Automatic Taxonomy Construction from Keywords via Scalable Bayesian Rose Trees

Yangqiu Song, *Member, IEEE,* Shixia Liu, *Senior Member, IEEE,* Xueqing Liu, and Haixun Wang, *Member, IEEE*

Abstract—In this paper, we study a challenging problem of deriving a taxonomy from a set of keyword phrases. A solution can benefit many real-world 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 context 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 $O(n \log n)$ using a nearest-neighbor-based approximation, 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 Web search query taxonomy illustrates the usefulness of our approach for specific domains.

Index Terms—Bayesian Rose Tree, Hierarchical Clustering, Short Text Conceptualization, Keyword Taxonomy Building

1 Introduction

Taxonomy is a hierarchical classification of a set of things or concepts in a domain[1]. Building a taxonomy is an essential element of many applications. For example, in web search, organizing domain-specific queries into a hierarchy can help better understand the query and improve search results [49] or help with query refinement [38]. 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.

In this paper, we consider the challenging problem 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).

- *Yangqiu Song is with Department of Computer Science, University of Illinois at Urbana-Champaign, United States. E-mail: yqsong@illinois.edu*
- *Shixia Liu (corresponding author) is with School of Software, Tsinghua University, Beijing, China. E-mail: shixia@tsinghua.edu.cn*
- *Xueqing Liu is with Department of Computer Science, University of Illinois at Urbana-Champaign, United States. E-mail: xliu93@illinois.edu*
- *Haixun Wang is with Google Research, United States. E-mail: haixun@google.com*

1. http://en.wikipedia.org/wiki/Taxonomy

The problem of inducing a taxonomy from a set of keywords has one major challenge. Although using a set of keywords allows us to 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 results) 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 a hierarchical clustering approach [9], [40]. 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." As a result, the 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 both knowledge and context are needed to create a taxonomy out of a set of keywords. 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, a 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"). On the other hand, context is also important. It is unlikely that the knowledgebase (especially a general-purpose one) knows about every subsumption relationship in the specific domain. For example, it has no idea that x is a sub-concept of y. However, through 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 context, we can establish the relationship between x and y.

Specifically, we deduce a set of *concepts* for each keyword phrase using a general-purpose knowledgebase of world knowledge [51] and we obtain the *context* information of the keyword from a search engine, which is a set of words co-occurring in the search snippets. After enriching the keyword set using knowledge and context, we use hierarchical clustering to automatically induce a new taxonomy. We adopt Bayesian Rose Tree (BRT) [5] as a basic algorithm for taxonomy induction since it is simple to implement and easy to understand and be interpreted. Moreover, we also extend the basic BRT with two different marginal distributions, i.e., Dirichlet compound multinomial (DCM) and Von Mises-Fisher (vMF) distributions, as well as two nearest-neighbor-based approximation methods. DCM [32] and vMF [2] are very useful for modeling high-dimensional sparse data, which is the case of our taxonomy induction faces with. DCM is particularly designed for discrete features, while vMF can be used for both discrete and real-value features. For both distributions, the nearest neighbors are searched to reduce the examination of clusters for aggregation. Thus, this approximation can significantly reduce the computational complexity of the clustering algorithm.

A preliminary version of this work appeared in KDD 2012 [31]. We have extended the framework with the vMF distribution and provide a systematic comparison with the previously proposed DCM distribution. We show that vMF is better both in efficiency and effectiveness. Our paper offers three technical contributions. First, we illustrate how to derive the concepts and context from each keyword. Second, we present a way to formulate the taxonomy building as a hierarchical clustering problem based on the concepts and context of the keywords. And finally, we scale up the method to millions of keywords.

2 Related Work

2.1 Domain-Specific Taxonomy Building

Recently, automatically creating a domain-specific taxonomy has attracted a lot of interest [11], [17], [33], [34], [35], [37], [44]. The assumption is that the text corpus accurately represents the domain. Although these text-corpus based approaches have achieved some success [34], [44], they have several disadvantages.

First, for a highly focused domain, it is very difficult to find a text corpus that accurately characterizes that domain. Second, even if we can find a corpus that accurately characterizes the domain, we may still have a data sparsity problem [51]. High quality patterns typically have very low recall. For example, it is well known that Hearst patterns [22] (i.e., "such as" patterns) have a high 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.

Compared to previous methods, we provide a way to cluster keywords on the fly for a given keyword set. Therefore, we do not need a specific corpus to extract the taxonomic relationships of keywords. Instead, we naturally group them with similar semantic meanings.

2.2 General Knowledgebases

Instead of building taxonomies from a text corpus, we can also extract a domain-specific taxonomy from a large, general-purpose knowledgebase. A general knowledgebase can be constructed manually or automatically.

Manually constructed knowledgebases, including WordNet [18], the Open Directory Project (ODP)[2], Wikipedia[3], Cyc project [29] and Freebase [6], are constructed the fine-grained human annotation. They have revealed their limitations in terms of resources and labor requirements. Often they provide a relatively smaller set of concepts and offer incomplete formulation of the concepts of terms used in day-to-day communication [42].

On the contrary, automatically constructed knowledgebases take advantage of the large amount of Web information and recent growth in parallel computing infrastructure and techniques [13]. Recent developments in such knowledgebases use natural language processing and machine learning techniques to automatically detect useful knowledge from raw text. Examples include KnowItAll [15], TextRunner [3], WikiTaxonomy [36], YAGO [45], NELL [7], and Probase [51]. The major limitations of a general-purpose knowledgebase are that it usually has low coverage of a highly focused domain and it may produce ambiguous interpretations for highly specialized terms in the domain. For example, the coverage of ODP for classifying URLs is about 60% [49]. Given 50 million randomly sampled Web search queries, Probase can cover about 81% of them. However, there are still many tail queries that cannot be identified. Therefore, a general methodology for constructing a taxonomy given a set of keywords should be developed.

2. http://www.dmoz.org/
3. http://www.wikipedia.org/

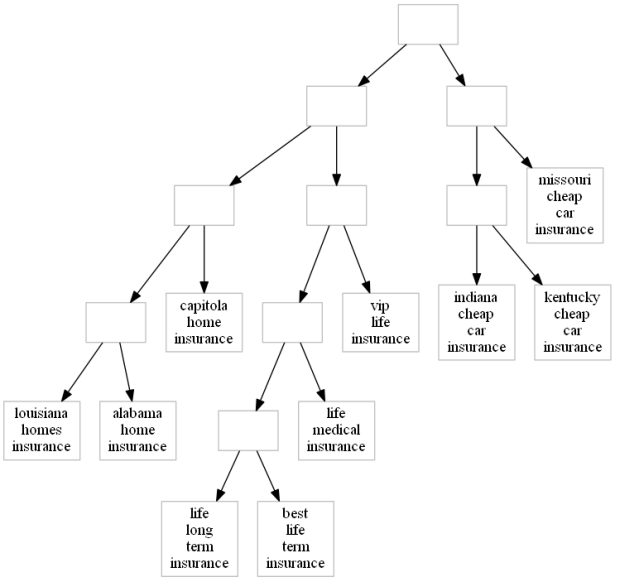


Fig. 1. Example of a binary-branch taxonomy.

Compared to general knowledgebases, we build a keyword taxonomy upon them, instead of extracting from them. Our method leads to better coverage because we also make use of context information for the keywords as complementary information. The context information is extracted from a search engine and therefore offers better coverage of keywords.

