \text{err}'(i) + w_v \cdot |x(v) - \hat{x}(i)|$

Algorithm ISOTONESMOOTH $(\text{err}, \text{val}, \hat{x})$

1. $val^* = \text{argmin}_{i \in \{1 \ldots |\hat{x}|\}}$ $\text{err}(\text{root}(T), i)$
2. $p(\text{root}(T)) = val^*; \quad y(\text{root}(T)) = \hat{x}(val^*)$
3. `for` v in a breadth-first search order of T
4. $p(v) = \text{val}(v, p(\text{parent}(v))); \quad y(v) = \hat{x}(p(v))$

Figure 4.4: Algorithm to solve Problem 4.4. Array x contains the original classifier scores and $\hat{x}$ is the set of unique values in x. w_v and γ_v denote the node-specific weights and penalties. BUILDERRORRELAX constructs functions $\text{err}(\cdot, \cdot)$ and $\text{val}(\cdot, \cdot)$ which are then used by ISOTONESMOOTH to find the smoothed scores $y(\cdot)$.

Now we are ready to present the dynamic program to solve Problem 4.4 (pseudo-code in Figure 4.4). The input to the system is $x(\cdot)$, the original classifier scores; $\hat{x}$ is the set of unique values in x. The algorithm BuildErrorRelax, invoked on the root node, recurses over nodes of T in a depth first order (step 6) and fills up the index function val and error function err. The index function $\text{val}(v, i)$ holds the index of the value that node v should take in the optimal solution when its parent takes the value $\hat{x}(i)$, while the function $\text{err}(v, i)$ stores the total cost of the optimal smoothed scores in the subtree rooted at v when $y(v) = \hat{x}(i)$. In these respects, BuildErrorRelax is identical to the algorithm for the strict monotonicity property presented in Section 4.2.2. The main difference is that now the cost of the solution doesn't just come from the L_1 error, but also from the penalties. Hence, while picking a value for a child node we have to consider both the cost of the optimal solution in the subtree of the child and the cost of the child's value differing from the parent value. To add to the complexity, we have to consider the latter cost only when the child has the maximum value amongst its siblings.

While operating on a node v, for each possible value $\hat{x}(i)$ that v can set itself to, BuildErrorRelax first obtains the best value assignments for its children that are less than or equal to $\hat{x}(i)$ (step 10). At this stage, only the cost of the optimal solutions in the subtree of a child is considered while determining its best value (step 8); for now the cost, due to penalties, of a child's value differing from $\hat{x}(i)$ is ignored. The val array entries of the children are set to these best values (step 11) and the costs are added up (step 12). While processing each child this way another table errchildren is populated with the additional cost of moving one of the children to be the maximum child under v (step 13). Once all children values have been set this way, in a

second pass the errchildren table is used to determine which child should be moved, and what value it should be moved to, so that the sum of the cost from its subtree and penalty w.r.t. the parent value is minimum (step 17). Once the child and its new value are determined, step 18 and step 19 update the val array and the cost of the current node v respectively. Note that the initial assignment of values to children might not change in this second pass if the original child with the maximum value also costs the least once we take into account penalties. Once BUILDERRORRELAX has filled the val array, the function ISOTONESMOOTH uses it to compute the optimal values for each node in the tree.

To demonstrate the correctness of this algorithm, we first show that the restriction of the optimal solution to a subtree is also the optimal solution for the subtree under the constraints imposed by its parent. Consider the subtree rooted at any non-root node $v \in T$. Now suppose the smoothed score $y(\text{parent}(v))$ is specified and also whether v has the maximum value of its siblings in the optimal solution. If v does not have the maximum value then let $z(\cdot)$ be the smoothed scores of the optimal solution to the regularized tree isotonic regression problem for this subtree, under the additional constraint that $z(v) \leq y(\text{parent}(v))$. If v does have the maximum value then let $z(\cdot)$ represent the optimal smoothed scores in T_v such that they minimize $c(z) + \gamma_v \cdot |y(\text{parent}(v)) - z(v)|$ subject to $z(v) \leq y(\text{parent}(v))$, where $c(z)$ is the cost of the subtree T_v under $z(\cdot)$.

Lemma 4.9. *For all nodes i in the subtree of v, $y(i) = z(i)$.*

Proof. This Lemma can be proved by similar reasoning as Lemma 4.6. Consider a smoothed solution $w(\cdot)$ where $w(i) = z(i)$ for all nodes i in the subtree

of v, and $w(i) = y(i)$ otherwise. It is clear that since $z(\cdot)$ obeys the monotonicity property and $z(v) \leq y(\text{parent}(v))$, the solution $w(\cdot)$ obeys the monotonicity property. Now, the cost $c(w)$ is the sum of the cost for the smoothed scores $z(i)$ in the subtree of v and the cost for the scores $y(k)$ for all other nodes, plus the penalty of each parent's value differing for the maximum of its children's values. Thus, the difference between $c(w)$ and $c(y)$ is just the difference in L_1 and penalty costs for $z(i)$ and $y(i)$ in the subtree of v, including the difference between $\gamma_v \cdot (y(\text{parent}(v)) - z(v))$ and $\gamma_v \cdot (y(\text{parent}(v)) - y(v))$. For this cost we know that $z(\cdot)$ is the optimal. The lemma follows. □

In order to proceed with showing correctness of our algorithm, we have to next show that the two separate loops in steps 9-15 and steps 16-19 do an optimal job of assigning values to children nodes. The first loop assigns children values only based on the costs within their subtrees. The second loop then changes the value of a single child node making it the maximum amongst all siblings. Hence, we need to prove that this one transformation results in the optimal assignments of values to children.

Consider a node $v \in T$ with children $u_1, \ldots, u_\ell$. Let $y(\cdot)$ be the optimal solution to Problem 4.4 for the subtree T_v when $y(v)$ is constrained to be some value $\hat{x}(i)$. Also, let $y'(\cdot)$ be a valid solution with $y'(v) = \hat{x}(i)$ obtained after execution of steps 5-15 in algorithm BuildErrorRelax in Figure 4.4.

Lemma 4.10. *For a node $v \in T$ with children $u_1, \ldots, u_\ell$, at most one child $u_m = \text{argmax}\{y(u_i)\}$ will be such that $y'(u_m) \neq y(u_m)$. All other children will have the same values in $y'(\cdot)$ and $y(\cdot)$.*

Proof. Let $u_j \neq u_m$ be a child of node v such that $y'(u_j) \neq y(u_j)$. There can be two cases, (1) $y'(u_j) \leq y(u_m)$ and (2) $y'(u_j) > y(u_m)$. For case (1), we can

undo the move of u_j from $y'(u_j)$ to $y(u_j)$ and reduce the cost of the solution. This is because $y'(u_j)$ is the cheapest solution for u_j less than or equal to $y(v)$ (from step 10 of BUILDERRORRELAX). This case implies that $y(\cdot)$ is not the optimal solution and so it is not possible. For case (2), once again we can reset u_j from $y(u_j)$ to $y'(u_j)$ and obtain a cheaper solution. This is because cost of the subtree T_{u_j} is lower at $y'(u_j)$ than at $y(u_j)$ (step 10), and since $y'(u_j)$ is closer to $y(v)$ than $y(u_m)$, the cost from penalties is lower too. no other node than u_m could have □

Theorem 4.11. *Algorithm* ISOTONESMOOTH *in Figure 4.4 solves Problem 4.4 exactly.*

Proof. By Lemma 4.8, in the optimal solution, a node can take only take values from a finite sized set, and by Lemma 4.6, combining the optimal smoothed scores for subtrees yields the optimal smoothed scores for the entire tree. Hence, all that remains to be shown is that BUILDERRORRELAX finds optimal assignments for the children u_l of a given node v. For each value $\hat{x}(i)$ the parent can take, by steps 8 and 10 each child is assigned to its optimal value $\text{val}(u_l, i)$ less than or equal to $\hat{x}(i)$. The additional cost of the maximum child u_l assigned to $\hat{x}(j)$ is $\text{err}(u_l, j) - \text{err}(u_l, \text{val}(u_l, i)) + \gamma_v \cdot (\hat{x}(i) - \hat{x}(j))$. Hence, storing additional costs in errchildren by step 13 and extracting smallest cost increases via step 17 returns the child that causes the least increase in cost via Equation (4.2). By Lemma 4.10 it is sufficient to adjust the value of only one such child value to obtain the optimal solution. □

COMPLEXITY. The space complexity of the algorithm is $O(n^2)$ as there are $O(n)$ entries in the dynamic programming table for each node. In the algorithm BUILDERRORRELAX, step 2 takes $O(n^2)$ time, step 7 takes $O(n^2)$ time

amortized over all calls (this loop is called for each node only once), and the loops in step 9 and step 16 can be done in $O(n^2 \log n)$ time by storing errheap and errchildren values in heaps and then running over the values $i \in \{1 \dots |\hat{x}|\}$ in ascending order of $\hat{x}(i)$. Hence, the total running time is $O(n^2 \log n)$. Note that this is same as the complexity of the algorithm for the strict case (in Figure 4.4) and also the best time complexity of previously known algorithms for the non-regularized forms of tree isotonic regression [AHKW06].

4.3 Regularized Unimodal Regression

In the previous section we formulated the task of smoothing classifier outputs in Scenarios I and II as regularized isotonic regression problems. In this section we will discuss how to perform classifier output smoothing under the conditions of Scenario III as described in the introduction to the chapter.

4.3.1 Formulation

Consider classification under Scenario III. The instances, in this case, can belong to any of the nodes in the hierarchy; in other words, the range of τ is T. Moreover, the classifier at node v is trained to distinguish the instances that belongs to v from instances that belong to all other nodes; even from instances belonging to child nodes of v. In such a setting, a instance belongs to, and is associated with, only a single class/node.

In terms of classifiers scores this means that the nodes in the tree should be labeled in such a fashion that there is only one node with a peak value and the rest of the values "fall away". By this we mean that values on nodes monotonicity decrease as we move away from the "peak-node". Notice that such a set of values satisfy the relaxed monotonicity constraint from Subsection 4.2.1

when the root of the tree is the peak-node. However, since in our applications edges are directed and the root of the hierarchy has a special meaning (the superclass), we cannot just move it to the peak-node. Therefore, we soften the notion of a peak-node to mean a node whose value is higher than all its parents and descendants in the tree. This definition leaves room for multiple peak-nodes in a set of smoothed scores on a hierarchy. This is useful because documents often have multi-modal topic distributions and might legitimately belong to multiple nodes in the hierarchy. These types of situations are often encountered in real world applications, such as classification into a web taxonomy, which often have a faceted structure [Hea06]. Hence, we define the property that we want smoothed scores to satisfy under Scenario III.

Property 4.12 (Classification Tree Unimodality)**.** *Smoothed classifier scores* $y(\cdot)$ *are unimodal on the taxonomy, if for each leaf node* $v \in \text{leaf}(T)$*, where the values of nodes on the path from root are* $\{y_1 \dots y_v\}$*, there exists a node* i*, such that* $y_1 \leq y_2 \leq \dots y_i \geq y_{i+1} \geq \dots y_v$*.*

For any path from the root to a leaf node v, multiple nodes can qualify as the node i in the property above if they all have the maximum score. Of all these nodes the one that is closest to the leaf node will be referred to as the *cross-over* node. We call the part of the path from the root to the cross-over node as the "up-phase" and the rest of the path to leaf v as the "down-phase". Note that for a path either the up-phase or the down-phase may not exist; the scores might only increase or only decrease. All that the property above guarantees is that the down-phase will follow the up-phase, if both exist. There exists one cross-over node for each path, though, multiple paths might share cross-over nodes. A node in the tree that acts as a cross-over node for all paths from the root to the leaves in its subtree is called the

"peak-node". Notice that by the very definition of peak-nodes, they cannot be ascendants or descendants of each other in the tree. It is these peak-nodes in our smoothed scores, with values above a threshold, that will be predicted as true labels by our approach.

Now that we have stated the property that we expect smoothed classifier scores to follow, we can state the problem of finding such a solution.

Problem 4.13 (Regularized Unimodal Regression)**.** *Given classifier scores $x(\cdot)$, find smoothed scores $y(\cdot)$ that minimize*

$$\sum_{v \in T} w_v \cdot |x(v) - y(v)| + \sum_{v,u \in T, u \in \text{child}(v)} \gamma_{vu} \cdot |y(v) - y(u)| \tag{4.3}$$

while satisfying Property 4.12, where w_v and γ_u are node-specific weights and penalties, respectively.

Let us compare this problem with regularized *isotonic* regression (Problem 4.4). One major difference is the nature of the constraints that need to be satisfied; in this case the values from the root to the children can increase and then decrease. However, there are other differences as well. First, and most importantly, the penalty term of the cost function in the current problem penalizes all differences between parent and child node values. This is in contrast to Problem 4.4 where a single child of each parent is penalized. The reason is that in the current problem as we traverse down the tree from the root, we want increases in score only if the path leads to the true peak-node. All other paths should remain at the same score as their parents (preferably zero). Also, as we move down the subtree of the true peak-node, we want that the values should go to zero very quickly, as according to our scenario there is just one peak-node on any path from the root to the leaves. The second difference is

that the penalty function γ now can depend on the parent-child pair; in previous problems the penalty function was parameterized by the parent node. This gives us additional flexibility in encoding our domain knowledge about the taxonomy into the smoothing process. Finally, since the parent value might be higher or lower than the child value, we need to use their absolute difference in computing the penalties.

4.3.2 Algorithm

Before we present the algorithm for obtaining an optimal solution to Problem 4.13, we will show that in the optimal solution all unique classifier values will come from the finite set of original classifier values.

Proposition 4.14. *For Problem 4.13 there exists an optimal solution, $y(\cdot)$, where, for all $i \in T$ there is a $j \in T$ such that $y(i) = x(j)$.*

Proof. This result follows directly from Lemma 4.8. □

The basic idea behind the algorithm is that there is a unique cross-over node on every path from the root to the leaves. Hence, each node that is on the up-phase of the path to the leaves can offer its children the option to be lower or higher than it in value. A node that is on the down-phase can only offer its children one option of being lower than it in value. In order to implement this, algorithm BuildErrorUnimodal builds up the err and val functions, which store the optimal error and value assignments of nodes. The value $\text{err}(v, i, phase)$ stores the least value of error possible when a node v takes the value indexed by i when it is a part of the up/down phase. The entry $\text{val}(v, i, phase)$ stores the index of the value of v where the least error is

obtained when the parent takes the value indexed by i and is on the up/down phase of the path.

The BuildErrorUnimodal algorithm has a similar structure to the two dynamic programs presented previously in this chapter. It is invoked on the root node and recursively calls itself on the children in a depth first order so as to fill the err and val arrays. The major difference in the current program is the way it finds optimal child assignments for the two up/down phases. For each value $\hat{x}(i)$ that a parent node v can take, first the optimal values for the children in the down-phase are found (steps 8 & step 11) and the err and val arrays are updated (steps 12 & 13). The optimal error and value of a child on the down-phase are also stored in the arrays mindnerr and mindnval (steps 12 & 13). Next the optimal values for the children constrained to be higher than $\hat{x}(i)$ are found (steps 9 & 15). Since the parent in the up-phase can offer its children the option of being higher as well as lower than its value, both options are compared, and the one with the lower optimal error is chosen (steps 17-20). Once again, the val and err arrays are updated.

Once the val array table has been filled by BuildErrorUnimodal the algorithm UnimodalSmooth traces the paths from the root that give least error. It uses the current value of "phase" to look at the appropriate entries of the val array (steps 5 & 7). Moreover, it detects when a crossover point has been reached by checking the optimal value of the current node with that of the parent (step 8). The function UnimodalSmooth is then recursively called on the children with the updated value of phase.

Algorithm BUILDERRORUNIMODAL $(v, x, \hat{x})$

1. `if` (v is a leaf) `then`
2. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
3. $\text{err}(v, i, up) = \text{err}(v, i, dn) = w_v \cdot |x(v) - \hat{x}(i)|$
4. `else`
5. `for` child u of node v
6. BUILDERRORUNIMODAL$(u, x, \hat{x})$
7. `for` $i = 1 : |\hat{x}|$ /* all values node u can take */
8. $\text{errheapdn}(i) = \text{err}(u, i, dn) - \gamma_{vu} \cdot \hat{x}(i)$
9. $\text{errheapup}(i) = \text{err}(u, i, up) + \gamma_{vu} \cdot \hat{x}(i)$
10. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
11. $val^* = \text{argmin}_{j \in \{1 \ldots |\hat{x}|\}, \hat{x}(j) \leq \hat{x}(i)}$ errheapdn(j)
12. $\text{errdn}'(i) += \text{mindnerr}(i) = \text{err}(u, val^*, dn) + \gamma_{vu} \cdot |\hat{x}(i) - \hat{x}(val^*)|$
13. $\text{val}(u, i, dn) = \text{mindnval}(i) = val^*$
14. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
15. $val^* = \text{argmin}_{j \in \{1 \ldots |\hat{x}|\}, \hat{x}(j) \geq \hat{x}(i)}$ errheapup(j)
16. $tmperr = \text{err}(u, val^*, up) + \gamma_{vu} \cdot |\hat{x}(i) - \hat{x}(val^*)|$
17. `if`$(tmperr > \text{mindnerr}(i))$ `then`
18. $\text{errup}'(i) += \text{mindnerr}(i); \text{val}(u, i, up) = \text{mindnval}(i)$
19. `else`
20. $\text{errup}'(i) += tmperr; \text{val}(u, i, up) = val^*$
21. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
22. $\text{err}(v, i, up) = \text{errup}'(i) + w_v \cdot |x(v) - \hat{x}(i)|$
23. $\text{err}(v, i, dn) = \text{errdn}'(i) + w_v \cdot |x(v) - \hat{x}(i)|$

Algorithm UNIMODALSMOOTH $(v, \text{err}, \text{val}, \hat{x}, \text{phase})$

1. `if` (v is the root) `then`
2. $(val^*, phase) = \text{argmin}_{i \in T, j \in \{up, dn\}}$ $\text{err}(\text{root}(T), i, j)$
3. $p(\text{root}(T)) = val^*; y(\text{root}(T)) = \hat{x}(val^*)$
4. `else if` $(phase == dn)$ `then`
5. $p(v) = \text{val}(v, p(\text{parent}(v)), dn); y(v) = \hat{x}(p(v))$
6. `else`
7. $p(v) = \text{val}(v, p(\text{parent}(v)), up); y(v) = \hat{x}(p(v))$
8. `if` $(y(v) < y(parent(v)))$ `then` $phase = dn$
9. `for` $u \in child(v)$
10. UNIMODALSMOOTH$(u, \text{err}, \text{val}, \hat{x}, \text{phase})$

Figure 4.5: Algorithm to solve Problem 4.13. Array x contains the original classifier scores and $\hat{x}$ is the set of unique values in x. w_v and γ_{vu} denote the node-specific weights and penalties. BUILDERRORUNIMODAL constructs functions $\text{err}(\cdot, \cdot)$ and $\text{val}(\cdot, \cdot)$ which are then used by UNIMODALSMOOTH to find the smoothed scores $y(\cdot)$.

To show that this algorithm is correct we first show that the restriction of the optimal solution to the subtree under a node is also the optimal solution for the subtree under the constraints from the node's parent. Consider the subtree rooted at a non-root node $v \in T$. Suppose the parent score $y(\text{parent}(v))$ is given and, without any loss of generality, that the current phase is up. Then let $z(\cdot)$ be the optimal scores for nodes in subtree T_v such that $c(z)+\gamma_{parent(v),z} \cdot |y(\text{parent}(v)) - z(v)|$ is minimized subject to $z(v) \geq y(\text{parent}(v))$, where $c(z)$ is the cost of the subtree T_v under $z(\cdot)$. Then the following lemma holds (also for the case when phase=down):

Lemma 4.15. *For all nodes i in the subtree of v, $y(i) = z(i)$.*

Proof. This lemma can be proved using arguments identical to the proof of Lemma 4.9. □

Theorem 4.16. *Algorithm* UnimodalSmooth *in Figure 4.5 solves Problem 4.13 exactly.*

Proof. For every possible smoothed score of the parent $\hat{x}(i)$, the algorithm finds the best value for each child lesser than $\hat{x}(i)$ (taking into account penalty) and uses it for the down-phase solution. For the up-phase solution it compares each child's best values higher and lower than $\hat{x}(i)$ (once again, taking penalty into account) and uses the one with lower error. By Lemma 4.5, the parent can take only finitely many smoothed scores in the optimal solution, and by Lemma 4.6, combining the optimal smoothed scores for subtrees yields the optimal smoothed scores for the entire tree. Finally, algorithm UnimodalSmooth starts with the minimum error value for the root considering both the up and down phases. Before recursively calling itself on a child node, the UnimodalSmooth function checks if a phase change has occurred. □

COMPLEXITY. Let the number of nodes be $n = |T|$, and so $|\hat{x}|$ can at most be n. The dynamic programming table has both the up and down phase entries for each possible value of each node. Hence the storage goes up by a factor of 2 over the ISOTONESMOOTH algorithm, and is still $O(n^2)$. In the algorithm BUILDERRORUNIMODAL, step 2 takes $O(n^2)$ time, step 7 takes $O(n^2)$ time amortized over all calls (this loop is called for each node only once), and the loops in step 10 and 14 can be performed in $O(n^2 \log n)$ time by storing errheapdn and errheapup values in heaps and then running over the values $i \in \{1 \ldots |\hat{x}|\}$ in ascending order of $\hat{x}(i)$. Hence, the total running time is $O(n^2 \log n)$, the same as the ISOTONESMOOTH algorithm.

4.4 Experimental Setup

In this section we provide details of the framework under which we evaluate the effect of smoothing of classifier outputs on accuracy of classification. First we describe the datasets used in the evaluation. Next we enumerate the different aspects of smoothing performance that we want to compare and the measures that compute them. Finally, we end the section with implementation details of our classification and smoothing system.

4.4.1 Datasets

The following are the datasets that we used in our evaluation of our algorithms under the three scenarios mentioned earlier. All datasets used in this thesis are available from `http://www.ideal.ece.utexas.edu/~kunal/thesis/`.

TEXT DATASETS. We use the 20-newsgroups dataset to perform our empirical analysis in the text classification domain. This dataset has been extensively used for evaluating text categorization techniques [RR01]. It contains a total of 18, 828 documents that correspond to English-language posts to 20 different newsgroups, with a little fewer than a 1000 documents in each. The dataset presents a fairly challenging classification task as some of the categories (newsgroups) are very similar to each other with many documents cross-posted among them (e.g., alt.atheism and talk.religion.misc). In order to evaluate our classifier smoothing schemes we use the hierarchical arrangement of the 20 newsgroups/classes constructed during experiments in [PRG06]. The hierarchy is shown in Figure 4.1 and we refer to it as TAXONOMYI.

Since all documents in the 20-newsgroups taxonomy belong to leaf level

nodes, it serves to evaluate our approach on Scenario I. In order to represent the conditions encountered under Scenario III, we constructed "hybrid" documents that represent the content of internal nodes in the hierarchy. For each internal node class, hybrid document were constructed by combining documents from a subset of its children classes. The size and constituents of this subset were randomly picked. Care was taken to ensure that the number of distinct words in a hybrid document as well as its length were similar to the documents being combined. For each internal node around 1000 new documents were created this way. We refer to this modified taxonomy as TaxonomyII.

Remote Sensing Datasets. In addition to text data, we also evaluate our approach on hierarchies of classes where the instances are pixels in hyperspectral images. We use two separate datasets which comprise hyperspectral images of the NASA's John F. Kennedy Space Center (KSC) [Mor02], Florida and the Okavango Delta, Botswana [HCCG05]. These datasets have previously been described in detail in Section 3.4. In that section we used our approach to create n-ary tree based hierarchies of the classes in the two datasets. In this chapter we use these hierarchies (Figures 3.4(a) and 3.5(a)) for our smoothing experiments. In both the Botswana and the KSC hierarchies all pixels or data instances belong to leaf level classes, and hence these hierarchies fall under Scenario I.

These remote sensing datasets have also been used for evaluating techniques on the task of *Knowledge Transfer* [RGC05]. This task involves predicting class labels for pixels from a hyperspectral image that is spatially or temporally removed from the training data image. The motivation for this task is that it is very expensive to retrieve ground truth for hyperspectral images, and so transferring knowledge learned images of one area to another is

extremely useful. The difficulty in the task is due to the fact that a different area or a different time of image acquisition causes changes in hyperspectral signatures so that the distribution from which training and test instances are drawn are not the same. For the knowledge transfer task we use as a test dataset an image in the BOTSWANA dataset that is spatially removed from the training data. For the KSC dataset, our test dataset is an image that is spatially as well as temporally removed.

BIOINFORMATICS DATASET. This dataset was derived from the *Mousefunc* competition [2], which involved predicting the functional labels (from the Gene Ontology [AMBCea00]) for mouse genes using multiple sources of evidence. We experimented with the task of predicting labels for the 361 GO taxonomy terms with more than 30 genes associated with them. This left us with a dataset of 6790 genes. In terms of sources of evidence we restricted ourselves to using the 8537 protein annotation features from the Pfam [FMSB+06] and Interpro [MAea07] datasets as they were available for all the genes in our dataset. The GO taxonomy terms are arranged in a directed acyclic graph; we performed our smoothing over a tree structure embedded in the taxonomy, which was found using a bread-first traversal from the root. We call this tree structure along with the mouse genes as GOTAXONOMY. Since genes can belong to the internal nodes of our GOTAXONOMY, the bioinformatics dataset helps us evaluate our approach on taxonomies under Scenario II.

[2] `http://hugheslab.med.utoronto.ca/supplementary-data/mouseFunc_I`

4.4.2 Evaluation Measures

We report our results using a few different evaluation measures to highlight various aspects of the performance of our smoothing approach.

Measures for Scenario I and Scenario II. A standard measure we use is *classification accuracy* microaveraged over all the classes in the dataset. The classification accuracy is fraction of instances for which the true labels and the predicted labels match. Later in the chapter we will show that smoothing classifier outputs results in an increase in the number of correct predictions. While the classification accuracy showcases improvements in performance from the classifier's perspective, we can also analyze performance from each data instance's perspective.

Each instance has multiple true labels (a taxonomy node and all its parents on the path to the root). Hence, we can pose a question about whether our approach finds all true labels of a document successfully. We measure this performance in terms of precision and recall of true labels for each instance. *Precision* is the fraction of class labels predicted as positive by our approach that are actually true labels, while *recall* is the fraction of true class labels that are also predicted by our approach as positive. These precision-recall numbers can be summarized by their harmonic mean, which is also known as the *F–measure*. These measures are widely used in machine learning as well as information retrieval literature.

For each instance, the decision about which labels are positive and which negative is made by ordering the labels based on their classifier scores and then applying a threshold. Hence, we can obtain many different pairs of precision-recall values by varying the threshold. Typically as we lower the threshold

(make it easier for labels to be positive) precision drops and recall improves; the effect is exactly the opposite when the threshold is increased. While we can better study these precision-recall trade-offs by plotting all values for varying threshold, it becomes quickly unwieldy if we want to examine many parameterizations of our approach.

Receiver Operating Characteristics (ROC) graphs convey similar information as the precision-recall graphs. ROC graphs are two-dimensional graphs in which the *true positive rate* (the fraction of the true labels that were correctly predicted) is plotted on the Y-axis and the *false positive rate* (the fraction of the negative labels that were mistakenly predicted) is plotted on the X-axis. One of the characteristics of the ROC graphs that makes them attractive for collections like 20-newsgroup is the fact that the graphs are insensitive to the ratio of the number of positive and negative labels. A detailed description of the ROC graphs can be found in [Faw03].

Since we want to compare different algorithms, and often across different parameterizations, we need a quantitative measure of a given ROC graph. One that is often employed is the *area under the curve (AUC)* score of a ROC graph. A labeling that randomly predicts labels for documents has an expected AUC score of 0.5, while a perfect labeling system scores an AUC of 1.

Measures for Scenario III. Scenario III presents a different sort of challenge in the sense that each test instance has only a single true label. We are mostly interested in finding whether the single true label was found after the application of our smoothing approach. Hence, we use a couple of different measures for the evaluation.

The first measure is the *Precision* of the set of labels predicted as true

(before and after smoothing). Since there is only a single true label, this measure essentially depends on how many labels were returned as true and whether the actual true label was one among them. A class is considered predicted true if its score is above 0.5, and when no labels are predicted true the precision is considered to be 0. However, this measure doesn't discriminate between situations when the true label has the highest or lowest classifier score among the all the labels predicted true. To measure this aspect of performance, we compute the fraction of instances on which the true label also has the highest score. We call this measure the *AvgPrecision@1*, and it is equivalent to classification accuracy when only the label with the highest score is predicted as true.

4.4.3 Classification Algorithms

We trained one classifier for each node, internal as well as leaf-level, of the taxonomies. As classification algorithms we used the Support Vector Machine [Vap95] and Naive Bayes classifier [Mit97], both of which have been shown to be very effective in the automated classification task [RK04, PRG06]. In order to ensure that the outputs of distinct classifiers/nodes that are being smoothed are comparable to each other, we transformed the classifier outputs into posterior probabilities [WLW04, HWL06].

4.4.4 Parameter Settings

All results reported in this section were obtained after a 5-fold cross validation. Hence, in each fold 80% of the data was used for training. Out of that 10% was held out to be used as a validation set for adjusting parameters. Each performance number reported in this section is averaged over the 5 folds.

The variation in performance across folds was typically on the order of 1000^{ths} and so all improvements reported in this section are statistically significant.

The text classification and bioinformatics tasks SVM classifier was trained with a linear kernel and the "C" parameter was learned using the validations set by searching over {0.1,1,10} as possible values. These set of values have been seen to be effective in past work [PRG06]. For hyperspectral classification SVM classifier with RBF kernel was used. The "C" and "γ"parameter for this classifier were varied over {0.1,1,10} and {0.01,0.1,1,10} respectively and set using a validation dataset. Both classification algorithms were trained without any feature selection as both are fairly robust to overfitting (our classifier's performances were very close to those previously recorded in [PRG06]). We believe that while extensive feature selection might have improved performance by a couple of percentage points, these benefits would have been available to both the baseline as well as the smoothing approach and would not have qualitatively altered the results we discuss in our evaluation sections.

While performing smoothing on classifier outputs the node specific penalty values for each node is the same, unless indicated explicitly. Similarly, in all experiments in this chapter, the weights for all classifiers are set to 1.

Finally, while evaluating the results of classification and smoothing any measure, such as precision or recall, that needs a binary prediction uses 0.5 as the score threshold. This makes sense since our classifiers give us posterior probabilities for classes being the true label. Moreover, when multiple classes are predicted (our classification and smoothing approach both allow it) we choose one class randomly for our classification accuracy measure.

