Automatic Taxonomy Construction from Keywords

Xueqing Liu[§‡], Yangqiu Song[§], Shixia Liu[§], Haixun Wang[§]
{v-xueqli,yangqiu.song,shliu,haixunw}@microsoft.com
[§]Microsoft Research Asia, Beijing, China; [‡]Tsinghua University

ABSTRACT

Taxonomies, especially the ones in specific domains, are becoming indispensable to a growing number of applications. State-of-the-art approaches assume there exists a text corpus to accurately characterize the domain of interest, and that a taxonomy can be derived from the text corpus using information extraction techniques. In reality, neither assumption is valid, especially for highly focused or fast-changing domains. In this paper, we study a challenging problem: Deriving a taxonomy from a set of keyword phrases. A solution can benefit many real life applications because i) keywords give users the flexibility and ease to characterize a specific domain; and ii) in many applications, such as online advertisements, the domain of interest is already represented by a set of keywords. However, it is impossible to create a taxonomy out of a keyword set itself. We argue that additional knowledge and contexts are needed. To this end, we first use a general purpose knowledgebase and keyword search to supply the required knowledge and context. Then we develop a Bayesian approach to build a hierarchical taxonomy for a given set of keywords. We reduce the complexity of previous hierarchical clustering approaches from $O(n^2 \log n)$ to $O(n \log n)$, so that we can derive a domain specific taxonomy from one million keyword phrases in less than an hour. Finally, we conduct comprehensive large scale experiments to show the effectiveness and efficiency of our approach. A real life example of building an insurance-related query taxonomy illustrates the usefulness of our approach for specific domains.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; G.3 [**Probability and Statistics**]: *Nonparametric statistics*; H.2.8 [**Database Management**]: Database applications—*Data mining*

General Terms

Algorithms; Experimentation

Keywords

Knowledgebase, Taxonomy Building, Hierarchical Clustering, Bayesian Rose Tree, Query Understanding

1. INTRODUCTION

Taxonomy plays an important role in many applications. For example, in web search, organizing domain-specific queries into a hierarchy can help better understand the queries and improve search results [23] or help with query refinement [17]. In online advertising, taxonomies about specific domains (e.g., insurance, which is the most profitable domain in online ads) are used to decide the relatedness between a given query and bidding keywords.

Recently, much work has been devoted to taxonomy induction, particularly with respect to automatically creating a domain-specific ontology or taxonomy [14, 15, 16]. The value of automatic taxonomy constructing is obvious: Manual taxonomy construction is a laborious process, and the resulting taxonomy is often highly subjective, compared with taxonomies built by data-driven approaches. Furthermore, automatic approaches have the potential to enable humans or even machines to understand a highly focused and potentially fast changing domain.

Most state-of-the-art approaches for domain-specific taxonomy induction work as follows: First, it selects a text corpus as its input. The assumption is that the text corpus accurately represents the domain. Second, it uses some information extraction methods to extract ontological relationships from the text corpus, and uses the relationships to build a taxonomy. For instance, to derive a medical or biological taxonomy, a commonly used corpus is the entire body of biomedical literature from MEDLINE, life science journals, and online books (e.g., those on PubMed).

Although these text-corpus based approaches have achieved some success, they have several disadvantages. For example, for a highly focused domain, it is very difficult to find a text corpus that accurately characterizes that domain. Intuitively, it is easier to find a text corpus (e.g., the entire set of ACM publications) for big topics such as "computer science," but it is much more difficult to find one for a specific topic such as "big data for business intelligence," as articles about such topics are likely to be dispersed in many different fields and forums. Furthermore, we are often interested in new domains or fast changing domains, which makes it even more difficult to find a characterizing text corpus.

Furthermore, even if we can find a corpus that accurately characterizes the domain, we may still have a problem of data sparsity. Intuitively, the more highly focused the domain, the smaller the text corpus that is available for that domain. The problem is exacerbated by our limited power in understanding natural language text for identifying ontological relationships. To build a taxonomy from a text corpus, we usually bootstrap from a limited set of heuristic patterns. However, high quality patterns typically have very low recalls. For instance, it is well known that Hearst patterns [8] (i.e., "such as" patterns) have a high level of accuracy, but it is unrealistic to assume that the text corpus expresses every ontological relationship in “such as” or its derived patterns, especially when the text corpus is not large enough.

Instead of building taxonomies from a text corpus, we can also consider extracting a domain-specific taxonomy from a big, general purpose taxonomy such as the Open Directory Project (ODP)[1] or Wikipedia[2]. However, a general purpose knowledgebase usually has low coverage on a highly focused domain, and it may also produce ambiguous interpretations for highly specialized terms in the domain.

In this paper, we consider the challenge of inducing a taxonomy from a set of keyword phrases instead of from a text corpus. The problem is important because a set of keywords gives us the flexibility and ease to accurately characterize a domain, even if the domain is fast changing. Furthermore, in many cases, such a set of keywords is often readily available. For instance, search engine companies are interested in creating taxonomies for specific advertising domains. Each domain is described by a set of related ad keywords (bid phrases).

The problem of inducing a taxonomy from a set of keywords has one major challenge. Although by using a set of keywords we can more accurately characterize a highly focused, even fast-changing domain, the set of keywords itself does not contain explicit relationships from which a taxonomy can be constructed. One way to overcome this problem is to enrich the set of keyword phrases by aggregating the search results for each keyword phrase (i.e., throwing each keyword phrase into a search engine, and collecting its top k search result) into a text corpus. Then it treats the text corpus as a bag of words and constructs a taxonomy directly out of the bag of words using some hierarchical clustering approach [5, 18]. The problem with this approach is that the corpus represents the context of the keyword phrases, rather than the conceptual relationships that exist among the keyword phrases. For instance, the search results for the query “auto insurance” contain a very limited number of articles on how “auto insurance” is defined or characterized. The majority of articles either introduce a certain type of car insurance or talk about car insurance in passing. From such articles, we can extract context words such as “Florida,” “Fords,” or “accident.” The resulting context can be arbitrary and not insurance related.

To tackle this challenge, we propose a novel, “knowledge+context” approach for taxonomy induction. We argue that, in order to create a taxonomy out of a set of keywords, we need knowledge and contexts beyond the set of keywords itself. For example, given two phrases “vehicle insurance” and “car insurance,” humans know immediately that “car insurance” is a sub-concept of “vehicle insurance,” because humans have the knowledge that a car is a vehicle. Without this knowledge, the machine can hardly derive this relationship unless the extended corpus from the keywords (e.g., using keyword search) happens to describe this relationship in a syntactic pattern that the machine can recognize (e.g., “vehicles such as cars”). But the context is also important. It is unlikely that the knowledgebase (especially a general purpose one) knows about every subsuming relationship in the specific domain. For example, it has no idea that x is a sub-concept of y. However, through the context, we may find that x is highly related to z, and in the knowledgebase z is a sub-concept of y. Thus, using knowledge and contexts, we can establish the relationship between x and y.

Incorporating both knowledge and contexts in taxonomy building is not easy. To this end, we formalize our approach as hierarchical clustering over features generated from a general knowledgebase and search context. More specifically, we first deduce a set of concepts for each keyword phrase using a general purpose knowledgebase [19] and then obtain its context information from a search engine. After enriching the keyword set using knowledge and contexts, we make use of Bayesian-based hierarchical clustering to automatically induce a new taxonomy. Our paper presents three technical contributions: First, we illustrate how to derive the concepts and context from each keyword. Second, we present the way to formulate the taxonomy building to a hierarchical clustering problem based on their concepts and context. And finally, we scale up the method to millions of keywords.

