Acclimatizing Taxonomic Semantics for Hierarchical Content Classification

Lei Tang
Dept. of Comp. Sci. & Eng.
Arizona State University
Tempe, Arizona, USA
L.Tang@asu.edu

Jianping Zhang
AOL Inc.
22000 AOL way
Dulles, Virginia, USA
JZhang6805@aol.com

Huan Liu
Dept. of Comp. Sci. & Eng.
Arizona State University
Tempe, Arizona, USA
Huan.Liu@asu.edu

ABSTRACT

Hierarchical models have been shown to be effective in content classification. However, we observe through empirical study that the performance of a hierarchical model varies with given taxonomies; even a semantically sound taxonomy has potential to change its structure for better classification. By scrutinizing typical cases, we elucidate why a given semantics-based hierarchy does not work well in content classification, and how it could be improved for accurate hierarchical classification. With these understandings, we propose effective localized solutions that modify the given taxonomy for accurate classification. We conduct extensive experiments on both toy and real-world data sets, report improved performance and interesting findings, and provide further analysis of algorithmic issues such as time complexity, robustness, and sensitivity to the number of features.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Text Analysis*; I.2.6 [**Artificial Intelligence**]: Learning—*Knowledge acquisition*; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, Management, Experimentation

Keywords

Hierarchical Classification, Taxonomy Adjustment, Text Classification, Hierarchical Modeling

1. INTRODUCTION

The unregulated and open nature of the Internet and the explosive growth of the Web create a pressing need to provide various services for content categorization. One example of such services is parental control, which provides an mechanism for parents to prevent their children from accessing inappropriate online content. Automatic content categorization technologies are used by Web service providers to develop Web filters that can block children's access attempts to certain undesirable web sites and guide them to child-amicable sites. On one hand, parents want accurate content categorization that effectively blocks and guides; on the other hand, it is desirable for the service to provide explicit blocking structure and categories to explain how a decision is made to parents. Another useful property is that by providing the decision path how a blocking/non-blocking is made, it can aid experts in investigating any purported problems or complaints. If necessary, the expert can manually fine-tune the classification model. Typical flat models, which ignore the taxonomy information and treat content categorization as a multi-class problem, do not satisfy this requirement. The hierarchical classification attempts to achieve both accurate classification and increased comprehensibility. It has also been shown in literature that hierarchical models outperform flat models in training efficiency, classification efficiency, and classification accuracy [10, 13, 16, 8, 3, 23, 12].

Currently, almost all hierarchical methods rely on certain predefined content taxonomies. Content taxonomies are usually created for ease of content management or access, so semantically similar categories are grouped into a parent category. A subject expert or librarian is employed to organize the category labels into a hierarchy. Such a taxonomy is often generated independent of data (documents). Hence, there may exist some inconsistency between the given taxonomy and data, leading to poor classification performance. We elaborate some potential reasons. First, semantically similar categories may not be similar in lexical terms. Most content categorization algorithms are statistical algorithms based on the occurrences of lexical terms in contents. Hence, a semantically sound hierarchy does not necessarily lead to the intended categorization result.

Second, even for the same set of categories, there could be different semantically sound taxonomies. Semantics cannot guarantee a unique taxonomy. Various users might use different taxonomies to organize the categories. For example, we have a number of sports teams. Some people might first separate them according to the area (like Arizona, California, Oregon, etc.) and then according to the sports type (e.g., football, basketball, etc.). Others might first separate the sports teams according to the sports type and then areas. Both taxonomies are reasonable in terms of semantics. From the perspective of hierarchical classification, however, the

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.

two taxonomies will likely result in different performance. Hence, we need to investigate the impact of different hierarchies (taxonomies) on classification.

In addition, semantics may change over time. For example, when the semantic taxonomy was first generated, people would not expect the category *Hurricane* related to *Politics*, and likely put it under *Geography*. However, after investigating the data recently collected, we notice that a good number of documents in category *Hurricane* are actually talking about the disasters Hurricane Katrina and Rita in the United States and the responsibility and the faults of FEMA[1] during the crises. Based on the content, it is more reasonable to put *Hurricane* under *politics* for better classification. The example demonstrates the stagnant nature of a taxonomy and the dynamic change of semantics reflected in data. This motivates the data-driven adaptation of a given taxonomy in hierarchical classification.

In this paper, we first review the related work on hierarchical content classification and taxonomy generation in Section 2; empirically show the impact of different taxonomies on classification in Section 3; formulate the problem and discuss about the challenges in addressing the problem in Section 4; investigate representative pathologies and study why and how a semantically sound taxonomy can be adapted using available data and introduce an effective method in Section 5; present the experimental results and further study in Sections 6 and 7; and conclude this work in Section 8.

2. RELATED WORK

We briefly survey the work on state-of-the-art hierarchical classification and taxonomy generation.

2.1 Hierarchical Classification

A semantically sound taxonomy can be used as a base for a divide-and-conquer strategy. A classifier is built independently at each internal node of the hierarchy using all the documents of the subcategories of this category, and a document is labeled using these classifiers to greedily select sub-branches until we reach a leaf node, or certain constraints are satisfied(like the score should be larger than a threshold [8] or the predictions of adjacent levels should be consistent [22]). Feature selection is often performed at each node before constructing a classifier [10, 4].

To build the hierarchical model, different base classifiers are employed including association rules [20], naïve Bayes classifiers [10], neural networks [21, 16], and support vector machines [8, 17, 12]. As the greedy approach for classification might be too optimistic, researchers propose to traverse all the possible paths from the root to the leaves. In [8], the authors use a sigmoid function to map the prediction of support vector machine at each node to a probability and then multiply these probabilities along one path. The path with the highest probability is selected. Another way is to set a threshold at each level and just take those branches when the corresponding prediction's score is larger than the threshold. It is demonstrated that hierarchy model outperforms flat(non-hierarchical) model marginally. And these two methods show little difference. In [10], a greedy approach with naïve Bayes classifiers is exploited and a significant accuracy improvement is observed.

[1]Federal Emergency Management Agency

One advantage of the hierarchy-based approach is its efficiency in training and testing, especially for a very large taxonomy [23, 12]. Hierarchical models make it easy to modify and expand a taxonomy, like add one sub-category, delete one category, merge several into one. For each modification, it is not necessary to update the classifiers of all the nodes since the classifiers are built independently. We just need to update a small portion of the classifiers. Therefore, the hierarchical approach is preferred when facing a large taxonomy.

