Scalable Construction of Topic Directory with Nonparametric Closed Termset Mining

Hwanjo Yu, Duane Searsmith, Xiaolei Li, Jiawei Han
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
hwanjoyu@uiuc.edu, dsears@ncsa.uiuc.edu, xli10@uiuc.edu, hanj@cs.uiuc.edu

ABSTRACT

A topic directory, *e.g.*, Yahoo directory, provides a view of a document set at different levels of abstraction and is ideal for the interactive exploration and visualization of the document set. We present a method that dynamically generates a topic directory from a document set using a frequent closed termset mining algorithm. Our method shows experimental results of equal quality to recent document clustering methods and has additional benefits such as automatic generation of topic labels and determination of a clustering parameter.

Keywords

topic directory, document clustering, hierarchical clustering

1. INTRODUCTION

A topic directory is a hierarchical document tree or graph structure in which each node has a *topic label* (or a cluster description) and corresponding *documents*. The topic of a higher node conceptually covers its children nodes. For a *static* topic directory, *e.g.*, Yahoo Web directory, the taxonomy is static and manually constructed by the domain experts, and documents are classified into the taxonomy by (non-)automatic classifiers. Such static directories are usually used for organizing and searching targeted documents. On the other hand, *dynamic* topic directories are constructed automatically given a fixed document set or a temporal interest across document sets, *e.g.*, browsing the main news of year 2000 from the AP news data. The directory and topic labels are constructed dynamically (*i.e.*, no preset taxonomy) based on the contents of the document set.

Construction of a dynamic topic directory requires the techniques of *hierarchical document soft-clustering* and *cluster summarization* for constructing topic labels. Recent studies in document clustering show that UPGMA [4] and bisecting k-means [7, 3] are the most accurate algorithms in the categories of agglomerative and partitioning clustering algorithms respectively and outperform other recent hierarchical clustering methods in terms of the clustering quality [7, 4, 3]. However, such clustering methods (1) do not provide cluster descriptions, (2) are not scalable to large document sets (for UPGMA), (3) require the user to decide the number of clusters a priori which is usually unknown in real applications, and (4) focus on hard-clustering (whereas in the real world a document could belong to multiple categories).

Another recent approach is to use frequent itemset mining to construct clusters with corresponding topic labels [2, 5]. This approach first run a frequent itemset mining algorithm, *e.g.*, Apriori [1], to mine frequent termsets from a document set. (An itemset corresponds to a termset, *i.e.*, a set of terms, in this case.) Then they cluster documents based on only the low-dimensional frequent termsets. Each frequent termset serves as the topic label of a cluster. The corresponding cluster consists of the set of documents containing the termset. This method indeed turns out to be as accurate as the other leading document clustering algorithms (*e.g.*, bisecting k-means and UPGMA) in terms of clustering quality [5], and is more efficient since it substantially reduces the dimensions when constructing clusters.

However, this approach introduces another critical issue – *determination of the support threshold*. Since clusters are constructed by frequent termsets and cluster size is the support of termset, the number of clusters (*i.e.*, previously a user parameter) is now determined by the support threshold (*i.e.*, a new user parameter). The support threshold affects the entire cluster processing in terms of the quality and scalability: An over-set threshold could delay the mining time exponentially and also generate too many frequent terms. An under-set threshold could generate a too abstract directory to cover every document in the set. The seemingly best way to adjust the support threshold is to run the mining algorithm multiple times with different support thresholds from small to large, and probe the information about the abstraction level of the directory or the *cluster coverage, i.e., what portion of documents is covered by the clusters*. However, then we would end up losing the benefit of using mining algorithm for document clustering. The entire clustering process becomes unscalable and need tedious manual optimization.

We propose a *nonparametric closed* termset mining method for efficient topic directory construction, which (1) adjusts the support threshold before running the mining algorithm

by introducing an FT-tree (See Section 2.1), and (2) run the most efficient frequent closed termset mining algorithm – CLOSET+ [8]. While the previous clustering methods [5, 2] use all the frequent termsets to construct hierarchical clusters, only *closed* termsets are meaningful in hierarchical clustering (discussed in Section *2.1.3*). We finally present an efficient way to build the soft-clusters from the initial clusters. Soft-clustering is necessary for many applications because a document can belong to multiple clusters. Empirically, our method shows clustering quality as high as most recent document clustering methods but is more efficient. It also naturally produces topic labels for the clusters using frequent closed termsets.