The rest of the paper is organized as follows. In Section 2, we introduce the background of hierarchical clustering and Bayesian rose trees. We then present algorithms, analyze their complexity, and discuss some implementation issues in Section 3. Section 4 describes experiments that demonstrate the effectiveness and efficiency of our approach. In Section 5, we conclude our work and present several future works.

2. MULTI-BRANCH CLUSTERING

Hierarchical clustering is a widely used clustering method [9]. The advantage of hierarchical clustering is that it generates a tree structure (a dendrogram) which is easy to interpret. Two strategies are used for hierarchical clustering, agglomerative and divisive, where the agglomerative approach builds a tree from bottom up by combining the two most similar clusters at each step, and the divisive approach builds a tree from top down by splitting the current cluster into two clusters at each step.

Traditional hierarchical clustering algorithms construct binary trees. However, binary branches may not be the best model to describe a set of data’s intrinsic structure in many applications. Fig. 1 gives an example. The goal here is to create a taxonomy from a set of insurance-related keyword phrases. It assumes we have a good similarity measure that enables us to group related keywords in an optimal way. Still, the outcome might be sub-optimal or unnatural. In the figure, “indiana cheap car insurance,” “kentucky cheap car insurance,” and “missouri cheap car insurance” are on different levels, but apparently they belong to the same cluster. This disadvantage is introduced by the model not the similarity measure, which means that no matter how accurately we understand the data, the result will still be sub-optimal.

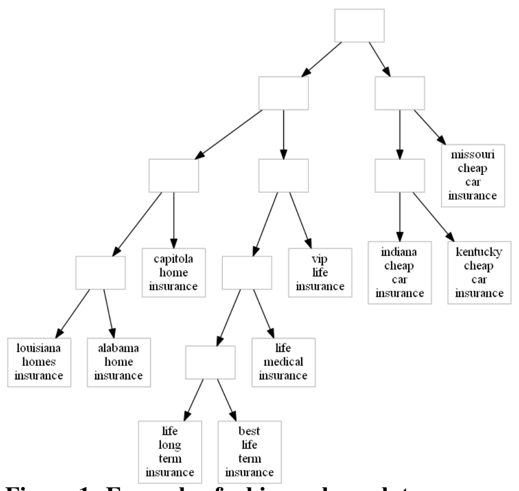


Figure 1: Example of a binary-branch taxonomy.

To remedy this, multi-branch trees are developed. Compared with binary trees, they have a simpler structure and better interpretability. An example of multi-branch clustering is shown in

[1]http://www.dmoz.org/
[2]http://www.wikipedia.org/

Fig. 2, where nodes such as "indiana cheap car insurance," "kentucky cheap car insurance," and "missouri cheap car insurance" are grouped under the same parent node.

Currently, there are several multi-branch hierarchy clustering approaches [1, 3, 10]. The methods proposed by Adams et al [1] and Knowles et al [10] are based on the Dirichlet diffusion tree, which use an MCMC process to infer the model. Blundel et al [3] adopts a simple, deterministic, agglomerative approach called BRT (Bayesian Rose Tree). In this work, we adopt BRT as a basic algorithm for taxonomy induction. The major steps of BRT are shown in Algorithm 1.

Algorithm 1 is a greedy agglomerative approach. In the beginning, each data point is regarded as a tree on its own: $T_i = \{\mathbf{x}_i\}$ where $\mathbf{x}_i$ is the feature vector of ith data. For each step, the algorithm selects two trees T_i and T_j and merges them into a new tree T_m. Unlike binary hierarchical clustering, BRT uses three possible merging operations [3]:

- **Join:** $T_m = \{T_i, T_j\}$, that is, T_m has two child nodes.
- **Absorb:** $T_m = \{\text{children}(T_i) \cup T_j\}$, that is, T_m has $|T_i| + 1$ child nodes.
- **Collapse:** $T_m = \{\text{children}(T_i) \cup \text{children}(T_j)\}$, that is, T_m has $|T_i| + |T_j|$ child nodes.

Specifically, in each step, Algorithm 1 greedily finds two trees T_i and T_j to maximize the ratio of probability:

$$\frac{p(\mathcal{D}_m|T_m)}{p(\mathcal{D}_i|T_i)p(\mathcal{D}_j|T_j)} \quad (1)$$

where $p(\mathcal{D}_m|T_m)$ is the likelihood of data $\mathcal{D}_m$ given the tree T_m, $\mathcal{D}_m$ is all the leaf data of T_m, and $\mathcal{D}_m = \mathcal{D}_i \cup \mathcal{D}_j$. The probability $p(\mathcal{D}_m|T_m)$ is recursively defined on the children of T_m:

$$p(\mathcal{D}_m|T_m) = \pi_{T_m} f(\mathcal{D}_m) + (1 - \pi_{T_m}) \prod_{T_i \in \text{Children}(T_m)} p(\mathcal{D}_i|T_i) \quad (2)$$

where $f(\mathcal{D}_m)$ is the marginal probability of the data $\mathcal{D}_m$ and π_{T_m} is the "mixing proportion." Intuitively, π_{T_m} is the prior probability that all the data in T_m is kept in one cluster instead of partitioned into sub-trees. In BRT [3], π_{T_m} is defined as:

$$\pi_{T_m} = 1 - (1 - \gamma)^{n_{T_m} - 1} \quad (3)$$

where n_{T_m} is the number of children of T_m, and $0 \leq \gamma \leq 1$ is the hyperparameter to control the model. A larger γ leads to coarser partitions and a smaller γ leads to finer partitions.

The major cost of this bottom-up hierarchy construction approach is dominated by the following two steps:

(1) looking for pairs of clusters to merge;

(2) calculating the likelihood associated with the merged cluster (Eq. (1)).

Assume in the current round there are c clusters. The two steps above take $O(c^2)$ time. At the start of the algorithm, we have $c = n$, the number of data points. For all the clusters merging into one cluster, if we first compute the likelihood by Eq. (1), and sort them before we search for merging pairs, it will take $O(n^2 \cdot C_V + n^2 \log n)$ time complexity, where C_V is the maximum number of non-zero elements in all the initial vectors $\mathbf{x}_i$'s. For high-dimensional text data, the document will have hundreds of words on average. Therefore, C_V cannot be ignored when we analyze the overall complexity compared to $\log n$. This is not applicable for any large-scale data set. Moreover, this approach will have $O(n^2)$ memory cost. For 100,000 data points, this will take $3 \times 8 \times 10^{10} = 240G$ bytes memory to contain all the pairs' likelihood, if we use 8 bytes to store double-precision value and have three types of merging likelihoods ("join," "absorb," and "collapse").

Algorithm 1 Bayesian Rose Tree (BRT).

```
Input: A set of documents $\mathcal{D}$.
$T_i \leftarrow \mathbf{x}_i$ for $i = 1, 2, \cdots, n$
$c \leftarrow n$
while $c > 1$ do
    1. Select $T_i$ and $T_j$ and merge them into $T_m$ which maximizes
        $L(T_m) = \frac{p(\mathcal{D}_m|T_m)}{p(\mathcal{D}_i|T_i)p(\mathcal{D}_j|T_j)}$,   (4)
        where the merge operation is join, absorb, or collapse.
    2. Replace $T_i$ and $T_j$ with $T_m$ in the tree.
    3. $c \leftarrow c - 1$
end while
```

3. KEYWORD TAXONOMY BUILDING

In this section, we introduce our approach for building a domain specific taxonomy from a set of keyword phrases augmented with knowledge and contexts. First, we obtain knowledge and contexts related to the keywords. In our approach, we obtain knowledge (concepts that correspond to each keyword phrase) using a technique called short text conceptualization [19] and a general purpose knowledgebase called Probase [11, 24], and we obtain contexts by submitting the queries to a commercial search engine to retrieve all snippet words. Second, we build the taxonomy using a multiple branch hierarchical clustering approach. In this section, we initially introduce how to leverage concepts and contexts, and then we present the methods that speed up the taxonomy building for a large set of keywords.

