Understand Short Texts by Harvesting and Analyzing Semantic Knowledge

Wen Hua, Zhongyuan Wang, Haixun Wang, Member, IEEE, Kai Zheng, Member, IEEE, and Xiaofang Zhou, Senior Member, IEEE

Abstract—Understanding short texts is crucial to many applications, but challenges abound. First, short texts do not always observe the syntax of a written language. As a result, traditional natural language processing tools, ranging from part-of-speech tagging to dependency parsing, cannot be easily applied. Second, short texts usually do not contain sufficient statistical signals to support many state-of-the-art approaches for text mining such as topic modeling. Third, short texts are more ambiguous and noisy, and are generated in an enormous volume, which further increases the difficulty to handle them. We argue that semantic knowledge is required in order to better understand short texts. In this work, we build a prototype system for short text understanding which exploits semantic knowledge provided by a well-known knowledgebase and automatically harvested from a web corpus. Our knowledge-intensive approaches disrupt traditional methods for tasks such as text segmentation, part-of-speech tagging, and concept labeling, in the sense that we focus on semantics in all these tasks. We conduct a comprehensive performance evaluation on real-life data. The results show that semantic knowledge is indispensable for short text understanding, and our knowledge-intensive approaches are both effective and efficient in discovering semantics of short texts.

Index Terms—Short text understanding, text segmentation, type detection, concept labeling, semantic knowledge.

1 Introduction

Infor mation explosion highlights the need for machines to better understand natural language texts. In this paper, we focus on short texts which refer to texts with limited context. Many applications, such as web search and microblogging services etc., need to handle a large amount of short texts. Obviously, a better understanding of short texts will bring tremendous value.

One of the most important tasks of text understanding is to discover hidden semantics from texts. Many efforts have been devoted to this field. For instance, named entity recognition (NER) [1] [2] locates named entities in a text and classifies them into predefined categories such as persons, organizations, locations, etc. Topic models [3] [4] attempt to recognize “latent topics”, which are represented as probabilistic distributions on words, from a text. Entity linking [5] [6] [7] [8] [9] [10] [11] focuses on retrieving “explicit topics” expressed as probabilistic distributions on an entire knowledgebase. However, categories, “latent topics”, as well as “explicit topics” still have a semantic gap with humans’ mental world. As stated in Psychologist Gregory Murphy’s highly acclaimed book [12], “concepts are the glue that holds our mental world together”. Therefore, we define short text understanding as to detect concepts mentioned in a short text.

Fig. 1 demonstrates a typical strategy for short text understanding which consists of three steps:

- Text Segmentation - divide a short text into a collection of terms (i.e., words and phrases) contained in a vocabulary (e.g., “book disneyland hotel california” is segmented as [book disneyland hotel california]);
- Type Detection - determine the types of terms and recognize instances (e.g., both “disneyland” and “california” are recognized as instances in Fig. 1, while “book” is recognized as a verb and “hotel” a concept);
- Concept Labeling - infer the concept of each instance (e.g., “disneyland” and “california” refer to the concept theme park and state respectively in Fig. 1).

Overall, three concepts are detected from short text “book disneyland hotel california” using this strategy, namely theme park, hotel, and state in Fig. 1.

Fig. 1. An example of short text understanding.

Although the three steps for short text understanding sound quite simple, challenges still abound and new approaches must be introduced to handle them. In the following, we use several examples to illustrate such a need.

Challenge 1 (Ambiguous Segmentation).

- “april in paris lyrics” vs. “vacation april in paris”

Both a term and its sub-terms can be contained in the vocabulary, leading to multiple possible segmentations for a short text. However, a valid segmentation should maintain semantic coherence. For example, two segmentations can be derived from “april in paris lyrics”, namely [april in paris lyrics] and [april paris lyrics]. However, the former is a better segmentation according to the knowledge that “lyrics” is more semantically related with
The rest of this paper is organized as follows: in Sec. 2, we briefly summarize related work in the literature of text processing; then we introduce some notations used in this paper, and define the problem of short text understanding formally in Sec. 3; our approaches and experiments are described in Sec. 4 and Sec. 5 respectively, followed by a brief conclusion and discussion of future work in Sec. 6.

2 RELATED WORK

In this section, we discuss related work in three aspects: text segmentation, POS tagging, and semantic labeling.

Text Segmentation. We consider text segmentation as to divide a text into a sequence of terms. Existing approaches can be classified into two categories: statistical approaches and vocabulary-based approaches. Statistical approaches, such as N-gram Model [30] [31] [32], calculate the frequencies of words co-occurring as neighbors in a training corpus. When the frequency exceeds a
predefined threshold, the corresponding neighboring words can be treated as a term. Vocabulary-based approaches extract terms in a streaming manner by checking for existence or frequency of a term in a predefined vocabulary. In particular, the Longest Cover method, which is widely-adopted for text segmentation [13] [14] [15] [16] [17] due to its simplicity and real-time nature, searches for longest terms contained in a vocabulary while scanning the text. The most obvious drawback of existing methods for text segmentation is that they only consider surface features and ignore the requirement of semantic coherence within a segmentation. This will lead to incorrect segmentations in cases such as “vacation april in paris” described in Challenge 1. To this end, we propose to exploit context semantics when conducting text segmentation.

**POS Tagging.** POS tagging determines lexical types (i.e., POS tags) of words in a text. Mainstream POS tagging algorithms fall into two categories: rule-based approaches and statistical approaches. Rule-based POS taggers attempt to assign POS tags to unknown or ambiguous words based on a large number of handcrafted [18] [19] or automatically learned [20] [21] [22] linguistic rules. Statistical POS taggers avoid the cost of constructing tagging rules by building a statistical model automatically from a corpora and labeling untagged texts based on those learned statistical information. Most of the widely-adopted statistical approaches employ the well-known Markov Model [23] [24] [25] [26] [27] [28] [29] which learns both lexical probabilities ($P(tag|word)$) and sequential probabilities ($P(tag_{i} | tag_{i-1},tag_{i-2},...,tag_{i-n})$) from a labeled corpora and tags a new sentence by searching for tag sequence that maximizes the combination of lexical and sequential probabilities. Note that both rule-based and statistical approaches to POS tagging rely on the assumption that texts are correctly structured. In other words, texts should satisfy tagging rules or sequential relations between consecutive tags. However, this is not always the case for short texts. More importantly, all of the aforementioned work only considers lexical features and ignores word semantics. This will lead to mistakes sometimes, as illustrated in the case of “pink songs” described in Challenge 3. Our work attempts to build a tagger which considers both lexical features and underlying semantics for type detection.

**Semantic Labeling.** Semantic labeling discovers hidden semantics from a natural language text. According to the representation of semantics, existing work on semantic labeling can be roughly classified into three categories, namely named entity recognition (NER), topic modeling, and entity linking. NER locates named entities in a text and classifies them into predefined categories (e.g., persons, organizations, locations, times, quantities and percentages, etc.) using linguistic grammar-based techniques as well as statistical models like CRF [1] and HMM [2]. Topic models [3] [4] attempt to recognize “latent topics”, which are represented as probabilistic distributions on words, based on observable statistical relations between texts and words. Entity linking [5] [6] [7] [8] [9] [10] [11] employs existing knowledgebases and focuses on retrieving “explicit topics” expressed as probabilistic distributions on the entire knowledgebase. Despite the high accuracy that has been achieved by existing work on semantic labeling, there are still some limitations. First, categories, “latent topics”, as well as “explicit topics” are different from human-understandable concepts. Second, short texts do not always observe the syntax of a written language which, however, is an indispensable feature used in mainstream NER tools. Third, short texts usually do not contain sufficient content to support statistical models like topic models.

The work most related to ours are conducted by Song et al. [16] and Kim et al. [17] respectively, which also represent semantics as concepts. [16] employs the Bayesian Inference mechanism to conceptualize instances and short texts, and eliminates instance ambiguity based on homogeneous instances. [17] captures semantic relatedness between instances using a probabilistic topic model (i.e., LDA), and disambiguates instances based on related instances. In this work, we observe that other terms, such as verbs, adjectives, and attributes, can also help with instance disambiguation. Therefore, we incorporate type detection into our framework for short text understanding and conduct instance disambiguation based on various types of context information.

### 3 Problem Statement

In this section, we briefly introduce some concepts and notations used in the paper. Then we formally define the problem of short text understanding, and give an overview of our framework.

