Using Relational Similarity between Word Pairs for Latent Relational Search on the Web

Nguyen Tuan Duc, Danushka Bollegala, Mitsuru Ishizuka
The University of Tokyo
{*duc, danushka*}*@mi.ci.i.u-tokyo.ac.jp, ishizuka@i.u-tokyo.ac.jp*

***Abstract*—Latent relational search is a new search paradigm based on the degree of analogy between two word pairs. A latent relational search engine is expected to return the word *Paris* as an answer to the question mark (?) in the query {(Japan, Tokyo), (France, ?)} because the relation between *Japan* and *Tokyo* is highly similar to that between *France* and *Paris*. We propose an approach for exploring and indexing word pairs to efficiently retrieve candidate answers for a latent relational search query. Representing relations between two words in a word pair by lexical patterns allows our search engine to achieve a high MRR and high precision for the top 1 ranked result. When evaluating with a Web corpus, the proposed method achieves an MRR of 0.963 and it retrieves correct answer in the top 1 for 95.0% of queries.**

***Keywords*-latent relational search, relational similarity, analogical search**

I. Introduction

Latent relational search is a new search paradigm inspired by previous research on analogy [1] and relational similarity [2]. Latent relational search can be used for mapping knowledge from a well-known domain to an unknown domain. For example, an Apple user can search for information about a music player by Microsoft using the query {*(Apple, iPod), (Microsoft, ?)*}, for which the answer is *Zune* [3]. Therefore, latent relational search can be effectively used when a user does not know the keywords to search for (e.g., the keyword *Zune* in the above example).

We propose an approach to extract word pairs from a text corpus (e.g., the Web) and represent relations between words using lexical patterns to build an index for a latent relational search engine. Following the previous work on relational similarity measurement between two word pairs [2], [4], [5], we rank the search results according to their relational similarities with the query. The proposed method for representing the relation between the two words in a word pair enables our search engine to achieve both a high precision and recall.

The remainder of this paper is organized as follows. In the next section, we describe related work on latent relational search and relational similarity measurement. We then give an overview of the search engine in Section III. We describe the proposed algorithm for word pair extraction and relation representation to build the index in Section IV. We describe the proposed method for ranking candidate answers using a relational similarity measure in Section V. In Section VI, we present our experimental results. Finally, we conclude the paper in Section VII.

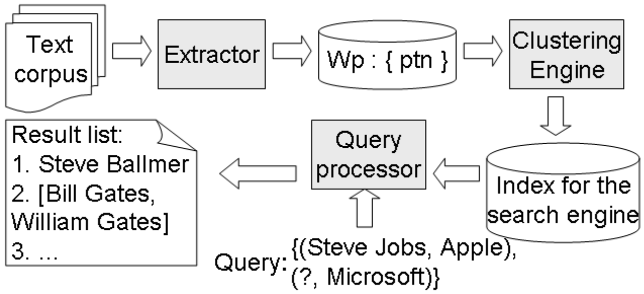
Text corpus
Extractor
Wp : { ptn }
Clustering Engine
Index for the search engine
Query processor
Result list:
1. Steve Ballmer
2. [Bill Gates, William Gates]
3. ...
Query: {(Steve Jobs, Apple), (?, Microsoft)}

Figure 1. Overview of the latent relational search engine

II. Related work

Research on measuring relational similarity between two word pairs such as [2], [4], [5] suggests a method for ranking candidate word pairs in latent relational search. Relation of a pair of words is represented by lexical patterns, i.e., the context where the word pair appeared. Using lexical pattern frequencies as feature vector for a word pair, these studies achieve a high precision on the task of measuring semantic similarity between two word pairs. However, these studies assume that the two word pairs to measure relational similarity {(A, B), (C, D)} are given so we can retrieve lexical patterns of each pair.

One implementation of latent relational search is described by Kato et al. in [3]. This method represents the relations between two words in a given word pair by using the bag-of-words model. It does not requires a local index for searching (it uses an existing keyword-based Web search engine to find the answer). However, it does not achieve a high precision because representing relation with the bag-of-words model does not allow the relational similarity between two word pairs to be precisely measured. To achieve a high precision, the relational similarity between (A, B) and (C, D) should be measured using a well-defined method such as [2], [5], in which the relation between C and D is represented by lexical patterns that are in the same sentence with the pair (C, D).