2.3 Hierarchical Clustering

Hierarchical clustering is a widely used clustering method [25]. Previously, using hierarchical clustering to generate a taxonomy has been explored in [16], [17], [9], [40]. Among them, [16], [17] used syntactic pattern distribution to measure the class of verb frames and bottom-up breadth-first clustering algorithm to generate the hierarchy. In contrast, we focus more on noun phrases. Moreover, [9], [40] used context words to represent the keywords and conduct hierarchical clustering. Compared with these approaches, we not only employ the context information of keywords, but also explicitly express their concepts, enabling us to explicitly describe the clusters.

From the perspective of methodologies, two strategies are used for hierarchical clustering: agglomerative and divisive. The agglomerative approach [23], [25], [39], [46] builds a tree from the bottom up by combining the two most similar clusters at each step, while the divisive approach [25], [43], [27] builds a tree from the top down by splitting the current cluster into two clusters at each step. Since divisive clustering approaches need to determine which cluster to split at each step and how many clusters to split, it is difficult to control and less deterministic compared with agglomerative hierarchical clustering approaches. Thus, we base our approach on agglomerative hierarchical clustering algorithms and briefly introduce them in the following section.

Traditional agglomerative hierarchical clustering algorithms construct binary trees [23], [25], [39], [46]. However, binary branches may not be the best model to describe the 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. The assumption is that we have a good similarity measure that enables us to group related keywords in an optimal manner. 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. Clearly, 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.

To remedy this, multi-branch trees have been developed [5]. An example of multi-branch clustering is shown in 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 hierarchical clustering approaches [1], [5], [26]. The methods proposed by Adams et al. [1] and Knowles et al. [26] are based on the Dirichlet diffusion tree, which uses an MCMC process to infer the model. Blundel et al. [5] propose a deterministic and agglomerative approach called BRT. We choose BRT because it is faster than the MCMC based approaches and it is deterministic, which means, it will generate the same tree with different trails as if the parameters are set to be the same. We extend the BRT with two different marginal distributions, and provide approximate mechanisms based on nearest neighbor search to significantly speed up the building process.

3 Bayesian Rose Tree (BRT)

In this section, we briefly introduce the BRT [5] algorithm. The original BRT is introduced with a set of binomial distributions for text data analysis. In this work, we modify the marginal distribution as a Dirichlet compound multinomial and Von Mises-Fisher distributions. Here we introduce the base algorithm of our work, the original BRT proposed in [5].

A rose tree is a mixture distribution over a set of partitions $\phi(T_m) \in \mathcal{P}_m$ of data at leaves $\mathcal{D}_m = \text{leaves}(\text{T}_\text{m})$:

$$p(D_m|T_m) = \sum_{\phi(T_m)\in\mathcal{P}_m} p(\phi(T_m))p(\mathcal{D}_m|\phi(T_m)), \quad (1)$$

where $p(\phi(T_m))$ is the probability of the partition $\phi(T_m)$ and $p(\mathcal{D}_m|\phi(T_m))$ is the probability of data given the partition. Since the number of partitions can be exponentially large, we follow [5] to use the following recursive definition of the probability:

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

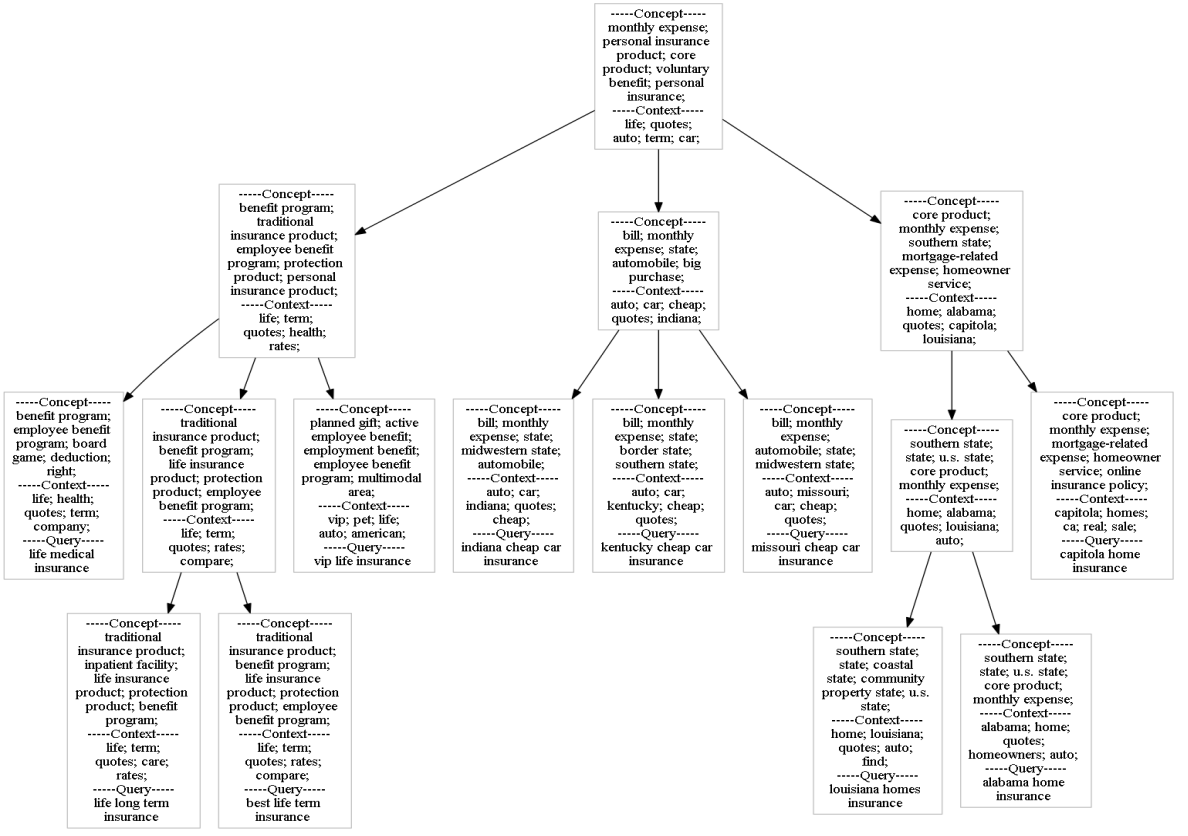


Fig. 2. Example of Multi-branch Query Taxonomy.

where $f(\mathcal{D}_m)$ is the marginal probability of 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. This means with probability π_{T_m}, tree T_m has no sub-structures, that is, its child nodes are individual data samples; while with probability $1 - \pi_{T_m}$, it is represented by a set of child nodes. $\prod_{T_i \in \text{Children}(T_m)} p(\mathcal{D}_m|T_m)$ is the likelihood that the tree is partitioned into sub-trees. In BRT [5], π_{T_m} is defined as:

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

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 steps of BRT comprise greedily and agglomeratively merging the data and clusters inside the tree. In the beginning, each data sample is regarded as a tree on its own: $T_i = \{\mathbf{x}_i\}$ where $\mathbf{x}_i$ is the feature vector of ith data sample. 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 [5]:

- **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 node.
- **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, the BRT algorithm greedily finds two trees, T_i and T_j, to maximize the ratio of probability:

$$\frac{p(\mathcal{D}_m|T_m)}{p(\mathcal{D}_i|T_i)p(\mathcal{D}_j|T_j)} \tag{4}$$

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

The major cost of this bottom-up hierarchy construction approach is dominated by:

(1) searching for pairs of clusters to merge;
(2) calculating the likelihood associated with the merged cluster based on Eq. (4).