#### 3.1 Preliminary Concepts

<table>
<thead>
<tr>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>short text book hotel california</td>
</tr>
<tr>
<td>p</td>
<td>segmentation (book hotel california)</td>
</tr>
<tr>
<td>t</td>
<td>term hotel, california, hotel california</td>
</tr>
<tr>
<td>t_r</td>
<td>type book, book, company</td>
</tr>
<tr>
<td>t_r_c</td>
<td>concept cluster vector ([theme park, park, [company]...)</td>
</tr>
</tbody>
</table>

#### 3.1.1 Vocabulary, Term, and Segmentation

**Definition 1 (vocabulary).** A vocabulary is a collection of words and phrases (of a certain language).

We download a list of English verbs and adjectives from an online dictionary - YourDictionary\(^3\), and harvest a collection of attributes, concepts, and instances from a well-known knowledgebase - Probase\(^4\). Altogether, they constitute our vocabulary. To cope with the noise contained in short texts, we further extend the vocabulary to incorporate abbreviations and nicknames of instances. These can be obtained from web corpus or existing knowledgebases. In particular, we construct a list of synonyms\(^5\) from Wikipedia’s redirect links, disambiguation links, as well as hypertexts and hyperlinks between Wikipedia articles. For example, from the redirect information between “nyc” and “new york city”, we know that “nyc” is an abbreviation of “new york city”; similarly, form the disambiguation link of “big apple”, we obtain that “big apple” is a nickname of “new york city”.

**Definition 2 (term).** A term $t$ is an entry in the vocabulary.

We represent a term as a sequence of words, and denote $|t|$ as the length (number of words) of term $t$. Example terms are “hotel”, “california”, and “hotel california”, etc.

**Definition 3 (segmentation).** A segmentation $p$ of a short text is a sequence of terms $p = [t_1, ... , t_l]$ such that:


\(^3\) http://www.yourdictionary.com/
\(^4\) http://research.microsoft.com/en-us/projects/probase/
\(^5\) The synonym dictionary is publicly available at http://probase.msra.cn/dataset.aspx.
1) terms cannot overlap with each other, i.e., $t_i \cap t_{i+1} = \emptyset, \forall i$; 
2) every non-stopword in the short text should be covered by a term, i.e., $s - \bigcup_{i=1}^L t_i \subset \text{stopwords}$.

For example, a possible segmentation of “vacation april in paris” is \{vacation\} \{april\} \{in\} \{paris\}, where only stopword “in” is ignored from the original short text. For “new york times square,” although both “new york times” and “times square” are terms in our vocabulary, \{new\} \{york\} \{times\} \{square\} is invalid according to our restrictions because the two terms overlap with each other.

### 3.1.2 Type and Typed-Term

**Definition 4 (type).** Type denotes the lexical or semantic role a term plays in a text.

Lexical types include verb and adjective. We consider lexical types in this work for two reasons. First, verbs and adjectives can help with instance disambiguation, as discussed in Challenge 4. Second, one of the most important applications of short text understanding is to calculate semantic similarity between short texts, whereas incorrect detection of lexical types will cause mistakes in calculating semantic similarity. Consider the following example, wherein “watch” is a verb in “watch free movie” while a concept in “watch omega”. These two short texts are semantically dissimilar, as the former is about watching a free movie while the latter searches for a famous brand of watch called “omega”. However, if we do not consider lexical types and label “watch” in “watch free movie” as an instance or a concept, it will lead to incorrectly high similarity between these two short texts.

- “watch\_free\_movie” vs. “watch\_omega”

Semantic types include attribute, concept, and instance. POS tagging only distinguishes lexical types and ignores the differences between semantic types, which will also cause mistakes in calculating semantic similarity between short texts. In the examples below, “population” can be both an attribute of concept country and an instance of concept geographic statistics. The first pair of short texts are both about geographic statistics of a country (i.e., same concept but different attributes), while the second pair of short texts are about statistical information of a country and an animal respectively (i.e., same attribute but different concepts). This leads to larger semantic similarity in the first pair than the second pair. We can see that concepts and instances contribute more to the semantics of a short text than attributes, which verifies the necessity to distinguish semantic types.

- “population china” vs. “climate china”
- “population china” vs. “population panda”

**Definition 5 (typed-term).** A typed-term $\bar{t}_r$ refers to a term with a specific type $\bar{t}_r$.

A term can be labeled with multiple types and thus can be mapped to multiple typed-terms. We denote the set of possible typed-terms for a term as $T = \{t_i|1 \leq i \leq m\}$, which can be obtained directly from the vocabulary. For example, we observe that term “book” appears in verb-list, concept-list as well as instance-list of our vocabulary, thus the possible typed-terms of “book” are \{book\_v, book\_c, book\_i\}.

### 3.1.3 Knowledgebase and Concept Cluster Vector

**Definition 6 (knowledgebase).** A knowledgebase stores mappings between instances and concepts. Some existing knowledgebases also associate each concept with certain attributes.

In this work, we use Probase [33] as our knowledgebase. Probase is a huge semantic network of concepts (e.g., country), instances (e.g., china) and attributes (e.g., population). It mainly focuses on two types of relationships, namely the isA relationship between instances and concepts (e.g., china isA country) and the isAttributeOf [34] relationship between attributes and concepts (e.g., population isAttributeOf country). We use Probase for two reasons. First, Probase’s broad coverage of concepts makes it more representative of humans’ mental world, in comparison with other knowledgebases such as Freebase, WordNet, DBPedia, etc. Knowledge in Probase is acquired automatically from a corpus of 1.68 billion webpages, and it contains 2.7 million concepts and 16 million instances, which results in more than 20.7 million mappings between instances and concepts. Second, unlike traditional knowledgebases that simply treat knowledge as black or white, Probase quantifies many measures such as popularity and typicality which are important to cognition.

- Popularity $p(c|e)$, e.g., $p(c = company|e = apple)$ measures how likely people will think of the concept “company” when they see the term “apple”;
- Typicality $p(e|c)$, e.g., $p(e = steve jobs|c = ceo)$ measures how likely “steve jobs” will come into mind when people think about the concept “ceo”.

We can directly obtain the semantics (i.e., concepts) of an instance from Probase. However, some of Probase’s concepts are actually similar. For example, “apple” can belong to concepts company, it company, big company, software company, etc. In order to find the most appropriate semantics of “apple” in short text “price apple”, all these concepts must be checked and removed one by one, which is obviously a waste of time. To represent semantics in a more compact manner and to speed up the process of instance disambiguation, we employ the K-Medoids [35] algorithm to cluster similar concepts contained in Probase. Our intuition is that if two concepts share many instances, they are similar to each other. Readers can refer to [36] for more details on concept clustering.

**Definition 7 (concept cluster vector).** We represent the semantics of a typed-term as a concept cluster vector $i\bar{C} = ((C_1, W_1), (C_2, W_2), ..., (C_N, W_N))$, where $C_i$ represents a concept cluster and $W_i$ its weight, as formally defined in Eq. 1.

$$i\bar{C} = \begin{cases} 
\emptyset & \text{if } r \in \{v, adj, att\} \\
\langle C, 1 \rangle & \text{if } r = c \\
\langle C_i, W_i > | 1 = i, ..., N \rangle & \text{if } r = e
\end{cases}$$

In Eq. 1, we distinguish three circumstances: 1) verbs, adjectives, and attributes have no hypernyms in Probase, thus we specifically define their concept cluster vectors as empty; 2) for a concept, only the concept cluster it belongs to will be assigned with the weight 1 and all the other concept clusters will be assigned with the weight 0; 3) for an instance, we retrieve its concepts from Probase and weigh each concept cluster by the summation of weights of containing concepts. More formally, $W_i = \sum_{c \in C_i} p(c|i)$ where $p(c|i)$ is the popularity score harvested by Probase.
3.2 Problem Definition and Framework Overview

**Definition 8 (Short Text Understanding).** Given a short text \( s \) written in a natural language, we generate a semantic interpretation of \( s \) represented as a sequence of typed-terms namely \( \vec{s} = \{ t_i | i = 1, ..., |s| \} \), and the semantics of each instance is labeled with the top-1 concept cluster.

As shown in Fig. 1, the semantic interpretation of short text “book disneyland hotel california” is \{book[6], disneyland[3], hotel[2], california[5], [time,] \}. We divide the task of short text understanding into three subtasks corresponding to the three steps mentioned in Sec. 1 respectively:

- Text Segmentation - given a short text, find the most semantically coherent segmentation;
- Type Detection - for each term, detect its best type;
- Concept Labeling - for each ambiguous instance, re-rank its concept clusters according to the context.

