Fast and Large-scale Unsupervised Relation Extraction

Sho Takase[†] **Naoaki Okazaki**[†‡] **Kentaro Inui**[†]
Graduate School of Information Sciences, Tohoku University[†]
Japan Science and Technology Agency (JST)[‡]
{takase, okazaki, inui}@ecei.tohoku.ac.jp

Abstract

A common approach to unsupervised relation extraction builds clusters of patterns expressing the same relation. In order to obtain clusters of relational patterns of good quality, we have two major challenges: the semantic representation of relational patterns and the scalability to large data. In this paper, we explore various methods for modeling the meaning of a pattern and for computing the similarity of patterns mined from huge data. In order to achieve this goal, we apply algorithms for approximate frequency counting and efficient dimension reduction to unsupervised relation extraction. The experimental results show that approximate frequency counting and dimension reduction not only speeds up similarity computation but also improves the quality of pattern vectors.

1 Introduction

Semantic relations between entities are essential for many NLP applications such as question answering, textual inference and information extraction (Ravichandran and Hovy, 2002; Szpektor et al., 2004). Therefore, it is important to build a comprehensive knowledge base consisting of instances of semantic relations (e.g., authorOf) such as *authorOf* ⟨*Franz Kafka, The Metamorphosis*⟩. To recognize these instances in a corpus, we need to obtain patterns (e.g., "X write Y") that signal instances of the semantic relations.

For a long time, many researches have targeted at extracting instances and patterns of specific relations (Riloff, 1996; Pantel and Pennacchiotti, 2006; De Saeger et al., 2009). In recent years, to acquire a wider range knowledge, Open Information Extraction (Open IE) has received much attention (Banko et al., 2007). Open IE identifies relational patterns and instances automatically without predefined target relations (Banko et al., 2007; Wu and Weld, 2010; Fader et al., 2011; Mausam et al., 2012). In other words, Open IE acquires knowledge to handle open domains. In Open IE paradigm, it is necessary to enumerate semantic relations in open domains and to learn mappings between surface patterns and semantic relations. This task is called unsupervised relation extraction (Hasegawa et al., 2004; Shinyama and Sekine, 2006; Rosenfeld and Feldman, 2007).

A common approach to unsupervised relation extraction builds clusters of patterns that represent the same relation (Hasegawa et al., 2004; Shinyama and Sekine, 2006; Yao et al., 2011; Min et al., 2012; Rosenfeld and Feldman, 2007; Nakashole et al., 2012). In brief, each cluster includes patterns corresponding to a semantic relation. For example, consider three patterns, "X write Y", "X is author of Y" and "X is located in Y". When we group these patterns into clusters representing the same relation, patterns "X write Y" and "X is author of Y" form a cluster representing the relation *authorOf*, and the pattern "X is located in Y" does a cluster for *locatedIn*. In order to obtain these clusters, we need to know the similarity between patterns. The better we model the similarity of patterns, the better a clustering result correspond to semantic relations. Thus, the similarity computation between patterns is crucial for unsupervised relation extraction.

29th Pacific Asia Conference on Language, Information and Computation pages 96 - 105
Shanghai, China, October 30 - November 1, 2015

We have two major challenges in computing the similarity of patterns. First, it is not clear how to represent the semantic meaning of a relational pattern. Previous studies define a feature space for patterns, and express the meaning of patterns by using such as the co-occurrence statistics between a pattern and an entity pair, e.g., co-occurrence frequency and pointwise mutual information (PMI) (Lin and Pantel, 2001). Some studies employed vector representations of a fixed dimension, e.g., Principal Component Analysis (PCA) (Collins et al., 2002) and Latent Dirichlet Allocation (LDA) (Yao et al., 2011; Riedel et al., 2013). However, the previous work did not compare the effectiveness of these representations when applied to a collection of large-scaled unstructured texts.

Second, we need design a method scalable to a large data. In Open IE, we utilize a large amount of data in order to improve the quality of unsupervised relation extraction. For this reason, we cannot use a complex and inefficient algorithm that consumes the computation time and memory storage. In this paper, we explore methods for computing pattern similarity of good quality that are scalable to huge data, for example, with several billion sentences. In order to achieve this goal, we utilize approximate frequency counting and dimension reduction. Our contributions are threefold.

