Hierarchical Clustering for Unsupervised Topical Concept Taxonomy Generation

Abstract

Topic taxonomies can be useful inputs for information retrieval and semantic analysis. Recent research has looked at automatically generating these taxonomies rather than laboriously annotating them by hand. This project experiments with TaxoGen, an existing system for unsupervised topical concept taxonomy generation by hierarchical spherical k-means clustering. We test out three metrics of dynamically selecting k at each taxonomy node by using the Davies-Bouldin Index, the Calinski-Harabasz Index, and cluster silhouette scores. We also propose changes in implementation to reduce document and keyword loss between taxonomy levels to improve taxonomy generation for sparse corpora.

Introduction

Topic extraction and taxonomy generation are important tasks for understanding and organizing text corpora, and their results can also be useful inputs for information retrieval and semantic analysis. A topic taxonomy is a tree where each node is a topic, and where child nodes are hypernyms of the parent. However, many knowledge-base specific taxonomies are laboriously hand-crafted. Recent research has looked at automatically generating these taxonomies.

Many researchers have tackled these two complementary tasks as a two-stage process which involve first extracting topics from a corpus and then constructing a hierarchy between
these topics. (Mao et. al, 2018) Most current research has looked at taxonomy generation for term-level taxonomies, with only a single term at each node in the taxonomy. (Song et. al, 2015; Zafar et. al, 2016)

One class of methods uses static pattern matching to extract taxonomy relations, such as Hearst patterns. One example of a Hearst pattern is “such NP as {NP,}* {(or|and)} NP.” This pattern allows the extraction of “Shakespeare” and “Herrick” as hyponyms of “author” from the phrase “works by such authors as Herrick and Shakespeare” (Hearst 1992). Unfortunately, static pattern matching, though precise, tends to have low recall, as it cannot detect relations between words that do not explicitly match one of the patterns used.

(Semi-)supervised methods form another approach to taxonomy generation. For example, TaxoRL formulates a reinforcement-learning approach that combines term selection and term placement via a policy network, training the agent with cumulative rewards through a holistic tree metric (Mao et al, 2018). However, supervised methods require the collection of a rich set of supervised information from experts, which may be difficult for a specific technical domain.

Hence, in contrast to these prevailing approaches, our project focuses on unsupervised topical concept taxonomy generation. In contrast to a term-level taxonomy, a topical concept taxonomy is a node that contains multiple terms which represent a concept. We choose to use a topical concept taxonomy approach over a term-level taxonomy approach because the latter suffers from high redundancy and limited informativeness. High redundancy occurs because many terms can refer to the same concept—a taxonomy each term is a node will have many redundant expressions of the same concept (e.g. ‘ml’ and ‘machine learning’). Limited informativeness is also a problem because representing a node with a single term causes ambiguity as to what the term means (e.g. ‘bat’ versus (“bat”, ‘flying_squirrel’, ‘flying_fox’)).
We focus on improving an existing unsupervised system, TaxoGen, which uses top-down recursive hierarchical clustering to generate a taxonomy. (Zhang et. al, 2018)

Algorithm 1: Adaptive clustering for topic splitting.

Input: A parent topic \( C \); the number of sub-topics \( K \); the term representativeness threshold \( \delta \).

Output: \( K \) sub-topics of \( C \).

1. \( C_{sub} \leftarrow C \);
2. while True do
   3. \( S_1, S_2, \ldots, S_K \leftarrow \text{spherical-kmeans}(C_{sub}, K) \);
   4. for \( k \) from 1 to \( K \) do
      5. for \( t \in S_k \) do
         6. \( r(t, S_k) \leftarrow \text{representativeness of term } t \text{ for } S_k \);
         7. if \( r(t, S_k) < \delta \) then
            8. \( S_k \leftarrow S_k \setminus \{t\} \);
      9. \( C'_{sub} \leftarrow S_1 \cup S_2 \cup \ldots \cup S_K \);
   10. if \( C'_{sub} = C_{sub} \) then
      11. Break;
   12. \( C_{sub} \leftarrow C'_{sub} \);
3. Return \( S_1, S_2, \ldots, S_K \);