	No Smoothing	With Smoothing	
		$\gamma = 0$	$\gamma = \infty$
Classification Accuracy	0.74	0.734	0.86 (16.2% ↑)
AUC score	0.927	0.927	0.96 (3.6% ↑)
F-measure	0.87	0.872	0.91 (4.5% ↑)

Table 4.1: Dataset: TaxonomyI. Performance increases in SVM classifier through isotonic smoothing.

	No Smoothing	With Smoothing	
		$\gamma = 0$	$\gamma = \infty$
Classification Accuracy	0.67	0.67	0.76 (13.4% ↑)
AUC score	0.907	0.907	0.935 (3% ↑)
F-measure	0.828	0.828	0.853 (3% ↑)

Table 4.2: Dataset: TaxonomyI. Performance increases in Naive Bayes classifier through isotonic smoothing.

4.5 Evaluation on TaxonomyI under Scenario I

In TaxonomyI documents can only belong to leaf-level nodes, and under Scenario I, classifiers are trained to distinguish the content within a node from the content outside. Hence, we trained a classifier for each node such that the positive set of documents belonged to the leaf-level classes under the node, and the negative set of documents to all other leaf-level classes.

4.5.1 Classification Performance

First we discuss the performance of isotonic smoothing in terms of average classification accuracy per class and average AUC per document. In Figure 4.6 we plot both these measures against varying values of penalty. As we vary the penalty from 0 to ∞, the problem changes from simple isotonic regression to enforcing the strict monotonicity constraints. We can see from the plots that this progression of problems also translates into improved performance; both

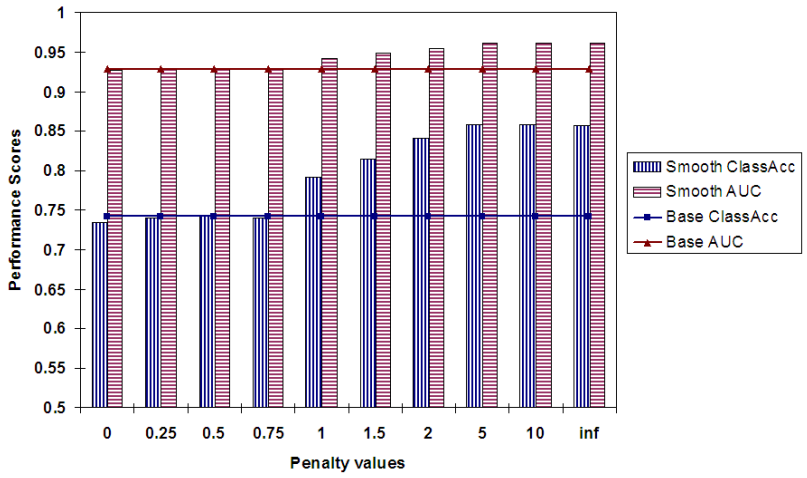


Figure 4.6: Dataset: TAXONOMYI. Performance of smoothing outputs of SVM classifiers as measured by classification accuracy and area under the ROC curve. The horizontal lines are the baseline scores obtained by the classifiers without the smoothing.

classification accuracy and AUC increase significantly with higher penalties. This shows that our method for smoothing classifier scores improves performance from the perspectives of both the classes as well as the documents.

Figure 4.7 plots the values for precision, recall, and F-measure averaged across all documents. Once again, these values are plotted against increasing penalty values. As we can see F-measure rises as penalties are increased and we move towards enforcing strict monotonicity constraints. These strict constraints ensure that the value of the parent is equal to the value of at least one of its children. This type of smoothing takes care of situations where leaf-level classes are mislabeled while their parent nodes are correctly labeled. In these cases, high penalty values make children conform to the parent's score correcting the error, resulting in increased precision and recall. However, strict

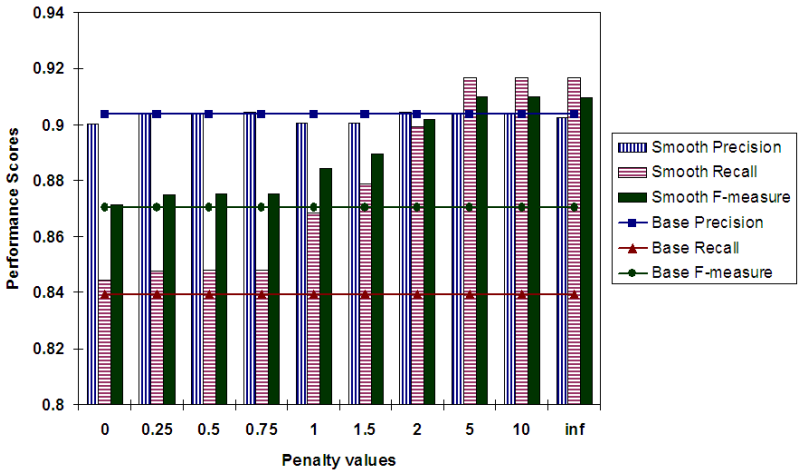


Figure 4.7: Dataset: TAXONOMYI. Performance of smoothing outputs of SVM classifiers as measured by increases in F-measure. The horizontal lines are the baseline scores obtained by the classifiers without the smoothing.

constraints can also sometime lead to some false positives - especially in shallow hierarchies like the 20-newsgroups - causing a decrease in precision. These trends are exactly what we observe in Figure 4.7. However, the increase in recall compensates for the slight decrease in precision by far, resulting in a higher overall F-measure score.

Since we are evaluating on a dataset that falls under Scenario I, and the strict monotonicity property was framed for just such a scenario, it makes sense that of all penalty values, $\gamma = \infty$ results in best performance. However, it is also interesting to observe the behavior of our dynamic program for low and high range of penalties. As we can see from Figure 4.6 and Figure 4.7 for penalty values between 0 and 1 there is hardly any change in performance from simple isotonic regression ($\gamma = 0$). This is because, in this range of penalty the cost to a node for deviating from its parent's smoothed value is less than

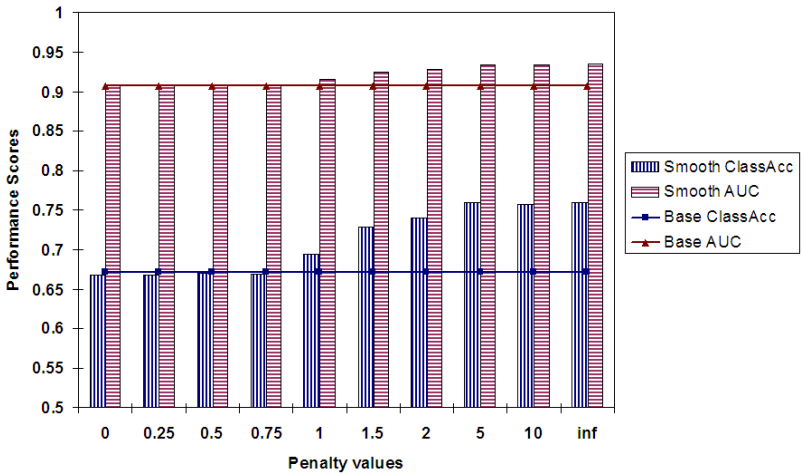


Figure 4.8: Dataset: TAXONOMYI. Performance of smoothing outputs of Naive Bayes classifiers as measured by classification accuracy and area under the ROC curve. The horizontal lines are the baseline scores obtained by the classifiers without the smoothing.

the cost from L_1 error for deviating from its own original value. Hence, the regularization term gets no chance to correct certain types of common errors, especially in shallow hierarchies like TAXONOMYI. Also, as penalty increases well above 1 ($\gamma \approx 5$) the increase in performance saturates. This is because once penalty becomes sufficiently large it becomes impossible to violate any strict monotonicity constraints (a node's value always equals the maximum of its children's value) and the smoothing behaves as if penalty was set to ∞.

We summarize the performance of our smoothing approach in Table 4.1 and Table 4.2. As we can see, over and above just classification, smoothing provides considerable gains in terms of classification accuracy over classes and precision-recall of labels for a document. According to our results, simple isotonic regression without penalties results in almost no improvement, high-

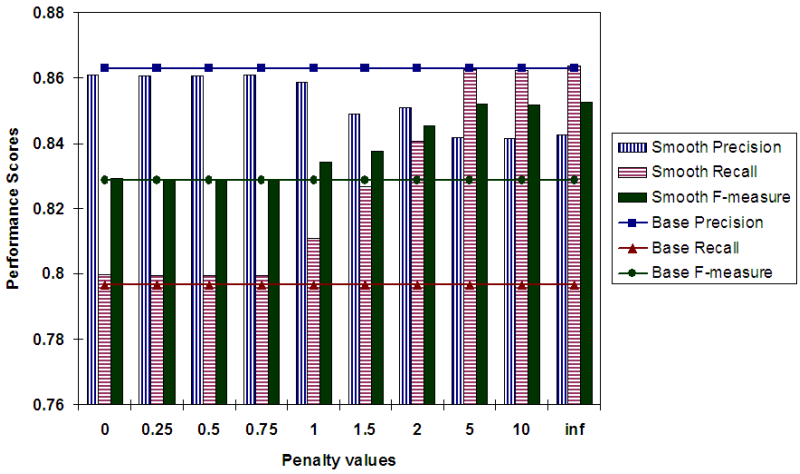


Figure 4.9: Dataset: TAXONOMYI. Performance of smoothing outputs of Naive Bayes classifiers as measured by increases in F-measure. The horizontal lines are the baseline scores obtained by the classifiers without the smoothing.

lighting that the gains are due to the regularization aspect of our approach.

4.5.2 The Effect of Missing Classifier Scores

In certain applications, especially those involving dynamic, fast changing, and vast corpora like the Web, we may not have the time or the data to train classifiers for each node (internal or leaf-level) in a hierarchical classification system. In such situations we can classify test instances for nodes with trained classifiers, while resorting to guessing at values for nodes without classifiers. In this section we evaluate whether smoothing the outputs of classifiers that have been trained can help us predict scores for classifiers that haven't been learned.

In order to simulate situations like these we randomly select a set fraction of nodes in our 20-newsgroups taxonomy that we don't train classifiers

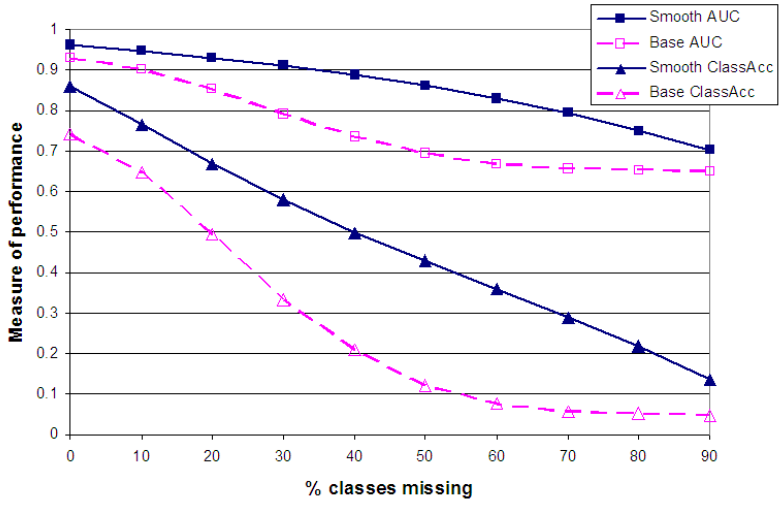
for. Then we apply our smoothing approach to the tree of nodes (some with missing values) and see if the smoothed scores of the nodes with missing values match the true labels. In our dynamic program the nodes with missing values are given a weight of zero so that they don't contribute to the L_1 error. The smoothing approach hence replaces the values of the missing nodes with whatever value that helps reduce the cost of isotonic smoothing. However, as the number of missing nodes increases the amount of information provided to the smoothing algorithm decreases and, therefore, we expect the performance of the whole system to also degrade.

In order to provide baseline performance we replace the missing classifier scores randomly with a true or a false - we bias the random predictions by the observed priors for the missing class. This class prior information is gathered from the training data for the class. In situations where classifier values are missing because of lack of training data, we can use other priors for these replacements (maybe average size of other classes in the data). Note that we didn't use the class prior information in the smoothing approach to predicting missing values.

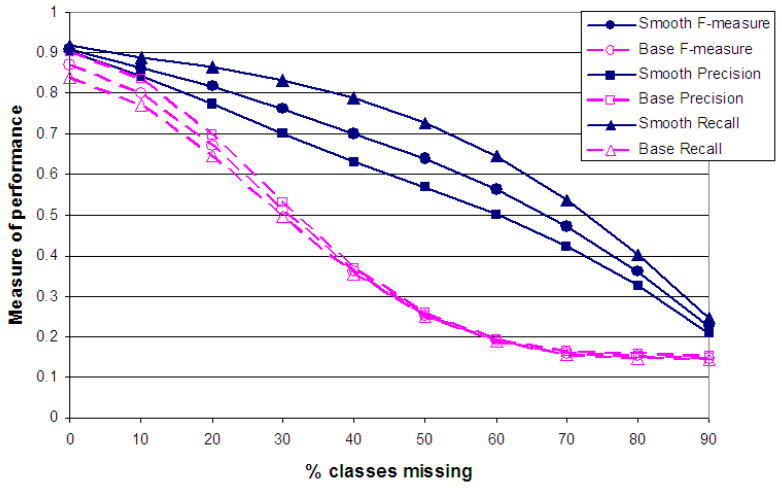
In Figure 4.10 we examine the performance of our system as the fraction of missing classes is increased (on the X-axis). The performance is measured in terms of the metrics mentioned earlier in this paper. As we can see from the plots, as the fraction of missing values increases the performance decreases. However, the decrease in the quality of smoothed outputs is far lower than the baseline predictions. Even though the smoothed and baseline predictions start with similar accuracy values, the difference between their performances grows dramatically with increasing number of missing values. For instance, in Figure 4.10(a) the classification accuracy after smoothing is 16% higher

than baseline with no missing values and this difference grows to 254% at 50% missing values. Similarly, the corresponding numbers for AUC are 3% and 24%. Figure 4.10(b) graphs performance in terms of precision, recall, and F-measure against varying amounts of missing values. Once again as the number of missing values increases, the difference in performance of smoothed outputs over baseline balloons: at 50% missing values, smoothing outperforms baseline by 155% in terms of F-measure.

The decrease in performance of baseline predictions is very dramatic at the beginning but after sufficient number of values are missing the effect of predicting with priors kicks in and the accuracy stabilizes. Since our smoothing approach does not use the knowledge of class priors, its performance never stops decreasing and at around 90% missing values the accuracy of smoothed scores and baseline scores is similar again. Hence, devising a well founded way to incorporate such prior information into the smoothing will improve the performance of our approach even more, especially in adverse conditions with many missing values.

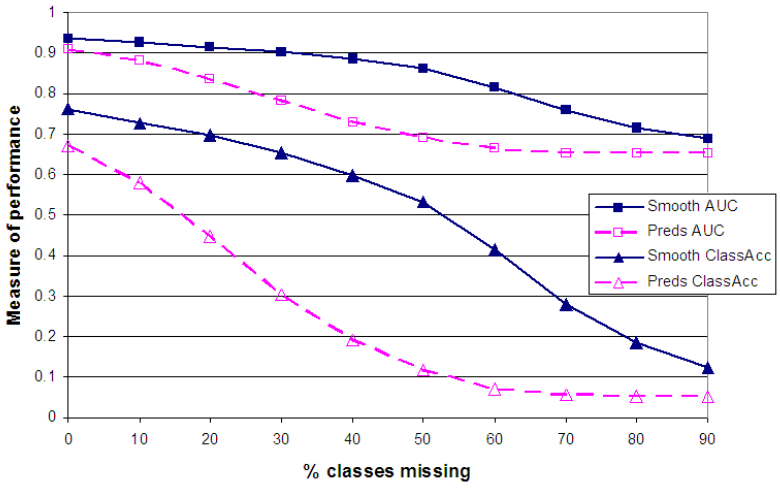


(a)

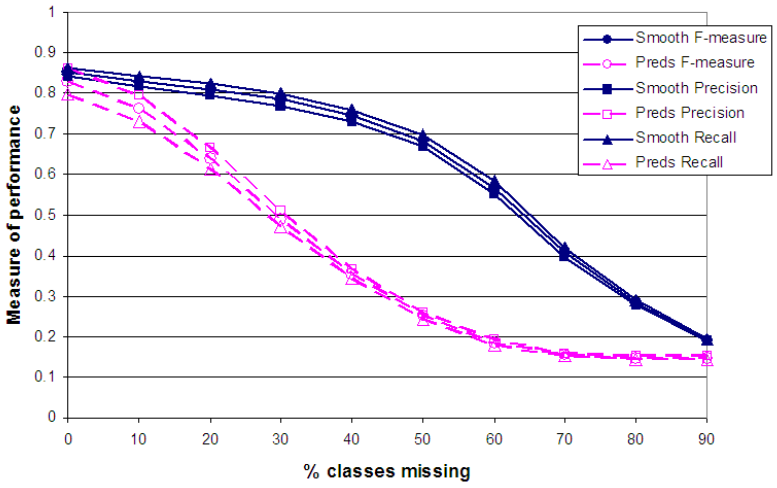


(b)

Figure 4.10: Dataset: TAXONOMYI. Performance with SVMs under Scenario I with missing values. Curves with pink no-fill shapes are performance of raw classification scores. Curves with blue solid shapes are performance after smoothing..



(a)



(b)

Figure 4.11: Dataset: TAXONOMYI. Performance with Naive Bayes classifiers under Scenario I with missing values. Curves with pink no-fill shapes are performance of raw classification scores. Curves with blue solid shapes are performance after smoothing.

	No Smoothing	With Smoothing	
		$\gamma = 0$	$\gamma = \infty$
Classification Accuracy	0.918	0.919	0.935 (1.9% ↑)
AUC score	0.973	0.973	0.979 (0.6% ↑)
F-measure	0.959	0.959	0.961 (0.2% ↑)

Table 4.3: Dataset: BOTSWANA. Performance increases in SVM classifier through isotonic smoothing.

	No Smoothing	With Smoothing	
		$\gamma = 0$	$\gamma = \infty$
Classification Accuracy	0.909	0.906	0.932 (2.5% ↑)
AUC score	0.917	0.972	0.979 (6.7% ↑)
F-measure	0.96	0.96	0.966 (0.6% ↑)

Table 4.4: Dataset: KSC. Performance increases in SVM classifier through isotonic smoothing.

4.6 Evaluation on Remote Sensing Data under Scenario I

In the previous section we evaluated our approach under the constraints on Scenario I on TAXONOMYI. In this section we will repeat the experiments on datasets from the hyperspectral analysis domain: the BOTSWANA and KSC datasets described earlier. Since all instances in these datasets must belong to leaf-level classes, these datasets also fall under Scenario I.

4.6.1 Classification Performance

The results of our smoothing experiments are summarized in Table 4.3 and 4.4. As we can see the accuracy in terms of various measures is increased very slightly when we performing smoothing in comparison to when we do not perform smoothing. The reason is that the accuracy of hyperspectral classification is extremely high even without smoothing. This leaves smoothing with

	No Smoothing	With Smoothing	
		$\gamma = 0$	$\gamma = \infty$
Classification Accuracy	0.677	0.676	0.732 (8.1% ↑)
AUC score	0.876	0.876	0.902 (3% ↑)
F-measure	0.787	0.785	0.806 (2.4% ↑)

Table 4.5: Dataset: BOTSWANA (Knowledge Transfer). Performance increases in SVM classifier through isotonic smoothing.

	No Smoothing	With Smoothing	
		$\gamma = 0$	$\gamma = \infty$
Classification Accuracy	0.584	0.588	0.62 (6.2% ↑)
AUC score	0.83	0.83	0.85 (2.4% ↑)
F-measure	0.737	0.738	0.747 (1.4% ↑)

Table 4.6: Dataset: KSC (Knowledge Transfer). Performance increases in SVM classifier through isotonic smoothing.

very few errors to correct. In fact, while increases in classification accuracy seem very small, 2% and 2.5% for BOTSWANA and KSC respectively, these results represent a $> 10\%$ reduction in classification error.

The variations in accuracy as we change the value of the penalty parameter is similar to what we expect for a dataset that falls under Scenario I: the accuracy in all measures increases as we increase the penalty parameter. The general trends are similar to the plots in Figures 4.12 and 4.13 for BOTSWANA and Figures 4.14 and 4.15 for KSC.

4.6.2 Evaluation on the Knowledge Transfer Task

The task of predicting class labels for instances from an unseen test dataset after learning from ground truth available for a distinct training dataset is called Knowledge Transfer. The motivation for this problem in the current scenario is the high cost of obtaining ground truth for hyperspectral data.

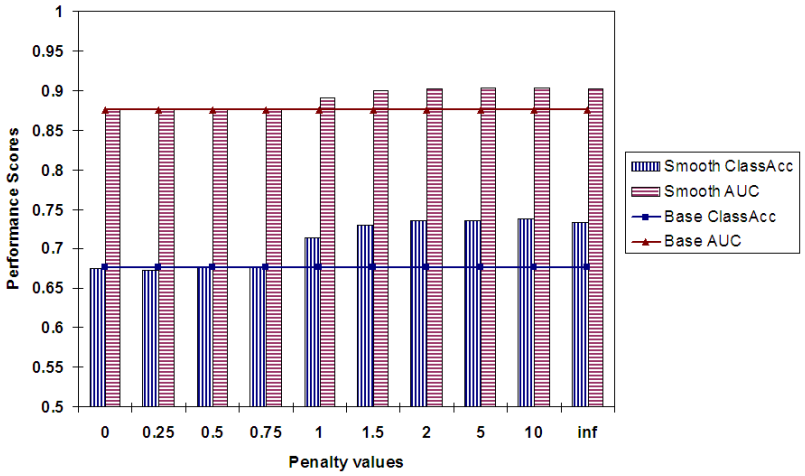


Figure 4.12: Dataset: BOTSWANA (Knowledge Transfer). Performance of smoothing outputs of SVM classifiers as measured by classification accuracy and area under the ROC curve. The horizontal lines are the baseline scores obtained by the classifiers without the smoothing.

Moreover, additional factors such as location and time of image acquisition cause hyperspectral signatures for the same class to vary. Hence, knowledge transfer techniques can help us leverage available labeled data to classify pixels in new hyperspectral images.

In this section we use our smoothing approach to apply the hierarchical classifier learned on training data to independent test sets in our two remote sensing datasets. For the BOTSWANA dataset the independent test set is spatially removed while for the KSC dataset the independent test set is spatially as well as temporally different. The hope is that while the signatures for any class over the train and test set would be different, the smoothing approach would be able to correct some of the misclassification by taking into account the membership scores of neighboring classes in the taxonomy.

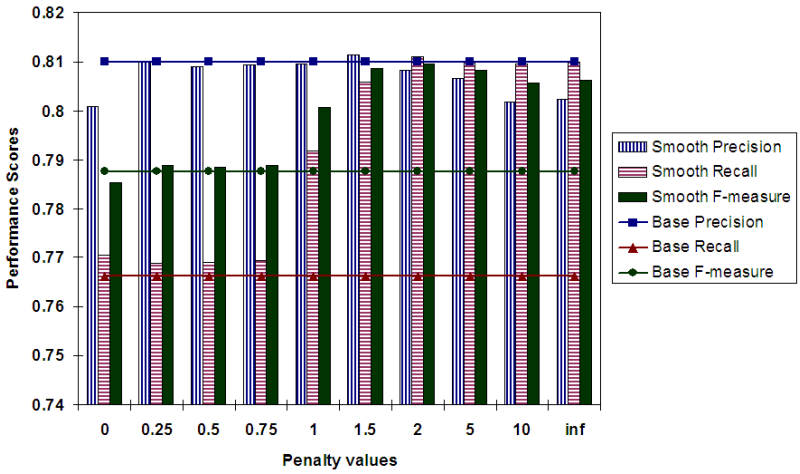


Figure 4.13: Dataset: BOTSWANA (Knowledge Transfer). Performance of smoothing outputs of SVM classifiers as measured by increases in F-measure. The horizontal lines are the baseline scores obtained by the classifiers without the smoothing.

The results of our experiments on knowledge transfer are summarized in Tables 4.5 and 4.6. First observation to make is that the knowledge transfer is a much harder problem than standard hyperspectral classification. For both the BOTSWANA and KSC the classification accuracy in the standard setting was higher than 90%. In the knowledge transfer setting this value drops to 67% for BOTSWANA and 58% for KSC. However, upon smoothing we see significant gains in classification performance after smoothing in terms of all measures. The increase in classification accuracy is 8% in the case of BOTSWANA dataset and around 6.2% for the KSC dataset.

The variation in accuracy measures while varying the penalty parameter value is plotted in Figures 4.12 and 4.13 for BOTSWANA and Figures 4.14 and 4.15 for KSC. As we can see increasing the penalty parameter results in an

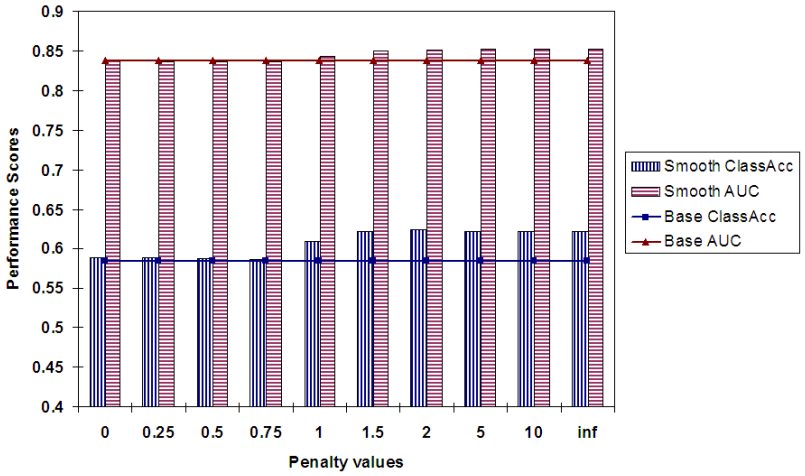


Figure 4.14: Dataset: KSC (Knowledge Transfer). Performance of smoothing outputs of SVM classifiers as measured by classification accuracy and area under the ROC curve. The horizontal lines are the baseline scores obtained by the classifiers without the smoothing.

increase in the classification performance. This agrees with our intuition that strictly enforcing strict monotonicity constraints results in the best smoothing for datasets which correspond to Scenario I.

4.6.3 The Effect of Missing Classifier Scores

In Section 4.5.2 we investigated the effect of missing classifier values on the classification performance before and after smoothing. We showed that as the fraction of nodes in the taxonomy with missing values increases, the classification performance drops. However, the drop after smoothing is much lesser than the drop before smoothing. In this section we perform the same experiment for the remote sensing datasets.

As before in our current experiments we randomly drop the classification

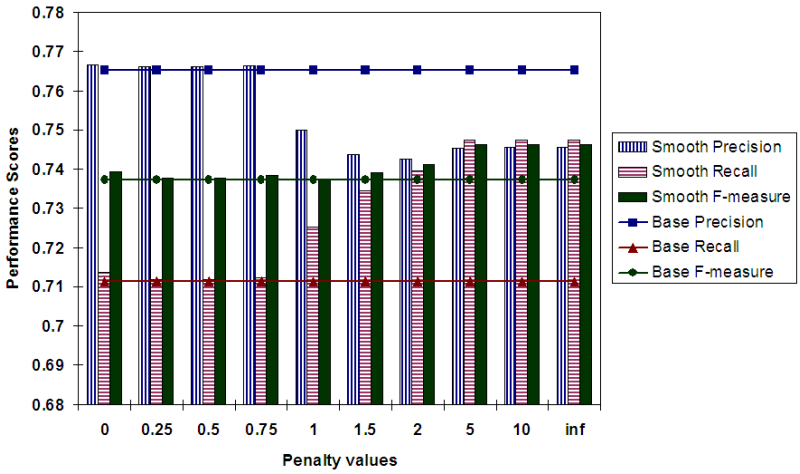
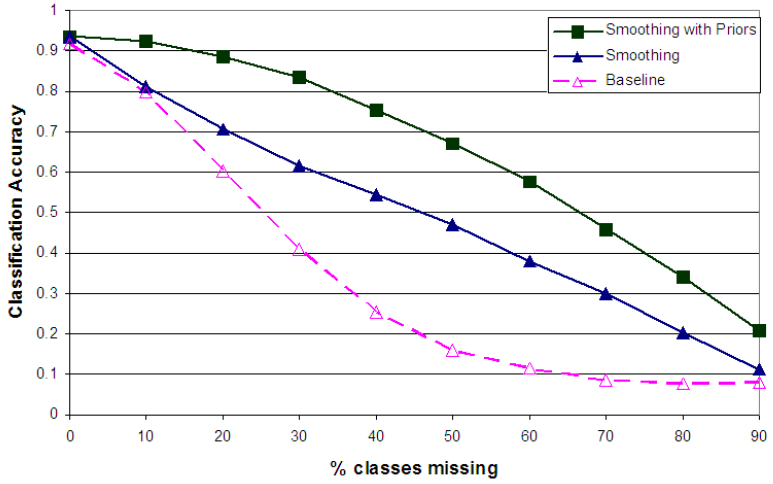


Figure 4.15: Dataset: KSC (Knowledge Transfer). Performance of smoothing outputs of SVM classifiers as measured by increases in F-measure. The horizontal lines are the baseline scores obtained by the classifiers without the smoothing.

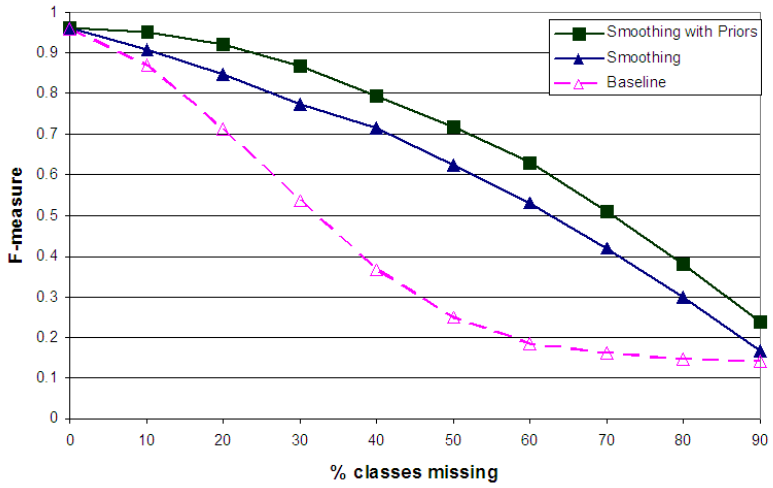
scores for a fraction of class nodes. These missing values are then predicted using the class priors in the case of the baseline (no smoothing). The smoothing approach sets the missing value nodes to zero weight before running regularized isotonic regression. The final smoothed score for the missing nodes is used as the predicted value. In Figures 4.16 and 4.17 we plot the accuracy measures for the baseline as well as the smoothing approaches in relation to the fraction of missing classifier values. As we can see, in terms of both the classification accuracy and f-measure the performance drops significantly for the baseline as the fraction of missing classifier scores increases. This drop, however, is much gentler in the case of smoothing. This results in significant difference in performance at 50% missing values: for the Botswana dataset the classification accuracy and f-measure upon smoothing are 210% and 160% higher respectively than the baseline. Similar results also hold for the KSC

dataset.