![Fig. 2. Framework overview.](image)

Fig. 2. Framework overview.

In the offline part, we construct index on the entire vocabulary and acquire knowledge from web corpus and existing knowledgebases. Then, we pre-calculate semantic coherence between terms which will be used for online short text understanding. In the online part, we perform text segmentation, type detection, and concept labeling, and generate a semantically coherent interpretation for a given short text.

4 Methodology

In this section, we describe the details of our framework for short text understanding, i.e., the offline and online processing respectively.

4.1 Offline Processing

A prerequisite to short text understanding is the knowledge about semantic relatedness between terms. We describe how we construct the co-occurrence network\(^6\) and quantify semantic coherence in this section. After that, we introduce the indexing strategy to allow for approximate term extraction on the vocabulary, as well as the approach to determine instance ambiguity.


4.1.1 Constructing Co-occurrence Network

We construct a co-occurrence network to model semantic relatedness. The co-occurrence network can be regarded as an undirected graph, where nodes are typed-terms and edge weight \( w(\vec{x}, \vec{y}) \) formulates the strength of semantic relatedness between typed-terms \( \vec{x} \) and \( \vec{y} \). We observe that

- Terms of different types occur in different contexts. In Fig. 3, “watch” as a verb co-occurs with concept movie while “watch” as an instance co-occurs with “buy” and “price”. Therefore, the co-occurrence network should be constructed between typed-terms instead of terms;
- The more frequently two typed-terms co-occur in a sentence, the higher the semantic relatedness will be;
- The closer two typed-terms appear in a sentence, the higher the semantic relatedness will be;
- Common terms (e.g., "item" and "object") which co-occur with almost every other term are meaningless in modeling semantic relatedness, thus the corresponding edge weights should be penalized.

Based on these observations, we build a co-occurrence network as follows: 1) We scan every distinct sentence from a web corpus, and obtain part-of-speech tags using Stanford POS tagger. For words tagged as verbs or adjectives, we derive their stems and get a collection of verbs and adjectives. For noun phrases, we check them in the vocabulary and determine their types (attribute, concept, instance) collectively by minimizing topical diversity. Our intuition is that the number of topics mentioned in a sentence is usually limited. For example, “population” can be an attribute of country as well as an instance of geographical data. Assume that the collection of noun phrases parsed from a sentence is \{“china”, “population”\}, then “population” should be labeled as an attribute in order to limit the topic of the sentence to be country only. Using this approach, we can obtain a set of attributes, concepts and instances. Take “Outlook.com is a free personal email from Microsoft” as another example. The collection of typed-terms we get after analyzing this sentence is \{outlook[1], free[1], personal[1], email[1], microsoft[1]\}. 2) Given the set of typed-terms derived from a sentence, we add a co-occur edge between each pair of typed-terms. To estimate edge weight, we first calculate the frequency of two typed-terms appearing together using the following formula:

\[
f_i(\vec{x}, \vec{y}) = n_i \cdot e^{-dist_i(\vec{x}, \vec{y})}
\]

Here, \( n_i \) is the number of times sentence \( s \) appears in the web corpus, and \( dist_i(\vec{x}, \vec{y}) \) is the distance between typed-terms \( \vec{x} \) and \( \vec{y} \) (i.e., number of typed-terms in-between) in that sentence. \( e^{-dist_i(\vec{x}, \vec{y})} \) is used to penalize distant co-occurrence. We then aggregate frequencies among sentences, and weigh each edge by a modified tf-idf formula.

\[
f(\vec{x}, \vec{y}) = \sum_i f_i(\vec{x}, \vec{y})
\]

\[
w(\vec{x}, \vec{y}) = \frac{f(\vec{x}, \vec{y})}{\sum_z f(\vec{x}, z)} \cdot \log \frac{N}{N_{nei}(\vec{y})}
\]

Here, \( f(\vec{x}, \vec{y}) \) estimates the probability that humans think of typed-term \( \vec{y} \) when seeing \( \vec{x} \). \( N \) is the total number of typed-terms contained in the co-occurrence network, and \( N_{nei}(\vec{y}) \) is the number of co-occurrence neighbors of \( \vec{y} \). Therefore, the idf part of this formula penalizes typed-terms that co-occur with almost every other typed-term.
There are some obvious drawbacks in the above approach. First, the number of typed-terms is extremely large. Recall that Probase contributes 2.7 million concepts and 16 million instances to our vocabulary. This will increase storage cost and affect the efficiency of calculation on the network. Second, concept-level co-occurrence is more useful for short text understanding, when semantic coherence is considered. Therefore, we compress the original co-occurrence network by retrieving concepts of each instance from Probase, and then grouping similar concepts together into concept clusters. The nodes in the compressed version of the co-occurrence network are verbs, adjectives, attributes and concept clusters, and the edge weights (i.e., $w(\mathbf{x}, C)$ and $w(C_1, C_2)$) are aggregated from the original network. In Fig. 3, “lyrics” co-occurs with concept song as well as instances “April in Paris” and “hotel California” in the original co-occurrence network. Whereas in the compressed version, it only co-occurs with concept cluster song. In this way, we reduce the size of the co-occurrence network to a large extent. We use the compressed network in the remaining of this work to estimate semantic coherence.

**Fig. 3.** An example of compressed co-occurrence network.

### 4.1.2 Scoring Semantic Coherence

We define **Affinity Score (AS)** to measure semantic coherence between typed-terms. In this work, we consider two types of coherence: similarity and relatedness (co-occurrence). We believe that two typed-terms are coherent if they are semantically similar or they often co-occur on the web. Therefore, the Affinity Score between typed-terms $\mathbf{x}$ and $\mathbf{y}$ can be calculated as follows:

$$S(\mathbf{x}, \mathbf{y}) = \max(S_{\text{sim}}(\mathbf{x}, \mathbf{y}), S_{\text{co}}(\mathbf{x}, \mathbf{y}))$$  \hspace{1cm} (5)

In Eq. 5, $S_{\text{sim}}(\mathbf{x}, \mathbf{y})$ is the semantic similarity between typed-terms $\mathbf{x}$ and $\mathbf{y}$, which can be calculated directly as the cosine similarity between their concept cluster vectors.

$$S_{\text{sim}}(\mathbf{x}, \mathbf{y}) = \cosine(\mathbf{x}, \mathbf{C}_{\mathbf{x}}, \mathbf{y}, \mathbf{C}_{\mathbf{y}})$$  \hspace{1cm} (6)

$S_{\text{co}}(\mathbf{x}, \mathbf{y})$ measures semantic relatedness between typed-terms $\mathbf{x}$ and $\mathbf{y}$. We denote the co-occur concept cluster vector of typed-term $\mathbf{x}$ as $\mathbf{C}_{\text{co}(x)}$ which can be retrieved from the compressed co-occurrence network, and the concept cluster vector of typed-term $\mathbf{y}$ as $\mathbf{y}, \mathbf{C}$. We observe that the larger the overlapping between these two concept cluster vectors, the stronger the relatedness between typed-terms $\mathbf{x}$ and $\mathbf{y}$. Therefore, we calculate $S_{\text{co}}(\mathbf{x}, \mathbf{y})$ as follows:

$$S_{\text{co}}(\mathbf{x}, \mathbf{y}) = \cosine(\mathbf{C}_{\text{co}(x)}, \mathbf{y}, \mathbf{C}_{\mathbf{y}})$$  \hspace{1cm} (7)

### 4.1.3 Indexing Vocabulary for Approximate Term Extraction

Approximate term extraction aims to locate substrings in a text which are similar to terms contained in a predefined vocabulary. To quantify the similarity between two strings, many similarity functions have been proposed including token-based similarity functions (e.g., jaccard coefficient) and character-based similarity functions (e.g., edit distance). Due to the prevalence of misspellings in short texts, we use edit distance as our similarity function to facilitate approximate term extraction.