Hierarchies can also be used to assign different misclassification costs. Recently, new hierarchical classification based on margin theory and kernel methods are introduced [6, 19, 3, 15]. A concomitant of these methods' superior performance is their unbearable computational cost for training.

However, we notice that all the previous works paid little attention to the quality of the taxonomy which we need to consider in real-world applications.

2.2 Taxonomy Generation via Clustering

Some researchers propose to generate a taxonomy from data for document management or classification. Most of these works exploit some hierarchical clustering algorithms for this task. There are two directions for hierarchical clustering: agglomerative and divisive.

In [2, 5, 11], all employ a hierarchical agglomerative clustering (HAC) approach. In [2], the centroids of each class are used as the initial seeds and then projected clustering method is applied to build the hierarchy. During the process, the cluster with too few documents is discarded. Thus, the taxonomy generated by this method might have different categories than predefined. The authors evaluate their generated taxonomies by some user study and find it is comparable to the Yahoo directory. In [11], a linear discriminant projection is applied to the data first and then a hierarchical clustering method is exploited to generate a dendrogram, which is a binary tree. For classification, the authors change the dendrogram to a two-level tree according to the cluster coherence, and hierarchial models yield classification improvement over flat models. But it is not sufficiently justified why a two-level tree should be adopted. Meanwhile, [5] proposes HAC+P which is similar to the previous approach. Essentially, it adds one postprocessing step to automatically change the binary tree obtained from HAC, to a wide tree with multiple children. However, we have to specify parameters as the maximum depth of the tree, the minimum cluster size and the cluster number preference at each level. These parameters make this method very ad hoc.

Comparatively, the work in [14] falls into the category of divisive hierarchical clustering. The authors generate a taxonomy with each node associated with a list of categories. Each leaf node has only one category. This algorithm basically uses two centroids of categories which are furthest as the initial seeds and then applies Spherical K-Means [7] with $k = 2$ to divide the cluster into 2 sub-clusters. Each category is assigned to one sub-cluster if most of its documents belong to the sub-cluster (its ratio exceeds a predefined parameter). Otherwise, this category is associated to both sub-clusters. Another difference of this method from other HAC methods is that it will generate a taxonomy with one category possibly occurring in multiple leaf nodes.

Though the clustering approach could generate a new taxonomy, it has some limitations: it either generates a binary

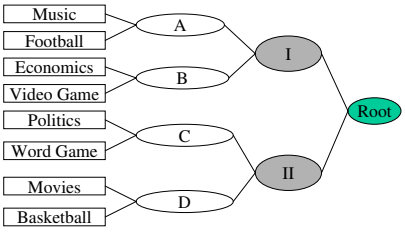

Figure 1: A "bad" hierarchy

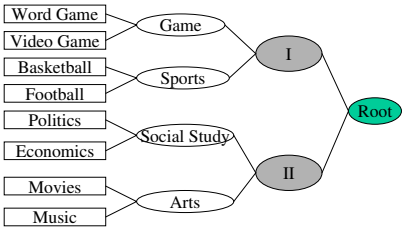

Figure 2: A "good" hierarchy

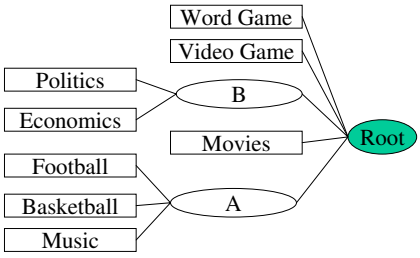

Figure 3: An adjusted hierarchy

tree, or requires to specify the maximum depth of the taxonomy or the number of children nodes under each parent category, which turns out to be rather ad hoc. Our experiments show that the clustering-based hierarchy does not perform as well as the semantics-based hierarchy (Sec. 6.3).

3. TAXONOMIES IN HIERARCHICAL CLASSIFICATION

To show that taxonomies impact on classification accuracy, we conducted a proof-of-concept experiment using 1800 web pages obtained from AOL.com [1] with 8 categories: *Basketball*, *Football*, *Politics*, *Economics*, *Movies*, *Music*, *Video Game*, and *Word Game*. We filter out the tag information and extract the text and meta information in the pages. Each page is represented as a vector of words. We organize the 8 categories into three different taxonomies as shown in Figures 1, 2 and 3, respectively. The hierarchy in Figure 1, a relatively "bad" hierarchy, is obtained by grouping dissimilar categories, while the hierarchy in Figure 2, a relatively "good" hierarchy, is obtained by grouping similar categories. Actually, we can obtain the same "good" hierarchy by always joining two nearest clusters if we define the dissimilarity of two categories as the distance between the centroids of the two categories. The "bad" hierarchy is generated by always merging two clusters with the largest distance. The "good" hierarchy is semantically sound. The third hierarchy in Figure 3 is adjusted from the hierarchy in Figure 1 by applying our algorithm described in Section 5.

We build hierarchical classification models on the three hierarchies, respectively. A classifier is built for each of the nodes in the hierarchy. A naïve Bayes classifier was used. The greedy search algorithm was used for classifying a document. Figure 4 shows the average result of macro-averaged recall and F-measure with a 10-fold cross validation. The x-axis is the number of features selected using information gain at each internal node of the hierarchy.

Clearly, a significant classification difference exists between these three hierarchical models, especially when the number of features selected in each internal node is relatively small. The difference diminishes as more features are selected. In Section 7.3, we discuss this phenomenon in detail. We notice that the hierarchy in Figure 3 performs better than the semantically sound, "good" hierarchy. This observation suggests that we can improve the classification performance even for a semantically sound hierarchy.