We also performed an experiment with the smoothing approach in which we used the class prior to set the weight for each missing node. Here each missing value was replaced by a zero value but weighted according to the prior probability that the instance did not belong to the node. Hence, while predicting the smoothed score for a node with a missing value our approach would take into account the neighboring nodes' values as well as the prior probability of the missing node taking value zero. The results from this experimented are plotted in Figures 4.16 and 4.17 as the curve labeled "Smoothing with Priors". As we can see from the plots smoothing while taking into account prior information gives a significant gain in classification performance over and above simple smoothing. In fact at 50% missing classifier scores, smoothing with priors gives us a performance boost of 42% in classification accuracy and 16% in f-measure over and above standard smoothing in the case of BOTSWANA dataset. Similar improvements in performance are seen for the KSC dataset.

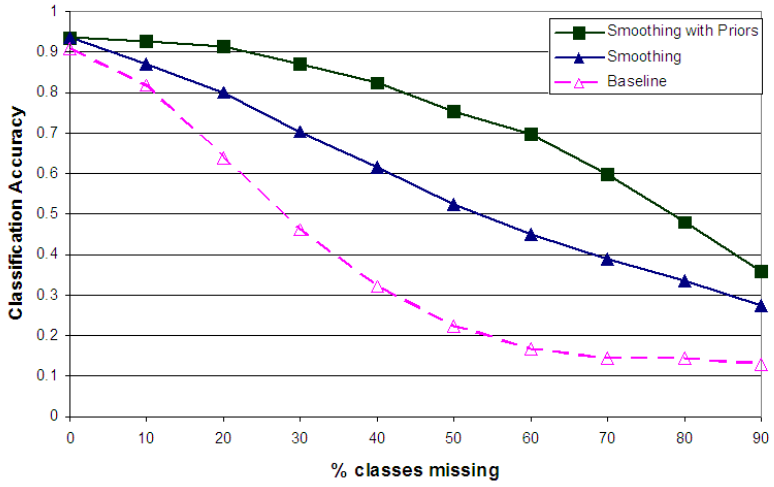


(a)

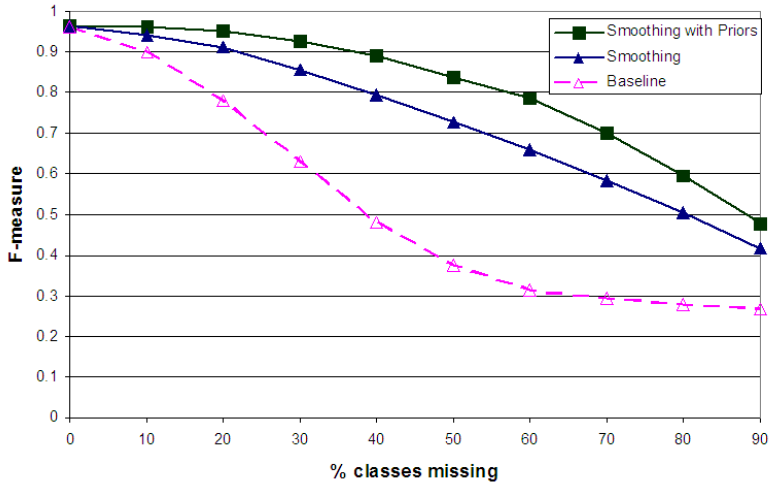


(b)

Figure 4.16: Dataset: BOTSWANA. Performance with SVMs under Scenario I with missing values. Curves with pink no-fill shapes are performance of raw classification scores. Curves with blue solid shapes are performance after smoothing..



(a)



(b)

Figure 4.17: Dataset: KSC. Performance with SVMs under Scenario I with missing values. Curves with pink no-fill shapes are performance of raw classification scores. Curves with blue solid shapes are performance after smoothing..

	No Smoothing	With Smoothing	
		$\gamma = 0$	$\gamma = \infty$
Classification Accuracy	0.26	0.32 (23% ↑)	0.122
AUC score	0.78	0.79 (1.3% ↑)	0.78
F-measure	0.535	0.565 (5.6% ↑)	0.50

Table 4.7: Dataset: GOTaxonomy. Performance increases in SVM classifier through isotonic smoothing.

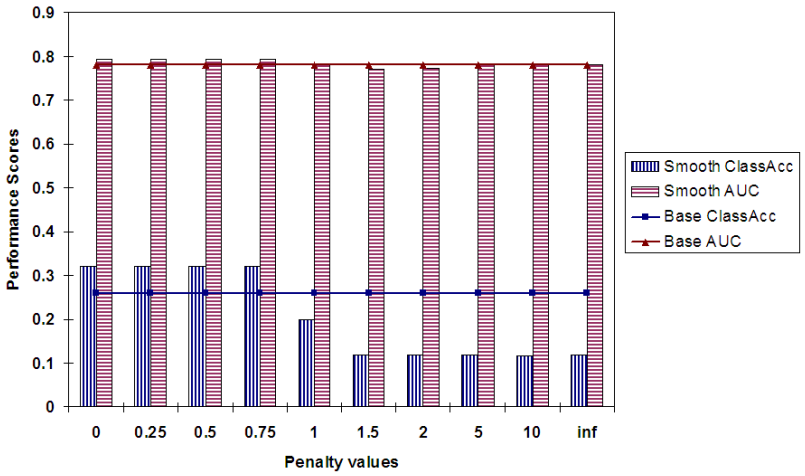


Figure 4.18: Dataset: GOTaxonomy. Performance of smoothing outputs of SVM classifiers as measured by classification accuracy and area under the ROC curve. The horizontal lines are the baseline scores obtained by the classifiers without the smoothing.

4.7 Evaluation on GOTaxonomy under Scenario II

We trained one classifier for each node, internal as well as leaf-level, of GOTaxonomy. In this dataset, genes can belong to any node of the tree. Hence, while training a classifier for a node the positive set of genes came from all classes in the subtree of the node. All the genes from classes outside the

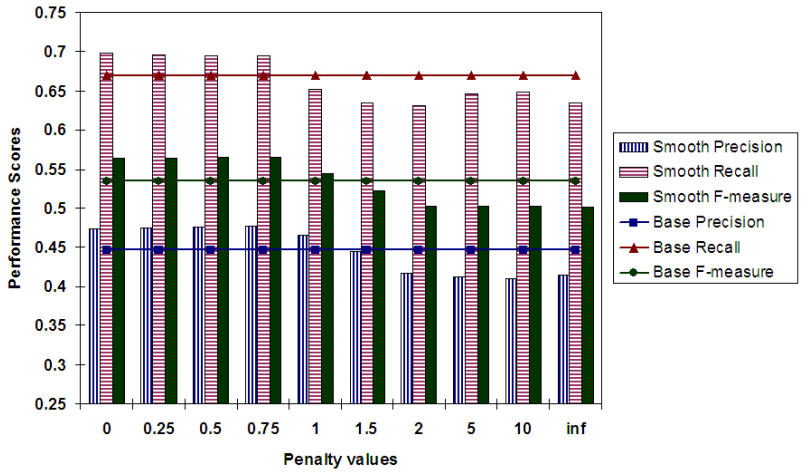


Figure 4.19: Dataset: GOTaxonomy. Performance of smoothing outputs of SVM classifiers as measured by increases in F-measure. The horizontal lines are the baseline scores obtained by the classifiers without the smoothing.

subtree of the node formed the negative set.

Here while smoothing we cannot assume that a node's score must equal the maximum of its children's score. The document being classified may not belong to any of the children and hence a node's score can be larger than all of its children's scores. In such a scenario the role of regularization is more subtle than in previous experiments. Very aggressive regularization will enforce strict monotonicity constraints which do not match the problem. Hence, the regularization penalty will have to be large enough to undo classification errors but also low enough to leave room for legitimate cases where documents are not classified into any child.

In fact, we observe that simple isotonic regression ($\gamma = 0$) is sufficient to correct classification errors. We summarize the results of our experiments in Table 4.7. From the table we can see that isotonic smoothing results in sig-

nificant gains in classification accuracy (23%) and f-measure (5.6%) over the baseline of simple classification. In Figure 4.18 and Figure 4.19 we plot the performance of our smoothing approach with varying values of the penalty parameter. As we can see smoothing the outputs of classifiers via isotonic regression gives us significant improvements in terms of all measures. Moreover, any increase in penalty above $\gamma = 0$ causes a decrease in classification performance. This ties in with our intuition about the performance of our smoothing approach on datasets under Scenario II.

4.8 Evaluation on TAXONOMYII under Scenario III

Under Scenario III classifiers are trained to distinguish documents belonging to a node from those belonging to all other nodes. Hence, each instance must have only one true class label. Therefore, as mentioned in Section 4.4 we use Precision and AvgPrecision@1 to evaluate our smoothing approach.

First let us consider the performance of smoothing on instances for which the original classifiers predict more than one class as true label. Instances with only one class predicted as true will not be affected much by our approach since these set of scores often already satisfy unimodality constraints. Hence, we measure how well our smoothing approach repairs classification scores that predict more than one class as true. These numbers are plotted in Figures 4.20 and 4.21.

In Figure 4.20(a) we plot the precision of the smoothed solutions against increasing penalty values. The horizontal line is the baseline performance of classification without any smoothing. As we can see, for certain values of penalty the precision of the smoothed scores is much higher than the original classifier scores. For extreme values of penalty, the accuracy of the smoothing

procedure drops. The amount of improvement made by smoothing all instances as opposed to only instances with two or more classes predicted as true labels is shown in Figure 4.20(b). As we can see the performance of smoothing is much higher over instances that have multiple predicted labels in the original score. This is because smoothing with regularization tends to select one of the predicted labels and reduces the others to zero. This has an effect of improving precision if the correct label is chosen.

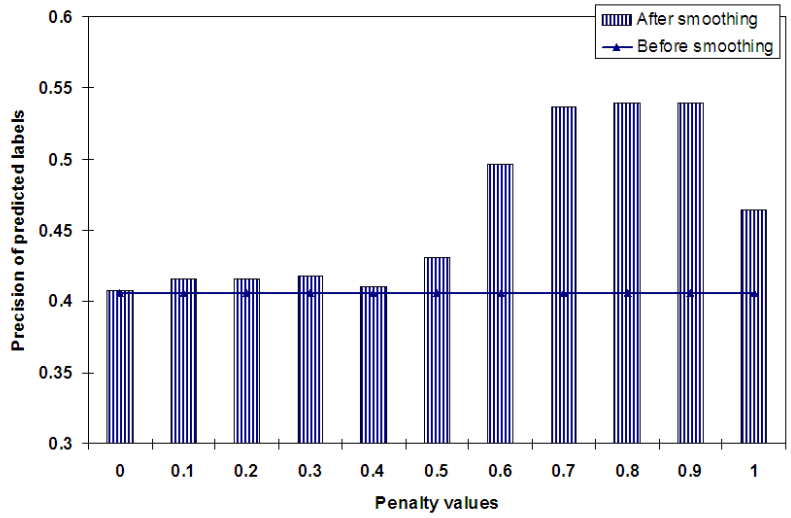
This observation can be confirmed using the AvgPrecision@1 measure. We plot this measure in Figure 4.21(a) against increasing penalty values. We can see that at the appropriate values of penalty, the chances that the true label has the highest score is around 20% higher after we smooth the original classifier scores. Once again the improvement is higher for documents that have more than one class predicted as true in the original classification scores. This is because when the original scores for a document have only one predicted label, then even if this label is incorrect, the smoothing cannot find the true label, reducing the overall improvement in the test data. Smoothing is most useful in finding the true label from among multiple predicted ones.

Effect of Constraining Cross-over Nodes to Value 1.

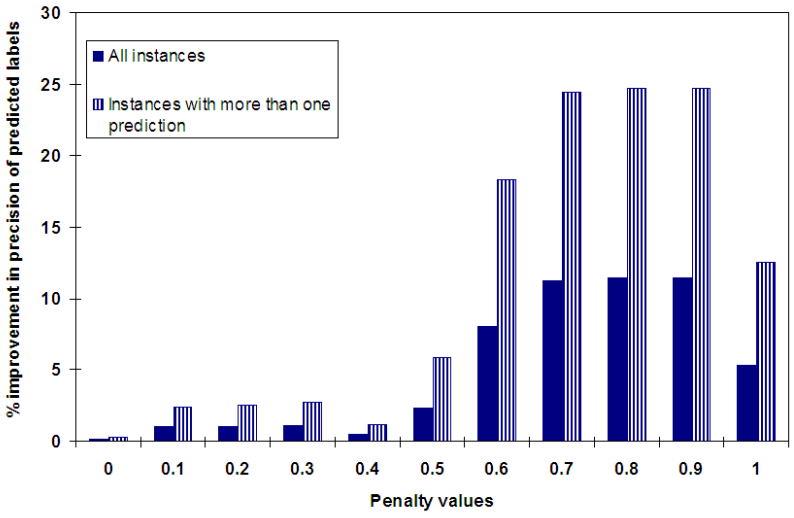
In a bid to reduce false positive classification errors we make an slight modification to our formulation. We now require that every cross-over node have a smoothed score of 1. The intuition behind this approach is that forcing cross-over nodes to have a final value of 1, we make it too expensive for all but the true label to have a score 1. For example, if a prospective cross-over node has an optimal smoothed score of 0.6, now that it is forced to have a score of 1, it might be cheaper to reduce the score to 0, since moving it to 1 will incur an extra cost from the L_1 error and also extra penalty-based cost.

The possible ill-effects of this formulation is that it might result in no node qualifying as a cross-over node. Also if multiple nodes do qualify as cross-over nodes, they will all have the same score of 1 and our AvgPrecision@1 metric might suffer.

The results from this experiment are plotted in Figure 4.22. As we can see while there is no appreciable difference in the best value of precision attained, constraining cross-over nodes to a smoothed score of 1 increases the range of penalties over which smoothing performs well. Unlike results in Figure 4.20, our new approach shows appreciable increases in accuracy over the whole range of penalties. This is because as explained above the constraint of forcing cross-over nodes to 1 acts as a penalty term increasing the cost of predicting classes with mediocre scores as true. However, as predicted the new constraint does cause a slight reduction in the AvgPrecision@1 measure as multiple class that are predicted true all have score 1 and we have to choose one among them randomly.

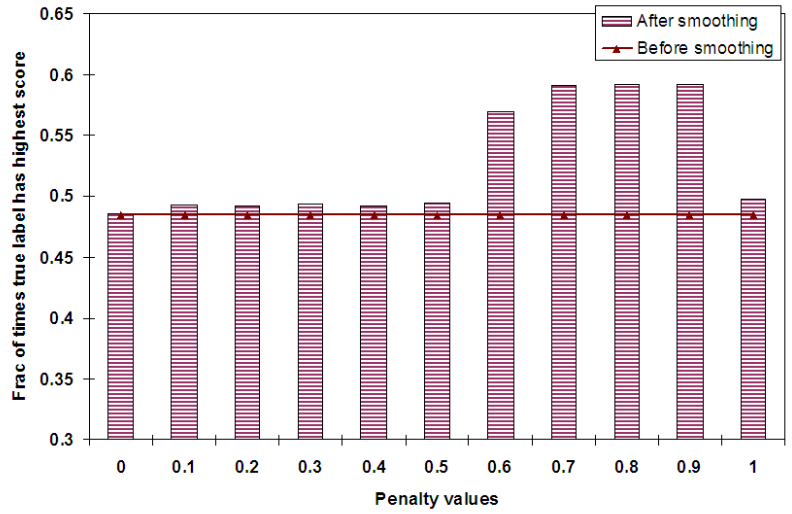


(a) Precision with varying penalty

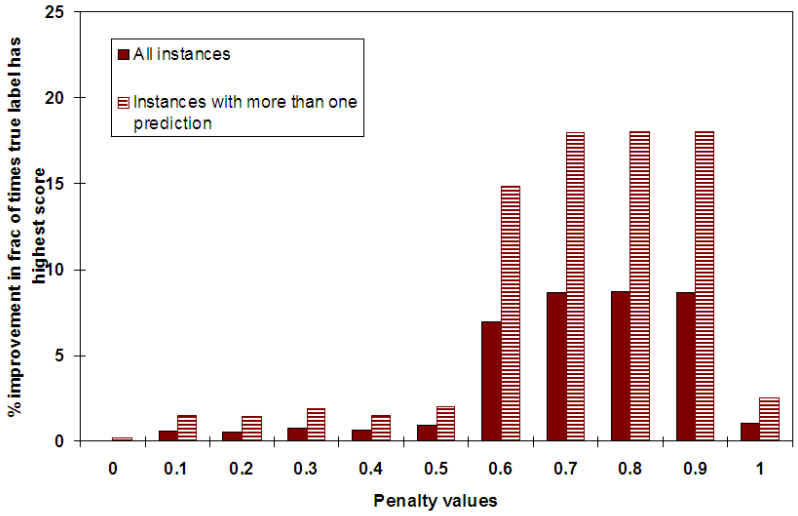


(b) Improvement in precision with varying penalty

Figure 4.20: Precision of classes predictions as true labels before and after smoothing with SVM classifiers on TaxonomyII under Scenario III.

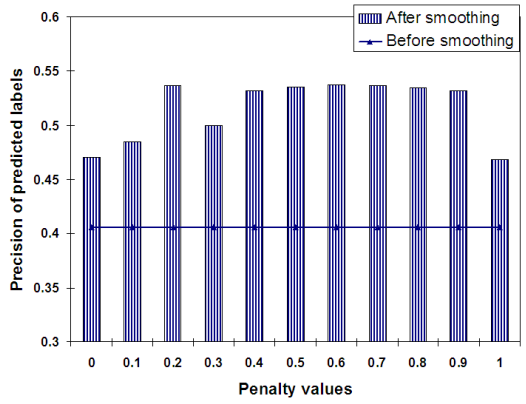


(a) AvgPrecision@1 with varying penalty

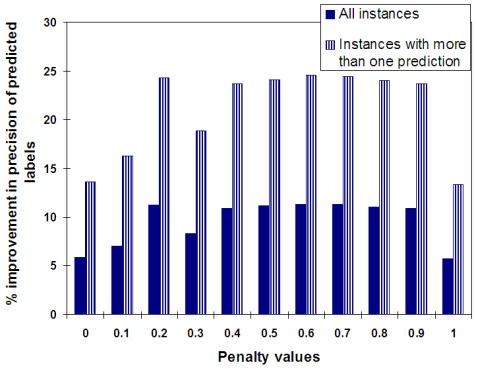


(b) Improvement in AvgPrecision@1 with varying penalty

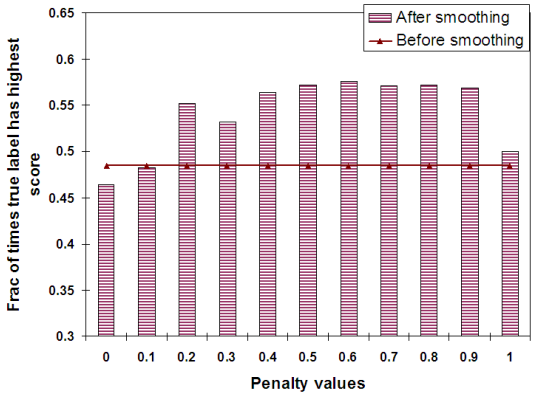
Figure 4.21: AvgPrecision@1 of classes predictions as true labels before and after smoothing with SVM classifiers on TaxonomyII under Scenario III.



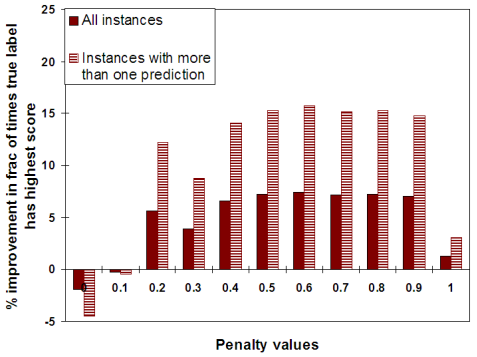
(a) Precision



(b) Improvement in Precision



(c) AvgPrecision@1



(d) Improvement in AvgPrecision@1

Figure 4.22: Performance with SVM classifiers on TAXONOMYII under Scenario III when constraining the cross-over nodes to have a smoothed value of 1.

4.9 Discussion and Conclusions

Our Approach vs Top-down Classification.

The standard top-down classification model involves learning a classifier for each internal node of the taxonomy, which directs a test instance to one of the children nodes. A test instance is then classified down the tree, starting from the classifier at the root node, until it arrives at a leaf node. In contrast to this model, in our work in this chapter we have proposed an alternate way to perform hierarchical classification. Our system learns a one-vs-all classifier for each internal node in the taxonomy. A test instance is then classified by each one-vs-all classifier and the classifier scores are smoothed to yield the final membership scores.

The top-down classification model has a serious problem in that errors in classification are propagated down the tree. Once an error is made at an internal node classifier, the test instance cannot arrive at the correct leaf node. In contrast, as we have shown empirically in this chapter, our classification approach can be used to correct many of the errors made by classifiers. A second drawback of top-down classification is its inability to deal with missing classifiers at internal nodes. Without a classifier to guide us at a node, we wouldn't know which child node to propagate the test instance to. Classifiers could be missing because of lack to time to train them or abrupt changes in taxonomy structure. Once again, as we have seen in this chapter, our approach deals with missing classifiers very gracefully.

For the flexibility of recovering from classification errors and missing classifiers, our approach pays in terms of time needed to classify a test instance. Our method has to classify a test instance through each classifier learned: this

takes time linear in the size of the tree. A top-down classifier has to only classify an instance down one path to a leaf node: this takes time on the order of log of the size of the tree. However, our approach can be significantly sped-up by indexing classifiers using approaches similar to [ABP06, PC06], and through parallelization techniques such as the map-reduce framework [DG04a]. These speed-ups are not afforded to the top-down classification model because classification at each successive node can only be done after the decision from the classification at parent node is known.

Enforcing Classifier Constraints while Training.

There is some recent work on enforcing constraints from hierarchies or general graphs within classifiers. Some of this work has been cited in Section 2.4. In our approach we enforce constraints as a post-processing step to classification. This might seem like a slightly greedy approach especially if the integrated classifier does a very good job of reducing classification error while satisfying constraints. However, these classifiers typically are very complex and need large amounts of training data for learning. Moreover, while they are trained to satisfy hierarchical constraints on training data, they might not do so while classifying unseen test instances. Finally, it is often the case that different classification algorithms work well for different domains. Hence, we might need to modify a different classification algorithm to enforce constraints for each domain that we encounter.

In contrast our approach can work with any off-the-shelf classifier since it is only concerned with smoothing final classification scores. These classifiers can be as simple or as complex as each application domain demands and our approach will work with all of them. Finally, as our approach smooths output

classifier scores for test instances, the final scores will always satisfy the hierarchical constraints. That being said our approach can also benefit from some information about the training data (such as class priors etc). Integration of training data characteristics into the smoothing process for increased accuracy is part of the future direction for this work.

Experiments with the "Best" Classifiers.

In this chapter we perform experiments with SVMs used as classifiers at each internal node of the taxonomy. While we tuned the classifiers to obtain the extremely good accuracy, in particular to mimic classifier performances in Chapter 3, we didn't do feature selection or use techniques like shrinkage [MRMN98] to achieve the absolute best performance.

The reason is that our central goal in this chapter is to show that smoothing improve classification accuracy and not to achieve the absolute best classification accuracy. We believe that using slightly better classifiers will not change the qualitative nature of our results. Of course, if the classifiers already have very good accuracy then the improvements through smoothing are smaller; this can be seen in the results in Tables 4.3 and 4.4. However, as we have shown in our experiments, in such situations the smoothing process still results in modest improvements in classification performance.

Concluding Remarks.

In conclusion, in this chapter we demonstrated how hierarchical relationships between classes in a taxonomy can be translated into constraints on the outputs of the classifiers learned over them. The problem of smoothing the

classifier outputs to satisfy these constraints was formulated as novel optimization problems that we call regularized isotonic and unimodal regressions. We gave efficient algorithms to find solve these problems exactly. Using real-world datasets, we also showed that performing smoothing as a post-processing step after classification can significantly improve accuracy.

Chapter 5

Page-level Template Detection via Isotonic Smoothing

5.1 Introduction

Template material is common content or formatting that appears on multiple pages of a site. Almost all pages on the web today contain template material to a greater or lesser extent. Common examples include navigation sidebars containing links along the left or right side of the page; corporate logos that appear in a uniform location on all pages; standard background colors or styles; headers or drop-down menus along the top with links to products, locations, and contact information; banner advertisements; and footers containing links to homepages or copyright information. The template mechanism is used to support many purposes, particularly navigation, presentation, and branding.

There is no single dominant mechanism by which templates appear in web pages. At one extreme is web site design software that allows a user to single-handedly manage a medium-size web site, formally editing and applying templates to groups of pages as necessary. At the other extreme is the personal web site in which the owner copies the same fragment of HTML from one page to the next in order to provide a uniform look and feel, and diligently avoids the overhead of changing templates too frequently. Other familiar mechanisms include application servers that implement page templates in code; dynamically generated pages that wrap content into a template; portal servers that arrange content into cells with arbitrary content around them; and content management systems that manage templates.

Templates are a Cause for Concern.

On today's web, templates are a significant cause for concern. As we show below, templates are responsible for roughly 40–50% of the content on the web. The repeated occurrence across a website of content purporting to be original misleads search engines, page classification, clustering, link analysis, and other applications providing advanced text analysis on web content. Furthermore, an accurate assessment of whether the content of a page has changed is critical in several applications. First, crawlers may behave more efficiently based on knowledge of the change rate of pages. Second, alerting applications should not alert users due to template changes. And third, any applications that support trending over web data should not be misled into believing that a site has changed significantly due to a template modification.

Further, we show that the proportion of templated text on the web has been growing consistently for nearly a decade, and thus all these applications will need even greater awareness of templates in coming years.

On the other hand, effectively recognizing templates brings several advantages. Once extracted, they can be used to identify key pages on a website, such as the products page of a company, or the entry point or each school of a university. Pages that share a template can also be grouped together into a cluster that may not be apparent using other mechanisms. Finally, once templates have been identified, any analysis algorithms can realize a nearly two-fold improvement in storage and processing requirements, by exploiting redundancy.

Automated Detection of Templates.

Unfortunately, no simple and completely effective algorithm for template extraction is known. Techniques for the problem fall into two families. *Local* techniques operate on an individual page without reference to other pages, while *global* techniques consider a family of pages together and exploit the property that templates occur many times. Purely local techniques are effective at stripping away certain kinds of banners and navigational material, but these techniques are only heuristics and are somewhat error-prone as the web changes. It is quite common, for example, for certain paragraphs of textual content in the middle of a page to be templates—detecting this without reference to global information is essentially impossible. Global techniques, on the other hand, achieve very high precision, since its relatively rare that content that repeats many times is not a template. However, there do exist templates that don't manifest themselves via repetition and global approaches typically miss these.

In this chapter, we present two global techniques for template detection and one that operates with only information local to a webpage. We also describe how we can exploit the output of our global approaches to obtain

better page-level template detection. Finally, we describe a novel approach to smoothing values over tree structures that can be used to further improve the accuracy of our system.

Study of Volume and Evolution of Templates.

Using our global template detection algorithms we perform two studies to investigate the nature, the volume, and the evolution of templates on today's web. In the first, we randomly sample two hundred sites from a large crawl containing approximately fifty million sites and two billion pages. We hand-classify this site-level sample into seven categories such as personal sites, catalog sites, community sites and so on. We then analyze the nature and prevalence of templates within sites belonging to each category. For each site, we create a uniform random sample of the crawled content from the site of approximately two hundred pages, and study the commonality of templating across this sample; thus, our study captures only templates that occur on some small fraction of the pages on a site.

In our second study, we consider the evolution of template usage. Using crawls from the Internet Archive [Kah], we study multiple snapshots of pages from two collections: the hand-classified sites from our first study and the sites studied by Ntoulas et al. [NCO04]. We gathered approximately 72K page instances during this study, over 1380 snapshots of time. According to our studies, the volume of templated material is 40–50% of the total bytes on the web, and this quantity has been growing at the rate of 6–8% over the last 8 years with no signs of slowing.

Accuracy and Impact of Template Detection.

To validate our algorithms we perform an extensive set of experiments measuring their accuracy and their impact on third-party applications. In terms of detecting content within templates, our algorithm achieves a F-measure in excess of 0.65 for text and 0.75 for links on a human-labeled test set. Within these results we isolate the impacts of using just the local information to score page-section as templates, and of smoothing the template scores via regularized isotonic regression. As for the impact of template detection, we show that removing templates as a pre-processing step boosts the accuracy of standard web mining tasks on our datasets, by as much as 140% on duplicate detection and 18% on webpage classification. We also compare the performances of our global and local approaches and identify conditions under which one is preferred over the other.

5.2 Site-level Template Detection

In this section we describe our global approaches to template detection. As mentioned before, global approaches rely on the property that template material is often repeated across multiple pages on a website. Hence, through the rest of the chapter, we refer to these approaches as SiteLevel.

We consider two algorithms, one based on the DOM structure of the web page, and the other based on syntactic sequences of characters. DOM-based algorithms provide efficient representations (as a typical page may contain 10-20K of content but only around 100 DOM nodes), and perform well on hierarchical templating schemes using table layouts. Text-based algorithms, on the other hand, are amenable to a class of probabilistic speedups, and

perform well in jsp-style templates, as the material in the template need not correspond strictly to the DOM tree.

5.2.1 DOM-based algorithm

This algorithm uses the DOM structure of the pages on a website by searching for nodes of the DOM tree that are repeated across multiple pages on the website. It is based on the work of Rajagopalan and Bar-Yossef [BYR02], and Yi and Liu [YL03], but contains simplifications from those techniques.

