Topic Taxonomy Adaptation for Group Profiling

Lei Tang[†], Huan Liu[†], Jianping Zhang[‡], Nitin Agarwal[†], and John J. Salerno[⋆]

A topic taxonomy is an effective representation that describes salient features of virtual groups or online communities. A topic taxonomy consists of topic nodes. Each internal node is defined by its vertical path (i.e., ancestor and child nodes) and its horizonal list of attributes (or terms). In a text-dominant environment, a topic taxonomy can be used to flexibly describe a group's interests with varying granularity. However, the stagnant nature of a taxonomy may fail to timely capture the dynamic change of group's interest. This paper addresses the problem of how to adapt a topic taxonomy to the accumulated data that reflect the change of group's interest to achieve dynamic group profiling. We first discuss the issues related to topic taxonomy. We next formulate taxonomy adaptation as an optimization problem to find the taxonomy that best fits the data. We then present a viable algorithm that can efficiently accomplish taxonomy adaptation. We conduct extensive experiments to evaluate our approach's efficacy for group profiling, compare the approach with some alternatives, and study its performance for dynamic group profiling. While pointing out various applications of taxonomy adaption, we suggest some future work that can take advantage of burgeoning Web 2.0 services for online targeted marketing, counterterrorism in connecting dots, and community tracking.

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

Additional Key Words and Phrases: Topic Taxonomy, Group Interest, Dynamic Profiling, Text Hierarchical Classification, Taxonomy Adjustment

1. INTRODUCTION

With the prolific and expanded use of Internet and increasing success of the concept of Web 2.0 (e.g., *flickr*, *del.icio.us*, *youtube*, *myspace*, *digg* and *facebook*), virtual communities and online interactions have become a vital part of human experience. Members of virtual communities[1] tend to share similar interests or topics. For example, there can be two groups browsing news at some website such as *digg.com*: one is interested in topics related to *Meteorology*, while the other in *Politics*; A blogger (say the owner of *http://hunch.net/*) who publishes blog posts actively on "machine learning" often has links on his/her blog site to other bloggers who

[1] In this work, *group* and *community* are used interchangeably.

[†] {*L.Tang, Huan.Liu, Nitin.Agarwal.2*}*@asu.edu*. Department of Computer Science and Engineering, Fulton School of Engineering, Arizona State University, Tempe, AZ 85287-8809.
[‡] *jzhang@mitre.org*. The MITRE Corporation, McLean, VA 22102.
[⋆] *John.Salerno@rl.af.mil*. Air Force Research Laboratory, AFRL/IFEA, Rome NY, 13441-4505.

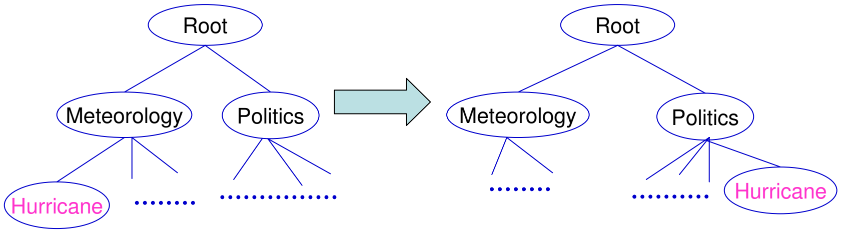


Fig. 1. “Hurricane” Example

concentrate on “machine learning” as well. It would be interesting to find these like-minded individuals for developing many promising applications including alert systems, direct marketing, group tracking, etc. One way is to profile a group, then search for additional groups that match the profile.

As group interest might change over time, a static group profile cannot keep pace with an evolving environment. In this work, we aim to address the issue of *dynamic online group profiling* in a text-dominant environment. In particular, we investigate two key issues: **(1) how to describe a group** - we study how to sensibly represent a group and what comprises a group profile; and **(2) how to track changes of group interests.** Evolving group interests present challenges to group profiling to keep up with the changes. We elaborate a viable approach that takes into account the above two issues in regards to dynamic profiling.

In a text-dominant environment, a set of topics is a sensible way of describing the interest of a group. Police might want to track a coterie with interest in topics related to “dirty bombs”, “massive destruction”, or “sabotage” to thwart crimes before they occur; a company might want to find different groups who are interested in its products (e.g., brands, functionality, or price ranges); an organization might just be interested in the opinions of various groups on the major policies (e.g., “boosting the US force presence in Iraq”), critical decisions (e.g., GM’s voluntary departure packages). Since a group consists of people with shared interests, one intuitive way of describing a group is to clip a group with some topics shared by most of the members in the group. A refined way is to associate each topic with keywords (features). These keywords can be supplied by human beings, or extracted using some feature selection methods [Forman 2003; Liu and Yu 2005].

However, the topics associated with different communities can be inordinate, and the number of relevant features to distinguish between topics can be huge. For example, the Yahoo! directory used in [Liu et al. 2005] has 292,216 categories (one category is a topic). Facing a large number of topics, we need to find a more suitable representation. Organizing the topics into a tree-structured[2] taxonomy or hierarchy is an alternative, as it provides more contextual information with refined granularity compared with a flat list. The left tree in Figure 1 shows one simple example of a topic taxonomy. Basically, each group is associated with a list of topics. Each topic can be either a non-leaf (internal) node like *Meteorology* or *Politics*, or a leaf node like *Hurricane*. Different groups can have shared topics.

[2] This structure allows one node to be the child of multiple parent nodes.

A topic taxonomy is often provided by human beings based on topic semantics or abridged from a very large taxonomy like Yahoo! or Google directory. It is a relatively stable description. However, group interests develop and change. Let us look at an example about “Hurricane”. As shown in Figure 1, in a conventional topic taxonomy, the category *Hurricane* is likely under *Meteorology*, and not related to *Politics*. Suppose we have two groups: one is interested in *Meteorology* and the other in *Politics*. The two groups have their own interests. One would not expect that “Hurricane” is one of the key topics under *Politics*. However, in a period of time in 2005, there was a surge of documents/discussions on “Hurricane” under *Politics*. Before we delve into why this happened, this example suggests *the change of group interests and the need for corresponding change of the taxonomy*. A good number of documents in category *Hurricane* are more about *Politics* because Hurricanes ‘Katrina’ and ‘Rita’ in the United States in 2005 caused unprecedented damages to life and properties; and some of the damages might be due to the responsibility and faults of FEMA[3] in preparation for and responding to the disasters.

This example demonstrates some inconsistency between a stagnant taxonomy and changing interests of an online group. Group interests might shift and the semantics of a topic could be changed due to a recent event. To enable a topic taxonomy to profile the changing group interest, we need to allow the topic taxonomy to adapt accordingly and reflect the change, which necessitates the need for *dynamic group profiling*. The dynamic changes of semantics are reflected in documents under each category, just like in the *hurricane example*. This observation motivates us to adjust a given topic taxonomy in a data-driven fashion. Figure 2 illustrates a typical process of topic taxonomy adaption. By observing the difference between the original taxonomy and the newly generated taxonomy, we notice that topics can emerge and disappear for various groups. Given recent data (e.g., blog posts, visited web pages, submitted search queries) and a given topic taxonomy, we aim to automatically find a revised taxonomy that is consistent with the data and captures dynamic group interests.

In this paper, we systematically study the effect of taxonomy on dynamic group profiling, including efficacy and efficiency. We first discuss the impact of topic taxonomies on group profiling in Section 2; formulate the taxonomy adaptation problem in Section 3; discuss about the challenges in addressing the problem and introduce two approaches to perform taxonomy adaptation: Greedy and TopDown in Section 4; present the experimental results and further study and analysis in Section 5. We review existing literature related to group profiling and taxonomy adaptation in Section 6; and discuss some future work and potential applications of our method in Section 7.