In practice, a semantics based taxonomy is always exploited for hierarchial classification. As mentioned in the introduction, the taxonomic semantics might not be compatible with specific data and applications and can be ambiguous in certain cases, the semantic taxonomy might lead

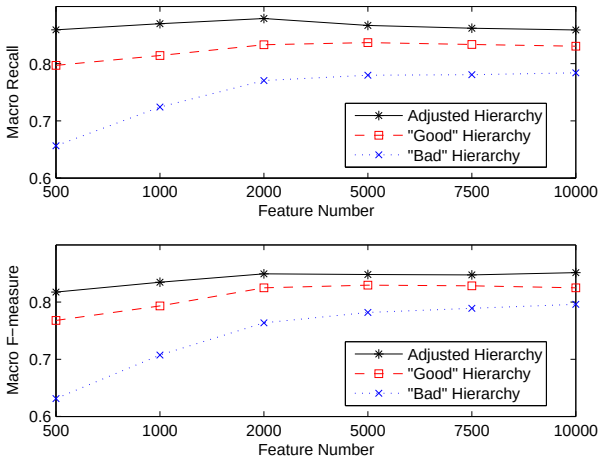

Figure 4: Performance of different hierarchies

hierarchical classifications astray. Our question is whether or not we can find a better hierarchy than the given semantically sound hierarchy so that a reasonably good hierarchical model can be derived for data classification.

One possible approach to address the problem is to "start from scratch": ignore the given taxonomic information and generate a new taxonomy based on the data. As we see in Section 2.2, some researchers have done some work on this approach based on hierarchical clustering. But all the approaches either generate a binary tree which might not be good for hierarchical classification, or requires humans to set the height and the number of children under each node.

The semantics based taxonomy is a form of prior knowledge and provides valuable information. With this prior knowledge, we can narrow down our hypothesis space and efficiently find reliable hierarchies with good classification performance and generalizability. Therefore, instead of abandoning the given hierarchy information, we propose to modify the given hierarchy gradually to generate a data-driven hierarchy, so that classification improvement can be achieved.

4. PROBLEM FORMULATION

Before we formalize our problem, we present several definitions concerning hierarchies as follows.

DEFINITION 1 (ADMISSIBLE HIERARCHY). *Let $L = \{L_1, L_2, \cdots, L_m\}$ denote the categories at the leaf nodes of a taxonomy H, and $C = \{C_1, C_2, \cdots, C_n\}$ denote the categories of data D. H is an admissible hierarchy for D if $m = n$ and there's a one-to-one mapping between L and C.*

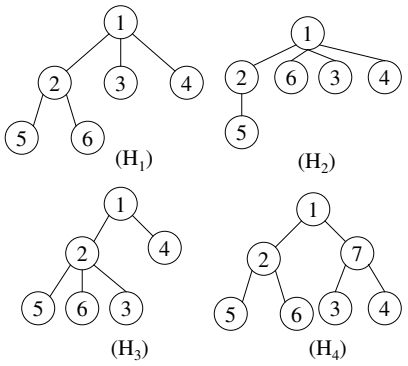

Figure 5: Elementary Operations. H_1 is the original hierarchy. H_2, H_3 and H_4 are obtained by performing different elementary operations. H_2: Promote node 6; H_3: Demote node 3 under node 2; H_4: Merge node 3 and node 4.

Definition 2 (Optimal Hierarchy).

$$H_{opt} = \arg\max_H p(D|H) = \arg\max_H \log p(D|H)$$

where H is an admissible hierarchy for the given data D.

In other words, the optimal hierarchy given a data set should be the one with maximum likelihood. The brute-force approach to find the optimal hierarchy is to try all the admissible hierarchies and output the optimal one. Unfortunately, even for a small set of categories, there could be a huge number of admissible hierarchies. If there are n categories, there are at least $n \times (n-1) \times \cdots \times 1 = n!$ different binary trees. Not to mention those trees with multiple children. Thus, it is impractical to try all the possible hierarchies and pick the optimum and a more effective way should be explored.

The given hierarchy provides valuable information for classification and helps reduce the search space to find the intended optimal hierarchy. In order to incorporate this knowledge into into our problem formulation, we first give the definition of hierarchy difference.

Definition 3 (Hierarchy Difference). *Hierarchy difference between two admissible hierarchies H and H' is the minimum number of elementary operations(see below) to transform H into H'. Suppose the minimum number of operations is k, we denote the difference between H and H' as*

$$\| H' - H \| = k$$

In order to change a hierarchy to another admissible hierarchy, we have three elementary operations:

- **Promote**: roll up one node to upper level;
- **Demote**: push down one node to its sibling;
- **Merge**: merge two sibling nodes to form a super node;

As shown in Figure 5, H_1 is the original hierarchy. H_2, H_3 and H_4 are obtained by promoting Node 6 to its upper level, demoting Node 3 under its sibling Node 2, and merging Node 3 and 4, respectively. Node 7 is a newly generated node after modification. Notice that the set of leaf nodes remain unchanged. As the set of categories at the leaf nodes remains unchanged, we do not require the operation of splitting one node into two.

Given explicit hierarchy difference, we have the constrained optimal hierarchy defined below.

Definition 4 (Constrained Optimal Hierarchy). *Given a hierarchy H_0, if there exists a sequence of hierarchies $Q = \{H_1, H_2, \cdots, H_n\}$ such that*

$$\begin{aligned} p(D|H_i) &\geq p(D|H_{i-1}) \\ \| H_i - H_{i-1} \| &= 1 \qquad (1 \leq i \leq n) \end{aligned}$$

and

$$\begin{aligned} &\forall H' \qquad s.t. \| H' - H_n \| = 1 \\ &we\ have \quad p(D|H') \leq p(D|H_n) \end{aligned}$$

then H_n is a constrained optimal hierarchy for H_0 and D.

Actually, the constrained optimal hierarchy(COH) is the hierarchy that is reachable from the original hierarchy following a list of hierarchies with likelihood increase between consecutive ones. When we reach a COH, we cannot find a neighbor hierarchy with higher likelihood than it. In nature, each COH is a local optimum.

If we state our problem as that of search, then a given hierarchy is a sensible starting point in our attempt to reach the optimal hierarchy following a short path. Hence, we formulate our challenge as follows.

Hierarchy Search Problem: *Given data D, and a taxonomy H_0, can we find a hierarchy H_{opt} such that*

$$H_{opt} = \arg\max_H \log p(D|H)$$

where H is a constrained optimal hierarchy for D and H_0.

5. EFFECTIVE SOLUTION

To address the hierarchy search problem, we first need to address the following:

1) How to estimate the likelihood of data given a hierarchy ($P(D|H)$ in Definition 2)?

2) While the hierarchy search problem proposes to select the best among the constrained optimal hierarchies, it is computationally intractable to obtain all the constrained optimal hierarchies.

3) How to find the neighbors of a hierarchy? There could be a huge number of neighbors by performing only one elementary operation for a specific hierarchy especially when the number of nodes in the tree is large.