- We build a system for unsupervised relation extraction that is practical and scalable to large data.
- Even though the proposed system introduces approximations, we demonstrate that the system exhibits the performance comparable to the one without approximations.
- Comparing several representations of pattern vectors, we discuss a reasonable design for representing the meaning of a pattern.

2 Methods

2.1 Overview

As mentioned in Section 1, semantic representations of relational patterns is key to unsupervised relation extraction. Based on the distributional hypothesis (Harris, 1954), we model the meaning of a relational pattern with a distribution of entity pairs co-occurring with the pattern. For example, the meaning of a relational pattern "X write Y" is represented by the distribution of the entity pairs that fills the variables (X, Y) in a corpus. By using vector representations of relational patterns, we can compute the semantic similarity of two relational patterns; for example, we can infer that the patterns "X write Y" and "X is author of Y" present the similar meaning if the distribution of entity pairs for the pattern "X write Y" is similar to that for the pattern "X is author of Y".

Researchers have explored various approaches to vector representations of relational patterns (Lin and Pantel, 2001; Rosenfeld and Feldman, 2007; Yao et al., 2011; Riedel et al., 2013). The simplest approach is to define a vector of a relational pattern in which an element in the vector presents the co-occurrence frequency between a pattern and an entity pair. However, the use of raw frequency counts may be inappropriate when some entity pairs co-occur with a number of patterns. A solution to this problem is to use a refined co-occurrence measure such as PMI (Lin and Pantel, 2001). In addition, we may compress a vector representation with a dimensionality reduction method such as PCA because pattern vectors tend to be sparse and high dimensional (Yao et al., 2011; Riedel et al., 2013).

Meanwhile, it may be difficult to implement the above procedures that can handle a large amount of data. Consider the situation where we find 1.1 million entity pairs and 0.7 million relational patterns from a corpus with 15 billion sentences. Even though the vector space is sparse, we need to keep a huge number of frequency counts that record co-occurrences of the entity pairs and relational patterns in the corpus.

In order to acquire pattern vectors from a large amount of data, we explore two approaches in this study. One approach is to apply an algorithm for approximate counting so that we can discard unimportant information in preparing pattern vectors. Another approach is to utilize distributed representations of words so that we can work on a semantic space of a fixed and compact size. In short, the former approach reduces the memory usage for computing statistics, whereas the latter compresses the vector space beforehand.

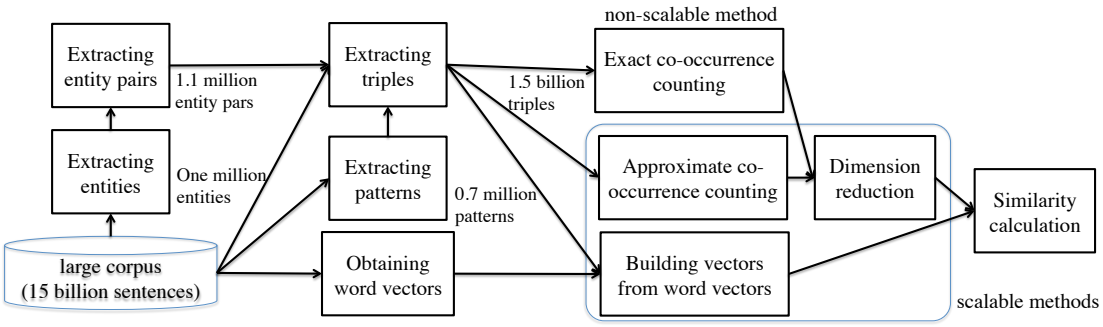

Figure 1: Overview of the system for unsupervised relation extraction

Figure 1 illustrates the overview of the system of unsupervised relation extraction presented in this paper. We extract a collection of triples each of which consists of an entity pair and a relational pattern (Section 2.2). Because this step may extract meaningless triples, we identify entity pairs and relational patterns occurring frequently in the corpus. We compute co-occurrence statistics of entity pairs and relational patterns to obtain pattern vectors. Section 2.3 describes this process, followed by an online variant of PCA in Section 2.4. Furthermore, we present two approaches that improve the scalability to large data in Section 2.5.

2.2 Extracting triples

In this study, we define a triple as a combination of an entity pair and a relational pattern that connects the two entities. In order to extract meaningful triples from a corpus, we mine a set of entities and relational patterns in an unsupervised fashion.