2. SCALABLE CONSTRUCTION OF TOPIC DIRECTORY WITH NONPARAMETRIC CLOSED TERMSET MINING

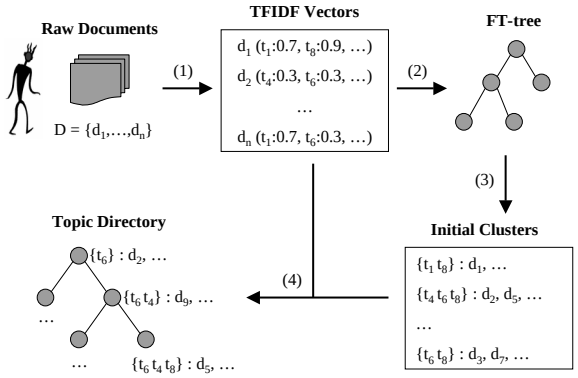

Figure 1: Framework

Every text clustering method preprocesses documents in several steps, such as removing stopwords (*i.e.*, "I", "am", "and") and word-stemming (*i.e.*, merging the same words of different forms like "term" and "terms"). After we preprocess the raw documents (Step (1) in Figure 2), each document can be represented as a vector of the weighted term frequencies, i.e., *term frequency* × *inverse document frequency (TFIDF)*, which the information retrieval community calls a *vector space model*. Our algorithm applies TFIDF. However, in our running examples, we will simply use TF for better understanding.

Starting from this vector space model, we construct FT-tree to mine closed termsets and then construct the initial clusters (Step (2)). Note that termsets are found based on word presence not on the TFIDF. After that, we construct the initial clusters from the FT-tree (Step (3)), which can be done without scanning the TFIDF vectors. The initial clusters are a list of a frequent closed termset with the documents that contain the termset. So, the documents are duplicated in multiple clusters within the initial clusters. When we construct the final topic directory with maximally *max_dup* number of document duplications (Step (4)), we use the original TFIDF vectors to trim the duplication from the initial clusters.

2.1 Nonparametric Closed Termset Mining for Document Clustering

2.1.1 FT-tree Construction

The *FP-tree* published in [6] is a prefix tree with sorted items in which each node contains an item and the support of the itemset from root to path. The FP-tree has proven to be an efficient structure for mining frequent (closed) itemsets [8]. The *FT-tree* is similar to the FP-tree except that the FT-tree includes document IDs in addition. For instance, Figure 2(a) shows the FT-tree constructed from document set D of the table in Fig. 2. The FT-tree can be defined as the FP-tree including document ID at the last node of the corresponding path. Constructing a FT-tree is also similar to constructing a FP-tree except that when we insert a termset representing a document (i.e., a pattern in the FP-tree) into the tree, we insert the document ID at the last node. Note that each document ID will show only once the FT-tree because each document or termset is represented by only one path in the FT-tree, so multiple paths cannot have the same document ID.

2.1.2 Probing Support Threshold

How can we efficiently identify the maximal sup_thr without running a mining algorithm, such that the clusters (i.e., the mined termsets) generated from the sup_thr cover every document in the document set (or cluster coverage = 1.0*)?* To illustrate, consider the FT-tree of Figure 2(a) that is constructed from the table in Fig. 2. We start pruning the tree from the bottom. (Since a FT-tree is a prefix tree with sorted items, as is a FP-tree, the lower nodes contain the items of lower supports.) The item f of $support = 2$, i.e., the two nodes of thick lines in Figure 2(a), will be pruned first. If the pruned nodes contain any document IDs, we pass the IDs to their parents nodes. Thus, the FT-tree after pruning f becomes the tree of Figure 2(b). As you see, the parent nodes a and b now in Figure 2(b) contain the IDs d_1 and d_4 respectively. This means that after we prune a term f, documents d_1 and d_4 – previously covered by termsets $\{e, b, a, f\}$ and $\{b, f\}$ respectively – are now covered by termsets $\{e, b, a\}$ and $\{b\}$. Next, we prune the term a of $support = 3$, i.e., the three nodes of thick lines in Figure 2(b). Then, the tree of Figure 2(b) becomes the tree of Figure 2(c). In other words, documents d_1, d_5 and d_2 – previously covered by termsets $\{e, b, a\}$, $\{e, c, d, a\}$ and $\{c, d, a\}$ respectively – are now covered by termsets $\{e, b\}$, $\{e, c, d\}$ and $\{c, d\}$. When we start pruning the terms b and d of the next higher $support = 4$, i.e., the four nodes of thick lines in Figure 2(c), we find that d_4 will not be covered by any node since its parent is $Null$. Thus, we stop the pruning procedure here, and the maximal *sup_thr* that covers every document is 4.

