Category Hierarchy Maintenance: A Data-Driven Approach

Quan Yuan[†], Gao Cong[†], Aixin Sun[†], Chin-Yew Lin[‡], Nadia Magnenat-Thalmann[†]
[†]School of Computer Engineering, Nanyang Technological University, Singapore
{qyuan1@e., gaocong@, axsun@, nadiathalmann@}ntu.edu.sg
[‡]Microsoft Research Asia, Beijing, China
cyl@microsoft.com

ABSTRACT

Category hierarchies often evolve at a much slower pace than the documents reside in. With newly available documents kept adding into a hierarchy, new topics emerge and documents within the same category become less topically cohesive. In this paper, we propose a novel automatic approach to modifying a given category hierarchy by redistributing its documents into more topically cohesive categories. The modification is achieved with three operations (namely, *sprout*, *merge*, and *assign*) with reference to an auxiliary hierarchy for additional semantic information; the auxiliary hierarchy covers a similar set of topics as the hierarchy to be modified. Our user study shows that the modified category hierarchy is semantically meaningful. As an extrinsic evaluation, we conduct experiments on document classification using real data from Yahoo! Answers and AnswerBag hierarchies, and compare the classification accuracies obtained on the original and the modified hierarchies. Our experiments show that the proposed method achieves much larger classification accuracy improvement compared with several baseline methods for hierarchy modification.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Filtering

Keywords

Category Hierarchy, Hierarchy Maintenance, Classification

1. INTRODUCTION

With the exponential growth of textual information accessible, category hierarchy becomes one of the most widely-adopted and effective solutions in organizing large volume of documents. Hierarchy provides an organization of data by different levels of abstraction, in which each node (or category) represents a topic that is shared by the data in it. The connection between two nodes denotes supertype-subtype relation. Examples include Web directories provided by Yahoo! and Open Directory Project (ODP), hierarchies for community-based question-answering services by Yahoo! Answers (YA) and AnswerBag (AB), product hierarchies by online retailers like Amazon and eBay, as well as the hierarchies for news browsing at many news portal websites. Figure 1 shows a small portion of Yahoo! Answers hierarchy. Questions in the same category are usually relevant to the same topic.

Hierarchy enables not only easy document browsing, but also searching of the documents within user defined categories or subtrees of categories. Additionally, hierarchy information can be utilized to enhance retrieval models to improve the search accuracy [4]. On the other hand, users' information needs can only be satisfied with *the documents* accessible through the hierarchy (but not the category hierarchy itself). That is, the usefulness of a hierarchy heavily relies on the effectiveness of the hierarchy in properly organizing the existing data, and more importantly accommodating the newly available data into the hierarchy. Given the fast growth of text data, continuously accommodating large volume of newly available text data into a hierarchy is nontrivial. Automatic text classification techniques are often employed for efficient categorization of newly available documents into category hierarchies. However, hierarchy often evolves at a much slower pace than its documents. Two major problems often arise after adding many documents into a hierarchy after some time.

- *Structure Irrelevance*. A category hierarchy may well reflect the topical distribution of its data at the time of construction. However, as new topics always emerge from the newly coming documents, there is no proper category in the hierarchy to accommodate these new documents, leading to putting these documents in less relevant categories. As the result, some categories contain less topically cohesive documents. Moreover, some categories become less discriminative with respect to the current data distribution. One example is the two categories Printers and Scanners in YA, for there emerged many questions about multi-functional devices which are related to both printers and scanners, leading to ambiguity between these two categories.

- *Semantics Irrelevance*. Semantics may change over time which calls for a better organization of the documents [15]. For instance, when creating the hierarchy, experts are more likely to put category Petroleum under Geography. However, after the disaster of BP Gulf Oil Spill, a lot of news articles in category Petroleum are about the responsibility of the Obama Administration. These documents have stronger connection to category Politics than Geography. It is therefore more reasonable to put these documents under Politics for better document organization.

These two problems not only hurt user experiences in accessing information through the hierarchy, but also result in poorer classification accuracy for the classifiers categorizing newly available documents because of the less topically cohesive categories [13].

SIGIR'12, August 12–16, 2012, Portland, Oregon, USA.

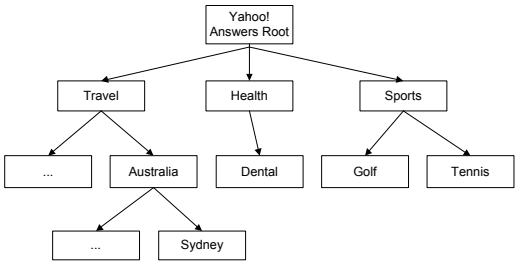


Figure 1: Portion of Yahoo! Answers Hierarchy

Consequently, the poorer classification accuracy further hurts user experience in browsing and searching documents through the hierarchy. This calls for ***category hierarchy maintenance****, a task to modify the hierarchy to make it better reflect the topics of its documents, which in turn would improve the classification accuracy*. Although a category hierarchy is relatively stable, many websites have modified or adjusted their hierarchies in the past. In May 2007, Yahoo! Answers added a new category Environment into her hierarchy, and added several categories like Facebook and Google under Internet later. By comparison, eBay adjusted her hierarchy more frequently, because there always emerge new types of items, like tablets and eReaders.

Hierarchy modification is nontrivial. Manual modification of category hierarchy is a tedious and difficult task, because it is hard to detect the semantic changes as well as the newly emerged topics. This motivates the *data-driven automatic modification* of a given hierarchy to cope with semantic changes and newly emerged topics. This is a challenging task because of at least two reasons, among others. First, the *resultant modified category hierarchy* (hereafter called **modified hierarchy** for short) should largely retain the semantics of the existing hierarchy and keep its category labels semantically meaningful. Second, the categories in the modified hierarchy shall demonstrate much higher topical cohesiveness, which in turn enables better classification accuracy in putting new documents into the modified hierarchy.

To the best of our knowledge, very few work has addressed the hierarchy modification problem (see Section 2). Tang *et al.* propose a novel approach to modifying the relations between categories aiming to improve the classification accuracy [15]. However, their proposed method does not change the leaf categories of the given hierarchy, and thus cannot solve the aforementioned problems. For example, the method may move the leaf category Petroleum to be child category of Politics. However, it is more reasonable to partition the documents in Petroleum into two categories: one being the child category of Geography, and the other child category of Politics. The method [15] fails to do so since it is unable to detect the newly emerged hidden topic "Petroleum politics".

In this paper, we propose a data-driven approach to modify a given hierarchy (also called as the original hierarchy) with reference to an auxiliary hierarchy using three operations (namely, *sprout*, *merge*, and *assign*). An auxiliary hierarchy is a category hierarchy that covers a similar set of topics as does the given hierarchy (*e.g.,* the Yahoo! hierarchy and ODP can be used as auxiliary hierarchy to each other). Similar to the concept of bisociation [8], our approach discovers finer and more elaborate categories (also known as hidden topics) by projecting the documents in the given hierarchy to the auxiliary hierarchy. This operation, similar to a cross-product operation between the categories from the given hierarchy and the categories from the auxiliary hierarchy, is named *sprout* [1]. The similar hidden topics are then *merged* to form new categories in the modified hierarchy. The *assign* operation rebuilds the parent-child relations in the modified hierarchy. The category labels in the modified hierarchy are either borrowed from or generated based on both the original and the auxiliary hierarchies. We emphasize that the reuse of category labels from original and auxiliary hierarchies largely ensures semantically meaningful category labels in the modified hierarchy. When such an auxiliary hierarchy is unavailable, the given hierarchy can be used as an auxiliary hierarchy. Because of the three operations (*i.e., sprout*, *merged*, and *assign*), we name our approach the *SMA* approach. The main contributions are summarized as follows.

1) We propose a novel data-driven approach *SMA* to automatically modify a category hierarchy making it better reflect the topics of its documents. The proposed approach exploits the semantics of the given hierarchy and an auxiliary hierarchy, to guide the modification of the given hierarchy.

