Automatic Construction of N-ary Tree Based Taxonomies

Kunal Punera* Suju Rajan Joydeep Ghosh

{kunal,suju,ghosh} @ ece.utexas.edu

IDEAL-2006-06 †

Intelligent Data Exploration & Analysis Laboratory

(Web: http://www.ideal.ece.utexas.edu/)

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712
U.S.A.

September 2005

*This work was done while the author was at Yahoo! Research.

†© 2006 Kunal Punera, Suju Rajan, and Joydeep Ghosh

Abstract

Hierarchies are an intuitive and effective organization paradigm for data. Of late there has been considerable research on automatically learning hierarchical organizations of data. In this paper, we explore the problem of learning n-ary tree based hierarchies of categories with no user-defined parameters. We propose a framework that characterizes a "good" taxonomy and also provide an algorithm to find it. This algorithm works completely automatically (with no user input) and is significantly less greedy than existing algorithms in literature. We evaluate our approach on multiple real life datasets from diverse domains, such as text mining, hyper-spectral analysis, written character recognition etc. Our experimental results show that not only are n-ary trees based taxonomies more "natural", but also the output space decompositions induced by these taxonomies for many datasets yield better classification accuracies as opposed to classification on binary tree based taxonomies.

1 Introduction

Hierarchical taxonomies have become an important tool in the organization of knowledge in many domains. 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 structure. 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. Many internet directories such as Yahoo[1] and DMOZ[2] are organized as hierarchies. Taxonomies have also become integral knowledge management tools in corporate intranets [CKKS02, PLP+04].

APPLICATIONS OF TAXONOMIES.

Apart from the knowledge management applications of taxonomies mentioned above, 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. [CKPT92, 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].

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 and DMOZ taxonomies were created manually by employees and volunteers[3] respectively. This manual labeling process is, however, time-consuming, expensive, and, in the case of an changing and expanding corpus like the World Wide Web, inherently incomplete.

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.

AUTOMATIC TAXONOMY CONSTRUCTION.

In this paper we tackle the problem of arranging a given set of categories 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). There have been several proposed approaches that seek to solve parts of this problem. Kumar et al. [KGC02], Vural and Dy [VD04], and Punera et al. [PRG05] proposed 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 tree. There has also been some work on learning n-ary tree structured hierarchies [GR04, SKO01] but these

[1] www.yahoo.com

[2] www.dmoz.org

[3] www.dmoz.org/about.html

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. 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.

SUMMARY OF CONTRIBUTIONS.

Here we present some salient contributions of this paper.

- 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.
- 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.
- 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.
- Through experiments 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.

Our approach is detailed in Section 2. Then we present experimental results evaluating our approach on a diverse set of datasets in Section 4. Finally, we conclude the paper in Section 5.

2 Approach

We begin with a detailed description of the problem of automatic taxonomy construction and then propose a solution to it, which we refer to as Automatic Taxonomy Generator (ATG). Henceforth in this paper, 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.

2.1 Automatic Taxonomy Construction

We broadly define the problem of automatic taxonomy construction as finding an arrangement of classes in a hierarchy. In this paper 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 and Garden", "Clothing and 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 and Garden", "Clothing and 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.

FORMAL DEFINITION.

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 $C_w = leaf(T_w)$. 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 in this paper a collection of sets of classes such as $\{C_{v_i} : 1 \leq i \leq m\}$ is sometimes shortened to $\{C_{v_i}\}_{i=1}^{m}$.

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 this paper.

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 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 $\pi_i = \pi(C_i)$

$$\begin{aligned} JS_\pi(\{C_1, C_2\}) =& \pi_1 KL(p_{C_1}, \pi_1 p_{C_1} \\ &+ \pi_2 p_{C_2}) + \pi_2 KL(p_{C_2}, \pi_1 p_{C_1} + \pi_2 p_{C_2}) \end{aligned}$$

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

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

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 paper we use $JS(\{c_j : c_j \in C_{v_i}\})$ to refer to the JS divergence between the set of distributions p_{c_j}; and $JS(\{C_{v_i}, C_{v_j}\})$ 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

$$\sum_{i=1}^{m} \pi(C_{v_i}) JS_{\pi'}\left(\{c_j : c_j \in C_{v_i}\}\right) \tag{2}$$

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 $\forall i, j \neq i$

$$\begin{aligned} JS_{\pi''}\left(\{C_{v_i}, C_{v_j}\}\right) > min\{&JS_{\pi''}\left(\{C_{v_i}, C_w\}\right), \\ &JS_{\pi''}\left(\{C_{v_j}, C_w\}\right)\} \end{aligned} \tag{3}$$

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

The objective function in Equation (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 2 would be trivially minimized if C_w was partitioned into $\|C_w\|$ singleton subsets, we need to constraint the solution.