2.2.1 Extracting entities

We define an entity mention as a sequence of nouns. Because quite entity mentions consist of two or more nouns (e.g., "Roadside station" and "Franz Kafka"), we adapt a simple statistical method (Mikolov et al., 2013) to recognize noun phrases. Equation 1 computes the score of a noun bigram $w_i w_j$,

$$\mathrm{score}(w_i, w_j) = \mathrm{cor}(w_i, w_j) * \mathrm{dis}(w_i, w_j), \quad (1)$$

$$\mathrm{cor}(w_i, w_j) = \log \frac{f(w_i, w_j) - \delta}{f(w_i) \times f(w_j)}, \quad (2)$$

$$\mathrm{dis}(w_i, w_j) = \frac{f(w_i, w_j)}{f(w_i, w_j)+1} \frac{\min\{f(w_i), f(w_j)\}}{\min\{f(w_i), f(w_j)\}+1}. \quad (3)$$

Here, $f(w_i)$ denotes the frequency of the noun w_i, and $f(w_i, w_j)$ does the frequency of the noun bigram $w_i w_j$. The parameter δ is a constant value to remove infrequent noun sequences. Consequently, $\mathrm{cor}(w_i, w_j)$ represents the degree of the connection between w_i and w_j. However, $\mathrm{cor}(w_i, w_j)$ becomes undesirably large if either $f(w_i)$ or $f(w_j)$ is small. We introduce the function $\mathrm{dis}(w_i, w_j)$ to 'discount' such sequences.

We form noun phrases whose scores are greater than a threshold. In order to obtain noun phrases longer than two words, we run the procedure four times, decreasing the threshold value[1]. In this way, we can find, for example, "Franz Kafka" as an entity in the first run and "Franz Kafka works" in the second run. After identifying a set of noun phrases, we count the frequency of the noun phrases in the corpus, and extract noun phrases occurring no less than 1,000 times as a set of entities.

2.2.2 Extracting entity pairs

After determining a set of entities, we discover entity pairs that may have semantic relationships in order to locate relational patterns. In this study, we extract a pair of entities if the entities co-occur in more than 5,000 sentences. We denote the set of entity pairs extracted by this procedure E.

2.2.3 Extracting patterns

As a relational pattern, this study employs the shortest path between two entities in a dependency tree, following the previous work (Wu and Weld, 2010; Mausam et al., 2012; Akbik et al., 2012). Here, we introduce a restriction that a relational pattern must include a predicate in order to reject semantically-

[1]The threshold values are 10 (first time), 5 (second time), and 0 (third and fourth times).

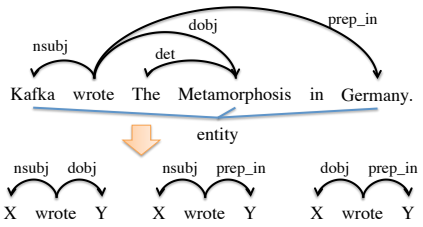

Figure 2: Example of parsed sentence and extracting patterns

ambiguous patterns such as "X of Y". Additionally, we convert an entity into a variable (i.e., X or Y). Consider the sentence shown in Figure 2 as an example. An arrow between words expresses a dependency relationship. This sentence contains three entities: "Kafka", "The Metamorphosis" and "German". Therefore, we obtain three patterns, "X $\xleftarrow{nsubj}$ wrote $\xrightarrow{dobj}$ Y", "X $\xleftarrow{nsubj}$ wrote $\xrightarrow{prep\_in}$ Y" and "X $\xleftarrow{dobj}$ wrote $\xrightarrow{prep\_in}$ Y"[2]. Counting the frequency of a pattern, we extract one appearing no less than 1,500 times in the corpus. We denote the set of relation patterns P hereafter.

2.3 Building pattern vectors

We define a vector of a relational pattern as the distribution of entity pairs co-occurring with the pattern. Processing the whole collection of the corpus, we extract mentions of triples $(p, e), p \in P, e \in E$. For example, we obtain a triple,

$$\begin{aligned} p = &\quad \text{X} \xleftarrow{nsubj} \text{wrote} \xrightarrow{dobj} \text{Y}, \\ e = &\quad \langle \text{"Franz Kafka"}, \text{"The Metamorphosis"} \rangle, \end{aligned}$$

from the sentence "Franz Kafka wrote The Metamorphosis." The meaning of the pattern p is represented by the distribution of the entity pairs co-occurring with the pattern.

