Semi-Supervised Convolution Graph Kernels for Relation Extraction

Xia Ning* Yanjun Qi†

Abstract

Extracting semantic relations between entities is an important step towards automatic text understanding. In this paper, we propose a novel Semi-supervised Convolution Graph Kernel (SCGK) method for semantic Relation Extraction (RE) from natural English text. By encoding sentences as dependency graphs of words, SCGK computes kernels (similarities) between sentences using a convolution strategy, i.e., calculating similarities over all possible short single paths on two dependency graphs. Furthermore, SCGK adds three semi-supervised strategies in the kernel calculation to enable soft-matching between (1) words, (2) grammatical dependencies, and (3) entire sentences, respectively. From a large unannotated corpus, these semi-supervision steps learn to capture contextual semantic patterns of elements inside natural sentences, and therefore alleviate the lack of annotated examples in most RE corpora. Through convolutions and multi-level semi-supervisions, SCGK provides a powerful model to encode both syntactic and semantic evidence which are important for effectively recovering the relational patterns of interest. We perform extensive experiments on five RE benchmark datasets which aim to identify interaction relationships from biomedical literature. Our results demonstrate that SCGK achieves the state-of-the-art performance on the task of semantic relation extraction.

keywords: Relation Extraction, Graph Kernels, Semi-supervised Learning, Natural Language Processing

1 Introduction

Natural Language Processing (NLP) aims to understand and organize unstructured text into structured format, which could enable automatic machine translation, semantic information retrieval or advanced question answer, etc. As a basic step towards automatic text understanding, the task of Relation Extraction (RE) tries to detect if a sentence describes a semantic relation between two entities of interest or not. Many methods have been proposed to solve the relation extraction task over years, which primarily differ from two perspectives.

- How to represent sentences and elements inside the sentences, e.g. words [10]. Widely used sentence representations include parse tree [18] and dependency parsing [32, 14]. Popular word representations include Part-of-Speech (POS) tagging, bag-of-words and ontologies [26].
- Which learning method to use [3]. Researchers have tried kernel [25, 1] methods under Support Vector Machine (SVM) [29] framework, generative graphical models [26], neural networks [26] and conditional random fields [3].

Despite years of progress, automatic RE still remains a challenging task due to two reasons. First of all, feature representations of natural sentences is hard for RE problem. Because the task is associated to both the syntactic structures and the semantic patterns of natural text. Secondly, the lack of sufficient annotated examples for model training also limits the capability of current RE systems.

To tackle both challenges, here we propose a novel Semi-supervised Convolution Graph Kernel method, referred to as SCGK, to solve Relation Extraction task as a sentence classification problem using Support Vector Machine classifier. In our method, each sentence is represented as a graph with words as graph vertices and syntactic dependencies between words as corresponding edges. Consequently the dependency graph representation provides a powerful structure to encode grammatical patterns between words. To encode semantic patterns beyond syntax, SCGK proposes three semi-supervised steps to groups similar elements inside text sentences. For instance, the semi-supervision on words provides an embedded representation for each word in the dictionary which was learnt to capture contextual semantic similarities between words from a large unannotated corpus. Finally a convolution kernel strategy is proposed to calculate the similarities (i.e. kernels under SVM framework) between sentences using not only the dependency graph structures, but also the semi-supervised semantic represenations of text elements in the sentences. Essentially the proposed convolution strategy calculates similarities over all possible short single paths from two dependency graphs. This is partly motivated by the fact that semantic relations between name entities are mostly localizing to effective substructures in dependency graphs. In summary, SCGK provides a unified model to combine text semantic patterns, sentence syntactic structures, and local relational substructures together, which are

*Department of Computer Science & Engineering, University of Minnesota, Twin Cities. xning@cs.umn.edu

†Machine Learning Department, NEC Labs America. qyj@cs.cmu.edu

all essential parts for solving relation extraction problems.

The rest of the paper is organized as follows. Section 2 formally defines the relation extraction problem and provides a background introduction of graph kernel. Then in Section 3.1, graph representation of natural text sentences is formulated. Section 3 introduces the convolution graph kernel for RE. Then in Section 4 we describe different levels of semi-supervision added in SCGK. A brief review of related work is provided in Section 5. Finally in Section 6, we present the experimental setup and the performance comparisons.

2 Background

2.1 Problem Definition *Relation Extraction* is a classic NLP problem, where given a sentence, it aims to detect if there exists a certain *semantic* relationship between two entities of interest in it. RE is commonly formulated as a binary classification problem as following: we treat a given sentence S as a sequence of n words (denoted by w_i with $i \in \{1, ..., n\}$), among which there exist two known entities e_1 and e_2 (that are also words).

$$S = w_1 w_2 \cdots e_1 \cdots e_2 \cdots w_{n-1} w_n \tag{2.1}$$

For a certain type of relationship R, a RE system aims to learn a function $\mathcal{F}_R$ so that

$$\mathcal{F}_R(S) = \begin{cases} +1 & \text{if } e_1 \text{ and } e_2 \text{ are associated by the relation } R \\ -1 & \text{otherwise} \end{cases}$$

All RE systems have two key components: (1) data representation, that is, how to encode the semantic and syntactic information within text sentences in a meaningful style. (2) learning algorithm which utilizes the sentence representation to optimally classify whether given sentences are related to a predefined relation R or not.

2.2 Support Vector Machines and Kernels In this paper, we adopt Support Vector Machines (SVM) [29] framework as the learning algorithm for solving RE. SVM is a widely used binary classification approach, which achieves the state-of-the-art performance in many application domains. Given a set of positive training instances $\mathcal{C}^+$ and a set of negative training instances $\mathcal{C}^-$, the SVM framework learns a classification function $f(x)$ of the following form

$$f(x) = \sum_{c_i \in \mathcal{C}^+} \lambda_i^+ \mathcal{K}(x, c_i) - \sum_{c_i \in \mathcal{C}^-} \lambda_i^- \mathcal{K}(x, c_i) \tag{2.2}$$

where λ_i^+ and λ_i^- are non-negative weights that are computed during training by maximizing a quadratic objective function. $\mathcal{K}(.,.)$ is called the *kernel* function which is computed to measure the similarity between two instances (e.g. between x and training instance c_i in equation 2.2). The kernel fucntion must satisfy two mathematical requirements: it must be symmetric, that is, $\mathcal{K}(x, x') = \mathcal{K}(x', x)$, and positive semi-definite.