2) We evaluate the proposed approach using data from three real-world hierarchies, Yahoo! Answers, Answerbag, and ODP. The user study shows that the modified hierarchy fits with the data better than the original one does. As we argue that the categories in the modified hierarchy are more topically cohesive compared to the original hierarchy, we employ text classification as an extrinsic evaluation. Our experimental results show that the classifiers trained on the modified hierarchy achieve much higher classification accuracy (measured by both macro-F_1 and micro-F_1), than the classifier built on the original hierarchy, or the classifiers modeled on the hierarchies generated by three baseline methods, including the state-of-the-art method in [15] and the hierarchy generation method in [2].

The rest of this paper is organized as follows. Section 2 surveys the related work. We describe the research problem and overview the proposed approach in Section 3. The three operations are detailed in Section 4. The experimental evaluation and discussion of the results are presented in Section 5. Finally, we conclude this paper in Section 6.

2. RELATED WORK

Hierarchy Generation. Hierarchy generation focuses on extracting a hierarchical structure from a set of categories, each containing a set of documents. The generation process can be either fully automatic [2, 5, 10] or semi-automatic [1, 7, 20]. The semi-automatic approaches involve interaction with domain experts in the hierarchy generation process. In the following, we review the fully automatic approaches in more detail.

Aggarwal *et al.* use the category labels of documents to supervise hierarchy generation [2]. They first calculate the centroids of all categories and use them as the initial seeds. Similar categories are merged and clusters with few documents are discarded. The process is iterated to build the hierarchy. User study is employed to evaluate the quality of the generated hierarchy.

Punera *et al.* utilizes a divisive hierarchical clustering approach, which first splits the given set of categories into two sets of categories, and each such set is partitioned recursively until it contains only one category [10].

An algorithm for generating hierarchy for short text is proposed by Chuang *et al.* [5]. They first create a binary-tree hierarchy by hierarchical agglomerative clustering, and then construct a multi-way-tree hierarchy from the binary-tree hierarchy. They use both classification measurement and user evaluation to evaluate the generated hierarchy.

Different from hierarchy generation which assumes a set of categories as input, our hierarchy modification method takes a hierar-

[1] We would like to thank an anonymous reviewer for suggesting the connection with bisociation [8] and the name *sprout*

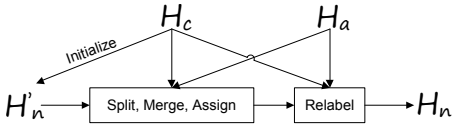


Figure 2: Overview of SMA

chy as the input. Hierarchy generation does not change the given categories hence it cannot solve the structure irrelevance problem.

Hierarchy Modification. To the best of our knowledge, [15] is the only work on adjusting a hierarchy to improve the classification accuracy. The method introduces three operations. The *promote* operation lifts a category to upper level; the *merge* operation generates a new parent for a category and its most similar sibling; the *demote* operation either demotes a category as a child of its most similar sibling, or makes the sibling a child of the category. For each category in the given hierarchy, promote operation is tested, followed by merge and demote operations, in a top-down manner. The operation comes into effect if it can improve the classification accuracy. The approach iterates the process until no improvement can be observed or some criterion is met. In experiments, this method outperforms clustering-based hierarchy generation method in terms of classification accuracy. However, this method does not change the leaf categories, leaving the topically incohesive leaf categories untouched.

Discussion. With the existing work on either hierarchy generation or hierarchy modification, the leaf categories in the modified hierarchy (*i.e.,* either generated or modified) remain unchanged. Clearly, without changing leaf categories, the topical incohesiveness among documents in the same leaf category remains unaddressed. Consequently, the likely poorer classification accuracy for these topically incohesive categories results in poorer document organization in the hierarchy. In this paper, we therefore propose an automatic approach to modify a given hierarchy where the leaf categories could be split or merged so as to better reflect the topics of the documents in the hierarchy.

3. SMA APPROACH OVERVIEW

We observe that each category in a hierarchy may contain several "hidden topics", each of which is topically cohesive, *e.g.,* category Computer contains hidden topics like Internet Programming, Operating Systems, etc. With more documents adding to a category hierarchy, new "hidden topics" emerge within a single category leading to topical incohesiveness among its documents (see Section 1). Our proposed approach, therefore, aims to find the hidden topics within each category and then *sprout* categories based on its hidden topics, *merge* similar hidden topics to form new categories, and then *assign* the parent-child relation among categories. We name our approach ***SMA*** after its three major operations.

The key challenges in the approach include: (i) How to detect the "hidden topics" at the appropriate granularity? (ii) How to evaluate the similarity between "hidden topics"? and (iii) How to assign the parent-child relation between the unmodified and modified categories? Further, recall from Section 1, the modified hierarchy has to largely retain the semantics of the existing hierarchy, with meaningful category labels and topically cohesive categories. In the following, we give a high-level overview of the SMA approach and then detail the three major operations in the next Section.

The framework of our SMA algorithm is illustrated in both Figure 2 and Algorithm 1, where $\mathcal{H}_c$ is the category hierarchy to be modified, $\mathcal{H}_n$ is the modified hierarchy, $\mathcal{H}'_n$ is the intermediate hierarchy during the modification process, and $\mathcal{H}_a$ is the auxiliary hierarchy.

Algorithm 1: SMA algorithm for hierarchy modification

Input: $\mathcal{H}_c$: category hierarchy to be modified
$\mathcal{H}_a$: auxiliary hierarchy
λ: minimum coverage ratio
θ: maximum loss ratio
Output: $\mathcal{H}_n$: modified category hierarchy

```
1  H'_n ← H_c;
2  h ← number of levels of H_c;
3  foreach Level ℓ from 2 to h of H'_n do
4      foreach Category C_i of H'_n on level ℓ do
5          Φ_{C_i} ← ProjectedCategories(C_i, H_a, λ, θ);
6          Sprout(C_i, Φ_{C_i}, H'_n, H_a);
7      n_ℓ ← number of categories on level ℓ of H_c;
8      Merge(ℓ, n_ℓ, H'_n);
9      Assign(ℓ, H'_n);
10 H_n ← Relabel(H_c, H_a, H'_n);
11 return H_n
```

Auxiliary Hierarchy. Briefly introduced in Section 1, an auxiliary hierarchy $\mathcal{H}_a$ is a hierarchy covering similar topics as the given hierarchy $\mathcal{H}_c$. For example, Yahoo! hierarchy and ODP hierarchy can be auxiliary hierarchy to each other. Similarly, Yahoo! Answers and AnswerBag can be auxiliary hierarchy to each other.

The auxiliary hierarchy $\mathcal{H}_a$ plays an essential role in finding hidden topics. Note that the hidden topics are not readily present in the auxiliary hierarchy, and our approach does not simply use the structure of auxiliary hierarchy as part of the modified hierarchy. Instead, they contain semantics from both the original hierarchy and the auxiliary hierarchy. We use the following example to illustrate. Suppose that the original hierarchy has two categories, Action movie and Comedy movie, and the auxiliary hierarchy contains two categories America and Asia. Our approach will find that Action movie has two hidden topics, namely American action movie and Asian active movie; Comedy movie also has two hidden topics, namely American comedy movie and Asian comedy movie.

The auxiliary hierarchy also plays an important role in guiding the merge operation, which is to merge similar hidden topics to generate the categories of the modified hierarchy. Continue with the earlier example, after merging the generated hidden topics, we may get new categories–American movie and Asian movie (if "action vs. comedy" is evaluated to be less discriminative compared with "American vs. Asian"). The semantics of the hierarchy to be modified, together with the semantics of the auxiliary hierarchy, will be exploited to define the similarity between hidden topics.

Validated in our experiments (Section 5), our approach is equally applicable when the original hierarchy $\mathcal{H}_c$ is used as the auxiliary hierarchy to itself.