In this study, we compare two statistical measures of co-occurrence: the raw frequency (FREQ) and PMI (PMI). In FREQ setting, a relational pattern p is represented by a vector whose elements present the frequency of co-occurrences $f(p, e)$ of every entity pair $e \in E$. PMI refines the strength of co-occurrences with this equation,

$$\mathrm{PMI}(p, e) = \log \frac{\frac{f(p,e)}{M}}{\frac{\Sigma_{i \in P} f(i,e)}{M} \frac{\Sigma_{j \in E} f(p,j)}{M}} \times \mathrm{dis}(p, e). \quad (4)$$

Here, $f(p, e)$ presents the frequency of co-occurrences between a pattern p and an entity pair e; and $M = \sum_i^P \sum_j^E f(i, j)$. The discount factor $\mathrm{dis}(p, e)$ is defined similarly to Equation 3. In PMI setting, we set zero to the value for an entity pair e if $\mathrm{PMI}(p, e) < 0$.

2.4 Dimensionality reduction for pattern vectors

The vector space defined in Section 2.3 is extremely high dimensional and sparse, encoding all entity pairs as separate dimensions. The space may be too sparse to represent the semantic meaning of relational patterns; for example, entity pairs ("Franz Kafka", "the Metamorphosis") and ("Kafka", "the Metamorphosis") present two different dimension even though "Franz Kafka" and "Kafka" refer to the same person. In addition, we need an associative array to compute the similarity of two sparse vectors.

In order to map the sparse and high dimensional space into a dense and compact space, we use *Principal Component Analysis* (PCA). In essence, PCA is a statistical procedure that finds principal components and scores of a matrix. PCA is closely related to *Singular Value Decomposition* (SVD), which factors an $m \times n$ matrix $\mathbf{A}$ with,

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^t. \quad (5)$$

Here, $\mathbf{U}$ is an $m \times m$ orthogonal matrix, $\mathbf{V}$ is an $n \times n$ orthogonal matrix and $\mathbf{\Sigma}$ is an $m \times n$ diagonal matrix storing singular values. Each column of $\mathbf{V}$ corresponds to a principal component, and each column of $\mathbf{U}\mathbf{\Sigma}$ corresponds to a score of a principal component of $\mathbf{A}$.

However, a full SVD requires heavy computations while we only need principal components corresponding to the top r singular values of $\mathbf{A}$. This hinders the scalability of the system, which obtains a huge co-occurrence matrix between patterns and entity pairs. We solve this issue by using the randomized algorithm proposed by Halko et al. (2011).

[2] In the experiments, we use a collection of Japanese Web pages. However, we explain the procedure with an English sentence because the procedure for extracting patterns is universal to other languages.

Algorithm 1 Space saving for each pattern

Input: N: counter size for each pattern
Input: D: a set of triples (p, e)
Output: $c_{p,e}$: counter for each pattern p

```
1:  for all (p, e) ∈ D do
2:      if T_p does not exist then
3:          T_p ← ∅
4:      end if
5:      if e ∈ T_p then
6:          c_{p,e} ← c_{p,e} + 1
7:      else if |T_p| < N then
8:          T_p ← T ∪ {e}
9:          c_{p,e} ← 1
10:     else
11:         i ← argmin_{i∈T_p} c_{p,i}
12:         c_{p,e} ← c_{p,i} + 1
13:         T_p ← T ∪ {e} \ {i}
14:     end if
15: end for
```

The goal of this algorithm is to find an $r \times n$ matrix $\mathbf{B}$ storing the compressed information of the rows of $\mathbf{A}$. We first draw an $n \times r$ Gaussian random matrix $\mathbf{\Omega}$. Next, we derive the $m \times r$ matrix $\mathbf{Y} = \mathbf{A\Omega}$. We next construct an $m \times r$ matrix $\mathbf{Q}$ whose columns form an orthonormal basis for the range of $\mathbf{Y}$. Here, $\mathbf{QQ}^t\mathbf{A} \approx \mathbf{A}$ is satisfied. Finally, we obtain the matrix $\mathbf{B} = \mathbf{Q}^t\mathbf{A}$, in which $\mathbf{Q}^t$ compresses the rows of $\mathbf{A}$.

We compute principal component scores for r dimensions by applying SVD to $\mathbf{B}$. The computation is easy because $r \ll m$. In this study, we used redsvd[3], an implementation of Halko et al. (2011). We represents the meaning of a pattern with the scores of r principle components.