3.1 Knowledge and Context

We use information obtained from a general-purpose knowledgebase and a search engine to augment the given set of keyword phrases.

The knowledgebase we use is Probase [11, 24], which has been demonstrated useful for Web search [21, 22]. The core of Probase consists of a large set of *isa* relationships, which are extracted from a text corpus of 1.68 billion web pages. For example, "... Spanish artists such as Pablo Picasso ..." is considered a piece of evidence for the claim that "Pablo Picasso" is an instance of the concept *Spanish artist*. Probase also contains other information. For example, for each concept, it contains a set of attributes that describe the concept. One unique feature of Probase is the broadness of its coverage. Probase contains millions of concepts, from well-known ones such as "country" and "artists" to small but concrete ones such as "wedding dress designers" and "renewable energy techniques." The richness of Probase enables us to identify and understand millions of concepts people use in their daily communication.

The knowledgebase facilitates us in deriving concepts from keyword phrases and we can then use the concepts to enrich the keyword phrase. For instance, given "microsoft and apple," we derive concepts such as *IT companies*, *big companies*, etc., and given "apple and pear," we derive concepts such as *fruit* or *tree*. However, these are still not enough for understanding the keyword phrase. We need to enhance the knowledgebase in two aspects.

- In order to model text for inferencing, we need to make the knowledge in Probase probabilistic. To this end, we introduce a set of probabilistic measures. For example, $P($*instance*

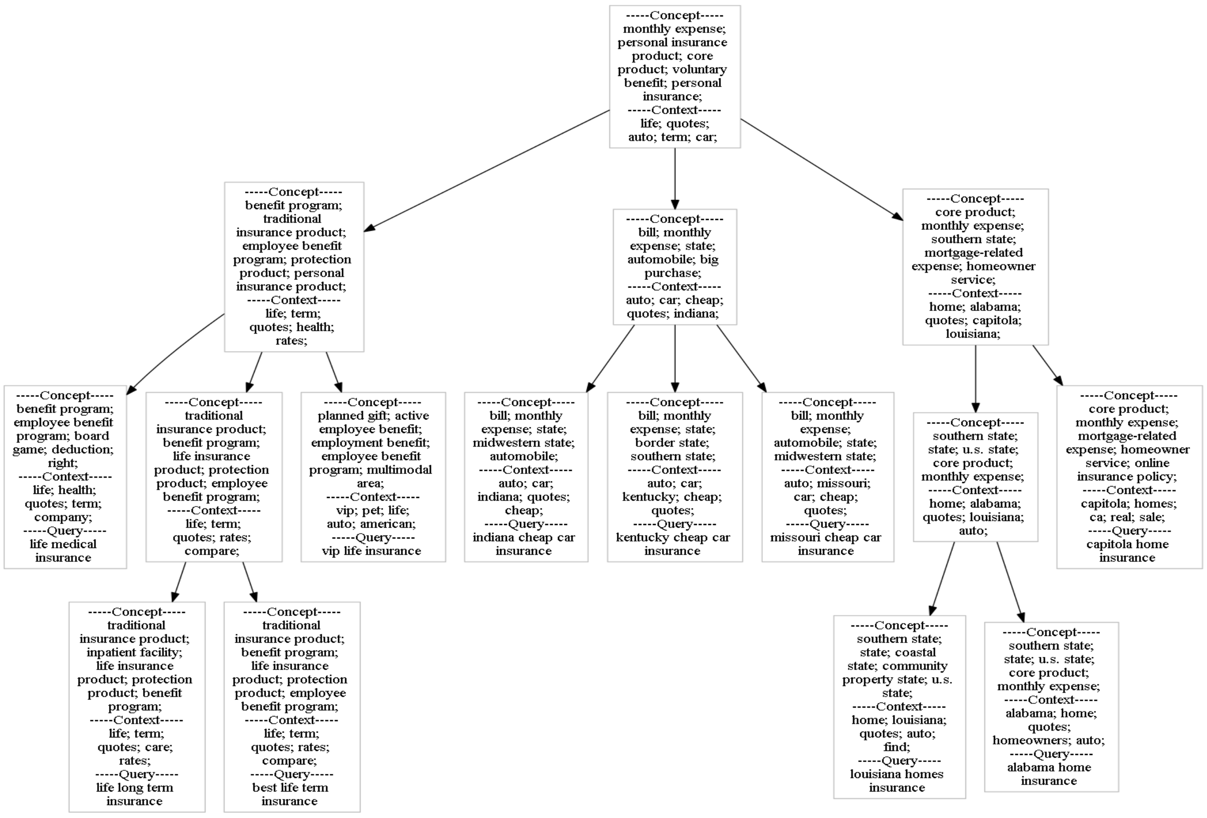

-----Concept----- monthly expense; personal insurance product; core product; voluntary benefit; personal insurance; -----Context----- life; quotes; auto; term; car;
-----Concept----- benefit program; traditional insurance product; employee benefit program; protection product; personal insurance product; -----Context----- life; term; quotes; health; rates;
-----Concept----- bill; monthly expense; state; automobile; big purchase; -----Context----- auto; car; cheap; quotes; indiana;
-----Concept----- core product; monthly expense; southern state; mortgage-related expense; homeowner service; -----Context----- home; alabama; quotes; capitola; louisiana;
-----Concept----- benefit program; employee benefit program; board game; deduction; right; -----Context----- life; health; quotes; term; company; -----Query----- life medical insurance
-----Concept----- traditional insurance product; benefit program; life insurance product; protection product; employee benefit program; -----Context----- life; term; quotes; rates; compare;
-----Concept----- planned gift; active employee benefit; employment benefit; employee benefit program; multimodal area; -----Context----- vip; pet; life; auto; american; -----Query----- vip life insurance
-----Concept----- bill; monthly expense; state; midwestern state; automobile; -----Context----- auto; car; indiana; quotes; cheap; -----Query----- indiana cheap car insurance
-----Concept----- bill; monthly expense; state; border state; southern state; -----Context----- auto; car; kentucky; cheap; quotes; -----Query----- kentucky cheap car insurance
-----Concept----- bill; monthly expense; automobile; state; midwestern state; -----Context----- auto; missouri; car; cheap; quotes; -----Query----- missouri cheap car insurance
-----Concept----- southern state; state; u.s. state; core product; monthly expense; -----Context----- home; alabama; quotes; louisiana; auto;
-----Concept----- core product; monthly expense; mortgage-related expense; homeowner service; online insurance policy; -----Context----- capitola; homes; ca; real; sale; -----Query----- capitola home insurance
-----Concept----- traditional insurance product; inpatient facility; life insurance product; protection product; benefit program; -----Context----- life; term; quotes; care; rates; -----Query----- life long term insurance
-----Concept----- traditional insurance product; benefit program; life insurance product; protection product; employee benefit program; -----Context----- life; term; quotes; rates; compare; -----Query----- best life term insurance
-----Concept----- southern state; state; coastal state; community property state; u.s. state; -----Context----- home; louisiana; quotes; auto; find; -----Query----- louisiana homes insurance
-----Concept----- southern state; state; u.s. state; core product; monthly expense; -----Context----- alabama; home; quotes; homeowners; auto; -----Query----- alabama home insurance

Figure 2: Example of Multi-branch Query Taxonomy.