Hence, we propose to obtain an approximate solution by developing some heuristics. For the first problem, instead of directly estimating the likelihood of data given a hierarchy, we use some statistics to estimate the quality of a hierarchy. Since in most classification tasks, people focus on macro-averaged recall or f-measure, we use them as the classification performance of a hierarchical model to measure its likelihood. In particular, we use macro-averaged recall to estimate the conditional likelihood. Concerning the second problem, we exploit a greedy approach to find the best constrained hierarchy. In each search step, we always choose the neighbor node with largest likelihood improvement. As we still need to consider the number of neighbors of a hierarchy, we can apply certain heuristics to find those promising neighbors and remove those non-promising from further consideration. In this section, we design some heuristics by studying several pathologies with various hierarchies and then provide an algorithm that accustoms the given taxonomy according to data.

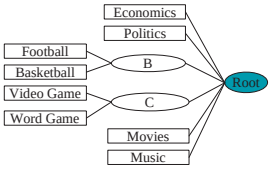

Figure 6: H_A

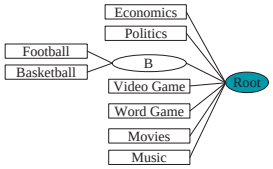

Figure 7: H_B

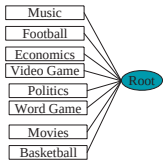

Figure 8: H_C

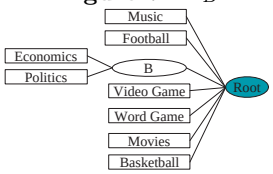

Figure 9: H_D

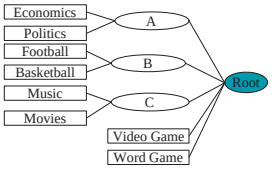

Figure 10: H_E

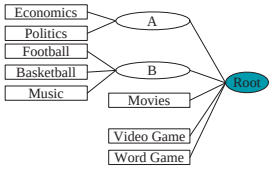

Figure 11: H_F

Table 1: Performance Improvement of Hierarchies. (Results are obtained by 10-fold cross validation by selecting 500 features at each node.)

	H_A	H_B	H_C	H_D	H_E	H_F
Recall	0.8177	0.8350	0.8392	0.8532	0.8452	0.8531
F	0.7980	0.8049	0.8074	0.8168	0.8062	0.8128

5.1 Some Pathologies

We examine three typical cases involving hierarchy adaptation as follows:

Pathology 1: A child category is not represented well at the parent node. This situation is common especially when the number of categories is large or the distribution of categories is imbalanced. In order to escape from the shadow of the parent node, we need to roll up the node to its parent level. Figure 6 shows one toy example of a semantics-based hierarchy. However, the categories *Word Game* and *Video Game*, though both related to games, actually is not very proper to put under the same parent node C, as it doesn't represent its two children well. So when we roll up one of its child node to the upper level resulting in hierarchy H_B as in Figure 7, the macro-recall as in Table 1 increases from 0.8177 to 0.8350.

Pathology 2: Two nodes under the same parent node share so many common features that they become indistinguishable. For this case, we can consider merging these two nodes to form a super node. We hope that by selecting the common features first, we can separate the two categories at lower level by selecting more specific features. The cause of ambiguity might be that one node resembles strongly some of the other node's subcategories. Then another possible solution is to demote one node under the other node. We also use a simple example to show this. Figures 8 and 9 demonstrate this situation. As we merge *Economics* and *Politics*, not surprisingly, the classification recall improves to 0.8452 from 0.8531.

Pathology 3: It is possible that one category is related to two different parent categories. But in the hierarchy, it can only be located in one place. So for a specific application,

Input:
H_0: Predefined hierarchy
T: Training data
V: Validation set
δ: Stopping criterion
Output:
H_{best}: the approximate best hierarchy
Algorithm:
1 $S_{pre} = 0$; $H_{best} = H_0$;
2 S_{best}=evaluateHierarch(H_0,M,V);
3 $O_{flag} = false$;
4 **while** ($S_{best} - S_{pre} > \delta$)
5 N_{list}={all nodes in H_{best}};
6 **repeat**
7 $Node$=getNodeToCheck(N_{list});
8 H_{list}=generateNeighbors($Node$,O_{flag});
9 [H, S]=findBest(H_{list});
10 **if** $S > S_{best}$
11 $S_{pre} = S_{best}$; $S_{best} = S$; $H_{best} = H$;
12 updateNodeList(N_{list}, H_{best}, $Node$);
13 **end**
14 **until** $N_{list} == null$;
15 $O_{flag} = \neg O_{flag}$;
16 **end**
17 **return** H_{best}

Figure 12: Hierarchy Adjusting Algorithm

this semantic-based hierarchy might not be good. H_E in Figure 10 and H_F in Figure 11 demonstrate this situation. Again, H_E is a semantic-justified hierarchy. However, in our toy data, there are some articles talking about sports music. Thus the hierarchial model on the data yields better result(from 0.8452 to 0.8531 for recall) when we put *Music* under sports as in H_F. For such kind of symptoms, we have to lift *Music* first and then demote it under B, essentially involving two elementary operations.

The three phenomena help us develop heuristics aiming to reach a better hierarchy via certain operations to one node. As we know, the nodes at upper level affect more in the classification process and thus should be considered with higher priority. This is equivalent to a preference to check the node with shortest depth first in search of promising nodes to expand.

5.2 Algorithm

Our algorithm consists of multiple iterations(Figure 12). In each iteration, we traverse the hierarchy using a top-down approach and search for better hierarchies. For each search step, we have the following procedures:

1. Identification of the node to check.
2. Identification of promising hierarchy neighbors concerning a node.
3. Identification of the best neighbor.
4. Update of current best hierarchy.

We discuss each procedure below.

5.2.1 Identification of the node to check

Clearly, the nodes at the upper level affects more in the classification process and should be considered with higher

Procedure: getNodeToCheck()
Input: N_{list}, A list of nodes in a hierarchy
Output: $Node$, the node to check
  check all the nodes in the list;
  set $Node$ to the node with the highest level;
  remove $Node$ from the list N_{list};
  return $Node$;

Procedure: generateNeighors();
Input: N, the node to check;
  O_{flag}, the operation flag to denote promote operation or merge/demote operation.
Output: C_{list},a list of promising hierarchy neighbors
  if $O_{flag} == false$
    H_{cand} =hierarchy by promoting N;
    $C_{list} = \{H_{cand}\}$;
  else
    $N_{similar}$ =the most ambiguous sibling node for N;
    H_1 =hierarchy by merging N and $N_{similar}$;
    H_2 =hierarchy by demoteing N as $N_{similar}$'s child;
    H_3 =hierarchy by demoting $N_{similar}$ as N's child;
    $C_{list} = \{H_1, H_2, H_3\}$;
    remove invalid hierarchies from C_{list};
  end
  return C_{list}.