There have been some recent studies on approximate term extraction. We adopt and extend the trie-based method [37] in this work, considering its smaller index size compared with other approaches such as NGPP [38] and Faerie [39], as well as its efficiency for large edit distance threshold. Specifically, given an edit distance threshold $\tau$, we divide each term into $\tau + 1$ segments evenly. The pigeonhole principle guarantees that if a substring is similar to a term with respect to $\tau$, it must contain at least one segment of that term. We build a segment-based inverted index on the entire vocabulary, where the entries are segments and each segment is associated with an inverted list of terms containing the segment. Given a short text, we adopt the search-extension algorithm proposed in [37] to find all possible terms. In other words, we first enumerate every substring of a short text and check whether it matches a segment using the trie structure. In this way, we obtain a set of segments contained in the short text. Then for each segment and the corresponding substring, we extend the substring to a longer substring similar to a term in the inverted list.

The most notable limitation of the existing trie-based framework is that it utilizes one specific edit distance threshold $\tau$. However, our vocabulary contains a large amount of abbreviations as well as multi-word instances which require different edit distance thresholds. For example, in order to recognize misspelled multi-word instances, we sometimes need a large edit distance threshold of at least 2. But when we apply the same edit distance threshold to abbreviations, it will lead to mistakes (e.g., “nyc” and “ntu” will be regarded as similar). To this end, we extend the trie-based framework to allow for various edit distance thresholds at the same time. The problem is how to determine the value of $\tau$ for different terms. It can be expected that $\tau$ depends on the length of terms. In other words, the longer a term is, the more possible it will be misspelled and the more mistakes there will be. Therefore, we collect a large-scale short text dataset from search engines and microblogging sites, and invite colleagues to label misspelled terms along with their edit distances. We observe a near step-like distribution between edit distance and term length, which is then used as our guideline for determining edit distance threshold for different terms.

### 4.1.4 Determining Instance Ambiguity

The focus of concept labeling is instance disambiguation, making it important to determine whether an instance is ambiguous or not. Handling unambiguous instances is a waste of time and will cause over-filtering sometimes. A straightforward approach to determine instance ambiguity is to check the number of concepts (or concept clusters) it belongs to. However, the accuracy of such a method is highly dependent on the granularity of concept space in a knowledgebase. A coarse-grained knowledgebase will miss some ambiguous instances, while a fine-grained knowledgebase might lead to false positive (i.e., an unambiguous instance is incorrectly recognized as ambiguous). In this work, we adopt a fine-grained knowledgebase, Probase, with a huge coverage of 2.7 million concepts and 5000 concept clusters. We introduce a method to reduce false positive in determining instance ambiguity by analyzing various correlations between concept clusters. One thing to note is that instance ambiguity can be influenced by spatial-temporal features. For example, “apple” was not an ambiguous instance
before the year 1976 when the Apple Company was founded. As another example, “river bank” is ambiguous only in Northern California where it can refer to both the sloped side of a river and a regional bank. Our current work focus on understanding general short texts without any spatial-temporal information, but we leave it as a future work.

![Fig. 4. Examples of senses as hierarchies of concept clusters.](image)

Ambiguity is a subjective concept. We conduct a user study to learn how humans determine instance ambiguity. We provide annotators with a set of instances together with their top-10 concept clusters, and ask them to label whether these instances are ambiguous or not. According to the user study results, we obtain three useful findings: 1) All annotators regard instances such as “dog” as unambiguous although they belong to multiple concept clusters. These concept clusters (e.g., predator, animal, creature, etc.) actually constitute a hierarchy which we denote as a Sense in this work, as shown in Fig. 4. 2) Some of the annotators label instances such as “googles” as unambiguous, although they belong to multiple senses. These senses (e.g., search engine and company) are actually highly related with each other since they have a large proportion of common instances. 3) All annotators label instances such as “apple” as ambiguous because they belong to multiple unrelated senses (e.g., fruit and company). Based on these findings, we introduce three levels of instance ambiguity, and propose methods to determine ambiguity level by analyzing the hierarchical and overlapping relationships between concept clusters.

- Ambiguity level 0 refers to instances that most people regard as unambiguous. These instances contain only one sense, such as “dog” (animal) and “california” (state);
- Ambiguity level 1 refers to instances that both ambiguous and unambiguous make sense. These instances usually contain more than one senses, but all of these senses are related to some extent, such as “googles” (company & search engine) and “nike” (brand & company);
- Ambiguity level 2 refers to instances that most people think as ambiguous. These instances contain two or more unrelated senses, such as “apple” (fruit & company) and “jaguar” (animal & company).

In this work, we only focus on disambiguation of instances that belong to ambiguity level 2.

Fig. 5 shows an example of our approach to determining instance ambiguity. More specifically, given an instance, we first recognize senses by constructing hierarchies among its concept clusters. We notice that concept cluster A can be treated as a child of concept cluster B if: 1) most of the instances contained in A also belong to B; or 2) most of the popular instances contained in A also belong to B. Probase currently has a triangle problem, that is the transitivity fails sometimes or the link between terms $t_a$ and $t_c$ is missing although $t_a$ is an instance of $t_b$ which is in turn an instance of $t_c$. Nevertheless, we can still assume that the concept clusters constructing a hierarchy should share most of the popular instances though the total number of common instances is small. In order to formulate the above two situations, we define a Hierarchy Score to represent the probability of concept cluster A as a child of concept cluster B.

$$
\alpha \ast \frac{|E(A) \cap E(B)|}{|E(A)|} + (1 - \alpha) \ast \frac{\sum_{e \in E(A) \cap E(B)} p(e|A)}{\sum_{e \in E(A)} p(e|A)}
$$

(8)

Here, $\frac{|E(A) \cap E(B)|}{|E(A)|}$ is used to model the first situation, where $E(A) = \{e|\exists c, e \in c \land c \in A\}$ is the set of instances contained in concept cluster A and $|E(A)|$ is the size of this instance set. $\sum_{e \in E(A) \cap E(B)} p(e|A)$ is used to model the second situation, where $p(e|A)$ is the probability of instance e belonging to concept cluster A which can be aggregated from the typicality score $p(e|c)$ stored in Probase. After calculating the hierarchy score between each pair of concept clusters of an instance, we employ the Graph Cut algorithm to cluster concept clusters with high hierarchy scores together to form a sense. As depicted in the example of Fig. 5, the set of senses derived from the instance “puma” is \{cat,species,company,brand,shoe,product\}.

In real world, many senses are similar or related or they share a large amount of instances, such as company and brand, brand and product, meat and animal, etc. We cluster these senses together using the Graph Cut algorithm. Here, the similarity or relatedness between two senses $s_a$ and $s_b$ is defined as the maximum similarity or relatedness between their concept clusters, which is in turn defined as the weighted proportion of common instances. More formally,

$$
sim(s_a, s_b) = \max_{C_i \in e_a \land C_j \in e_b} \cosine(W(E(C_i)), W(E(C_j)))
$$

(9)

where $W(E(C)) = (w(e_1|C), w(e_2|C), ..., w(e_n|C))$ is the weight vector of the instances in concept cluster C, and each $w(e|C)$ is aggregated from the typicality score $p(e|c)$ stored in Probase. In Fig. 5, we recognize senses {company,brand} and {shoe,product} to be related and cluster them together.

![Ambiguity Level = 2](image)

Finally, we determine the ambiguity level of an instance based on the result of sense detection and sense clustering as follows:

$$
level = \begin{cases} 
0 & |S| = 1 \\
1 & |S| > 1 \land |S| \leq |C| \\
2 & |S| > 1 
\end{cases}
$$

(10)

where $|S|$ is the number of senses after sense detection and $|S| \leq |C|$ is the number of sense clusters after sense clustering.
4.2 Online Processing

There are basically three tasks in online processing of short texts, namely text segmentation, type detection, and concept labeling.

4.2.1 Text Segmentation

We can recognize all possible terms from a short text using the trie-based framework described in Sec. 4.1.3. But the real question is how to obtain a coherent segmentation from the set of terms. We use two examples in Fig. 6 to illustrate our approach of text segmentation. Obviously, “april in paris lyrics” is a better segmentation of “april in paris lyrics” than [april paris lyrics], since “lyrics” is more semantically related to songs than to months or cities. Similarly, “vacation april in paris” is a better segmentation of “vacation april in paris”, due to higher coherence among “vacation”, “april”, and “paris” than that between “vacation” and “april in paris”.

We use a graph to represent candidate terms and their relationships. In this work, we define two types of relations among candidate terms:

• Mutual Exclusion - Candidate terms that contain a same word are mutually exclusive. For example, “april in paris” and “april” in Fig. 6 are mutually exclusive, because they cannot co-exist in the final segmentation;

• Mutual Reinforcement - Candidate terms that are related mutually reinforce each other. For example, in Fig. 6, “april in paris” and “lyrics” reinforce each other because they are semantically related.