4 Keyword Taxonomy Building

To build a taxonomy effectively and efficiently, we summarize our framework in Fig. 3 and explain it as follows.

- **Keyword representation:** We first use knowledge of concepts and context for keyword representation, which is introduced in Section 4.1. We use the technique called conceptualization [42] based on Probase [51] to acquire concepts and use search results as a resource to extract context.
- **Feature vector modeling:** Then we introduce the text modeling approaches in Section 4.2. We

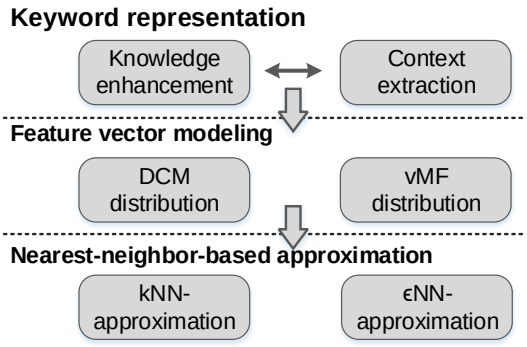


Fig. 3. The pipeline of taxonomy construction

extend our previous work [31] from the Dirichlet compound multinomial distribution (Section 4.2.1) to the more flexible Von Mises-Fisher distribution (Section 4.2.2) and discuss the difference between these two modeling approaches.

- **Nearest-neighbor-based approximation:** Finally, we show how to approximate the original BRT using nearest neighbors of clusters to speed up the building process. Specifically, we first introduce the k-nearest-neighbor based approach and its extension with Spilltree [30] (Section 5.1). Then we introduce the ϵ-ball-nearest-neighbor based approach, which is implemented with the PPJoin+ algorithm [52] (Section 5.2).

4.1 Knowledge and Context

We propose a "knowledge+context" approach to augment the representation of keywords and build a better taxonomy. The knowledge information is obtained from a general-purpose knowledgebase and the context information is obtained from search engine snippets.

We use the knowledgebase Probase, which is constructed by automatic ontology population [50]. 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 [51]. 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 that it contains millions of concepts, from well-known ones such as "country" and "artists" to small but concrete concepts 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 enables us to derive concepts from a keyword phrase in order to enrich it. 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 ways.

First, in order to get a more comprehensive representation of keywords for inference, we introduce a set of probabilistic measures. For example, $P(instance|concept)$ tells us how typical the *instance* in the given *concept* is [51]. 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 short text string. In addition to $P(instance|concept)$ we also obtain $P(concept|instance)$, $P(concept|attribute)$, and $P(attribute|concept)$ [42]. 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 *instance* and *concept* co-occur in the same sentence with the patterns Probase used for extraction, and $n(concept)$ is the frequency of the *concept*.

Second, a keyword phrase may be syntactically and semantically complicated and require sophisticated chunking and parsing to identify meaningful terms. For example, 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. We use two heuristic rules to detect the instances/attributes contained in the short text using Probase. First, we prefer longer phrases since they tend to be more specific [48]. For example, for the query "truck driving school pay after training," we obtain "truck driving," "driving school," "pay," and "training," which are all longest instances in Probase. Although "driving" is also an instance in Probase, we do not use it since it is not the longest instance. Second, we prefer the high frequency phrases known by Probase. For example, if two instances/attributes are of the same length, we choose the one connected to more concepts. A more detailed algorithm is shown in [41].

During the phrase detection, we also use a rule based plural words converter to check whether the converted phrase is in Probase. Moreover, we use synonyms mined from Probase to enrich the concepts [28]. Finally, 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 [42].

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 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. However, they can still 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" (since no one has mentioned it in a "such as" pattern). 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 a greater similarity to the query containing "monster.com."

4.2 Feature Vector 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 3). The original BRT approach [5] assumes that the data is 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 context 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 $f(\mathcal{D})$. Therefore, we first derive a set of weighted terms that represent the context and the concept, and then we use the Dirichlet compound multinomial (DCM) distribution [32] and the Von Mises-Fisher (vMF) distribution to model the data.

Given a set of keyword phrases $\{kw_1, \cdots, kw_n\}$, we derive a list of (t_j, w_j) pairs for each keyword kw_i, where t_j is either a concept produced by the knowledgebase, or a context word generated by search, and the w_j is derived as follows:

$$w_j = \lambda \cdot freq(t_j) + (1-\lambda) \cdot \sum_i^n C_i \cdot P(t_j|kw_i) \qquad (6)$$

where λ is a parameter that controls how much we value context compared to concepts. In practice, we select the λ so that the largest frequencies of concept and context words are the same value. $freq(t_j)$ is the frequency of a term j in the context derived from the search results; $P(t_j|kw_i)$ is the probability of the term as a concept given keyword phrase kw_i, and is provided by the knowledgebase (as in Eq. (5)); C_i is the frequency of kw_i in the knowledgebase, and $C_i \cdot P(t_j|kw_i)$ is used as the frequency of the term as a concept. We then set the feature vector of a keyword kw_i as $\mathbf{x}_i = (x_i^{(1)}, ..., x_i^{(V)}) \in \mathcal{R}^V$. Here V is the vocabulary size and term frequencies $x_i^{(j)} = w_j$. Other approaches that generate distributional descriptions of keywords include using co-occurrence of syntactic or semantic patterns in the context [10], [20], [21], [24]. Compared with these approaches, we not only employ the context information of keywords, but also explicitly express their concepts, enabling us to explicitly describe the clusters.

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. In this work, we employ the DCM and vMF distributions.

4.2.1 DCM distribution:

We first use the DCM distribution [32] to represent the marginal distribution $f(\mathcal{D})$. DCM is derived based on multinomial and Dirichlet conjugate distributions. The 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 set of features underlying DCM is such that we first sample a multinomial distribution from the Dirichlet distribution, and then sample the features based on the multinomial distribution. The multinomial distribution can be regarded as a sub-topic distribution, which makes certain features appear more likely in a feature vector [32]. DCM integrates 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 $p(\mathbf{x}_i|\boldsymbol{\theta})$ is defined by:

$$p(\mathbf{x}_i|\boldsymbol{\theta}) = \frac{m!}{\prod_j^V x_i^{(j)}!} \prod_{j=1}^V p(x_i^{(j)}|\boldsymbol{\theta}) = \frac{m!}{\prod_j^V x^{(j)}!} \prod_{j=1}^V [\theta^{(j)}]^{x_i^{(j)}}, \qquad (7)$$

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

The Dirichlet distribution prior is:

$$p(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \frac{\Gamma(\sum_{j=1}^{V}\alpha^{(j)})}{\prod_{j=1}^{V}\Gamma(\alpha^{(j)})}\prod_{j=1}^{V}[\theta^{(j)}]^{\alpha^{(j)}-1} = \frac{1}{\Delta(\boldsymbol{\alpha})}\prod_{j=1}^{V}[\theta^{(j)}]^{\alpha^{(j)}-1}, \quad (8)$$

where $\boldsymbol{\alpha} = (\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(V)})^T \in \mathcal{R}^V$, and the Gamma function[4] has the property $\Gamma(x+1) = x\Gamma(x)$. The "Dirichlet delta function" $\Delta(\boldsymbol{\alpha}) = \frac{\Gamma(\sum_{j=1}^{V}\alpha^{(j)})}{\prod_{j=1}^{V}\Gamma(\alpha^{(j)})}$ is introduced for convenience.