$|concept)$ tells us how typical the *instance* in the given *concept* is. Intuitively, knowing that both "robin" and "penguin" are birds is not enough. We need to know "robin" is a much more typical bird than "penguin," that is, when people talk about birds, it is more likely that they are thinking about a robin than a penguin. Such information is essential for understanding the intent behind a piece of short text. Besides $P(instance|concept)$ we also obtain $P(concept|instance)$, $P(concept|attribute)$, and $P(attribute|concept)$. These values are calculated during the information extraction process, for example:

$$P(instance|concept) = \frac{n(instance, concept)}{n(concept)} \quad (5)$$

where $n(instance, concept)$ denotes the number of times that the*instance* and *concept* co-occur in the same sentence in a given corpus, and $n(concept)$ is the frequency of the *concept*.

- A keyword phrase may be syntactically and semantically complicated and require sophisticated chunking and parsing to identify meaningful terms. As shown in Fig. 3, we conceptualize "indiana cheap car insurance" by first recognizing the terms "indiana," "cheap car," and "car insurance" that appear in Probase, and then we derive concepts such as *state*, *car insurance*, *bill*, etc. Furthermore, each concept is associated with a probability score that indicates the strength of the concept. A detailed description of the conceptualization techniques can be found in [19].

Although Probase contains millions of concepts, we cannot expect it to cover everything. To remedy this, we collect the "context" of a keyword phrase, and use the context to supplement the

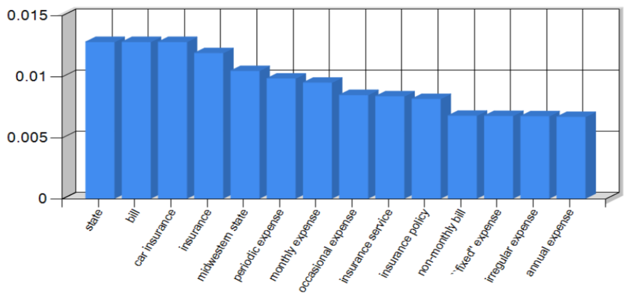


Figure 3: Conceptualization of "indiana cheap car insurance".

concepts related to the phrase. To obtain the context, we submit the keyword phrase to a search engine, and collect the top ten snippets in the search results. The context is represented as a bag of words. The quality of the context is much lower than that of the concepts obtained from Probase, in the sense that for a given word in the bag-of-words context, we do not know its semantic relationship with the keyword. But still, they can be useful especially when the concepts we obtained are insufficient. As an example, consider two keyword phrases, "www.monster.com" and "monster.com." Probase knows that "monster.com" is a job site, but it knows nothing about "www.monster.com." Thus, conceptualization will report that the two phrases have zero similarity. Through the search engine, we find that "www.monster.com" is associated with phrases such as *web site, online job site, job board*, etc. Thus, by adding the context information, the query containing "www.monster.com" will have greater similarity to the query containing "monster.com."

3.2 Text Modeling

To cluster the data into a hierarchy, we first need to model the data. This corresponds to calculating $f(\mathcal{D})$ in Eq.(2), the marginal distribution of data $\mathcal{D}$ (Section 2). The original BRT approach [3] assumes that the data can be modeled by a set of binary features that follow the Bernoulli distribution. In other words, features are not weighted. In our approach, we use features that consist of concepts and contexts to represent the data. Since even a short piece of text may contain multiple topics or concepts, it is important to rank them by their significance. Thus, unlike BRT, we incorporate weights into the marginal distribution $f(\mathcal{D})$.

Given a set of keyword phrases $\{keyword_1, \cdots, keyword_n\}$, we derive a list of $(term, weight)$ pairs for each keyword, where the *term* is either a concept produced by the knowledgebase, or a context word generated by search, and the *weight* is derived as follows:

$$weight_j = \lambda \cdot freq(term_j) + (1-\lambda) \cdot \sum_i^n C_i \cdot P(term_j|keyword_i)$$

where λ is a parameter that controls how much we value the context compared to concepts; $freq(term_j)$ is the frequency of a term j in the context derived from search results; $P(term_j|keyword_i)$ is the probability of the term as a concept given keyword phrase $keyword_i$, and is provided by the knowledgebase (as in Eq (5)); C_i is the frequency of $keyword_i$ in the knowledgebase, and $C_i \cdot P(term_j|keyword_i)$ is used as the frequency of the term as a concept. We then set the feature vector $\mathbf{x}_i$ with the term frequencies $weight_j$'s for $keyword_i$.

In the hierarchical clustering algorithm, once two keywords or keyword clusters are grouped together, the grouped keyword cluster can contain multiple topics. For example, initially we have four keywords "China," "India," "Germany," and "France." Although these four keywords share some common concepts such as *country* and *nation*, we can still distinguish them based on the concepts with smaller weights. First, "China" and "India" will be grouped together since they share many concepts like *Asian country* and *emerging market*. "Germany" and "France" will also be grouped together because they share concepts like *European country* and *Western nation*. After that, these two clusters will be grouped together. The final cluster actually contains multiple topics, i.e. both *Asian country* and *European country*. Therefore, we need a distribution to better capture this characteristic.

To this end, we use the DCM distribution [13] to represent the marginal distribution $f(\mathcal{D})$. DCM is derived based on multinomial and Dirichlet conjugate distributions. Multinomial distribution can naturally characterize the co-occurrence counts of terms, while the prior distribution, i.e., Dirichlet distribution, can be regarded as smoothing over counts. The generative process of a document underlying this modeling is that we first sample a multinomial distribution from Dirichlet distribution, and then sample a document based on the multinomial distribution. Multinomial distribution can be regarded as a document-specific sub-topic distribution, which makes certain words appear more likely in a particular document [13]. DCM integrates out the intermediate multinomial distribution, and thus it represents either more general topics or multiple topics. In hierarchical clustering, we incrementally merge clusters. Therefore, DCM is more appropriate for evaluating whether two clusters (with multiple topics) should be merged.

Specifically, the likelihood of multinomial distribution $p(\mathbf{x}|\boldsymbol{\theta})$ is defined by:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \frac{m!}{\prod_j^V x^{(j)}!} \prod_{j=1}^V p(x^{(j)}|\boldsymbol{\theta}) = \frac{m!}{\prod_j^V x^{(j)}!} \prod_{j=1}^V [\theta^{(j)}]^{x^{(j)}}, \quad (6)$$

Algorithm 2 Nearest-neighbor-based BRT.

```
Input: A set of documents $\mathcal{D}$.
Initialization 1: Set $T_i = \mathbf{x}_i$ for $i = 1, 2, \cdots, n$; number of clusters $c = n$.
Initialization 2: Find the nearest neighbors $\mathcal{N}(T_i)$ for each cluster, and
compute all the likelihood scores.
while $c > 1$ do
    1. Find $T_i$ and $T_j$ in all neighborhood sets $\{\mathcal{N}_k(T_i)\}$, whose merge
    maximizes Eq. (1), where $m \in$ {join, absorb, collapse}.
    2. $T_m \leftarrow$ the result of merge on $T_i$ and $T_j$
    3. Delete $T_i$ and $T_j$.
    4. Find the nearest neighbors set $\mathcal{N}(T_m)$ for the new cluster.
    5. $c \leftarrow c - 1$
end while
```

where V is the vocabulary size, $x^{(j)}$ is the frequency of term $v^{(j)}$, $m = \sum_j^V x^{(j)}$, and $\boldsymbol{\theta} = (\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(V)})^T \in \mathcal{R}^V$ are the parameters of multinomial distribution.