Algorithm Overview. Shown in Figure 2 and Algorithm 1, $\mathcal{H}'_n$ is first initialized to $\mathcal{H}_c$ (line 1). In a top-down manner, the SMA algorithm modifies the hierarchy level by level. Note that the root category is the only category at level 1. Starting from level 2, for each category C_i in this level, the documents contained in C_i is projected to the auxiliary hierarchy $\mathcal{H}_a$. A set of categories from $\mathcal{H}_a$ each of which contains a reasonable number of documents originally from C_i is identified to represent C_i's hidden topics (line 5). The two parameters, minimum coverage ratio λ and maximum loss ratio θ, adjust the number of hidden topics. New finer categories

are then *sprouted* from C_i according to the hidden topics and the documents in category C_i are assigned to these finer categories (or hidden topics) (line 6). Given the expected number of categories n_ℓ on level ℓ (line 7), the *merge* operation forms n_ℓ number of new categories on level ℓ of the intermediate hierarchy $\mathcal{H}'_n$ (line 8). If the current level is not the lowest level in the hierarchy, the parent-child relations between the modified categories and the unmodified categories on the next level are assigned (line 9). The last step in the SMA algorithm is to generate category labels with reference to both the original and auxiliary hierarchies (line 10).

4. SPROUT, MERGE, AND ASSIGN

We detail the three operations to address the challenges in the SMA framework (*i.e.,* to identify hidden topics, evaluate the similarity between hidden topics, and assign the parent-child relation).

4.1 Sprout Operation

The sprout operation first discovers the hidden topics for the documents in a category C_i and then sprouts the category. Without loss of generalization, a leaf category is represented by all documents belonging to the category; a non-leaf category is represented by all documents belonging to any of its descendent categories.

4.1.1 Discovery of Hidden Topics

Ideally, for a category we find a set of its hidden topics, which are comprehensive and cohesive, and have no overlap. This is however a challenging task. We proceed to give an overview of the proposed method. Given a category C_i in the intermediate hierarchy $\mathcal{H}'_n$ during the modification process (see Algorithm 1), we assign all its documents into the categories of the auxiliary hierarchy $\mathcal{H}_a$, and get a set of candidate categories from $\mathcal{H}_a$ in a tree-structure. Each candidate category contains a number of documents from C_i. Then, with the consideration of both cohesion and separation, we select a set of categories from the tree as hidden topics. The selection process is modeled as an optimization problem. We now elaborate the details.

Document Projection. To assign documents from C_i to $\mathcal{H}_a$, we represent a document by its word feature vector, and a category in $\mathcal{H}_a$ by its centroid vector. Based on cosine similarity between the document and the centroids, we recursively assign each document $d \in C_i$ to $\mathcal{H}_a$ from its root to a leaf category along a single path of categories. If a good number of documents from C_i are assigned to a category C_a in $\mathcal{H}_a$, then the topic of C_a is relevant to C_i, and the semantics of C_a can be used to describe a hidden topic of C_i. Thus, multiple categories in C_a can be identified to describe all hidden topics of C_i. For example, large number of documents from category Programming assigned into two categories Security and Network in an auxiliary hierarchy, implies that Programming has two hidden topics: Network Programming and Security Programming. We have also tried to build a classifier on $\mathcal{H}_a$ to assign documents from C_i to $\mathcal{H}_a$ using Naive Bayes and support vector machine, respectively, and the set of generated hidden topics is almost the same.

The process of assigning documents from a category C_i in $\mathcal{H}'_n$ to categories in $\mathcal{H}_a$ is called *projection*. We denote the set of documents *projected* from category C_i to category C_a by $\pi(C_i \to C_a)$. If C_a is a leaf category, then $\pi(C_i \to C_a)$ denotes the set of documents from C_i that are projected into C_a; if C_a is a non-leaf category, then $\pi(C_i \to C_a)$ denotes the set of documents projected into any of the descendent leaf categories of C_a in $\mathcal{H}_a$.

Candidate Topic Tree. Based on the projection, we select categories from $\mathcal{H}_a$ to represent the hidden topics of C_i. A selected category can be either a leaf category or a non-leaf category. Before describing the selection process, we introduce the notions of *major category* and *minor category*. Let λ denote the *minimum coverage ratio* parameter.

Definition 1 (**Major Category**). *A category C_a from $\mathcal{H}_a$ is a major category for category C_i if $|\pi(C_i \to C_a)|/|C_i| \geq \lambda$.*

Definition 2 (**Minor Category**). *A category C_a from $\mathcal{H}_a$ is a minor category for category C_i if $|\pi(C_i \to C_a)|/|C_i| < \lambda$.*

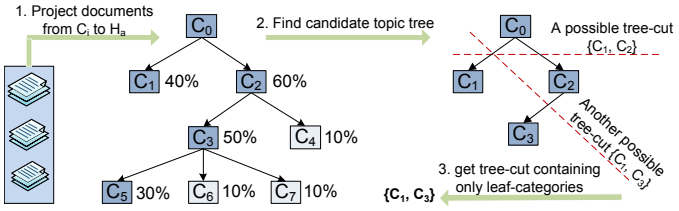


Figure 3: Generate candidate topic tree for C_i using $\mathcal{H}_a$

For example, suppose $\lambda = 15\%$. As shown in Figure 3, category C_i is projected to the categories in $\mathcal{H}_a$. In the left tree, in which each number besides a node represents the percentage of documents of C_i projected to the node, the nodes in dark color are major categories while the others are minor categories.

Naturally, only the major categories are considered candidate categories to represent hidden topics of C_i because a good number of documents in C_i are projected into them. However, not all major categories need to be selected because of two reasons. First, let C_p be the parent of a major category C_a. By definition, the parent of a major category is also a major category. Selecting both C_a and C_p would lead to semantic overlap. Second, assume all C_p's other child categories are minor categories, but altogether those minor categories contain a large number of documents. Then selecting C_a but not C_p would lead to a significant loss of documents from C_i (hence semantic loss). We therefore define the notion of *loss ratio*.

Definition 3 (**Loss Ratio**). *The loss ratio of a leaf category is defined as 0. For a non-leaf category C_a, let C_a^μ be the set of minor categories among C_a's child categories. The loss ratio of C_a with respect to C_i, the category being projected, is the ratio between the projected documents in all its child minor categories and C_a's projected documents,* i.e., $\frac{\sum_{C' \in C_a^\mu} |\pi(C_i \to C')|}{|\pi(C_i \to C_a)|}$.

We set a threshold *maximum loss ratio* θ. After projecting documents from C_i to categories in $\mathcal{H}_a$, we only keep the major categories whose parent's loss ratio is smaller than θ. Note that, if a non-leaf category is not selected in the above process, the subtree rooted at this category is not selected. After the selection, we obtain a sub-hierarchy from $\mathcal{H}_a$ containing only eligible major categories, which is called *candidate topic tree* for C_i, denoted by $\mathcal{T}_{C_i}$.

For example, suppose $\theta = 30\%$. The *candidate topic tree* for C_i is shown on the right hand side of Figure 3. Although node C_5 is a major category, it is not part of the *candidate topic tree* since the loss ratio ((10%+10%)/50% = 40%) of its parent node C_3 is larger than θ.

Hidden Topic Selection. We next present how to choose a set of nodes from $\mathcal{T}_{C_i}$ to represent hidden topics of C_i. Ideally, we expect the hidden topics to be comprehensive but not overlap with each other. Hence, we use *tree-cut* to define the selection [16].

Definition 4 (**Tree-Cut**). *A tree-cut is a partition of a tree. It is a list of nodes in the tree, and each node represents a set of all*

leaf nodes in a subtree rooted by the node. The sets in a tree-cut exhaustively cover all leaf nodes of the tree, and they are mutually disjoint.

There exist many possible tree-cuts for $\mathcal{T}_{C_i}$ to generate hidden topics. Two example tree cuts for the *candidate topic tree* in Figure 3 are $\{C_1, C_2\}$ and $\{C_1, C_3\}$. Among all possible tree-cuts, we aim to choose the tree-cut such that each resultant hidden topic (category) is cohesive and well separated from other categories in the tree-cut. In the following, we prove that the tree-cut containing only leaf nodes of the *candidate topic tree* satisfies this requirement. Note that a leaf node in $\mathcal{T}_{C_i}$ is not necessary a leaf category in $\mathcal{H}_a$. For example, in Figure 3, C_3 is leaf node of the candidate topic tree, but not a leaf category in the auxiliary hierarchy.

We proceed to show the proof. We first define the Sum of Square Error (SSE) of cohesion for a category C_a.

$$SSE(C_a) = \sum_{d \in C_a} (d - \overline{c}_a)^2,$$

where d is a document and $\overline{c_a}$ is the centroid of category C_a.