Construction of the DOM tree for a page requires that the page first be cleaned. This is a substantial problem on the Web due to the diverse set of languages, authors, and tools; and also due to the excellent efforts of web browsers to render badly-formed HTML correctly. We modified an existing HTML parsing and cleaning library called HyParSuite [Cha] to address this problem, maintaining offsets to nodes in the original unclean page so that the links and text inside and outside templates may be extracted later. The algorithm then operates in two passes.

First Pass. The first pass iterates over all the pages in the website and dumps information about all the DOM nodes in a page. This information consists of the hash of the content of the node (template-hash) and the start and end offsets into the original file. The template-hash is calculated using the HTML content within the node's start and end tags and DOM node's name, attributes, and their values. For example, consider the following HTML substructure:

```
<td><a href='...'>Click here</a> to visit ...</td>
```

This structure consists of four HTML nodes. The top-most node is the `<td>` node. The template-hash of this node will be computed from the entire HTML string. The `<a>` tag is a child of the `<td>` node and its template-hash will be calculated using the the contents between the `<a>` and `</a>` tags inclusive of the tags. Text nodes are constructed for stretches of text in HTML files and the above example consists of two text nodes.

Thus the template-hash is a compressed representation of the HTML tag and its contents. Counting the number of times a template-hash is encountered in a website tells us the number of times a specific HTML node is seen. Hence, the first pass keeps track of the number of times each template-hash has been seen in the website and passes this information to the second pass.

Second Pass. The second pass then scans this information and computes a set of template-nodes for each page. A HTML node in a particular page is said to be a template-node if the following conditions are met: first, the occurrence count of the node's template-hash is within a specified threshold; and second, the node is not a child of any other template-node.

Sibling template nodes are then coalesced to produce the templates on a page. The coalescing process permits small gaps of changing content in the final templates produced. This is useful for templates with dynamic content, where small portions of the template content changes while the essential HTML and text structure remains the same.

Parameter Settings. The DOM-based algorithm is parameterized by the upper and lower thresholds on the number of occurrences of template-nodes. A lower-threshold value of 1 will cause the entire web page to be regarded as a

single template, as the root of the page always occurs at least once. The upper-threshold parameter prevents the algorithm from detecting extremely small HTML constructs like `<BR>` as templates just because they are fairly common in HTML files. Other than removing small commonly-occurring HTML nodes from consideration, the upper-threshold does not have significant impact on the quality of templates detected.

For the experiments reported below, the lower threshold is set to 10% of the number of pages scanned on each site, while the upper threshold is set conservatively to the full number of pages scanned, since the volume of small templates detected does not contribute significantly to the overall proportions. 200 pages were scanned per site. The processing runs at an average of 17.5 seconds per site on a 2.4GHz Pentium IV machine.

5.2.2 Text-based algorithm

The text-based algorithm does not make use of HTML structural information. The page is pre-processed to remove all HTML tags, comments, and text within `<script>` tags. The resulting *detagged content* is typically 2-3 times smaller than the original HTML. The algorithm operates henceforth on this representation.

The algorithm detects templates using a two-pass sliding-window controlled by four parameters: a window size W, a fragment frequency threshold F, a sampling density D, and a page sample size P. All are described below in more detail.

First Pass. In the first pass, P pages are sampled uniformly at random

from the crawled pages of the site[1] and a window of size W is slid over the text of those pages. At each offset, a counter is incremented for the fragment contained in the window. Those fragments which occur at least F times in the sample are passed to the second pass.

For efficiency, we introduce the sampling density parameter D in the first pass. A counter for a fragment is only kept if the hash of the fragment is zero modulo D. Thus, only 1 in every D fragments will be considered, but the downsampling is performed such that if a certain fragment is counted on one page, it will be counted on all pages. Other downsampling mechanisms, such as retaining every D^{th} fragment, do not have this essential property. We choose $D \approx W$ in order to increase the likelihood that after the filtering process concludes, consecutive fragments are contiguous. A coalescing process in the second pass ensures that the total volume of template text is counted correctly. A value of $D = 0$ in the experiments means all fragments are used.

Second Pass. In the second pass, each page is scanned for these frequent fragments, and overlapping or contiguous fragments are coalesced into a single template. At the end of the second pass, we have a set of template hashes which are either individual or coalesced fragments. These hashes are stored in a hash table, so that a new page can be broken into fragments and scanned quickly for templates.

Parameter Settings. Figure 5.1 shows the performance of this algorithm for various values of the parameters. These studies were performed on a

[1]Note that a uniform sample is critical here; if we were instead to crawl only the first few levels of a site, for example, significant biases could be introduced.

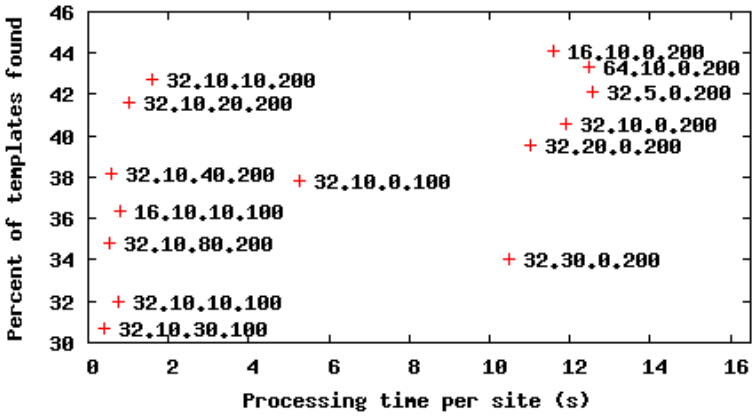


Figure 5.1: Running time and aggregate detection performance for a variety of parameters. Each point is labeled with the parameters W.F.D.P

2.4GHz Pentium IV machine: the running time varies from 0.4 to 12.5 seconds per site, compared to 17.5 seconds per site for the DOM-based algorithm.

The `32.10.0.200` data point represents the algorithm with no downsampling of the number of available templates. Increasing or decreasing W results in a greater proportion found. However, D can now be set to achieve equivalent performance, with much improved running time. With $D = 40$ we achieve a similar proportion detected, with a running time of 0.59 seconds per site, or 3 ms per page, achieving a speedup of 20 times over the non-randomized approach.

Note that if P is set to a smaller value, the detection accuracy changes. As the number of pages sampled is decreased, F must decrease too, in order to detect the same fragments. With very small values of F, however, there is a risk of detecting greater numbers of spurious fragments.

In our experiments, we apply the algorithm with parameter settings 32.10.0.200.

5.3 Volume and Evolution of Webpage Templates

In this section we report on our studies on the nature, volume, and evolution of templates on the Web. We perform these studies using our two SiteLevel approaches.

5.3.1 Methodology

Our concern is to analyze the prevalence and nature of templates across the entire web without introducing unnecessary biases towards a particular subset. To begin, we make use of the IBM WebFountain data set, a large crawl containing over two billion pages and fifty million sites. From this set, we select uniformly at random a subset of two hundred sites, each containing at least two hundred pages[2]; we refer to this dataset as *unbiased*. The scale of the initial collection provides a broad underlying sample space from which to resample. We then manually classify these sites into categories and report results of template behavior for each category.

In addition to studying the amount of templated content on the web, we also study how templating behavior varies across seven site categories determined by inspection of the two hundred sites. These categories are intended to reflect various genres or modes of content that occur on the web, without regard to the nature of the content. Each has implications for the kinds of formatting and quantities of information that occur on each page. The categories are:

- Brochure. The online presence of a company or organization, typically

[2]The requirement that each site contain at least two hundred pages introduces a bias; we discuss the nature of this and other biases below.

containing events, reviews, press releases and diverse other information.

- Catalog. Listings of products, usually for sale.
- Community. Sites with content submitted and managed by a large number of individuals.
- Documents. Sites containing reference material. Many academic and government sites fall into this category.
- News. Sites which contain regular and editorially controlled updates on some range of topics. Most often this is local news or news devoted to specific topics.
- Personal. Homepage of a single individual, irregularly updated and containing a mix of content.
- Portal. Links to contents elsewhere. Often these are local portals, for a particular city or region.

A dating site, for example, falls most naturally into the "Catalog" category, even though the "products" are not really for sale. If the site also contained a chat forum, it would also fall into the "Community" category; thus, multiple assignments are allowed in our categorization.

Of the 200 sites, 109 were labeled, and the remainder were either pornographic (about 3%), no longer existent (about 15%), or not in a language understandable by the authors (about 30%); see below for a discussion of the biases introduced by this labelling. The number of sites in each category are shown in Table 5.1. Roughly 5% of sites are news sites, much fewer than the

News	5
Personal	8
Community	14
Documents	14
Catalog	40
Brochure	42
Portal	16

Table 5.1: The number of websites in each category.

number of community and personal sites. The dominance of the commercial sector of the web is clear from the number of catalog and brochure sites.

In addition to studying the templates on today's web, we also examine the evolution of templates a from 1996 to 2004, based on a crawl of pages stored in the Internet Archive [Kah]. We study two sets of sites. The first set is the collection of 109 unbiased sites described above. We will refer to this data set as the *unbiased* set. Of the 109 sites in the set, we found at least one snapshot for 78 of them.

Our second evolutionary dataset covers more popular web sites, and is better represented in the Internet Archive's historical database. Ntoulas et al. [NCO04] used a set of 157 sites in order to study changes over time. While this set may be less representative of the web at large, it is perhaps more representative of the types of content that people typically browse, and it has been extensively studied by Ntoulas and his co-authors, allowing us to place our results in context. We found at least one snapshot for 105 of the 157 sites in the set.

We successfully crawled approximately 72K pages from the Internet Archive from these two datasets representing 1380 snapshots of a website at a partic-

	Unbiased	Popular
Non-empty Websites	78	105
Total pages	32K	42K
Avg snapshots/site	5	8
Year of Snapshot	Unbiased #Snapshots	Popular #Snapshots
1996	2	19
1997	5	51
1998	10	46
1999	15	64
2000	50	162
2001	60	165
2002	98	178
2003	194	198
2004	24	39
Total	458	922

Table 5.2: Internet Archive data volumes for Unbiased and Popular collections of websites.

ular time. Some details about these data sets are shown in Table 5.2.

Summary of Biases: The following biases exist in our sample. First, we consider only sites with at least two hundred pages in our crawl. Pages that lie on smaller sites represent approximately 20% of the overall crawl, and thus represent a non-negligible fraction; nonetheless, for technical reasons, we focus on the 80% of pages that belong to larger sites. Second, we consider non-pornographic sites only; we thus report results for the non-pornographic region of the web. Third, our classification results apply only to sites in English, but all other results apply to sites in all languages. This bias is difficult to overcome without enlisting the skills of many assistants. Third, the crawling of sites is performed by a commercial crawler, which encodes many design decisions that may influence its behavior for or against a particular site. Finally, our

	Text-Based	DOM HTML	DOM Text	DOM Links
Brochure	56	59	53	55
Catalog	66	59	57	51
Community	64	51	50	53
Documents	35	57	26	58
News	12	15	8	12
Personal	67	68	77	52
Portal	44	48	39	43
OVERALL	53	53	46	49

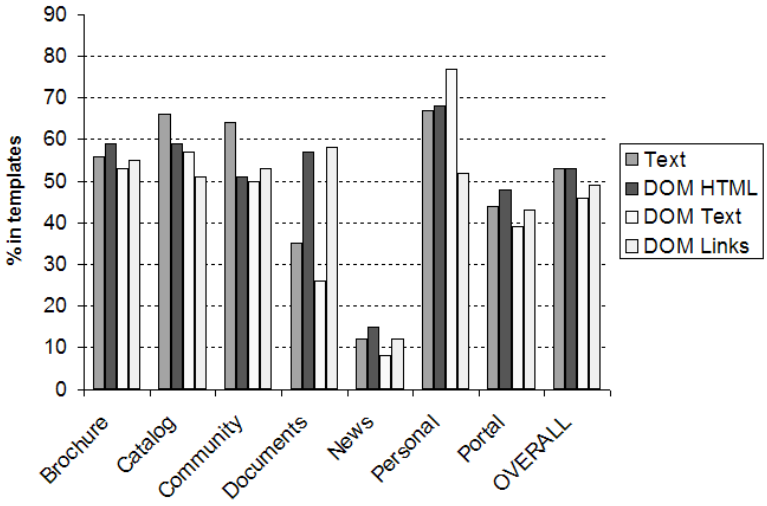


Figure 5.2: Proportions of templated content for all categories

"popular" set of sites used in the evolutionary study is biased towards websites with high pagerank. Experiments on this set of sites give us more robust measurements and help us compare our results with those in past studies. However, we also perform experiments on an unbiased dataset in order to confirm that none of our conclusions are due to the biases. Overall, however, we believe the scope of the underlying dataset makes the results reasonably representative.

5.3.2 Templates on Today's Web

We ran both the DOM-based and the text-based algorithms over the unbiased sample set. The text-based algorithm reports the fraction of text content within templates on each page. The DOM-based algorithm reports the fraction of template versus non-template HTML content on the page, and then through post-processing of the resulting templates, also reports the fractions of links and text that appear within a template.

The two algorithms should report similar values for the fraction of text content that appears in templates. An examination of the results shows that the reported fractions of template content on average differ by only 7%, and show a similar level of agreement for each individual category. Given the extremely different approaches taken by these two approaches, we find the measures of fraction of template content to be fairly stable across these approaches.

The results are shown in Figure 5.2. The figure shows a significant difference between the volumes of templates across the different categories. Overall, the amount of template text on a page is around 50%, but this is significantly lower for News sites, and significantly higher for Personal sites. The types of text found in templates also vary across categories: for example, there are noticeably more links in templates in the Documents category.

5.3.3 Evolution of Templates

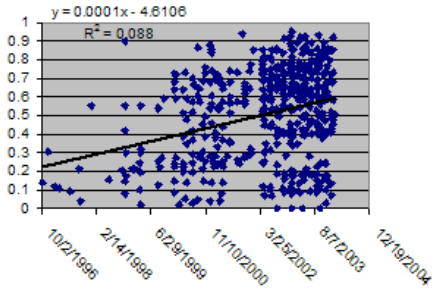
For each snapshot in the unbiased and popular evolutionary datasets, we identified templates using the DOM based detection method, and considered six regions on each page: links, text, and HTML within and outside templates.

	Unbiased Sites			Popular Sites		
Category	96–01	02–04	All	96–01	02–04	All
Links	44%	55%	52%	32%	42%	36%
HTML	39%	46%	44%	32%	40%	35%
Text	28%	38%	35%	21%	28%	24%

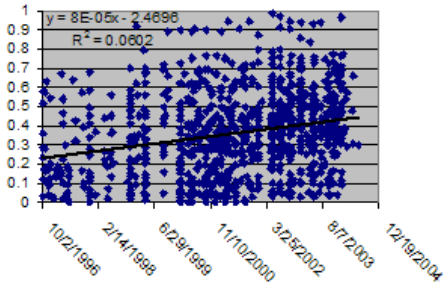
Table 5.3: Fraction of links, HTML, and text that appears in templates by data collection and date range.

Fraction of Template Content over Time.

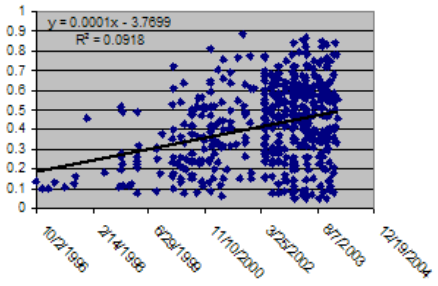
Our first set of evolutionary results covers the fraction of content that appears inside versus outside templates as a function of time. The results and trends are similar for popular and unbiased sites, so we report only results for popular sites as the number of snapshots is larger. Figures 5.3 are scatter plots in which each point represents a website from our popular dataset at a particular point in time (i.e., one of the snapshots of Table 5.2). The x axis represents the time of the snapshot. The y axis is the fraction of content on the page that occurs inside a template. For instance, Figure 5.3(a) and Figure 5.3(b) show the increase in the fraction of links within templates over time for the unbiased and popular set of sites respectively. While coverage for sites in the 1990s is sparser, it is clear that snapshots from 2002 and 2003 show a significantly larger proportion of sites with more links in templates. The best fit trend line shows a growth of 8% per year in the fraction of links that are inside a template.



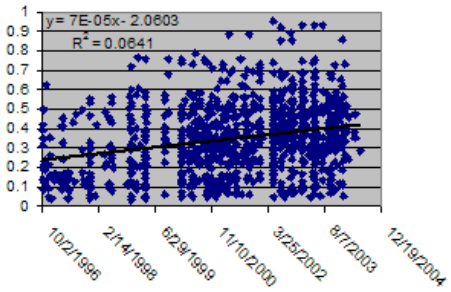
(a) Links in Unbiased Websites



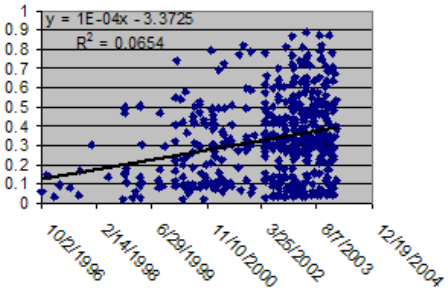
(b) Links in Popular Websites



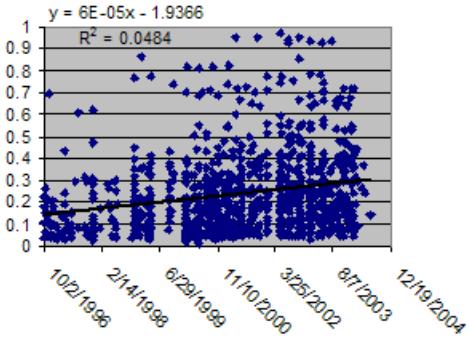
(c) HTML in Unbiased Websites



(d) HTML in Popular Websites



(e) Text in Unbiased Websites



(f) Text in Popular Websites

Figure 5.3: Fraction of content inside versus outside templates as a function of time.

Figures 5.3(d) and 5.3(f) show the same type of scatter plot for the fraction of the bytes of HTML, and the bytes of detagged content, that appear within templates for the popular websites. The best fit growth rates are about 7% and 6% respectively. Total bytes of HTML again shows a mass of more heavily-templated pages in more recent years. While many recent pages have more than 70% of their links in templates, this is not true for total HTML content, supporting the intuition that pages may contain menus, headers, footers, and sidebars with a large number of navigational links, but will still contain some reasonable amount of non-template content. Similar trends are also seen for the unbiased set of sites.

Table 5.3 shows summary information for these figures. The popular sites show less overall template activity than the unbiased sites, though with similar trends. The unbiased sites from 2002 onwards show 38% of their text, 46% of their HTML, and fully 55% of their links in templates. Combining this aggregate information with the trend lines, we see that a large and rapidly-growing fraction of links appear in templates, suggesting that template-based navigation continues to increase in popularity. The aggregate results shown in this table are normalized for site size and number of internet archive crawls per site. Thus, the results should be taken as representative of the "average" page in the given collection.

Rate of Change of Webpages.

Ntoulas and his co-authors crawled each site of the popular set weekly, and performed experiments to capture the amount of change noted each week; this amount was found to be very small for most changes. We conducted a similar experiment to check whether the amount of change would be higher if we first removed templates from these pages. The Internet Archive crawls

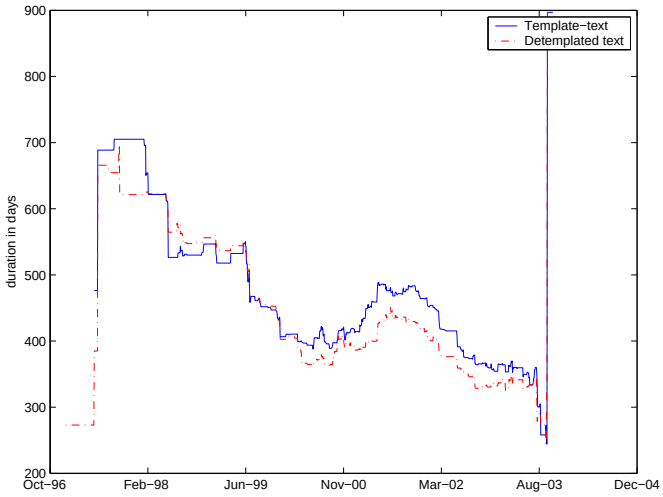


Figure 5.4: Average duration of all templates and detemplated pages existing at each point in time.

pages much less frequently than once per week, so the change on each visit will be on average much larger in our case. However, from our data we can estimate changes that occur less frequently than every hundred days.

We perform the following experiment. Consider a series of n snapshots of a web page, and let $x_1, \ldots, x_n$ be the value of the templated region of the page at each timestep. Let t_i be the time of the i^{th} snapshot. We will apply exactly the same approach to the *detemplated region* of the page; that is, all content on the page other than the templates. In this analysis, we consider the text content rather than the HTML or links. Consider the i^{th} snapshot, x_i. If $x_1 \neq x_i \neq x_n$ then we say that the value x_i is *bracketed*, meaning that we saw the page before this template appeared, and thus we have some estimate of the date when it appeared; and we saw the page after the template had disappeared, and thus we have an estimate of the date when it disappeared. For any bracketed value x_i, we define the *first* value $f(x_i)$ as the index i' at

which $x_{i'} = x_i$, but $x_{i'-1} \neq x_i$. Likewise, the *last* value $\ell(x_i)$ is the index i' such that $x_{i'} = x_i$ but $x_{i'+1} \neq x_i$. The beginning $B(x_i)$, the time at which the template appeared, is then estimated to be $(t_{f(x_i)} - t_{f(x_i)-1})/2$. Likewise, the end $E(x_i)$ is estimated to be $(t_{\ell(x_i)+1} - t_{\ell(x_i)})/2$. Notice that these times must all exist if x_i is bracketed. Finally, the duration $D(x_i)$ is taken to be $E(x_i) - B(x_i)$.

For any time t, we say the active templates at t are all the templates such that $B(x_i) \leq t \leq E(x_i)$. Notice that the active templates at time t are all the templates that both exist on some page at time t and are bracketed (so that we can estimate their duration). The average duration at time t is then the average of the duration of all templates that are active at time t. Figure 5.4 shows the average duration as a function of time. The figure also shows a second curve in which the value of x_i is not the templated region of the page, but is the remainder of the page (that is, the detemplated region). In both cases, the average duration of a template can be seen to shrink dramatically over time, implying that the rate at which both content and templates are changing is shrinking. The average duration of templates is slightly larger than that of detemplated text, but the difference does not appear to be significant.

Figure 5.5 shows the histogram over the entire timeframe of the study of the average durations of templates. Due to the refresh rate of the Internet Archive, we do not have detailed information for content that changes more frequently than every hundred days. However, the figure demonstrates that both templates and detemplated content typically last for between fifty and three hundred days, with perhaps five percent remaining for two years or more.

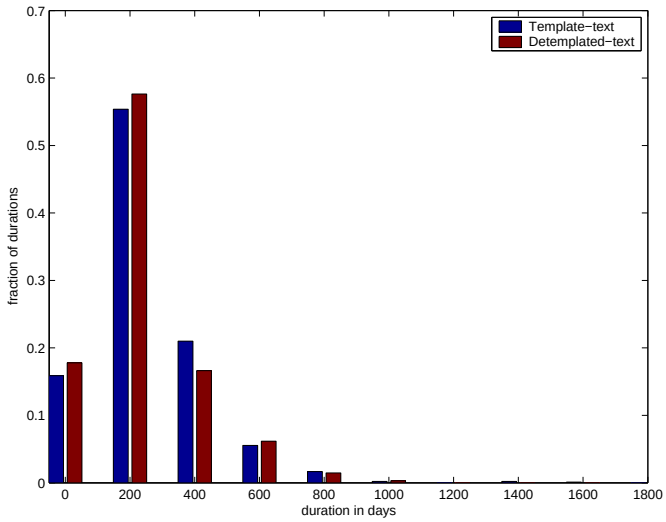


Figure 5.5: Histogram of durations of templates and detemplated content over all pages.

Change Rate vs Change Magnitude.

Figure 5.6 shows an analysis of changes in the text content of pages from one version of a page to another. For two documents with word sets A and B, the magnitude of the change is taken to be: $1 - 2\frac{|A \cap B|}{|A|+|B|}$. The figure shows the distribution of the magnitude of change for the detemplated region of the page and for the entire page. Changes of magnitude 65% or larger are about twice as likely in the detemplated text, suggesting that results on large changes may be biased by the presence of a significant and unchanged template. Overall, however, the results in this figure are very similar to those of Ntoulas et al.

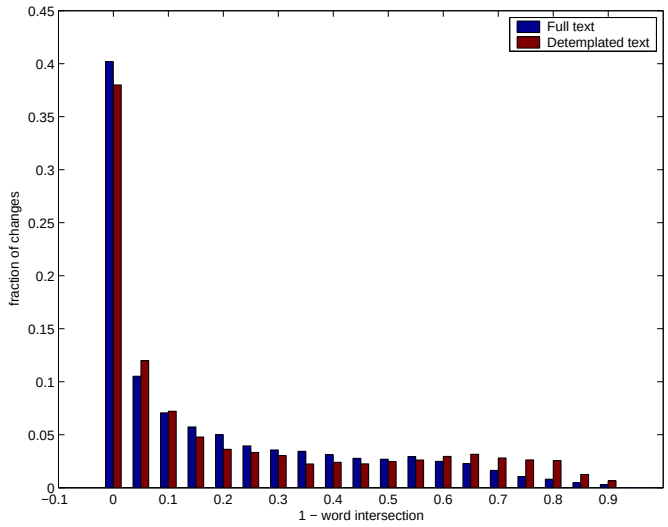


Figure 5.6: Distribution of magnitude of change in full text and detemplated content.

5.3.4 Conclusions from our study

Templates represent 40–50% of the total bytes on the web, and this fraction continues to grow at a rate of approximately 6% per year. Similarly, the fraction of visible words, and the fraction of hyperlinks appearing in templates is extremely high. This finding implies that: (1) the graph structure of the web is increasingly dominated by boilerplate, suggesting that link analysis algorithms require understanding of templates; (2) with increased bandwidth, site creators are spending an increasing fraction of their resources on conveying information that has little raw content value, suggesting that improved caching and delivery mechanisms are needed.

5.4 Page-level Template Detection

Most existing methods for template detection operate on a per website basis by analyzing several webpages from the site and identifying content and/or structure that repeats across many pages. We proposed two such SiteLevel measures in the previous sections. While these global template detection methods offer a lot of promise, they are of limited use because of the following two reasons. First, site-level templates constitute only a small fraction of all templates on the web. For instance, page- and session-specific navigation aids such as "Also bought" lists, ads, etc. are not captured by the site-level notion of templates. Second, these methods are error prone when the number of pages analyzed from a site is statistically insignificant, either because the site is small, or because a large fraction of the site is yet to be crawled. In particular, they are totally inapplicable when pages from a new website are encountered for the first time. An alternative paradigm that avoids many of these pitfalls is to detect templates on per webpage basis, i.e., "page-level" template detection. This is especially attractive since it can be easily deployed as a drop-in module in existing crawler work-flows.

A tempting approach to page-level template detection is to extract sufficiently rich features from the DOM nodes and train a classifier to assign "templateness" scores to each node in a DOM tree. While this approach is entirely plausible, it has several handicaps. First, for the classifier to have a reasonable performance both accurate and comprehensive training data is required; this can involve prohibitive human effort. Second, by classifying each DOM node in isolation this approach does not take a global view of the templateness of nodes in the DOM tree.

In this chapter we develop a framework for page-level template detection.

5.4.1 Framework

In this section we describe the proposed framework for the page-level template detection problem. We first fix some notation.

Recall the DOM tree representation of an HTML document where each node in the DOM tree corresponds to an HTML fragment; we identify the DOM node with the fragment it represents. Let T be the rooted DOM tree corresponding to the document. From here onwards, we use the tree T as a metaphor for the document. Let templ(T) denote the set of all nodes in T that are templates. We use $v \in T$ to denote that node v belongs to tree T, parent(v) to denote the parent of i in T, child(v) to denote the set of children of v in T, and root(T) to denote the root of T.

Let $\mathcal{H}$ denote the set of all possible DOM nodes. In the page-level template detection problem, we seek a boolean function $\tau : \mathcal{H} \rightarrow \{0, 1\}$ such that $\tau(v) = 1$ for all $v \in$ templ(T), and $\tau(i) = 0$ otherwise. In a relaxed version of the problem, we seek a function $\tilde{\tau} : \mathcal{H} \rightarrow [0, 1]$ where if $v \in$ templ(T) and $w \notin$ templ(T), then $\tilde{\tau}(v) > \tilde{\tau}(w)$; using an appropriate threshold, we can round $\tilde{\tau}$ to make it boolean.

A first-cut approach to page-level template detection would be to extract sufficiently rich features from the DOM nodes (in the context of a page) and train a classifier $x : \mathcal{H} \rightarrow [0, 1]$ to score the "templateness" of each node in a given page. While this appears plausible, it has several issues when scrutinized closely. The first set of issues revolve around the construction of the training data for our classifier. For the classifier to learn the notion of "templateness" of DOM nodes on the web in general, it must be trained comprehensively over all forms of templates that it is likely to encounter. The heterogeneity and scale of the web imply that a huge corpus of accurate and diverse training data

will be required. These requirements present a daunting task that demands tremendous human effort. Secondly, this approach to classification ignores the global property of templateness in the DOM tree, crisply stated as follows.

Property 5.1 (Templateness Monotonicity)**.** *A node in the tree is a template if and only if all its children are templates. In other words, the function $\tau(\cdot)$ is monotone on the tree.*

As is apparent, by working on each node of T in isolation, a naive classifier misses this intuitive relationship among templateness of nodes in the tree.

Our three step framework is meant precisely to address these issues, and is described below.

1. Automatic Generation of Training Data. The first step is the automatic generation of training data. To this end, we use the site-level template detection paradigm of [GPT05]. Note that even though site-level template detection is less feasible as a web-scale template detection mechanism, we show that we can still use it to generate training data for our approach.

The basic intuition behind the site-level template detection approach is the following. One of the common properties of templates is that they occur repeatedly across many pages on a single site. Therefore, if a DOM node occurs many times on different pages from a single site, then it lends credible evidence that this DOM node perhaps corresponds to a template.

We now describe a generic algorithm that we will call SiteLevel (θ). This algorithm operates on a site by site basis. For each site, it obtains a set $\mathcal{T}$ of random pages from the site. Then, for each page $T \in \mathcal{T}$ and for every DOM node $i \in T$, it computes $h(i)$, where $h(\cdot)$ is a random hash function. Let

$I_\theta^+ \subseteq \mathcal{H}$ be the set of DOM nodes that occur on at least θ fraction of pages in $\mathcal{T}$. Note that using hashes, this set can be identified efficiently. SiteLevel returns I_θ^+ as the set of DOM nodes deemed templates.