2.3 Graph Kernels In our RE system we adopt the graph representation for describing the relational patterns in natural text sentences where graph nodes representing words and graph edges representing grammatical dependency between words. Details of this data representation is provided in Section 3.1. Consequently, each sentence instance x maps to a graph and the kernel fucntion $\mathcal{K}(x, c_i)$ between sentences essentially involves constructing a kernel between graphs, i.e. so-called "graph kernel".

As a structured data representation, graphs have been widely used in many real applications, e.g. chemoinformatics, drug discovery, and social networks analysis, to study relationships between structured objects. Graphs are natural data structures to model relational structures, where nodes represent objects and edges model the relations between them. Enormous efforts have been dedicated to manipulating graphs, particularly within SVM framework and kernel methods [12]. The challenge of "graph kernel" involves the definition of a kernel that captures the semantics inherence between two graphs and at the same time is reasonably efficient to evaluate.

Kernels between graphs were previously proposed by [11] (random walk graph kernels), extended by [2] and also extended to marginalized kernels on graph [19]. The specific type of graph kernels used in this paper is the convolution graph kernels [15], which aims to capture structural similarities using the substructures on two graphs. Details are provided in the next section.

3 Convolution Graph Kernel for Relation Extraction

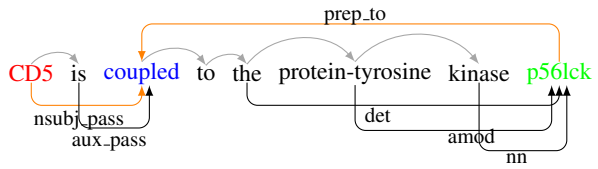
In this section, we introduce a convolution graph kernel to extract a certain semantic relationship between two entities from natural english text. The method first converts english sentences into a dependency parsing graph representation (Section 3.1), and then calculates the kernel function between two sentences using a "convolution" strategy on graphs (Section 3.2).

3.1 Dependency Graph Representation of Sentences English sentences could be naturally converted into certain graph structures where nodes represent words and edges describe connections between words. The two most popular ones include the widely used parse tree (the concrete syntax tree [6]) and the typed dependency parsing graph from the Stanford dependency scheme [24].

One problem associated with the parse tree representation is that words, which map to tree leaves, are not directly connected with other words, but via common ancestors. As a result, when a typical tree kernel tries to look for the linkage between two leaves (words) from parse trees, it is unable to directly describe the relational pattern between entities.

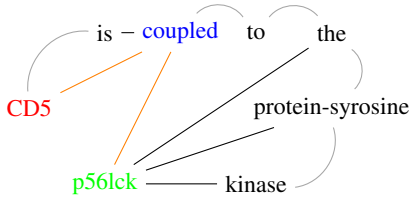
The other typed dependency graph representation is based on grammatical dependency relationship among words

Figure 1: Typed dependency representation for a sentence



In a given sentence, there exist two entities of interest CD5 and p56lck. Edges are labeled with the types of dependencies between two words. The **dark** and orange edges map to dependency relations, where orange edges are critical for relation extraction. For example, word "CD5" is the subject of word "coupled" in the passive format (i.e., nsub_pass dependency), word "kinase" is a noun compound modifier for word "p56lck" (i.e., nn), word "protein-tyrosine" is an adjectival modifier for word "p56lck" (i.e., amod dependency). The gray arrows correspond to the sequential-order edges between words, if no dependency exists.

Figure 2: Dependency graph representation of a sentence



A dependency graph representation for the sentence "CD5 is coupled to the protein-syrosine kinase p56lck.". The directions on edges are dropped. The edges are typed with their dependency labels (though not shown in the figure). Color code of edges means: (1) grey for linear-order edge; (2) black for dependency relation edge; (3) orange for edges critical for relational pattern extraction between two entities of interest (CD5 and p56lck).

[24] and has been previously used in comparsing natural language sentences [1]. In this scheme, dependencies between pairs of words are defined as a set of triplets

$$d(w_i, w_j, g_k), \tag{3.3}$$

where a *grammatical* function g_k exists from word w_i to word w_j. Dependency g is formally defined by Stanford dependency scheme to have 52 possible types (i.e. grammatical relations) and all types of dependencies belong to binary relations. An examplar list of dependency triplets for a real english sentence is provided in Figure 1.

We adopt this dependency-based graph representation in our kernel approach, where edges connecting vertices (words) are labeled with their inferred dependency function g_k from Stanford dependency parser [24]. Such a dependency-based graph naturally encodes the grammatical structure within a sentence since edges indicate grammatical roles. Compared to conventional parse trees, this graph format is more explicit and more direct in describing the syntatic information.

Figure 2 provides an example of how we construct a graph from the dependency triplets of a given sentence, i.e. Figure 1. Two modifications are added beyond dependency triplets to build the graph, (1) we drop those directions existing in dependency relations to make them undirected edges. This is a reasonable generalization since the predefined relationships covered by most RE task are symmetric, i.e. undirected. (2) It is normally observed that the semantics patterns of a word is associated with its local neighboring words to some extent, e.g. local semantic "chunk". Thus, we add edges between adjacent words (i.e., sequential-order edges from the original linear structure of the sentence). It is worth to mention that we do not add duplicate edges between nodes in the resulting graph.

To summarize, an unweighted undirected graph referred to as $\mathcal{G}_{\mathcal{D}}(S)$, is derived for a sentence S from its full set of dependency relations supplemented by its original linear-order structure (i.e. a special case of dependencies), The whole set is denoted by $D(S)$ in the following.

$$\mathcal{G}_{\mathcal{D}}(S) = (V(S), E(S)) \tag{3.4}$$

Here $V(S)$ is the set of vertices, with each $v_i \in V(S)$ representing a certain word w. $E(S)$ is the set of typed edges, each representing a dependency or a sequential pairwise order. We denote an edge in $E(S)$ as $e(v_i, v_j, k)$ where $v_i \in V(S)$, $v_j \in V(S)$ and $\exists d(w_i, w_j, g_k) \in D(S)$.

In the rest of the paper, we call such graphs as dependency graphs. We may use "sentence", "graph" and "sentence graph" interchangeably, if not specified. The same protocol applies to "word" vs "vertex", "dependency" vs "edge", respectively.

3.2 Graph Decomposition by Convolution To use SVM framework for RE, we need to define a kernel function on graphs which is able to map graphs (sentences) into a new feature space where sentences with similar entity relational patterns are closer/similar to each other compared to those graphs(sentences) with different relational patterns.

As explained in Section 2.3, designing kernels on graphs is a challenging task over years with respect to both computational complexity and discriminative power. For the target RE problem, this is even more challenging since we need to formulate a graph kernel, which takes into consideration of both semantic and syntactic structures insider natural sentences.