Procedure: updateNodeList();
Input: N_{list}, the node list needs to check;
  H, the hierarchy representing the operation;
  $Node$, the node being checked;
Output: an updated node list N_{list}
  switch ($H.operation$)
    case promote: N=$Node$'s grandparent;
      add all N's descendants to N_{list};
      break;
    case merge:
    case demote: N=$Node$'s parent;
      add all N's descendants to N_{list};
      break;
  end
  return N_{list};

Figure 13: Procedure definitions

priority. Therefore, we maintain a list of nodes in the hierarchy. At each step, we pick the node with shallowest depth and remove it from the list to avoid future consideration(Please refer to Figure 13 for details).

5.2.2 Identification of promising neighbors

As argued in previous sections, the number of neighbors of one hierarchy could be huge. Therefore, rather than considering all the nodes in the tree to generate the hierarchy, we focus on performing operations to one specific node in the hierarchy at each step.

However, the priority of each elementary operations must not be neglected. As we see in previous pathologies, especially for the third one, we need to perform promoting operations first to achieve better taxonomy. This operation could break up all the bad parent-child relations and make the hierarchy more consistent with the data in general. Thereafter, merging and demoting are employed to adapt the hierarchy more specifically consistent for hierarchical classification. So we always check promoting a node first to avoid getting stuck under a wrong parent node. Therefore, in one iteration, we just check the promising hierarchies by performing promoting operations. In another iteration, we just check the hierarchies by performing demoting or merging.

When we perform merging or demoting to one node, it is not necessary for us to try all the possible pairs of nodes under the same parent. We can just focus on the category which is most similar to the node we currently check. We define the similarity in terms of ambiguity score below:

DEFINITION 5 (AMBIGUITY SCORE). *Given two classes A and B, suppose the percentage of class A classified as class B is P_{AB}, and the percentage of class B classified as Class A is P_{BA}, then ambiguity score = P_{AB} + P_{BA}.*

Therefore, for one node, we just pick the sibling node with highest ambiguity score and generate possible good neighbors by merging these two nodes and demoting one node to the other. Notice that not all the neighbor hierarchies are valid. If one leaf node becomes a non-leaf node, it is invalid as categories are the leaf nodes in this work. These invalid hierarchies must be removed from consideration. The detailed procedure is in Figure 13.

5.2.3 Identification of the best neighbor

This procedure compare all the promising neighbors and find the best hierarchy among them. Given a list of hierarchies, we just build a hierarchical model based on each hierarchy, and then evaluate it on some validation data to attain some classification statics(in particular, macro-averaged recall in our work). Then the best hierarchy and the corresponding statistic are returned.

5.2.4 Update of current best hierarchy

After we obtained the best hierarchy in the neighbor list, we could compare it with current best hierarchy. If the classification statistic is better than current value, then we replace current best hierarchy with the best hierarchy just found and update the list of nodes to check. Otherwise, we remain unchanged and continue to check the next node.

Each time we changed the hierarchy, we have to update the list of nodes to check(refer to Figure 13). We actually just push to the list all the nodes that will be affected by the operation, and all the other nodes in the list remain unchanged. Suppose N is the node being checked. If the hierarchy is obtained by performing promoting, then all the children of N's grandparent should be rechecked. We can revisit the cases in Figure 5. H_2 is generated by promoting node 6 in H_1. If H_2 is just a subtree in a huge taxonomy, then all the other nodes' classifiers except the descendants of node 1 remain unchanged. So we just push all the descendants of node 1 into the list. Similarly, when we perform merging and demoting we just need to push all the descendants of N's parent to the list. Therefore, as we perform demoting and merging to node 3 in H_1 resulting in H_3 and H_4, respectively, only the subtree of node 1 will be affected. All the changes are local and we just update the nodes that is affected by the modification.

More importantly, as we use top-down approach to traverse the tree, we avoid unnecessarily checking the nodes at lower levels whenever there's a change at higher levels.

The detailed algorithm to adjust a hierarchy is presented in Figure 12. In sum, the algorithm basically consists of multiple iterations. In each iteration, we check each node of the taxonomy in a top-down approach and generate promising hierarchies (neighbors) according to an operation flag. It is believed that promoting should perform first, so as in Figure 12, we set the flag to $false$ at the initial iteration(Line 3). Then the operation flag is switched to $true$ at the end of one iteration(Line 15), so that in the next iteration, we merge two nodes or demote one node to deepen

the hierarchy. This pairwise iterations will keep going until the performance improvement on the validation set is lower than the predefined stopping criterion.

6. EXPERIMENTS AND RESULTS

Earlier we show that semantics-based hierarchy does not necessarily lead to a good hierarchial classification model. We conduct experiments with different data sets including the toy data and some real-world application.

6.1 Experiment Settings

We perform 10-fold cross validation to all the data sets. In each fold, we apply our algorithm to the training data with a predefined hierarchy. After we obtain the adjusted hierarchy, we build hierarchical models based on training data by selecting various number of features at each node. The model is then evaluated on the test data. As the class distribution of the data is not balanced, we report the average results in terms of macro recall and macro F-measure. When we apply our hierarchy adjusting algorithm to the training data, the criterion to evaluate the quality of a hierarchy is macro-averaged recall. 500 features are selected using information gain to build the hierarchical model. To gain efficiency, the classifier at each node we exploited is multi-class multinomial naïve Bayes classifier(NBC). The data fragmentation problem becomes serious with a large number of categories. Keeping a portion of training data as the validation set becomes pretty unstable and might lose generalization capability. For simplicity, we set the validation set the same as the training data to guide the hierarchy modification. The stopping criterion for hierarchy adaptation is until no classification performance can be improved on the training data.