III. Overview of the latent relational search engine

The entire latent relational search system is illustrated in Fig. 1. The input for building the index is a text corpus (such as a set of crawled web pages). These text documents are fed to the Extractor, which extracts word pairs and lexical

978-0-7695-4191-4/10 $26.00 © 2010 IEEE
DOI 10.1109/WI-IAT.2010.167

patterns that represent relation between each pair. For example, from the sentence “Steve Jobs is the CEO of Apple”, the Extractor would extract the pair (*Steve Jobs*, *Apple*) and lexical patterns that represent the relation between “Steve Jobs” and “Apple” such as “X is the CEO of Y”, “X * CEO * Y”, …(X and Y are variables that represent two entities in the relation, the wildcard “*” represents zero or more words). Next, the extracted word pairs and lexical patterns are input to the Clustering Engine which groups semantically similar lexical patterns (such as “X is the CEO of Y” and “Y’s CEO is X”) into a same pattern cluster and groups different surface forms of an entity into a same word cluster (e.g., “Steve Jobs”, “Steven Jobs”). When a query such as {(Steve Jobs, Apple), (?, Microsoft)} is input, the Query Processor finds a candidate answer set and ranks the answers as described in the following sections. An answer may be a single word (“Steve Ballmer”) or a cluster of words (“[Bill Gates, William Gates, W. H. Gates]”).

IV. Word pair extraction and relation representation

A. Extracting word pairs

To extract named entities of interest, we use a Named Entity Recognizer (NER) to tag the input sentences[1]. From a tagged sentence, we extract all named entity pairs that maintain the order of two entities in the original sentence. Note that, we do not know the relation types in advance. We blindly extract all pairs that might hold some relations and then use several filters to obtain informative pairs as describe later. We can also index all types of word pairs by using a POS-tagger a long with a Named Entity Recognizer.

B. Representing semantic relation by lexical patterns

Following previous work [2], [4], [5], we also use lexical patterns to represent semantic relation between two words of a word pair that appeared in the same sentence. The relation between two words in a word pair is therefore represented by a vector of lexical pattern frequencies. To extract lexical patterns that might represent semantic relation between two words C and D in a sentence S, we consider the following sub-string of S:

$$b_1b_2...b_kCw_1w_2...w_mDa_1a_2...a_p$$

That is, the sub-string contains of k words before C, the word C, the gap between C and D, the word D and p words after D. To eliminate the differences between inflected forms of a word, we use a word stemmer[2] for stemming all words in the above sub-sequence. We then generate all n-grams ($n \leq 7$) from the above sub-sequence. We omit all n-grams which contain only b_i or contain only a_i (e.g., $b_1b_2b_3$ or $a_3a_4a_5$). For n-grams which contain only w_i (i.e, $w_iw_{i+1}...w_j$), we change them into the form “$X * w_iw_{i+1}...w_j * Y$”. Similarly, for n-grams which do not contain Y (e.g, $b_kXw_1w_2$), we change them into $b_kXw_1w_2{*}Y$. And finally, for n-grams which do not contain X, we append “$X*$” before them: $X * w_iw_{i+1}...w_mYa_1$. Only n-grams with at least one content word (i.e., not stop word and not the variable X or Y) are selected.

Comparing to pattern extraction algorithms in previous research [2], [4], [5], the proposed algorithm is adapted to improve the recall of relational search. First, we eliminate the differences between inflected forms of a word by stemming the input sequence. Second, we allow sub-sequences that neither contain X nor Y. This makes the probability that two entity pairs have common lexical patterns higher because we do not need a complete match between the sequences in the gap of each pair. For example, consider the two sentences: “Obama is the 44th and current president of the U.S” and “Sarkozy is the current president of France”. If we allow the pattern “current president of” (i.e., we generate the pattern “X * current president of * Y”) then we have a common pattern between two pairs *(Obama, U.S)* and *(Sarkozy, France)*.