Note that we can compute this maximal *sup_thr* without actually pruning the tree but not by searching over the tree from the bottom to find the first node whose parent is $Null$. However, showing the "flow" of document IDs in the tree as *sup_thr* increases helps users understand the relations among the *sup_thr*, the covered documents, and the length of the termset that covers the documents. In addition to the shown example, there are certain subtleties. For instance, suppose that a document set contains very few "outlier" documents that do not share any terms with other documents in the set, then the maximal *sup_thr* becomes very low for mined termsets to cover such outlier documents. In such cases, the document coverage information of Table *2.1.2* becomes very useful in determining the proper *sup_thr*. Col-

ID	termset	ordered
d_1	a, b, e, f	e, b, a, f
d_2	a, c, d	c, d, a
d_3	e	e
d_4	b, f	b, f
d_5	a, c, d, e	e, c, d, a
d_6	c, d	c, d
d_7	c, d	c, d
d_8	c, e	e, c
d_9	b, e	e, b
d_{10}	b, e	e, b

Document set D

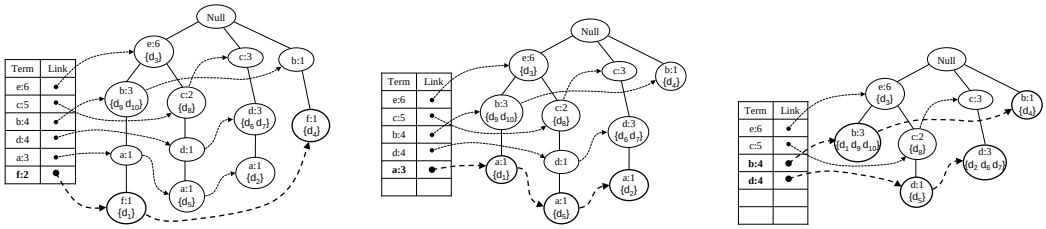

Figure 2: Determining Support

sup	Cov	Not Covered Doc.
4	1.0	
5	0.9	d_4
6	0.6	d_2 d_6 d_7
7	0.0	d_1 d_3 d_5 d_8 d_9 d_{10}

Table 1: Coverage table

Cluster	Doc. IDs
$< e >$	$\{d_1 d_3 d_5 d_8 d_9 d_{10}\}$
$< c >$	$\{d_2 d_5 d_6 d_7 d_8\}$
$< b >$	$\{d_1 d_4 d_9 d_{10}\}$
$< cd >$	$\{d_2 d_5 d_6 d_7\}$

Table 2: Initial clusters

umn "Coverage" in the table denotes the portion of documents that is covered by the corresponding *sup_thr*. Column "Not Covered Doc. IDs" denotes the actual document IDs that are not covered by the *sup_thr*. This coverage table can be efficiently generated from the FT-tree before mining frequent terms.

2.1.3 Mining Closed Termsets from FT-tree

As noted in [8], mining frequent *closed* termsets can lead to orders of magnitude smaller result termsets than mining frequent termsets while retaining the completeness, i.e., from the concise result set, it is straightforward to generate all the frequent termsets with accurate support counts. Closed termsets are meaningful for constructing a topic directory since non-closed termsets are always covered by closed sets.

Since FT-tree subsumes FP-tree, we can simply apply the most recent closed itemset mining algorithm CLOSET+ [8] on an FT-tree. Running CLOSET+ on the FT-tree of Figure 2(c) with $sup\_thr = 4$ generates closed termsets: $< e >$, $< c >$, $< b >$, $< cd >$.