6.2 Results on Toy Data

We first test our algorithm on the toy data sets described in Section 3. Figure 14 shows the performance improvement after hierarchy modification. The legend *"Good" Hie Adjust* and *"Bad" Hie Adjust* stand for the performance of hierarchy adjustment starting from the "good" hierarchy (Figure 2)and the "bad" hierarchy (Figure 1), respectively. Clearly, after hierarchy modification, the performance improves for both cases, especially when the number of features is small (say 500).

Interestingly, even if the predefined hierarchy is very bad or even misleading, our algorithm can still recover to a reasonably good hierarchy whose performance is even better than the hierarchy given by intuition. Actually, the hierarchy in Figure 3 is one example of hierarchies adjusted starting from the bad hierarchy. Though our algorithm starts from a semantically unsound hierarchy, it is possible to reach a semantically sound hierarchy finally.

6.3 Results on Real-World Data

We applied our algorithm to two real world data sets provided by an Internet company. One is about the topics of social study (Soc); the other includes topics of children's interests (Kids). Categories of both data sets are organized as a taxonomy. Text and meta information were extracted from web pages and the vector space model was applied to represent web pages. Table 2 summarizes the two data sets.

In order to examine if a predefined semantics-based hierarchy can provide useful prior knowledge for search, we also

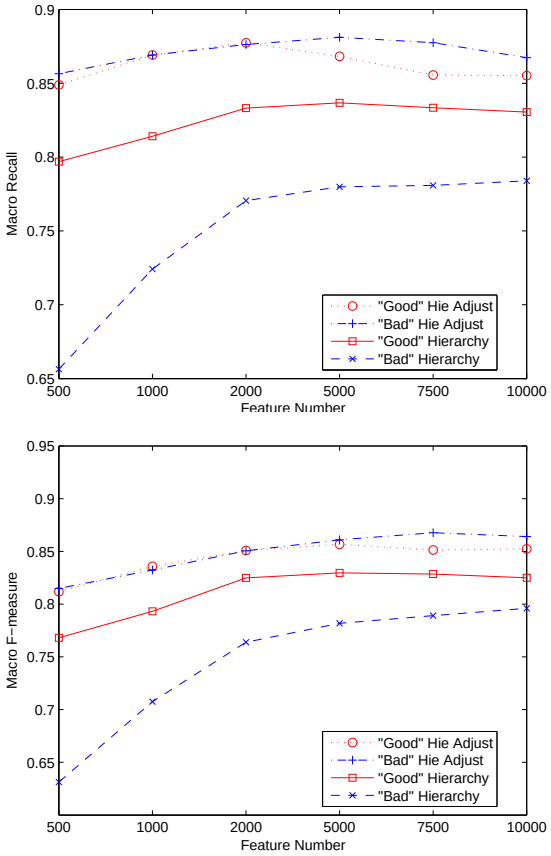

Figure 14: Performance on Toy data

Table 2: Real-World Data Description

	Soc	*Kids*
#classes	69	244
#nodes in the hierarchy	83	299
Height of the hierarchy	4	5
#instances	5248	15795

compared with the "start from scratch" approach: ignore the predefined taxonomy and do hierarchial clustering on training data to obtain the taxonomy. We did a preliminary study to compare a divisive clustering approach in [14] with agglomerative clustering algorithm as in [5], and found the latter approach(HAC+P) is not comparable to the former clustering approach for our application. The difficulty lies at choosing proper critical parameters of HAC+P like the dimensionality to calculate the similarity, the number of maximum depth and the preferred number of clusters of each node. Therefore, we just use the former clustering approach as the baseline in our experiment.

The curves of "clustering" in Figures 15 and 16 denote the performance of the clustering approach. There is a clear association between the performance and the number of categories. It is reasonable to expect that the recall and F-measure are not very high as we have 69 categories in *Soc* and 244 classes in *Kids*. The semantics based hierarchy eventuates better hierarchical classification performance than the clustering-based hierarchy. This set of results also indicates that the prior knowledge embedded in a taxonomy is useful in classification.

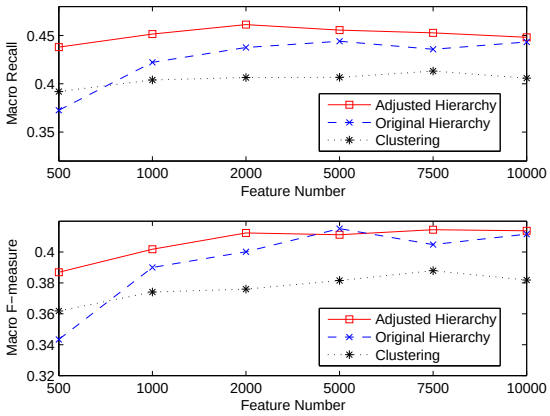

Figure 15: Performance on *Soc* Data

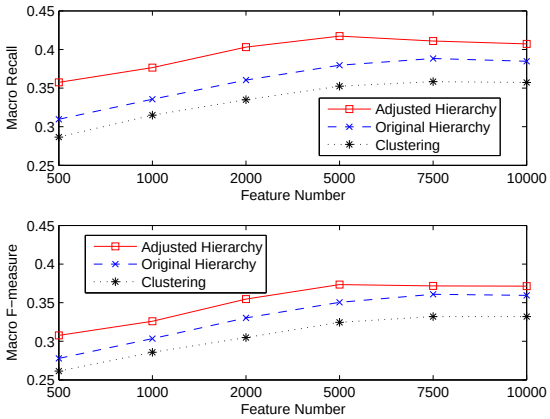

Figure 16: Performance on *Kids* Data

Comparatively, our algorithm, which starts from the given hierarchy, could achieve significant improvement over the original taxonomy on both data sets. This shows that we can automatically adjust the content taxonomies for more accurate classifiers. One interesting result is: when we select more and more features, the difference between the newly generated hierarchy and given hierarchy wanes. We discuss this in Section 7.3.

Tables 3 and 4 summarize some statistics on both data sets: the number of iterations, hierarchy evaluations, hierarchy modification elementary operations and the height of the hierarchy after modification. After a constant number of iterations, our algorithm will stop at a hierarchy which performs better than the original hierarchy.

7. FURTHER ANALYSIS

The experiments have shown that given data and a predefined hierarchy, we can find a better taxonomy for accurate classification. As we use a "wrapper" model: build a hierarchical model and evaluate it for each neighbor hierarchy, we need to check its time complexity. Besides, we keep modifying the hierarchy until no performance improvement on the training data is observed, is it possible that the hierarchy over-fits the data? Here, we provide further analysis and discuss how the number of features affects the performance.