2. TOPIC TAXONOMIES IN GROUP PROFILING

A topic taxonomy is a concise representation for group profiles. Using a structural hierarchy[4] of topics to describe groups exhibits several merits:

(1) Fewer terms for representing a topic. Each node in the topic taxonomy has

[3] Federal Emergency Management Agency

[4] *Hierarchy* and *taxonomy* are used interchangeably henceforth.

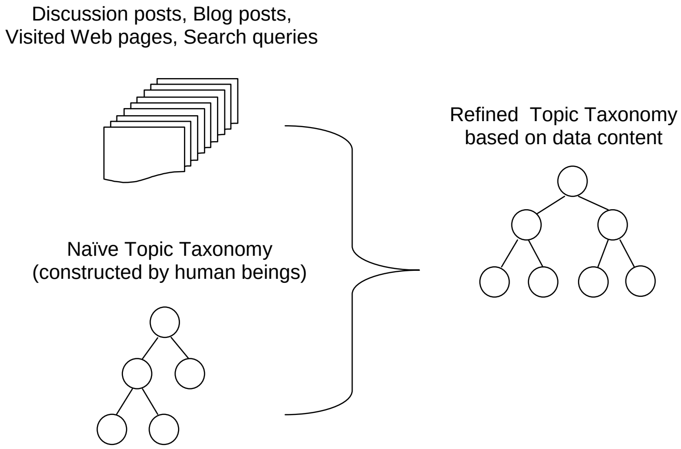

Discussion posts, Blog posts,
Visited Web pages, Search queries
Naïve Topic Taxonomy
(constructed by human beings)
Refined Topic Taxonomy
based on data content

Fig. 2. Topic Taxonomy Adaptation Process

a smaller number of sub-categories rather than a flat list of all topics. These sub-categories can be differentiate by **a small set of features**. The sets of reduced features shed light to utilize more complex models for profiling, without encountering many of the standard computational and robustness difficulties [Koller and Sahami 1997] in the context of classification. The literature also confirms that hierarchical models (which utilize the structure of the taxonomy) often outperform flat models (which perform classification without taxonomy) in training efficiency, classification efficiency, and accuracy [Koller and Sahami 1997; McCallum et al. 1998; Ruiz and Srinivasan 1999; Dumais and Chen 2000; Cai and Hofmann 2004; Yang et al. 2003; Liu et al. 2005].

(2) Concise representations of adjustable granularity. Some groups might be interested in "sports", while some other groups might be interested in more specific topics such as "football", "basketball", or "baseball". Using a flat representation would mix up all these topics since they are overlapped with "sports". Taxonomies, on the other hand, can flexibly provide topics with varied granularity to serve different needs of various groups.

(3) Rich contextual information. Within a taxonomy, each topic is an internal or leaf node in a path originating from the root node. This path suggests the context of a topic, providing more detailed information than a flat list of topics. Each node is further described by a set of features (terms) providing additional semantic information. Given a topic taxonomy, it is easy to find related or similar topics via parent, sibling, child nodes. Taxonomies also facilitate the visualization of relationships between different groups and the detection of related or similar groups.

The core problem now is how to find a *good* taxonomy, which means that it can accurately represent a group profile. Several ways can be exploited to find the profile for each group. Given some labeled training data, for example, a classifier can be constructed. This training data can either be provided by human experts, or derived from the tags associated with data gleaned from the Web, if such information is available. With a robust classifier built from the available data, new documents can be labeled automatically by the classifier. Therefore, the corresponding classification performance is one effective way of *indirectly* measuring how good a topic taxonomy is in group profiling. In other words, the quality of a topic taxonomy now boils down to the classification performance (e.g. recall, precision, ROC, etc.) based on the taxonomy.

A good taxonomy can be obtained via different methods:

1) Extracted from a general grand taxonomy like Yahoo! or Google directory,
2) Provided by human experts, or
3) Generated via hierarchical clustering on topics.

The taxonomy provided by the above methods are relatively stable, and cannot scale up to capture the dynamic change of group interests.

Given the dynamic group profiling problem, we notice the following challenges that should be addressed in search of a suitable method to find a good taxonomy:

- *Dynamic.* The method must adaptively find a topic taxonomy to reflect the dynamic change in the data.
- *Accurate.* The obtained taxonomy must provide an accurate profile for each group. Since each group is profiled using topics and keywords associated with each topic, precise profiling necessitates accurate hierarchical document classification.
- *Efficient.* The method proposed must be efficient in adapting a taxonomy to keep pace with the prolific growth of online documents. The method should scale well to handle large number of documents as well as topics.
- *Automatic.* It is desirable for the method to minimize human involvement in this process, achieving efficiency and efficacy.

Clearly, methods 1–3 cannot serve the need outlined above. We propose *topic taxonomy adaptation* in this work to attain a good taxonomy. In practice, a semantics-based taxonomy can be provided as a seed through method 1 or 2. The provided taxonomy can be considered as a form of prior knowledge and contains valuable information. With this prior knowledge, we can narrow down the hypothesis space and efficiently find reliable hierarchies with good classification performance and generalizability. Instead of "start-from-scratch" as of method 3, we propose to modify a given taxonomy gradually and generate a data-driven taxonomy, so as to achieve classification improvement for accurate dynamic group profiling.

The **topic taxonomy adaptation** problem can be rephrased as follows: *Given a taxonomy, find a refined taxonomy such that an accurate hierarchical classification model can be induced for dynamic group profiling.*

3. TAXONOMY ADAPTATION

For dynamic group profiling, the basic problem is how to find a refined taxonomy to effectively capture the characteristics of online groups given a taxonomy. We assume that leaf-level topics are always there for simplicity. This could be done by including a large variety of topics. But the topics of internal nodes in a taxonomy could emerge and disappear as new documents arrive. Before we formulate our problem, we present several definitions concerning hierarchies as follows:

Definition 3.1 *Admissible Hierarchy.* Let $L = \{L_1, L_2, \cdots, L_m\}$ denotes the categories at the leaf nodes of a taxonomy H, and $C = \{C_1, C_2, \cdots, C_n\}$ denotes 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.

Definition 3.2 *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 finding 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.

Suppose there are n leaf nodes, one approach to construct a taxonomy is: pick two categories to form a new parent node; then merge this parent node with a new leaf node to form another new parent node; continue this process until no leaf nodes are left. Then we end up with a highly unbalanced binary tree. Clearly, the final taxonomy structure depends on the order of picking leaf nodes. Hence, we could have $O(n \times (n-1) \times \cdots \times 1) = O(n!)$ different hierarchies. Note that this is only one strategy to construct a binary tree and many other admissible binary trees are not considered yet. Not to mention those n-ary trees. Actually, this problem is highly related to Steiner tree problem [Hwang and Richards 1992] which is proved to be NP-complete. It is impractical to try all the possible hierarchies and pick the optimal one. A more effective way should be explored.

The given hierarchy provides valuable information for classification and can serve as a seed to find the intended optimal hierarchy. In order to change a hierarchy to another admissible hierarchy, we define three elementary operations:

Promote: roll up one node to upper level,

Demote: push down one node to its sibling, and

Merge: merge two sibling nodes to form a super node.

As shown in Figure 3, 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 (the super node) after modification. Note that the set of leaf nodes remains unchanged.

THEOREM 3.3. *The elementary operations are complete for hierarchy transformation.*

In other words, we can transform one hierarchy H to any other admissible hierarchy H' by using just the above three operations. The proof is trivial as we can transform H to a 1-level tree by promoting all the nodes to its upper level until it reaches the first level. Then, according to the structure of H' , merging and demoting can be applied to construct the hierarchy.