2.2 Constructing Initial Clusters

For each frequent closed termset, we construct an initial cluster to contain all the documents that contain the itemset. Initial clusters are not disjoint because one document may contain several termsets. We will restrain the maximal number of duplications of each document in the clusters in Section 2.3. The termset of each cluster is the *cluster label* – identity of each cluster. Cluster labels also specify the set-containment relationship of the hierarchical structure in topic directory.

Using FT-tree, we do not need to scan the documents to construct the initial clusters while the previous methods [5] do. Document IDs are included in a FT-tree. To retrieve all the documents containing a closed termset, we need to find all the paths containing the termset; the document IDs below the paths are all the documents containing the termset. Figure 2.2 describes the method and rationale for retrieving the document IDs for each closed termset to construct initial clusters. Table *2.1.2* shows the initial clusters constructed from the FT-tree of our running example (Figure 2(c)).

2.3 Topic Directory Construction

- Input: frequent closed termsets
- Output: initial clusters (pairs of termset and document IDs)

Method:

- for each closed termset T
  - for each node t in the sidelink of the last term of T from the header table
    - if the path from the root to t contains the termset T, assign to the termset with the document IDs in and below t

Figure 3: Constructing initial clusters from FT-tree

After initial clusters are constructed, Step (4) builds a topic directory from the initial clusters and the TFIDF vectors. Before building the topic directory, we prune the directory by (1) removing "inner termsets" (Section *2.3.1*) and (2) constraining the maximal number of document duplication (Section *2.3.2*). After that, a topic directory is constructed (Section *2.3.3*) and the first level nodes are finally merged (Section *2.3.4*).

2.3.1 Removing Inner Termsets

Doc	Cluster Labels
d_1	$< e >, < b >$
d_2	$(< c >), < cd >$
d_3	$< e >$
d_4	$< b >$
d_5	$< e >, (< c >), < cd >$
d_6	$(< c >), < cd >$
d_7	$(< c >), < cd >$
d_8	$< e >, < c >$
d_9	$< e >, < b >$
d_{10}	$< e >, < b >$

Table 3: Clusters for each document. termsets within parentheses are inner termsets

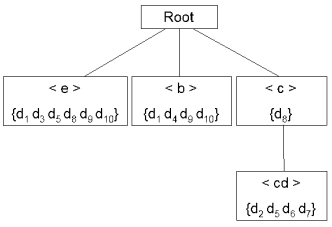

Table 4: Topic directory

If multiple nodes in the same path in a directory contain the same documents, to minimize the document redundancy, we only leave the one in the lowest node and remove the others. This is done by removing "inner termsets" – among frequent closed termsets, the termsets whose superset exists in the same document, e.g., in Table *2.3.1*, termset $< c >$ in document d_2 is an *inner termset* as its superset $< cd >$ also exists in d_2. Removing inner termsets will not cause an empty node in the directory and will not affect the clustering quality.

2.3.2 Constraining Document Duplication

We allow the user to set the maximal number of duplication max_dup of each document in the directory. By allowing the directory to be a graph and $max\_dup \geq 1$, our method naturally supports soft clustering, which is necessary for many applications (*e.g.*, Yahoo directory) because a document can belong to multiple clusters. We refer to the original TFIDF vectors to exclude inferior nodes for each document by applying a heuristic score function such as $score(d, T) = \sum_{t \in T} d \times t$ where $d \times t$ denotes the vector of term t in document d.

2.3.3 Constructing Topic Directory

Constructing a topic directory from the document-cluster list, e.g., Table 2.3.1 with $max\_dup = 2$, can be done in a top-down way.

- Input: nodes (termsets), document-cluster list
- Output: topic directory

Main:

- for m = 1 to maximal length of nodes
  - for node of length = m
    - link(node, m)
- connect document IDs to corresponding nodes using the document-cluster list

link(node, m):

- if $m = 0$, then link node to root,
  else:
  - if there exist inner nodes of length $m - 1$,
    then link the node to them as a child,
    else link(node, m-1)

Figure 4: Constructing topic directory

We start building a directory from the root: link the nodes of length one at the first level, and link the nodes of larger length to their inner nodes as children nodes. Figure 2.3.3 describes the method of constructing a topic directory. The topic directory from Table 2.3.1, i.e., $max\_dup = 2$, is shown in Figure 2.3.1.