Given a set of categories $\{C_a\}$ $(1 \le a \le k)$, the Total-SSE and Total Sum of Square Between (Total-SSB), denoted by ε_E and ε_B respectively, are $\varepsilon_E = \sum_{a=1}^{k} SSE(C_a)$ and $\varepsilon_B = \sum_{a=1}^{k} |C_a|(\overline{c} - \overline{c_a})^2$, where $\overline{c}$ is the centroid of documents in all categories $\{C_a\}$. It is verified that, given a set of documents, the sum of ε_E and ε_B is a constant value [14]: $\varepsilon_E + \varepsilon_B = \sum_{a=1}^{k} \sum_{d \in C_a} (d - \overline{c})^2$. Thus, maximizing separation is equivalent to minimizing cohesion error. We therefore formulate the problem of selecting categories from $\mathcal{T}_{C_i}$ to represent hidden topics for category C_i as following:

$$\Phi_{C_i} = \arg\min_{\mathcal{S}} \sum_{C_a \in \mathcal{S}} SSE(C_a), \text{ where } \mathcal{S} \text{ is a tree-cut on } \mathcal{T}_{C_i}. \quad (1)$$

This problem can be reduced to the maximum flow problem by viewing $\mathcal{T}_{C_i}$ as a flow network. Thus, it can be solved directly by Ford-Fulkerson method [6]. However, its complexity is relatively high. Note that we need to solve the optimization problem for every category in the original hierarchy, and thus an efficient algorithm is essential.

Lemma 1: The SSE of a category is not smaller than the Total-SSE of its child categories.
Proof: see Appendix A.

Lemma 1 enables us to use an efficient method to solve Eq.1 as follows. According to Lemma 1, specializing a category by its child categories can reduce Total-SSE. That is, among all possible tree-cuts in $\mathcal{T}_{C_i}$, the cut that contains only leaf categories has the minimum value of Total-SSE.

In summary, for a category C_i in $\mathcal{H}'_n$, we build a candidate topic tree $\mathcal{T}_{C_i}$ and the leaf nodes of $\mathcal{T}_{C_i}$ are used to represent the hidden topics of C_i. The pseudocode is given in Procedure 2 ProjectedCategories. As discussed, according to Lemma 1, we only need the leaf nodes of the candidate topic tree $\mathcal{T}_{C_i}$ as the result. Instead of explicitly building $\mathcal{T}_{C_i}$ and then finding $\mathcal{T}_{C_i}$'s tree cut containing only leaf nodes, we find $\mathcal{T}_{C_i}$'s leaf nodes directly in our procedure. More specifically, we start from the root category of $\mathcal{H}_a$ in a top-down manner (the root node is a major category by definition as its coverage ratio is 1). Each time we get a unprocessed categories C_a from the list of projected categories $\Phi_{C_i}[]$, and check its child categories (lines 3-4). The major categories among the child categories are put into a major category list (lines 6-8) for further testing on loss ratio. If the loss ratio of C_a is smaller than maximum loss ratio θ, then C_a is replaced by its child major categories (lines 9-11); otherwise, C_a is selected as a candidate category (line 13). We iterate the process until all major categories are processed (line 14).

Procedure ProjectedCategories

Input: C_i: the category to be sprouted
$\mathcal{H}_a$: the auxiliary hierarchy
λ: minimum coverage ratio
θ: maximum loss ratio
Result: $\Phi_{C_i}[]$: the list of projected categories for C_i

```
1  Φ_{C_i}[] ← {root category of H_a};
2  repeat
3      C_a ← Φ_{C_i}[].getNextUnprocessedCategory();
4      C_List[] ← child categories of C_a;
5      M_List[] ← {};
6      foreach Category C of C_List[] do
7          if |π(C_i → C)| / |C_i| ≥ λ then
8              M_List[].add(C);
9      if 1 − Σ_{C∈M_List[]} |π(C_i → C)| / |π(C_i → C_a)| < θ then
10         Φ_{C_i}[].add(M_List[]);
11         Φ_{C_i}[].remove(C_a);
12     else
13         mark C_a as processed
14 until No more unprocessed category in Φ_{C_i}[]
15 return Φ_{C_i}[]
```

4.1.2 Sprout Category

For a category C_i of $\mathcal{H}'_n$, we sprout it based on the projected categories Φ_{C_i} returned by Procedure ProjectedCategories. Recall that each of the projected category $C_a \in \Phi_{C_i}$ represents a hidden topic of C_i and contains a good number of documents projected from C_i, *i.e.,* $\pi(C_i \to C_a)$. We sprout C_i with $|\Phi_{C_i}|$ number of categories. However, not all documents from C_i are contained in all these newly sprouted categories, *i.e.,* $\sum_{C_a \in \Phi_{C_i}} |\pi(C_i \to C_a)| \le |C_i|$. For those documents in C_i but not contained in any of the newly sprouted categories, we assign them to their nearest sprouted categories. As the result, each document in C_i is now contained in one and only one sprouted category of C_i.

Example 4.1: Shown in Figure 4[2], suppose after applying procedure ProjectedCategories, we find Network is projected to Security, Protocol, and Cable. According to the three hidden topics, we sprout Network into three categories Network Security, Network Protocol, and Network Cable. □

The sprout operation may be reminiscent of the work on hierarchy integration, aiming to integrate a category from a source hierarchy into similar categories in the target hierarchy, which has a different purpose from our mapping. Most of proposals (e.g., [3, 19]) on hierarchy integration employ a hierarchical classifier built on the target hierarchy to classify each document in the source hierarchy into a leaf node of the target hierarchy, which is too fine a granularity to represent hidden topics as in our approach. Frameworks that can map a category to categories on proper levels in target hierarchy are proposed (*e.g.,* [17]). However, they do not take the cohesion and separation between mapping categories into account, which are essential to find good hidden topics in our approach. Thus, they cannot be applied to our work.

4.2 Merge Operation

The sprout operation in Section 4.1 generates a set of sprouted categories, each representing a hidden topic. The merge operation

[2]For clarity, we recommend viewing all diagrams in this paper from a color copy.

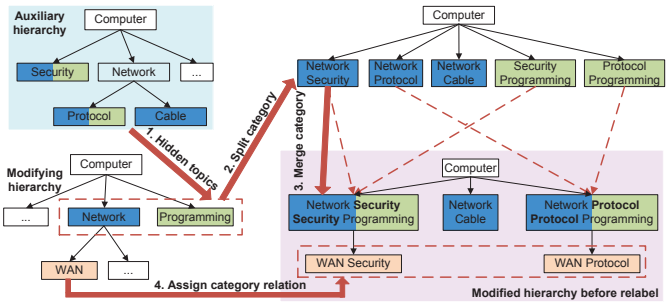


Figure 4: SMA operations by example. The hidden topics and sprouted categories for a category of the original hierarchy are in the same color. The Network and Programming categories have 3 and 2 hidden topics, respectively, leading to 5 sibling sprouted categories. These 5 categories are merged into 3 categories and the category WAN is reassigned to 2 of the merged categories.

aims to combine the newly sprouted categories with similar hidden topics.

Suppose we are now working on level ℓ of the intermediate modified hierarchy $\mathcal{H}'_n$ and we have a set of sprouted categories originated from the categories on level ℓ. Our task is to merge some of these sprouted categories such that the number of resultant categories on level ℓ is the same as before (i.e., n_ℓ). Note that the number of resultant categories can also be specified by users. To ease the presentation, the modified hierarchy has the same size as the given hierarchy in our discussion.

During merge, we need to consider an important constraint — we can only merge categories under the same parent category. Thus, existing clustering algorithms need to be modified to accommodate such a constraint. Another key issue here is how to define the similarity between two sprouted categories by considering their semantics enclosed in $\mathcal{H}_c$ and $\mathcal{H}_a$. In the following, we first define a similarity measure and then describe our merge method.

We consider two aspects when defining the similarity for a pair of sprouted categories C_1 and C_2 on level ℓ: (i) the distribution of their documents over categories of $\mathcal{H}_c$ and $\mathcal{H}_a$, and (ii) the similarity between the categories within $\mathcal{H}_c$ and $\mathcal{H}_a$, respectively.