The constraint in Equation (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 their 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 priors π'' to uniform in Equation (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 (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). 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 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). From all such candidate-pairs we pick the one that has the smallest value for $(\pi(C_{v_i}) + \pi(C_{v_j}))JS_\pi(\{C_{v_i}, C_{v_j}\})$ 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.

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

Corollary 1. *Merging clusters will always result in an increase in the value of the objective function.*

Proposition 1 can be proved by noting that the change in objective value due to merging sets C_{v_i} and C_{v_j} is

$$\begin{aligned}\delta = &\left(\pi(C_{v_i}) + \pi(C_{v_j})\right) JS_{\pi'}\left(\{c : c \in C_{v_i} \cup C_{v_j}\}\right) \\ &- \pi\left(C_{v_i}\right) JS_{\pi'}\left(\{c : c \in C_{v_i}\}\right) \\ &- \pi\left(C_{v_j}\right) JS_{\pi'}\left(\{c : c \in C_{v_j}\}\right) \\ &= (\pi(C_{v_i}) + \pi(C_{v_j}))JS_\pi(\{C_{v_i}, C_{v_j}\}) \end{aligned} \tag{4}$$

where $\pi' = \pi_c/\pi(C_v)$ are the class priors normalized within each cluster. Equation (4) can be obtained by using Theorem 4 in [DMK02]. Then Corollary 1 follows from the non-negativity of Jensen-Shannon divergence.

Proposition 2. *The* `findPartition` *algorithm in Figure 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 $JS_\pi(\{p_i\})$ is convex in p_i for a fixed π. This property of our algorithm ensures that

the procedure `constructTaxonomy` in Figure 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 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 of the intermediate clustering with 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 1. We refer the readers to the original paper for more details about the AIB algorithm.

Inspite of having the same cost function as our approach (ATG), AIB differs in two significant ways. Firstly, while ATG yields n-ary tree based taxonomies, AIB only constructs binary trees. Secondly, 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 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.

4 Experiments

In this section we report the results of empirical evaluation of n-ary and binary tree based taxonomies generated by ATG and AIB respectively on a variety of datasets. But first we describe the datasets as well as the experimental setup.

4.1 Datasets

We experiment on standard datasets that have classes related to each other by a possible hierarchical structure.

20-NEWSGROUPS. This standard text dataset[4] [Lan95] consists of 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 subcategory 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.

[4] `http://people.csail.mit.edu/jrennie/20Newsgroups/`

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 paper we use remote sensing data [5] 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 10 landcover types which can be broadly classified into the upland and wetland classes. Classes 1, 3, 4, 5 and 6 are all tress 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 6), 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 represents 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.

4.2 Implementation

For the 20-Newsgroups dataset, the data was preprocessed to remove 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.

[5] http://www.csr.utexas.edu/hyperspectral/codes.html

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-newsgroup 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 the Appendix, one can see that the ATG method yields n-ary taxonomies that reflects the underlying class affinities well. In the case of the 20-Newsgroups data, the first level of the ATG taxonomy (Figure 4) shows the different clusters of classes that exists 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, one has to look at the average digits representation [DMS99] to interpret the hierarchy better. Note that the average figures as illustrated in [DMS99] has 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. In particular, for the Glass dataset, we recover the exact hierarchical structure specified in the UCI-ML description of the dataset.

While the taxonomies generated by the AIB (Figure 5) 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 7) 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-newsgroup 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 clumped 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.

4.4 Classification Accuracies

A hierarchical taxonomy can be used as a classifier in which a multi-class problem can be broken down into a set of simpler problems. If the hierarchies are well-defined, each sub-problem would be simpler than the original one and would also typically require a smaller set of features to resolve it [KS97]. Hierarchical classifiers have been shown to give small improvements in classification accuracies over a flat set of classes [DC00, SKO01]. Another advantage of hierarchies is that when labeled data is scarce, shrinkage techniques [MRMN98] can be used to improve the parameter estimates at a node by using the corresponding estimates of its parent nodes.