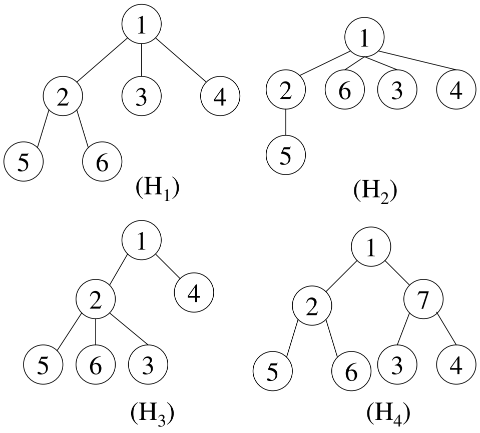


Fig. 3. Elementary Operations. H_1 is the original hierarchy. H_2, H_3 and H_4 are obtained by performing different elementary operations. H_2: Promote node 6; H_3: Demote node 3 under node 2; H_4: Merge node 3 and node 4.

Definition 3.4 *Hierarchy Difference.* Hierarchy difference between two admissible hierarchies H and H' is the minimum number of elementary operations to transform H into H' . Suppose the minimum number of operations is k, we denote the difference between H and H' as

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

Hierarchy difference actually represents the minimum edit distance of two hierarchies in terms of our defined elementary operations. Given explicit hierarchy difference, we define the constrained optimal hierarchy below.

Definition 3.5 *Constrained Optimal Hierarchy.* Given a hierarchy H_0, if there exists a sequence of admissible hierarchies $Q = \{H_1, H_2, \cdots, H_n\}$ such that their conditional probabilities satisfy the following

$$\begin{aligned} P(D|H_i) &\geq P(D|H_{i-1}) \\ \| H_i - H_{i-1} \| &= 1 \qquad (1 \leq i \leq n) \\ \forall H' \text{ if } \| H' - H_n \| &= 1, \quad P(D|H') \leq P(D|H_n) \end{aligned}$$

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

In other words, the constrained optimal hierarchy (COH) is the hierarchy that is attainable from the original hierarchy following a list of admissible hierarchies with likelihood increase between consecutive ones. When we reach a COH, we cannot find a neighboring hierarchy with higher likelihood than it. By its definition, each COH is a local optimum. If we state our problem as that of search, then a provided 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, find a hierarchy H_{opt} such that*

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

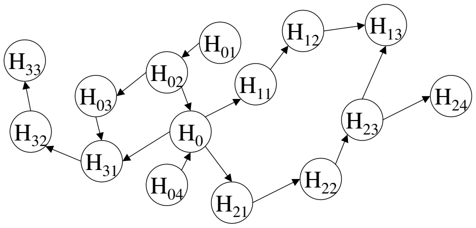


Fig. 4. Hierarchy Search Space

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

Put it another way, we can consider hierarchy search problem as searching in the hierarchy space as in Figure 4. All the hierarchies in the figure are admissible for some data, and an arrow from H_i to H_j denotes likelihood increase if we transform H_i to H_j by just one hierarchy adjustment elementary operation. If there is no link between two nodes (hierarchies), then one hierarchy cannot be transformed to the other by just one operation. For the given hierarchy H_0, there are three constrained optimal hierarchies, H_{13}, H_{24} and H_{33}. Notice that there are actually two paths leading to H_{13}. And two constrained optimal hierarchies (H_{13} and H_{24}) might share partial search path (H_0 to H_{23}). As the topic changes during group profiling are often not many or mostly local, the optimal hierarchy is expected to reside within the vicinity of a given hierarchy. The optimal hierarchy should be one of the constrained optimal hierarchies. As shown in Figure 4, the optimal hierarchy should be chosen among H_{13}, H_{24} and H_{33} as they yield the maximal likelihood.

4. CHALLENGES AND SOLUTIONS

4.1 Challenges

As for the hierarchy search problem, we need to address the following subproblems:

1) How to compute the likelihood of data given a hierarchy ($P(D|H)$ in Definition 3.2)?

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 promising 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. Suppose the average number of branching factor and the total number of nodes of the hierarchy are b and n, respectively. For each node, there are three kinds of operations: promote to its parent level; merge with a sibling node; demote to a child of a sibling node. Thus, the total number of neighbors is $O((2(b-1)+1)\times n) = O(2bn)$. Among all these neighbors, most of them are not necessarily better than current one. It is desirable to identify those promising neighbors only.

Hence, we propose to obtain an approximate solution by developing some heuristics. As for the first subproblem, we actually want to use it to compare two given hierarchies. Since the topic identification performance indirectly indicates how effective our profiling is, we approximate it by comparing two hierarchies' classification performance estimates. As in most classification tasks, the class distribution is highly imbalanced, accuracy would be biased toward the majority class. Researchers focus on macro-averaged recall (also known as balanced accuracy) or f-measure [Yang and Pedersen 1997; Liu et al. 2005] rather than accuracy. Here, we use them as the classification performance to measure likelihood change.

Concerning the second problem, we exploit a greedy approach to find the best constrained hierarchy. In each search step, we always choose the neighboring node with largest likelihood improvement. Other variants of search methods like beam search can also be explored if time and hardware resources permit. However, we still need to consider the number of neighbors of a hierarchy. Based on some pathology study in [Tang et al. 2006], we can apply certain heuristics to find those promising neighbors and remove those non-promising from further consideration. In this section, we present some heuristics and then provide algorithms that accustom the given taxonomy according to the data.

4.2 A Greedy Approach

We first give some definitions to facilitate the description of the heuristics.

Definition 4.1 *High Miss/Low Miss.* For a node in the hierarchy, if it is misclassified at the parent level, then this misclassification is called High Miss. If it is misclassified as its sibling under the same parent node, then it's a Low Miss.

Heuristic 4.2. If the proportion of High Miss of one node is significantly larger than that of the Low Miss, that is,

$$High\,Miss > Low\,Miss + \xi$$

where ξ is a user defined parameter, we lift this node to the upper level.

Basically, if a node is misclassified a lot at parent level, then we'll consider lifting it up to obtain better result.

Definition 4.3 *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 $= P_{AB} + P_{BA}$.

Heuristic 4.4. We can calculate the Ambiguity Score for each pair of categories under the same parent node. For each subtree in the hierarchy, we pick the sibling pair A and B with highest ambiguity score. If $|P_{AB} - P_{BA}| \leq \gamma$ where γ is a predefined threshold, then we merge A and B to form a super category; Otherwise, if $P_{AB} > P_{BA} + \gamma$, we shift Class A as B's child; If $P_{BA} > P_{AB} + \gamma$, then we move Class B under Class A.

Intuitively, the ambiguity score is the overlapping area of two categories. Hence, it can help identify the most similar two categories. In the heuristic, we can find the dominant class by comparing P_{AB} and P_{BA}, and then demote one class as the other class's sub-category. Otherwise, neither of them dominates the other and so they are merged to form a super category.

Based on these heuristics, the search space of hierarchies is significantly reduced. We can now use a wrapper model to search for better hierarchies. That is, for a given

```
Input:
    $H_0$: Predefined hierarchy
    $T$: Training data
    $V$: Validation set
Output:
    $H_{opt}$: the approximate best hierarchy
Algorithm:
1. $score_{best} = 0$, $H_{list} = \{H_0\}$
2. $flag = false$ (denoting whether or not the hierarchy is changed)
3. For each hierarchy $H_i$ in $H_{list}$, build a hierarchical model based on $T$ and evaluate its
   performance on $V$. If the corresponding statistic $score$ is larger than $score_{best}$, then
   $flag = true$, $score_{best} = score$ and $H_{opt} = H_i$.
4. If $flag == false$, return $H_{opt}$.
5. Generate neighbors for $H_{opt}$ by checking each node in $H_{opt}$ according to Heuristic 1
   and 2. Add all these neighbors to $H_{list}$.
6. If $H_{list}$ is empty, return $H_{opt}$; Otherwise, goto step 2.
```

Fig. 5. Greedy Hierarchy Search Algorithm

hierarchy, we generate promising neighboring hierarchies and evaluate the hierarchy on some data to get its performance statistics. The hierarchy with maximum likelihood increase is thus selected. This procedure repeats until no neighboring hierarchy with likelihood increase could be found. The Greedy Hierarchy Search Algorithm is given in Figure 5.