Let $\mathcal{L}_c$ be the set of categories on level ℓ in the original hierarch $\mathcal{H}_c$, $\mathcal{L}_s$ be the set of categories sprouted from $\mathcal{L}_c$, and $\mathcal{L}_a$ be the set of projected categories in auxiliary hierarchy representing the hidden topics of the categories in $\mathcal{L}_c$. That is, $\mathcal{L}_a = \bigcup_{C_i \in \mathcal{L}_c} \Phi_{C_i}$.

For a sprouted category $C_s \in \mathcal{L}_s$, its document distribution over $\mathcal{L}_c$ is defined to be the ratios of its documents in each of the categories in $\mathcal{L}_c$. That is, the document distribution of C_s can be modeled as a $|\mathcal{L}_c|$-dimensional vector $\mathbf{v_{cs}}$. The j-th element of $\mathbf{v_{cs}}$ is $\frac{|C_s \cap C_j|}{|C_s|}$ (*i.e.,* the portion of C_s's documents also contained in C_j), where C_j is the j-th category of $\mathcal{L}_c$. Similarly, we get the data distribution vector $\mathbf{v_{as}}$ for C_s over $\mathcal{L}_a$ based on the ratio of documents in C_s projected to each of the categories in $\mathcal{L}_a$. Because $\mathbf{v_{cs}}$ and $\mathbf{v_{as}}$ usually have different dimensionality for different sprouted category C_s's, we extend $\mathbf{v_{cs}}$ to be $|\mathcal{H}_c|$-dimensional (each category is one dimension) by filling up zeros for corresponding categories in $\mathcal{H}_c$ but not in $\mathcal{L}_c$. Similarly $\mathbf{v_{as}}$ is extended to be $|\mathcal{H}_a|$-dimensional.

We use two matrices $\mathbf{M_c}$, $\mathbf{M_a}$ to represent the similarity between categories of $\mathcal{H}_c$ and $\mathcal{H}_a$, respectively. $\mathbf{M_c}$ is a $|\mathcal{H}_c|$- by-$|\mathcal{H}_c|$ matrix and $\mathbf{M_a}$ is a $|\mathcal{H}_a|$-by-$|\mathcal{H}_a|$ matrix. Each element m_{ij} in a matrix represents the similarity in the corresponding hierarchy between a pair of categories C_i and C_j, which is defined by *Inverted Path Length* [12]: $m_{ij} = \frac{1}{1+path(C_i,C_j)}$, where $path(C_i, C_j)$ is the length of path between C_i and C_j in the hierarchy.

Considering both document distribution and structural similarity from the two hierarchies, the similarity between two sprouted categories C_1 and C_2 on level ℓ of $\mathcal{H}'_n$ is defined as:

$$Sim(C_1, C_2) = (\mathbf{v_{c1}^T} \cdot \mathbf{M_c} \cdot \mathbf{v_{c2}}) + (\mathbf{v_{a1}^T} \cdot \mathbf{M_a} \cdot \mathbf{v_{a2}}).$$

This similarity definition considers both the similarity estimated based on $\mathcal{H}_c$ and the similarity estimated based on $\mathcal{H}_a$. With similarity between two sprouted categories defined, we proceed to detail the merge operation.

We first explain the notion of *sibling sprouted category* through an example. Let C_{i1} and C_{i2} be the two categories sprouted from category C_i, and C_{j1} and C_{j2} be the two categories sprouted from C_j. If C_i and C_j in $\mathcal{H}'_m$ are both children of category C_p, then naturally, all the newly sprouted categories C_{i1}, C_{i2}, C_{j1}, C_{j2} are children of C_p. These four example categories are known as sibling sprouted categories. All the five sprouted categories shown in Figure 4 are sibling sprouted categories.

The merge operation is as follows. We first calculate the similarity between sibling sprouted categories on level ℓ. Then, we pick up the pair of categories with the largest similarity, and merge them into a category, and recompute its similarities with its sibling sprouted categories. The process iterates until the number of remaining categories on ℓ equals n_ℓ, the number of categories on level ℓ of the original hierarchy $\mathcal{H}_c$. When all the sibling sprouted categories under the same parent node are merged into a single category, we shrink the single category into its parent node. Note that we cannot merge two sprouted categories on level ℓ if they have different parent node.

Example 4.2: Recall Example 4.1. We sprout Network into three categories Network Security, Network Protocol, and Network Cable. Suppose there is another category Programming on the same level of Network and sprouted into Security Programming, and Protocol Programming (see Figure 4). Based on the similarity, Network Security and Security Programming are merged together (both are about the Security topic), and Network Protocol and Protocol Programming are merged to generate a new category about protocol. □

4.3 Assign Operation

After modifying categories (by sprout and merge) on level ℓ of $\mathcal{H}'_n$, the original parent-child relations between the categories on level ℓ and the categories on next level $\ell+1$ do not hold any longer. Hence we need to reassign the parent-child relation.

Based on the fact that a non-leaf category of a hierarchy subsumes the data of all its descendants, we rebuild the children for each of the modified categories on ℓ by checking document containment. If the documents of a category on level ℓ are also contained in a category on level $\ell+1$, then the latter category is assigned to be the children of the former category. In other words, for each category C on ℓ, we calculate the intersection of documents between C and the categories on $\ell+1$. The intersections form new children for category C. Because one category has only one parent category, if a category has intersections with more than one category on level ℓ, the category will be split into multiple categories, each containing the intersection with one category on level ℓ.

Example 4.3: Recall Example 4.2. Suppose WAN was a child category of Network before sprout (see Figure 4). After sprout and merge, Network no longer exists and WAN lost its parent. We compare the documents of WAN and the two newly formed categories after merge (*i.e.,* Network Security & Security Programming and

Table 1: Statistics of the three hierarchies

Hierarchy	$\mathcal{H}_{YA}$	$\mathcal{H}_{AB}$	$\mathcal{H}_{ODP}$
Number of documents	421,163	148,822	203,448
Number of leaf nodes	75	195	460
Number of non-leaf nodes	40	70	98
Height	4	5	4

Network Protocol & Protocol Programming). If WAN has overlap with both categories, then WAN have two hidden topics (about security and protocol). Thus, we divide WAN into two categories and assign them to different parent nodes, shown in Figure 4. □

4.4 Relabel

Unlike most of previous work, our approach is able to automatically generate readable labels for every modified category. By projecting documents from $\mathcal{H}'_n$ to $\mathcal{H}_a$, we can consider the three hierarchies $\mathcal{H}'_n$, $\mathcal{H}_c$ and $\mathcal{H}_a$, which contain the same set of data. For every document in $\mathcal{H}'_n$, we trace its labels in $\mathcal{H}_c$ and $\mathcal{H}_a$, and use them together as the label of the document; the semantic of the new label is the intersection of semantics of its two component labels. We then aggregate such labels for all documents in a category of $\mathcal{H}'_n$, and use them as candidate labels for the category. The candidate labels for a category are ranked according to the proportion of documents in their corresponding original categories from $\mathcal{H}_c$ and $\mathcal{H}_a$. In this paper, the top-1 ranked label is chosen.

The labels generated in this way are mostly readable and semantically meaningful, as reflected in our user study (see Section 5.3) and case study (see Section 5.4). Nevertheless, a manual verification of the labels for the newly generated categories can be employed when the proposed technique is used in real applications.

5. EXPERIMENTS

We designed two sets of experiments. The first set of experiment, similar to that in [15], is to evaluate whether the modified hierarchy improves the classification accuracy. Discussed in Section 1, if a category hierarchy better reflects the topics of its contained documents and each category in the hierarchy is topically cohesive, then better classification accuracy is expected than that on a hierarchy with less topically cohesive categories. The second set of experiments employs a user study to manually evaluate the semantic quality of the modified hierarchy following the settings in [2,5]. Finally, we report a case study comparing a part of Yahoo! Answers hierarchy with its modified hierarchy.