In order to evaluate the utility of the hierarchy for classification, we made use of the taxonomies generated by each algorithm to build hierarchical classifiers. Experiments were performed using one-vs-all SVM classifier [RR01] at each level of an 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 (over 0.125, 0.25, 0.5, 1, and 2) as well as the kernel width of the Gaussians (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 extraction 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 as the base classifier in 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 2. The superior classification accuracies of the ATG-Bayesian classifiers, under the limited data conditions, validates our assumption about the utility of using a "more natural" tree to learn a hierarchical classifier. Figure 4.4 shows that for datasets like that of Text, even "powerful" classifiers such as SVMs benefit more 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 might speed 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. It can be seen that for all datasets using the ATG hierarchy 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 the 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.

5 Conclusions

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.

6 Acknowledgments

We would like to thank Srujana Merugu and Ravi Kumar for helpful discussions.

References

[AAS03] Giordano Adami, Paolo Avesani, and Diego Sona. Bootstrapping for hierarchical document classification. In *CIKM '03*, pages 295–302, New York, NY, USA, 2003. ACM Press.

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

[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.

[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.

[CKPT92] Douglass R. Cutting, David R. Karger, Jan O. Pedersen, and John W. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 318–329, New York, NY, USA, 1992. ACM Press.

[DC00] Susan Dumais and Hao Chen. Hierarchical classification of web content. In *SIGIR '00*, pages 256–263. ACM Press, 2000.

[DMDH02] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proc. WWW10*, pages 662–673, 2002.

[DMK02] Inderjit S. Dhillon, Subramanyam Mallela, and Rahul Kumar. Enhanced word clustering for hierarchical text classification. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 191–200, New York, NY, USA, 2002. ACM Press.

[DMS99] I. Dhillon, D. Modha, and W. Spangler. Class visualization of high-dimensional data with applications. Technical report, IBM Almaden Research Center, San Jose, CA 95120, 1999.

[FFR97] Adam Farquhar, Richard Fikes, and James Rice. The ontolingua server: a tool for collaborative ontology construction. *Int. J. Hum.-Comput. Stud.*, 46(6):707–727, 1997.

[GR04] Jacob Goldberger and Sam Roweis. Hierarchical clustering of a mixture model. In *NIPS*, 2004.

[GTC05] Stephen C. Gates, Wilfried Teiken, and Keh-Shin F. Cheng. Taxonomies by the numbers: building high-performance taxonomies. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 568–577, New York, NY, USA, 2005. ACM Press.

[HCCG05] J. Ham, Y. Chen, M. M. Crawford, and J. Ghosh. Investigation of the random forest framework for classification of hyperspectral data. *IEEE Trans. Geoscience and Remote Sensing*, 43(3):492–501, 2005.

[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.

[KGC01] S. Kumar, J. Ghosh, and M. M. Crawford. Best-bases feature extraction algorithms for classification of hyperspectral data. *IEEE Trans. Geoscience and Remote Sensing*, 39(7):1368–1379, 2001.

[KGC02] S. Kumar, J. Ghosh, and M. M. Crawford. Hierarchical fusion of multiple classifiers for hyperspectral data analysis. *Pattern Analysis and Applications, spl. Issue on Fusion of Multiple Classifiers*, 5(2):210–220, 2002.

[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.

[KS97] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 170–178, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[Lan95] Ken Lang. Newsweeder: Learning to filter netnews. In *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.

[Me04] D. R. Maddison and K.-S. Schulz (ed.). The tree of life web project: http://tolweb.org, 2004.

[Mor02] J. T. Morgan. *Adaptive Hierarchical Classifier with Limited Training Data*. PhD thesis, Dept. of Mech. Eng., Univ. 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 *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 359–367, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[NM00] N. F. Noy and M. A. Musen. Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of AAAI 2000*, pages 450–455. AAAI Press / The MIT Press, 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 *SIGUCCS '04: Proceedings of the 32nd annual ACM SIGUCCS conference on User services*, pages 256–261, New York, NY, USA, 2004. ACM Press.

[PRG05] Kunal Punera, Suju Rajan, and Joydeep Ghosh. Automatically learning document taxonomies for hierarchical classification. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1010–1011, New York, NY, USA, 2005. ACM Press.