2.5 Improving the scalability to large data

As described previously, it may be difficult and inefficient to count the exact numbers of co-occurrences from a large amount of data. In this study, we explore two approaches: approximate counting (Section 2.5.1) and distributed representations of words (Section 2.5.2).

2.5.1 Approximate counting

We may probably not need exact counts of co-occurrences for representing pattern vectors because a small amount of elements in a pattern vector greatly influence the similarity computation. In other words, it may be enough to find top-k entity pairs with larger counts of co-occurrences for each pattern, and to ignore other entity pairs with smaller counts. The task of finding top-k frequent items has been studied extensively as *approximate counting algorithms*.

We employ *Space Saving* (Metwally et al., 2005), which is the most efficient algorithm to obtain top-k frequent items. Algorithm 1 outlines the Space Saving algorithm adapted for counting frequencies of co-occurrences. The space saving algorithm maintains at most N counters of co-occurrences for each pattern p.

For each triple (p, e), where p and e present a pattern and an entity pair, respectively, the algorithm checks if the co-occurrence count for the triple (p, e) is available or not. If it is available (Line 5), we increment the counter as usual (Line 6). If it is unavailable but the number of counters kept for the pattern is less than N (Line 7), we initialize the counter with one (Lines 8 and 9). If the count is unavailable and if the pattern has already maintained N counters, the algorithm removes a counter $c_{p,i}$ with the least value, and creates a new counter for the entity pair e with the approximated count $c_{p,i} + 1$ (Lines 11–13).

This algorithm has the nice property that the error of the frequency count of a frequent triple is within a range specified by the number of counters N. In addition, Metwally et al. (2005) describes a data structure for finding the least frequent triple efficiently in Line 11. We can obtain the frequency counts of the top-k frequent triples if we set the number of counters N much larger than k. In this way, we can find the frequency count of the top-k frequent triple $f(p, e)$ approximately, and assume frequency counts of other triples zero.

2.5.2 Building pattern vectors from word vectors

A number of NLP researchers explored approaches to representing the meaning of a word with fixed-length vectors (Bengio et al., 2003; Mikolov et al., 2013). In particular, word2vec[4], an implementation of Mikolov et al. (2013), received much attention in the NLP community.

[3] https://code.google.com/p/redsvd/wiki/English

[4] https://code.google.com/p/word2vec/

Switching our attention to the pattern feature vector, our goal is to express the semantic meaning of a relational pattern with a distribution of entity pairs. Here, we explore the use of low-dimensional word vectors learned by word2vec from the large corpus: the meaning of a pattern is represented by the distribution of entity vectors. Thus, we obtain the vector representation of a relational pattern p,

$$\boldsymbol{p} = \sum_{e \in E} f(p, e) \begin{bmatrix} \boldsymbol{v}_{e_0} \\ \boldsymbol{v}_{e_1} \end{bmatrix}. \quad (6)$$

Here, $\boldsymbol{v}_{e_0}$ denotes the vector for an entity e_0 in the entity pair e, and $\boldsymbol{v}_{e_1}$ does the vector for another entity e_1 in the pair e.

3 Experiments

3.1 Data

For our experimental corpus, we collected 15 billion Japanese sentences by crawling web pages. To remove noise such as spam and non-Japanese sentences, we apply a filter that checks the length of a sentence, determines whether the sentence contains a specific character in Japanese (*hiragana*), and then checks the number of symbols. As a result of filtering, we obtained 6.3 billion sentences. We then parsed these sentences using Cabocha[5], a Japanese dependency parser. For preprocessing, we extracted 1 million entities, 1.1 million entity pairs, and 0.7 million patterns. Finally, we extracted about 1.5 billion triples from the corpus.

We then manually checked some of the pattern pairs extracted from Wikipedia that were also contained within the 6.3 billion sentence corpus. Specifically, we first extracted frequent patterns from some domains in Wikipedia for obtaining the patterns representing a specific relation. We selected patterns referring to an illness, an author, and an architecture as target domains. Next, we gathered patterns sharing many entity pairs because these patterns may represent the same relation. We obtained 527 patterns and 4,531 pattern pairs. Four annotators classified these pairs into the same relation or not. We then randomly sampled 90 pairs and found an average of 0.63 for the Cohen's kappa value for two annotators. We annotated the 4,531 pairs by two or more annotators. Therefore, if two or more annotators labeled a pair with the same relation, we regarded the pair as the same relation. Finally, we acquired 720 pairs expressing the same relation. We applied each method to 4,531 pairs and we identified patterns with higher than threshold. We investigated whether the pair is included in the same relation pairs.