5.1 Data Set

We use data from three real-world hierarchies: Yahoo! Answers, AnswerBag, Open Directory Project, denoted by $\mathcal{H}_{YA}$, $\mathcal{H}_{AB}$ and $\mathcal{H}_{ODP}$, respectively. Since the modified category hierarchy contains a different set of leaf nodes, the labels for documents given in the original dataset do not stand in modified hierarchy. Manual annotation of documents in the modified hierarchy is therefore unavoidable. To make the annotation manageable, we selected the documents from two major topics Sports and Computers from these three hierarchies (because the annotators are familiar with both topics). Nevertheless, the number of documents in the two major topics in the three hierarchies ranges from 148,822 to 421,163, and the number of categories ranges from 115 to 558. These numbers are large enough for a valid evaluation. Table 1 reports the statistics on the three hierarchies.

$\mathcal{H}_{YA}$: Obtained from the Yahoo! Webscope datatset[3], $\mathcal{H}_{YA}$ contains 421,163 documents (or questions) from Sports and Computers & Internet categories.

$\mathcal{H}_{AB}$: We collected 148,822 questions from Recreation & Sports and Computers categories from AnswerBag to form $\mathcal{H}_{AB}$. Categories with fewer than 100 questions are pruned and all affected questions are moved to their parent categories.

$\mathcal{H}_{ODP}$: The set of 203,448 documents from Sports and Computers categories are collected[4] in $\mathcal{H}_{ODP}$. Categories containing fewer than 15 documents or located on level 5 or deeper are removed in our experiments.

The preprocessing of the documents in all three hierarchies includes stopword removal and stemming. Terms occurred no more than 3 times across the datasets are also removed.

5.2 Evaluation by Classification

The proposed SMA algorithm modifies a category hierarchy to better reflect the topics of its documents, which in turn should improve the classification performance. Following the experimental setting in [15], we evaluate the effectiveness of hierarchy modification by comparing the classification accuracies obtained by the same hierarchical classification model applied on the original category hierarchy and the modified hierarchy, respectively.

Another three methods for hierarchy modification are employed as the baselines, namely, Bottom Up Clustering (BUC), Hierarchical Acclimatization (HA) [15], and Supervised Clustering (SC) [2]. Table 2 gives a summarized comparison of the three baselines with the proposed SMA, and Section 5.2.1 briefs the baseline methods.

The modified hierarchies by all the methods evaluated in this paper have the same size as the original hierarchy (*i.e.,* same number of levels, and same number of categories in each level). For each hierarchy modification method, we evaluate the percentage of classification accuracy increment obtained by the same classification model (*e.g.,* Support Vector Machine) on the modified hierarchy over the original hierarchy. The classification accuracy is measured by both micro-average F_1 (Micro-F_1) and macro-averaged F_1 (Macro-F_1) [18]. The former gives equal weight to every document while the latter weighs categories equally regardless the number of documents in each category.

We remark that this is a fair evaluation for all the methods, each generating a hierarchy with the same size as that of the original hierarchy, where the same classification method is applied to the modified hierarchies to evaluate the improvement of each modified hierarchy over the original one in terms of classification accuracy.

5.2.1 Baseline Methods

Baseline 1: Bottom Up Clustering (BUC). In this method, each leaf category is represented by the mean vector of its contained documents. The categories are then clustered in a bottom-up manner using K-means to form a hierarchy.

Baseline 2: Hierarchical Acclimatization (HA). The HA algorithm is reviewed in Section 2, In simple words, it employs promote, demote and merge operations to adjust the internal structure, but leaves the leaf nodes unchanged [15].

Baseline 3: Supervised Clustering (SC). Given a set of documents with labels, SC first calculates the mean vector of each category as the initial centroid and then reassigns the documents to the categories based on the cosine similarity with their centroids. Then, similar categories are merged and minor categories are removed. These procedures are repeated, and during each iteration, a

[3] Available at `http://research.yahoo.com/Academic_Relations`.

[4] Available at `http://www.dmoz.org/rdf.html`.

Table 2: Comparison of baseline methods with SMA

Aspect/Methods	BUC	HA [15]	SC [2]	SMA
Utilize original hierarchy	×	√	×	√
Change leaf category	×	×	√	√
Utilize auxiliary hierarchy	×	×	×	Optional

constant portion of features with smallest term-frequencies are set to zero (projected out) [2]. The process stops when the number of features left is smaller than a pre-defined threshold. This method cannot generate category labels. We take the most frequent words in a category to name it.

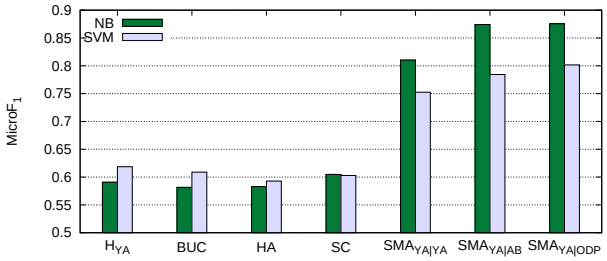
5.2.2 *Experiments on Yahoo! Answers*

From the data of $\mathcal{H}_{YA}$, we randomly selected 500 questions as test data (used for classification evaluation with manual annotations). To evaluate the possible improvement in classification accuracy, the same set of test documents are classified on the original (or unmodified) $\mathcal{H}_{YA}$, and the modified $\mathcal{H}_{nYA}$'s. Two classifiers, multinominal Naive Bayes (NB) and Support Vector Machine (SVM) classifiers are used as base classifiers for hierarchical classification. We build Single Path Hierarchical Classifier (SPH) [9] as it performs better than other hierarchical classification methods for question classification according to the evaluation [11]. In the training phase of SPH, for each internal node of the category tree, SPH trains a classifier using the documents belonging to its descendent nodes. In the testing phase, a test document is classified from the root to a leaf node in the hierarchy along a single path.

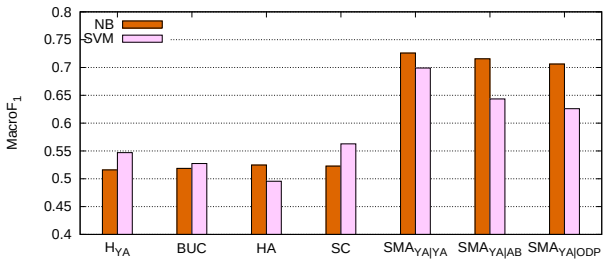
SMA Settings. Recall that SMA uses auxiliary hierarchy in the modification process. We evaluated SMA with three settings, to modify $\mathcal{H}_{YA}$ using $\mathcal{H}_{YA}$, $\mathcal{H}_{AB}$, and $\mathcal{H}_{ODP}$ as auxiliary hierarchy, respectively. The three settings are denoted by $SMA_{YA|YA}$, $SMA_{YA|AB}$, and $SMA_{YA|ODP}$, respectively. The first setting is to evaluate the effectiveness of using the original hierarchy as auxiliary hierarchy, and the last two are to evaluate the effectiveness of using external hierarchies.

Parameter Setting. Before evaluating the test documents on the modified hierarchies, we set the parameters required by SMA for hierarchy modification. Recall that SMA requires two parameters: minimum coverage ratio λ and maximum loss ratio θ. Usually parameters are set using a development set or through cross-validation. In our case, however, there is no ground truth on how good a modified hierarchy is and manual assessment of every modified hierarchy for parameter tuning is impractical. We therefore adopt a bootstrapping like approach described below.

After the test data selected, the remaining data is used for hierarchy modification. We split the remaining data of $\mathcal{H}_{YA}$ into 3 parts: P_1, P_2, P_3, and the proportion of their sizes is 12:3:1. Using P_1 for $\mathcal{H}_{YA}$ and a given auxiliary hierarchy, we obtain a modified hierarchy $\mathcal{H}_{nYA}$. Naturally, all documents in P_1 have category labels from hierarchy $\mathcal{H}_{nYA}$. We then build a classifier using all documents in P_1 and their labels from $\mathcal{H}_{nYA}$. The classifier classifies documents in P_2 and P_3. Assume that the classifier gives reasonably good classification accuracy, then all documents in P_2 and P_3 have their category labels assigned according to $\mathcal{H}_{nYA}$. With these labels, we can evaluate the classification accuracy of documents in P_3 by the classifier built using P_2 on $\mathcal{H}_{nYA}$. Intuitively, if a hierarchy $\mathcal{H}_1$ better organizes documents than another hierarchy $\mathcal{H}_2$, then the classifier trained on $\mathcal{H}_1$ is expected to have higher classification accuracy for P_3 than a classifier built on $\mathcal{H}_2$. We then select the parameters leading to the best classification accuracy for P_3. In our experiments, the parameters (*i.e.,* λ and θ) set for $SMA_{YA|YA}$, $SMA_{YA|AB}$, and $SMA_{YA|ODP}$ are (0.29 and 0.11), (0.17 and 0.17), (0.38 and 0.08), respectively.