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

$$f_{DCM}(\mathcal{D}) = \int_{\boldsymbol{\theta}}\prod_{i}^{n}p(\mathbf{x}_i|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{\alpha})d\boldsymbol{\theta} = \prod_{i}^{n}\frac{m!}{\prod_{j}^{V}x_i^{(j)}!}\cdot\frac{\Delta(\boldsymbol{\alpha}+\sum_i\mathbf{x}_i)}{\Delta(\boldsymbol{\alpha})}. \quad (9)$$

Using this marginal distribution $f_{DCM}(\mathcal{D})$, we can integrate the weights into the keyword feature vector.

4.2.2 vMF distribution:

In text analysis, the most popular metric to evaluate the similarity between two high-dimensional sparse documents is cosine similarity. This is because the text data is first mapped onto a high-dimensional unit hypersphere and then the similarity is evaluated based on the angle between two text data. vMF is the distribution that characterizes this type of directional data. In our modeling, we set both the prior and likelihood distribution to vMF and the marginal distribution has the following form:

$$f_{vMF}(\mathcal{D}) = \int_{\boldsymbol{\mu}}\prod_{i}^{n}p(\mathbf{x}_i|\boldsymbol{\mu},\kappa)p(\boldsymbol{\mu}|\boldsymbol{\mu}_0,\kappa_0)d\boldsymbol{\mu} = \frac{c_V(\kappa_0)\prod_i^n c_V(\kappa)}{c_V(\|\kappa\sum_i\mathbf{x}_i+\kappa_0\boldsymbol{\mu}_0\|)}. \quad (10)$$

The likelihood is:

$$p(\mathbf{x}_i|\boldsymbol{\mu},\kappa) = c_V(\kappa)\exp(\kappa\boldsymbol{\mu}^T\mathbf{x}_i) \quad (11)$$

where $\|\mathbf{x}_i\| = 1$, $\|\boldsymbol{\mu}\| = 1$, $\kappa \geq 0$ and the vocabulary size $V > 2$. Here $\boldsymbol{\mu}$ is the directional parameter that characterizes the direction of the cluster center. κ is the concentration parameter where a larger value implies stronger concentration about the mean direction. The normalization constant is given by:

$$c_V(\kappa) = \frac{\kappa^{V/2-1}}{(2\pi)^{V/2}I_{V/2-1}(\kappa)}, \quad (12)$$

where $I_r(\cdot)$ is the modified Bessel function of the first kind and order r [2].

The prior of vMF is also a vMF distribution:

$$p(\boldsymbol{\mu}|\boldsymbol{\mu}_0,\kappa_0) = c_V(\kappa_0)\exp(\kappa_0\boldsymbol{\mu}_0^T\boldsymbol{\mu}) \quad (13)$$

where we also have $\|\boldsymbol{\mu}_0\| = 1$, $\kappa_0 \geq 0$.

The choice of vMF enables the clustering algorithm to be more flexibile for using different text features. Although in this paper we use the frequency of terms (or concepts) as the features to represent keywords, there are many cases where we want to use real-value features, e.g., TFIDF (term frequency–inverse document frequency), instead of discrete features. In these cases, vMF is still applicable while DCM is not. We will see in the experiments section that vMF performs better on the evaluated data sets, both from efficiency and effectiveness perspectives.

Given these two types of modeling distributions, we can build the taxonomy from keywords using the BRT algorithm. Next, we will introduce how to construct the taxonomy more efficiently.

5 Efficient Taxonomy Construction

The complexity of the original BRT is analyzed as follows. Assume in the current round there are c clusters. The two steps described in Section 3 take $O(c^2)$ time. At the start of the algorithm, we set $c = n$, the number of data samples. For all the clusters merging into one cluster, if we first compute the likelihood in Eq. (4), 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 to any large-scale data set. Moreover, this approach suffers from $O(n^2)$ memory cost. For $100,000$ data samples, this will take $3 \times 8 \times 10^{10} = 240G$ bytes memory to contain the likelihood of all the pairs, if we use 8 bytes to store double-precision values and have three types of merging likelihood ("join," "absorb," and "collapse"). Thus, BRT is not applicable to large-scale data sets.

The most time consuming process in agglomerative clustering is searching all the 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 "caching" a set of nearest neighbors for each data sample. Then the search complexity only depends on the number of nearest neighbors. However, searching for nearest neighbors still incurs a cost of $O(n)$ for each data sample. The time complexity of finding k-nearest neighbors can be reduced to $O(\log n)$ using techniques such as KD-trees [4] and Metric trees [47]. 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) [19] can be used when the number of dimensions is large.

4. For integer variables, Gamma function is $\Gamma(x) = (x-1)!$. For real numbers, it is $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}\mathrm{d}t$.

Algorithm 1 Nearest-neighbor-based BRT.

Input: A set of data samples $\mathcal{D}$.
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. (4), where $m \in \{\text{join, absorb, collapse}\}$.
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

Particularly, Spilltree [30] relaxes the non-overlapping constraints of Metric trees and incorporates the technique used in LSH, thus combining 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) [8]. kNN finds k-nearest neighbors for each data sample and is not concerned with the density of the data. ϵNN uses a spherical ball to bound all the nearest neighbors within the ball. In our taxonomy construction approach, we significantly reduce the time complexity by using methods such as Spilltree [30] and PPJoin+ [52].

5.1 kNN-Approximation

This paper introduces 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 1.

kNN-BRT: Using the kNN approach, we first find the k nearest neighbors for each data sample and then 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 the k-nearest neighbors of a data sample, we retain a minheap of size k to maintain the data with the largest similarities scores. When a new data sample comes that has a larger similarity score than the top value (the smallest one in the minheap), we replace the top index with the new data sample. 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 [30] to further reduce the time complexity.

Spilltree is a generalization of Metric trees [47]. Metric trees partition the data space into binary trees, and retrieve the nearest neighbors by a DFS (depth first search). Metric trees will be less efficient when the number of dimensionality is large (e.g., larger than 30 [30]). To solve this problem, the Spilltree algorithm offers two major modifications:

1. It introduces a random projection before partitioning the data space.
2. It introduces the overlapping/non-overlapping regions between nodes when partitioning each sub-tree. When searching for the nearest neighbors in the sub-trees with overlapping partitions, Spilltree searches one branch only.

According to the Johnson-Lindenstrauss Lemma [12], embedding a data set of dimension n to an $O(\log n)$ dimensional space does not lead to much distortion for pairwise distances. As a result, a brute-force search in the projected space provides a $(1 + \varepsilon)$-NN search [30] in the original space. Thus, by projecting the data to 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, the original metric trees perform a DFS to find the nearest neighbors. By introducing 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 succeed 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 accuracy of both overlapping and non-overlapping nodes is ensured [30]. Overall, the time complexity of a Spilltree search is $O(\log n)$ [30].

The random projection to d-dimensional space has a 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 samples 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.

5.2 ϵNN-Approximation

In ϵNN-approximation, for each data sample, we retain the nearest neighbors whose similarity with the data samples are larger than a pre-defined threshold ϵ. The flow chart of the ϵNN-BRT algorithm is shown in Alg. 2. Using ϵNN-approximation reduces the time complexity to $O(n^2 \cdot C_V)$. The storage of ϵNN depends on the number of 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 merge, because when the threshold ϵ is too large, ϵNN will not return any nearest neighbors.

PPJoin-BRT: To support ϵNN-approximation, we need to efficiently find ϵ-neighbors of a data sample.

Algorithm 2 Bayesian Rose Tree with ϵNN Approximation (ϵNN-BRT).