Most previous RE work relied the assumption that words locating between the candidate entities or connecting them are highly likely to carry information regarding the target relationship [1]. Another important observation we find is that semantic relations between entities range mostly over short substructures in the dependency graphs (Figure 2).

Figure 3: Graph decomposition into short single paths

CD5 — is CD5 — is – coupled CD5 - coupled – to
CD5 - coupled - p56lck protein-syrosine - kinase — p56lck
the – p56lck coupled – to ·····

A decomposition from dependency graph for the sentence "CD5 is coupled to the protein-syrosine kinase p56lck.". Again, the directions on edges are dropped. The edges are typed but not shown here.

Thus we propose to explore a convolution strategy to investigate dependency graphs from its possible substructures, e.g. short single paths on the graph. The idea of "convolution" kernel has been proposed previously for structured data classification. Conceptually, it states that if a big structure can be *decomposed* into some sub-structures, then by combining the kernels on sub-structures it gives a kernel on the original big structure. One competitive advantage of convolution kernel is that it analyzes structured data using a *"bottom-up"* fashion, which fits well to our representation where dependency graphs have rich substructures on different levels of details.

Then the main question becomes how to decompose a dependency graph into proper substructures and what kernels can be used on its substructures.

Inside dependency graphs we constructed in Section 3.1, each vertex is only locally related to its neighborhood (of small size). Given this observation, intuitively we could decompose a dependency graph into small units in which each vertex (word) is only connected to the close neighbors. Thus, we propose to decompose the graph into all possible single (i.e., no circles) paths *up* to a certain length. Such single short paths represent local *vertex-edge-vertex* (i.e. *word-dependency-word*) patterns, which could well preserve the grammatical relationship between vertices (words). An example of such decomposition is shown in Figure 3. If we consider the edge directions from the original dependency graphs, the decomposition can only leave us a significantly fewer number of single paths, where the most informative ones might get lost. For instance, the orange path "CD5 — coupled — p56lck" in Figure 2 does not exist in the directed version of the graph in Figure 1. This is part of the reason that we drop all the directions in the dependency graph construction.

The proposed convolution decomposition has a number of advantages. First of all, single path is much easier to handle with compared to graphs. At the same time the paths provide direct and informative signals to entity relation entraction. For example, in Figure 3, the single path "CD5 — coupled — p56lck" covers the entities of interest "CD5" and "p56lck" via a word "coupled". The existence of such single path is a strong indication of the target relationship (e.g. protein interaction relation) between the two entities. Secondly, the decomposition can be extremely fast in terms of running time if the graph is sparse, which is exactly the case for most dependency graphs. Thirdly, it is much easier to develop kernel functions for single paths. In particular, we could use the concept of "convolution" again (details discussed later in this section).

A single path p from a dependency graph $\mathcal{G}_{\mathcal{D}}(S)$ is composed from a sequence of words and their associated dependencies

$$p = (w_i, d_{i,j}, w_j, \cdots, w_p, d_{p,q}, w_q) \tag{3.5}$$

where word w_i and w_j are connected by the dependency edge $d_{i,j}$. The length of a single path is defined as the number of edges(dependencies) it contains. The entire set of up-to-size-n single paths from a sentence graph $\mathcal{G}_{\mathcal{D}}(S)$ is denoted as $\mathcal{P}^n(\mathcal{G}_{\mathcal{D}}(S))$.

3.3 Convolution Graph Kernels via Single Paths

Given two graphs S and S' decomposed into all possible single paths (up-to-size-n), a convolution kernel (denoted as $\mathcal{K}_{\mathcal{G}}$), is defined as the sum of kernels on paths (denoted as $\mathcal{K}_p$), that is

$$\mathcal{K}_{\mathcal{G}}(\mathcal{G}_{\mathcal{D}}(S), \mathcal{G}_{\mathcal{D}}(S')) = \sum_{p \in \mathcal{P}^n(\mathcal{G}_{\mathcal{D}}(S))} \sum_{p' \in \mathcal{P}^n(\mathcal{G}_{\mathcal{D}}(S'))} \mathcal{K}_p(p, p') \Pr(p|\mathcal{G}_{\mathcal{D}}(S)) \Pr(p'|\mathcal{G}_{\mathcal{D}}(S')) \tag{3.6}$$

where $\Pr(p|\mathcal{G}_{\mathcal{D}}(S))$ is the probability that single path p happens in the graph $\mathcal{G}_{\mathcal{D}}(S)$ and it can be calculated as the ratio of path count over sum of all path counts.

3.3.1 Convolution Path Kernel

In Equation 3.6, $\mathcal{K}_p(p, p')$ describes a kernel on single paths. We again apply the concept of "convolution", where a single path could be decomposed into even smaller substructures such that a convolution path kernel can be defined based on smaller substructures. Since a single path (Equation 3.5) consists of only word nodes and dependency edges, a straightforward way to decompose is to split it into words and dependencies. Therefore, we define the path kernel $\mathcal{K}_p$ as following: given two single path p and p',

$$p(w_1, d_{1,2}, w_2, \cdots, w_i, d_{i,m}, w_m)$$
$$p'(w'_1, d'_{1,2}, w'_2, \cdots, w'_j, d'_{j,n}, w'_n)$$

then

$$\mathcal{K}_p(p, p') = \begin{cases} \mathcal{K}_w(w_1, w'_1) \prod_{i=1}^{|p|-1} \{\mathcal{K}_d(d_{i,i+1}, d'_{i,i+1}) \mathcal{K}_w(w_{i+1}, w'_{i+1})\}, \\ \quad \text{if } |p| = |p'| \\ 0, \qquad \text{otherwise} \end{cases} \tag{3.7}$$

where $\mathcal{K}_w$ is a kernel defined on words and $\mathcal{K}_d$ is a kernel on dependencies. Essentially $\mathcal{K}_p(p, p')$ is the dot product

of corresponding word kernel values and dependency kernel values after aligning the two paths.

The defined path kernel $\mathcal{K}_p$ has three properties:

- First of all, there exist two different alignments between two paths (i.e., w_1 aligned against w'_1, or w_1 aligned against w'_n) which results in three different path kernel values. We use the maximum of the three values as the final kernel value between the two paths (not explicitly shown in Equation 3.7). This is to maximize the possibility that two paths are aligned optimally. However, this nonlinear $\max$ operator raises some validity issue for the graph kernel. Thus we use the method described in [27] to convert a symmetric matrix into a valid kernel matrix. This conversion uses the transductive setting, that is, kernelize the symmetric matrix including not only training data, but also the testing data (a common practice to kernelize matrices).
- Secondly, $\mathcal{K}_p$ only considers similarities between single paths with the same length, since the optimal alignment between paths of different lengths is computationally hard to handle. Also such an optimal alignment involves a sub-path alignment with the short-length path, which has been covered in $D(S)$ already.
- With Equation 3.7, the path kernel value gets smaller when the path length grows longer. Intuitively, this is desired since longer paths carry less direct pattern information related to RE.