(a) Micro-F_1



(b) Macro-F_1

Figure 5: Micro-F_1 and Macro-F_1 on the modified hierarchies by three baselines and three SMA settings, and on the original Yahoo! Answers hierarchy.

Test Data Annotation. With the chosen parameters, each SMA setting generated a modified hierarchy using P_1 as $\mathcal{H}_{YA}$ and its corresponding auxiliary hierarchy. The preselected 500 test questions are used as test data to fairly evaluate the modified hierarchies by the three SMA settings and the baseline methods. Recall that the 500 questions are not included in the three parts (P_1, P_2, and P_3) for parameter setting. Because BUC and HA do not change the leaf categories, the original labels of the 500 questions remain applicable. For SC and SMA, both changing leaf categories, we invited two annotators to label the 500 questions to their most relevant leaf categories in the modified hierarchies. We synthesized the results of the annotators, and assigned the labels for questions. If two annotators conflicted about a label, a third person made the final judgment. The dataset and their annotations are available online [5].

Classification Results. Classification accuracy measured by Micro-F_1 using NB and SVM as base classifiers for the six methods (*i.e.,* three baselines BUC, HA, SC, and the three SMA settings) on modified hierarchies is reported in Figure 5(a). For comparison, the classification accuracy on the original (or unmodified) Yahoo! Answers hierarchy is also reported under column named $\mathcal{H}_{YA}$. Figure 5(b) reports Macro-F_1.

As shown in Figures 5(a) and 5(b), all the three settings of SMA achieve significant improvement over the results obtained on the original hierarchy. For example, using NB as the base classifier, $SMA_{YA|YA}$ improves Micro-F_1 over the results on the original hierarchy by 41.0% and improves Macro-F_1 by 40.3%. NB achieves better accuracy than SVM probably because NB was used as the base classifier for parameter setting. The three baseline modification methods only slightly improve the classification accuracy over the original hierarchy and even deteriorate the accuracy in some cases. Recall that SC and SMA modify leaf categories while BUC

[5] http://www.ntu.edu.sg/home/gaocong/datacode.htm

Table 3: Classification accuracy on modifying AnswerBag

Measure/Hierarchy	$\mathcal{H}_{AB}$	$SMA_{AB\|YA}$	Improvement(%)
Macro-F_1 (NB)	0.3444	0.5638	63.7%
Macro-F_1 (SVM)	0.3691	0.4933	33.6%
Micro-F_1 (NB)	0.4671	0.6669	42.8%
Micro-F_1 (SVM)	0.4371	0.5697	30.3%

Table 4: Comparison on appropriateness of category labels

Judgement	Number of documents
$\mathcal{H}_{nYA}$ is better than $\mathcal{H}_{YA}$	12
$\mathcal{H}_{nYA}$ is not as good as $\mathcal{H}_{YA}$	1
Both are equally good	81
Neither is good	6

and HA only modify internal structures of hierarchy without changing leaf categories. All the methods that change leaf categories outperform the methods that keep leaf categories unchanged. Note that $SMA_{YA|YA}$ significantly outperforms SC. One possible reason could be that $SMA_{YA|YA}$ utilizes the semantics of the original hierarchy in hierarchy modification while SC does not.

We observe that the auxiliary hierarchies employed by SMA have effect on the classification accuracy of the modified hierarchy. Measured by Macro-F_1, $SMA_{YA|YA}$ without using an external hierarchy slightly outperforms its counterpart $SMA_{YA|AB}$ or $SMA_{YA|ODP}$, which uses an external hierarchy; while in terms of Micro-F_1, $SMA_{YA|YA}$ performs worse than do its counterparts.

5.2.3 Experiments on AnswerBag

In this set of experiments, SMA is used to modify the AnswerBag hierarchy. The main purpose is to evaluate whether SMA remains effective when the size of the auxiliary hierarchy is smaller than the one to be modified. Specifically, we use $\mathcal{H}_{YA}$ as auxiliary hierarchy to modify $\mathcal{H}_{AB}$ and evaluate the classification accuracy as we did in the earlier set of experiments. Note that the $\mathcal{H}_{AB}$ has 265 categories which is more than twice of the 115 categories contained in $\mathcal{H}_{YA}$. The parameters λ and θ were set as 0.17 and 0.08, respectively, using the parameter setting approach described earlier. The classification accuracy is reported in Table 3. Observe that $SMA_{AB|YA}$ improves the classification accuracy (Macro- and Micro-F_1) by 30% to 63%, compared with the result obtained before hierarchy modification. This demonstrates that the proposed SMA approach is effective for different hierarchies, even if the size of the auxiliary hierarchy is smaller than the hierarchy to be modified.

5.3 User Study

A good category hierarchy must be semantically meaningful: (i) Its category labels should be easy to understand, facilitating data browsing; and (ii) Its category structure should reflect the topics of its data. We would like to note that it is challenging to evaluate these. We evaluate the modified hierarchy by $SMA_{YA|AB}$ through two types of user study by following the methods [2, 5], respectively. Through the study, we aim to quantify both the appropriateness of the category labels and the structure of the modified hierarchy.

Category Labels. Following a similar setting as in [2], we randomly selected 100 questions from the labeled test set originated from Yahoo! Answers. For each question, we gave the path of the categories in $\mathcal{H}_{YA}$ from the second level category to the leaf category, and similarly the category path from $\mathcal{H}_{nYA}$ (by $SMA_{YA|AB}$). We asked three students to annotate which category path better reflects the topic of the question. Which hierarchy a category path was originated from was not provided to the annotators. Given a question, each volunteer is asked to rate each path from 1(lowest) to 5(highest) based on its quality. hen, we select one of the following choices based on the averaged ratings (r_{YA} and r_{nYA} for the two paths, respectively). (1)$\mathcal{H}_{nYA}$ is better than $\mathcal{H}_{YA}$, if $r_{nYA}, r_{YA} \in [3, 5]$ and $r_{nYA} > r_{YA}$; (2)$\mathcal{H}_{nYA}$ is not as good as $\mathcal{H}_{YA}$, if $r_{nYA}, r_{YA} \in [3, 5]$; (3)Both are equally good, if $r_{nYA}, r_{YA} \in [3, 5]$ and $r_{nYA} = r_{YA}$; (4)Neither is good, if $r_{nYA}, r_{YA} \in [1, 3)$. The statistics of the labels are reported in Table 4. The table shows that the number of questions having better labels in $\mathcal{H}_{nYA}$ is larger than that in $\mathcal{H}_{YA}$ although for majority of questions, the category paths from the two hierarchy are equally good. This result also suggests that the generated labels well reflects the content of categories.

Table 5: Averaged scores of $\mathcal{H}_{YA}$ and $\mathcal{H}_{nYA}$ (by $SMA_{YA|AB}$)

Measure/Hierarchy	$\mathcal{H}_{YA}$	$\mathcal{H}_{nYA}$
Cohesiveness	5.00	6.00
Isolation	4.00	4.67
Hierarchy	5.00	5.33
Navigation Balance	4.50	4.50
Readability	6.00	5.67

Category Structure. Following the evaluation approaches in [5], we evaluate the quality of Yahoo! Answers hierarchy and the modified hierarchy by five measures. *Cohesiveness*: Judge whether the instances in each category are semantically similar. Since it is impractical to read all questions in a large category, we randomly select 50 questions from each category for cohesiveness evaluation. *Isolation*: Judge whether categories on the same level are discriminative from each other.We also use the 50 randomly selected questions to represent each category. *Hierarchy*: Judge whether the concepts represented by the categories become finer from top to bottom. *Navigation Balance*: Judge whether the number of child categories for each internal category is appropriate. *Readability*: Judge whether the concept represented by each category is easy to understand.