The Dirichlet distribution prior is:

$$p(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \frac{\Gamma(\sum_{j=1}^V \alpha^{(j)})}{\prod_{j=1}^V \Gamma(\alpha^{(j)})} \prod_{j=1}^V [\theta^{(j)}]^{\alpha^{(j)}-1} = \frac{1}{\Delta(\boldsymbol{\alpha})} \prod_{j=1}^V [\theta^{(j)}]^{\alpha^{(j)}-1}, \quad (7)$$

where $\boldsymbol{\alpha} = (\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(V)})^T \in \mathcal{R}^V$, and the Gamma function[3] has the property $\Gamma(x+1) = x\Gamma(x)$. The "Dirichlet delta function" $\Delta(\boldsymbol{\alpha}) = \frac{\prod_{j=1}^V \Gamma(\alpha^{(j)})}{\Gamma(\sum_{j=1}^V \alpha^{(j)})}$ is introduced for convenience.

Then the marginal distribution $f(\mathcal{D})$ is given by:

$$f_{DCM}(\mathcal{D}) = \int_{\boldsymbol{\theta}} \prod_i^n p(\mathbf{x}_i|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\boldsymbol{\alpha}) d\boldsymbol{\theta} = \prod_i^n \frac{m!}{\prod_j^V x_i^{(j)}!} \cdot \frac{\Delta(\boldsymbol{\alpha} + \sum_i \mathbf{x}_i)}{\Delta(\boldsymbol{\alpha})}. \quad (8)$$

Using this marginal distribution $f_{DCM}(\mathcal{D})$, we can seamlessly integrate the weights into the term feature vector. Next we will introduce how to construct the tree more efficiently.

3.3 Efficient Taxonomy Construction

As we show in Section 2, BRT has time complexity $O(n^2 \cdot C_V + n^2 \log n)$, and space complexity $O(n^2)$, which is not applicable to large-scale problems. The most time consuming process in agglomerative clustering lies in searching all candidate cluster pairs to find the best pairs to merge. If we can reduce the search space for BRT by pruning the pairs of keyword clusters that are most unlikely to be merged, then we can reduce the cost of searching agglomerative clustering. We propose to "cache" a set of nearest neighbors for each data point. Then the search complexity only depends on the number of the nearest neighbors. However, searching for nearest neighbors still incurs a cost of $O(n)$ for each data point. The time complexity of finding k nearest neighbors can be reduced to $O(\log n)$ using techniques such as KD-trees [2] and Metric trees [20]. However, these techniques are not suitable for high-dimensional data, since they partition the data space dimension by dimension. Approximation methods such as LSH (Locality-Sensitive Hashing) [7] can be used when the number of dimensions is large. Particularly, Spilltree [12] relaxes the non-overlapping constraints of Metric trees and incorporates the technique used in LSH, and thus combines the benefits of both methods.

In this section, we focus on adapting two major types of nearest neighbor approaches for efficient taxonomy construction: k-nearest-neighbor (kNN) and ϵ-ball-nearest-neighbor (ϵNN) [4]. kNN finds k nearest neighbors for each data and is not concerned the density of the data. ϵNN uses a spherical ball to bind all the nearest

[3]For integer variables, Gamma function is $\Gamma(x) = (x-1)!$. For real numbers, it is $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$.

neighbors within the ball. In our taxonomy construction approach, we significantly reduce the time complexity by using methods such as Spilltree [12] and PPJoin+ [25].

*3.3.1 k*NN-*Approximation*

We introduce two kNN-based approximation approaches based on BRT. A flowchart for using the nearest neighbors to approximate the construction procedure of BRT is shown in Algorithm 2.

***k*NN-BRT:** Using the kNN approach, we first find the k nearest neighbors for each data point, and then we check the possibility of merging within the neighborhood set. We denote $\mathcal{N}_k(\mathbf{x})$ as the k nearest neighbor set of data $\mathbf{x}$. To find k nearest neighbors of a data, we keep a minheap with size k to maintain the data with largest similarities scores. When new data comes and the similarity score is larger than the top value (the smallest one in the minheap), we replace the top index with the new data. Compared to BRT, the space cost is significantly reduced from $O(n^2)$ to $O(nk)$. The time complexity is also reduced to $O(n^2 \cdot C_V + n^2 \log k)$.

Spilltree-BRT: Using the k nearest neighbors to construct BRT is still time consuming. We then use the Spilltree algorithm [12] to further reduce the time complexity.

Spilltree is a generalization of metric trees [20]. Metric trees partition the data space into binary trees, and retrieve the nearest neighbors by DFS (depth first search). Metric trees will be less efficient when the number of dimensionality is large (e.g., larger than 30 [12]). The Spilltree algorithm offers two major modifications:

1. Introduce random projection before partitioning the data space.

2. Introduce the overlapping/non-overlapping regions between nodes when partitioning each sub-tree. While searching for the nearest neighbors in the sub-trees with overlapping partitions, Spilltree searches one branch only.

According to the Johnson-Lindenstrauss Lemma [6], embedding a data set of dimension n to an $O(\log n)$ dimensional space has little distortions for pairwise distances. As a result, brute-force search in the projected space provides a $(1+\varepsilon)$-NN [12] in the original space. Thus, by projecting the data onto a much lower dimensional space, high precision can be guaranteed while the time cost is significantly reduced, especially when the original data has millions of dimensions.

Moreover, original metric trees perform a DFS to find the nearest neighbors. By introducing the overlapping nodes, Spilltree adopts a combined strategy of a defeatist search and DFS. The defeatist search may fail for non-overlapping nodes if a query and its nearest neighbors belong to different branches. However, it is guaranteed to be successful for the overlapping nodes when the shortest distance in the overlapping regions is larger than or equal to the distance between the query and its nearest neighbor. By setting an appropriate tolerance parameter τ, the accuracies of both overlapping and non-overlapping nodes are ensured [12]. Overall, the time complexity of search for Spilltree is $O(\log n)$ [12].

The random projection to d-dimensional space has the time complexity of $O(nd \cdot C_V)$. Building a Spilltree costs $O(nd \log n)$. To use Spilltree to search the k nearest neighbors, we also keep a minheap to maintain k data points when traversing the Spilltree. This step will cost $O(nd \log n \log k)$ time for all the data. In summary, using Spilltree to build BRT costs $O(nd \cdot C_V + nd \log n \log k)$. Compared to the kNN-BRT algorithm, using Spilltree will cost additional $O(Vd + nd)$ memory to store the random projection matrix and the Spilltree.

Table 1: Comparison of computational complexity and memory requirements of different algorithms. (C_V is the number of non-zero elements in the vector x and $L = \sum_j x^{(j)}$)

Algorithm	Time complexity	Memory requirement
BRT	$O(n^2 \cdot C_V + n^2 \log n)$	$O(n^2)$
kNN-BRT	$O(n^2 \cdot C_V + n^2 \log k)$	$O(nk)$
Spilltree-BRT	$O(nd \cdot C_V + nd \log n \log k)$	$O(nk + Vd + nd)$
PPJoin-BRT	$O(n^2[(1-\epsilon^2)L + \log L])$	$O(nf(\epsilon) + nL)$

*3.3.2 ϵ*NN-*Approximation*

In ϵNN-approximation, for each data point, we keep its nearest neighbors whose similarity with the data point is larger than a pre-defined threshold ϵ. This reduces the time complexity to $O(n^2 \cdot C_V)$. The storage of ϵNN depends on the number of the neighbors that satisfy the ϵ threshold. However, we might need to re-run the ϵNN algorithm in order to ensure we can find candidates to be merged, since when the threshold ϵ is too large, ϵNN will not return any nearest neighbors.