Word Kernel $\mathcal{K}_w$ represents the kernel on words in Equation 3.7. Each word w is represented by a d-dimensional real value vetor $\mathcal{E}(w)$ (based on their patterns in a large unlabeled corpora, discussed in details in Section 4.1). Thus we define a word kernel as follows (k is a parameter to tune):

$$\mathcal{K}_w(w, w') = \exp(-k \times \|\mathcal{E}(w), \mathcal{E}(w')\|^2) \tag{3.8}$$

Dependency Kernel In Equation 3.7, $\mathcal{K}_d$ is kernel between the dependencies (including sequential-order edges). There exist only a few types of syntactic dependencies. If the two dependencies have the same type of grammatical function g associated, $\mathcal{K}_d(d, d') = 1$, otherwise the kernel value gets zero. We enforce the similarity between any grammatical dependency and sequential order as zero. Since such a similarity matrix is a valid kernel matrix after normalization.

4 Semi-Supervised Convolution Graph Kernel

4.1 Semi-Supervision on Words The dependency graph constructed in Section 3.1 mainly emphasizes the syntax structure inside a sentence. However, for relation extraction, semantic pattern is also important. So we look for strategies to introduce sentence semantics (i.e. meanings) into the kernel calculation. Since sentence meanings are heavily expressed by its words, this comes down to the question of word representation.

A typical way to represent the words is to use the dictionary index of the word in the vocabulary under consideration. Alternatively the words' POS tagging is also a good candidate, which provides a simple way to cluster words and has been used by many text analysis algorithms. However, a notable drawback with this single-integer approach is that it could not capture semantic patterns of words into account. Instead, we use a word embedding method learning to map every word in the dictionary into a d-dimensional real value vector. Words with similar meanings are mapped into points that are closer in the new d-dim space (with respect to Euclidean distance). Similarly words with different semantic meanings are represented with points locating far away from each other in the d-dim space.

There exist many ways to learn this mapping from a large unannotated text corpus. We adopt a so-called semi-supervised "language model (LM)" method proposed in [7] in our framework. LM uses a multiple-layer perceptron network classifier and modify it with a so-called "lookup table" layer (as the first layer) which converts word tokens to real value vectors. The whole LM aims to learn a latent space in which words with similar meanings can be automatically clustered together. The meanings of words are considered by looking at the word's contextual neighbors (local short word window with length 7). LM proposed a semi-supervised task that forces two sentences with the same semantic labels to have similar representations in the shared layers of neural network, and vice versa. Training for this task is achieved by assigning a positive label to genuine fragments of natural language, and negative labels to fragments that have been synthetically generated. Thus, a 50-dimension vector of real values is learned for each word (i.e., graph vertex) in the embedding space (represented by the "lookup table" layer.

Such embedding representation offers enriched semantic information of words in a context-sensitive way. For instance the most similar words of word "protein" includes "ligand, subunit, proteins, receptor, molecule" using our LM embedding. Also also the real value representation makes it possible to quantitatively *compare* semantics among words (i.e. by soft-matching of embedding vectors). Other word embedding strategy could also be used in calculating word kernel Equation 3.8. For instance, word co-occurrence-based models are very typical in text categorization field, in which researchers have tried to group words based on their co-occurrence statistics. The LM embedding we used has been shown to give better performance than co-occurrence-based models in [20].

4.2 Semi-Supervision on Dependencies Kernel value $\mathcal{K}_d(d, d')$ is actually decided by the dependency type g and g' inside d and d', respectively (see Equation 3.3). Stanford dependency scheme [24] defined totally 52 possible types of dependency. Plus the linear order edge type we add in

our graph representation, we have 53 types of edges. The similarity between various edge types are different. Thus we consider the distribution patterns of dependency edges based on their co-occurrence relationship in the unlabeled text corpus, where a co-occurrence based method is applied to generate a similarity matrix between dependency types. Then soft matching between the dependency edges becomes $\mathcal{K}_d(d, d') = \text{cooccurrence-similarity}(g, g')$, which provides further semantic evidence beyond words. Co-occurrence of two dependencies is defined as if such dependencies share a common word (3.3). That is, large dependency similarity indicates that corresponding dependencies are more likely to occur together.

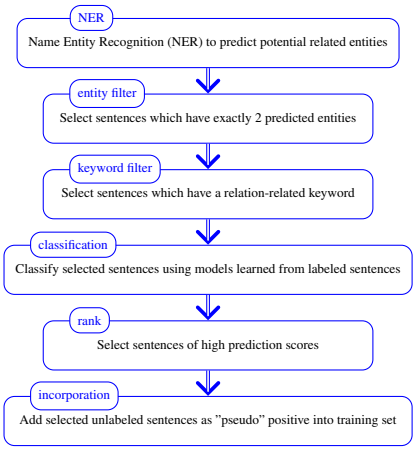
4.3 Semi-Supervision on Sentences Besides imposing semi-supervision on words and dependencies, we also introduce a semi-supervised learning strategy at the level of whole sentence. We modify a "self-training" [28] strategy to select unlabeled sentences that are highly likely to contain the target relation, and then add them as pseudo positive examples into the training set. This is motivated by the fact that the annotated data sets for RE are mostly very small which largely restricts the performance of relation extractions. Here we leverage more unlabeled sentences, which are highly likely to be positive (i.e., having relation of interest between entities), into training set in order to to boost the detection performance. Shown in Figure 4 our framework relies on a Name Entity Recognizer (NER) to recognize entities of interest first. Then we apply a keyword filter to select those sentences with interested entities recognized and containing relation-related keywords (e.g. "coupled" for the interaction relationship). The selected sentences gets classified by an existing RE system that we build from labeled data. Those sentences having high prediction scores are then used as pseudo positive examples and added into a new round of RE model training. The whole process is named as Self-Sentence-Learning (SSL) (Figure 4).

4.4 Discussion of SCGK In summary, the proposed method provides a number of advantages:

- Novel graph representation, which encodes very rich semantic and syntactic information simultaneously;
- Multi-level semi-supervision, i.e., word embedding, dependency similarity and pseudo positive sentences;
- Convolution graph kernel which recovers relation patterns between entities;
- Succinct manipulation of structured data without any requirement of manual adjustment.