2. Classification. The second step is to use the set of DOM nodes I_θ^+ identified by SiteLevel as training data for a classifier. For this, we first identify appropriate features of DOM nodes in I_θ^+, in the context of the pages they appear in. We then train a classifier $x : \mathcal{H} \to [0, 1]$ using these features of the DOM nodes, treating those in I_θ^+ as positive examples; the output of the classifier is a templateness score for a given DOM node in a tree. The hope in using a classifier is that it can distill features from site-level templates that can be generalized to all templates on the web. This can help us identify templates that don't manifest themselves by repeatedly occurring across multiple pages on a website — templates that a pure site-level template detection approach cannot discover by itself. As we empirically observe in Section 5.7.2, this is indeed what happens.

3. Isotonic Smoothing. At this point, one could use the classifier to assign a templateness score $x(\cdot)$ to each DOM node in the given page T. However, as we argued earlier, this does not fully capture the essence of the problem since the templateness scores assigned by the classifier to each DOM node in isolation may not satisfy the property that a node is a template if and only if all its children are templates (Property 5.1). On the other hand, assuming the classifier has reasonable accuracy, the scores it assigns makes sense for most, if not for all, of the nodes. The question now is how to reconcile the score assigned by the classifier with the monotonicity property of the templates.

To handle this question, we first consider a natural generalization of the monotonicity property for the case of real valued templateness scores. Suppose $y(v)$ is the templateness score of a node v in the tree. Then, $y(\cdot)$ is said to satisfy *generalized templateness monotonicity* if for every internal node v, with children $u_1, \ldots, u_\ell$, $y(v) = \min\{y(u_1), \ldots, y(u_\ell)\}$, i.e., the templateness of an internal node is the equal to the least of its children's templateness scores.

Note that generalized monotonicity ensures, first, that the templateness score of a node is at least the templateness score of its parent, and second, that the templateness score of the parent equals the templateness score of all its children, when the children all have same templateness score. We also have an additional requirement that the templateness score $y(\cdot)$ be close to the $x(\cdot)$ scores assigned by the classifier. Generalized monotonicity together with this closeness requirement leads to the problem of generalized isotonic regression on trees, which we solve in this chapter.

While we defer the detailed description of our solution to the next section, we now highlight the advantages of our framework. Besides addressing the issues with using just the classifier scores, our framework offers additional benefits. First, the overall framework is simple and modular. Second, any off-the-shelf classifier can be used, instead of having to design one that works specifically for the given DOM tree structure. Third, as we will see later, a neat by-product of isotonic smoothing is that we obtain a sectioning of a webpage; this can be useful in many applications.

5.5 Step-regularized Tree Isotonic Regression

In this section we formulate and solve the generalized isotonic regression problem on trees. Recall that we are given as input a DOM tree with each node labeled by a score assigned by the classifier. The purpose of isotonic regression is to fix these scores so that they satisfy the monotonicity constraints, while remaining as faithful as possible to the original classifier scores. Let $x(v)$ be the classifier score for each node $v \in T$ and let $y(v)$ be the smoothed score we wish to obtain.

The first step in our formulation is to alter the generalized monotonicity property in two ways. First, we only ensure that the templateness score of a node is *at most* the least of its children's scores, instead of *equal* to it. This relaxation is derived from the current domain in which the cost of misclassifying a non-template as a template is much higher than vice versa. Hence, if according to the classifier an internal node's template score is much lower than that of all of its children, then we would want to respect that. Second, we introduce a regularization that penalizes if, for a node v, the templateness score $y(v)$ is different from those of its children $y(u_1), \ldots, y(u_\ell)$. Clearly, if $y(u_1) = \cdots = y(u_\ell)$, then this regularization will try to ensure that $y(v) = y(u_1)$.

Thus, we have

(1) For every internal node v with children $u_1, \ldots, u_\ell$, $y(v) \leq \min\{y(u_1), \ldots, y(u_\ell)\}$.

For purposes of regularization, we develop the notion of compressed score that embodies sectioning of the DOM tree into subtrees. A *compressed score* is a function $\hat{y} : T \to [0, 1] \cup \{\perp\}$ with the following properties:

(2) $\hat{y}(\text{root}(T)) \neq \perp$, and

(3) if v is an ancestor of u and $\hat{y}(v) \neq \perp \neq \hat{y}(u)$, then $\hat{y}(v) < \hat{y}(u)$.

Let the *size* $|\hat{y}|$ of the compressed score be the number of places where $\hat{y}$ is defined; $|\hat{y}| = |\{v \mid v \in T, \hat{y}(v) \neq \perp\}|$.

For all $v \in T$ such that $\hat{y}(v) = \perp$, let $\text{anc}(v)$ be the closest ancestor of v such that $\hat{y}(\text{anc}(v)) \neq \perp$; note that such an ancestor always exists by (2). We now interpolate $\hat{y}$ to a unique y as follows.

$$y(v) = \begin{cases} \hat{y}(\text{anc}(v)) & \text{if } \hat{y}(v) = \perp \\ \hat{y}(v) & \text{otherwise} \end{cases}$$

It is clear that if $\hat{y}$ satisfies (2) and (3), then the corresponding interpolated y satisfies (1). Also, given a y satisfying (1), it is easy to construct the unique $\hat{y}$. From now on, we use the smoothed score y and its compressed counterpart $\hat{y}$ interchangeably.

Finally, the *cost* of a smoothed score y with respect to x is defined as

(4) $c(y) = \gamma \cdot |\hat{y}| + d(x, y)$,

where γ is a penalty term that captures the cost of each new smoothed score and $d(\cdot, \cdot)$ is some distance function. Note that unlike the regularization in Chapter 4, where the penalty depended on the absolute difference between parent child values, the current regularization is a step function. Any difference between parent child values is charged a penalty of γ. It is also possible to have a node-specific penalty γ_v for node v; for simplicity of exposition, we state the algorithm in terms of a node-independent term γ.

This cost function and the tree structure lead to a regularized version of the isotonic regression problem.

Problem 5.2 (Step-regularized Tree Isotonic Regression)**.** *Given a tree T and $x : T \to [0, 1]$, find $y : T \to [0, 1]$ that satisfies (1) and minimizes $c(y)$ as given*

by (4). [3]

For the rest of the chapter, we take $d(\cdot, \cdot)$ to be the L_1 norm since it is robust against outliers.

Before presenting the algorithm we discuss a key property of the L_1 distance measure that aids us in designing an efficient algorithm for this problem. We show that the optimal smoothed scores in y can only come from the classifier scores in x.

Lemma 5.3. *There exists an optimal solution, $\hat{y}$, where, for all $i \in T$, if $\hat{y}(i) \neq \perp$, then there is a $j \in T$ such that $\hat{y}(i) = x(j)$.*

Proof. This lemma is proved by the same reasoning as Lemma 4.5. □

We build a dynamic program (pseudo-code in Figure 5.7) using the above result to obtain an algorithm for the regularized tree isotonic regression problem. Let $x(\cdot)$ be the original "templateness" scores and $\hat{x}$ be the set of unique values in x. Algorithm BuildErrorStep builds up an index function $\text{val}(v, i)$ and an error function $\text{err}(v, i)$ for each node $v \in T$. The value $\text{err}(v, i)$ represents the cost of the optimal smoothed scores in the subtree rooted at v if when its smoothed score is set to $y(v) = \hat{x}(i)$. Using the err function, index $\text{val}(v, i)$ is set to the hold the index of the optimal smoothed score for node v given by $y(v) = \hat{x}(\text{val}(v, i))$, when the $y(\text{parent}(v)) = \hat{x}(i)$.

[3] Note that a special case of our problem has been considered before in statistics and computer science contexts; it is usually referred to as the *isotonic regression problem*: given $\vec{x} = x_1, \ldots, x_n$, find $\vec{y} = (y_1, \ldots, y_n)$ such that $y_1 \leq \cdots \leq y_n$ and $d(\vec{x}, \vec{y})$ is minimized, where $d(\cdot, \cdot)$ is some distance function. It is easy to extend this definition to the case when the y_i's have to respect a given partial order, say, imposed by a tree.

Algorithm BUILDERRORSTEP $(v, x, \hat{x})$

1. `if` (v is a leaf) `then`
2. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
3. $\text{err}(v, i) = w_v \cdot |x(v) - \hat{x}(i)|$
4. `else`
5. `for` child u of node v
6. BUILDERRORSTEP$(u, x, \hat{x})$
7. `for` $i = 1 : |\hat{x}|$ /* all values child u can take */
8. $\text{errheap}(i) = \text{err}(u, i)$
9. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
10. $\text{val}^* = \text{argmin}_{j \in \{1 \ldots |\hat{x}|\}, \hat{x}(j) \leq \hat{x}(i)}$ errheap(j)
11. `if` $(\text{err}(u, i) > \text{err}(u, \text{val}^*) + \gamma_{vu})$ `then`
12. $\text{err}'(i) = \text{err}(u, \text{val}^*) + \gamma_{vu}$; $\text{val}(u, i) = \text{val}^*$
13. else
14. $\text{err}'(i) = \text{err}(u, i)$; $\text{val}(u, i) = i$
15. `for` $i = 1 : |\hat{x}|$ /* all values node v can take */
16. $\text{err}(v, i) = \text{err}'(i) + w_v \cdot |x(v) - \hat{x}(i)|$

Algorithm ISOTONESMOOTH$(\text{err}, \text{val}, \hat{x})$

1. $\text{val}^* = \text{argmin}_{i \in \{1 \ldots |\hat{x}|\}}$ $\text{err}(\text{root}(T), i)$
2. $p(\text{root}(T)) = \text{val}^*$; $y(\text{root}(T)) = \hat{x}(\text{val}^*)$
3. `for` v in a breadth-first search order of T
4. $p(v) = \text{val}(v, p(\text{parent}(v)))$; $y(v) = \hat{x}(p(v))$

Figure 5.7: Algorithm to solve Problem 5.2. Array x contains the original classifier scores and $\hat{x}$ is the set of unique values in x. w_v denote the node-specific weights. BUILDERRORSTEP constructs functions $\text{err}(\cdot, \cdot)$ and $\text{val}(\cdot, \cdot)$ which are then used by ISOTONESMOOTH to find the smoothed scores $y(\cdot)$.

For a child u of a node v, if $\text{val}(u, i)$ is the same as i, i.e, the optimal value for v and $\text{parent}(v)$ are the same $\hat{x}(i)$, then the only contribution to cost from node u is the L_1 distance between $x(u)$ and $\hat{x}(\text{val}(u, i))$, otherwise there is an additional γ cost as well. The algorithm computes this error function by finding the values for children of v that cost the least and are feasible in that they are greater than value for v (step 10). Upon finding such a value for a child a decision (step 11) is made between (a) continuing with the parent's value and only adding the error of the child at parent's value to err' (step 12), or (b) creating a new section with a new value for the child and paying in the full cost (error of child at new value + γ) in err' (step 14). Once all error functions have been computed, the optimal smoothed scores are obtained using Algorithm IsotoneSmooth, which starts with the best index $p(\text{root}(T))$ at the root, and progressively finds the best index $p(\cdot)$ for nodes lower down in the tree.

To demonstrate the correctness of this algorithm, we show that the restriction of the optimal solution to a subtree is also the optimal solution for the subtree under the monotonicity constraint imposed by its parent.

Consider the subtree rooted at any non-root node $i \in T$. Now suppose the smoothed score $y(\text{parent}(i))$ is specified. Then, let $z(\cdot)$ be the smoothed scores of the optimal solution to the regularized tree isotonic regression problem for this subtree, under the additional constraint that $z(i) \geq y(\text{parent}(i))$.

Lemma 5.4. *For all nodes j in the subtree of i, $y(j) = z(j)$.*

Proof. This lemma follows from the proof of Lemma 4.6. □

Theorem 5.5. *Algorithm* IsotoneSmooth *in Figure 5.7 solves the step-regularized tree isotonic regression problem.*

Proof. The algorithm computes up the optimal smoothed scores for each subtree, i.e., the $\mathrm{err}(\cdot,\cdot)$ arrays, while maintaining (1) for every possible smoothed score of the parent. By Lemma 5.3, the parent can take only finitely many smoothed scores in the optimal solution, and by Lemma 5.4, combining the optimal smoothed scores for subtrees yields the optimal smoothed scores for the entire tree. □

COMPLEXITY. Let $|T| = n$. The space required per node is $O(n)$, and so the total space required is $O(n^2)$. Next, we consider the running time of the algorithm. In the dynamic program, step 2 takes $O(n^2)$ time, step 7 takes $O(n^2)$ time amortized over all calls, and step 9 can be done in $O(n^2 \log n)$ time by storing errheap values in a heap and then running over the nodes $j \in T$ in ascending order of $x(j)$. Hence, the total running time is $O(n^2 \log n)$. Note that this is same as the complexity of the algorithms for other forms of regularized isotonic regression introduced in Chapter 4 and also the best time complexity of previously known algorithms for the non-regularized forms of tree isotonic regression [AHKW06].

5.6 Details of the System

In this section we describe the details of both the classification and smoothing aspects of our system.

5.6.1 Constructing Training Data

As mentioned before, we used a site-level template detection algorithm to generate training data for our classifier. The construction of training data involved two distinct steps: collecting webpages and obtaining labeled templates. We sampled $3,700$ websites from the Yahoo! search engine index such that each website had at least 100 webpages. We also biased the sampling process slightly towards picking good quality host domains, and avoided picking pornographic or spam websites. Then, for each website we downloaded at most 200 randomly picked webpages.

All DOM nodes that occurred on more than 10% of the pages of any website were tagged as site-level templates. Since we wanted to learn a classifier for all internal DOM nodes we wanted representative labeled data from all levels of the DOM trees. Hence, for each internal node we computed how much of its HTML was part of a site-level template. DOM nodes with more than 85% of their HTML content within site-level templates were also labeled as templates. The rest of the DOM nodes were used as instances of the non-templates class.

Note that the condition required for tagging a node as template is very strong. This is done intentionally for two reasons. First, recall that a node is a non-template if *any* node in its subtree is a non-template. And second, the cost of misclassifying a non-template as a template is much higher than that of the reverse error.

5.6.2 Learning the Classifier

There are multiple steps involved in learning the classifier. Each of these steps is described below.

PREPROCESSING. Each webpage is preprocessed and parsed so that features can be extracted for its DOM nodes. The preprocessing step involves cleaning the HTML code using Hypar[4], annotating the DOM nodes with position and area information using Mozilla[5], and parsing the HTML to obtain a DOM tree structure. The text in the HTML page is also processed to remove stop words.

FEATURE EXTRACTION. The training data that we employ for learning corresponds to site-level templates. However, we want our classifier to generalize to the global definition of templates. This makes the process of feature extraction very critical. From each DOM node, we extract features that we believe are indicative of whether or not that DOM node is a template. For example, intuitively, if the text within a DOM node shares a lot of words with the title of the webpage, then perhaps it is not a template node. Similarly, the distance of a DOM node from the center for the page indicates its importance to the main purpose of the page, and hence its templateness.

In a similar fashion, we constructed several other features from the position and area annotations of DOM nodes as well as from the text, links, anchortext they contain. The most discriminative features turned out to be: closeness to the margins of the webpage, number of links per word, fraction of text within

[4] `www.cse.iitb.ac.in/~soumen/download`
[5] `www.mozilla.org`

anchors, the size of the anchors, fraction of links that are intra-site, and the ratio of visible characters to HTML content.

Classifier Training. We trained Logistic regression classifiers [Mit97] over the set of features described above. Apart from performing very well, these classifiers have the additional benefit that their classification output can be interpreted as the probability of belonging to the predicted class. In our exploratory experiments we observed that distributions of feature values varied heavily depending on the area of the DOM node. This is because template and non-template nodes have very different characteristics at different levels of the DOM trees; these levels can be approximated by the area of the node. Hence, we trained four logistic regression models for DOM nodes of different sizes. Now, given a webpage, the appropriate logistic model is applied to each node of the DOM tree, and the output probabilities are fed to our post-classification smoothing function.

5.6.3 Smoothing Classifier Scores

The smoothing algorithm allows arbitrary choices of penalty values for each tree node. However, in the domain of template detection, there are several desiderata that a good penalty function must try to achieve. We list these below, along with the particular functions that we considered, and the one that we finally settled upon.

Desiderata for Penalties. There are three main desiderata for a smoothing algorithm in the context of template detection. First, nodes that are too small in area should not form segments of their own. Such nodes have very little content, and their classification scores are unreliable. Also, having such

small segments impairs the applicability of webpage segmentation to page visualization and browsing.

Second, adding nodes as segments should be easier as we move up from leaves to the root. The smoothed values assigned to nodes high up in the tree impose constraints on the possible values in their entire subtree. If creation of new segments is hard, such nodes may merge with other nodes to form larger segments whose smoothed scores may be drawn too far away from the classification score of the node itself, thus hurting all nodes in their subtree.

Third, if a child node accounts for a large fraction of the area of its parent node, then it should be harder to set the child to a value different from that of its parent. This encourages the smoothing algorithm to form large sections without too much nesting, which agrees with our intuitions about how webpage segments are created.

Handling Very Small Nodes. All nodes whose area is less than 2000 sq pixels are neither classified nor smoothed; they are "hidden," and their effect is rolled into their parent node. Thus, a node with k hidden children acts as if it were $(k+1)$ nodes, all with the same classification value. This reduces to multiplying the distance measure $d(\cdot,\cdot)$ with $(k+1)$, and the smoothing algorithm can handle it trivially. This heuristic goes some way in achieving the first desideratum.

Penalty Functions. We experimented with several penalty functions, attempting to achieve the aforementioned desiderata. Starting with a user-defined constant c, several transformations for γ_i (penalty for node i) were tried:

(1) $\gamma_i = c \cdot N/N_i$, where N_i is the number of nodes in the subtree rooted at i, and N is the total number of nodes in the DOM tree. This penalty is high for nodes near the leaves and low for nodes near the root, satisfying the first and second desiderata.
(2) $\gamma_i = c \cdot A/A_i$, where A_i is the area of node i, and A the area of the whole HTML page. Again, this penalty satisfies the first and second desiderata.
(3) $\gamma_i = c \cdot A_{\text{parent}(i)}/A_i$, where $A_{\text{parent}(i)}$ is the area of the parent of node i. This tries to achieve the third desideratum.

We tried all combinations of these penalties, over a large range of values for the constant c, and visually inspected the results of smoothing on a few webpages. We finally settled on setting $\gamma_i = 0.01 \cdot A/A_i$, which gave the best results. For the rest of this chapter, unless specified otherwise, the penalty is always set to this function.

5.7 Experiments

We now present an empirical evaluation of our system, called PageLevel. Using human-labeled data, we show in Section 5.7.1 that our approach is very effective in detecting the template sections of webpages. Then, in Sections 5.7.2 and 5.7.3 we show that applying template detection as a pre-processing step significantly improves accuracy on standard web mining tasks such as duplicate webpage detection and webpage classification. We also show that template removal using PageLevel provides more benefits than using the more expensive SiteLevel approach.

5.7.1 Template Detection Performance

The desiderata for a template detection system are as follows: (a) it must divide the webpage into segments separating template and non-template content; and (b) it must accurately identify the webpage segments as template or non-template. In this section we show that our system, PageLevel, achieves both these objectives.

Datasets. In order to evaluate the template detection performance of PageLevel we manually created two labeled datasets.

Common. We selected and manually labeled 44 pages from websites that were commonly visited by the authors. The selected pages come from a diverse set of domains, such as, news websites like NYTimes and CNN, university websites like UTexas-Austin, etc. For each webpage, the manual labeling process identified the largest possible HTML fragments that were either entirely template or non-template. These HTML fragments correspond to nodes in the webpage DOM tree. Hence, for each webpage we labeled an antichain of nodes through the DOM tree, forming an exhaustive and disjoint cover of all leaf nodes.

Random. In order to evaluate the algorithms on webpages more representative of the general Web, we manually labeled 100 pages selected uniformly at random from the DMOZ directory[6]. The selected set of webpages is a mix of topically focused content or hub pages and entry points to larger websites. As was done for the Common dataset, the labeling process identified for each webpage an antichain of DOM nodes and marked each node as either template or non-template.

[6] `www.dmoz.org`

Dataset		PageLevel Basic	PageLevel Smooth
Common	Text	0.56	**0.60** (7.1% ↑)
	AT	0.65	**0.71** (9.2% ↑)
	Links	0.69	**0.73** (5.8% ↑)
Random	Text	0.63	**0.66** (4.8% ↑)
	AT	0.71	**0.73** (2.8% ↑)
	Links	0.75	**0.77** (2.7% ↑)

Table 5.4: Accuracy of PageLevel on Common and Random datasets in terms of F-measure.

Template Detection Accuracy. As we demonstrate later in this section, text and links present within the template regions mislead standard web-mining algorithms for tasks such as duplicate detection and automated classification. Here we measure the efficacy of PageLevel in identifying text and links that occur within templates. We report accuracy in terms of F-measure, which is the harmonic mean of precision (p) and recall (r); i.e. $f = 2pr/(p + r)$. In the current setting, precision is the fraction of words (links) identified by PageLevel as occurring within templates that are also manually placed within templates. Recall is the fraction of all words (links) manually labeled as lying within template regions that PageLevel also correctly identifies as templates. This evaluation setting has previously been used by Vieira et al. [VSP+06].

In Table 5.4, we present the accuracy numbers (in terms of F-measure) achieved by PageLevel for the two datasets: Common and Random. We present accuracies for two variations of our approach: PageLevel Basic only applies the classifier to the DOM nodes individually, while PageLevel Smooth, in addition also performs Isotonic smoothing on the templateness scores. The performance is measured along multiple dimensions: Text, Anchor-

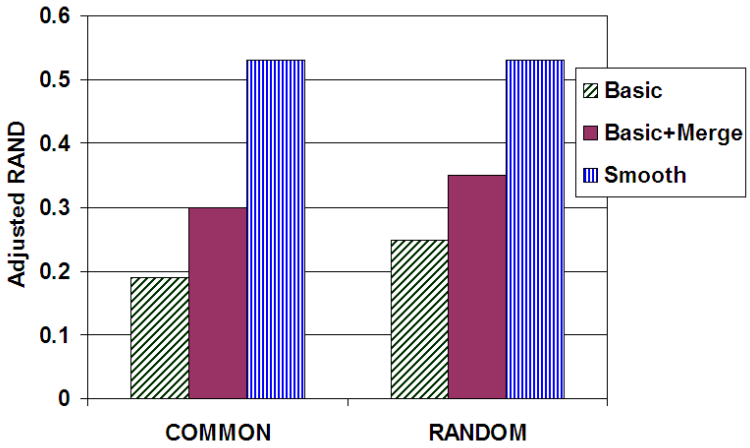


Figure 5.8: Segmentation performance of PageLevel Basic, PageLevel Basic+Merge, and PageLevel Smooth.

text (AT), and Links. As is clear from the table, our approach is very effective in identifying all types of page content that lies within template regions. Furthermore, smoothing is shown to significantly improve accuracy across all dimensions of evaluation, in some cases by almost as much as 10%. An interesting observation is that the accuracy over the Common dataset is slightly lower than that over Random. This is because the template structure in webpages in Common is more extensive than those in Random. Further, note that the gains afforded by the isotonic smoothing are larger on the more difficult of the two datasets.

Segmentation Accuracy. As we mentioned in Section 5.5, a by-product of the isotonic smoothing algorithm is a segmentation of the page into DOM nodes that act as the roots of the template and non-templates regions. Here, we show that the segmentation found by PageLevel closely matches the manually labeled segments.

Notice that the manual segmentation, an antichain of nodes of the DOM tree, induces a grouping of the leaves in which each node in the segmentation defines a group. A leaf then belongs to the group corresponding to the segment node that covers it. Similarly, the segmentation output by PageLevel, even though it allows for nested segments, also induces a grouping of leaves. Each leaf can be considered as belonging to the group corresponding to its closest ancestor in the segmentation. Hence, we can evaluate the PageLevel segmentation against the manually labeled one by comparing the corresponding groupings using the adjusted RAND index [HA85]. The adjusted RAND index is a measure of how similar two groupings are, i.e., whether pairs of objects (leaves) are together in both groupings, or in different groups in both groupings. It is used as a preferred measure of agreement between clusterings [MC86]. The value of the adjusted RAND index is upper bounded by 1, and its expected value for a random clustering is 0.

In Figure 5.8 we plot the accuracy of PageLevel segmentation in terms of adjusted RAND. The PageLevel Basic and PageLevel Smooth approaches have been described above. As we can see the PageLevel Basic algorithm achieves close to random results, but this is expected since it is not performing any smoothing of scores and hence almost every leaf is in a group of its own. In contrast the segmentation discovered by the isotonic smoothing function (PageLevel Smooth) conforms very well to the manually labeled segments. In order to put the accuracy of PageLevel Smooth in context, we also present numbers for a PageLevel Basic + Merge heuristic. This approach does a "naive" smoothing of the classifier scores by grouping adjacent leaves together when their templateness scores differ by less than δ (the best δ was found by exhaustive search). As we can see from the plot, "merging" improves the scores

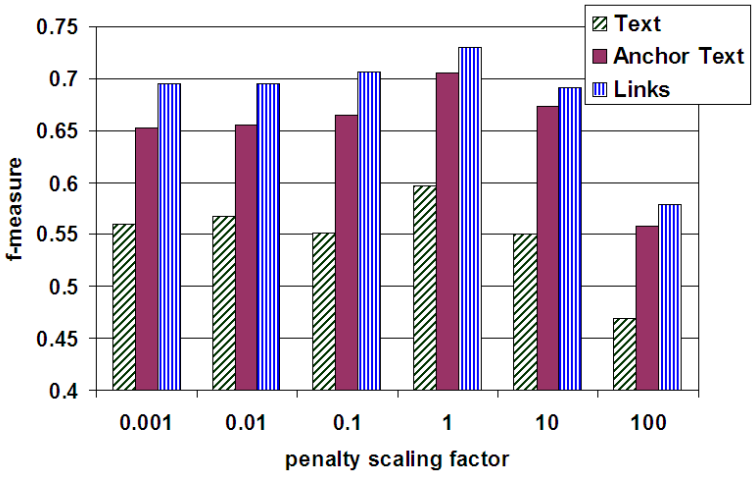


Figure 5.9: Variation in template detection accuracy on the COMMON dataset with changing values of penalty. The x-axis represents the factor being multiplied into the penalty.

of the PAGELEVEL Basic; however, the results are still far lower than those achieved by isotonic smoothing. This shows that the smoothing operation is constructing highly non-trivial segmentations of webpages.

Effect of Variations in Penalty. We have shown above that PAGELEVEL successfully obtains and labels template segments within webpages. Here we discuss the sensitivity of our approach to penalty parameters in the isotonic smoothing function.

Figure 5.9 plots the variation in template detection accuracy on the COMMON dataset with changing values of penalty. In the plot, the x-axis represents the factor multiplied into the penalty in order to vary it. As we can see, an increase or decrease in penalty results in an decrease in the template detection accuracy. However, the decrease is larger with higher values of penalty as this results in very few segments and hence a mixing up of template and

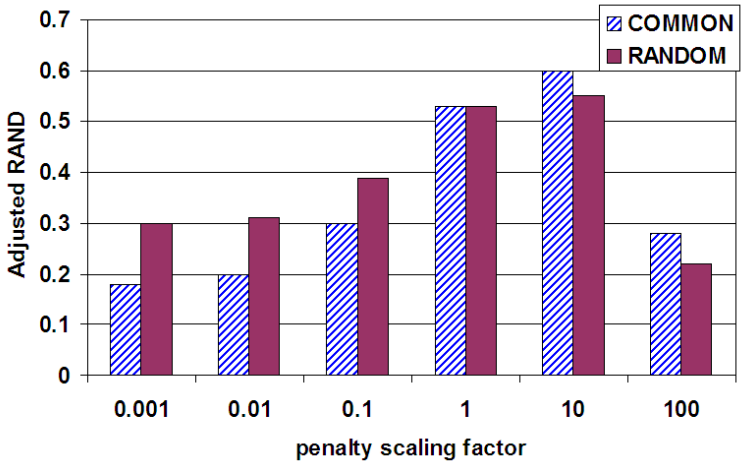


Figure 5.10: Variation in segmentation accuracy on both datasets with changing values of penalty. The x-axis represents the factor being multiplied into the penalty.

non-template structures into the same segment. Lower values of penalty do not give us the improvements inherent in smoothing, but they do not reduce the discriminative power of the Basic classifier. The same behavior is seen for RANDOM as well.

Variations in segmentation accuracy on both datasets with changing values of penalty are plotted in Figure 5.10. Just as in the case of template detection accuracy, the segmentation accuracy also forms a unimodal curve, dropping with high and low values of penalty. However, the drop in segmentation accuracy is larger for changes in penalty values as compared to drop in detection accuracy. This is because the smoothing impacts the segmentation performance more directly, as compared to detection performance. As we increase (decrease) penalty values the number of groups of leaves obtained are lesser (greater) than the manually labeled groups. Both these changes negatively impact the segmentation performance. Another interesting difference is

that template detection accuracy achieves high values even when segmentation performance is not at its peak. The reason is that the manual labeling is binary (template or non-template), while the segments we find are labeled with real numbered scores. Hence, we can still achieve a high template detection accuracy when the smoothing function places leaves into groups smaller than those in the manual labellings. However, groups smaller than those in the manual labeling causes a decrease in segmentation accuracy. This indicates that if achieving good segmentation is our primary objective, using a slightly higher value of penalty might be advantageous.

To summarize, in this section we showed that PageLevel accurately segments webpages, and also labels the segments appropriately as template or non-template. Further, we showed that isotonic smoothing is critical to its success, contributing to increases in both segmentation and template detection accuracies. We were unable to provide any comparisons with the site-level approach on the human labeled data, since SiteLevel needs many pages from each website in order to make template judgments for pages.

Next we show that webpage template detection is very useful as a preprocessing step in several applications, such as finding webpages with duplicate content, and webpage classification. Furthermore, since in these datasets we have several webpages from the same website available, we also present an evaluation comparing PageLevel with the site-level template detection approach.

5.7.2 Application to Duplicate Detection

Duplicate webpages and mirrored websites present challenging problems to web search engines that crawl and index them. Duplicated pages use up valuable index space and duplicate results returned for search queries spoil the user experience. Hence, detection of duplicates on the Web in a scalable fashion has been the subject of much research [BBDH00, BGMZ97, CSGM00]. Most duplicate detection methods rely on the concept of shingles. For each webpage, shingles are extracted by moving a window of fixed length over the text and the ones with the N smallest hash values are stored. Two documents that share shingles are then considered to be near-duplicates.