[5] https://code.google.com/p/cabocha/

3.2 Experimental settings

We evaluated the quality of pattern vectors build by the proposed approaches on the similarity calculation. Concretely, we investigated the impact on accuracy and computation time. For the evaluation, we computed the cosine similarity based on the feature vectors obtained by each method. We compared the following methods.

Exact counting (baseline): We counted the co-occurrence frequency between an entity pair and a pattern in triples using a machine with 256GB of memory (EXACT-FREQ). In addition, we calculated PMI defined in equation 4 based on the co-occurrence frequency (EXACT-PMI).

Exact counting + PCA: Using PCA, we converted EXACT-FREQ and EXACT-PMI into the fixed dimensional vector EXACT-FREQ+PCA and EXACT-PMI+PCA. We determined the number of dimensions as 1,024 based on comparisons[6] between 256, 512, 1,024, and 2,048.

Approximate counting: We counted the co-occurrence frequency using approximate counting explained in Section 2.5.1 (APPROX-FREQ). The counter size N was 10,240 and we used the top 5,120 frequent entity pairs as a feature. Moreover, we obtained PMI based on the result of approximate counting (APPROX-PMI).

Approximate counting + PCA: Using PCA, we converted APPROX-FREQ and APPROX-PMI into the fixed dimensional vector APPROX-FREQ+PCA and APPROX-PMI+PCA. Similar to Exact counting + PCA, we selected the dimension of feature vectors as 1,024.

Exact counting + word2vec: We obtained pattern feature vectors using the result of word2vec (EXACT-FREQ+WORD2VEC). Moreover, instead of calculating the feature vector by co-occurrence frequency, we weight the entity vector with PMI

[6] The result of 1,024 dimensional vector is close to the one of 2,048. We selected 1,024 since the smaller the number of feature dimensions are, the faster we calculate similarity.

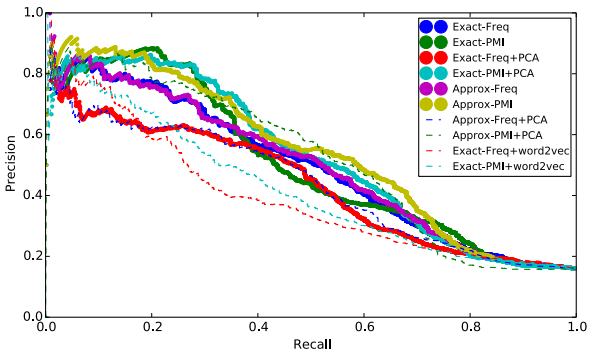

Figure 3: Precision and recall of each method

(EXACT-PMI+WORD2VEC). We trained word2vec using all entities, verbs, and adverbs in the corpus on four AMD Opteron 6174 processors (12-core, 2.2GHz). It took about 130 hours to train word2vec (the number of threads was 42 and window size was 5). Similar to PCA, we selected the dimension of each word vector as 512: namely, the dimension of pattern vectors was 1,024 because of concatenating.

3.3 Evaluating accuracy of each method

Figure 3 shows the precision and recall of each method. We illustrated this graph by changing the threshold value. We focus attention on the difference between exact counting and approximate counting. These results of EXACT-FREQ and APPROX-FREQ were about the same. On the other hand, APPROX-PMI outperformed EXACT-PMI in most areas. These results demonstrated that approximate counting is enough to compute co-occurrence frequency between a pattern and an entity pair. The results also suggest that approximate counting possibly improves the performance.

Comparing the results with PCA and without PCA, the figure shows that PCA does not always improve the performance. However, APPROX-PMI+PCA achieved the best performance in most areas. For computation time, we verify that PCA is important in this aspect.

In contrast, feature vectors based on word2vec worsened the performance against not only approximate counting but also exact counting. Although we expected that the vector formed by word2vec was appropriate for representing the meaning of a pattern, EXACT-FREQ+WORD2VEC was worse than all the other methods in Figure 3. We suspect that this result was caused by separating entity pairs into entities in feature generation. Concretely, for obtaining pattern feature vectors using word2vec, we concatenate the sum of vectors assigned to one side of entity pairs and the ones assigned to the other side. Therefore, there is a possibility that we obtain pattern pairs with a high similarity when both of the patterns contain one of the same entity types. In other words, we need to encode not entities separately but maintaining entity pair as a pattern feature vector.