Furthermore, most graph-related kernels need to be taken care of their computational costs. In case of our convolution graph kernels, computational efficiency indeed deserves some additional discussion. As we explained above, in order to generate a convolution graph kernel, two steps

Figure 4: Semi-supervision with self-sentence-learning



have to be completed. The first one is to find all possible single paths from all graphs under consideration. The second step is to compare path similarities. If these two steps are carried out independently, then a lot of computation time can be wasted on doing same calculation multiple times (i.e., calculation of similarity between two long single paths p_1 and p_2 involves the calculation of similarity between two short single paths $p'_1 \in p_1$ and $p'_2 \in p_2$). So in our implementation, we couple the two steps in order to perform the similarity calculation efficiently. All the single paths are found by finding short paths first and then extending them to longer ones. Path similarities are saved once they are calculated, and therefore they can be reused later when longer paths involve corresponding shorter paths.

5 Related Work

5.1 Kernels for RE Task Kernel methods have been previously explored on RE tasks over years. A simple tree kernel from Zelenko *et al.* [30] defined a matching function between shallow parse trees which is recursively calculated by matching all tree nodes from root to leaves. Culotta *et al.* [8] and Zhang *et al.* [31] proposed more general version of tree kernel, but are still restricted to tree representations. Later researchers began to consider representations derived from more general dependency graph structures. Bunescu *et al.* proposed a shortest path kernel in [4], but is limited to representing only a single path in the full dependency graph. Airola *et al.* proposed a all-path graph kernel in [1], which represented a sentence as a weighted dependency graph and used POS tags to represent the word vertex. The authors ran a random walk on the graph which then calculate the kernel as the sum of element-wide product between adjacency matrices of the all-path graphs. Differently from above works, recently Kuksa *et al.* [21] treated the relation extraction task as a string classification problem using a semi-supervised

string kernel approach. Word semantics patterns was also added by a semi-supervised word embedding there.

5.2 Semi-supervised Learning Supervised techniques are often restricted by the availability of labeled examples. Therefore, semi-supervised learning has become popular. The primary philosophy of semi-supervised learning is to utilize unlabeled data together with labeled data to improve performance. Unlabeled data is usually abundant. Meanwhile it is reasonable to assume that related unlabeled data can also carry useful information to benefit learning process. There exist many semi-supervised learning algorithms, including self-training [28], transductive SVM [16] graph-based regularization [33], entropy regularization [13], etc. [5] provides a nice survey of the field. Our SSL step essentially is a "self-training" algorithm.

6 Experiments

6.1 Datasets & Setup To evaluate our methods, we use five benchmark datasets which are widely used for relation extraction on biomedical literatures. These five datasets contain sentences that describe protein-protein interaction events between protein entities. The raw sentences are publicly accessible [1]. These datasets are referred as AIMED, BioInfer, HPRD50, IEPA and LLL, respectively, and Table 1 lists their characteristics. In addition to the labeled datasets as in Table 1, we randomly chose a set of 200K sentences from Pubmed[1] as the pool of unlabeled data for semi-supervised strategies.

In the labeled datasets, there might exist more than two protein entities in a single sentence, among which only two entities are the ones we are interested in and they have been indicated in the sentences. In order to distinguish positive relations existing in one sentence from each other and also from those false ones, we convert each original sentence into multiple *relation instances* such that in each relation instance, two entities of interest are labeled with PROT1, PROT2 and other protein entities are labeled using PROT. Only those relation instances, having the correct two entities of interest and with these two entities being annotated as the true relation, get positive labels.

6.1.1 SVM Model Learning We used the publicly available support vector machine tool SVM^{light} [17] that implements an efficient soft margin optimization algorithm. In all of our experiments, we study the regularization parameter C that controls the margin width, and the cost-factor parameter j by which training errors on positive examples outweigh errors on negative examples. The reason for the optimization on j is that all the datasets have notable imbalance between positive and negative training sets.

Table 1: Dataset characteristics for relation extraction

Dataset	#+ve	#-ve	#dict	#d/sen	senlen	#rel/sen
AIMED	991	4784	3180	26.9	33.0	5.0
BioInfer	2534	7053	3470	31.5	42.3	8.8
HPRD50	163	270	920	23.4	31.2	3.0
IEPA	335	482	2463	30.0	36.5	1.7
LLL	164	166	537	30.3	37.6	4.3

In this table, #+ve is the number of possitive training instances (i.e., sentences with true protein-protein interaction relations), and #-ve is the number of negative training instances. #dict is the size of dictionary (i.e,, the number of distinct words in the corresponding dataset). #d/sen is the average number of dependencies per sentence. senlen is the average sentence length. #rel/sen is the average number of protein entity pairs per sentence.

6.1.2 Name Entity Recognizer (NER) In order to perform the protein name entity recognition which is necessary in SSL step (Section 4), we use a NER system built from [20]. This NER system is based on the conditional random fields (CRFs) [23] approach, and achieves the state-of-art protein name recognition performance. Due to the space limit, we omit the details of this system.

6.2 Evaluation Metrics Our methods is evaluated using a five-fold cross validation strategy. However, how to split sentences into training and testing sets remains as an issue [1] for relation extraction tasks. Since each original sentence can produce multiple relation instances as discussed abve in Section 6.1, the train-test split needs to make sure that sentence relation instances from the same original sentence should not be split into both train and test, in order to guarantee the relative independency between the two sets. To achieve this, we first randomly split the original sentences into training and testing sets, then we generate multiple relation instances within each set. Although the raw sentences of five benchmark RE datasets are publicly available, there still lacks a benchmark enumeration and splits such that people can compare their methods with, so our processed sentences and splits are made accessible[2] for the sake of comparison.

To evaluate our system, we use *F* value and *AUC*, which are popular metrics used for relation extraction evaluation. F value is the harmonic mean of precision and recall

$$\mathrm{F} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

where precision is defined as the ratio of true positives (TP) from prediction over all predicted positives (i.e., true positives (TP) plus false positives (FP)). Recall is defined as the ratio of true positive (TP) from prediction over all

[1] http://www.ncbi.nlm.nih.gov/pubmed

[2] http://www-users.cs.umn.edu/∼xning/bionlp_data.tar.gz

positives (i.e., true positives (TP) plus false negatives (FN)). The threshold to cut prediction score is searched to give a best F value.

AUC [9] describes the normalized area under the curve that plots the true positives against the false positives for different score thresholds for classification (receiver operating characteristic curve).