Input: A set of data samples $\mathcal{D}$.
Initialization 1: Set $T_i = \mathbf{x}_i$ for $i = 1, 2, \cdots, n$, number of clusters $c = n$.
Initialization 2: Set threshold $\epsilon = 0.9$. Find ϵ nearest neighborhoods for each cluster, and compute all the likelihood scores.
while $c > 1$ **do**
  if No available clusters in the candidate set to merge. **then**
    Run ϵNN algorithm for current clusters with smaller threshold $\epsilon = 0.9 \times \epsilon$.
  end if
  1. Find T_i and T_j in all the neighborhood sets $\{\mathcal{N}_\epsilon(T_i)\}$, and merge operation m that maximizes Eq. (4), where $m \in$ {join, absorb, collapse}.
  2. $T_m \leftarrow$ the result of operation m on T_i and T_j.
  3. Find ϵ nearest neighborhoods set $\mathcal{N}_\epsilon(T_i)$ for the new cluster.
  4. Delete T_i and T_j.
  5. $c \leftarrow c - 1$.
end while

TABLE 1
Comparison of computational complexity and memory requirements of different algorithms. (C_V is the number of non-zero elements in vector $\mathbf{x}$ 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	$O(n^2[(1-\epsilon^2)L + \log L])$	$O(nf(\epsilon) + nL)$

We adopt the PPJoin+ [52] approach for this purpose. PPJoin+ uses two types of filtering, prefix filtering and suffix filtering, to filter the data samples 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 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 $(1-\epsilon^2)\sum_j x_i^{(k)}, \epsilon < 1$ [52]. Suffix filtering, 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 implement 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, which shows that Spilltree-BRT has the least time complexity. However, it requires more memory.

TABLE 2
Comparison of likelihood for query data (the results of BRT-vMF are positive because the vMF distribution is density instead of discrete probability).

Methods	Likelihood
BRT-DCM	$-1.9140816 \times 10^7 \pm 3.81483 \times 10^5$
kNN-BRT-DCM	$-1.9169448 \times 10^7 \pm 3.82904 \times 10^5$
Spilltree-BRT-DCM	$-1.9258698 \times 10^7 \pm 3.62756 \times 10^5$
PPJoin-BRT-DCM	$-1.9261274 \times 10^7 \pm 4.00327 \times 10^5$
BRT-vMF	$1.8358411 \times 10^8 \pm 5.01538 \times 10^6$
kNN-BRT-vMF	$1.8354566 \times 10^8 \pm 4.99842 \times 10^6$
Spilltree-BRT-vMF	$1.8353889 \times 10^8 \pm 5.04271 \times 10^6$
PPJoin-BRT-vMF	$1.7894110 \times 10^8 \pm 5.07942 \times 10^6$

TABLE 3
Comparison of likelihood for news data (the results of BRT-vMF are positive because the vMF distribution is density instead of discrete probability).

Methods	Likelihood
BRT-DCM	$-1.1727374 \times 10^6 \pm 1.68893 \times 10^4$
kNN-BRT-DCM	$-1.1727220 \times 10^6 \pm 1.65647 \times 10^4$
Spilltree-BRT-DCM	$-1.1727759 \times 10^6 \pm 1.68754 \times 10^4$
PPJoin-BRT-DCM	$-1.1728185 \times 10^6 \pm 1.65829 \times 10^4$
BRT-vMF	$1.0722783 \times 10^8 \pm 1.01769 \times 10^6$
kNN-BRT-vMF	$1.0722238 \times 10^8 \pm 1.01736 \times 10^6$
Spilltree-BRT-vMF	$1.0695835 \times 10^8 \pm 1.01597 \times 10^6$
PPJoin-BRT-vMF	$1.0657703 \times 10^8 \pm 1.00084 \times 10^6$

6 Experiments

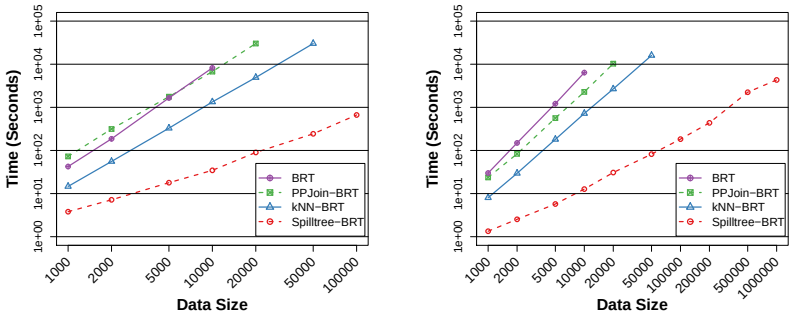
To demonstrate the effectiveness and scalability of the proposed taxonomy building algorithms, we tested the original BRT and the nearest-neighbor-based methods on two real-world data sets. The nearest-neighbor-based methods include kNN-BRT, Spilltree-BRT, and PPJoin-BRT. In this section, we introduce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

6.1 Data Sets

In all the experiments, we tested different algorithms based on two data sets. The first one was a query data set consisting of 114,076 queries, where the concept and context information were extracted using the methods introduced in Section 3.1. The vocabulary size was 912,792 and the average number of non-zero elements in the data was 935. Here, the features was created by expanding the query with the related search snippets and concepts. We also evaluated the algorithms on a larger news data set. It contained one million news articles. The vocabulary size was 252,174 and the average number of words was 242. All the experiments were conducted on a 64-bit server with 2.5GHz CPU and 32G of memory. The memory cost for the whole news data set was around 10G.

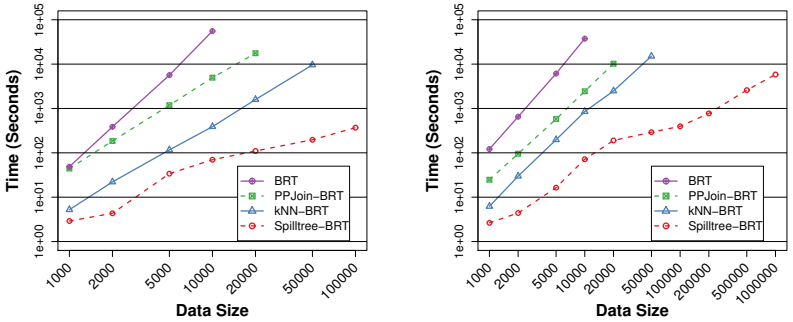
6.2 Likelihood of Taxonomy Building

We compared the likelihood of different algorithms. Likelihood $p(\mathcal{D}|T)$ is utilized to measure how much the model fits the data. A larger likelihood value means that the tree better fits the data. If we want to compare different trees, likelihood can be used to judge whether the two trees can fit the data equally as well. In this



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

Fig. 4. Time cost comparison of different algorithms based the DCM distribution.



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

Fig. 5. Time cost comparison of different algorithms based on the vMF distribution.

experiment, we use the likelihood measure to evaluate whether the fitness of the approximate algorithm is comparable to that of the original BRT, following the original BRT paper [5].