We denote $\mathbf{P}(wp)$ as the set of all lexical patterns with which the word pair wp appeared:

$$\mathbf{P}(wp) = \{p_1, p_2, \ldots, p_n\} \tag{1}$$

We denote $\mathbf{W}(p)$ as the set of all word pairs with which the pattern p appeared:

$$\mathbf{W}(p) = \{wp_1, wp_2, \ldots, wp_m\} \tag{2}$$

We denote the frequency of co-occurrence of the word pair wp_i with the pattern p_j in a same sentence as $\mathrm{f}(wp_i, p_j)$. The word pair frequency vector $\mathbf{\Phi}(p)$ of a lexical pattern p is then defined as:

$$\mathbf{\Phi}(p) = (\mathrm{f}(wp_1, p), \mathrm{f}(wp_2, p), \ldots, \mathrm{f}(wp_m, p))^{\mathrm{T}} \tag{3}$$

Similarly, the pattern frequency vector of a word pair wp is defined as:

$$\mathbf{\Psi}(wp) = (\mathrm{f}(wp, p_1), \mathrm{f}(wp, p_2), \ldots, \mathrm{f}(wp, p_n))^{\mathrm{T}} \tag{4}$$

C. Pattern clustering

Even when lexical patterns are stemmed, two similar word pairs often share only a small number of identical lexical patterns because a relation can be expressed in several ways in natural language (e.g., “X acquired Y” and “X bought Y” are semantically similar). To recognize semantically similar patterns p and q, we first define a similarity measure between the two lexical patterns using their word pair frequency vectors $\mathbf{\Phi}(p)$ and $\mathbf{\Phi}(q)$. Then we use a sequential pattern clustering algorithm as described in [2] to group similar patterns into a same cluster. For each pattern, the algorithm finds the cluster whose centroid has maximum cosine similarity with the pattern. If this similarity is above a pattern clustering similarity threshold θ then the pattern is added to the cluster, otherwise, the pattern forms a new

[1] We use the Stanford Named Entity Recognizer available at http://nlp.stanford.edu/software/CRF-NER.shtml

[2] We use the Porter Stemmer in the NLTK toolkit: http://nltk.googlecode.com/svn/trunk/doc/api/nltk.stem.porter-module.html

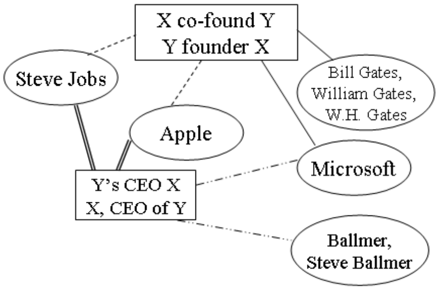

Figure 2. The index for the search engine. Each entity cluster is represented by an ellipse, each lexical pattern cluster is represented by a rectangle.

singleton cluster itself. To filter out patterns that are specific to a word pair and to reduce the time for pattern clustering, we execute the pattern clustering algorithm only for patterns which appeared above 10 times in the database.

We also use the same clustering algorithm for clustering entities. Therefore, we group semantically similar entities into a same entity cluster.

After pattern clustering and entity clustering steps, we can build an index that contains information about entity clusters, lexical pattern clusters and relations between these entity clusters as shown in Fig. 2. For example, in Fig. 2, *Steve Jobs* is linked to *Apple* by two lexical pattern clusters: “X co-found Y” and “X, CEO of Y”.

V. Ranking search results using relational similarity

A. Retrieving candidate answers

A candidate answer pair for the query {(A, B), (C, ?)} is the pair $c = (C, X)$ that appears above five times and (C, X) has at least one common pattern which appears above ten times with the input pair. Therefore, the candidate answer set $\Re$ is:

$$\Re = \bigcup_{p \in \mathbf{P}(s) \wedge \mathrm{freq}(p) \geq 10} \{wp \in \mathbf{W}(p) | (wp[0] = C) \wedge \mathrm{freq}(wp) \geq 5\} \quad (5)$$

B. Ranking the result set

To filter out inappropriate answers and to rank the result list, we calculate the relational similarity between two word pairs s ($s = (A, B)$) and c ($c = (C, X)$) using their lexical pattern frequency vectors $\mathbf{\Psi}(s)$ and $\mathbf{\Psi}(c)$. We define the relational similarity $\mathrm{relsim(s,c)}$ between s and c using a modified version of cosine similarity of their pattern frequency vectors by considering two patterns that are in a same cluster as identical.