6.3 Experimental Results In this section, we present the performance of semi-supervised convolution graph kernels for relation extraction on five benchmark datasets. We present a complete set of experimental results, and then discuss the influence of each of the parameters independently.

Table 2 shows detailed results on five datasets (Section 6.1), when varying the parameter k (Equation 3.8) to control word similarity and also when varying the length limit n of single paths which each dependency graph is decomposed into. The set of studies in Table 2 are conducted without semi-supervision on sentence level.

6.3.1 Effects of Single Path Lengths In Table 2 we first focus on the results for cases with maximal path length as 1, 2 or 3. Fixing the k parameter and looking at the effects of maximal single path length (i.e. comparing each row of the tables independently), the observation is consistent: across all the datasets except LLL, the best performance, in terms of both F value and corresponding AUC, happens when the dependency graph is decomposed into single paths of size 2. This conforms to our assumption as in Section 3.2, that is, entity relation patterns may be very possibly captured/represented by a single path of length 2 in form of $(w_i d_{i,j} w_j d_{j,k} w_k)$, where the two words w_i and w_k are the entities of interest, and the word w_j is the one which semantically describes the entity relation. Also note that in case of path length 2, we do not need to specify the relation-related "keyword" as some other methods do [14, 25] but the model itself can implicitly handle that.

The performance of maximal path length as 1 is worse than that of length 2, and this is expected, since the representation of word-dependency-word cannot capture the entity interaction sufficiently. In such representation, there are two possibilities to encode such interactions. One way is that the two words in one single path are the entities of interest, and they are connected through single dependency between them. However, such dependency edge only encodes syntactic information, not semantic, and thus it only leads to sub-optimal results. The other way is that the two entities of interest are separate in two single paths, and these two single paths share a common word that semantically describes the entity interaction. This encoding is less explicit and less efficient than that in a $(w_i d_{i,j} w_j d_{j,k} w_k)$ form and it requires the learner not only to discover such single paths but also to combine them to generate useful knowledge.

LLL has its best results for single paths of length 1 because we believe it is a rather easy case. LLL has fewer but more balanced training and testing instances, and words involved are also fewer. Therefore, single paths of length 1 may already be sufficient to encode necessary information for LLL. However, as long as the training data get complicated like AIMED and BioInfer, the difference can be dramatic.

The performance of maximal path length as 3 is also worse than that of 2. This may because longer paths introduce more noise that is not very relevant to the entity relation patterns. Moreover, the signals corresponding to entity relation within length-3 single paths are not as strong as those within length-2 single paths.

6.3.2 Effects of Path Set In Table 2 we used single paths "up to" a certain length. For example, for single paths of up to length 3, we use paths of length 1, 2 and 3 all together without differentiating their importance. However, there can be some alternatives. One of them is to use paths of only a certain length. We tested this method by only using paths of length 2 and 3 separately (paths of length 1 has already been tested in Table 2). The results are shown in Table 3.

From Table 3, the overall trends again remain clear and quite consistent. Using paths of only length 2 still outperforms using paths of only length 1 or only 3 most of the time (except BioInfer and LLL). This again demonstrates that single paths of 2 have higher information encoding power than paths of other lengths. However, all the "sep" performance is worse than that using paths of "up to" the same length. This illustrates that single paths of length 1 and length 3 still encode some information (e.g., content of entity interactions) that is not captured by paths of length 2 alone, and thus together they contribute to performance improvement.

6.3.3 Effects of Sequential Order We did a study on the effects of adding sequential word orders into graph representation and Table 4 shows the comparison. Note that the sequential order leads to non-zero contributions between two edges only if the two edges are both sequential orders. The results from Table 4 show that adding sequential word orders into graph representation leads to performance improvement. This indicates that the sequential orders from the sentences do provide extra information like context around words that dependencies alone may not be able to cover. Another reason could be that without sequential word orders, the dependency graphs may become disconnected with multiple components, and thus information from different components cannot be jointly utilized. We also studied the entire spectrum of "w/o seq" performance across different k values and single path lengths, and the conclusion is the same as we have in Table 4. We do not present the entire set of results here due to

space limits.

6.3.4 Effects of Dependencies Dependencies between words serve an important role in our convolution graph kernel methods, so we study the effects of dependencies by removing all dependency edges from graph representation and adding sequential orders if necessary so as to connect all sentence words. The performance comparison is shown in Table 5.

Table 5 shows that using dependencies within graph representation outperforms using only sequential orders. However, for datasets like AIMED and BioInfer, the improvement seems not very significant. The reason for this is that, in order to construct a connected graph after removing dependencies, we added sequential word orders if necessary (when we construct dependency graphs, if two adjacent words are connected by a dependency relation, then we do not use the sequential word order between them). In this way, actually some dependency edges are not missing but only edge types are replaced by sequential word order type. We studied the entire spectrum of "w/o dep" performance across different k values and single path lengths, and the conclusion is the same as we have in Table 5 so we do not present the entire set of results.

By comparing the "w/o seq" column from Table 4 and the "w/o dep" column from Table 5, we notice that "w/o dep" (i.e., only sequential orders) setting outperforms "w/o seq" (i.e., only dependencies) setting. This is no surprise because similarly as discussed above, "w/o dep" actually including many dependency connections whereas "w/o seq" setting has on average 22% fewer edges than "w/o dep".

6.3.5 Effects of Word Similarity Fixing the maximal single path length and looking at the effects of the k value, that is, looking at each column of Table 2 independently, the trend is very clear: the best performance accords to the k parameter in range 0.01 – 0.05. This is somehow independent of what datasets are used. One reason for this data independency may be that the word embedding for all the datasets is learned in a semi-supervised fashion from a common set of unlabeled data, and thus the structure of the underlying embedding space is reflected by the similar k values across all the datasets.

Word Similarity from POS Part-of-Speech (POS) tagging is another popular candidate for word representation. POS indicates the syntactic roles of words in a sentence so it does not take any semantic information. We did a study on POS word representation by replacing all word embedding with POS tags, and estimating a POS-POS similarity matrix based on co-occurrence principle. Our results show significant degradation (for example, for AIMED, the F value of POS method is 0.457 within the best parameter setting, compared to 0.562 of word embedding method). The reason can be two folds. One is that POS is weak representation to capture semantic in the graph representation. The other is the POS co-occurrence based similarity matrix is not discriminant enough, or it is ill-suited to the SCGK framework. Due to these, we did not pursue any further to use POS.

6.3.6 Effects of Dependency Similarity Matrix The dependency similarity matrix used in graph kernel is estimated using co-occurrence based method from unlabeled sentences from Pubmed. It can also be estimated solely from training data. In our experiments, we observed better results from the dependency similarity matrix learned in a semi-supervised fashion. We did not show the comparison here to save space.