7.1 Time Complexity

Though our algorithm exploits a kind of wrapper model in search of a better hierarchy, the time complexity of algorithm is still linear in terms of, the number of categories and instances in the data. For naïve Bayes classifier, the training time and testing time is linear to the number of instances and dimensionality. For each category, we could summarize the statistic of term given classes using just one vector. Then, building a hierarchial model just costs $O(C_o d)$ where C_0 is the number of internal nodes in the hierarchy, and d is the dimensionality. However, evaluation still costs $O(hnd)$ where h is the average height of the hierarchy and n is the number of instances in the validation data. Therefore, for our algorithm, the main computational cost is spent on evaluating hierarchical models. Hence, the number of evaluations determines our time complexity.

In each iteration, we check each node in the hierarchy. For each node, we can generate no more than 3 neighbors as there are only three possible elementary operations. Let c denote the number of nodes in the hierarchy, then a node can never be checked more than c times in one iteration. The worst case is that each time we update the nodes list after checking a new node, we have to recheck the previous checked nodes . Then, the worst time complexity is for one iteration is $O(c^2hnd)$.

However, the bound above is pretty loose. As we take top-down traversal of the tree and all the hierarchy changes are local, the worst case can seldom happen based on a semantic-based hierarchy. In reality, on average, a node will be checked no more than twice in one iteration. As in Table 3, the average number of evaluations of one iteration is $539.8/5.6 = 96.39$. The number of nodes in the original hierarchy is 83, hence, each node will be checked roughly $96.39/83 \doteq 1.16 < 2$ times. Similarly, on *Kids* data (Table 4), each node will be checked roughly $3343.5/(9.7*299) \doteq 1.15 < 2$ times in one iteration. Hence, empirically, the time of one iteration should be roughly $O(2chnd) = O(chnd)$.

Table 3: *Soc* Performance statistics

Fold	Iterations	Evaluations	Operations	Height
1	6	582	43	6
2	6	585	37	9
3	3	265	33	7
4	5	449	41	6
5	7	805	67	13
6	8	754	68	6
7	5	448	48	9
8	5	494	42	11
9	3	268	44	5
10	8	748	62	7
ave	5.6	539.8	48.5	7.9

Table 4: *Kids* Performance statistics

Fold	Iterations	Evaluations	Operations	Height
1	12	4244	217	13
2	9	3014	172	10
3	10	3483	224	13
4	8	2275	190	13
5	7	2379	159	12
6	12	4181	166	15
7	10	3640	230	16
8	9	3312	185	18
9	11	3733	215	12
10	9	3174	221	16
ave	9.7	3343.5	197.9	13.8

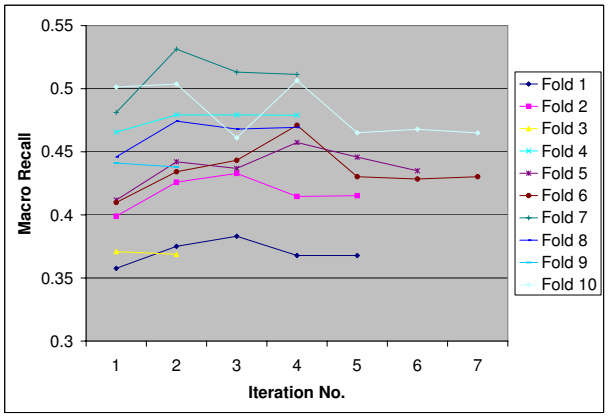

Figure 17: Over-fitting on *Soc*

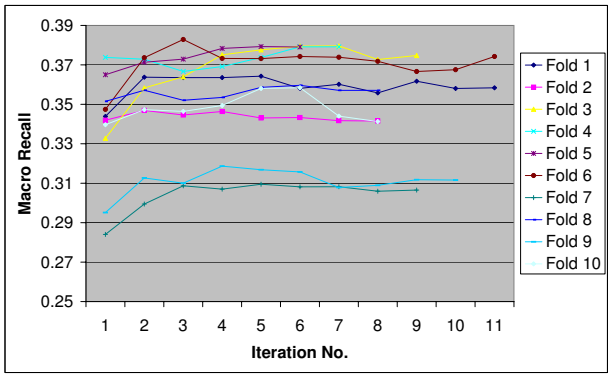

Figure 18: Over-fitting on *Kids*

In practice, the number of iterations is bounded by a constant(we use I to denote it). Hence, the total time complexity of our algorithm is $O(Ichnd)$ which is linear. Here, I is the number of iterations, c is the number of nodes in the hierarchy, h is the height of the hierarchy, n is the number of instances, d is the dimensionality.

7.2 Robustness

As mentioned in the experiments part, we keep hierarchy modification until no classification improvement could be observed on training data. However, the hierarchy might over-fit the training data as the number of iterations increases. We build hierarchical models on the training data based on the hierarchy after each iteration and test them on the testing data. We show the trend on both *Soc* and *Kids* in Figures 17 and 18, respectively. Clearly, the performance on the testing data does not necessarily improve with the iteration number increase. For most of the folds, the performance on the testing data remains unchanged or goes down after 2 iterations. This tells us that over-fitting might occur if we run our algorithm too many iterations.

Instead of stopping until no improvement on the training data can be observed, we run our algorithm just two iterations to obtain a robust hierarchy. Figure 19 compares the performance of our algorithm running multiple iterations and mere 2 iterations. On *Soc*, running our algorithm just 2 iterations results in a more robust hierarchy compared with many iterations. On *Kids*, we also obtain a hierarchy as good as the one obtained following the original algorithm.

Moreover, the computational time is also reduced sharply. As shown in Table 5, the majority of the hierarchy modifications (operations) is done after just 2 iterations. But the average number of evaluations decreases significantly. As argued in the previous section, the key issue to the time complexity of our algorithm is the number of evaluations. By reducing the number of evaluations, the computational time is significantly reduced.