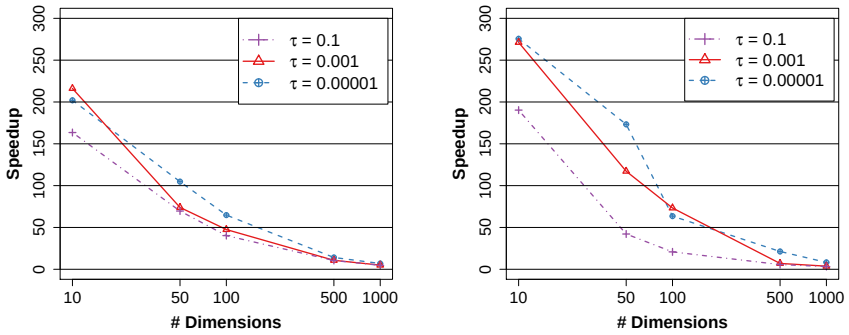
As shown in Tables 2 and 3, the nearest-neighbor-based methods are comparable with the original BRT. Particularly, the likelihood of kNN-BRT was sometimes even better than the original BRT. The reason is that searching for the candidate pairs from the nearest neighbors can reduce the data noise. Therefore, it led to a better local optima. This phenomenon has also been observed in [8].

6.3 Scalability of Taxonomy Building

To test the scalability of the algorithms, we first conducted 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 was set to 10. The overlapping parameter τ used in Spilltree was set to 0.001. We show the results based on DCM distribution modeling in Fig. 4 and the results based on vMF distribution modeling in Fig. 5.

As shown in Figs. 4(a), 4(b), 5(a), and 5(b), the Spilltree-BRT algorithm was the fastest. kNN-BRT also performed faster than BRT and PPJoin-BRT. PPJoin-BRT was not as fast as expected because the process of finding the nearest neighbors must be re-run if there is no candidate cluster pairs to merge. Particularly, for the query data set with the DCM distribution, PPJoin-BRT was even worse than the original BRT algorithm. This is due to the fact that PPJoin was significantly affected by the average length of documents. The average length of documents in the query data is about four times that of the news data. For the news data set, PPJoin-BRT performed better than BRT.

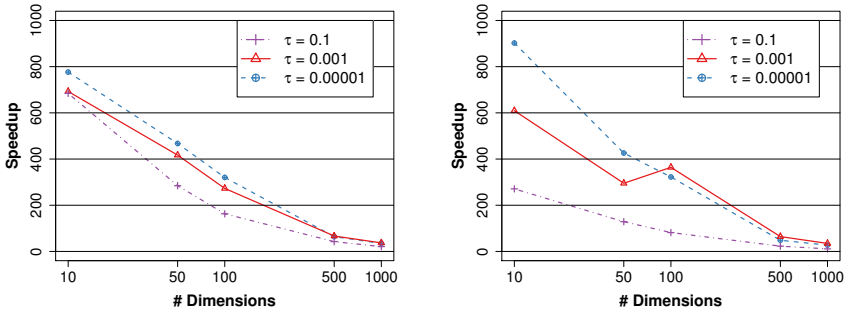
Since Spilltree-BRT is the fastest algorithm, we also investigated how different Spilltree parameters affect time cost. The following experiments were conducted with 10,000 data samples. The comparison results between the nearest-neighbor-based methods and the original BRT are shown in Figs. 6(a), 6(b), 7(a), and 7(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 samples we allowed to be overlapped by different tree nodes in a Spilltree. As shown in the figures, the larger overlapping parameter resulted in slower algorithm execution since it made the algorithm backtrack more often to the parent nodes [30]. Moreover, in each curve, we also show how the projected dimensions affected the time cost. Generally, fewer dimensions led to faster algorithm execution. Spilltree-BRT can be 200–300 times faster for the DCM distribution and 600–1000 times faster for the vMF distribution than BRT when we projected the data onto a ten-dimensional space.



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

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

Fig. 6. The impact of different parameters and settings for kNN based methods (10,000 data samples).



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

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

Fig. 7. The impact of different parameters and settings for kNN based methods (10,000 data samples).

6.4 Top KNN Search

To investigate the quality of the proposed taxonomy building method, we also applied it to the top KNN search problem. This problem was defined as: given a query in the form of short text, we retrieved similar queries based on the concept and context features from an already built tree. We used K to distinguish the notation of k used for kNN-based construction of taxonomy. In this experiment, we investigated whether Spilltree, kNN-BRT, and Spilltree-BRT can benefit this problem.

We first selected 1,000 queries to build the taxonomy trees by leveraging the Spilltree algorithm and a different algorithms of BRTs. The experiments of Spilltree were conducted by doing a grid search for tolerance parameter $\tau = \{0.2, 0.3, 0.4\}$. The experiments of all BRT-related algorithms were conducted by doing a grid search for parameters $\alpha = \{5, 10, 15\}$ (Eq. (8)), $\kappa = \kappa_0 = \{1, 100, 10, 000\}$ (Eq. (10)) and $\gamma = \{0.1, 0.5, 0.9\}$ (Eq. (3)). We have chosen the best results from this experiment. Then we randomly selected 100 queries to search the KNN of each query. The ground truth was generated based on the cosine similarity. We evaluated 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 Tables 4, 5, and 6.

TABLE 4
Comparison of the accuracy of Spilltree for top K searches. (d is the dimension of random projection.)

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

From the analysis of the results, we can draw the following conclusions: First, the dimension of random projection significantly affects the accuracy of Spilltree. In general, the more random projection features we used, the better the accuracy was. 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 kNN-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 were significantly better than Spilltree. Increasing d or k can both improve the accuracy. This is because larger d resulted in a better searching accuracy of the nearest neighbors, while larger k led to more

TABLE 5
Comparison of the accuracy of BRT using the DCM distribution for top K searches. (k is the number of the nearest neighbors.)

Algorithm	Top 5	Top 10
BRT-DCM	0.894±0.252	0.929±0.180
PPJoin-BRT-DCM	0.868±0.247	0.852±0.271
kNN-BRT-DCM ($k = 1$)	0.910±0.248	0.903±0.253
Spilltree-BRT-DCM		
($d = 10, k = 1$)	0.738±0.391	0.761±0.365
($d = 50, k = 1$)	0.880±0.276	0.868±0.280
($d = 100, k = 1$)	0.901±0.234	0.897±0.229
($d = 500, k = 1$)	0.909±0.226	0.918±0.205
kNN-BRT-DCM ($k = 5$)	0.920±0.211	0.930±0.182
Spilltree-BRT-DCM		
($d = 10, k = 5$)	0.694±0.415	0.691±0.410
($d = 50, k = 5$)	0.897±0.241	0.922±0.198
($d = 100, k = 5$)	0.928±0.192	0.939±0.162
($d = 500, k = 5$)	0.928±0.196	0.946±0.153
kNN-BRT-DCM ($k = 20$)	0.893±0.254	0.898±0.248
Spilltree-BRT-DCM		
($d = 10, k = 20$)	0.810±0.335	0.830±0.313
($d = 50, k = 20$)	0.905±0.239	0.931±0.189
($d = 100, k = 20$)	0.921±0.214	0.946±0.157
($d = 500, k = 20$)	0.933±0.183	0.955±0.134

TABLE 6
Comparison of the accuracy of BRT using the vMF distribution for the top K searches. (d is the dimension of random projection.)