[RR01] Jason Rennie and Ryan Rifkin. Improving multiclass text classification with the support vector machine. In *Massachusetts Institute of Technology. AI Memo AIM-2001-026*, 2001.

[SKO01] Eran Segal, Daphne Koller, and Dirk Ormoneit. Probabilistic abstraction hierarchies. In *Proceedings NIPS*, 2001.

[ST99] Noam Slonim and Naftali Tishby. Agglomerative information bottleneck. In *Proceedings of NIPS-12*, 1999.

[VD04] V. Vural and J. G. Dy. A hierarchical method for multi-class support vector machines. In *21st Intl. Conf. on Machine learning (ICML)*, 2004.

[ZLPY04] Li Zhang, ShiXia Liu, Yue Pan, and LiPing Yang. Infoanalyzer: a computer-aided tool for building enterprise taxonomies. In *CIKM '04*, pages 477–483, New York, NY, USA, 2004. ACM Press.

A Figures

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)
4. From P, select the pair (C_{v_i}, C_{v_j}) which has the lowest value for the expression in Equation (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 1: Pseudo-code for the proposed approach

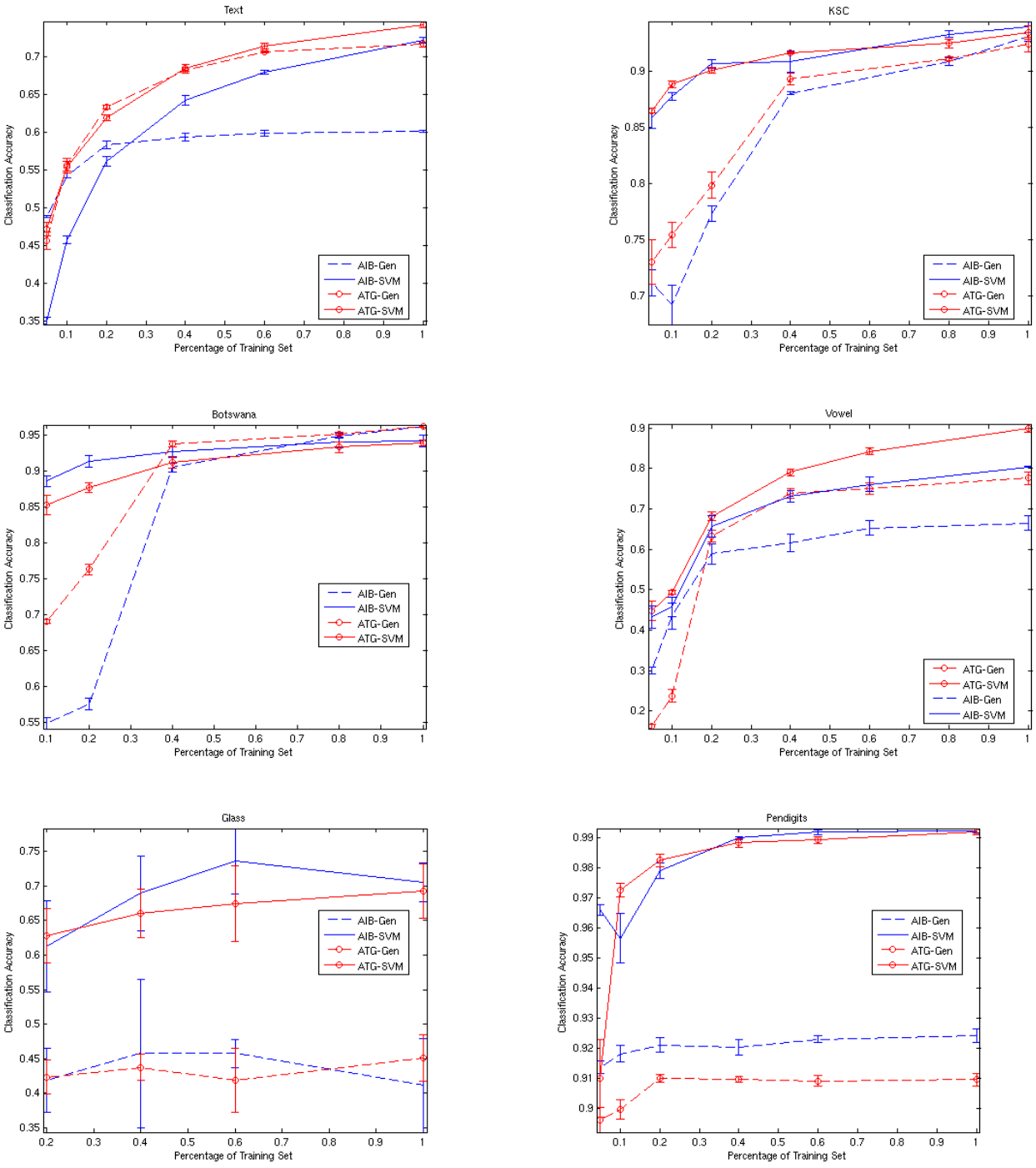

Figure 2: Learning rates using pre-learned hierarchies.

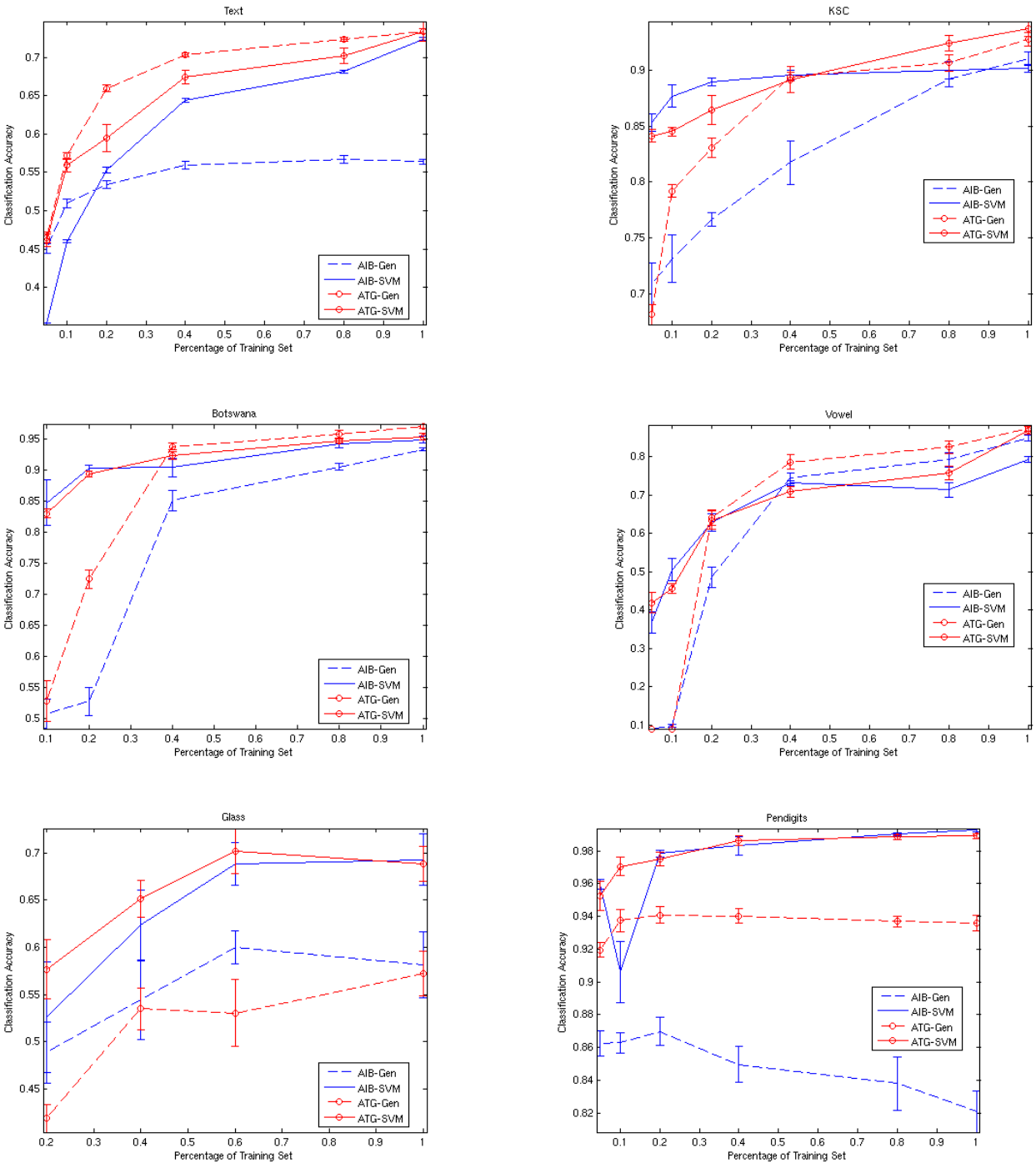

Figure 3: Learning rates when hierarchies are built from limited data.