Figure 1: Adaptive clustering in TaxoGen

With a static user-specified \( k \), TaxoGen starts from the taxonomy root (with all documents, all keywords, and all globally trained keyword embeddings) and uses spherical \( k \)-means clustering to group corpus keywords into topic clusters with their corresponding keyword embeddings. For each of the \( k \) topic clusters, TaxoGen removes terms that are not representative (lines 6-8 in Figure 1). The writers call this elimination “adaptive clustering.” TaxoGen then recurs into each topic cluster and performs the clustering again after retraining local word embeddings from the documents in that cluster, because globally trained word embeddings do not have sufficient specificity and discriminative power to perform clustering at lower levels of the taxonomy.
Problem description

TaxoGen thus presents two main innovations for the taxonomy generation problem: adaptive clustering and locally trained embeddings. The existing system, however, faces some issues that have not been resolved.

First, TaxoGen requires the user to specify a static $k$ constant that determines the branching factor of the taxonomy. This is maintained as a constant throughout the taxonomy generation process. This leads to two major concerns. How can the user decide $k$ before running the program? What if the “best” taxonomy structure requires different values of $k$ at different levels of the taxonomy? For instance, a corpus might be “wide but shallow”—a corpus may contain many topics at the root, but the topic branching factor decreases as we descend the taxonomy. Alternatively, a corpus might be “narrow but deep”—containing few topics at the root, but each topic’s branching factor increases as we descend the taxonomy. Motivated by these questions, we experiment with three traditional scoring metrics to select $k$ dynamically at each topic node: the Calinski-Harabasz Index, the Davies-Bouldin Index, and the cluster silhouette scores.

Moreover, we also encountered some problems with TaxoGen’s code when using sparse datasets in practice (the AAN papers dataset). The taxonomy terminated relatively early without recurring to the maximum depth specified for certain branches, as the number of keywords dropped drastically between each level of the taxonomy. We uncovered two issues with the TaxoGen implementation that caused keyword and document loss as the program descends the taxonomy, and experimented with two measures to tackle this problem for sparse datasets.
**Approach**

*Dynamic k-selection for spherical k-means clustering*

We implemented three metrics to select $k$ dynamically at each topic node: the Calinski-Harabasz Index, the Davies-Bouldin Index, and the cluster silhouette score. Instead of specifying a static $k$ for all levels of the taxonomy, the user now specifies a ceiling ($max_k$, inclusive) and floor ($min_k$, inclusive, at least 2) for $k$ and specifies the metric to score the choice of $k$. We perform spherical $k$-means clustering for $k$ from $min_k$ to $max_k$ and pick the $k$ with the highest score from the chosen metric.

All three metrics represent different mathematical formalizations of what a “good” clustering is. Metric scores improve as inter-cluster distance and intra-cluster density increase, corresponding to how a “good” clustering has dense, well-separated clusters.

The silhouette score metric is the mean of all points’ silhouette’s scores. A point’s silhouette score is defined by the following equation:

$$ s = \frac{b - a}{\max (a, b)} $$

where $a$ is the mean distance between a point and all other points in the same cluster, and $b$ is the mean distance between a point and all other points in the next nearest cluster. As such $a$ rewards intra-cluster density while $b$ rewards inter-cluster distance. A high silhouette score corresponds to a “good” clustering.

The Davies-Bouldin (DB) Index is defined by the following equations:

$$ DB = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} R_{ij} $$
\[ R_{ij} = \frac{s_i + s_j}{d_{ij}} \]

where \( d_{ij} \) is the distance between cluster centroid \( i \) and \( j \), and \( s_i \) is the cluster diameter of cluster \( i \), i.e. the average distance between each point of the cluster and the centroid. Intuitively, \( R_{ij} \) represents the “similarity” between two clusters: the term \( s_i \) corresponds to intra-cluster density while \( d_{ij} \) corresponds to inter-cluster distance. The DB Index scores the clustering by taking the average of each cluster’s max similarity with every other cluster. Thus, a low DB Index score corresponds to a “good” clustering because it indicates dissimilar, separable and dense clusters.