Problems Caused by Templates. The templates regions often contain text whose purpose is orthogonal to the main content of the webpage. Hence, this templated content must not be used while making decisions about whether pages are duplicates. For example, text present within navigation bars, copyright notices etc., must not be compared when two pages are being checked for duplicate material. The presence of templated content of webpages can foil duplicate detection algorithms whenever the shingling process retains shingles from the templated regions. Two pages that have absolutely the same content, say the exact same AP news story repeated across two different news websites, might be considered non-duplicates if the shingling process retains shingles from the template regions of the webpages as this portion of the webpages is different. This can lead to false negatives and cause us to return duplicate results for queries. Similarly, two webpages with the same templated content but different main content might be considered duplicates if all the shingles hit the templated region. This can result in false positives and cause

us to ignore valuable content on the web. In this section we evaluate the effect of templates on duplicate detection performance, and also also compare the template detection performance of PageLevel to the site-level approach.

The Lyrics Dataset. We constructed the Lyrics dataset by obtaining the webpages containing lyrics for the same song from three different websites. This way we knew that the webpages from different websites containing lyrics to the same song should be considered duplicates[7]. We also knew that webpages containing lyrics of different songs, irrespective of what website they come from, should be considered non-duplicates. We were able to obtain 2359 webpages from the websites `www.absolutelyrics.com`, `www.lyricsondemand.com`, and `www.seeklyrics.com` containing lyrics to songs by artists ABBA, BeeGees, Beatles, Rolling Stones, Madonna, and Bon Jovi. We chose to get lyrics by a few diverse artists in order to minimize the possibility of cover songs. The Lyrics dataset consists of 1711 duplicate pairs (webpages with lyrics of the same song from different websites) and 2058 non-duplicate pairs (webpages with lyrics of different songs from the same website).

Experimental Setup. SiteLevel was run on all the pages on each lyrics website and the threshold parameter was set to 10%. This setting was seen to perform well in [GPT05].

We used a standard shingling process. Before the shingling was performed the text of the webpage is made lowercase and only alphanumeric characters are retained. Shingles are computed over moving windows of 6 consecutive words each, and the 8 minimum hashes are stored for each webpage. A pair

[7] Actually, these might only be near-duplicates, due to transcription errors on the different pages. However, this affects all algorithms equally.

	Total Pairs	PageLevel	SiteLevel	FullText
Dup	1711	**1299** (**76**%)	730 (42.7%)	529 (30.9%)
Non-Dup	2058	**1885** (**91.6**%)	1712 (83.2%)	1781 (86.5%)

Table 5.5: Number of duplicate and non-duplicate pairs detected by the shingling approach after removing templates detected by PageLevel and SiteLevel . FullText indicates no template detection and removal.

of pages is tagged as a duplicate if there are at least 4 matching hashes out of the 8 for each webpage. We run this experiment under different settings: (1) all segments (and words) are used (FullText), (2) only segments tagged as non-template by PageLevel are used, and (3) only segments tagged as non-template by SiteLevel are used for shingling.

Results. The results of our duplicate detection experiments are presented in Table 5.5. Not detecting and removing template content (FullText) performs very badly, especially in flagging duplicate pairs. Using the PageLevel approach to clean the data before shingling recovers 76% of the duplicate pairs, and 92% of non-duplicate ones. These represent an improvement of 140% and 6% respectively over the FullText approach. Finally, PageLevel also outperforms the SiteLevel template detection approach by a large margin in both flagging duplicate and non-duplicate pairs.

The Lyrics dataset is not representative of the density of duplicates and non-duplicate pairs found on the web; we created it to highlight the problems posed by templates to duplicate detection algorithms. Hence, while the numbers seen in these experiments will not apply exactly to the web in general, the results are indicative of the benefits of template detection and removal, and the dataset serves as an appropriate test-bed for comparing the algorithms

PageLevel and SiteLevel.

Discussion. Why does PageLevel outperform SiteLevel even though it is trained on the output of the latter? Comparing errors performed by the two on the LYRICS dataset offers us an opportunity to investigate this. As stated before, errors occur when shingles come from templated regions of the page. Many of the errors committed by SiteLevel involved shingles from a segment on "Popular lyrics by this Artist" that seemed to change based on the artist whose song lyrics were being displayed. Since this segment changed within webpages on the same site, SiteLevel was unable to identify it as a template. However, thanks to the careful selection of DOM node features in the page-level classifier, PageLevel generalizes beyond the site-level training data. Thus, it picked out such segments as templates, boosting its accuracy significantly.

5.7.3 Application to Webpage Classification

Automated classification of webpages is a well-studied problem and numerous approaches have been proposed for it. While many sources of information like hyperlinks [CDI98], site structure [KPT06], etc. are often used, techniques for classifying the text content of webpages [Mit97] form the mainstay of webpage classification. In this section we perform experiments on the effect of template content on the classification of textual content of webpages, and show that template removal using PageLevel gives a boost in accuracy.

Problems Caused by Templates. Even though classification algorithms are very good at identifying and removing noisy features, in certain scenarios templates can present a challenging problem. Consider a binary classification

problem between classes *Camera* and *Notebook*. If the template terms (noise) in both classes are the same, a classifier would be able to detect and remove it, say, using the correlation of features to the class labels. However, the noisy features could differ across the two classes; say, the webpages in *Camera* class come from CNET, and those in *Notebook* class come from PCConnection, the classifier will not be able to remove the template content automatically, making template detection as a pre-processing step imperative.

Dataset and Experimental Setup. For the classification experiments we used a subset of the dataset used by Vieira et al. [VSP+06]. The dataset consisted of webpages on 5 topics (*Camera, Notebook, Mobile, Printer, TV*) obtained from 4 websites, CNET, J&R, PCConnection, and ZDNET. Details of this dataset can be obtained in [VSP+06]. From this data, we constructed binary classification problems in which training data for classes C_1 and C_2 were taken from different websites, and the rest of the data for these classes were used as test data. For instance, in one binary problem, C_1 is *Camera* and C_2 is *Notebook*. The training data for C_1 and C_2 comes from CNET and PCConnection respectively. The test data for C_1 then comprises J&R, PCConnection, and ZDNET, while that for C_2 comprises CNET, J&R, and ZDNET. This evaluation setting has been used previously [VSP+06, YLL03]. We employed a Naive Bayes classifier[8] for the binary classification problems. The classification accuracy numbers we report are averaged over all possible binary classification problems of the type mentioned above.

We run this experimental setup with different amounts of webpage cleaning: (1) with all segments (and words) (FullText), (2) with only segments

[8] www.cs.cmu.edu/~mccallum/bow/

Categories		PageLevel Smooth	PageLevel Basic	SiteLevel	FullText
camera	mobile	60.17	59.57	**64.17**	55.10
camera	notebook	**40.03**	35.24	35.61	30.48
camera	printer	**41.18**	39.75	38.84	32.76
camera	tv	37.21	39.51	**40.67**	34.82
mobile	notebook	66.92	64.94	**70.26**	60.45
mobile	printer	**28.55**	24.5	24.16	21.06
mobile	tv	24.79	**24.85**	23.86	23.03
notebook	printer	**53.91**	48.7	43.95	39.9
notebook	tv	**50.94**	50.2	48.85	43.47
printer	tv	47.57	**48.7**	44.12	41.17
Average		**45.13**	43.6	43.45	38.22

Table 5.6: Averaged classification accuracies on 2-class problems. The training data for the two categories was selected from different websites causing template content to be learned as discriminating features. Moreover, the test instances followed an adversarial distribution which made the problem extremely difficult. The best accuracies for each class combination are in **bold**.

tagged as non-template by SiteLevel, (3) only segments tagged as non-template by the PageLevel Basic algorithm (4) only segments tagged as non-template by the PageLevel Smooth algorithm. Recall that in PageLevel Basic algorithm the smoothing function is disabled and only raw templateness classifier assigned scores are used.

Results. The results of the classification experiments are presented in Table 5.6. The best accuracies for each class combination are highlighted in bold. First, we point out that the problem that we constructed above is extremely hard and hence results in accuracies that are well below that of random guessing. In other words, the classifiers are actively misleading us in terms of classifying test instances. Since we used training data for each class from a different website, the template structure (which is very uniform within each class) is learned as excellent discriminants. Since the test set is adversarial in

nature in that it is devised to actively fool the classifiers - instances of topic of class 2 from the website used in class 1, and vice versa, are included in the test set - the classifiers perform worse than random guessing.

However, as we can see, using template detection as a pre-processing step always improves the classification accuracy of the Naive Bayes classifier. Furthermore, webpage cleaning via the PageLevel algorithm outperforms SiteLevel on a majority of class combinations. Even among the PageLevel approaches, the use of isotonic smoothing of the templateness classifier's output results in better template detection and removal, as evidenced by an increase in the Naive Bayes classifier accuracy. In the final analysis, webpage template detection and removal via our PageLevel system increases classification accuracy on this dataset by an average of 18%.

5.8 Conclusions

In this chapter, we reported on our studies of templated content on the Web that showed that templates are a significant and growing problem. We then presented a framework for classifier based page-level template detection that constructs the training data and learns the notion of "templateness" automatically using the site-level template detection approach. We formulated the smoothing of classifier assigned templateness scores as a regularized isotonic regression problem on trees, and presented an efficient algorithm to solve it exactly; this may be of independent interest. Using human-labeled data we empirically validated our system's performance, and showed that template detection at the page-level, when used as a pre-processing step to webmining applications, such as duplicate detection and webpage classification, can boost accuracy significantly.

Chapter 6

Hierarchical Topic Segmentation of Websites

6.1 Introduction

As the major established search engines vie for supremacy, and new entrants explore a range of technologies to attract users, we see researchers and practitioners alike seeking novel analytical approaches to improve the search experience. One promising family of approaches that is generating significant interest is analysis at the level of websites, rather than individual webpages. There are a variety of techniques for exploiting site-level information. These include detecting multiple possibly-duplicated pages from the same site [Aum03, BBDH00], determining entry points [CHR01], identifying spam and porn sites [KSGM03], detecting site-level mirrors [BBDH00], extracting site-wide templates [GPT05] and structures [RT00], and visualizing content at the site level [Gib04].

In this chapter we consider *site segmentation*, a particular form of site-level analysis in which a website must be segmented into one or more largely uniform regions. The segmentation may be performed based on the topics discussed in each region, or based on the look and feel, or based on the authorship, or other factors. We focus specifically on *topical segmentation*, i.e., segmenting a site into pieces that are largely uniform in the topics they discuss. Such a topical segmentation offers many potential advantages:

1. Various algorithms that are currently applied to websites could more naturally be applied to topically-focused segments.
2. Websearch already incorporates special treatment for pages that are known to possess a given topic—for instance, many engines provide a link to the topic in a large directory such as the Yahoo! Directory, Wikipedia, or the Open Directory Project. These approaches can naturally be extended when several pages from a search result list lie within a topically-focused segment.
3. The resultant segments provide a simple and concise site-level summary to help users who wish to understand the overall content and focus of a particular website.
4. A host such as an ISP may contain many individual websites, and a topical segmentation is a useful input to help tease out the appropriate granularity of a site.
5. Website classification is a problem that has been addressed using primarily manual methods since the early days of the web, in part because sites typically do not contain a single uniform class. Segmentation is an important starting point for this larger problem.

Site segmentation may be viewed from two distinct perspectives. First, it may be viewed as a constrained clustering problem in which the allowable segments represent constraints on the possible clusters that the algorithm may return. At the same time, site segmentation may be viewed as an extended form of site-level classification in which the algorithm may choose to classify either the entire site, or various sub-sites. The measure we propose for the quality of a segmentation is much simpler than standard measures from machine learning. As a result, while the problem may be viewed as a constrained version of the NP-hard clustering problem, or an extended version of classification that incorporates a search for the appropriate objects to classify, the simple measure of segmentation quality, combined with the class of allowable segmentations, will allow us to provide an algorithm to return the *optimal* segmentation in polynomial time. To achieve this bound, we employ a dynamic programming algorithm that is quite different from traditional algorithms for either clustering or classification.

One could consider many different classes of allowable segmentations of a website, for example based on the hierarchical structure of the site, or based on clusters in the intra-site link graph, or based on regions of the site that display some commonality of presentation template, and so forth. We will focus specifically on segmentations that respect the hierarchical structure of a website, for two reasons. First, we believe that of the many possible approaches to segmenting websites, hierarchy is the most natural starting point. Site authors often think in terms of a site being made up of several sub-sites, each of which may contain sub-structure of its own; and the layout of pages on a website often follows a "folder" structure inducing a natural hierarchy. And second, in many applications an individual segment must be returned to the

user in some succinct manner. Rather than simply returning a long list of URLs located at various positions within the site, it is desirable to return instead a pointer to a particular sub-site.

In general, the hierarchical structure of a website may be derived from the tree induced by the URL structure of the site, or mined from the intra-site links or the page content of the site. Our algorithm makes use of whatever hierarchical information is available about a site to constrain the possible segmentations. We show that 85-90% of sites exhibit a non-trivial form of hierarchy based on the URL tree that can exploited by our algorithm for segmentation. The remaining fraction of sites might have a latent hierarchical structure that could be mined by further analysis of intra-site links or content, but that is beyond the scope of this work.

Thus, our work is on *hierarchical topic segmentation* (HTS): the segmentation of websites into topically-cohesive regions that respect the hierarchical structure of the site.

Formulation.

Consider a tree whose leaves have been assigned a class label or a distribution on class labels, perhaps by a standard page-level classifier. A distribution is induced on an internal node of the tree by averaging the distributions of all leaves subtended by that internal node. These distributions, along with a hierarchical arrangement of all the pages in the site, are provided to the HTS algorithm. The algorithm must return a set of segmentation points that optimally partition the site. The objective function for the segmentation is a combination of two competing costs: the cost of choosing the segmentation points (the nodes) themselves and the cost of assigning the leaves to the closest

chosen nodes. Intuitively, the *node selection* cost models the requirements for a node to serve as a segmentation point, while the *cohesiveness* cost models how the selection of a node as a segmentation point improves the representation of the content within the subtree rooted at it. For example, in a particular instance of the problem, the node selection cost can capture the requirement that the segments be distinct from one another and the cohesiveness cost can capture the requirement that the segments be pure. The underlying tree structure enables us to obtain an efficient polynomial-time algorithm to solve the HTS problem.

To complete the overview of HTS, we provide a brief discussion of the difference between segmentation and classification. The general website classification problem tries to assign topics to websites by employing features that are broad and varied. A few example features for this broader problem include the topic of each page, the internal hyperlinks on the site, the commonly link-to entry points to the site, with their anchor-text, the general external link structure, the directory structure of the site, the link and content templates present on the site, the description, title, and h1-6 tags on key pages on the site, and so forth. The final classes in a website classification problem may be distinct from the classes employed at the page level. HTS, on the other hand, specifically focuses on aggregating the topic labels on webpages into subtrees according to the hierarchy of a site, in order to convey information such as, "This entire sub-site is about Sports." Thus, HTS attacks the problem of determining whether and how to split the site, but is only the beginning of a broader research problem of classifying websites using rich features. The broader problem is of great interest in both binary cases (is the site spam? is it porn?) and multi-class cases (to what topics should I assign this site?). We

believe that a clean and elegant solution to the HTS problem is essential to fully address the more general site classification problem.

Our Contributions.

We provide a rigorous formulation of HTS for websites that is general enough to capture many different hierarchical topic segmentation schemes. We show how to encode two natural requirements within our formulation: the segments themselves should be sufficiently 'distinct' from each other and the webpages in a segment should be reasonably 'pure' in topic. We also present a polynomial-time algorithm to solve the HTS problem optimally.

We conduct an extensive set of experiments to evaluate the performance of our algorithm with various natural cost measures on hand-labeled as well as semi-synthetic websites. We show that a judicious choice of the node selection cost and cohesiveness cost can vastly improve the performance of the algorithm.

Organization.

Section 6.2 presents the framework for the HTS problem. Section 6.3 contains algorithm for the HTS problem as well as definitions for the cohesiveness and node selection costs. Finally, the experimental results are presented and discussed in Section 6.4.

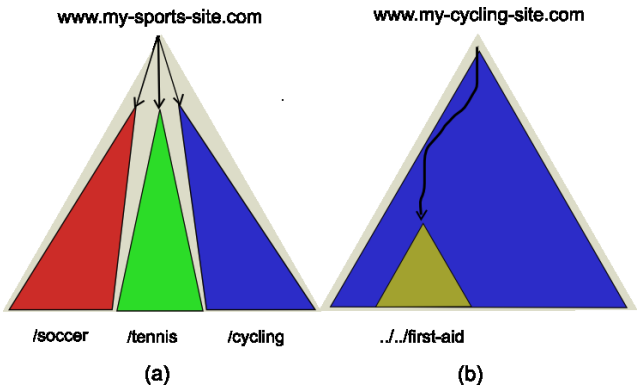


Figure 6.1: Two websites with different organization of topics along the URL directory structure.

6.2 Formulation

We begin this section with a brief discussion of HTS, and then present a general mathematical formulation.

6.2.1 Hierarchical Topic Segmentation

Consider the problem of describing the topical content of a website to a user. If the site is topically homogeneous we could provide the user with the URL of the site and a topic label representing the content. Our segmentation algorithms should do exactly this. However, most sites are not homogeneous, and in fact the organization of topics within directories can determine the best way to summarize site content for the user.

For instance, consider the two hypothetical websites shown in Figure 6.1. The site in panel (a) contains sub-sites on different topics, while the site in panel (b) contains a single topically coherent tree expect for a small directory deep in the site structure. In the first case we could describe the site using the top-level directories, such as `www.my-sports-site.com/tennis`, and for each such directory give its prevailing topic, such as Sports/Tennis.

For panel (b) on the other hand, we could tell the user that the entire site (`www.my-cycling-site.com`) is about Sports/Cycling, except that a small piece at `www.my-cycling-site.com/.../first-aid/` is about Health/First-Aid. As this example shows, it is quite reasonable to describe a site using nested directories if this is the best explanation for the content.

In general, we wish to make optimal use of the user's attention and convey as much information about the site as possible using the fewest possible directories, i.e., internal nodes. Hence, each directory we call out to the user should provide significant additional information about the site.

This informal description of the problem is in terms of explaining the contents of a website to a user. The other application areas listed in Section 6.1 leverage the same framework, but make use of the final description in other ways. Generally, the goal is to return a concise segmentation of a website into topically coherent regions.

Here we note that while in this chapter we restrict ourselves to segmentations that follow the directory (URL) tree, our approach can be applied to any hierarchical structure within a website. Indeed, websites with trivial URL based hierarchical structure, for instance dynamic pages with URLs of the form `http://mysite.com/show.php?productid=42`, are increasing in number, especially in the e-commerce domain. However, besides being the first step to study the segmentation problem, our restriction to URLs captures the vast majority ($85 - 90\%$) of websites, and allows us to study how to make use of this key element of site structure. Our approach could be applied to the remaining websites by first mining their latent hierarchical structure by a deeper analysis of links, content, or URL [BYKS07], but that is beyond the scope of this chapter.

6.2.2 Formal Definition

The natural approach to modeling a directory structure is by a rooted tree whose leaves are individual pages.[1] We assume that there is a page-level classifier that assigns class labels or a distribution over class labels to each page of the directory structure. This induces a distribution on the internal nodes of the tree as well, by uniformly combining the distribution of all descendant pages. Our notion of cohesiveness of a subtree will be based upon the agreement between each leaf with the distribution at its parent. We require a few definitions to make this notion formal.

Let T be a rooted tree with n leaves; let leaf(T) denote the leaves of T and let root(T) denote its root. Let Δ be the maximum degree of a node in T. Let L be the set of class labels. We assume that each leaf x in the tree T has a distribution p_x over L, generated by some page-level classifier. We will write $p_x(i)$ to denote the probability that leaf x has class label i. For an internal node u with leaves leaf(u) in the subtree rooted at it we define the distribution of labels at u as follows:

$$p_u(i) = \frac{1}{|\text{leaf}(u)|} \sum_{x \in \text{leaf}_{(u)}} p_x(i).$$

A subset S of the nodes of T is said to be a *segmentation* of T if, for each leaf x of T, there is at least one node $y \in S$, such that x is a leaf in the subtree rooted at y. For example, S is always a segmentation if root(T) $\in S$. Given a parameter k, the goal now is to find a segmentation of size at most k whose components are cohesive. For a leaf $x \in$ leaf(T) let $S_x \in S$ be the first element

[1] If internal nodes also correspond to pages, we simply model them using the standard "index.html" convention.

of S on the ordered path from x to root(T). We will say that x *belongs* to S_x, and we will define a *cohesiveness* cost $d(x, S_x)$ that captures the cost of assigning x to S_x. Further, we will define a *node selection* cost $c(y, S)$ that gives the cost of adding y to S. The overall cost of a particular segmentation S is then

$$\beta \sum_{y \in S} c(y, S) + (1 - \beta) \sum_{x \in \text{leaf}(T)} d(x, S_x), \tag{6.1}$$

where β is a constant controlling the relative importance of the node selection cost and the cohesiveness cost. Our algorithms then find the lowest-cost segmentation, given functions $c(\cdot)$ and $d(\cdot)$ representing the problem instance.

The formulation in (6.1) is reminiscent of the uncapacitated facility location (UFL) problem in combinatorial optimization. In UFL, we are given a graph (V, E), a parameter k, and each vertex v has a cost $c(v)$ and the goal is to choose $S \subseteq V$ with $|S| = k$ such that $\sum_{v \in S} c(v) + \sum_{u \in V} \min_{v \in S} d(u, v)$ is minimized. Here, d is the graph metric defined by E. In this most general version, UFL is NP-hard. In our case, G is only a tree and the distance function is more general and is not necessarily a metric.

6.3 Segmentation Algorithm

Our algorithmic approach is based on a general dynamic program that optimizes the objective function of (6.1). This dynamic program works for any cohesiveness cost $d(\cdot)$ and node selection cost $c(\cdot)$. It runs in time $O(k^2nd)$. After describing the dynamic program, we then present a set of candidate cohesiveness costs and node selection costs. We compare these different approaches empirically in Section 6.4.

6.3.1 A Generic Algorithm

The idea behind the dynamic program is the following. Given a subtree whose root has δ children, the optimal way of adding at most k nodes from the subtree to the segmentation must follow one of two patterns. In the first pattern, we add the root of the subtree and then recurse on the children with a budget of $k-1$. In the second pattern, we do not include the root of the subtree, and instead recurse on the children with a budget of k. A naive way of implementing the recursion would result in segmenting k (or $k-1$) into δ pieces in all possible ways. This is expensive, if $\delta \gg 2$. To circumvent this, we show a simple transformation that will convert the tree to binary, without changing the optimum.

Construct a new tree from the original tree T in the following way, starting from root(T). Suppose y is an internal node of T with children $y_1, \ldots, y_\delta$ and $\delta > 2$. Then, this node is replaced by a binary tree of depth at most $\lg \delta$ with leaves $y_1, \ldots, y_\delta$. The cost $c(\cdot)$ of $y, y_1, \ldots, y_\delta$ are the same as before and the cost of the newly created internal nodes are set to ∞; this is so that they never get selected in any solution. The construction is recursed on each of $y_1, \ldots, y_\delta$. It is easy to see that the optimum solution of (6.1) on the new tree is the same as on T. Furthermore, the size of the new tree at most doubles and the depth of the tree increases by a factor of $\lg \Delta$, where Δ is the maximum degree of a node in T. This construction has been used previously; see for instance [FGK$^+$05, Tam96].

From now on, we will assume that the tree is binary. Let S denote the current solution set. Let $C(x, S, k)$ be the cost of the best subtree rooted at node x using a budget of k, given that S is the current solution. Recall that S_x, if it exists, is the first node along the ordered path from x to the root of

the tree T in the current solution S. If S_x exists, then all $x' \in \text{leaf}(T_x)$ (leaves in the subtree under x) can always be covered by S_x, each with cost $d(x', S_x)$.

Let x_1, x_2 denote the two children of x. The update rule for the dynamic program is given by

$$C(x, S, k) = \min \begin{cases} \min_{k'=0}^{k}(C(x_1, S, k') \\ \quad + \ C(x_2, S, k - k')) \\ c(x, S) + \min_{k'=0}^{k-1}(C(x_1, S \cup \{x\}, k') \\ \quad + \ C(x_2, S \cup \{x\}, k - k' - 1)). \end{cases} \tag{6.2}$$

The top term corresponds to not choosing x to be in S and the bottom term corresponds to choosing x to be in S.

The base cases for the dynamic program are

- $C(x, S, k)$ where $x \in \text{leaf}(T)$. If we forbid including leaves in the solution and S_x doesn't exist, we set this cost to be ∞. If leaves of T are permitted to be part of the solution, the cost is given by

  $$C(x, S, k) = \begin{cases} \min\{c(x, S), d(x, S_x)\} & \text{if } S_x \text{ exists} \\ c(x, S) & \text{otherwise} \end{cases}$$

  (Note that if exactly k nodes are desired, then we can set $C(x, S, k)$ to ∞ whenever $k > 1$.)

- $C(x, S, 0)$, where the cost is given by

  $$C(x, S, 0) = \begin{cases} \sum_{x' \in \text{leaf}(T_x)} d(x', S_x) & \text{if } S_x \text{ exists} \\ \infty & \text{otherwise} \end{cases}$$

  This corresponds to assigning all nodes in the subtree T_x to the node S_x, if it exists, since we are out of budget in this case.

The dynamic program is invoked as $C(\text{root}(T), \emptyset, k)$. There are $knd \lg \Delta$ entries in the dynamic programming table and each update of an entry takes

$O(k)$ time as given by (6.2). So, the total running time of the dynamic program is $O(k^2 \cdot n \cdot d \cdot \lg \Delta)$. Note that in the dynamic program, we compute $c(x, S)$ in terms of a partial solution set S that has been constructed so far and not in terms of the final solution set S as indicated in (6.1); however, since $c(x, S)$ depends only on the elements in S that are on the path from x to root(T) and since we compute S top down, these are equivalent.

Proposition 6.1. *The above algorithm solves the optimization problem in (6.1). The running time of the algorithm is* $O(k^2 \cdot n \cdot d \cdot \lg \Delta)$.

Notice that the node selection cost $c(\cdot)$ is helpful to incorporate heuristic choices and requirements. For instance, we might want to ensure that if two nodes, one of which is a parent of the other, are chosen in the solution, then they are guaranteed to have different distributions. This is accomplished by setting $c(\cdot)$ for the child node to be sufficiently high in such situations.

6.3.2 Cost Measures

We now suggest some different variants of the node selection cost c and the cohesiveness cost d that appear in (6.1). The different choices of these functions result in algorithms that make different trade-offs and are optimized for different conditions.

6.3.2.1 Cohesiveness Costs

We propose three cohesiveness costs that capture the purity of a topical segment. The first cost is based on information theory and the next two are geometric in nature.

KL Cost Measure. This cost measure is based on the Kullback–Leibler divergence in information theory. For every page x and the node S_x to which it belongs we define the cost of the assignment to be

$$d(x, S_x) = \mathrm{KL}(p_x \parallel p_{S_x}) = \sum_{\ell \in L} p_x(\ell) \log \left(\frac{p_x(\ell)}{p_{S_x}(\ell)} \right).$$

The KL-divergence or the relative entropy of two distributions p_x and p_{S_x} over an alphabet L is the average number of extra bits needed to encode data drawn from p_x using a code derived from p_{S_x}. This corresponds to minimizing the wastage in description cost of leaves of the tree using the internal nodes that are selected. This property makes the KL-divergence an intuitive choice for the cohesiveness cost.

Squared Euclidean Cost Measure. The distance between a leaf x (webpage) and an internal node S_x (directory) can be computed using the squared Euclidean distance between the corresponding class distributions. Therefore,

$$d(x, S_x) = \|p_x - p_{S_x}\|^2 = \sum_{\ell \in L} |p_x(\ell) - p_{S_x}(\ell)|^2.$$

The sum of squared Euclidean cost has previously been extensively used in many applications.

Cosine Cost Measure. Drawing from information retrieval, the negative cosine dissimilarity measure may be employed as a cohesiveness cost, as follows:

$$d(x, S_x) = -\langle p_x, p_{S_x} \rangle = -\sum_{\ell \in L} p_x(\ell) p_{S_x}(\ell).$$

The cosine cost measure has previously been successfully used for clustering documents [BDGS05].

6.3.2.2 Node Selection Costs

Having presented three possible cohesiveness costs $d(\cdot)$, we turn now to the node selection cost $c(\cdot)$, representing the penalty for adding a new element into S. Our goal is to penalize a new node if it provides little information beyond its parent. We propose a cost measure to implement this condition, which we call the α-measure. This cost measure in the context of decision tree induction was introduced by Quinlan [Qui] and is referred to as *information gain ratio*. It is defined as follows.

Let T be a tree consisting of subtrees $T_1, \ldots, T_s$. Say we wish to encode the label of a particular leaf of T, and are allowed two possible encoding schemes. In the first scheme, we simply communicate the label using an optimal code based on the distribution of labels in T. In the second scheme, we first communicate whether or not the designated leaf lies in T_1, and then encode the label using a tailored code for either T_1 or $T \setminus T_1$ as appropriate. The second scheme corresponds to adding T_1 to the segmentation. Its overall cost cannot be better than the first, but if T_1 is completely distinct from $T \setminus T_1$ then (and only then) the cost of the second scheme will be equivalent to the first. Let $p_1 = |T_1|/|T|$ be the probability that a uniformly-chosen leaf of T lies in T_1. Then the cost of communicating whether a leaf lies within T_1 is $H(p_1)$. In the worst case, T_1 will look identical to $T \setminus T_1$ and the second scheme will actually be $H(p_1)$ bits more expensive than the first: the information about the subtree provides no leverage to the user. We may therefore characterize the value of subtree T_1 relative to its parent by asking where on the scale between $H(T)$ and $H(T) + H(p_1)$ the cost of the second scheme lies. With this intuition in mind, we now provide the formal definition of the cost measure.