Method	10k	100k	all (664k)
EXACT-PMI	55m	121hr	8,499hr
APPROX-PMI	38m	110hr	7,441hr
APPROX-PMI+PCA	4m	7hr	785hr

Table 1: Similarity calculation time of each method with one thread

From Figure 3, approximate counting is effective for the similarity calculation. In addition, PCA is useful for representing the meaning of a pattern in the compact space.

3.4 Evaluating computation time

Table 1 demonstrates the similarity calculation time of EXACT-PMI, APPROX-PMI and APPROX-PMI+PCA for processing 10k patterns, 100k patterns, and 664k patterns (the maximum). We executed a program written in C++ on four AMD Opteron 6174 processors (12-core, 2.2GHz) with 256GB of the memory. We measured the calculation time using a single thread. Note that, we split the calculation targets and predicted computation time based on the result of division and split number, because much time is required to complete the calculation for 100k and 664k patterns. For 100k patterns, we split the calculation targets into 48 groups. For 664k pattern, we split the calculation targets into 4,096 groups.

APPROX-PMI executed quicker than EXACT-PMI because APPROX-PMI decreased the number of non-zero features for each pattern. Nevertheless, APPROX-PMI took a large amount of time for the similarity calculation. On the other hand, the computation time of APPROX-PMI+PCA was much smaller than that of EXACT-PMI and APPROX-PMI. As a result, it is necessary to reduce the amount of dimensions because APPROX-PMI

would take 7,441 hours (about a year) to calculate all pattern similarity with one thread. We conclude that it is necessary to prepare low dimensional feature vectors using dimension reduction or word vectors for completing similarity calculation in a realistic time.

4 Related work

Unsupervised relation extraction poses three major challenges: extraction of relation instances, representing the meaning of relational patterns, and efficient similarity computation. A great number of studies proposed methods for extracting relation instances (Wu and Weld, 2010; Fader et al., 2011; Fader et al., 2011; Akbik et al., 2012). We do not describe the detail of these studies, which are out of the scope of this paper.

Previous studies explored various approaches to represent the meaning of relational patterns (Lin and Pantel, 2001; Yao et al., 2012; Mikolov et al., 2013). Lin and Pantel (2001) used co-occurrence statistics of PMI between an entity and a relational pattern. Even though the goal of their research is not on relation extraction but on paraphrase (inference rule) discovery, the work had a great impact to the research on unsupervised relation extraction. Yao et al. (2012) modeled sentence themes and document themes by using LDA, and represented the meaning of a pattern with the themes together with the co-occurrence statistics between patterns and entities. Recently, methods inspired by neural language modeling received much attentions for representation learning (Bengio et al., 2003; Mikolov et al., 2010; Mikolov et al., 2013). In this study, we compared the raw frequency counts, PMI, and word embeddings (Mikolov et al., 2013).

In order to achieve efficient similarity computation, some researchers used entity types, for example, "Franz Kafka" as a co-referent of *writer* (Min et al., 2012; Nakashole et al., 2012). Min et al. (2012) obtained entity types by clustering entities in a corpus. When computing the similarity values of patterns, they restricted target pattern pairs to the ones sharing the same entity types. In this way, they reduced the number of similarity computations. Nakashole et al. (2012) also reduced the number of similarity computations by using entity types obtained from existing knowledge bases such as Yago (Suchanek et al., 2007) and Freebase (Bollacker et al., 2008). However, it is not so straightforward to determine the semantic type of an entity in advance because the semantic type may depends on the context. For example, "Woody Allen" stands for an actor, a movie director, or a writer depending on the context. Therefore, we think it is also important to reduce the computation time for pattern similarities by simplifying the semantic representation of relational patterns.

The closest work to ours is probably Goyal et al. (2012). Their paper proposes to use algorithms of count-min sketch (approximate counting) and approximate nearest neighbor search for NLP tasks. They applied these techniques for obtaining feature vectors, reducing the dimension of the vector space, and searching similar items to a query. Even though they demonstrated the efficiency of the algorithms, they did not demonstrate the effectiveness of the approach in a specific NLP task.

5 Conclusion

In this paper, we presented several approaches to unsupervised relation extraction on a large amount of data. In order to handle large data, we explored three approaches: dimension reduction, approximate counting, and vector representations of words. The experimental results showed that approximate frequency counting and dimension reduction not only speeds up similarity computation but also improved the quality of pattern vectors.