4.3 A Top-Down Approach

We noticed in our experiments (see Section 5) that the Greedy Hierarchy Search Algorithm, though effective, did plenty of redundant work in each step to search for neighboring hierarchies. Actually, two neighboring hierarchies would share most operations to find their neighbors. In other words, if one operation results in an improvement for current hierarchy, it's likely to yield improvement on a neighboring hierarchy as well. Therefore, it is not necessary to check all the operations in each step. Instead, we propose to traverse the hierarchy using a top-down approach and check each node to search for better hierarchies.

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 shallowest nodes first in search of promising nodes to expand.

Our top-down approach (TopDown) consists of multiple iterations (Figure 6). For each search iteration, we have the following procedures:

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

We discuss each procedure below.

```
Input:
    $H_0$: Predefined hierarchy
    $T$: Training data
    $V$: Validation set
    $\delta$: Stopping criterion
Output:
    $H_{best}$: the approximate best hierarchy
Algorithm:
    $S_{pre} = 0$; $H_{best} = H_0$;
    $S_{best}$=evaluateHierarch($H_0$,$M$,$V$);
    $O_{flag} = false$;
    while ($S_{best} - S_{pre} > \delta$)
      $N_{list}$={all nodes in $H_{best}$};
      repeat
        $Node$=getNodeToCheck($N_{list}$);
        $H_{list}$=generateNeighbors($Node$,$O_{flag}$);
        [$H$, $S$]=findBest($H_{list}$);
        if $S > S_{best}$
          $S_{pre} = S_{best}$; $S_{best} = S$; $H_{best} = H$;
          updateNodeList($N_{list}$, $H_{best}$, $Node$);
        end
      until $N_{list} == null$ ;
      $O_{flag} = \neg O_{flag}$;
    end
    return $H_{best}$
```

Fig. 6. Top-Down Hierarchy Search Algorithm

4.3.1 *Identification of the node to check.* Clearly, the nodes at the upper level affect more in the classification process and should be considered with higher priority. Therefore, we maintain a list of nodes in the hierarchy. At each iteration, we pop the node with the shallowest depth and remove it from the list to avoid future consideration (refer to Figure 7 **getNodeToCheck** for details).

4.3.2 *Identification of promising neighbors.* Since the number of neighbors of one hierarchy could be huge, 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. Three elementary operations have different priorities. In order to sever the wrong parent-child relations, we need to first promote the node. 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 on 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. Therefore, for one node, we just pick the sibling node with highest ambiguity score and generate possible good neighbors by merging these two nodes or by demoting one node to

```
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 demoting $N$ 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 $N_{list}$;
```

Fig. 7. Procedure definitions

the other. Notice that not all the neighboring 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 **generateNeighbors** is in Figure 7.

4.3.3 *Identification of the best neighbor.* This procedure compares all the promising neighboring hierarchies and find the best one among them. Given a list of hierarchies, we just build a hierarchical model based on each hierarchy, and then evaluate it on the validation data to obtain some classification statistics (in particular, macro-averaged recall in our work). The best hierarchy and the corresponding statistics are returned (Line 9 in Figure 6).

4.3.4 *Update of current best hierarchy.* After we obtain the best hierarchy in the neighbor list, we could compare it with the current best hierarchy. If the classification statistic is better than the current one, we replace the current best hierarchy with the best hierarchy just found and update the list of nodes to check. Otherwise, the hierarchy remains unchanged, and we continue with the next node (Lines 10-13 in Figure 6).

Each time we change the hierarchy, we have to update the list of nodes to check (refer to **updateNodeList** in Figure 7). We actually just push to the list all the nodes that will be affected by the operation. Suppose N is the node being checked. If the hierarchy is obtained by promoting, all the children of N's grandparent should be rechecked. We can revisit the cases in Figure 3. 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 remains 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. Furthermore, as we use top-down approach to traverse the tree, whenever there's a change at one node, its children will not be affected. This avoids unnecessary checking of nodes.

The detailed algorithm is presented in Figure 6. In summary, 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. Since promoting should perform first, in Figure 6, 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 δ.

The major difference of TopDown and Greedy approaches is efficiency. As for the Greedy approach, we have to check all the possible operations to all the nodes, whereas TopDown considers only one node in each search step while traversing the possible neighboring hierarchies. The efficiency difference will be reported in the experiment part.

5. EXPERIMENTS AND ANALYSIS

Since the classification performance indicates the efficacy of a taxonomy for group profiling, here we use classification performance as a quality measure of a topic taxonomy. We conduct experiments on some real-world data sets to show the effectiveness of the proposed algorithms. These data sets are provided by an Internet company. One is about the topics of social study (*Soc*) shared by many small groups; the other focuses on children's interests (*Kids*). Topics in both data sets

Table I. Real-World Data Description

	Soc	*Kids*
#leaf-level topics	69	244
#nodes in topic taxonomy	83	299
Height of topic taxonomy	4	5
#instances	5248	15795
#terms	34003	48115

are organized into corresponding taxonomies. Text and meta information is extracted from web pages. After removing common stop words, a vector space model is applied to represent web pages. Table I summarizes the information about the two data sets. These two data sets contain a large number of categories and the class distribution is highly imbalanced as observed in Figure 8. Therefore, accuracy is not a good evaluation measure as it is biased toward the major class [Tang and Liu 2005]. Instead, we use macro-averaged recall and F-measure as our evaluation measure.

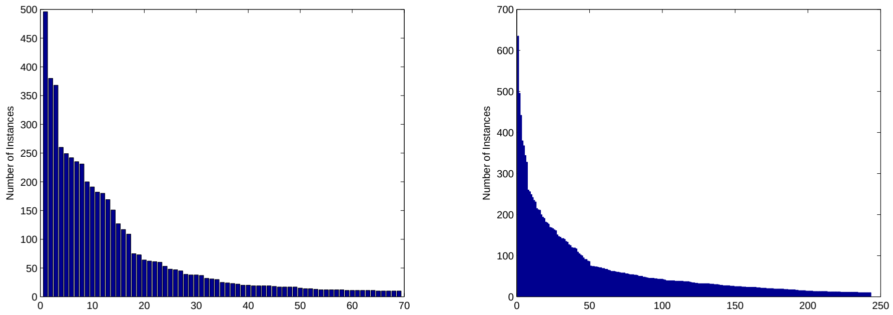
5.1 Experiment Settings

We perform 10-fold cross validation to both data sets. In each fold, we apply both Greedy approach and TopDown approach 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 numbers of features at each node. The model is then evaluated on the test data. The average results in terms of macro recall and macro F-measure are reported.

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 [Yang and Pedersen 1997] to build the hierarchical model. To gain efficiency, the classifier at each node we exploited is multi-class multinomial naïve Bayes classifier [McCallum and Nigam 1998]. The data fragmentation problem becomes serious with a large number of categories. For instance, some categories in *Soc* data have fewer than 10 instances. Keeping a portion of training data as the validation set makes the learning unstable and might lose generalization capability. Here, we set the validation set the same as the training data to guide the hierarchy modification. Independent validation sets can be a better option if sufficient training data is available. The stopping criterion for hierarchy adaptation is until no classification performance can be improved on the training data[5]. By some empirical pilot study, we set ξ in Heuristic 4.2 to 0 and γ in Heuristic 4.4 to 0.01. Cross validation can be exploited here to set the parameters.