2.3.4 Merging the First Level Nodes

Common mining algorithms usually generate a large number of frequent termsets of length one. Thus, a clustering method based on frequent termset mining tends to generate a lot of first level nodes, in which merging the first level nodes helps to provide users with more comprehensible interface. We merge the nodes of high similarity by creating a higher level node between the root and the similar nodes until the total number of the first level nodes becomes less than or equal to a user-specified number. We use a heuristic similarity function as follows:

$$sim(n_1, n_2) = \frac{\#\ of\ common\ documents\ in\ n1\ and\ n2}{\#\ of\ documents\ in\ n1\ and\ n2}$$

3. EXPERIMENT

We compare our method with other recent document clustering methods – agglomerative UPGMA [4], bisecting k-means [7, 3], and those using frequent itemset mining – FIHC [5], HFTC [2]. We used the same evaluation method and the same datasets as used in [5] except that, for Reuters, we do not exclude the articles assigned to multiple categories. Due to space limitations, we report the main results and leave the details to a technical report.

3.0.5 Performance comparison

Dataset	# of clus	TDC	FIHC	Bi k-means	UPGMA
Hitech	3	**0.57**	0.45	0.54	0.33
	15	**0.52**	0.42	0.44	0.33
	30	**0.48**	0.41	0.29	0.47
	60	**0.44**	0.41	0.21	0.40
	Ave.	**0.50**	0.42	0.37	0.38
Re0	3	**0.57**	0.53	0.34	0.36
	15	**0.51**	0.45	0.38	0.47
	30	**0.47**	0.43	0.38	0.42
	60	**0.41**	0.38	0.28	0.34
	Ave.	**0.49**	0.45	0.34	0.40
Wap	3	**0.47**	0.40	0.40	0.39
	15	0.45	0.56	**0.57**	0.49
	30	0.43	0.57	0.44	**0.58**
	60	0.41	0.55	0.37	**0.59**
	Ave.	0.44	0.52	0.45	**0.51**
Classic4	3	0.61	**0.62**	0.59	×
	15	**0.53**	0.52	0.46	×
	30	0.48	**0.52**	0.43	×
	60	0.41	**0.51**	0.27	×
	Ave.	0.50	**0.54**	0.44	×
Reuters	3	**0.46**	0.37	0.40	×
	15	**0.45**	0.40	0.34	×
	30	**0.42**	0.40	0.31	×
	60	**0.40**	0.39	0.26	×
	Ave.	**0.43**	0.39	0.33	×

Table 5: F-measure comparison. # of clus: # of clusters; ×: not scalable to run

Table 3.0.5 shows the overall performance of the four methods on the five data sets. TDC outperforms the other methods on data sets *Hitech*, *Re*0, and *Reuters*, and shows similar performance to FIHC for others. Table 3.0.5 shows the sup_thr of $coverage = 1.0$ determined for each data set.

	Hitech	Re0	Wap	Classic4	Reuters
sup_thr	363/2301	138/1504	333/1560	70/7094	174/10802

Table 6: sup_thr **of** $coverage = 1.0$ # of total document in each data set is within the parentheses.

4. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. Int. Conf. Very Large Databases (VLDB'94)*, pages 487–499, 1994.

[2] F. Beil, M. Ester, and X. Xu. Frequent term-based text clustering. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'02)*, pages 436–442, 2002.

[3] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proc. ACM SIGIR Int. Conf. Information Retrieval (SIGIR'92)*, pages 318–329, 1992.

[4] R. C. Dubes and A. K. Jain, editors. *Algorithms for clustering data*. Prentice Hall, 1998.

[5] B. C. M. Fung, K. Wang, and M. Ester. Herarchical document clustering using frequent itemsets. In *SIAM Int. Conf. Data Mining*, 2003.

[6] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generations. In *Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, 2000.

[7] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techiniques. In *KDD Workshop on Text Mining*, 2000.

[8] J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD'03)*, pages 236–245, 2003.