Finally, the Calinski-Harabasz (CH) Index is defined by the ratio between the within-cluster dispersion (\( W_k \)), which corresponds to intra-cluster density, and the between-cluster dispersion (\( B_k \)), which corresponds to inter-cluster distance. The CH Index is calculated as such:

\[ CH(k) = \frac{(n - k) B_k}{(k - 1) W_k} \]

where \( n \) is the number of points. The dispersions are defined by:

\[ B_k = \sum_{i=1}^{k} n_i \| G_i - G \|^2 \]
\[ W_k = \sum_{i=1}^{k} \sum_{p \in C_i} \| p - G_i \|^2 \]

where \( G_i \) is the centroid of cluster \( i \), \( G \) is the global centroid, \( n_i \) is the number of elements in cluster \( i \), \( C_i \) is the list of points in cluster \( i \), and \( p \) is a point. Thus, a high CH index indicates a good clustering, because between-cluster dispersion is high while within-cluster dispersion is low. We implemented and tested all the three metrics above with the sklearn Python library.
Reducing document and keyword loss

As discussed before, for the pre-existing TaxoGen implementation by the paper’s authors, document and keyword loss is significant when descending each level of the hierarchy. As a result, a taxonomy may terminate prematurely without recurring to the maximum depth specified for certain branches. This poses a problem for sparse datasets. We discovered that this is an implementation-specific issue.

First, the pre-existing TaxoGen implementation uses hard partitioning to allocate documents to each topic cluster, that is, each document is mapped to only one cluster, after using some metric to determine which cluster a document belongs to. TaxoGen then trains local embeddings when recurring downwards to that specific cluster (see Figure 2). This poses a problem during local embedding training, because two keywords that appear in the same document might belong to two different topic clusters. Since TaxoGen discards keywords in a cluster if they do not exist in the locally trained embeddings, this means that many keywords are discarded because of hard partitioning. Our implementation replaces hard partitioning with soft partitioning: as long as a document contains a keyword in the cluster, it is included in that cluster (see Figure 3).

---

1 The pre-existing TaxoGen implementation is available at https://github.com/franticnerd/taxogen.
Moreover, the pre-existing TaxoGen implementation also does not use all keywords in a cluster to select documents. It first uses adaptive clustering to eliminate keywords that are not “representative” of the cluster below a certain threshold (this is specified in the filter_thre parameter). Then, from the remaining keywords, it picks the top n_expand keywords (a user-
specified parameter) which are the most similar to the cluster topic and then uses these keywords to retrieve documents to consider to be included in the cluster. We remove both constraints to increase document recall in our implementation because of the problem of corpus sparsity. All cluster keywords are used to decide which documents belong in a given cluster. We disable the adaptive clustering module by setting $filter_{thre}$ to 0. We keep the parameter $n_{expand}$ as an option if a user wants to try the original implementation’s approach, but when $n_{expand}$ is set to 0 or a negative integer, all keywords are included.

**Data sets used**

We used the preprocessed DBLP dataset which the original TaxoGen paper used, where key phrases were chunked and extracted via a standard NP-chunker. The dataset includes around 600,000 computer science paper titles and some abstracts from a wide variety of sub-fields. We also used the AAN papers dataset from the Yale LILY Lab, which contains more than 20,000 paper titles and abstracts with regard to natural language processing. (Radev et. al, 2013, Radev et. al, 2009) We preprocessed the corpus in the same method as the original TaxoGen paper, but used the spaCy library for tokenization and NP-chunking. Embeddings were trained globally and locally via Word2Vec, using the skipgram method. For hyperparameters, we set $n_{expand}$ to 0, the maximum recursion depth to 4, and $filter_{thre}$ to 0.
Evaluation method and results

We randomly sampled up to 50 parent-child pairs from each taxonomy and evaluated if each pair is valid. Unfortunately, due to time constraints, we used only one annotator. The results are produced below. While a dynamic $k$-selection policy decreased the validity for the sparser AAN Papers dataset slightly, it increased the size of the generated taxonomy significantly.