In order to examine if a predefined semantics-based taxonomy can provide useful prior knowledge for search, we also compared with the "start from scratch" approach: ignore the predefined taxonomy and do hierarchical clustering on training data to obtain the taxonomy. We did a preliminary study to compare a divisive clustering approach in [Punera et al. 2005] with an agglomerative clustering algorithm in [Chuang and Chien 2004] (discussed in Section 6.2), and found that the latter (HAC+P) is not comparable to the former for our application. The difficulty

[5] The overfitting problem with this setting is studied later.



(a) *Soc* Data

(b) *Kids* Data

Fig. 8. Class Distribution

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.

5.2 Performance on Real-World Data

Figures 9 and 10 demonstrate the performance of different methods plus the standard deviation. The curves of "Clustering" and "Original" denote the performance of the clustering approach and that based on the original hierarchies, respectively. 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.

Comparatively, our algorithms, which start from a given hierarchy, achieve significant improvement over the original taxonomy on both data sets. This is more obvious when the number of categories is large whereas the features being selected are few. Both TopDown and Greedy approach are comparable and can automatically adjust the content taxonomies for more accurate classifiers. There is no significant difference between the two in terms of classification performance.

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 (Figure 9). 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 difference diminishes. 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

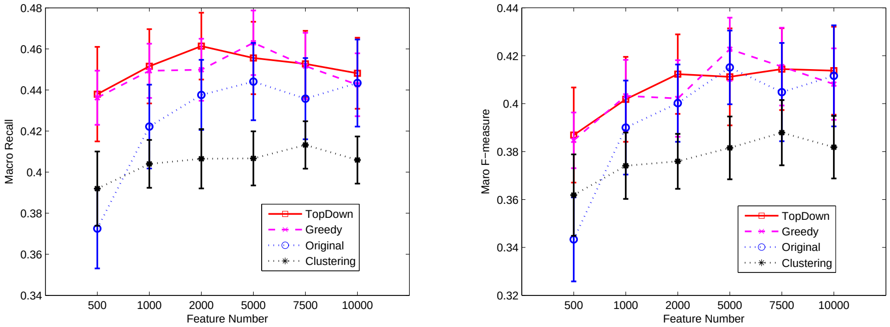


Fig. 9. Performance on *Soc* Data

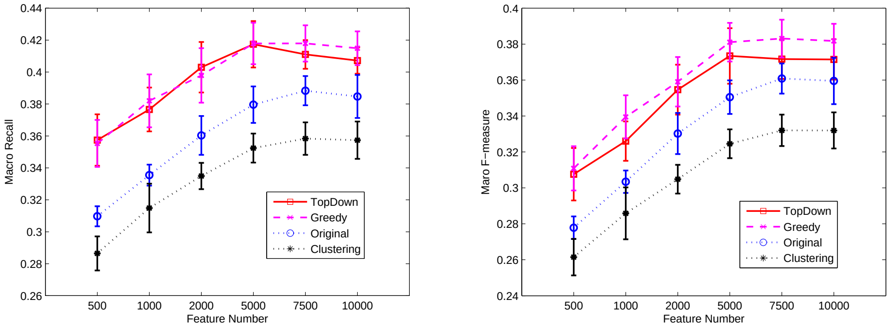


Fig. 10. Performance on *Kids* Data

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. However, when the taxonomy size is large, changing the taxonomy is more effective for performance improvement than selecting more features (Figure 10 and later on the Google directory benchmark data shown in Figure 14).

5.3 Greedy vs. TopDown Approach

Though no significant classification difference is observed between Greedy and TopDown approaches, the time complexity of the two differs. For naïve Bayes classifier, both the training time and test time are linear in terms of the number of instances and dimensionality. For each category, we could summarize the statistics of terms given the category using just one vector. Then, building a hierarchical model just costs $O(c_i d)$ where c_i 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. So the total number of evaluations determines the computational cost of our algorithms.

Tables II and III present the total number of evaluations for each method. When the number of nodes in the hierarchy varies from 83 in *Soc* to 299 in *Kids*, that is, an increase by around 4 times, the number of hierarchy evaluations for the greedy approach multiplies by around $\frac{10923.6}{616.9} \approx 18$ times. But for Top-down approach,

Table II. Greedy Performance Statistics

Dataset	Evaluations	Operations	Candidates
Soc	616.9 ± 241.9	18.9 ± 6.7	32.3 ± 1.8
Kids	10923.6 ± 2098.9	64.2 ± 13.0	170.3 ± 4.1

Table III. TopDown Performance Statistics

Dataset	Evaluations	Operations	Iterations
Soc	539.8 ± 191.9	48.5 ± 12.6	5.6 ± 1.8
Kids	3343.5 ± 665.1	197.9 ± 26.5	9.7 ± 1.6

the factor is $\frac{3343.5}{539.8} \approx 6$ times. This huge difference can also be derived from the following theoretical analysis.

For the **Greedy approach**, at each search iteration, the number of hierarchy neighbors is $O(c)$ where c is the number of nodes in the tree. If we finally perform p operations, the number of evaluations is $O(cp)$. The time complexity of Greedy approach is $O(cp \cdot hnd)$. Table II shows the average number of evaluations, operations, and hierarchy candidates of each iteration on *Soc* and *Kids* data. As the number of nodes in the hierarchy increases, both operations to reach a local optimum and the average number of candidates rises dramatically, which is approximately proportional to the number of the nodes in the hierarchy. Hence, the greedy approach runs approximately $O(c^2 \cdot hnd)$ in time.

For the **TopDown approach**, Tables III exhibits some statistics: the number of iterations, evaluations and elementary operations. Differently, the number of candidates is not presented as this algorithm generates at most 3 candidates in each search step. Let c denote the number of nodes in the hierarchy, then a node can never be checked more than c times in one iteration. In the worst case, 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 for one iteration is $O(c^2 \cdot hnd)$.

However, the bound above is loose. As we traverse the taxonomy top down and all the hierarchy changes are local, the worst case can seldom happen on a semantically reasonable hierarchy. In reality, we observe that on average, a node will be checked no more than twice in one iteration. As shown in Table III, 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, 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 $O(2chnd) = O(chnd)$. In practice, the number of iterations is bounded by a small constant I. We show in Sec. 5.4 that $I = 2$ is a good choice for TopDown. Hence, the total time complexity of our algorithm is $O(Ic \cdot hnd)$, i.e., linear.

5.4 Robustness

In our original TopDown approach, we keep modifying hierarchy until no classification improvement could be observed on training data. However, it is unclear whether the final hierarchy might over-fit the training data. Thus, we build hierarchical classification models on the training data based on the hierarchy after each iteration in TopDown and test them on the test data. We show the trend