The good candidate list Γ is obtained by filtering out all candidates with relational similarity smaller than a similarity threshold σ:

$$\Gamma = \{c \in \Re | \mathrm{relsim}(s, c) \geq \sigma\} \quad (6)$$

Moreover, for the input query {(A, B), (C, ?)}, we repeat the candidate retrieval process for the reversed query {(B, A), (?, C)}. If $s' = (B, A)$ and $c' = (X, C)$ then the score of a candidate c for the input pair s is defined as

$$\chi(s, c) = \mathrm{relsim}(s, c) + \frac{1}{2}\mathrm{relsim}(s', c') \quad (7)$$

We set the weight of the relational similarity of the reversed word pair ($\mathrm{relsim}(s', c')$) to $1/2$ because we prefer that the candidate appears in the original query rather than the reversed query.

Finally, we use the information about entity clusters to merge all candidates which are semantically identical (or similar) into a candidate cluster. The candidate cluster set Γ' is therefore $\Gamma' = \mathrm{Merged}(\Gamma)$. The score (for ranking) of a candidate cluster $K = \{c_1, c_2, \ldots, c_k\}$ is defined as

$$\mathrm{score}(s, K) = \frac{1}{k}\sum_{i=1}^{k} \chi(s, c_i) \quad (8)$$

The final result list is obtained by sorting Γ' in order of the clusters' scores from high to low.

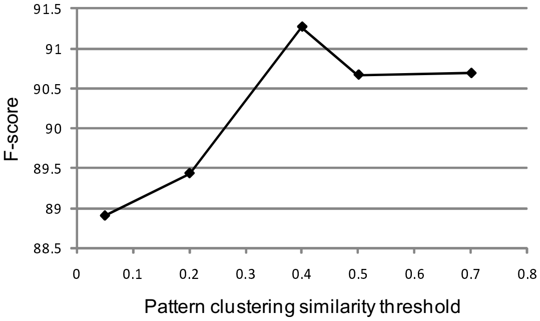

Figure 3. Average F-score of three relation categories (Person - Birthplace, Company - Headquarters, CEO - Company) while varying the pattern clustering similarity threshold θ (at σ = 0.05)

VI. Evaluation

A. Parameter tuning

To determine the appropriate value of the pattern clustering similarity threshold θ, we evaluate our system using 12000 documents containing four types of relations: Person - Birthplace (e.g., Einstein - Germany), Company - Headquarters (Microsoft - Redmond) , CEO - Company (Steve Jobs - Apple) and Acquirer - Acquiree (Google - Youtube). From those text documents, the system extracted 113742 word pairs and 2069121 lexical patterns. To avoid noisy word pairs and lexical patterns, our system considers only word pairs with frequency above five for searching and patterns with frequency above ten for pattern clustering. Consequently, the system only considers 4103 word pairs for searching and 27568 patterns for clustering.

We vary the lexical pattern clustering similarity threshold θ and evaluate the system using four query sets which correspond to four relation types above. We measure the precision, recall and F-score at each value of θ.

For queries with single correct result, such as $\{(\mathrm{Person}_i, \mathrm{Birthplace}_i), (\mathrm{Person}_j, ?)\}$ $(i \neq j)$, we

Table I

Performance of the search engine for each relation category (at $\theta = 0.4, \sigma = 0.05$)

Data set	Precision	Recall	F-score
Birthplace	98.89	98.89	98.89
Headquarters	90.59	85.56	88.00
CEO-comp.	95.56	95.56	95.56
Acquirer - Acquiree	81.34	-	-
Average	91.60	93.34	94.15

only evaluate the top 1 ranked result. For Acquirer - Acquiree relation, we evaluate the system with queries of type $\{(\mathrm{Acquirer}_i, \mathrm{Acquiree}_i), (\mathrm{Acquirer}_j, ?)\}$ $(i \neq j)$. Note that these queries have multiple correct answers (correct answers are in the set of companies that were acquired by $\mathrm{Acquirer}_j$). Recall for this query set can not be calculated because we only evaluate the top 10 answers of each query.