6.3.7 SSL framework Table 6 presents the performance of adding sentence level SSL learning, which is described in Section 4. The conclusion is quite clear. That is, by using extra unlabeled data, we can consistently improve the RE performance.

6.3.8 Comparison with other methods Table 7 shows the comparison the best performance between our methods and other methods described in [1] and [22]. We believe that our data sets are very close to those in [1] so that somehow we can directly compare with their reported results, though the preprocessing (i.e., entity chunking, splits, etc) is not identical. For the method in [22], we tested it on our splits to guarantee a fair comparison. The method in [1] stands for the state-of-the-art performance for the graph-kernel based methods. The method in [22] stands for the state-of-the-art performance of non-graph-kernel based methods.

From Table 7, our conclusion is that our semi-supervised convolution graph kernel method is able to produce comparable or better results than the state-of-the-art peers.

7 Conclusion

In this paper, we propose a novel Semi-supervised Convolution Graph Kernel method for relation extraction from natural languages. Our method takes advantages of typed dependency relations between words which result in graph representations of sentences. Then with two semi-supervised steps based upon unlabeled text sets, we represent each vertex on this graph with word embedding capturing contextual semantics, and describe each graph edge with their semantic categories. Furthermore, we construct pseudo training sentences utilizing unlabeled sentences in order to expand the training set and improve the prediction performance (i.e., semi-supervised learning on sentence level).

Our proposed kernel provides a power model to capture both semantic and syntactic evidence inside natural sentences. We demonstrate that our system reaches or outperforms the state-of-the-art performance on five relation

Table 7: Comparison of SCGK with other methods

Dataset	all-path		ASK		SCGK		SSL-SCGK	
	F	AUC	F	AUC	F	AUC	F	AUC
AIMED	0.564	0.848	0.554	0.824	0.562	0.821	0.572	0.834
BioInfer	0.613	0.819	0.614	0.798	0.606	0.799	0.613	0.806
HPRD50	0.797	0.730	0.727	0.777	0.762	0.819	0.767	0.819
IEPA	0.751	0.851	0.735	0.809	0.737	0.791	0.740	0.797
LLL	0.768	0.834	0.850	0.823	0.849	0.841	0.860	0.847

In this table, all-path is the method described in [1]. ASK is the string kernel method described in [22]. SCGK is our convolution graph kernel based method without semi-supervision on sentence level. SSL-SCGK is the convolution graph kernel method with semi-supervision on sentence level.

extraction benchmark data sets from biomedical literature. Also with three levels of semi-supervision, our system is feasible to work on RE problems with very few training examples.

References

[1] A. Airola, S. Pyysalo, J. Björne, T. Pahikkala, F. Ginter, and T. Salakoski. All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. *BMC bioinformatics*, 9 Suppl 11, 2008.

[2] K. M. Borgwardt, C. S. Ong, S. Schnauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21 Suppl 1:i47–i56, Jun 2005.

[3] M. Bundschus, M. Dejori, M. Stetter, V. Tresp, and H.-P. Kriegel. Extraction of semantic biomedical relations from text using conditional random fields. *BMC Bioinformatics*, 9(1):207, 2008.

[4] R. C. Bunescu and R. J. Mooney. A shortest path dependency kernel for relation extraction. *HLT '05: articles of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 724–731, 2005.

[5] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[6] E. Charniak. A maximum-entropy-inspired parser. Providence, RI, USA, 1999. Brown University.

[7] R. Collobert and J. Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 160–167, New York, NY, USA, 2008. ACM.

[8] A. Culotta and J. Sorensen. Dependency tree kernels for relation extraction. *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 423, 2004.

[9] T. Fawcett. Roc graphs: Notes and practical considerations for researchers, 2004.

[10] K. Fundel, R. Küffner, and R. Zimmer. Relex—relation extraction using dependency parse trees. *Bioinformatics*, 23(3):365–371, 2007.

[11] T. Gaertner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop*, pages 129–143. Springer-Verlag, August 2003.

[12] T. Gärtner, P. A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *COLT*, pages 129–143, 2003.

[13] Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. In *NIPS'05*, pages 529–536, 2005.

[14] Z. GuoDong, S. Jian, Z. Jie, and Z. Min. Exploring various knowledge in relation extraction. *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 427–434, 2005.

[15] D. Haussler. Convolution kernels on discrete structures. Technical report, University of Santa Cruz, 1999.

[16] T. Joachims. Transductive inference for text classification using support vector machines. In *ICML '99*, pages 200–209, 1999.

[17] T. Joachims. Optimizing search engines using clickthrough data. *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, 2002.

[18] N. Kambhatla. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 22, 2004.

[19] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. pages 321–328, 2003.

[20] P. Kuksa and Y. Qi. Semi-supervised bio-named entity recognition with word-codebook learning. 2010.

[21] P. Kuksa, Y. Qi, B. Bai, R. Collobert, J.Weston, V. Pavlovic, and X. Ning. Semi-supervised abstraction-augmented string kernel for multi-level bio-relation extraction. In *ECML PKDD 2010*, 2010.

[22] P. Kuksa, Y. Qi, B. Bai, R. Collobert, J. Weston, V. Pavlovic, and X. Ning. Semi-supervised abstraction-augmented string kernel for multi-level bio-relation extraction. In *Machine Learning and Knowledge Discovery in Databases*, volume

6322 of *Lecture Notes in Computer Science*, pages 128–144. Springer Berlin / Heidelberg, 2010.

[23] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[24] M. Marneffe, B. Maccartney, and C. Manning. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC-06*, pages 449–454, 2006.

[25] D. P. T. Nguyen, Y. Matsuo, and M. Ishizuka. Subtree mining for relation extraction from wikipedia. *NAACL '07: Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics*, pages 125–128, 2007.

[26] B. Rosario and M. A. Hearst. Classifying semantic relations in bioscience texts. *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 430, 2004.

[27] H. Saigo, J.-P. Vert, N. Ueda, and T. Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–1689, 2004.

[28] H. Scudder. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, 11(3):363–371, 1965.

[29] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, September 1998.

[30] D. Zelenko, C. Aone, A. Richardella, J. K, T. Hofmann, T. Poggio, and J. Shawe-taylor. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3:2003, 2003.

[31] M. Zhang, J. Zhang, J. Su, and G. Zhou. A composite kernel to extract relations between entities with both flat and structured features. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 825–832, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[32] S. Zhao and R. Grishman. Extracting relations with integrated information using kernel methods. *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 419–426, 2005.