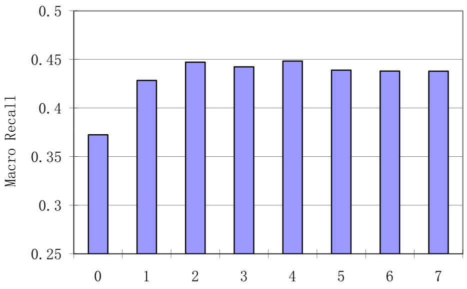


Fig. 11. Over-fitting on *Soc*

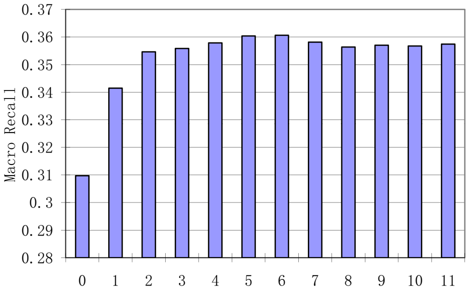


Fig. 12. Over-fitting on *Kids*

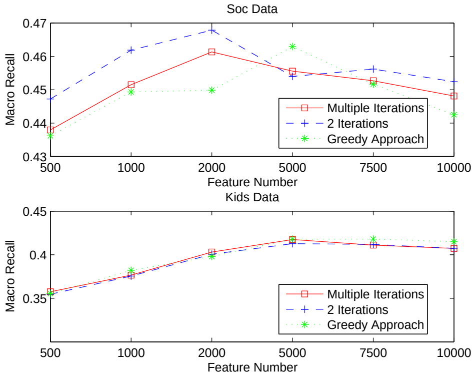


Fig. 13. Multiple vs. 2 Iterations

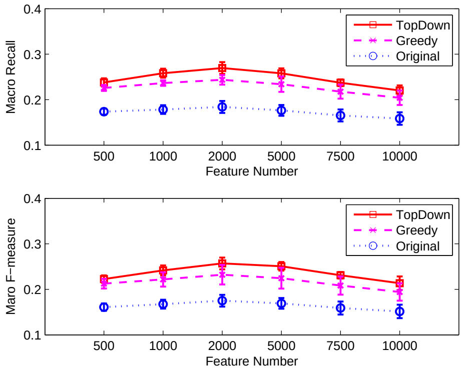


Fig. 14. Performance on Google Directory

Table IV. Efficiency Comparison

Data	Iterations	Evaluations	Operations
Soc	2	211.8 ± 18.3	38.5 ± 6.9
	Multiple (5.6±1.8)	539.8 ± 191.9	48.5 ± 12.6
Kids	2	784.9 ± 28.0	136.3 ± 11.7
	Multiple (9.7±1.6)	3343.5 ± 665.1	197.9 ± 26.5

in terms of average performance on both *Soc* and *Kids* in Figures 11 and 12, respectively. Iteration 0 denotes the performance of the original hierarchy. Clearly, the performance on the testing data does not necessarily improve as the iteration number increases. *Soc* and *Kids* achieve the maximum after 2 and 5 iterations, respectively. The largest jump between consecutive iterations occurs at the first 2 iterations. Then, the performance stabilizes for both cases.

Based on this observation, we suggest for TopDown to iterate twice to save computational cost and obtain a robust taxonomy. Notice that the number of iterations in TopDown varies depending each fold (as seen in Table III). Figure 13 compares the performance of our algorithm with multiple iterations and mere two 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 TopDown algorithm.

Meanwhile, the computational time is reduced sharply. The average number of evaluations and operations are shown in Table IV. 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.

The time complexity difference between TopDown with 2 iterations and Greedy is more easily observed when the taxonomy size is large. To verify this, a partial taxonomy of the Google directory is selected as a benchmark data set. We select a partial taxonomy from category *computers*, remove those categories with too few documents and finally obtain a taxonomy with 978 leaf nodes (categories) and in total 1207 nodes (including internal nodes) with 31197 documents.

We applied our proposed two approaches (Greedy and TopDown with only 2 iterations) to the data set. Unfortunately, the Greedy approach is still computationally too expensive for such a large data set to get a final solution. Thus, instead of letting the Greedy method "run forever", we interrupt it when Greedy runs twice the time as TopDown does, and the obtained hierarchy is then used as the Greedy's taxonomy. Figure 14 demonstrates the average result of 10-fold cross validation. Clearly, TopDown with 2 iterations is more accurate and efficient than Greedy. Note that the number of categories is very large here (978 classes). Hence, a tiny numerical improvement is indeed significant with respect to a large number of categories. This is also indicated by small standard deviations in the figure.

5.5 Dynamic Change of Taxonomies

In the previous experiments, we have shown that taxonomy adaptation can help to improve accuracy for topic identification. The content change in new incoming data is detected by our method to adapt the taxonomy to reflect the change. Since the taxonomy for real-world applications is so large, it is cumbersome to be included for illustration. In addition, the taxonomy evolvement is usually slow and extensive human efforts are required to verify taxonomy adaptation due to the changes of a very large-scale data. An alternative to verify taxonomy adaption is to perform a controlled experiment in which we know *a priori* the obvious content changes in the data and observe how a taxonomy adapts to the changing data. This controlled experiment can illustrate clearly the effect of dynamic changes of taxonomies.

To prepare for the controlled experiment, we crawled 1800 Web pages with 8 categories from a publicly available web site. The 8 categories are organized into a semantic-sound hierarchy in Figure 15(a) as the initial taxonomy. The data set is split into three folds (Folds 1, 2, and 3) to represent the snapshots collected at different time stamps. We then switch the content of class *Movies* with that of *Politics* in Fold 2. This way, we force the obvious change to happen and see if the taxonomy can adapt to the change. When Fold 1 is presented, the taxonomy of Figure 15(a) is changed to the one in Figure 15(b). Then Fold 2 is presented, as the contents of *Movies* and *Politics* are switched in this fold, the taxonomy is adapted to that in Figure 15(c). Notice that the position of *Movie* and *Politics* are swapped in the new taxonomy, and the taxonomy adapts to the change. *Movies* and *Economics* now belong to the same parent node indicating the similarity in their contents. Similarly, *Politics* becomes a sibling node of *Music* as expected. When Fold 3, in which the content is consistent with the category labels as in Fold 1, is presented, the taxonomy changes again to that in Figure 15(d) to reflect the change

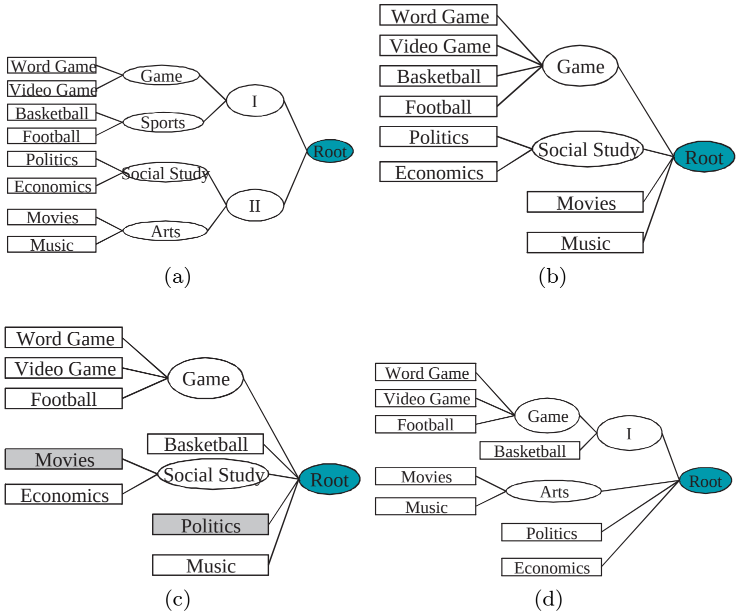


Fig. 15. Dynamic Changes of Taxonomies

in data from Fold 2 to Fold 3. *Movies* and *Music* are coupled again, and *Politics* and *Economics* are siblings. Clearly, the content changes in data are reflected in the corresponding taxonomies. We notice that *Sports* and *Games* are somehow mixed in all the three taxonomies. This could be attributed to the variance of each data fold.

6. RELATED WORK

Group profiling has been studied extensively in terms of customer relationship modeling [Bounsaythip and Rinta-Runsala 2001; Adomavicius and Tuzhilin 2001; Shaw et al. 2001; Chen et al. 2005]. In those works, a typical process is to apply an association rule algorithm [Agrawal et al. 1993] to mine interesting patterns from customer transactions. Based on the customer segmentation (group), some interesting patterns can be found in this group for future marketing. Our work adopts a different process than the typical customer profiling. Focusing on online groups such as Blogosphere or online Internet surfing activities, we adopt topic taxonomy to profile groups instead of potential patterns shared by customer transactions. The data we collect is mainly topics/tags and documents instead of customer information and transaction records. Our profiles actually act like concise summaries for individual online groups.