Based on these two types of relations, we construct a term graph (TG, as shown in Fig. 6) where each node is a candidate term. We associate each node with a weight representing its coverage of words in the short text excluding stop words. We add an edge between two candidate terms when they are not mutually exclusive, and set the edge weight to reflect the strength of mutual reinforcement as follows:

\[ w(x, y) = \max_{i,j} (\epsilon, \max_{i,j} S(\tilde{x}_i, \tilde{y}_j)). \]  

(11)

where \( \epsilon > 0 \) is a small positive weight, \([\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_m] \) is the set of typed-terms for term \( x \), \([\tilde{y}_1, \tilde{y}_2, ..., \tilde{y}_n] \) is the set of typed-terms for term \( y \), and \( S(\tilde{x}, \tilde{y}) \) is the affinity score between typed-terms \( \tilde{x} \) and \( \tilde{y} \) defined in Eq. 5. Since a term may potentially map to multiple typed-terms, we define the edge weight between two candidate terms as the maximum affinity score between their corresponding typed-terms. When two terms are not related, the edge weight is set to be slightly larger than 0 (to guarantee the feasibility of a Monte Carlo algorithm).

Now, the problem of finding the best segmentation is transformed into the problem of finding a sub-graph in the original TG such that the sub-graph

• is a complete graph (clique) - The selected terms are not mutually exclusive;

• has 100% word coverage excluding stop words;

• has the largest average edge weight - We choose average edge weight rather than total edge weight as the measure of a sub-graph, since the latter usually prefers shorter terms (i.e., more nodes and edges in the sub-graph), which is contradictory with the intuition of the widely-used Longest Cover algorithm.

Given that an edge exists between each pair of nodes as long as the corresponding terms are not mutually exclusive, we can arrive at the following theorem:

**Theorem 1.** Finding a clique with 100% word coverage is equivalent to retrieving a maximal clique from the TG.

**Proof.** If the retrieved clique \( G' \) is not a maximal clique of the original TG, then we can find another node \( v \) such that after inserting \( v \) and the corresponding edges into \( G' \), the resulting sub-graph is still a clique. Due to the special structure of TG, \( v \) is not mutually exclusive with any other node in \( G' \). In other words, they do not cover the same word. Therefore, adding \( v \) into \( G' \) will increase the total word coverage to be larger than 100%, which is obviously impossible.

Now we need to find a maximal clique with the largest average edge weight from the original TG. However, this is NP-hard. The naive Brute Force algorithm enumerates every possible subset of nodes, checks whether the resulting sub-graph is a maximal clique, calculates its average edge weight, and finally finds the one with the largest weight. Therefor, the time complexity of the Brute Force algorithm is \( O(2^n \cdot n^2) \) where \( n \) is the number of terms. Though \( n \) is not too large in the case of short texts, the Brute Force algorithm is still too time-consuming to support instant handling. In this work, we propose a randomized algorithm to obtain an approximate solution more efficiently, as described in Algorithm 1 and Algorithm 2.

**Algorithm 1 Maximal Clique by Monte Carlo (MaxCMC)**

**Input:**

\( G = (V, E); W(E) = \{w(e) | e \in E\} \)

**Output:**

\( G' = (V', E'); s(G') \)

1: \( V' = \emptyset; E' = \emptyset \)
2: while \( E \neq \emptyset \) do
3: randomly select \( e = (u, v) \) from \( E \) with probability proportional to its weight
4: \( V' = V' \cup \{u, v\}; E' = E' \cup \{e\} \)
5: \( V = V - \{u, v\}; E = E - \{e\} \)
6: for each \( t \in V \) do
7: if \( e' = (u, t) \notin E \) or \( e' = (v, t) \notin E \) then
8: \( V = V - \{t\} \)
9: remove edges linked to \( t \) from \( E; E = E - \{e' = (t, \ast)\} \)
10: end if
11: end for
12: end while
13: calculate average edge weight: \( s(G') = \frac{\sum w(e)}{|E'|} \)
Algorithm 2 Chunking by Maximal Clique (CMaxC)

Input:
$G = (V,E); W(E) = \{w(e)|e \in E\}$
number of times to run Algorithm 1: $k$

Output:
$G'_{best} = (V'_i,E'_i)$

1: $s_{max} = 0$
2: for $i = 1; i \leq k; i + +$ do
3: run Algorithm 1 with $|G'_i| = (V'_i,E'_i), s(G'_i)$ as output
4: if $s(G'_i) > s_{max}$ then
5: $G'_{best} = G'_i; s_{max} = s(G'_i)$
6: end if
7: end for

Algorithm 1 runs as follows: First, it randomly selects an edge $e = (u,v)$ with probability proportional to its weight. In other words, the larger the edge weight, the higher the probability to be selected. After picking an edge, it removes all nodes that are disconnected (namely mutually exclusive) with the picked nodes $u$ or $v$. At the same time, it removes all edges that are linked to the deleted nodes. This process is repeated until no edges can be selected. The obtained sub-graph $G'$ is obviously a maximal clique of the original TG. Finally, it evaluates $G'$ and assigns it with a score representing the average edge weight. In order to improve the accuracy of the above algorithm, we repeat it for $k$ times, and choose the maximal clique with the highest score as the final segmentation.

In Algorithm 1, the while loop will be repeated for at most $n_e$ times, since each time the algorithm removes at least one edge from the original TG. Here, $n_e$ is the total number of edges in TG. Similarly, the for loop in each while loop will be repeated for at most $n_v$ times. Therefore, the total time complexity of this randomized algorithm is $O(k \cdot n_e \cdot n_v)$ or $O(k \cdot n_v^2)$. Our experimental results in Sec. 5 verify the effectiveness and efficiency of this randomized algorithm.

4.2.2 Type Detection

Recall that we can obtain the collection of typed-terms for a term directly from the vocabulary. For example, term “watch” appears in instance-list, concept-list, as well as verb-list of our vocabulary, thus the possible typed-terms of “watch” are \{watch\_c, watch\_t, watch\_v\}. Analogously, the collections of possible typed-terms for “free” and “movie” are \{free\_adj, free\_v\} and \{movie\_c, movie\_t\} respectively, as illustrated in Fig. 7. For each term derived from a short text, type detection determines the best typed-term from the set of possible typed-terms. In the case of “watch free movie”, the best typed-terms for “watch”, “free”, and “movie” are watch\_c, free\_adj, and movie\_t respectively.

The Chain Model

Recall that traditional approaches to POS tagging consider lexical features only. Most of them adopt Markov Model [23] [24] [25] [26] [27] [28] [29] which learns lexical probabilities ($P(\text{tag}|\text{word})$) as well as sequential probabilities ($P(\text{tag}|\text{tag}_{-1},...,\text{tag}_{-n})$) from a labeled corpus of sentences, and tags a new sentence by searching for tag sequence that maximizes the combination of lexical and sequential probabilities. However, such surface features are insufficient to determine types of terms in the case of short texts. As we have discussed in Challenge 3, “pink” in “pink songs” will be mistakenly recognized as an adjective using traditional POS taggers, since both the probability of “pink” as an adjective and that of an adjective preceding a noun are relatively high. Whereas, “pink” is actually a famous singer and thus should be labeled as an instance, considering the fact that the concept song is much more semantically related to the concept singer than the color-describing adjective “pink”. Furthermore, the sequential feature ($P(\text{tag}|\text{tag}_{-1},...,\text{tag}_{-n})$) fails in short texts. In other words, the type of a term does not necessarily depend on types of preceding terms only. Therefore, better approaches should be invented to improve the accuracy of type detection.

Our intuition is that although lexical features are insufficient to determine types of terms derived from a short text, errors can be reduced substantially by taking into consideration semantic relations with surrounding context. We believe that the preferred result of type detection is a sequence of typed-terms where each typed-term has a high prior score obtained by considering traditional lexical features, and typed-terms in a short text are semantically coherent with each other.