<table>
<thead>
<tr>
<th>Method of $k$-selection</th>
<th>Static k ($k = 5$)</th>
<th>CH Index</th>
<th>DB Index</th>
<th>Silhouette Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of valid parent-child pairs</td>
<td>48.4</td>
<td>43.8</td>
<td>39.7</td>
<td>44.0</td>
</tr>
<tr>
<td># of parent-child pairs</td>
<td>31</td>
<td>53</td>
<td>145</td>
<td>66</td>
</tr>
</tbody>
</table>

*Table 1: Results from AAN Papers dataset*

In the case of the DBLP dataset, the silhouette coefficient $k$-selection policy actually had a slightly higher percentage of valid parent-child pairs compared to static $k$. The difference in the number of parent-child pairs in this case should not be taken to be that important because the results from the experiment came from a max taxonomy depth that was set to 4, and the results indicated that further recursion (and taxonomy construction) was possible.

<table>
<thead>
<tr>
<th>Method of $k$-selection</th>
<th>Static k ($k = 5$)</th>
<th>CH Index</th>
<th>DB Index</th>
<th>Silhouette Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of valid parent-child pairs</td>
<td>74.0</td>
<td>67.4</td>
<td>67.4</td>
<td><strong>75.5</strong></td>
</tr>
<tr>
<td># of parent-child pairs</td>
<td><strong>117</strong></td>
<td>28</td>
<td>62</td>
<td>52</td>
</tr>
</tbody>
</table>

*Table 2: Results from DBLP dataset*

---

2 A bug that occasionally appeared with both the old and new implementation is that it generates parent nodes and child nodes with the same term. If such a pair was sampled, we gave it a “N/A” score and removed it from consideration.
It is possible that the tradeoff in accuracy from dynamically selecting $k$ instead of selecting a static $k$ might not be a fair comparison, because we used the chosen static parameter of $k = 5$ from the TaxoGen code, and it could be the case that the researchers tried different values of $k$ and decided that 5 worked best for the DBLP dataset. One could argue that a fairer comparison should use a randomly selected static $k$.

Finally, figure 4 represents a subgraph of the taxonomy generated with the Davies-Bouldin Index $k$-selection metric for illustration purposes.

Figure 4: Section of taxonomy from our implementation using the DB Index and DBLP dataset

**Conclusion**

Our results show that modifying TaxoGen’s implementation to select $k$ dynamically generally results in a slight decrease in accuracy. However, it is possible that this is the tradeoff of automating hyperparameter search in an unsupervised context, where the correctness of the hyperparameter chosen is not immediately apparent until the human evaluation stage. Future
work can be done on making the evaluation process more comprehensive. This could be done via evaluating all taxonomy pairs for validity instead of only randomly sampling up to 50 pairs, and to evaluate topic cohesiveness—whether all the terms in the topic cluster belong in the same cluster.

Since our changes to TaxoGen’s implementation have been to tackle the problem of sparse datasets, it might be possible to use the implementation to generate concept graphs for a given book. Each chapter or paragraph can be treated as a document, and the system would take the corpus of documents as input. Such a task would be useful for generating study guides for books using the concept graph.

Another possible improvement to our current system could be parallelization. The current implementation constructs the taxonomy via a recursive depth-first approach, and as such the runtime is bounded by the number of nodes in the graph. It would be possible to parallelize the hierarchical clustering operation by using a breadth first approach where each thread processes a node, thus, with a sufficient number of threads, the runtime would be bounded by the maximum depth of the taxonomy rather than the number of nodes.
References


Yuning Mao, Xiang Ren, Jiaming Shen, Xiaotao Gu, and Jiawei Han. End-to-end reinforcement learning for automatic taxonomy induction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers), pp. 2462-72.


Zhang, Chao, Fangbo Tao, Xiusi Chen, Jiaming Shen, Meng Jiang, Brian Sadler, Michelle Vanni, and Jiawei Han. "TaxoGen: Constructing Topical Concept Taxonomy by Adaptive Term Embedding and Clustering." *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD 18*, 2018. doi:10.1145/3219819.3220064.
Appendix: Code and Data

The data is available on the Tangra server at:
- /data/clair/pyaan/taxogen_experiments/taxogen/data/dblp
- /data/clair/pyaan/taxogen_experiments/taxogen/data/aanp

The three necessary inputs are papers.txt, embeddings.txt, and keywords.txt

The code used for this implementation is present in the folder with which this report was submitted. For instructions on how to run the code, please view the README.md in the code/ folder.