The use of vector representation of words did not show an improvement. This is probably because we need to learn a vector representation specialized for patterns that encode the distributions of entity pairs. A future direction of this research is to establish a method to learn the representations of patterns jointly with the representations of words. Furthermore, it would be interesting to incorporate the meaning of constituent words of a pattern into the representation.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers 26 · 5820, 15H05318 and Japan Science and Technology Agency (JST).

References

Alan Akbik, Larysa Visengeriyeva, Priska Herger, Holmer Hemsen, and Alexander Löser. 2012. Unsupervised discovery of relations and discriminative extraction patterns. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING2012)*, pages 17–32.

Michele Banko, Michael J Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. 2007. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI2007)*, pages 2670–2676.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. 3:1137–1155.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*, pages 1247–1250.

Michael Collins, Sanjoy Dasgupta, and Robert E. Schapire. 2002. A generalization of principal components analysis to the exponential family. In *Advances in Neural Information Processing Systems 14 (NIPS2002)*, pages 617–624.

Stijn De Saeger, Kentaro Torisawa, Jun'ichi Kazama, Kow Kuroda, and Masaki Murata. 2009. Large scale relation acquisition using class dependent patterns. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining (ICDM2009)*, pages 764–769.

Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying relations for open information extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP2011)*, pages 1535–1545.

Amit Goyal, Hal Daumé, III, and Raul Guerra. 2012. Fast large-scale approximate graph construction for nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL2012)*, pages 1069–1080.

Nathan Halko, Per Gunnar Martinsson, and Joel A. Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288.

Zellig Harris. 1954. Distributional structure. *Word*, 10(23):146–162.

Takaaki Hasegawa, Satoshi Sekine, and Ralph Grishman. 2004. Discovering relations among named entities from large corpora. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (ACL2004)*, pages 415–422.

Dekang Lin and Patrick Pantel. 2001. Dirt - discovery of inference rules from text. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2001)*, pages 323–328.

Mausam, Michael Schmitz, Robert Bart, Stephen Soderland, and Oren Etzioni. 2012. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP2012)*, pages 523–534.

Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of the 10th International Conference on Database Theory (ICDT2005)*, pages 398–412.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the Neural Information Processing Systems Conference and Workshops (NIPS2013)*, pages 3111–3119.

Bonan Min, Shuming Shi, Ralph Grishman, and Chin-Yew Lin. 2012. Ensemble semantics for large-scale unsupervised relation extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL2012)*, pages 1027–1037.

Ndapandula Nakashole, Gerhard Weikum, and Fabian Suchanek. 2012. Patty: A taxonomy of relational patterns with semantic types. In *2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL2012)*, pages 1135–1145.

Patrick Pantel and Marco Pennacchiotti. 2006. Espresso: Leveraging generic patterns for automatically harvesting semantic relations. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (ACL2006)*, pages 113–120.

Deepak Ravichandran and Eduard Hovy. 2002. Learning surface text patterns for a question answering system. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL2002)*, pages 41–47.

Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. 2013. Relation extraction with matrix factorization and universal schemas. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL2013)*, pages 74–84.

Ellen Riloff. 1996. Automatically generating extraction patterns from untagged text. In *Proceedings of the 13th National Conference on Artificial Intelligence - Volume 2 (AAAI96)*, pages 1044–1049.

Benjamin Rosenfeld and Ronen Feldman. 2007. Clustering for unsupervised relation identification. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management (CIKM2007)*, pages 411–418.

Yusuke Shinyama and Satoshi Sekine. 2006. Preemptive information extraction using unrestricted relation discovery. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL2006)*, pages 304–311.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web (WWW 2007)*, pages 697–706.

Idan Szpektor, Hristo Tanev, Ido Dagan, and Bonaventura Coppola. 2004. Scaling web-based acquisition of entailment relations. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP2004)*, pages 41–48.

Fei Wu and Daniel S. Weld. 2010. Open information extraction using wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL2010)*, pages 118–127.

Limin Yao, Aria Haghighi, Sebastian Riedel, and Andrew McCallum. 2011. Structured relation discovery using generative models. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP2011)*, pages 1456–1466.

Limin Yao, Sebastian Riedel, and Andrew McCallum. 2012. Unsupervised relation discovery with sense disambiguation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL2012)*, pages 712–720.