PPJoin-BRT: To support ϵNN-approximation, we need to find ϵ neighbors of a data point efficiently. We adopt the PPJoin+ [25] approach for this purpose. PPJoin+ uses two types of filtering, prefix filtering and suffix filtering, to filter out the data points that do not satisfy certain constraints. In prefix filtering, it has been proven that the cosine similarity is larger than a threshold ϵ if and only if the number of overlapped terms between the two sets is larger than $\epsilon' = \lceil \epsilon \sqrt{L_i \cdot L_j} \rceil$, where $L_i = \sum_k x_i^{(k)}$ is the length of the document $\mathbf{x}_i$. Therefore, we can quickly filter out pairs of documents as if their overlap is larger than ϵ'. The time complexity of this step is reduced to $(1-\epsilon^2)\sum_j x^{(j)}, \epsilon < 1$ [25]. In suffix filtering, it first derives an upper bound of hamming distance H_{max} corresponding to the pre-defined threshold ϵ. Then we filter out the data if the lower bound of the hamming distance between two documents $H_{min}(\mathbf{x}_i, \mathbf{x}_j)$ is larger than the upper bound H_{max}. We also implemented an algorithm based on the binary search for the lower bound of the hamming distance. The overall time complexity of PPJoin+ is $O(n^2[(1-\epsilon^2)L + \log L])$, where L is the average length of documents. PPJoin-BRT takes $O(nf(\epsilon))$ memory to store the likelihood values of the nearest neighbors. Moreover, it needs an inverted index to facilitate prefix filtering, and the memory cost is $O(nL)$.

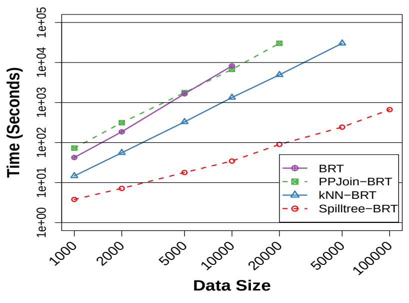
A comparison of all the algorithms is shown in Table 1. We can see that Spilltree-BRT has the least time complexity; however, it requires more memory.

4. EXPERIMENTS

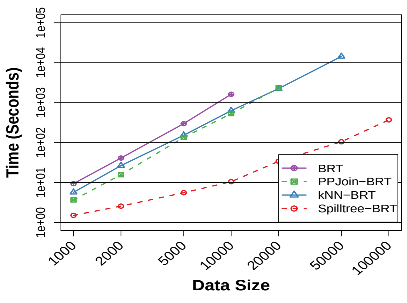
To demonstrate the effectiveness and scalability of the proposed taxonomy building algorithms, we test the original BRT and the nearest-neighbor-based methods on three different data sets. The nearest-neighbor-based methods include kNN-BRT, Spilltree-BRT, and PPJoin-BRT. In this section, we introduce the evaluation results in detail.

4.1 Performance of Clustering

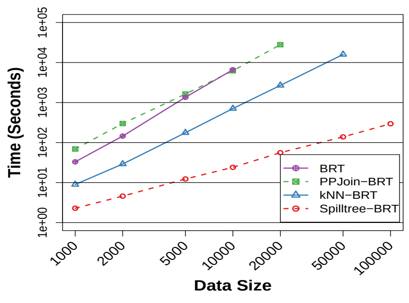
In this experiment, we use the 20-newsgroups data to evaluate the correctness of the algorithms. This data set forms a hierarchy with two levels. The first level has six clusters, and the second level contains 20 clusters. We randomly sample 2,000 documents to compare the clusters generated by different hierarchical clustering algorithms. We set all the parameters to be the same for Bayesian rose trees. In addition, k in kNN-BRT and Spilltree-BRT is set to 10. The dimension of random projection used in Spilltree-BRT is



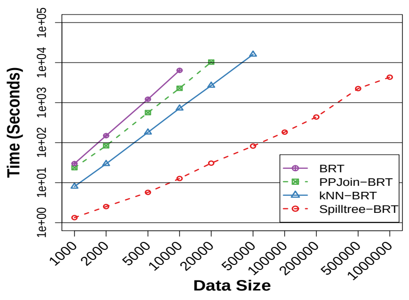
(a) End-to-end time for query data.



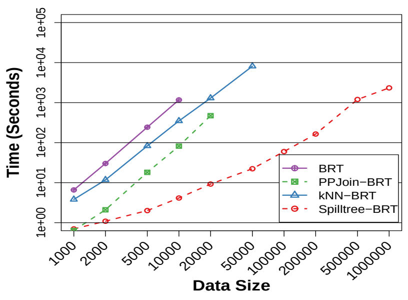
(b) Preprocessing time for query data.



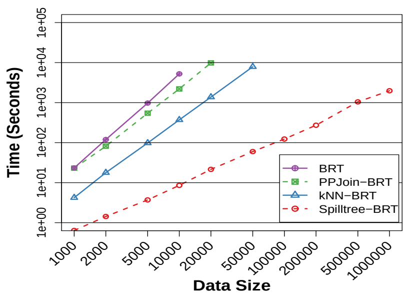
(c) Building time for query data.



(d) End-to-end time for news data.

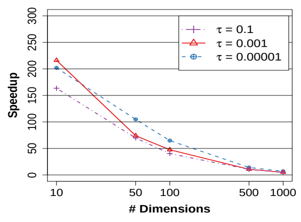


(e) Preprocessing time for news data.

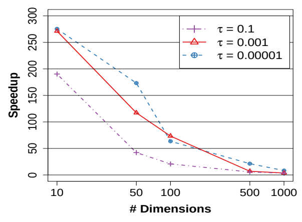


(f) Building time for news data.

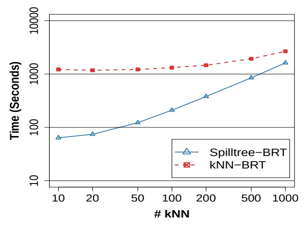
Figure 4: Time cost comparison of different algorithms.



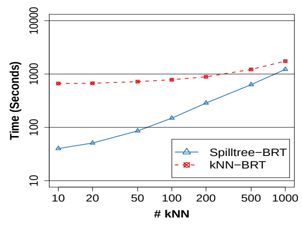
(a) Speedup over BRT with different parameters in Spilltree for query data.



(b) Speedup over BRT with different parameters in Spilltree for news data.



(c) Time cost comparison of k nearest neighbors for query data.



(d) Time cost comparison of k nearest neighbors for news data.

Figure 5: Comparison of the impact of different parameters and settings for kNN based methods (10,000 data samples).

set to 10. The overlapping parameter τ used in Spilltree [12] is set to 0.001.

We first compare the likelihood of different algorithms. As shown in Table 2, the nearest-neighbor-based methods are comparable with the original BRT. Particularly, the likelihood of kNN-BRT is better than the original BRT. The reason is that searching the candidate pairs from the nearest neighbors can significantly reduce the noise of the data. Thereby it leads to a better local optima. This phenomenon has also been observed in [4].

Then we leverage the NMI metric to compare the clusters of different levels in the generated trees with the human annotated ones (Table 3). All the results are based on ten trials of different executions. It can be seen that all the nearest-neighbor-based methods have similar performance with the original BRT. The kNN-BRT algorithm performs best with levels one and two, while the original BRT performs best with the third level. This observation is consistent with the likelihood results.

4.2 Scalability of Taxonomy Building

In this section, we investigate the scalability of the algorithms based on two data sets. The first one is a query data set consisting of 114,076 queries, where the concept and context information are extracted using the methods introduced in Section 3.1. The vocabulary size is 912,792 and the average number of words in the documents is 935. We also evaluate the algorithms on a larger news data set. It contains one million news articles. The vocabulary size is 252,174 and the average number of words is 242. All the experiments are conducted on a 64-bit server with 2.5GHz CPU and 32G of memory. The memory cost for the news data set is around 10G.