[33] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML'03*, pages 912–919, 2003.

Table 2: Results for SCGK method

AIMED

k	1		2		3	
	F	AUC	F	AUC	F	AUC
0.020	0.535	0.796	0.561	0.818	0.547	0.815
0.030	0.540	0.795	**0.562**	**0.821**	0.561	0.818
0.050	0.530	0.790	0.553	0.814	0.550	0.813
0.100	0.521	0.778	0.540	0.802	0.537	0.795

BioInfer

k	1		2		3	
	F	AUC	F	AUC	F	AUC
0.005	0.485	0.645	0.498	0.672	0.509	0.685
0.010	0.606	0.788	**0.606**	**0.799**	0.568	0.753
0.020	0.470	0.617	0.484	0.639	0.492	0.648
0.030	0.465	0.589	0.480	0.630	0.489	0.647

HPRD50

k	1		2		3	
	F	AUC	F	AUC	F	AUC
0.005	0.727	0.778	0.748	0.793	0.734	0.790
0.010	0.733	0.787	0.757	0.808	0.753	0.804
0.020	0.755	0.812	**0.762**	**0.819**	0.750	0.811
0.030	0.739	0.804	0.752	0.816	0.740	0.812

IEPA

k	1		2		3	
	F	AUC	F	AUC	F	AUC
0.005	0.708	0.763	0.715	0.758	0.708	0.750
0.010	0.712	0.771	0.725	0.783	0.622	0.500
0.020	0.721	0.782	**0.737**	**0.791**	0.733	0.794
0.030	0.720	0.781	0.731	0.788	0.724	0.781
0.050	0.718	0.770	0.719	0.772	0.622	0.500

LLL

k	1		2		3	
	F	AUC	F	AUC	F	AUC
0.020	0.832	0.822	0.836	0.825	0.843	0.826
0.030	0.842	0.830	0.846	0.831	0.829	0.811
0.050	**0.849**	**0.841**	0.833	0.823	0.830	0.805
0.100	0.726	0.536	0.734	0.572	0.764	0.614

In this table, k is the parameter in Equation 3.8 to control word similarity. Columns under "1", "2" and "3" correspond to the maximum path length 1, 2 and 3, respectively, up to which each dependency graph is decomposed into. Columns corresponding to "F" and "AUC" are the results in terms of F value and AUC, respectively. The **bold** numbers are the best performance given corresponding parameters.

Table 3: Effects of path set

	mthd	1		2		3	
		F	AUC	F	AUC	F	AUC
AIMED	upto	0.540	0.795	0.562	0.821	0.561	0.818
	sep			0.560	0.815	0.549	0.800
BioInfer	upto	0.606	0.788	0.606	0.799	0.568	0.753
	sep			0.591	0.776	0.469	0.594
HPRD50	upto	0.755	0.812	0.762	0.819	0.750	0.811
	sep			0.757	0.813	0.738	0.798
IEPA	upto	0.721	0.782	0.737	0.791	0.733	0.794
	sep			0.732	0.796	0.708	0.785
LLL	upto	0.849	0.841	0.833	0.823	0.830	0.805
	sep			0.825	0.754	0.816	0.740

In this table, columns of 1, 2 and 3 correspond to the results of using paths of length 1, 2 and 3, respectively. Rows of method "upto" show the results if using single paths up to the corresponding length, and rows of method "sep" show the results if using single paths separately. The results in this table are from the best parameter settings as in Table 2.

Table 4: Effects of sequential order

Dataset	w/o seq		w/ seq		imprv	
	F	AUC	F	AUC	F	AUC
AIMED	0.549	0.805	0.562	0.821	2.3%	2.0%
BioInfer	0.601	0.790	0.606	0.799	0.8%	1.1%
HPRD50	0.740	0.792	0.762	0.819	2.1%	3.4%
IEPA	0.708	0.774	0.737	0.791	4.1%	2.2%
LLL	0.805	0.769	0.849	0.841	5.5%	9.4%

In this table, columns of "w/o seq" are the results when no sequential orders are added in graph representation (i.e., only dependencies). Note in this case, the graphs can be disconnected. Columns of "w/ seq" are the results when sequential orders are added in graph representation. Columns of "imprv" are the percentage improvement brought by adding sequential orders. Columns of "F" and "AUC" are results in terms of F value and AUC. The results of "w/ seq" are from the best performance and corresponding parameters in Table 2. The results of "w/o seq" have the same experimental settings (i.e., path length, k, etc) as those of "w/ seq" except the sequential orders are absent.

Table 5: Effects of dependencies

Dataset	w/o dep		w/ dep		imprv	
	F	AUC	F	AUC	F	AUC
AIMED	0.559	0.817	0.562	0.821	0.5%	0.5%
BioInfer	0.601	0.794	0.606	0.799	0.8%	0.6%
HPRD50	0.726	0.792	0.762	0.819	5.0%	3.4%
IEPA	0.713	0.779	0.737	0.791	3.4%	1.5%
LLL	0.826	0.797	0.849	0.841	2.8%	5.5%

In this table, columns of "w/o dep" are the results when no dependencies are used in graph representation (i.e., only sequential word orders). Columns of "w/ dep" are the results when dependencies are added in graph representation. Columns of "imprv" are the percentage improvement brought by adding dependencies. Columns of "F" and "AUC" are results in terms of F value and AUC. The results of "w/ dep" are from the best performance and corresponding parameters in Table 2. The results of "w/o dep" have the same experimental settings (i.e., path length, k, etc) as those of "w/ dep" except the sequential orders are absent.

Table 6: Results from SSL

Dataset	w/o SSL		w/ SSL			imprv	
	F	AUC	thrd	F	AUC	F	AUC
AIMED	0.562	0.821	0.0	0.572	0.834	1.8%	1.6%
BioInfer	0.606	0.799	0.0	0.613	0.806	1.2%	0.9%
HPRD50	0.762	0.819	1.0	0.767	0.819	0.7%	0.0%
IEPA	0.737	0.791	0.8	0.740	0.797	0.4%	0.8%
LLL	0.849	0.841	1.1	0.860	0.847	1.3%	0.7%

In this table, columns corresponding to "w/o SSL" are the results from supervised learning setting, and the columns corresponding to "w/ SSL" are the results from semi-supervised setting. "thrd" under "w/ SSL" is the threshold we used to choose additional training instances (i.e., the sentences have a prediction score from supervised model higher than this threshold so as to be chosen as additional training instances.) The results for "w/o SSL" are from the best parameter settings as in Table 2.