Let x denote the current node we are considering adding to the solution

S. Recall that S_x is its nearest parent that is already a part of the solution S. We assume S_x exists (we will discuss this restriction further below) and for simplicity, denote S_x by y. Then let x' be a hypothetical node such that $\text{leaf}(T_{x'}) = \{\text{leaf}(T_y) \setminus \text{leaf}(T_x)\}$, i.e., the leaves under the subtree rooted at y but not x. Let $n = |\text{leaf}(T_y)|, n_x = |\text{leaf}(T_x)|$, and $n_{x'} = |\text{leaf}(T_{x'})|$. Here, the split cost is $H_2(n_x/n)$, the binary entropy. Then, the α-measure is defined to be

$$\alpha(x, y) = \frac{(n_x/n)H(x) + (n_{x'}/n)H(x') + H_2(n_x/n) - H(y)}{H_2(n_x/n)}$$

It can be seen that α takes values between 0 and 1, with lower values indicating a good split. The cost of adding a node to the solution is then

$$c(x, S) = c(x, y) = \alpha(x, y) \cdot n_x.$$

One requirement of using α-measure in the dynamic program is that we always need to select the root of T, i.e., $\text{root}(T) \in S$, in order to compute the cost of adding additional internal nodes. The requirement is not entirely unreasonable since the root directory of most sites contain a large number of files that cannot be made part of the solution on their own right and need the root to cover them.

6.4 Experiments

In this section we evaluate our algorithms on their ability to segment sites obtained from the World Wide Web. First, Section 6.4.1 describes the hand-labeled and semi-synthetic benchmark datasets we created, and Section 6.4.2 gives an overview of the experiments we run based on these benchmarks. Then in Section 6.4.3 and Section 6.4.4 we study the performance of our algorithm

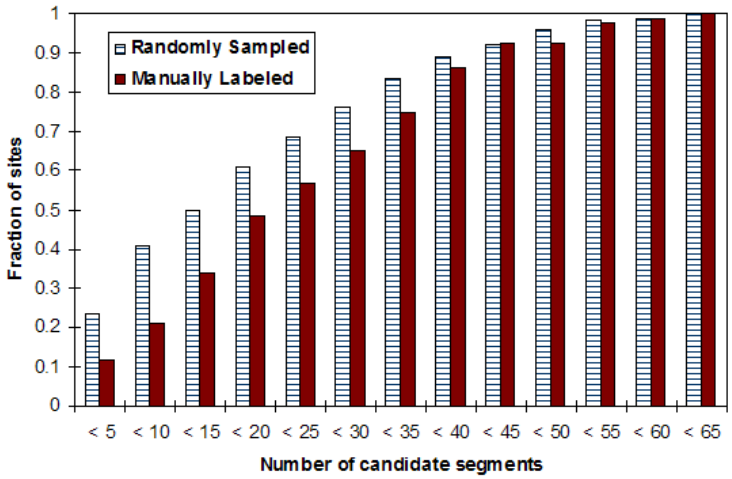


Figure 6.2: Cumulative distribution of number of candidate segments for all sites in our sample and for the sites we sampled for manual segmentation.

on both the benchmarks. In Section 6.4.5, we study the performance of the three cohesiveness cost measures with and without the node selection cost based on α-measure.

6.4.1 Website Segments: Obtaining Labeled Data

We used a page-level classifier available within Yahoo! that classifies pages into a taxonomy of 90 topics selected from the Yahoo! directory. From the site listings of these 90 topics we picked a random set of 2150 sites. For each of these sites we fetched all the URLs indexed by Yahoo!, (up to a maximum of 1000 per site) and applied the classifier in order to determine their assigned category labels. The category labels have an associated confidence measure that is ignored for the purposes of these experiments.

Hence, a site is represented by a set of URLs, each of which is labeled by one of 90 topics. In the URL tree corresponding to a particular website, a

candidate segment is defined as a node of the tree that has at least 1% of the total pages on the site under its subtree. In Figure 6.2 we plot the cumulative distribution of sites with different numbers of *candidate segments*. As we can see almost 25% of all sites have fewer than 5 nodes that can be selected as segments, and more than 50% have less than 15. We should note that in order to avoid uninteresting solutions to the HTS problem, we only consider sites that have at least two candidate nodes and more than 300 pages.

From this dataset, we generated two benchmark datasets. We refer to the first set as the *hand-labeled* website segments; this set contains 100 sites manually segmented into topically-cohesive regions. We refer to the second set as *semi-synthetic* website segments; it contains 1750 synthetically-generated websites. Each such site is created by artificially grafting together uniform regions from varying numbers of other websites, thus representing a benchmark with an unambiguous, known segmentation. We now describe the creation of each benchmark dataset in more detail.

Hand-Labeled Website Segments. We randomly sampled and manually segmented 100 websites from the Yahoo! directory. While sampling the sites around 10% were deemed to have a trivial directory structure based on URLs and were skipped. The cumulative distribution of the number of candidate segments in sites we labeled is plotted in Figure 6.2. As seen, only 10% of sites sampled for labeling have fewer than 5 candidate segments as compared to nearly 25% of the set of randomly sampled sites. Hence, while skipping websites with no directory structure we biased our sample towards sites that have larger number of candidate segments. This serves our purpose of robust evaluation of our approach as segmenting sites with very few candidate segments is a trivial and uninteresting task. Of 100 websites that were segmented

74 were segmented into two or more parts while the rest were labeled completely homogeneous (only one segment at the root directory). Among the former set of websites, the average number of segments per site was around 7, with the maximum being 18.

The criteria employed for manually selecting segments were the following. We always assumed that one segment is anchored at the root-level directory of the site; this was done to ensure complete coverage of all webpages. Subsequently, any directory that contained pages on a topic different from the aggregated topics at the root directory of the site was selected as a segment. We avoided selecting segments that were smaller than 1% of the site's size and those that were immediately enclosed within another segment on the same topic. These criteria were chosen to model the requirements mentioned in Section 6.2.1.

Semi-Synthetic Website Segments. The hand-labeled data can be used to measure our algorithm's ability to detect website segments as identified by humans. However, in order to perform more controlled evaluation of the algorithm's behavior we created a dataset with semi-synthetic segments. For this purpose we used the 26 sites that were manually labeled as homogeneous. We created a new site T' from site T_a by grafting k subtrees, from another set of sites $T_1, T_2, \ldots, T_k$, to internal directories of T_a. Since T_a and T_i are all relatively homogeneous w.r.t. topics, the new site tree T' should have $k+1$ segments (including the root directory), one from each of its constituent sites. We can now test our algorithms by measuring how many of the $k+1$ segments they discover. Certain precautions were taken while creating these hybrid sites. We only grafted subtrees that had 20 to 100 leaves under them. This ensures that the grafted subtree is larger than 1% of the hybrid site's size and

that the grafted content doesn't overwhelm the existing content of T_a. If that happens, our algorithms might create segments of subtrees from the original T_a as they will now be significantly different from the topic distribution at the root directory. We created 7 such datasets for $k = 1, \ldots, 7$, each with 250 hybrid sites.

6.4.2 Measuring Segmentation Performance

We detail our methodology and metrics. In Section 6.2.2 we hypothesized that the "correct" segmentation (including the number of segments) can be detected by finding the k that minimizes (6.1). We perform experiments on the hand-labeled as well as semi-synthetic datasets to verify whether this hypothesis holds. All experiments are run to select at most $k' = 30$ segments from the set of *candidate segments* (nodes with at least 1% of website's webpages under them). The root always selected as a segment. As the final number of segments in solutions can vary, the best way to report results is by computing the precision and recall w.r.t. to the manually labeled segments, i.e., the true solution. The *precision* of a solution is the fraction of segments in the solution that were identified by our human judges as appropriate segmentation points. Similarly, the fraction of hand-identified segments that are found by the algorithm represents the *recall* of its solution. The *f-measure*—the harmonic mean $2pr/(p + r)$ of the precision p and recall r—can be used to report the quality of a solution as a single number. For each combination of cohesiveness cost and node selection cost, by varying β we can obtain solutions with different precision and recall values. We expect that configurations with high β would be very conservative in the number of segments they find, since they bias the cost function in (6.1) towards not adding a node. Hence, these configurations

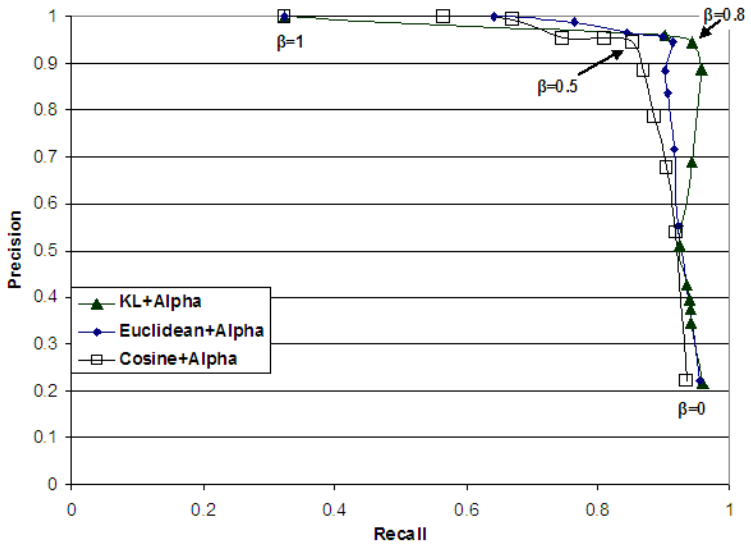


Figure 6.3: Precision–recall curves (with varying β) over the semi-synthetic benchmark. The values are averaged over all hybrid sites created (over different number of grafts settings).

should have high precision but low recall. We expect the opposite behavior for low β values, with these configurations achieving low precision and high recall scores.

6.4.3 Performance on Semi-Synthetic Benchmark

The precision–recall curves for the performance of different cohesiveness and node selection cost combinations over the semi-synthetic websites are plotted in Figure 6.3. The precision and recall values on the plot are averaged over all the 7 datasets with $k = 1, \ldots, 7$. These curves were computed by varying β from 0 to 1 in increments of 0.1. It can be seen that as we increase the value of β from 0 to 1, the curves move from the area of low precision, high recall to the area of high precision, low recall. On the plot we identify β values

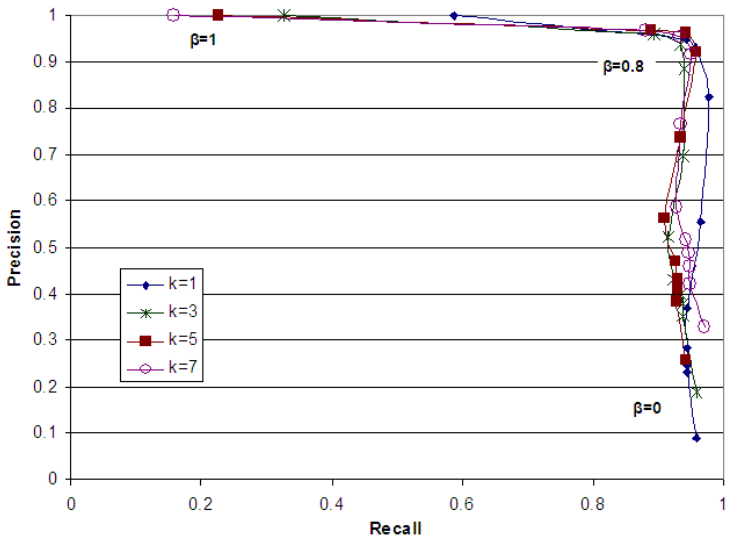


Figure 6.4: Precision–recall curves (with varying β) obtained by using KL+Alpha cost measures over the semi-synthetic benchmark. Different curves correspond to different number of grafts.

that provide good trade-off for the three combinations of cost measures. For KL+Alpha, $\beta = 0.8$ results in a precision/recall value of 0.94/0.94, while for Euclidean+Alpha and Cosine+Alpha the values at $\beta = 0.5$ are 0.94/0.91 and 0.94/0.85 respectively. This shows that in the case semi-synthetic websites, all three cost combinations are able to find the correct number of segments and their locations.

Figure 6.4 plots the precision–recall curves of the KL+Alpha algorithm on semi-synthetic sites with varying number of grafts separately. Here we plot the data for $k = 1, 3, 5, 7$. We can see that the behavior of all the curves is similar, with best precision–recall trade-off at $\beta = 0.8$. The only difference between the four curves is at $\beta = 0, 1$. This is because when $\beta = 0$, the algorithm adds a large number of segments to the solution and hence the precision suffers for

all curves. But number of true segments is different for each curve causing a different lowest precision value. Similarly, the lowest value of recall attained depends on the number of true segments and hence this value is lower for $k = 7$ than for $k = 1$. Finally, the best precision–recall values obtained are around 0.94/0.94 for all curves.

6.4.4 Performance on Hand-Labeled Benchmark

We consider the more difficult task of segmenting actual websites obtained from the World Wide Web. Figure 6.5 plots precision–recall curves for the performance of different cohesiveness and node selection cost combinations over the hand-labeled dataset. These curves were computed by varying β from 0 to 1 as in Section 6.4.3. On the plot we identify β values that provide reasonable trade-off of precision and recall for the three combinations of cost measures. For the KL+Alpha combination, $\beta = 0.8$ results in a precision–recall value of 0.79/0.62, while for Euclidean+Alpha and Cosine+Alpha the values at $\beta = 0.3$ are 0.76/0.69 and 0.8/0.67 respectively. The curves in Figure 6.5 show that for a robust set of values of β our algorithm produces very good segmentations of websites.

Now that we have seen that the algorithm finds more or less the correct segments, lets take a closer look at how well the algorithm performs in estimating the "correct" number of segments. Figure 6.6 shows the cumulative distribution of error in the number of segments detected. Here, the magnitude of error is the absolute difference in the number of segments found by our algorithm and the manual labeling. As seen, our algorithm finds the correct number of segments in nearly 40% of the cases and for more than 70% of cases the number of segments found is within ± 2 of the number of manually labeled

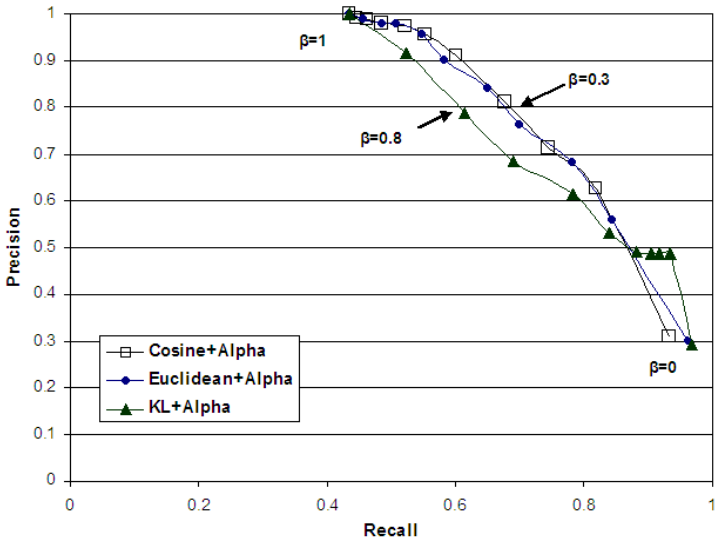


Figure 6.5: Precision–recall curve (with varying β) over the hand-labeled websites.

segments. Furthermore, the performance of all three cost combinations is very similar.

Comparing with Performance on Semi-synthetic Websites. The results in Figure 6.5 are similar to those for semi-synthetic benchmarks (Figure 6.3) in that the curves move from the area of low precision, high recall to the area of high precision, low recall as we increase β. There are, however, a couple of differences; the "knee" of the curves is more prominent and precision–recall values are higher for the semi-synthetic dataset.

These differences can be explained by pointing out that the true segments are much more unambiguously defined in the case of the semi-synthetic websites than the hand-labeled ones. In other words, in the case of semi-synthetic websites the benefit of adding a graft as a separate segment is much higher than the benefit of adding a subtree of T_a (see Section 6.4.1) as a segment

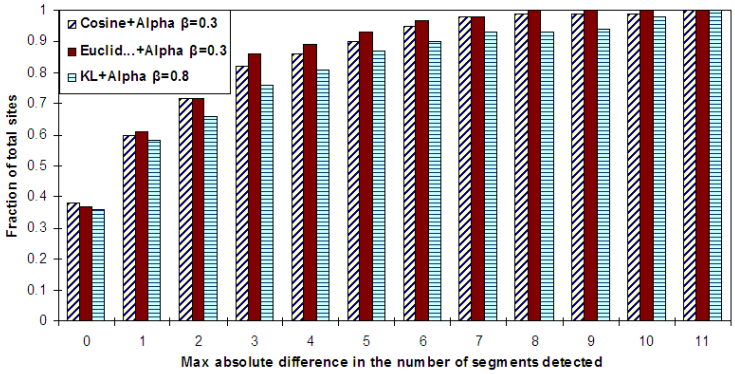


Figure 6.6: Cumulative distribution of the absolute error in the number of segments detected for the hand-labeled websites.

to the solution. This makes it easier to distinguish the true segments from T_a and hence we obtain very high precision and recall values. In the case of real websites, the benefit of adding nodes as segments are often very close to each other and in many cases the segment boundaries are fuzzy. Hence, even though we get fairly high values of precision and recall, there is a large range of β values over which the precision–recall trade-off is good.

A Relaxed Performance Criterion. The precision–recall curves plotted above take into account only segmentation points (directories), and treat even small differences in segmentation boundaries as total errors. Two segmentations with slightly different boundaries are equally acceptable if these differences do not impact too many webpages. Here we evaluate our algorithms using a measure that considers the context of segments as well as the segmentation points: the *Omega* measure, which has been previously used for comparing overlapping clusterings [CD88]. The solutions found by our algorithms can be considered overlapping clusterings, with each segment in the solution acting as a cluster and each webpage belonging to all segments on its

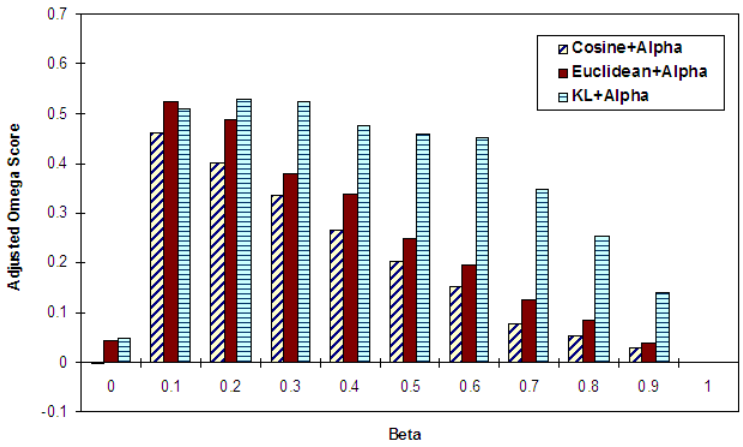


Figure 6.7: Adjusted Omega score obtained over the hand-labeled websites (with more than one true segment) for different values of β.

path to the root. In this context, the Omega measure computes the fraction of pairs of webpages that occur together under the same number of segments in both the segmentation being evaluated and the manually created segmentation. In Figure 6.7, we plot the Omega measure adjusted so that the expected performance value of a random segmentation is zero. Hence, a value of 0.5 can be interpreted as meaning that the segmentation under evaluation shows a 50% agreement with the manual segmentation, over and above any agreement that can be expected due to chance. The results in this plot are similar to the results in Figure 6.5, though with the higher recall (low β) region translating to slightly higher omega scores. The interesting point to note is that while the best performance of all cost measure combinations is similar, the KL+Alpha combination has the desirable property of giving good results over a much larger range of β than the other two cost measures.

Performance Variation with Number of Segments. Here we want to evaluate our algorithm's performance on tasks of varying difficulty. In general,

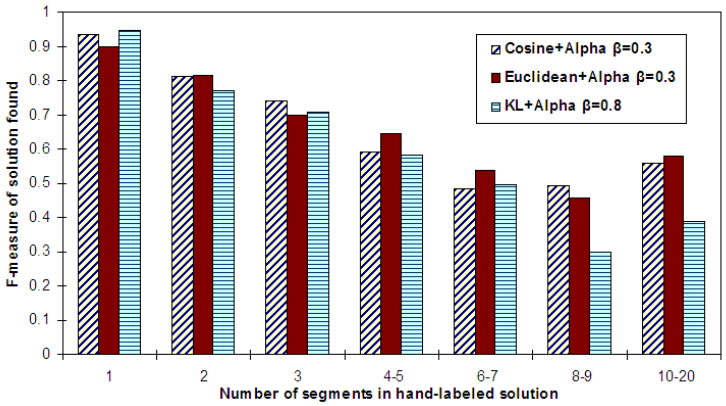


Figure 6.8: The averaged f-measure of segmentation found by the algorithm for websites with different number of segments in the labeled solution.

it is easier for the algorithm to find all the labeled segments in a site-tree if the number of segments is small. A partial reason is that the variance in the manual segmentation of site-trees increases as the number of prospective segments increases. Relatively cohesive sites with few directories of topically different content are easy for a human (and our algorithms) to segment. In Figure 6.8 we plot the f-measure of the segmentation found by our algorithm for websites in the hand-labeled set with different number of segments. From the plot we see that as the number of segments in the true solution increases, the f-measure drops to around 0.6 for both Euclidean and Cosine cohesiveness cost measures. The performance of the KL+Alpha combination, however, decreases significantly as the number of segments in the solution increases.

6.4.5 Exploring the Role of α-Measure

The α-measure acts as a regularization term in our objective function and is necessary to discover the correct number of segments. In this section we want to evaluate whether the α-measure also plays a role in the selection of

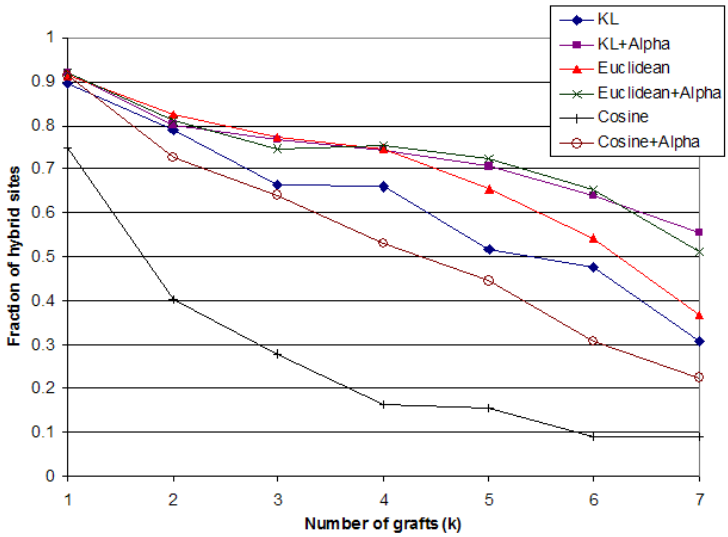


Figure 6.9: The fraction of runs in which all grafts in the hybrid tree were found vs number of grafts.

good segments, or at least in avoiding bad candidates. For this experiment we use our algorithm to segment the website trees in the semi-synthetic dataset into a *specified* number of segments. The intuition behind these experiments is that since the number of segments is fixed, the α-measure will only be able to affect the specific segments selected for the solution and not how many are selected. This will give us a way to compare solutions obtained with and without the use of α-measure to determine its impact. The reason we use the semi-synthetic dataset is because unlike the hand-labeled dataset, here the number of true segments is well-defined.

As stated in Section 6.3.1, our algorithm can be modified to work when the number of segments is fixed a priori. We used this modified algorithm to segment each tree into $k' = k + 1$ segments (number of grafts plus the root). The β value used for KL+Alpha combination was 0.8, while for Euclidean+Alpha and Cosine+Alpha it was 0.3. To run our algorithm without the α-measure,

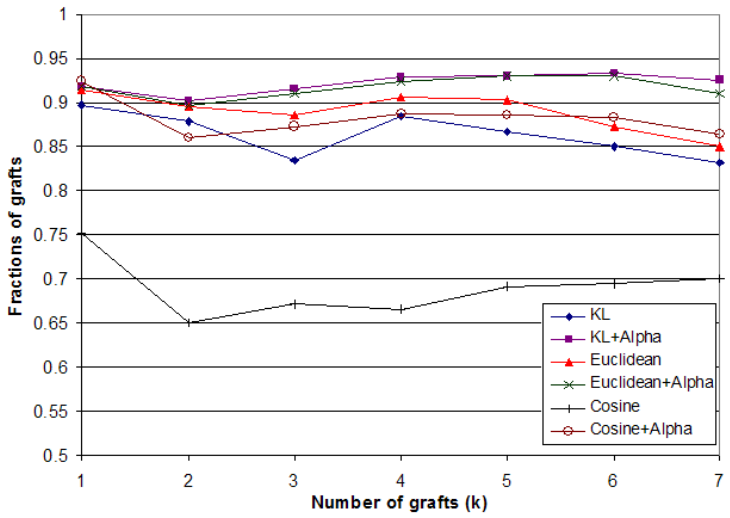


Figure 6.10: The recall of grafts in the hybrid tree vs number of grafts.

β was set to 0. Results for each different value of k are summarized in Figures 6.9 and 6.10. Each point in the plot is averaged over 250 websites.

Figure 6.9 plots the average fraction of sites (out of 250) for which all the graft points in the hybrid tree were detected by the algorithm as a function of the number of grafts (k). As we increase the number of grafts, the difficulty of identifying all the grafts increases and the fraction of sites perfectly segmented decreases. All cost measure combinations other than Cosine perform almost identically for $k = 1$, but as we increase k, their performance numbers diverge significantly. In all cases, techniques that use the α-measure perform better than their counterparts that don't. Moreover, as the difficulty of the task increases, the decrease in accuracy of techniques using α-measure is gentler.

Figure 6.10 plots the fraction of total grafts that were discovered (recall) by the algorithm for each value of k. The root segment of the hybrid tree, which is always detected, is not considered a graft and hence not counted in the

fraction of grafts detected. As we can see, in spite of the fact that performances in Figure 6.9 fall drastically as k is increased, the values in Figure 6.10 stay relatively the same. This shows that even though the algorithms aren't able to segment the entire hybrid tree perfectly as the problem becomes harder, they do discover most of the segments. As in the earlier experiments, techniques that employ the α-measure perform better than their counterparts that do not. These two experiments show that the α-measure is useful not just in regulating the size of the solution but also in identifying the "correct" segments when the solution size is specified.

6.5 Conclusions

In this chapter, we considered the problem of identifying and segmenting topically cohesive regions in the URL tree of a large website. We developed a general framework to use various cost measures for describing the quality of a segmentation; we also provided an efficient algorithm to find the best segmentation in this framework. Our experiments on hand-labeled and semi-synthetic benchmarks confirm the soundness of our framework and suggest that a judicious choice of cost measures can improve the precision/recall significantly.

Chapter 7

Conclusions and Future Work

In this chapter we present the conclusions from our work in this thesis and describe some potential directions for future research.

7.1 Conclusions

In this thesis we studied various scenarios and proposed several novel approaches where the knowledge encoded in hierarchical relationships between entities is exploited to improve the efficacy of data mining tasks like automated classification. In Chapter 3, we studied the induction of hierarchical relationships among classes in a flat taxonomy and proposed a completely automated approach to find them. Using several real-world datasets, we showed that our approach constructed more "natural" taxonomies than existing techniques, and that well structured hierarchies of classes can give significant gains in the accuracy of classifiers learned over them.

In Chapter 4, we presented approaches to further improve classification accuracy in taxonomies by post-processing classifier outputs to enforce constraints obtained from hierarchical links between classes. We studied several different scenarios with varying characteristics of taxonomies and classifier construction methods. In each scenario we showed how hierarchical relationships among classes can be translated into constraints among the corresponding classifier outputs. We formulated the problem of enforcing these constraints as regularized isotonic/unimodal tree regression problems and gave exact algorithms to solve them. Finally, using real-world datasets we showed that our smoothing approach corrects certain errors introduced by classifiers, thereby resulting in an increase in accuracy of classification.

In Chapter 5, we investigated the problem of webpage template detection and showed through large scale studies that the problem is significant and growing. We then proposed multiple functions that assigned "templateness" scores to each internal DOM node of a webpage. We argued that these templateness scores should satisfy a monotonicity property and formulated the problem of enforcing this property as step regularized isotonic regression on trees. We gave an efficient algorithm to solve this problem exactly, and then showed that smoothing the "templateness" scores over the DOM-tree corrects errors made by the scoring functions and also gives us a sectioning of the webpage. We also demonstrated that removing templates found by our approach has a positive effect on standard webmining tasks like duplicate detection and webpage classification.

In Chapter 6, we studied the task of identifying topically cohesive parts of a website. We formulated this task as the problem of finding segments nodes in the URL directory hierarchy such that each segment is itself pure in

topic as well as topically distinct from other identified segments. We defined purity and difference of topic using various cost measures and gave an efficient algorithm to find the optimal segmentation in terms of the resulting objective functions. Finally, we showed that with a judicious choice of cost measures our approach can closely mimic the performance of humans at the task of topical segmentation of websites.

7.2 Future Work

The work in this thesis can be extended along many dimensions. Possible directions for future research on the central idea of exploiting hierarchical relationships among entities for better classification is presented in Section 7.2.1. Other potential research involves the application settings that we explore in the thesis: template detection and website segmentation. These are addressed in Section 7.2.2.

7.2.1 Algorithmic Aspects

In Chapter 3, we presented an approach to automated construction of n-ary tree based taxonomies. The key idea in the work was a constraint that related the distances between sibling and parent nodes in valid taxonomies. The constraint, which was used in our approach to construct the top-down split of classes at each level, states that in a valid split the children nodes should be further from each other than they are to the parent node. In future work, this constraint can be generalized by requiring that the distance amongst the children be *at least a fraction* of the distance of the children from the parent. This fraction can then be adjusted to achieve flatter or deeper taxonomies. The final goal might be to learn this fraction parameter from labeled data, possibly

so as to maximize the accuracy of classifiers learned over the taxonomy. As the constraint controls the number of nodes in the taxonomy at each level, another direction of future work is to investigate whether the constraint boils down to a BIC like model selection criterion [Sch78].

Our work on smoothing the outputs of classifiers over a hierarchy (both of classes and objects) can be extended to take into account different domains as well as taxonomy structures, and the novel constraints that result from them. For instance, many taxonomies are naturally expressed as directed acyclic graphs (DAGs) as opposed to trees. Examples of these are the Yahoo! Web directory [Yah] and the Gene Ontology project [1]. It would be interesting to see what types of constraints are generated on outputs of classifiers when the smoothing has to be performed over the nodes of a DAG. Moreover, our exact algorithms from Chapters 4 and 5 are designed to optimize over tree structures and will need to be extended to work with DAGs.