More formally, we define Singleton Score (SS) to measure the correctness of a typed-term considering lexical features. To simplify implementation, we calculate singleton scores directly based on the results of traditional POS taggers. Specifically, we first obtain the POS tagging result of a short text using an open source POS tagger - Stanford Tagger\textsuperscript{2}. Then we assign singleton scores to terms by comparing their types and POS tags. Specifically, terms whose types are consistent with their POS tags will get a slightly larger singleton score than those whose types are different from their POS tags. Since traditional POS tagging methods cannot distinguish among attributes, concepts, and instances, we treat all of them as nouns. This guarantees types and POS tags to be comparable.

$$S_{sg}(\bar{x}) = \begin{cases} 1 + \theta & \bar{x}.r = \text{pos}(\bar{x}) \\ 1 & \text{otherwise} \end{cases}$$

(12)

In Eq. 12, $\bar{x}.r$ and $\text{pos}(\bar{x})$ are the type and POS tag of typed-term $\bar{x}$ respectively.

Based on singleton score which represents lexical features of typed-terms and affinity score which models semantic coherence between typed-terms, we formulate the problem of type detection into a graph model - the Chain Model. Fig. 7 (a) illustrates an example of the Chain Model.

We borrow the idea of first order bilexical grammar, and consider topical coherence between adjacent typed-terms, namely the preceding and the following one. In particular, we build a chain-like graph where nodes are typed-terms retrieved from the original short text, edges are added between each pair of typed-terms mapped from adjacent terms, and the edge weight between typed-terms $\bar{x}$ and $\bar{y}$ is calculated by multiplying the affinity score with the corresponding singleton scores.

$$w(\bar{x}, \bar{y}) = S_{sg}(\bar{x}) \cdot S(\bar{x}, \bar{y}) \cdot S_{sg}(\bar{y})$$

(13)

Here, $S_{sg}(\bar{x})$ is the singleton score of typed-term $\bar{x}$ defined in Eq. 12, and $S(\bar{x}, \bar{y})$ is the affinity score between typed-terms $\bar{x}$ and $\bar{y}$ defined in Eq. 5.

Now the problem of type detection is transformed into finding the best sequence of typed-terms collectively, which maximizes the total weight of the resulting sub-graph. That is, given a sequence of terms $\{t_1, t_2,..., t_k\}$ derived from the original short text,

text, we need to find a corresponding sequence of typed-terms \( \{t_1, t_2, \ldots, t_l\} \) that maximize:

\[
\sum_{i=1}^{l-1} w(t_i, t_{i+1})
\]

(14)

In the case of “watch free movie”, the best sequence of typed-terms detected using the Chain Model is \( [watch]\_c, free\_adj, movie]\_c \), as illustrated in Fig. 7 (a).

(a) Type detection result of “watch free movie using the Chain Model is \( [watch]\_c, free\_adj, movie]\_c \). (b) Type detection result of “watch free movie using the Pairwise Model is \( [watch]\_c, free\_adj, movie]\_c \).

Fig. 7. Difference between Chain Model and Pairwise Model.

The Pairwise Model

In fact, terms that are most related in a short text might not always be adjacent. Therefore, if we only consider semantic relations between consecutive terms, like in the Chain Model, it will lead to mistakes. In the case of “watch free movie” in Fig. 7 (a), the Chain Model incorrectly recognizes “watch” to be an instance, since “watch” is an instance of the concept product in our knowledgebase, and the probability of adjective “free” co-occurring with concept product is relatively high. However, when relatedness between “watch” and “movie” is considered, “watch” should be labeled as a verb. The Pairwise Model is able to capture such cross-term relations. More specifically, the Pairwise Model adds edges between typed-terms mapped from each pair of terms rather than adjacent terms only. In Fig. 7 (b), there are edges between nonadjacent terms “watch” and “movie”, in addition to those between “watch” and “free” as well as those between “free” and “movie”.

Like the assumption of Chain Model, the best sequence of typed-terms should be semantically coherent. One thing to note is that although cross-term relations are considered in the Pairwise model, a typed-term is not required to be related with every other typed-term. Instead, we assume that it should be semantically coherent with at least one other typed-term. Therefore, the goal of the Pairwise Model is to find the best sequence of typed-terms which guarantees that the maximum spanning tree (MST) of the resulting sub-graph has the largest weight. In Fig. 7 (b), as long as the total weight of edge between \( watch]\_c \) and \( movie]\_c \) and that between \( free\_adj \) and \( movie]\_c \) is the largest, \( [watch]\_c, free\_adj, movie]\_c \) can be successfully recognized as the best sequence of typed-terms for “watch free movie”, regardless of relations between \( watch]\_c \) and \( free\_adj \).

We employ the Pairwise Model in our prototype system as the approach to type detection. But we present the accuracy of both models in the experiments, in order to verify the superiority of Pairwise Model over Chain Model.

4.2.3 Concept Labeling

The most important task in concept labeling is instance disambiguation, which is the process of eliminating inappropriate semantics behind an ambiguous instance. We accomplish this task by re-ranking concept clusters of the target instance based on context information in a short text (i.e., remaining terms), so that the most appropriate concept clusters are ranked higher and the incorrect ones lower.

Our intuition is that a concept cluster is appropriate for an instance only if it is a common semantics of that instance and it achieves support from surrounding context at the same time. Take “hotel california eagles” as an example. Although both animal and music band are popular semantics of “eagles”, only music band is semantically coherent (i.e., frequently co-occurs) with the concept song and thus can be kept as the final semantics of “eagles”.

We have mentioned before that a term is not necessarily related with every other term in the short text. If irrelevant terms are used to disambiguate a target instance, most of its concept clusters will obtain little support, which will in turn lead to over-filtering. Therefore, we decide to use only the most related term to help with disambiguation. In the Chain Model and Pairwise Model, we have obtained the best sequence of typed-terms together with the weighted edges in-between, hence the most related term can be retrieved straightforwardly by comparing weights of edges connecting to the target instance.

Based on the aforementioned intuition, we model the process of instance disambiguation using a Weighted Vote approach. Assume that the target ambiguous instance is \( \bar{x} \) whose concept cluster vector is \( \bar{x}C = (\langle C_1, W_1 \rangle, \ldots, \langle C_N, W_N \rangle) \), and the most related typed-term used for disambiguation is \( \bar{y} \). Then the importance of each concept cluster in \( \bar{x} \)’s disambiguated concept cluster vector \( \bar{x}C = (\langle C_1, W'_1 \rangle, \ldots, \langle C_N, W'_N \rangle) \) is a combination of self-vote and context-vote. More formally,

\[
\bar{x}W'_i = V_{self}(C_i) \cdot V_{context}(C_i)
\]

(15)

Here, self-vote \( V_{self}(C_i) \) represents the original weight of concept cluster \( C_i \) obtained from Probase, namely \( V_{self}(C_i) = \bar{x}W_i \); context-vote \( V_{context}(C_i) \) represents the probability of \( C_i \) as a co-occurrence neighbor of the context \( \bar{y} \). In other words, self-vote \( V_{self}(C_i) \) can be calculated using Eq. 1, and context-vote \( V_{context}(C_i) \) is the weight of \( C_i \) in \( \bar{y} \)’s co-occurrence concept cluster vector which can be retrieved directly from the compressed co-occurrence network described in Sec. 4.1.1.

In the case of “hotel california eagles”, the original concept cluster vector of “eagles” is \((\langle animal,0.2379\rangle,\langle band,0.1277\rangle,\langle bird,0.1101\rangle,\langle celebrity,0.0463\rangle,\ldots)\) and the co-occurrence concept cluster vector of context term “hotel california” is \((\langle singer,0.0237\rangle,\langle band,0.0181\rangle,\langle celebrity,0.0137\rangle,\langle album,0.0132\rangle,\ldots)\). After disambiguation using Weighted Vote, the final concept cluster vector of “eagles” (after normalization) is \((\langle band,0.4562\rangle,\langle celebrity,0.1583\rangle,\langle animal,0.1317\rangle,\langle singer,0.0911\rangle,\ldots)\).

5 Experiment

We conducted comprehensive experiments on real-world datasets to evaluate the performance of our approach to short text understanding. All the algorithms were implemented in C#, and all the experiments were conducted on a server with 2.90GHz Intel Xeon E5-2690 CPU and 192GB memory.