We invited three students to evaluate the two hierarchies, $\mathcal{H}_{YA}$ and $\mathcal{H}_{nYA}$ (by $SMA_{YA|AB}$) and assigned scores ranging from 0 to 7 on each measure. The mean of the scores is reported in Table 5.

The cohesiveness of the modified hierarchy is better than the original one. A possible reason is that our approach detected the hidden topics and merged the most similar ones together. The isolation of the modified hierarchy is slightly better. This is probably because the proposed method takes isolation into consideration. To find out the reasons that caused the relatively low isolation of the original hierarchy, we get the list of categories with low scores from the annotators. As an example, a number of questions that are related to motor-cycling were put under Other - Auto Racing by their askers, resulting in low isolation between the two categories. The modified hierarchy does not deteriorate hierarchy quality, navigation balance, and readability of the original hierarchy on average. In summary, the modified hierarchy is of high quality comparable to the original hierarchy generated by domain experts.

5.4 Case Study

As a case study, we select three categories Software, Internet and Hardware from Yahoo! Answers as an example to illustrate the differences before and after modifying $\mathcal{H}_{YA}$. The modified hierarchy $\mathcal{H}_{nYA}$ is by $SMA_{YA|AB}$ utilizing AnswerBag as auxiliary hierarchy.

The two hierarchies are shown in Figure 6. We make the following observations: 1) Different from the original hierarchy, Software and Internet become three categories – Operating System & Application Software, Internet & E-mail and Internet Software. The third category is formed based on the overlapping part of the original two categories, which contains questions about instant messaging

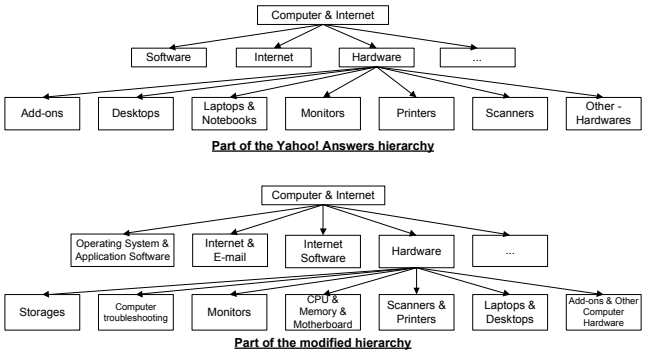


Figure 6: Part of Yahoo! Answers hierarchy and its modified hierarchy.

(IM) and blog software. This demonstrates that the proposed approach can discover and detach the overlapping hidden topics. 2) Two pairs of categories of the original hierarchy, (Laptops & Notebooks and Desktops), and (Printers and Scanners), are merged into two categories in the modified hierarchy, because of the high similarity between the categories within each pair. This shows that categories with high overlap in semantics are merged. 3) For Hardware, some hidden topics are discovered and new categories are formed, like Storage and CPU & Memory & Motherboard, whose questions come from Desktops, Add-ons and Other - Hardwares in the original hierarchy. These newly formed categories are more isolated from each other.

6. CONCLUSION

Category hierarchy plays a very important role in organizing data automatically (through classifiers built on the hierarchy) or manually. However, with newly available documents added into a hierarchy, new topics emerge and documents within the same category become less topically cohesive. Thus the hierarchies suffer from problems of structure irrelevance and semantic irrelevance, leading to poor classification accuracy of the classifiers developed for automatically categorizing the newly available documents into the hierarchy, which in turn leads to poorer document organization. To address these problems, we propose a novel approach SMA to modify a hierarchy. SMA comprises three non-trivial operations (namely, *sprout*, *merge*, and *assign*) to modify a hierarchy. Experimental results demonstrate that SMA is able to generate a modified hierarchy with better classification accuracy improvement over the original hierarchy than baseline methods. Additionally, user study shows that the modified category hierarchy is topically cohesive and semantically meaningful.

7. ACKNOWLEDGEMENTS

Quan Yuan would like to acknowledge the Ph.D. grant from the Institute for Media Innovation, Nanyang Technological University, Singapore. Gao Cong is supported in part by a grant awarded by Microsoft Research Asia and by a Singapore MOE AcRF Tier 1 Grant (RG16/10).

8. REFERENCES

[1] G. Adami, P. Avesani, and D. Sona. Bootstrapping for hierarchical document classification. In *CIKM*, pages 295–302, 2003.

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

[3] R. Agrawal and R. Srikant. On integrating catalogs. In *WWW*, pages 603–612, 2001.

[4] X. Cao, G. Cong, B. Cui, C. S. Jensen, and Q. Yuan. Approaches to exploring category information for question retrieval in community question-answer archives. *ACM Trans. Inf. Syst.*, 30(2):1–38, 2012.

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

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[7] S. C. Gates, W. Teiken, and K.-S. F. Cheng. Taxonomies by the numbers: building high-performance taxonomies. In *CIKM*, pages 568–577, 2005.

[8] A. Koestler. *The Act of Creation*. Penguin Books, New York, 1964.

[9] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *ICML*, pages 170–178, 1997.

[10] K. Punera, S. Rajan, and J. Ghosh. Automatically learning document taxonomies for hierarchical classification. In *WWW (Special interest tracks and posters)*, pages 1010–1011, 2005.

[11] B. Qu, G. Cong, C. Li, A. Sun, and H. Chen. An evaluation of classification models for question topic categorization. *JASIST*, 63(5):889–903, 2012.

[12] G. Siolas and F. d'Alché Buc. Support vector machines based on a semantic kernel for text categorization. In *IJCNN (5)*, pages 205–209, 2000.

[13] A. Sun, E.-P. Lim, and Y. Liu. What makes categories difficult to classify?: a study on predicting classification performance for categories. In *CIKM*, pages 1891–1894, 2009.

[14] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[15] L. Tang, J. Zhang, and H. Liu. Acclimatizing taxonomic semantics for hierarchical content classification from semantics to data-driven taxonomy. In *KDD*, pages 384–393, 2006.

[16] N. Tomuro. Tree-cut and a lexicon based on systematic polysemy. In *NAACL*, 2001.

[17] W. Wei, G. Cong, X. Li, S.-K. Ng, and G. Li. Integrating community question and answer archives. In *AAAI*, 2011.

[18] Y. Yang and X. Liu. A re-examination of text categorization methods. In *SIGIR*, pages 42–49, 1999.

[19] D. Zhang and W. S. Lee. Web taxonomy integration through co-bootstrapping. In *SIGIR*, pages 410–417, 2004.

[20] L. Zhang, S. Liu, Y. Pan, and L. Yang. Infoanalyzer: a computer-aided tool for building enterprise taxonomies. In *CIKM*, pages 477–483, 2004.

APPENDIX

A. PROOF OF LEMMA 1

Lemma 1: The SSE of a category is not smaller than the Total-SSE of its child categories.

Proof: Suppose there is a category C_p with k child categories $\{C_i\}_{i=1}^k$. For a child category C_i, the Sum of Square Distance (SSD) of its data to a data point x is: $SSD(C_i) = \sum_{d \in C_i}(d-x)^2$. We get the minimum value when $x = \frac{1}{|C_i|}\sum_{d \in C_i} d = \overline{c}_i$ which let $\frac{\mathrm{d}}{\mathrm{dx}}\sum_{d \in C_i}(d-x)^2 = 0$. Thus, when x is the mean of data in C_i (or $\overline{c}_i$), the SSD of C_i becomes SSE of C_i, and gets its minimum value. One step further, we have

$$\sum_{d \in C_i}(d-\overline{c}_i)^2 \leq \sum_{d \in C_i}(d-\overline{c}_p)^2,$$

where $\overline{c}_p$ is the mean of data of C_p. This demonstrates that the SSE of C_i is smaller than the SSD of data of C_i to the overall mean, and this result leads to

$$\sum_{i=1}^{k}\sum_{d \in C_i}(d-\overline{c}_i)^2 \leq \sum_{i=1}^{k}\sum_{d \in C_i}(d-\overline{c}_p)^2.$$

This demonstrates that the SSE of a category is not smaller than the Total-SSE of its child categories.