Table 2: Comparison of likelihood for 20-newsgroups data.

Methods	Likelihood
BRT	$-1.441 \times 10^6 \pm 1.637 \times 10^5$
kNN-BRT	$\mathbf{-1.392 \times 10^6 \pm 1.053 \times 10^5}$
Spilltree-BRT	$-1.473 \times 10^6 \pm 1.837 \times 10^5$
PPJoin-BRT	$-1.484 \times 10^6 \pm 1.348 \times 10^5$

We first conduct the experiments to examine the "end-to-end" time cost of different algorithms. We set $k = 10$ for kNN-BRT and Spilltree-BRT. The dimension of random projection used in Spilltree-BRT is set 10. The overlapping parameter τ used in Spilltree is set to 0.001. As shown in Figs. 4(a) and 4(d), the Spilltree-BRT al-

Table 3: Comparison of different algorithm with NMI scores.

Rose Tree Level	Level 1		Level 2		Level 3	
20-newsgroups cluster	6 Categories	20 Categories	6 Categories	20 Categories	6 Categories	20 Categories
BRT	0.055±0.006	0.075±0.008	0.202±0.010	0.295±0.016	**0.313±0.012**	**0.476±0.019**
*k*NN-BRT	**0.123±0.019**	**0.144±0.020**	**0.237±0.020**	**0.324±0.021**	0.311±0.014	0.454±0.020
Spilltree-BRT	0.076±0.004	0.101±0.007	0.189±0.007	0.273±0.114	0.279±0.007	0.423±0.011
PPJoin-BRT	0.088±0.010	0.113±0.012	0.190±0.033	0.269±0.062	0.258±0.038	0.389±0.065

gorithm is the fastest. *k*NN-BRT also performs faster than BRT and PPJoin-BRT. PPJoin-BRT is not as fast as expected because the process of finding the nearest neighbors must be re-run if there is no candidate cluster pair to merge. Particularly, for the query data set, PPJoin-BRT is even worse than the original BRT algorithm. This is due to the fact that PPJoin is significantly affected by the average length of documents. The average length of documents in query data is about four times that of the news data. For the news data set, PPJoin-BRT performs better than BRT.

To further analyze the performance of each algorithm in detail, we divide the execution into two parts: preprocessing and build. The detailed time costs for different algorithms are shown in Figs. 4(b), 4(c), 4(e), and 4(f). For BRT, during the preprocessing, we compute the likelihood values between data samples and then sort them. For the nearest neighbor based methods, the preprocessing mainly focuses on finding the nearest neighbors and computing the likelihood values. The building part of each algorithm gradually merges clusters. For different algorithms, merging clusters differs from searching for the candidate pair sets of different sizes. Moreover, after a new cluster is generated, different number of likelihood values will be computed, as illustrated in Algorithms 1 and 2. It can be concluded from the results that the time costs of *k*NN-BRT and Spilltree-BRT in these two parts is consistently lower than BRT. Specifically, the preprocessing of PPJoin-BRT is faster than *k*NN-BRT. However, for build is much slower. This again demonstrates our explanation that PPJoin-BRT will re-run the process of finding the nearest neighbors.

Since Spilltree-BRT is the fastest algorithm, we also investigate how different Spilltree parameters affect the time cost. The following experiments are conducted with 10,000 data samples. The comparison results between the nearest-neighbor-based methods and the original BRT are shown in Figs. 5(a) and 5(b). In the figures, different curves represent the time costs of the Spilltree algorithms with different overlapping tolerance parameters τ. This parameter controls how many data points we allow to be overlapped by different tree nodes in a Spilltree. As shown in the figures, the larger overlapping parameter results in slower algorithm execution, since it makes the algorithm backtrack more times back to the parent nodes [12]. Moreover, in each curve, we also show how the projected dimensions affect the time cost. Generally, fewer dimensions lead to faster algorithm execution. Spilltree-BRT can be 200–300 times faster than BRT when we project the data onto a ten-dimensional space. Finally, we evaluate how the number of the nearest neighbors affects the time cost. The results are shown in Figs. 5(c) and 5(d). We present the time costs of both *k*NN-BRT and Spilltree-BRT algorithms. It is shown that Spilltree-BRT performs better when the number of the nearest neighbors is small. This is also because it backtracks fewer times to the parent nodes when fewer neighbors are needed.

4.3 Top *K*NN Search

To investigate the quality of the proposed taxonomy building method, we also apply it to the top *K*NN search problem. This problem is defined as: given a query in the form of short text, we retrieve similar queries based on the concept and context features

Table 4: Comparison of the accuracy of top K search. (d is the dimension of random projection, and k is the number of the nearest neighbors.)

Algorithm	Top 5	Top 10
Spilltree (d = 10)	0.290±0.364	0.321±0.350
Spilltree (d = 50)	0.632±0.317	0.640±0.295
Spilltree (d = 100)	0.754±0.298	0.731±0.280
Spilltree (d = 500)	0.904±0.149	0.880±0.172
BRT	0.922±0.193	0.921±0.187
*k*NN-BRT (k = 1)	0.929±0.197	0.929±0.209
*k*NN-BRT (k = 5)	0.923±0.206	0.936±0.159
*k*NN-BRT (k = 20)	0.928±0.160	0.961±0.124
PPJoin-BRT	0.868±0.247	0.852±0.271
Spilltree-BRT (d = 10, k = 1)	0.512±0.468	0.496±0.463
Spilltree-BRT (d = 50, k = 1)	0.818±0.335	0.796±0.338
Spilltree-BRT (d = 100, k = 1)	0.852±0.300	0.843±0.297
Spilltree-BRT (d = 500, k = 1)	0.907±0.217	0.909±0.204
Spilltree-BRT (d = 10, k = 5)	0.600±0.451	0.587±0.437
Spilltree-BRT (d = 50, k = 5)	0.815±0.339	0.799±0.344
Spilltree-BRT (d = 100, k = 5)	0.844±0.292	0.824±0.299
Spilltree-BRT (d = 500, k = 5)	0.856±0.289	0.852±0.287
Spilltree-BRT (d = 10, k = 20)	0.777±0.365	0.777±0.340
Spilltree-BRT (d = 50, k = 20)	0.885±0.257	0.909±0.221
Spilltree-BRT (d = 100, k = 20)	0.891±0.242	0.902±0.227
Spilltree-BRT (d = 500, k = 20)	0.901±0.221	0.911±0.208

from an already built tree. We use K here to distinguish the notation of k used for *k*NN-based construction of taxonomy. In this experiment, we investigate whether Spilltree, *k*NN-BRT, and Spilltree-BRT can bring benefits to this problem.

We first select 1,000 queries to build the taxonomy trees by leveraging the Spilltree algorithm and different algorithms of BRTs. The experiments of Spilltree are conducted by doing a grid search for tolerance parameter $\tau = \{0.2, 0.3, 0.4\}$. The experiments of all BRT-related algorithms are conducted by doing a grid search for parameters $\alpha = \{5, 10, 15\}$ (Eq. (7)) and $\gamma = \{0.1, 0.5, 0.9\}$ (Eq. (3)). We choose the best results to be shown in this experiment. Then we randomly select 100 queries to search the *K*NN of each query. The ground truth is generated based on the cosine similarity. We evaluate the precision of the top K search problems for different algorithms, which is the proportion of the overlapped top K keywords retrieved by the generated trees and the brute-force search using the cosine similarity metric. The results are shown in Table 4. From the analysis of the results, we draw the following conclusions: First, the dimension of random projection significantly affects the accuracy of Spilltree. In general, the more random projection features we use, the better the accuracy is. The search accuracy can be improved from 30% to more than 90%. Second, the number of the nearest neighbors used to accelerate the building procedure does not affect the accuracy very much. For *k*NN-BRT, the accuracy does not show a significant change when changing the number of k. Third, Spilltree-BRT can further improve the accuracy of Spilltree. With the same projection dimensionality d, Spilltree-BRT results are significantly better than Spilltree. Increasing d or k can both improve the accuracy. This is because larger d results in a better searching accuracy of the nearest neighbors, while larger k leads

Table 5: Comparison of the time (in 10^{-6} seconds) of top K search. (d is the dimension of random projection.)