5.1 Benchmark

One of the most notable contributions in this work is that we build a generalized framework for short text understanding which can recognize best segmentations, conduct type detection, and
eliminate instance ambiguity explicitly based on various types of context information. Therefore, we manually picked 11 ambiguous terms (i.e., “april in paris” and “hotel california” with segmentation ambiguity; “watch”, “book”, “pink”, “blue”, “orange”, “population”, and “birthday” with type ambiguity; “apple” and “fox” with instance ambiguity) and randomly selected 1100 queries containing one of these terms from one day’s querylog (100 queries for each term) to check the performance of our framework for disambiguation. Furthermore, in order to examine the performance of our system on general queries, we randomly sampled another 400 queries without any restriction. Based on these queries, we constructed three testing datasets:

- ambig - queries containing ambiguous terms;
- general - general queries;
- all - all queries in our query dataset;

To verify the generalizability of our framework on other short texts, we randomly sampled 1500 tweets using Twitter’s API. We preprocessed the tweet dataset by removing some tweet-specific features, such as @username, hashtags, urls, etc., and other noise, such as words containing more than three continuous same characters (e.g., “Ooooooooops”). We divided each dataset into 5 disjoint parts, and invited 15 colleagues to label them (3 for each part). Final labels were based on majority vote.

### 5.2 Effectiveness

#### 5.2.1 Effectiveness of Text Segmentation

One prerequisite to text segmentation is to locate candidate terms. In order to cope with the noise in short texts, we construct a large-scale vocabulary which incorporates terms as well as their abbreviations and nicknames. Furthermore, we adopt and extend the trie-based method [37] to allow for approximate term extraction with varying edit distance constraints. We compare the performance of our approach (i.e., Trie with Varying edit distance, TrieV) with the trie-based method (i.e., Trie) and exact matching method (i.e., Exact), in terms of precision, recall, and f1-measure. From Table 2 we can see that approximate term extraction can obtain more terms from short texts than exact matching, at the cost of introducing slightly more extraction errors. By allowing for various edit distance thresholds depending on text length, TrieV improves the precision of Trie by reducing extraction errors caused by short terms, abbreviations, etc. Overall, TrieV achieves the highest f1-measure in both datasets. Note that the performance of all these term extraction methods is consistently better in the query dataset than in the tweet dataset. This is mainly because tweets are usually more informal and noisy than queries.

#### 5.2.2 Effectiveness of Type Detection

In this part, we compare our approaches to type detection (i.e., the Chain Model and Pairwise Model) with a widely-used, non-commercial POS tagger - Stanford Tagger. Since traditional POS taggers do not distinguish among attributes, concepts and instances, we need to address this problem first in order to make a reasonable comparison. We consider two situations here: 1) if the recognized term contains multiple words or its POS tag is *noun*, then we check the frequency of that term as an attribute, a concept and an instance respectively in our knowledgebase, and choose the type with the highest frequency as its label; 2) otherwise, we label the term according to its POS tag.

Table 4 demonstrates the precision of Stanford Tagger (ST), Chain Model (CM) and Pairwise Model (PM) for type detection. We use four kinds of precision to measure the effectiveness of these models:

- lexical-level: the percentage of correct lexical (i.e., verb and adjective) term-type pairs;
- semantic-level: the percentage of correct semantic (i.e., attribute, concept and instance) term-type pairs;
- term-level: the percentage of correct term-type pairs;
- text-level: the percentage of short texts whose term-type pairs are all correct.

The results of lexical-level, semantic-level, and term-level precision in the three query datasets (i.e., “ambig”, “general”, and “all”) illustrate similar trends with text-level precision. Therefore, we only present the differences at text level. For the other three levels of precision, we present results for the “all” query dataset. From Table 4, we can see that the Pairwise Model performs better than the Chain Model on all kinds of precision measures, which in turn provides better precision than the Stanford Tagger in both query dataset and tweet dataset. However, the precision improvement is slightly larger in the query data set than in the tweet dataset. This is mainly because tweets are more grammatically structured than

---

**Table 2**

<table>
<thead>
<tr>
<th>query</th>
<th>precision</th>
<th>recall</th>
<th>f1-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact</td>
<td>0.988</td>
<td>0.847</td>
<td>0.912</td>
</tr>
<tr>
<td>Trie</td>
<td>0.943</td>
<td>0.922</td>
<td>0.932</td>
</tr>
<tr>
<td>TrieV</td>
<td>0.964</td>
<td>0.918</td>
<td>0.950</td>
</tr>
<tr>
<td>Exact</td>
<td>0.980</td>
<td>0.718</td>
<td>0.829</td>
</tr>
<tr>
<td>Trie</td>
<td>0.938</td>
<td>0.847</td>
<td>0.890</td>
</tr>
<tr>
<td>TrieV</td>
<td>0.972</td>
<td>0.833</td>
<td>0.897</td>
</tr>
</tbody>
</table>

**Table 3**

<table>
<thead>
<tr>
<th>query</th>
<th>Longest Cover</th>
<th>MaxCBF</th>
<th>MaxCMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambig</td>
<td>0.905</td>
<td>0.980</td>
<td>0.970</td>
</tr>
<tr>
<td>general</td>
<td>0.937</td>
<td>0.990</td>
<td>0.982</td>
</tr>
<tr>
<td>all</td>
<td>0.954</td>
<td>0.994</td>
<td>0.979</td>
</tr>
<tr>
<td>tweet</td>
<td>0.961</td>
<td>0.980</td>
<td>0.973</td>
</tr>
</tbody>
</table>
keyword queries, making traditional POS tagging more reliable. Interestingly, the lexical-level precision of the Chain Model and the Pairwise Model is also larger than traditional POS taggers. Since the Stanford Tagger only pays attention to lexical features, it will mistakenly recognize “pink” in “pink songs” as an adjective, which is actually an instance of singer. The Chain Model and Pairwise Model, on the contrary, take context semantics into consideration and thus can solve the above problem. The Chain Model has the limitation that it only considers semantic relations between adjacent terms, which makes it incomparable with the Pairwise Model. Note that the precision improvement of the Pairwise Model over Stanford Tagger is much larger for ambiguous queries (12.3% in “ambig” dataset) than general queries (7.5% in “general” dataset), which further verifies the superiority of our approach to reduce ambiguity in type detection.

### Table 4

<table>
<thead>
<tr>
<th>Type</th>
<th>ST</th>
<th>CM</th>
<th>PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>lexical</td>
<td>0.865</td>
<td>0.967</td>
<td>0.978</td>
</tr>
<tr>
<td>semantic</td>
<td>0.944</td>
<td>0.969</td>
<td>0.973</td>
</tr>
<tr>
<td>term</td>
<td>0.932</td>
<td>0.968</td>
<td>0.974</td>
</tr>
<tr>
<td>ambig</td>
<td>0.837</td>
<td>0.945</td>
<td>0.960</td>
</tr>
<tr>
<td>general</td>
<td>0.902</td>
<td>0.962</td>
<td>0.977</td>
</tr>
<tr>
<td>all</td>
<td>0.876</td>
<td>0.955</td>
<td>0.967</td>
</tr>
<tr>
<td>lexical</td>
<td>0.962</td>
<td>0.965</td>
<td>0.984</td>
</tr>
<tr>
<td>semantic</td>
<td>0.950</td>
<td>0.970</td>
<td>0.977</td>
</tr>
<tr>
<td>term</td>
<td>0.944</td>
<td>0.963</td>
<td>0.980</td>
</tr>
<tr>
<td>text</td>
<td>0.891</td>
<td>0.958</td>
<td>0.968</td>
</tr>
</tbody>
</table>

Recall that we employ a singleton score to incorporate the result of traditional POS taggers in the Chain Model and Pairwise Model. We assign singleton score of a typed-term as \(1 + \theta\) when its type is consistent with its POS tag, and 1 otherwise. In other words, the variable \(\theta\) represents the amount of impact lexical features have on type detection results. Fig. 8 depicts the variation of type detection precision on terms and short texts, when \(\theta\) ranges from 0 to 1. We can see that the precision of type detection increases dramatically when context semantics and lexical features are combined to estimate best types (from \(\theta = 0\) to \(\theta = 0.1\)). However, as lexical features play an increasingly important role in the Chain Model and Pairwise Model, the precision decreases slightly (from \(\theta = 0.2\) to \(\theta = 1\)). Most notably, the precision of type detection using Chain Model and Pairwise Model when \(\theta = 0\) (namely only semantic features are considered) is larger than that of the Standford Parser depicted in Table 4. This proves that context semantics are more important than lexical features for determining types of terms in short texts.