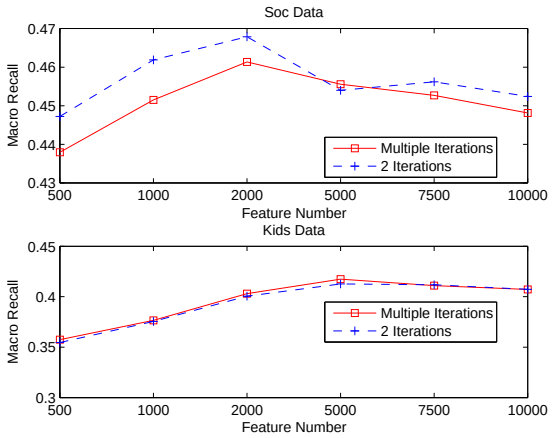

Figure 19: Multiple vs. 2 Iterations

Table 5: Efficiency Comparison

Data	Iterations	Evaluations	Operations
Soc	Multiple	539.8	48.5
Soc	2	211.8	38.5
Kids	Multiple	3343.5	197.9
Kids	2	784.9	136.3

7.3 Sensitivity to Number of Features

Feature selection can help improve classification efficiency, reduce noise in the data, and help avoid over-fitting, and is widely adopted in text classification [9, 24]. The number of features selected can have an important impact on classification accuracy. However, it is unclear about the impact of the number of selected features on the quality of a hierarchy. Hence, in our experiments, we selected a range of feature numbers to observe the performance changes with disparate settings. An interesting observation in the experimental results is that the differences in performance of the different hierarchies diminish with the increasing number of selected features. When the number of selected features is small (e.g., 500), a better hierarchy can significantly outperforms a worse hierarchy. When the number of features becomes large, performance differences reduce. This was observed in both toy and real-world data. In other words, the loss in accuracy in a bad hierarchy could be partially compensated by selecting more features.

This is because the subcategories of a good hierarchy share many features, but the subcategories of a bad one do not. For a good hierarchy, a small set of features is often sufficient to distinguish one category from another. When more features are selected, they are either redundant or irrelevant, causing potential performance deterioration. Since subcategories of a bad hierarchy do not share many terms, the increasing number of features can help better represent the parent category. An important implication is that more features should be selected for a hierarchy with lexically dissimilar subcategories than one with lexically similar subcategories. Although the loss in accuracy for a bad hierar-

chy can be partially recovered by selecting more features, it is clear that a large number of selected features results in lower classification efficiency, which is an extremely important performance metric for a content categorization service for Internet service providers.

8. CONCLUSIONS

Hierarchical models are effective for classification when we have a predefined semantically sound taxonomy. In this paper, we suggest that a given taxonomy may not necessarily lead to the best classification performance. By anatomizing some concrete examples, we illuminate why and how we could improve a semantically sound hierarchy for classification. After formulating the hierarchy search problem, we propose an effective data-driven approach to modify the given hierarchy. Experiments on the real-world data show that our data-driven algorithm can adapt a hierarchy to achieve improved classification performance. In addition, we also investigated the over-fitting problem associating with the number of iterations and how to mitigate the problem in effectively finding a robust hierarchy. The proposed approach is particularly applicable for an environment where a taxonomy is relatively stable than the changes in text data; in other words, this approach helps the taxonomy evolve by learning from data as shown in our "Hurricane" application in the introduction.

This paper is a starting point to adapt some prior hierarchy information according to the data. Much work remains to be done along this direction. For example, a "wrapper" model is used here, but a more efficient "filter" model can be tried. It is noticed that the number of features selected at each node can affect the performance and the structure of a hierarchy. When the class distribution is imbalanced, which is common in real-world applications, we should also pay attention to the problem of feature selection in order to avoid the bias associated with skewed class distribution [9, 18]. An effective criterion to select features can be explored in combination with the hierarchy information in this regard.

9. ACKNOWLEDGMENTS

Part of the work was done while the first Author was an Intern in AOL. Lei Tang is also partly supported by GPSA Research Grant of Arizona State University.

10. REFERENCES

[1] America Online Inc. http://www.aol.com/.

[2] Charu C. Aggarwal, Stephen Gates, and Philip Yu. On the merits of building categorization systems by supervised clustering. In *KDD*, pages 352–356, 1999.

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

[4] 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. *The VLDB Journal*, 7(3):163–178, 1998.

[5] Shui-Lung Chuang and Lee-Feng Chien. A practical web-based approach to generating topic hierarchy for text segments. In *CIKM*, pages 127–136, 2004.

[6] Ofer Dekel, Joseph Keshet, and Yoram Singer. Large margin hierarchical classification. In *ICML*, pages 209–216, 2004.

[7] Inderjit S. Dhillon, James Fan, and Yuqiang Guan. Efficient clustering of very large document collections. In *Data Mining for Scientific and Engineering Applications*. Kluwer Academic Publishers, 2001.

[8] Susan Dumais and Hao Chen. Hierarchical classification of web content. In *SIGIR*, 2000.

[9] George Forman. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, 3:1289–1305, 2003.

[10] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In *ICML*, pages 170–178, 1997.

[11] Tao Li and Shenghuo Zhu. Hierarchical document classification using automatically generated hierarchy. In *SDM*, 2005.

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

[13] Andrew McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *ICML*, pages 359–367, 1998.

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

[15] Joho Rousu, Craig Saunders, Sandor Szedmak, and John Shawe-Taylor. Learning hierarchical multi category text classification models. In *ICML*, 2005.

[16] Miguel E. Ruiz and Padmini Srinivasan. Hierarchical neural networks for text categorization (poster abstract). In *SIGIR*, pages 281–282, 1999.

[17] Aixin Sun and Ee-Peng Lim. Hierarchical text classification and evaluation. In *ICDM*, 2001.

[18] Lei Tang and Huan Liu. Bias analysis in text classification for highly skewed data. In *ICDM*, 2005.

[19] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*, pages 104-111, 2004.

[20] Ke Wang, Senqiang Zhou, and Shiang Chen Liew. Building hierarchical classifiers using class proximity. In *VLDB*, pages 363–374, 1999.

[21] Andreas S. Weigend, Erik D. Wiener, and Jan O. Pedersen. Exploiting hierarchy in text categorization. *Inf. Retr.*, 1(3):193–216, 1999.

[22] Wahyu Wibowo and Hugh E. Williams. Strategies for minimising errors in hierarchical web categorisation. In *CIKM*, pages 525–531, 2002.

[23] Yiming Yang, Jian Zhang, and Bryan Kisiel. A scalability analysis of classifiers in text categorization. In *SIGIR*, pages 96–103, 2003.

[24] Lei Yu and Huan Liu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):1–12, 2005.