We propose taxonomy adaptation to achieve dynamic group profiling. In this process, a hierarchical classification model is employed, and a better taxonomy is attained after adaptation given a provided taxonomy. Thus, we briefly survey the work on state-of-the-art hierarchical classification and taxonomy generation.

6.1 Hierarchical Classification

A topic 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 (e.g., the score should be larger than a threshold [Dumais and Chen 2000] or the predictions of adjacent levels should be consistent [Wibowo and Williams 2002]). Feature selection is often performed at each node before constructing a classifier [Chakrabarti et al. 1998; Liu and Motoda 2007].

To build the hierarchical model, different base classifiers are employed including association rules [Wang et al. 1999], naïve Bayes classifiers [Koller and Sahami 1997], neural networks [Weigend et al. 1999; Ruiz and Srinivasan 1999], and support vector machines [Dumais and Chen 2000; Sun and Lim 2001; Liu et al. 2005]. 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 [Dumais and Chen 2000], the authors use a sigmoid function to map the prediction of a 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 a hierarchical model marginally outperforms a flat(non-hierarchical) model. And these two methods show little difference. In [Koller and Sahami 1997], a greedy approach with naïve Bayes classifiers is exploited and a significant accuracy improvement is observed.

One advantage of the hierarchy-based approach is its efficiency in training and testing, especially for a very large taxonomy [Yang et al. 2003; Liu et al. 2005]. Hierarchical models make it easy to modify and expand a taxonomy, like add one sub-category, delete one category, or merge several categories 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. So 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 [Dekel et al. 2004; Cesa-Bianchi et al. 2006b; Tsochantaridis et al. 2004; Cai and Hofmann 2004; Rousu et al. 2005; Cesa-Bianchi et al. 2006a]. The main idea behind these methods is to map the document features or document-label features to a high-dimensional space so that a defined margin can be maximized. Variegated loss functions (misclassification costs) are obtained from the hierarchy. This loss function is incorporated into the margin formulation and then some tricks (variable/constraint selection, maintaining a working set, incremental conditional gradient ascent) are used for optimization. In [Cesa-Bianchi et al. 2006b; Tsochantaridis et al. 2004; Cai and Hofmann 2004; Rousu et al. 2005], the output space is a sequence of categories rather than just a label. All the possible paths from the root to leaves in the hierarchy are considered during training and the goal is to find an optimal sequence which maximizes the margin. A concomitant of these methods' superior performance is their unbearable computational cost for training. There are some other methods which use hierarchies for statistical smoothing and require

EM or cross validation to tune the parameters [McCallum et al. 1998; Toutanova et al. 2001; Veeramachaneni et al. 2005].

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, especially for dynamic group profiling of which topics might drift. This partly motivates us to propose our methods for taxonomy adaptation.

6.2 Taxonomy Generation via Clustering

Some researchers propose to generate a taxonomy from data for document management and classification. However, human beings are sometimes involved to aid the construction of taxonomies [Zhang et al. 2004; Gates et al. 2005], making it rather complicated to evaluate. Here, we concentrate on those methods constructing taxonomies automatically.

There are two directions for hierarchical clustering: agglomerative and divisive. In [Aggarwal et al. 1999; Chuang and Chien 2004; Li and Zhu 2005], all employ a hierarchical agglomerative clustering (HAC) approach. In [Aggarwal et al. 1999], 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 [Li and Zhu 2005], a linear discriminant projection is applied to the data first and then a hierarchical clustering method UPGMA [Jain and Dubes 1988] 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 hierarchical models yield classification improvement over flat models. But it is not sufficiently justified why a two-level tree should be adopted. Meanwhile, [Chuang and Chien 2004] proposes HAC+P which is similar to the previous approach. Essentially, it adds one post-processing step to automatically change the binary tree obtained from HAC, to a wide tree with multiple children.

Comparatively, the work in [Punera et al. 2005] 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 [Dhillon et al. 2001] 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.

Some practitioners adopt the Bayesian approach to build a topic taxonomy for text documents. The Cluster-Abstraction Model proposed in [Hofmann 1999], associates word distribution conditioned on classes for each node. The author uses a variance of EM algorithm to do clustering. Similarly, Probabilistic Abstraction Hierarchies presented in [Segal et al. 2001] also associates a class-specific probabilistic model (CPM) to each node and use KL divergence to define the distance of categories. Then a hierarchy which minimizes the overall distance and maximizes the likelihood is presented. In [Blei et al. 2003], the nested Chinese restaurant process is introduced as a prior for hierarchical extension to the latent Dirichlet allocation model [Blei et al. 2003]. Some recent works [Blei and Lafferty 2006; Chakrabarti et al. 2006; Airoldi et al. 2006] extend the clustering method to take into consideration the dynamic change of topics in evolving data as well, but mostly focus on a flat list of topics without taxonomy.

Most clustering approaches justify their taxonomies based on semantics. Semantically sound taxonomy may not necessarily result in good classification performance [Tang et al. 2006]. In addition, the update of a taxonomy based on clustering is not efficient. The clustering algorithm has to rerun from scratch each time when new data is collected. Our approach adapts a taxonomy automatically thus avoids unnecessary, repeated work.

7. CONCLUSIONS AND FUTURE WORK

In order to dynamically profile various online groups and communities for other tasks and potential applications, we propose a topic-taxonomy based profiling, as it provides more contextual information with varied granularity yet requires fewer terms to represent each group. However, a stable taxonomy fails to capture a group's interest shift reflected in changing data. Taxonomy adaptation is proposed to allow a taxonomy to keep up with the evolving data.

We propose two effective data-driven approaches to modify a given taxonomy: Greedy and TopDown. Experiments on the real-world data show that both algorithms can adapt a hierarchy to achieve improved classification performance. No significant difference in classification performance is observed between Greedy and TopDown. But TopDown with only 2 iterations avoids overfitting and outperforms Greedy dramatically in terms of time complexity and scalability. Our experiments also show that taxonomy adaptation can dynamically capture the content change in the evolving data.

This paper is a starting point for dynamic group profiling. Much work remains to be done along this direction. Some lines of immediate future work include:

- In this work, we assume the leaf-level topics in the taxonomy to be constant. In cases where a brand-new topic appears due to some recent new events, it would require to combine this work with topic detection and tracking [Allan 2002] to incorporate newly detected topics.
- Combining information epidemics [Gruhl et al. 2004] with our taxonomic representation can likely provide more useful and comprehensive profiles for group search and retrieval.
- How to specify a proper pace and time window to update the taxonomy requires more study for real-world applications. One simplest way is to update per day/week/month. A more interesting direction is to trigger the update automatically based on the content of newly collected documents.

Our profiles based on topic taxonomy provide a concise summary of various granularity for each online group. This kind of information is especially useful for group identification, and group relationship visualization. Our proposed approach for taxonomy adaptation is particularly applicable for an environment where the changes are reflected in data; our methods evolve a taxonomy by learning from the data as shown in the "Hurricane" example. Besides dynamic group profiling, taxonomy adaptation can also be used for some other potential applications, including automatic newswire feeder classification where each user subscribes to multiple topics, personalized email filtering and forwarding in which each user maintains a directory to store emails, online bookmark organization system where a topic taxonomy is maintained, and recommending systems and direct marketing.

8. ACKNOWLEDGMENTS

Lei Tang was partly supported by GPSA Research Grant of ASU.

REFERENCES

Adomavicius, G. and Tuzhilin, A. 2001. Using data mining methods to build customer profiles. *Computer 34,* 2, 74–82.

Aggarwal, C. C., Gates, S. C., and Yu, P. S. 1999. On the merits of building categorization systems by supervised clustering. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM Press, New York, NY, USA, 352–356.

Agrawal, R., Imielinski, T., and Swami, A. 1993. Mining association rules between sets of items in large databases. *SIGMOD Rec. 22,* 2, 207–216.