### Table 5

<table>
<thead>
<tr>
<th>Type</th>
<th>[16]</th>
<th>[17]</th>
<th>[15]</th>
<th>Our</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>term</td>
<td>0.694</td>
<td>0.701</td>
<td>-</td>
<td>0.943</td>
</tr>
<tr>
<td>text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ambig</td>
<td>0.498</td>
<td>0.503</td>
<td>-</td>
<td>0.884</td>
</tr>
<tr>
<td>general</td>
<td>0.356</td>
<td>0.356</td>
<td>-</td>
<td>0.909</td>
</tr>
<tr>
<td>all</td>
<td>0.325</td>
<td>0.326</td>
<td>-</td>
<td>0.890</td>
</tr>
<tr>
<td>tweet</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>term</td>
<td>-</td>
<td>-</td>
<td>0.841</td>
<td>0.922</td>
</tr>
<tr>
<td>text</td>
<td>-</td>
<td>-</td>
<td>0.774</td>
<td>0.894</td>
</tr>
</tbody>
</table>

5.2.3 Effectiveness of Short Text Understanding

One of the most notable contributions of our work is that we propose a generalized framework that can recognize best segmentations, conduct type detection, and eliminate instance ambiguity based on various types of context information. Therefore, we examine the effectiveness of short text understanding as a whole in this part. More specifically, we compare the performance of our framework with current state-of-the-art approaches to mining semantics from short texts, namely [16] which conducts instance disambiguation in queries based on similar instances, [17] which conducts instance disambiguation in queries based on related instances, and [15] which conducts instance disambiguation in tweets based on both similar and related instances. As before, we consider both term-level and text-level precisions to measure the effectiveness of short text understanding. We present the results in the three query datasets (i.e., “ambig”, “general”, and “all”) only at text level. For term-level precision, we present results achieved in the “all” query dataset. From Table 5 we can see that our approach performs dramatically better than current state-of-the-art approaches, since it can utilize various context information to conduct instance disambiguation. Note that the precision improvement of our framework over existing state-of-the-art methods is only slightly larger for ambiguous queries (38.1% in “ambig” dataset) than general queries (35.3% in “general” dataset), since many general queries also contain ambiguous instances. We observe 329 out of 517 instances (63.6%) derived from the “general” query dataset are ambiguous. Another interesting finding is that our framework improves precision to a larger extent in the query dataset (36.4%) than in the tweet dataset (12%). This can be explained from two aspects: first, tweets are usually longer than keyword queries, providing more context information for instance disambiguation; second, [15] disambiguates instances based on both similar instances and related instances. These two features contribute to more accurate semantic interpretations of short texts using [15] than those obtained using [16] or [17], which in turn results in smaller precision improvement in the tweet dataset than in the query dataset. Overall, our framework achieves not only high but also comparable precision of short text understanding in both datasets (89.0% in query dataset and 89.4% in tweet dataset), which verifies its effectiveness and robustness.

(a) Chain Model. (b) Pairwise Model.

Fig. 8. Precision of type detection when \(\theta\) increases.

In Sec. 4.1.4, we described how to determine whether an instance is ambiguous or not based on its semantics (i.e., concept clusters). Specifically, we group concept clusters into senses (i.e., hierarchies of concept clusters), cluster similar or related senses into sense clusters, and then determine instance ambiguity level according to the number of sense clusters, as illustrated in Fig. 5. To examine the performance of this method, we randomly sampled 900 instances (300 from each ambiguity level), and invited colleagues to label their ambiguity with three options: “ambiguous”, “unambiguous”, and “hard to say”. We observe
that our algorithm agrees with human annotators for most of the instances (82.2% agreement), and the disagreement mainly happens in instances from ambiguity level 1 (41% disagreement). In Equation 8, we combine two conditions that can indicate the hierarchical relationship between two concept clusters, namely the percentage of common instances and common popular instances, using a parameter $\alpha$. We demonstrate how the choice of $\alpha$ influences the precision of ambiguity detection in Fig. 9 (a). We can see that the percentage of common popular instances (i.e., $\alpha = 0$) contributes more to estimating hierarchical relationship than the percentage of common instances (i.e., $\alpha = 1$). This is mainly due to the triangle problem in Probase. We determine instance ambiguity in this work for two reasons. First, it is obvious that the time cost of short text understanding can be reduced by skipping unambiguous instances when conducting instance disambiguation. Second, disambiguating unambiguous instances will cause over-filtering sometimes, which in turn affects the accuracy of short text understanding, as verified in Fig. 9 (b).

![Figure 9. Effectiveness of instance ambiguity detection.](image)

5.3 Efficiency

As we know, short text understanding is usually regarded as an online task or an underlying step of many other text mining applications like classification and clustering. These applications usually need to handle millions of short texts at a time, which makes the efficiency of short text understanding extremely critical. Therefore, we examine the time requirement of our framework to verify its efficiency, as depicted in Fig. 10 (a). We can see that our framework can efficiently interpret a short text within hundreds of milliseconds, and the time requirement increases linearly with the ascending of text length. Note that the time required to handle a tweet is mostly shorter than that required to handle a query of the same length. One possible explanation is that a large proportion of words in tweets are stopwords which need little processing, while words in queries are mostly keywords. In Sec. 4.2.1, we described a randomized algorithm (MaxCMC) to reduce the time cost of the Brute Force algorithm (MaxCBF) for text segmentation. We depict the average time requirement of these two methods for text segmentation in Fig. 10 (b). The average time requirement of text segmentation is consistently larger in tweets than in queries, since tweets contain more words than queries in average.

6 Conclusion

In this work, we propose a generalized framework to understand short texts effectively and efficiently. More specifically, we divide the task of short text understanding into three subtasks: text segmentation, type detection, and concept labeling. We formulate text segmentation as a weighted Maximal Clique problem, and propose a randomized approximation algorithm to maintain

accuracy and improve efficiency at the same time. We introduce a Chain Model and a Pairwise Model which combine lexical and semantic features to conduct type detection. They achieve better accuracy than traditional POS taggers on the labeled benchmark. We employ a Weighted Vote algorithm to determine the most appropriate semantics for an instance when ambiguity is detected. The experimental results demonstrate that our proposed framework outperforms existing state-of-the-art approaches in the field of short text understanding. As a future work, we attempt to analyze and incorporate the impact of spatial-temporal features into our framework for short text understanding.

Acknowledgments

This work was partially supported by the National 863 High-tech Program under Grant No. 2012AA011001, the National Key Basic Research Program (973 Program) of China under Grant No. 2014CB340403, and the National Natural Science Foundation of China under Grant No. M13210007.

References


Wen Hua is currently a Postdoctoral Research Fellow at the University of Queensland. She received her bachelor and PhD degrees in computer science from Renmin University of China in 2010 and 2015 respectively. Her research interests include natural language processing, information retrieval and extraction, data mining, social media analysis, and graph management.

Zhongyuan Wang is currently a Researcher at Microsoft Research Asia. He received his bachelor, master, and PhD degrees in computer science from Renmin University of China in 2007, 2010, and 2015 respectively. His research interests include knowledge base, web data mining, online advertising, machine learning, and natural language processing.

Haixun Wang received the bachelor and master degrees in computer science from Shanghai Jiao Tong University in 1994 and 1996 respectively, and the PhD degree in computer science from the University of California, Los Angeles in 2000. He joined Google Research, Mountain View in 2013. His research interests include text analytics, natural language processing, knowledge base, semantic network, artificial intelligence, database language and system, graph data management, etc. He is a member of IEEE.

Kai Zheng is currently an ARC DECPRA (Australia Research Council Discovery Early Career Researcher Award) Fellow at the University of Queensland. He received his PhD degree in computer science from the University of Queensland in 2012. His research focus is to find effective and efficient solutions for managing, integrating and analyzing big data for business, scientific and personal applications. He has been working in the area of social-media analysis, spatial-temporal databases, uncertain databases, data mining and bioinformatics. He has also been appointed as adjunct professor in the School of Computer Science at Soochow University since 2015. He is a member of IEEE.

Xiaofang Zhou received the bachelor and master degrees in computer science from Nanjing University in 1984 and 1987 respectively, and the PhD degree in computer science from the University of Queensland in 1994. He is a professor of computer science with the University of Queensland. He is the head of the Data and Knowledge Engineering Research Division, School of Information Technology and Electrical Engineering. He is also an specially appointed adjunct professor at Soochow University, China. His research is focused on finding effective and efficient solutions to managing integrating, and analyzing very large amounts of complex data for business and scientific applications. His research interests include spatial and multimedia databases, high performance query processing, web information systems, data mining, and data quality management. He is a senior member of IEEE.