Our work in this thesis has focused on exploiting hierarchical relationships between classes of a taxonomy in Chapter 4 and between parts of composite objects in Chapter 5. However, we have not investigated the scenario where both the classes as well as the objects have hierarchical relationships amongst themselves simultaneously. This scenario is common enough; in fact we can construct it using the applications we studied in this thesis: we want to classify parts of webpages (which are represented as nodes of a DOM-tree) into a hierarchical taxonomy of topics. Here the membership of an object into a class will depend upon its membership in related classes as well as the membership of related objects in the class. It would be interesting to show that using

[1] `http://www.geneontology.org/`

both types of hierarchical relationships simultaneously gives larger increases in classification accuracy than using only one at a time.

7.2.2 Application Oriented

The application settings, in the contexts of which we studied our approaches to exploiting hierarchical relationships for classification, also offer exciting avenues for future research. In Chapter 5 we tackled the problem of template detection. We presented global (site-level) as well as local (page-level) approaches to solve the problem and evaluated the strengths and weaknesses of both. Site-level approaches, while requiring no training data and achieving high precision, often miss certain types of templates (achieving low recall) and require a lot of computational resources to implement in a search engine pipeline. Page-level approaches on the other hand, improve upon site-level techniques by learning a more general model of templates and being extremely efficient in resource usage, but are often not as accurate on repeated content type templates. What is needed is a hybrid approach that uses the strengths of both approaches. Note that our page-level system in Chapter 5 trains automatically by using the template data generated by a site-level algorithm, thus achieving some synergy. However, we believe that a tighter coupling of two paradigms could be used to achieve more gains in template detection performance.

Other interesting future work involves extending our work on website segmentation in Chapter 6. In this work we presented an approach to segment the URL directory structure within websites. However, our informal experiments show that around 10-15% of websites are completely dynamically generated and do not have any URL structure. We can still construct a URL struc-

ture between webpages from the dynamic URLs, such as `www.mysite.com/showpage.php?id=123&...`, using recent work in [BYKS07]. A much more elegant approach, though, would be to extend our framework and algorithms to deal with general graphs (not just trees), especially those induced by hyperlinks. Hyperlinking is a fundamental property of the World Wide Web, and all websites support it. Hence, the new framework and algorithms would be applicable to a much larger set of websites than our current approach.

Bibliography

[AAS03] Giordano Adami, Paolo Avesani, and Diego Sona. Bootstrapping for hierarchical document classification. In *Proc. of International Conference on Information and Knowledge Management*, pages 295–302. ACM Press, 2003.

[ABKS99] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 49–60. ACM Press, 1999.

[ABP06] Aris Anagnostopoulos, Andrei Z. Broder, and Kunal Punera. Effective and efficient classification on a search-engine model. In *Proc. of International Conference on Information and Knowledge Management*, pages 208–217. ACM Press, 2006.

[ACA06] Alekh Agarwal, Soumen Chakrabarti, and Sunny Aggarwal. Learning to rank networked entities. In *Proc. of the ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 14–23. ACM Press, 2006.

[AHKW06] S. Angelov, B. Harb, S. Kannan, and L.-S. Wang. Weighted isotonic regression under the L1 norm. In *Proc. of Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 783–791, 2006.

[AMBCea00] Blake J Ashburner M Ball C and et al. Gene ontology: tool for

the unification of biology. the gene ontology consortium. *Nature Genetics*, 25(1):25–29, May 2000.

[ASS00] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *Proc. of International Conference on Machine Learning*, pages 9–16. Morgan Kaufmann Publishers Inc., 2000.

[Aum03] David J. Aumueller. A tool for gathering, analysing, exporting, and visualizing the structure of a website. Master's thesis, University of Leeds, Institute of Communications Studies, 2003.

[Bal06] Shumeet Baluja. Browsing on small screens: Recasting web-page segmentation into an efficient machine learning framework. In *Proc. of International Conference on World Wide Web*, pages 33–42, 2006.

[BBDH00] Krishna Bharat, Andrei Broder, Jeffrey Dean, and Monika R. Henzinger. A comparison of techniques to find mirrored hosts on the WWW. *Journal of the American Society for Information Science*, 51(12):1114–1122, 2000.

[BBM02] Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. Semi-supervised clustering by seeding. In *Proc. of International Conference on Machine Learning*, pages 27–34. Morgan Kaufmann Publishers Inc., 2002.

[BBM04] Sugato Basu, Mikhail Bilenko, and Raymond J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proc.*

of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 59–68. ACM Press, 2004.

[BDGS05] Arindam Banerjee, Inderjit S. Dhillon, Joydeep Ghosh, and Suvrit Sra. Clustering on the unit hypersphere using von Mises–Fisher distributions. *Journal of Machine Learning Research*, 6:1345–1382, 2005.

[BGJT04] David Blei, Thomas Griffiths, Michael Jordan, and Joshua Tenenbaum. Hierarchical topic models and the nested Chinese restaurant process. In *Proc. of Advances in Neural Information Processing Systems*. MIT Press, 2004.

[BGMZ97] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997.

[BJ03] D. Blei and M. Jordan. Modeling annotated data. In *Proc. of Annual International ACM Conference on Research and Development in Information Retrieval*, pages 127–134, 2003.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[BMDG05] Arindam Banerjee, Srujana Merugu, Inderjit S. Dhillon, and Joydeep Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005.

[BMI06] Victor Boyarshinov and Malik Magdon-Ismail. Linear time isotonic and unimodal regression in the L1 and L∞ norms. *Journal of Discrete Algorithms*, 4(4):676–691, 2006.

[BVZ01] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[BYKS07] Ziv Bar-Yossef, Idit Keidar, and Uri Schonfeld. Do not crawl in the dust: different urls with similar text. In *Proc. of International Conference on World Wide Web*, pages 111–120. ACM Press, 2007.

[BYR02] Ziv Bar-Yossef and Sridhar Rajagopalan. Template detection via data mining and its applications. In *Proc. of International Conference on World Wide Web*, pages 580–591. ACM Press, 2002.

[CD88] Linda M. Collins and Clyde W. Dent. Omega: A general formulation of the rand index of cluster recovery suitable for non-disjoint solutions. *Multivariate Behavioral Research*, 23(2):231–242, 1988.

[CDAR98] Soumen Chakrabarti, Byron Dom, Rakesh Agrawal, and Prabhakar Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *VLDB Journal: Very Large Data Bases*, 7(3):163–178, 1998.

[CDI98] Soumen Chakrabarti, Byron Dom, and Piotr Indyk. Enhanced hypertext categorization using hyperlinks. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 307–318. ACM Press, 1998.

[CH04] Lijuan Cai and Thomas Hofmann. Hierarchical document categorization with support vector machines. In *Proc. of International Conference on Information and Knowledge Management*, pages 78–87. ACM Press, 2004.

[Cha] Soumen Chakrabarti. HyParSuite, `http://www.cse.iitb.ac.in/~soumen/download/`.

[Cha04] Deepayan Chakrabarti. Autopart: parameter-free graph partitioning and outlier detection. In *Proc. of European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 112–124. Springer-Verlag New York, Inc., 2004.

[CHR01] N. Craswell, D. Hawking, and S. Roberston. Effective site finding using link anchor information. In *Proc. of Annual International ACM Conference on Research and Development in Information Retrieval*, pages 250–257, 2001.

[CJT01] Soumen Chakrabarti, Mukul Joshi, and Vivek Tawde. Enhanced topic distillation using text, markup tags, and hyperlinks. In *Proc. of Annual International ACM Conference on Research and Development in Information Retrieval*, pages 208–216. ACM Press, 2001.

[CKKS02] W. F. Cody, J. T. Kreulen, V. Krishna, and W. S. Spangler. The integration of business intelligence and knowledge management. *IBM Systems Journal*, 41(4), 2002.

[CKP07] Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. Page-level template detection via isotonic smoothing. In *Proc. of In-*

ternational Conference on World Wide Web, pages 61–70. ACM Press, 2007.

[CKT92] Douglass R. Cutting, David R. Karger, and Jan O. Peders W. Tukey. Scatter/Gather: a cluster-based approach to browsing large llections. In *Proc. of Annual International ACM Conference on Research and Development in Information Retrieval*, pages 318–329. ACM Press, 1992.

[CRW01] A. S. Chulef, S. J. Read, and D. A. Walsh. A hierarchical taxonomy of human goals. *Motivation and Emotion*, 25(3):191–232, September 2001.

[CS02] Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2-3):201–233, 2002.

[CSGM00] Junghoo Cho, Narayanan Shivakumar, and Hector Garcia-Molina. Finding replicated web collections. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 355–366, 2000.

[CSZ06] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien, editors. *Semi-Supervised Learning*. MIT Press, 2006.

[CXMZ05] Yu Chen, Xing Xie, Wei-Ying Ma, and Hong-Jiang Zhang. Adapting web pages for small-screen devices. *Internet Computing*, 9(1):50–56, 2005.

[Dav00] Brian Davison. Recognizing nepotistic links on the web. In *Artificial Intelligence for Web Search*, pages 23–28. AAAI Press, 2000.

[DB95] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[DC00] Susan Dumais and Hao Chen. Hierarchical classification of web content. In *Proc. of Annual International ACM Conference on Research and Development in Information Retrieval*, pages 256–263. ACM Press, 2000.

[DFG01] I. Dhillon, J. Fan, and Y. Guan. Efficient clustering of very large document collections. In R. Grossman, G. Kamath, and R. Naburu, editors, *Data Mining for Scientific and Engineering Applications*. Kluwer Academic Publishers, 2001.

[DG04a] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proc. of Conference on Symposium on Opearting Systems Design & Implementation*, pages 10–23. USENIX Association, 2004.

[DG04b] Ludovic Denoyer and Patrick Gallinari. Bayesian network model for semi-structured document classification. *Information Processing and Management*, 40(5):807–827, 2004.

[DGMS01] M. Diligenti, M. Gori, M. Maggini, and F. Scarselli. Classification of HTML documents by hidden tree-Markov models. In

6th International Conference on Document Analysis and Recognition, pages 849–853, 2001.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[DLR77] Arthur Dempster, Nan Laird, and Donald Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[DMDH02] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proc. of International Conference on World Wide Web*, pages 662–673, 2002.

[DMK02] Inderjit S. Dhillon, Subramanyam Mallela, and Rahul Kumar. Enhanced word clustering for hierarchical text classification. In *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 191–200. ACM Press, 2002.

[DMK03] Inderjit S. Dhillon, Subramanyam Mallela, and Rahul Kumar. A divisive Information-Theoretic feature clustering algorithm for text classification. *Journal of Machine Learning Research*, 3:1265–1287, 2003.

[DMO] DMOZ. Open directory project, `http://www.dmoz.org`.

[DMPG05] Sandip Debnath, Prasenjit Mitra, Nirmal Pal, and C. Lee Giles. Automatic identification of informative sections of web pages. *IEEE Transactions on Knowledge and Data Engineering*, 17(9):1233–1246, 2005.

[DMS99] Inderjit Dhillon, Dharmendra Modha, and W. Scott Spangler. Class visualization of high-dimensional data with applications. Technical report, IBM Almaden Research Center, San Jose, CA 95120, 1999.

[EKS02] Martin Ester, Hans-Peter Kriegel, and Matthias Schubert. Web site mining: A new way to spot competitors, customers and suppliers in the world wide web. In *Proc. of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 249–258, 2002.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.

[Fal96] Andrew Fall. *Reasoning with taxonomies.* PhD thesis, Simon Fraser University, 1996. Adviser-Veronica Dahl.

[Faw03] Tom Fawcett. ROC graphs: Notes and practical considerations for data mining researchers. Technical Report HPL-2003-4, Hewlett Packard Laboratories, January 17 2003.

[Fel68] William Feller. *An Introduction to Probability Theory and Its Applications, Volume 1.* John Wiley & Sons, 1968.

[FFR97] Adam Farquhar, Richard Fikes, and James Rice. The ontolingua server: a tool for collaborative ontology construction. *Inter-*

national Journal of Human-Computer Studies, 46(6):707–727, 1997.

[FGK+05] Ronald Fagin, Ramanathan Guha, Ravi Kumar, Jasmine Novak, D. Sivakumar, and Andrew Tomkins. Multi-structural databases. In *Proc. of ACM Symposium on Principles of Database Systems*, pages 184–195, 2005.

[FKK+05] Ronald Fagin, Phokion Kolaitis, Ravi Kumar, Jasmine Novak, D. Sivakumar, and Andrew Tomkins. Efficient implementation of large-scale multi-structural databases. In *Proc. of International Conference on Very Large Databases*, pages 958–969, 2005.

[FMSB+06] R. D. Finn, J. Mistry, B. Schuster-Bckler, S. Griffiths-Jones, V. Hollich, T. Lassmann, S. Moxon, M. Marshall, A. Khanna, R. Durbin, S. R. Eddy, E. L. Sonnhammer, and A. Bateman. Pfam: clans, web tools and services. *Nucleic Acids Research*, 34(Database issue):247–251, January 2006.

[FST98] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.

[GGPC02] Eric Gaussier, Cyril Goutte, Ashok Popat, and Francine Chen. A hierarchical model for clustering and categorizing documents. In *Proc. of BCS-IRSG European Colloquium on IR Research*, pages 229–247, 2002.

[Gib04] David Gibson. Surfing the web by site. In *Proc. of International Conference on World Wide Web*, pages 496–497, 2004.

[GPT05] David Gibson, Kunal Punera, and Andrew Tomkins. The volume and evolution of web page templates. In *Proc. of International Conference on World Wide Web*, pages 830–839, 2005.

[GR04] Jacob Goldberger and Sam Roweis. Hierarchical clustering of a mixture model. In *Proc. of Advances in Neural Information Processing Systems*, 2004.

[GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: an efficient clustering algorithm for large databases. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 73–84, 1998.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[GTC05] Stephen C. Gates, Wilfried Teiken, and Keh-Shin F. Cheng. Taxonomies by the numbers: building high-performance taxonomies. In *Proc. of International Conference on Information and Knowledge Management*, pages 568–577. ACM Press, 2005.

[HA85] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 1985.

[Hay99] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[HCC04] Chien-Chung Huang, Shui-Lung Chuang, and Lee-Feng Chien. Liveclassifier: creating hierarchical text classifiers through web cor pora. In *Proc. of International Conference on World Wide Web*, pages 184–192. ACM Press, 2004.

[HCCG05] Jisoo Ham, Yangchi Chen, Melba M. Crawford, and Joydeep Ghosh. Investigation of the random forest framework for classification of hyperspectral data. *IEEE Transactions on Geoscience and Remote Sensing*, 43(3):492–501, 2005.

[Hea06] Marti A. Hearst. Clustering versus faceted categories for information exploration. *Communications of the ACM*, 49(4):59–61, 2006.

[HHW+04] Arnaud Le Hors, Philippe Le Hgaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document object model (dom) level 3 core specification. Technical report, W3C, `http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/DOM3-Core.html`, April 2004.

[HKP95] Marti A. Hearst, David R. Karger, and Jan O. Pedersen. Scatter/Gather as a tool for the navigation of retrieval results. In *Working Notes AAAI Fall Symp. AI Applications in Knowledge Navigation*, 1995.

[HKZ98] Jing Huang, S. Ravi Kumar, and Ramin Zabih. An automatic hierarchical image classification scheme. In *ACM Multimedia*, pages 219–228, 1998.

[Hof99] Thomas Hofmann. The cluster-abstraction model: Unsupervised learning of topic hierarchies from text data. In *Proc. of International Joint Conference on Artificial Intelligence*, pages 682–687. Morgan Kaufmann Publishers Inc., 1999.

[Hsu82] Wen-Lian Hsu. The distance-domination numbers of trees. *Operations Research Letters*, 1:96–100, 1982.

[HWL06] Tzu-Kuo Huang, Ruby C. Weng, and Chih-Jen Lin. Generalized Bradley-Terry models and multi-class probability estimates. *Journal of Machine Learning Research*, 7:85–115, 2006.

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for clustering data*. Prentice-Hall Inc., 1988.

[Joa99] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *Proc. of International Conference on Machine Learning*, pages 200–209. Morgan Kaufmann Publishers Inc., 1999.

[Kah] Brewster Kahle. The internet archive, `http://www.archive.org`.

[KaT05] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 2005.

[KCLH02] Hung-Yu Kao, Ming-Syan Chen, Shian-Hua Lin, and Jan-Ming Ho. Entropy-based link analysis for mining web informative structures. In *Proc. of International Conference on Information and Knowledge Management*, pages 574–581. ACM Press, 2002.

[KGC01] Shailesh Kumar, Joydeep Ghosh, and Melba M. Crawford. Best-bases feature extraction algorithms for classification of hyperspectral data. *IEEE Transactions Geoscience and Remote Sensing*, 39(7):1368–1379, 2001.

[KGC02] Shailesh. Kumar, Joydeep Ghosh, and Melba M. Crawford. Hierarchical fusion of multiple classifiers for hyperspectral data analysis. *Pattern Analysis and Applications, special Issue on Fusion of Multiple Classifiers*, 5(2):210–220, 2002.

[KH79] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems, part II: p-medians. *SIAM Journal on Applied Mathematics*, 37:539–560, 1979.

[KHC05] Hung-Yu Kao, Jan-Ming Ho, and Ming-Syan Chen. WISDOM: Web intrapage informative structure mining based on document object model. *IEEE Transactions on Knowledge and Data Engineering*, 17(5):614–627, 2005.

[KHK99] George Karypis, Eui-Hong (Sam) Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[KL51] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[Kom05] Sam H. Kome. Hierarchical subject relationships in folksonomies. Master's thesis, University of North Carolina at Chapel Hill, Nov 2005.

[KPT06] Ravi Kumar, Kunal Punera, and Andrew Tomkins. Hierarchical topic segmentation of websites. In *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 257–266, 2006.

[KS97] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In *Proc. of International Conference on Machine Learning*, pages 170–178. Morgan Kaufmann Publishers Inc., 1997.

[KS04] Hans-Peter Kriegel and Matthias Schubert. Classification of websites as sets of feature vectors. In *Proc. of IASTED International Conference on Databases and Applications*, pages 127–132, 2004.

[KSGM03] S. D. Kamvar, M. T. Scholsser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *Proc. of International Conference on World Wide Web*, pages 640–651, 2003.

[Kus99] Nicholas Kushmerick. Learning to remove internet advertisement. In *Proc. of the Third International Conference on Autonomous Agents (Agents'99)*, pages 175–181. ACM Press, 1999.

[Lan95] Ken Lang. Newsweeder: Learning to filter netnews. In *Proc. of International Conference on Machine Learning*, pages 331–339, 1995.

[Lin91] J. Lin. Divergence measures based on the shannon entropy.

IEEE Transactions on Information Theory, 37(1):145–151, January 1991.

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[MAea07] Nicola J. Mulder, Rolf Apweiler, and Teresa K. Attwood et al. New developments in the interpro database. *Nucleic Acids Research*, 35(Database Issue):224–228, 2007.

[Mat04] Adam Mathes. Folksonomies - cooperative classification and communication through shared metadata. In *Computer Mediated Communication - LIS590CMC*. University of Illinois Urbana Champaign, Dec 2004.

[MC86] Glenn Milligan and Martha Cooper. A study of the comparability of external criteria for hierarchical cluster analysis. *Multivariate Behavioral Research*, 21(4):441–458, 1986.

[Me04] D. R. Maddison and K.-S. Schulz (ed.). The tree of life web project: http://tolweb.org, 2004.

[MICAM02] Malik Magdon-Ismail, Hung-Ching Chen, and Yaser S. Abu-Mostafa. The multilevel classification problem and a monotonicity hint. In *Proc. of International Conference on Intelligent Data Engineering and Automated Learning*, pages 410–415. Springer-Verlag, 2002.

[Mit97] Thomas Mitchell. *Machine Learning*. McGraw Hill, 1997.

[MJDP+00] Tony Morton-Jones, Peter Diggle, Louise Parker, Heather O. Dickinson, and Keith Binks. Additive isotonic regression models in epidemiology. *Statistics in Medicine*, 19(6):849–859, 2000.

[MN98] Andrew McCallum and Kamal Nigam. A comparison of event models for Naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[Mor02] Joseph T. Morgan. *Adaptive Hierarchical Classifier with Limited Training Data*. PhD thesis, University of Texas at Austin, 2002.

[MRMN98] Andrew McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proc. of International Conference on Machine Learning*, pages 359–367. Morgan Kaufmann Publishers Inc., 1998.

[NCO04] Alexandros Ntoulas, Junghoo Cho, and Christopher Olston. What's new on the web?: the evolution of the web from a search engine perspective. In *Proc. of International Conference on World Wide Web*, pages 1–12. ACM Press, 2004.

[NJ02] Andrew N. Ng and Michael Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Proc. of Advances in Neural Information Processing Systems*, 2002.

[NM00] Natalya Fridman Noy and Mark A. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In

Proc. of Conference on Innovative Applications of Artificial Intelligence, pages 450–455. AAAI Press / The MIT Press, 2000.

[NMTM00] Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using em. *Machine Learning*, 39(2-3):103–134, 2000.

[PC06] Navneet Panda and Edward Y. Chang. Kdx: An indexer for support vector machines. *IEEE Transactions on Knowledge and Data Engineering*, 18(6):748–763, 2006.

[PDG02] B. Piwowarski, L. Denoyer, and P. Gallinari. Un modèle pour la recherche d'information sur des documents structurés. In *6th Journées internationales d'Analyse statistique des Données Textuelles*, 2002.

[PG05] Kunal Punera and Joydeep Ghosh. Clump: A scalable and robust framework for structure discovery. In *Proc. of IEEE International Conference on Data Mining*, pages 757–760. IEEE Computer Society, 2005.

[PG07] Kunal Punera and Joydeep Ghosh. *Advances in Fuzzy Clustering and its Applications*, chapter Soft Cluster Ensembles, pages 69–90. John Wiley & Sons, Ltd., 2007.

[Pie00] John Pierre. Practical issues for automated categorization of web sites. In *ECDL 2000 Workshop on Semantic Web*, 2000.

[PLP+04] Michael Pelikan, James Leous, Richard Pearce, Margaret E. Smith, and Russell Vaught. Searching for the needle in the

haystack: taxonomies, tags and targets. In *Proc. of Annual ACM SIGUCCS Conference on User services*, pages 256–261. ACM Press, 2004.

[PR03] Judea Pearl and Stuart Russell. Bayesian networks. In Michael A. Arbib, editor, *he Handbook of Brain Theory and Neural Networks*. MIT Press, 2nd edition, 2003.

[PRG05] Kunal Punera, Suju Rajan, and Joydeep Ghosh. Automatically learning document taxonomies for hierarchical classification. In *Proc. of International Conference on World Wide Web (Special interest tracks and posters)*, pages 1010–1011. ACM Press, 2005.

[PRG06] Kunal Punera, Suju Rajan, and Joydeep Ghosh. Automatic construction of n-ary tree based taxonomies. In *Proc. of IEEE International Conference on Data Mining - Workshops*, pages 75–79. IEEE Computer Society, 2006.

[PX99] P. M. Pardalos and G. Xue. Algorithms for a class of isotonic regression problems. *Algorithmica*, 23(3):211–222, 1999.

[Qui] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Qui90] J. Ross Quinlan. Induction of decision trees. In Jude W. Shavlik and Thomas G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990. Originally in *Machine Learning* 1:81–106, 1986.

[Qui93] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[RGC05] Suju Rajan, Joydeep Ghosh, and Melba M. Crawford. Exploiting class hierarchies for knowledge transfer in hyperspectral data. In *Multiple Classifier Systems*, pages 417–427. Springer Berlin, 2005.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. pages 318–362, 1986.

[RK04] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychology Review*, 65(6):386–408, November 1958.

[Rot02] Dan Roth. Reasoning with classifiers. In *Proc. of European Conference on Principles of Data Mining and Knowledge Discovery*, pages 489–493. Springer-Verlag, 2002.

[RR01] Jason Rennie and Ryan Rifkin. Improving multiclass text classification with the support vector machine. AI Memo AIM-2001-026, Massachusetts Institute of Technology, 2001.

[RT00] F. Ricca and P. Tonella. Web site analysis: Structure and evolution. In *Proc. of IEEE International Conference on Software Maintenance*, pages 76–86, 2000.

[RtY04] Dan Roth and Wen tau Yih. A linear programming formulation for global inference in natural language tasks. In *Proc. of*

Conference on Computational Natural Language Learning, pages 1–8, 2004.

[SAM97] Joseph Sill and Yaser S. Abu-Mostafa. Monotonicity hints. In *Proc. of Advances in Neural Information Processing Systems*, volume 9, page 634. The MIT Press, 1997.

[Sch78] Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[SEKX98] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Mining Knowledge Discovery*, 2(2):169–194, 1998.

[SFC02] Rahul Shah and Martin Farach-Colton. Undiscretized dynamic programming: Faster algorithms for facility location and related problems on trees. In *Proc. of Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 108–115, 2002.

[SFT01] Noam Slonim, Nir Friedman, and Naftali Tishby. Agglomerative multivariate information bottleneck. In *Proc. of Advances in Neural Information Processing Systems*, 2001.

[SG02] Alexander Strehl and Joydeep Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, December 2002.

[SKO01] Eran Segal, Daphne Koller, and Dirk Ormoneit. Probabilistic abstraction hierarchies. In *Proc. of Advances in Neural Information Processing Systems*, 2001.

[SL03] Aixin Sun and Ee-Peng Lim. Web unit mining: finding and classifying subgraphs of web pages. In *Proc. of International Conference on Information and Knowledge Management*, pages 108–115, 2003.

[Slo03] Noam Slonim. *The Information Bottleneck: Theory and Applications*. PhD thesis, The Hebrew University, 2003.

[SLWM04] Ruihua Song, Haifeng Liu, Ji-Rong Wen, and Wei-Ying Ma. Learning block importance models for web pages. In *Proc. of International Conference on World Wide Web*, pages 203–211. ACM Press, 2004.

[SS97] Michael J. Schell and Bahadur Singh. The reduced monotonic regression method. *Journal of the American Statistical Association*, 92(437):128–135, 1997.

[ST99] Noam Slonim and Naftali Tishby. Agglomerative information bottleneck. In *Proc. of Advances in Neural Information Processing Systems*, 1999.

[Sto00] Q. Stout. Optimal algorithms for unimodal regression. *Computing Science and Statistics*, 32:348–355, 2000.

[Tam96] Arie Tamir. An $o(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19:59–64, 1996.

[TH07] Robert Tibshirani and Trevor Hastie. Margin trees for high-dimensional classification. *Journal of Machine Learning Research*, 8:637–652, 2007.

[THA99] Loren Terveen, Will Hill, and Brian Amento. Constructing, organizing, and visualizing collections of topically related web resources. *ACM Transactions on Computer-Human Interaction*, 6(1):67–94, 1999.

[THG+03] YongHong Tian, TieJun Huang, Wen Gao, Jun Cheng, and PingBo Kang. Two-phase web site classification based on hidden Markov tree models. In *IEEE/WIC International Conference on Web Intelligence*, pages 227–236, 2003.

[TJHA05] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[TPB99] Naftali Tishby, Fernando Pereira, and William Bialek. The information bottleneck method. In *Proc. of the Annual Allerton Conference on Communication, Control and Computing*, pages 368–377, 1999.

[1] E. M. Edghill (translator). *Categories*. The University of Adelaide, 2007.

[TSW03] Martin Theobald, Ralf Schenkel, and Gerhard Weikum. Exploiting structure, annotation, and ontological knowledge for auto-

matic classification of XML data. In *Proc. of 6thInternational Workshop on the Web and Databases*, pages 1–6, 2003.

[TW04] Mike Thelwall and David Wilkinson. Finding similar academic web sites with links, bibliometric couplings and colinks. *Information Processing and Management*, 40(3):515–526, 2004.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[Vap95] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[VD00a] Shivakumar Vaithyanathan and Byron Dom. Hierarchical unsupervised learning. In *Proc. of International Conference on Machine Learning*, June 2000.

[VD00b] Shivakumar Vaithyanathan and Byron Dom. Model-based hierarchical clustering. In *Proc. of Conference on Uncertainty in Artificial Intelligence*, June 2000.

[VD04] Volkan Vural and Jennifer G. Dy. A hierarchical method for multi-class support vector machines. In *Proc. of International Conference on Machine Learning*, pages 105–114. ACM Press, 2004.

[VL99] Nuno Vasconcelos and Andrew Lippman. Learning mixture hierarchies. In *Proc. of Advances in Neural Information Processing Systems*, pages 606–612. MIT Press, 1999.

[VSP+06] Karane Vieira, Altigran Silva, Nick Pinto, Edleno Moura, Joao Cavalcanti, and Juliana Freire. A fast and robust method for web page template detection and removal. In *Proc. of International Conference on Information and Knowledge Management*, pages 256–267, 2006.

[WCH87] Morton E. Winston, Roger Chaffin, and Douglas Herrmann. A taxonomy of part-whole relations. *Cognitive Science*, 11(4):417–444, 1987.

[WLW04] Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal Machine Learning Research*, 5:975–1005, 2004.

[Yah] Yahoo! Web directory `http://dir.yahoo.com`.

[YL03] Lan Yi and Bing Liu. Web page cleaning for web mining through feature weighting. In *Proc. of International Joint Conference on Artificial Intelligence*, 2003.

[YL04] Xinyi Yin and Wee Sun Lee. Using link analysis to improve layout on mobile devices. In *Proc. of International Conference on World Wide Web*, pages 338–344, 2004.

[YLL03] Lan Yi, Bing Liu, and Xiaoli Li. Eliminating noisy information in web pages for data mining. In *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 296–305. ACM Press, 2003.

[ZE02] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proc. of*

ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 694–699, 2002.

[ZK05] Ying Zhao and George Karypis. Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 10(2):141–168, 2005.

[ZLPY04] Li Zhang, ShiXia Liu, Yue Pan, and LiPing Yang. InfoAnalyzer: a computer-aided tool for building enterprise taxonomies. In *Proc. of International Conference on Information and Knowledge Management*, pages 477–483. ACM Press, 2004.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 103–114, 1996.

Vita

Kunal Punera was born on January 15 1980 in the state of Goa in India. He received the Bachelor of Engineering degree in Computer Engineering from the University of Mumbai, India, in 2001. After graduation he worked as a research assistant for a year at the Indian Institute of Technology in Bombay, India.

Kunal entered the graduate program at The University of Texas at Austin in the fall of 2002 and joined the Intelligent Data Exploration and Analysis Lab headed by Dr. Joydeep Ghosh. He was awarded the Master of Science in Engineering degree in December 2004 upon the completion of his thesis work titled "Soft Cluster Ensembles" under the supervision of Dr. Ghosh. Since that time he has been a doctoral student working on problems in data mining and information retrieval for the World Wide Web. He has also completed internships at the IBM Almaden Research Center and Yahoo! Research.

Permanent address: B-1504, Gokul Gagan,
Thakur Village, Kandivli(East),
Mumbai 400101, India

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.