Airoldi, E. M., Fienberg, S. E., Joutard, C., and Love, T. M. 2006. Discovering latent patterns with hierarchical bayesian mixed-membership models. Tech. Rep. CMU-ML-06-101, School of Computer Science,Carnegie Mellon University.

Allan, J. 2002. *Introduction to topic detection and tracking.* Kluwer Academic Publishers, Norwell, MA, USA, 1–16.

Blei, D., Griffiths, T. L., Jordan, M. I., and Tenenbaum, J. B. 2003. Hierarchical topic models and the nested chinese restaurant process. In *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. MIT Press, Cambridge, MA.

Blei, D. M. and Lafferty, J. D. 2006. Dynamic topic models. In *ICML '06: Proceedings of the 23rd international conference on Machine learning.* ACM Press, New York, NY, USA, 113–120.

Blei, D. M., Ng, A. Y., and Jordan, M. I. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res. 3*, 993–1022.

Bounsaythip, C. and Rinta-Runsala, E. 2001. Overview of data mining for customer behavior modeling. http://virtual.vtt.fi/inf/julkaisut/muut/2001/customerprofiling.pdf.

Cai, L. and Hofmann, T. 2004. Hierarchical document categorization with support vector machines. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management.* ACM Press, New York, NY, USA, 78–87.

Cesa-Bianchi, N., Gentile, C., and Zaniboni, L. 2006a. Hierarchical classification: combining bayes with svm. In *ICML '06: Proceedings of the 23rd international conference on Machine learning.* ACM Press, New York, NY, USA, 177–184.

Cesa-Bianchi, N., Gentile, C., and Zaniboni, L. 2006b. Incremental algorithms for hierarchical classification. *J. Mach. Learn. Res. 7*, 31–54.

Chakrabarti, D., Kumar, R., and Tomkins, A. 2006. Evolutionary clustering. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM Press, New York, NY, USA, 554–560.

Chakrabarti, S., Dom, B., Agrawal, R., and Raghavan, P. 1998. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *The VLDB Journal 7,* 3, 163–178.

Chen, M.-C., Chiu, A.-L., and Chang, H.-H. 2005. Mining changes in customer behavior in retail marketing. *Expert Systems with Applications 28*, 773–781.

Chuang, S.-L. and Chien, L.-F. 2004. A practical web-based approach to generating topic hierarchy for text segments. In *CIKM '04: Proceedings of the thirteenth ACM international* *conference on Information and knowledge management*. ACM Press, New York, NY, USA, 127–136.

Dekel, O., Keshet, J., and Singer, Y. 2004. Large margin hierarchical classification. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*. ACM Press, New York, NY, USA, 27.

Dhillon, I. S., Fan, J., and Guan, Y. 2001. Efficient clustering of very large document collections. In *Data Mining for Scientific and Engineering Applications*. Kluwer Academic Publishers.

Dumais, S. and Chen, H. 2000. Hierarchical classification of web content. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM Press, New York, NY, USA, 256–263.

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

Gates, S. C., Teiken, W., and Cheng, K.-S. F. 2005. Taxonomies by the numbers: building high-performance taxonomies. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*. ACM Press, New York, NY, USA, 568–577.

Gruhl, D., Guha, R., Liben-Nowell, D., and Tomkins, A. 2004. Information diffusion through blogspace. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*. ACM Press, New York, NY, USA, 491–501.

Hofmann, T. 1999. The cluster-abstraction model: Unsupervised learning of topic hierarchies from text data. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 682–687.

Hwang, F. and Richards, D. 1992. The steiner tree problem. *Annals of Discrete Mathematics 53*.

Jain, A. K. and Dubes, R. C. 1988. *Algorithms for clustering data*. Prentice-Hall, Inc.

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

Li, T. and Zhu, S. 2005. Hierarchical document classification using automatically generated hierarchy. In *SIAM International Data Mining Conference*. Newport Beach, California, USA.

Liu, H. and Motoda, H., Eds. 2007. *Computational Methods of Feature Selection*. Chapman and Hall/CRC Press.

Liu, H. and Yu, L. 2005. Toward integrating feature selection algorithms for classification and clustering. *IEEE Trans. on Knowledge and Data Engineering 17,* 3, 1–12.

Liu, T.-Y., Yang, Y., Wan, H., Zeng, H.-J., Chen, Z., and Ma, W.-Y. 2005. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explor. Newsl. 7,* 1, 36–43.

McCallum, A. and Nigam, K. 1998. A comparison of event models for naive bayes text classification. In *In AAAI-98 Workshop on Learning for Text Categorization*.

McCallum, A., Rosenfeld, R., Mitchell, T. M., and Ng, A. Y. 1998. Improving text classification by shrinkage in a hierarchy of classes. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 359–367.

Punera, K., Rajan, S., and Ghosh, J. 2005. Automatically learning document taxonomies for hierarchical classification. In *WWW: Special interest tracks and posters of the 14th international conference on World Wide Web*. 1010–1011.

Rousu, J., Saunders, C., Szedmak, S., and Shawe-Taylor, J. 2005. Learning hierarchical multi-category text classification models. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*. ACM Press, New York, NY, USA, 744–751.

Ruiz, M. E. and Srinivasan, P. 1999. Hierarchical neural networks for text categorization (poster abstract). In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM Press, New York, NY, USA, 281–282.

Segal, E., Koller, D., and Ormoneit, D. 2001. Probabilistic abstraction hierarchies. In *Advances in Neural Information Processing Systems 14*. MIT Press, Vancouver, British Columbia, Canada, 913–920.

Shaw, M. J., Subramaniam, C., Tan, G. W., and Welge, M. E. 2001. Knowledge management and data mining for marketing. *Decis. Support Syst. 31,* 1, 127–137.

Sun, A. and Lim, E.-P. 2001. Hierarchical text classification and evaluation. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining.* IEEE Computer Society, Washington, DC, USA, 521–528.

Tang, L. and Liu, H. 2005. Bias analysis in text classification for highly skewed data. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining.* IEEE Computer Society, Washington, DC, USA, 781–784.

Tang, L., Zhang, J., and Liu, H. 2006. Acclimatizing taxonomic semantics for hierarchical content classification from semantics to data-driven taxonomy. In *KDD '06*. ACM Press, New York, NY, USA, 384–393.

Toutanova, K., Chen, F., Popat, K., and Hofmann, T. 2001. Text classification in a hierarchical mixture model for small training sets. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management.* ACM Press, New York, NY, USA, 105–113.

Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. 2004. Support vector machine learning for interdependent and structured output spaces. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning.* ACM Press, New York, NY, USA, 104.

Veeramachaneni, S., Sona, D., and Avesani, P. 2005. Hierarchical dirichlet model for document classification. In *ICML '05: Proceedings of the 22nd international conference on Machine learning.* ACM Press, New York, NY, USA, 928–935.

Wang, K., Zhou, S., and Liew, S. C. 1999. Building hierarchical classifiers using class proximity. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 363–374.

Weigend, A. S., Wiener, E. D., and Pedersen, J. O. 1999. Exploiting hierarchy in text categorization. *Inf. Retr. 1,* 3, 193–216.

Wibowo, W. and Williams, H. E. 2002. Strategies for minimising errors in hierarchical web categorisation. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management.* ACM Press, New York, NY, USA, 525–531.

Yang, Y. and Pedersen, J. O. 1997. A comparative study on feature selection in text categorization. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 412–420.

Yang, Y., Zhang, J., and Kisiel, B. 2003. A scalability analysis of classifiers in text categorization. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval.* ACM Press, New York, NY, USA, 96–103.

Zhang, L., Liu, S., Pan, Y., and Yang, L. 2004. Infoanalyzer: a computer-aided tool for building enterprise taxonomies. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management.* ACM Press, New York, NY, USA, 477–483.

Received February 2007; revised May 2007; accepted August 2007