Algorithm	Top 5	Top 10
BRT-vMF	0.934±0.179	0.951±0.139
PPJoin-BRT-vMF	0.948±0.222	0.948±0.222
kNN-BRT-vMF($k = 1$)	0.936±0.176	0.950±0.142
Spilltree-BRT-vMF		
($d = 10, k = 1$)	0.682±0.423	0.734±0.326
($d = 50, k = 1$)	0.864±0.309	0.865±0.297
($d = 100, k = 1$)	0.905±0.234	0.914±0.210
($d = 500, k = 1$)	0.897±0.245	0.924±0.193
kNN-BRT-vMF($k = 5$)	0.941±0.160	0.955±0.116
Spilltree-BRT-vMF		
($d = 10, k = 5$)	0.759±0.391	0.767±0.375
($d = 50, k = 5$)	0.923±0.206	0.930±0.179
($d = 100, k = 5$)	0.938±0.186	0.948±0.152
($d = 500, k = 5$)	0.936±0.183	0.954±0.132
kNN-BRT-vMF($k = 20$)	0.937±0.170	0.955±0.119
Spilltree-BRT-vMF		
($d = 10, k = 20$)	0.842±0.309	0.839±0.307
($d = 50, k = 20$)	0.933±0.189	0.945±0.150
($d = 100, k = 20$)	0.934±0.185	0.947±0.150
($d = 500, k = 20$)	0.932±0.185	0.956±0.127

candidate cluster pairs which increase the opportunity for finding the right clusters to merge. Fourth, the results of using DCM distribution modeling and vMF distribution modeling were consistent. The modeling distribution did not seem to affect the final results of K-nearest neighbor search in this data set. Moreover, the results of vMF were much better than those of DCM because vMF was consistent with the baseline cosine similarity.

We also tested the search time for Spilltree and Spilltree-BRT. 110K query data were used to build the trees and 200 queries were randomly selected to test the retrieval time. As shown in Table 7, Spilltree performs faster with a lower dimension of random projections. However, the accuracy was not good enough (see Table 4). With higher dimensions, the time cost of Spilltree increased very quickly. The major reasons are: (1) It costs more time to compare the similarity between two points for higher dimensionality; (2) Given the same parameter configuration, Spilltree will search more points in a high dimensional space. As shown in Tables 5 and 6, the accuracy of Spilltree-BRT was more than 90% when $d = 50$ and $k = 20$. Thus, we performed a grid search for $d = \{10, 50\}$ for Spilltree-BRT. All results were achieved in an acceptable amount of time. The search time of Spilltree-BRT depends on the structure of the tree. Thus, changing d and k only affected the building time of Spilltree-BRT and did not monotonically affect the search time.

Moreover, we found that when d was smaller, the time of Spilltree-BRT using the vMF distribution was comparable to or slightly larger than the one using the DCM distribution. However, when d was larger, the time of trees with the vMF distribution was smaller than with the DCM distribution. This may be because the trees based on the vMF distribution were much more skewed than the trees based on the DCM distribution. Therefore, it required more time to search for a good result. We can also verify from Tables 5 and 6 that the accuracy of vMF was lower than DCM when d was small. When d became larger, the tree's quality was much better. The time of vMF was much smaller than that of DCM because we used different mechanisms to accelerate the computation of these two distributions. For DCM, we cached all the possible logarithm values involved in the tree construction, and then looked up the table for a quick computation of log-likelihood in Eq. (9). For vMF, since the Bessel function in Eq. (12) was difficult to compute, we leveraged the approximation that has been used in [14] (in page 5, Section 5). The computational cost for the Bessel function can be significantly reduced. In this case, the search cost for vMF involved less computational operations than DCM and thus the time

TABLE 7
Comparison of the time (in 10^{-6} seconds) of top K search. (d is the dimension of random projection.)

	$d = 10$		$d = 50$		$d = 100$		$d = 500$	
Algorithm	Top 5	Top 10	Top 5	Top 10	Top 5	Top 10	Top 5	Top 10
Spilltree	2.29	1.86	657.02	693.02	1,103.84	1,159.44	3,867.02	4,443.81
Spilltree-BRT-DCM ($k = 1$)	34.38	35.94	143.75	251.56	1,724.99	2,195.30	2,425.03	2,621.91
Spilltree-BRT-DCM ($k = 5$)	48.44	40.63	34.38	43.75	1,782.82	1,814.07	2,135.97	2,223.47
Spilltree-BRT-DCM ($k = 20$)	148.44	371.88	50.00	50.01	1,496.89	1,482.82	2,121.89	2,112.51
Spilltree-BRT-vMF ($k = 1$)	203.13	203.13	85.94	85.94	231.25	267.19	1,975.01	2,092.20
Spilltree-BRT-vMF ($k = 5$)	140.63	117.20	203.13	242.19	46.88	31.25	442.19	657.83
Spilltree-BRT-vMF ($k = 20$)	882.82	890.63	414.07	406.25	78.13	101.56	309.20	451.30

of vMF was much smaller.

7 Implementation Issues

Since text data is high-dimensional and sparse, and the data we deal with is very large, we need to pay attention to some implementation issues. In addition to the computation of the Bessel function mentioned in the previous section, we also want to introduce two more implementation issues.

The high-dimensional sparse data was stored as *key-value pairs*. We compared it with two implementations, a hash-based dictionary and parallel sorted lists. The hash-based dictionary indexed the keys with "int" type and, when comparing two dictionaries of data, it takes $O(1)$ to search for each element. With the parallel sorted list, we first kept two lists of keys and values, and then sorted each data samples with its keys. In both implementations, for vector-level operations such as product, they take $O(\min\{\mathrm{nz}(\mathbf{x}_i), \mathrm{nz}(\mathbf{x}_j)\})$ time, where $\mathrm{nz}(\mathbf{x}_i)$ is the non-zero elements of a vector $\mathbf{x}_i$. The parallel sorted list takes $O(\mathrm{nz}(\mathbf{x}_i) \log[\mathrm{nz}(\mathbf{x}_i)])$ to sort the lists by keys. However, when the vocabulary size is large, i.e., more than one million words, the hash-based dictionary often generated a number of key conflicts. Moreover, it took more CPU time to compute the hash function to retrieve a key. Therefore, with real-world applications, we found that parallel sorted lists were 5 and 150 times faster than the hash-based dictionary for small and large numbers of elements in the vector, respectively.

When the data set is very large, there are numerical underflow issues when we compute the log-likelihood of Eq. (2). The likelihood can easily increase to 10^{-1000}–10^{-10000}, and thus we cannot do multiplication first and then compute the logarithm. Instead, we have adopted the following methods to incrementally compute the likelihood[5]:

$$\log(\prod_i p_i) \rightarrow \sum_i \log(p_i); \tag{14}$$

$$\log(\sum_i p_i) \rightarrow a + \log\left[\sum_i \exp[\log(p_i) - a]\right]$$

where $a = \max_i \log(p_i)$. In this way, computing the likelihood of a very large data set avoids the probability product and does not encounter numerical problems.

8 Conclusion and Future Work

This paper presents an approach to automatically derive a domain-dependent taxonomy from a set of keyword phrases by leveraging both a general knowledgebase and keyword search. Our approach can be regarded as specifying the general-purpose knowledge (or world knowledge) of taxonomy to a domain specified task by three steps: (1) world knowledge formulation, (2) concept deduction for domain keywords, and (3) hierarchical cluster induction. Compared with previous work [9], [40], the introduction of concepts in the representation enables the descriptive power of keyword clusters and thus better characterizes the abstractive information of the keywords. The incorporation of context information leads to better coverage complementary to the general-purpose knowledge. To better build the taxonomy, we provide two types of text distribution modeling methods and 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.

5. http://blog.smola.org/post/987977550/log-probabilities-semirings-and-floating-point-numbers

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.