Algorithm	Top 5	Top 10
Spilltree (d = 10)	2.29±5.53	1.86±5.06
Spilltree (d = 50)	657.02±158.82	693.02±115.78
Spilltree (d = 100)	1,103.84±253.28	1,159.44±147.72
Spilltree (d = 500)	3,867.02±777.99	4,443.81±633.41
Spilltree-BRT (d = 10, k = 1)	34.38±109.55	35.94± 110.17
Spilltree-BRT (d = 10, k = 5)	48.44±84.71	40.63±78.50
Spilltree-BRT (d = 10, k = 20)	148.44±977.40	371.88±1,407.09
Spilltree-BRT (d = 50, k = 1)	143.75± 978.46	251.56± 1,464.60
Spilltree-BRT (d = 50, k = 5)	34.38± 64.73	43.75± 73.55
Spilltree-BRT (d = 50, k = 20)	50.00± 85.24	50.00± 98.53
Spilltree-BRT (d = 100,k = 1)	1,724.99±6,136.47	2,195.30 ± 6,855.77
Spilltree-BRT(d = 100,k = 5)	1,782.82±8,498.16	1,814.07 ± 8,687.89
Spilltree-BRT(d = 100,k = 20)	1,496.89±4,237.15	1,482.82 ± 4,233.09
Spilltree-BRT(d = 500,k = 1)	2,425.03 ± 5,085.73	2,621.91 ± 5,237.34
Spilltree-BRT(d = 500,k = 5)	2,135.97±5,025.30	2,223.47± 5,030.34
Spilltree-BRT(d = 500,k = 20)	2,121.89±4,420.28	2,112.51 ± 4,458.08

to more candidate cluster pairs which increase the opportunity for finding the right clusters to merge.

We also test the searching time for Spilltree and Spilltree-BRT. 110K data is used to build the trees and 200 queries are randomly selected to test the retrieving time. As shown in Table 5, Spilltree performs faster with lower dimension of random projection. However, the accuracy is not good enough (see Table 4). With higher dimensions, the time cost of Spilltree increases very quickly. The major reasons are: (1) It cost more time to compare the similarity between two points for higher dimensionality; (2) Given the same parameter configuration, Spilltree will search more points in high dimensional space. We also perform a grid search for Spilltree-BRT. It can be seen that all results are achieved in acceptable time. The search time of Spilltree-BRT depends on the structure of the tree. Thus, changing d and k only affects the building time of Spilltree-BRT and does not monotonically affect the searching time.

5. CONCLUSION AND FUTURE WORK

In this paper, we present an approach that can automatically derive a domain-dependent taxonomy from a set of keyword phrases by leveraging both a general knowledgebase and keyword search. We first deduce concepts with the technique of conceptualization and extract context information from a search engine, and then induce the new taxonomy using a Bayesian rose tree. We provide three nearest-neighborhood-based methods to speed up the original Bayesian rose tree algorithm. Particularly, the Spilltree-based algorithm reduces the time and memory cost significantly. We also conducted a set of experiments to demonstrate the effectiveness and efficiency of the proposed algorithms.

We regard the work presented as initial, as there are improvements to be made as well as many directions to pursue. First, we will investigate how to use a set of random projections to exponentially reduce the errors of k nearest neighbor searches in the Spilltree algorithm. Second, we would like to use other locality sensitive hashing methods or latent semantic sensitive dimensionality reduction methods to improve the search accuracy in Spilltree. Third, we would like to apply our taxonomy building method to real-world applications (e.g., advertisement or personalized search) to demonstrate its effectiveness and usefulness.

Acknowledgements

We would like to thank Charles Blundell and Yee Whye Teh for their help on the implementation of the Bayesian rose tree, and thanks to Ting Liu for help on the implementation of Spilltree.

6. REFERENCES

[1] R. P. Adams, Z. Ghahramani, and M. I. Jordan. Tree-structured stick breaking for hierarchical data. In *NIPS*, 2010.
[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
[3] C. Blundell, Y. W. Teh, and K. A. Heller. Bayesian rose trees. In *UAI*, 2010.
[4] W.-Y. Chen, Y. Song, H. Bai, C.-J. Lin, and E. Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(3):568–586, 2011.
[5] S.-L. Chuang and L.-F. Chien. Towards automatic generation of query taxonomy: A hierarchical query clustering approach. In *ICDM*, pages 75–82, 2002.
[6] S. Dasgupta and A. Gupta. An elementary proof of the Lohnson-Lindenstrauss lemma. Technical report, MIT, 1999. ICSI Technical Report TR-99-006.
[7] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.
[8] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, pages 539–545, 1992.
[9] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31:264–323, September 1999.
[10] D. A. Knowles and Z. Ghahramani. Pitman-Yor diffusion trees. In *UAI*, 2010.
[11] T. Lee, Z. Wang, H. Wang, and S. Hwang. Web scale taxonomy cleansing. *PVLDB*, 4(12):1295–1306, 2011.
[12] T. Liu, A. W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *NIPS*, pages 825–832, 2005.
[13] R. E. Madsen, D. Kauchak, and C. Elkan. Modeling word burstiness using the Dirichlet distribution. In *ICML*, pages 545–552, 2005.
[14] I. Mani, K. Samuel, K. Concepcion, and D. Vogel. Automatcally inducing ontologies from corpora. In *Workshop on Computational Terminology*, 2004.
[15] R. Navigli, P. Velardi, and S. Faralli. A graph-based algorithm for inducing lexical taxonomies from scratch. In *IJCAI*, pages 1872–1877, 2011.
[16] H. Poon and P. Domingos. Unsupervised ontology induction from text. In *ACL*, pages 296–305, 2010.
[17] E. Sadikov, J. Madhavan, L. Wang, and A. Y. Halevy. Clustering query refinements by user intent. In *WWW*, pages 841–850, 2010.
[18] D. Shen, M. Qin, W. Chen, Q. Yang, and Z. Chen. Mining web query hierarchies from clickthrough data. In *AAAI*, pages 341–346, 2007.
[19] Y. Song, H. Wang, Z. Wang, H. Li, and W. Chen. Short text conceptualization using a probabilistic knowledgebase. In *IJCAI*, pages 2330–2336, 2011.
[20] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.
[21] J. Wang, H. Wang, Z. Wang, and K. Q. Zhu. Understanding tables on the web. In *ER*, 2012.
[22] Y. Wang, H. Li, H. Wang, and K. Q. Zhu. Toward topic search on the web. In *ER*, 2012.
[23] R. W. White, P. N. Bennett, and S. T. Dumais. Predicting short-term interests using activity-based search context. In *CIKM*, pages 1009–1018, 2010.
[24] W. Wu, H. Li, H. Wang, and K. Q. Zhu. Probase: A probabilistic taxonomy for text understanding. In *SIGMOD*, 2012.
[25] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.*, 36(3):15, 2011.