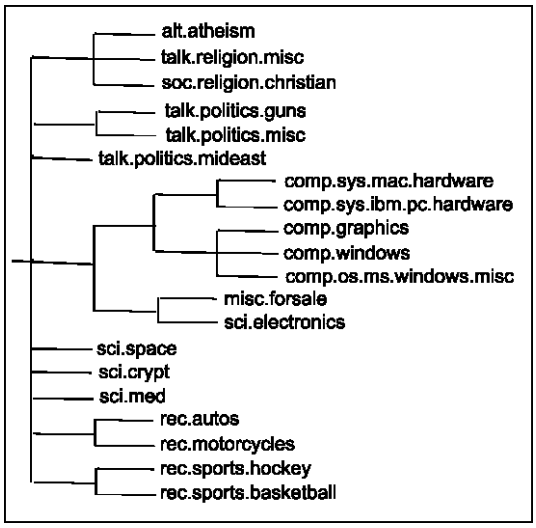

Figure 4: ATG on 20-newsgroup

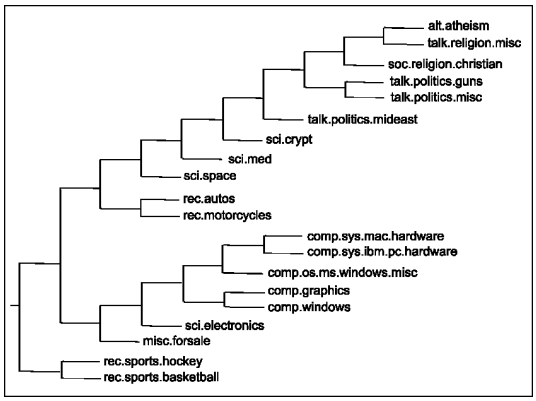

Figure 5: AIB on 20-newsgroup

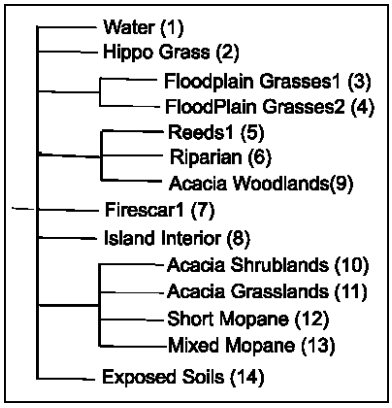

Figure 6: ATG on Botswana

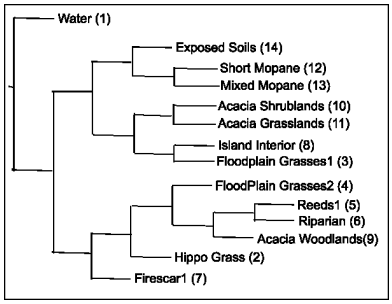

Figure 7: AIB on Botswana

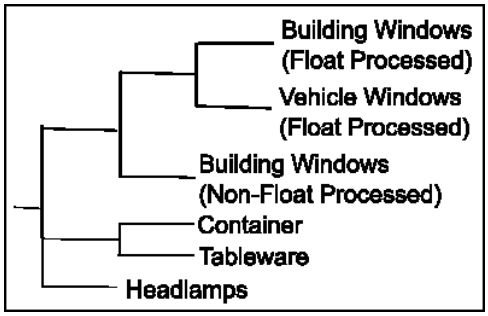

Figure 8: ATG on Glass

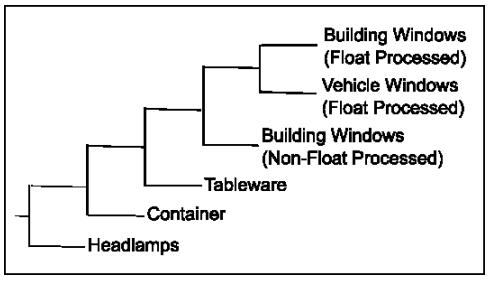

Figure 9: AIB on Glass