Fig. 3 shows the average F-score of three test sets that we can calculate recall (Person - Birthplace, Company - Headquarters and CEO - Company). When θ is 0.4, we obtained the maximum value of the average F-score. The shape of the graph does not change when we vary the value of the similarity threshold σ in the range $[0.03, 0.2]$. However, with $\sigma = 0.05$, we obtain the best average F-score (when σ is above 0.2, the recall is very small and the F-score drastically decreases).

B. Average precision, recall and F-score

We fix the parameter θ and σ at the values that gave the best performance in the parameter tuning phase (we set θ to 0.4 and σ to 0.05) and use a completely different corpus (containing 6000 documents referring to the same four relation types but with different instances) to evaluate the performance of the proposed system. We use the new corpus because we want to verify the parameter tuning process and prevent the bias to the corpus that is used for parameter tunning.

Table I shows the average performance of the proposed method for each type of queries. For queries with single correct answer (ranked at top 1) we achieve high precision and recall. For queries with multiple answers, we also achieve precision of 81.34% for the top 10 ranked results.

C. Comparison with previous method

It is difficult to make a fair comparison between the proposed method with the method by Kato et al. [3] because of the differences between the languages that are used in the experiments (English vs. Japanese) and the relation categories. However, we still gain some meaningful information from the comparison because there are some common relation types between our data and the data described in [3] (for example, the CEO-Company, Acquisition relation, ...).

Table II shows the comparison between the performance of the proposed method with the method of Kato et al. [3]. The data for comparison are the average performance of our system and the results that are described in [3] (the comparison uses only single correct answer queries). The MRR of the proposed method is 76.7% better than of the previous method (0.963 compared to 0.545). Moreover, the proposed method also outperforms the previous method in the percentage of queries with correct answer in the top 1.

Table II

Comparison between the proposed method with the previous method(@N is the percentage of queries where the correct answer is in the Top N results).

Method	MRR	@1	@5	@10	@20
Kato et al. [3]	0.545	43.3	68.3	72.3	76.0
Proposed method	0.963	95.0	97.8	97.8	97.8

We obtain a high performance because the proposed lexical pattern extraction algorithm works well. Word pairs with similar semantic relation might have slightly different lexical patterns in the gaps between two words in the pairs (e.g., "*Barrack Obama* is the 44th and current president of the *U.S*" and "*Nicolas Sarkozy* is the current president of *France*"). In this situation, the method in [3] gives a very small (if not 0) relational similarity because it considers only the exact lexical pattern in the gap between two words. On the other hand, the proposed lexical pattern extraction algorithm gives a high relational similarity for these pairs because it generates many patterns that are matched in two pairs (e.g., *X* * *president* * *Y*, *X* * *president of* * *Y*, *X* * *current president* * *Y*, *X* * *current president of* * *Y*).

The proposed system requires less than 10 seconds for processing a query. This query processing time can not be achieved in the previous work [3] because it needs to query a keyword-based Web search engine many times.

VII. Conclusion

We have presented an approach for extracting and representing relations between two entities for latent relational search. The proposed relation representation method enables our search engine to achieve a high performance because it works well even when the lexical patterns between word pairs are not identical. When evaluating with a Web corpus, the proposed method achieves a high precision and MRR while requiring a small query processing time.

References

[1] T. Veale, "The analogical thesaurus," in *Proc. of the Innovative Applications of Artificial Intelligence*. AAAI Press, 2003, pp. 137–142.

[2] D. Bollegala, Y. Matsuo, and M. Ishizuka, "Measuring the similarity between implicit semantic relations from the web," in *Proc. of WWW'09*. ACM, 2009, pp. 651–660.

[3] M. P. Kato, H. Ohshima, S. Oyama, and K. Tanaka, "Query by analogical example: relational search using web search engine indices," in *Proc. of CIKM'09*, 2009, pp. 27–36.

[4] D. Bollegala, Y. Matsuo, and M. Ishizuka, "Measuring the similarity between implicit semantic relations using web search engines," in *Proc. of WSDM'09*. ACM, 2009, pp. 104–113.

[5] P. D. Turney, "Similarity of semantic relations," *Computational Linguistics*, vol. 32, no. 3, pp. 379–416, 2006.