References

[1] R. P. Adams, Z. Ghahramani, and M. I. Jordan. Tree-structured stick breaking for hierarchical data. In *NIPS*, 2010.
[2] A. Banerjee, I. S. Dhillon, J. Ghosh, and S. Sra. Clustering on the unit hypersphere using von mises-fisher distributions. *Journal on Machine Learning Research*, 6:1345–1382, 2005.
[3] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI*, pages 2670–2676, 2007.
[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
[5] C. Blundell, Y. W. Teh, and K. A. Heller. Bayesian rose trees. In *UAI*, pages 65–72, 2010.
[6] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250, 2008.
[7] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, pages 1306–1313, 2010.
[8] 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.
[9] S.-L. Chuang and L.-F. Chien. Towards automatic generation of query taxonomy: A hierarchical query clustering approach. In *ICDM*, pages 75–82, 2002.
[10] P. Cimiano, S. Staab, and J. Tane. Automatic acquisition of taxonomies from text: FCA meets NLP. In *ECML/PKDD Workshop on Adaptive Text Extraction and Mining, Cavtat-Dubrovnik, Croatia*, pages 10–17, 2003.
[11] W. Dakka and P. G. Ipeirotis. Automatic extraction of useful facet hierarchies from text databases. In *ICDE*, pages 466–475, 2008.
[12] S. Dasgupta and A. Gupta. An elementary proof of the Lohnson-Lindenstrauss lemma. Technical report, MIT, 1999. ICSI Technical Report TR-99-006.
[13] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.
[14] C. Elkan. Clustering documents with an exponential-family approximation of the dirichlet compound multinomial distribution. In *ICML*, pages 289–296, 2006.
[15] O. Etzioni, M. Cafarella, and D. Downey. Webscale information extraction in knowitall (preliminary results). In *WWW*, 2004.

[16] D. Faure and C. Nédellec. A corpus-based conceptual clustering method for verb frames and ontology acquisition. In *In LREC workshop on adapting lexical and corpus resources to sublanguages and applications*, pages 5–12, 1998.

[17] D. Faure and C. Nédellec. Knowledge acquisition of predicate argument structures from technical texts using machine learning: The system ASIUM. In *EKAW*, pages 329–334, 1999.

[18] C. Fellbaum, editor. *WordNet: an electronic lexical database*. MIT Press, 1998.

[19] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[20] G. Grefenstette. Sextant: Exploring unexplored contexts for semantic extraction from syntactic analysis. In *ACL*, pages 324–326, 1992.

[21] R. Grishman and J. Sterling. Generalizing automatically generated selectional patterns. In *COLING*, pages 742–747, 1994.

[22] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, pages 539–545, 1992.

[23] K. A. Heller and Z. Ghahramani. Bayesian hierarchical clustering. In *ICML*, pages 297–304, 2005.

[24] D. Hindle. Noun classification from predicate-argument structures. In *ACL*, pages 268–275, 1990.

[25] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31:264–323, September 1999.

[26] D. A. Knowles and Z. Ghahramani. Pitman-Yor diffusion trees. In *UAI*, pages 410–418, 2010.

[27] S. LAMROUS and M. TAILEB. Divisive hierarchical k-means. In *CIMCA*, page 18, 2006.

[28] T. Lee, Z. Wang, H. Wang, and S. won Hwang. Web scale taxonomy cleansing. pages 1295–1306, 2011.

[29] D. B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, 1989.

[30] T. Liu, A. W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *NIPS*, pages 825–832, 2005.

[31] X. Liu, Y. Song, S. Liu, and H. Wang. Automatic taxonomy construction from keywords. In *KDD*, pages 1433–1441, 2012.

[32] R. E. Madsen, D. Kauchak, and C. Elkan. Modeling word burstiness using the Dirichlet distribution. In *ICML*, pages 545–552, 2005.

[33] I. Mani, K. Samuel, K. Concepcion, and D. Vogel. Automatcally inducing ontologies from corpora. In *Workshop on Computational Terminology*, 2004.

[34] O. Medelyan, S. Manion, J. Broekstra, A. Divoli, A.-L. Huang, and I. H. Witten. Constructing a focused taxonomy from a document collection. In *ESWC*, pages 367–381, 2013.

[35] R. Navigli, P. Velardi, and S. Faralli. A graph-based algorithm for inducing lexical taxonomies from scratch. In *IJCAI*, pages 1872–1877, 2011.

[36] S. P. Ponzetto and M. Strube. Deriving a large-scale taxonomy from wikipedia. In *AAAI*, pages 1440–1445, 2007.

[37] H. Poon and P. Domingos. Unsupervised ontology induction from text. In *ACL*, pages 296–305, 2010.

[38] E. Sadikov, J. Madhavan, L. Wang, and A. Y. Halevy. Clustering query refinements by user intent. In *WWW*, pages 841–850, 2010.

[39] R. S. Savage, K. A. Heller, Y. Xu, Z. Ghahramani, W. M. Truman, M. Grant, K. J. Denby, and D. L. Wild. R/bhc: fast bayesian hierarchical clustering for microarray data. *BMC Bioinformatics*, 10, 2009.

[40] D. Shen, M. Qin, W. Chen, Q. Yang, and Z. Chen. Mining web query hierarchies from clickthrough data. In *AAAI*, pages 341–346, 2007.

[41] Y. Song, H. Wang, W. Chen, and S. Wang. Transfer understanding from head queries to tail queries. In *CIKM*, pages 1299–1308, 2014.

[42] Y. Song, H. Wang, Z. Wang, H. Li, and W. Chen. Short text conceptualization using a probabilistic knowledgebase. In *IJCAI*, pages 2330–2336, 2011.

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

[44] E. Stoica and M. A. Hearst. Automating creation of hierarchical faceted metadata structures. In *NAACL HLT*, 2007.

[45] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[46] M. Telgarsky and S. Dasgupta. Agglomerative bregman clustering. In *ICML*, 2012.

[47] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.

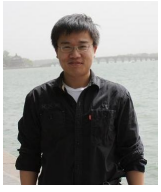
[48] Z. Wang, H. Wang, and Z. Hu. Head, modifier, and constraint detection in short texts. In *ICDE*, pages 280–291, 2014.

[49] 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.

[50] W. Wong, W. Liu, and M. Bennamoun. Ontology learning from text: A look back and into the future. *ACM Comput. Surv.*, 44(4):20, 2012.

[51] W. Wu, H. Li, H. Wang, and K. Q. Zhu. Probase: A probabilistic taxonomy for text understanding. In *SIGMOD*, pages 481–492, 2012.

[52] 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.



Yangqiu Song is a postdoctoral researcher at University of Illinois at Urbana-Champaign. He received his B.E. and PH.D. degree from Tsinghua University, China. His current research focuses on using machine learning and data mining to extract and infer insightful knowledge from big data, including the techniques of large scale learning algorithms, natural language understanding, text mining and visual analytics, and knowledge engineering.

Shixia Liu is an associate professor at Tsinghua University. Her research interests include visual text analytics, visual social analytics, and graph visualization. Before joining Tsinghua University, she worked as a lead researcher at Microsoft Research Asia and a research staff member at IBM China Research Lab. She received a B.S. and M.S. in Computational Mathematics from Harbin Institute of Technology, a Ph.D. in Computer Science from Tsinghua University. Shixia Liu is the corresponding author of this paper.

Xueqing Liu received the diploma degree in computer science from the Tsinghua University, China, in 2012. Currently, she is working toward the graduate degree in the Department of Computer Science, University of Illionois Urbana-Champaign. Her research interests include information retrieval and data mining.

Haixun Wang joined Google Research (Mountain View) in 2013. He received the B.S. and M.S. degrees in Computer Science from Shanghai Jiao Tong University in 1994 and 1996, the Ph.D. degree in Computer Science from University of California, Los Angeles in June, 2000. His research interests include text analytics, natural language processing, knowledge base, semantic network, artificial intelligence, database language and system, graph data management, etc.