Chapter 1

Survey on Kernel-Based Relation Extraction

Hanmin Jung, Sung-Pil Choi, Seungwoo Lee and Sa-Kwang Song

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/51005

1. Introduction

Relation extraction refers to the method of efficient detection and identification of predefined semantic relationships within a set of entities in text documents (Zelenco, Aone, & Richardella, 2003; Zhang, Zhou, and Aiti, 2008). The importance of this method was recognized first at the Message Understanding Conference (MUC, 2001) that had been held from 1987 to 1997 under the supervision of DARPA[1]. After that, the Automatic Content Extraction (ACE, 2009) Workshop facilitated numerous researches that from 1999 to 2008 had been promoted by NIST[2] as a new project. Currently, the workshop is held every year being the greatest world forum for comparison and evaluation of new technology in the field of information extraction such as named entity recognition, relation extraction, event extraction, and temporal information extraction. This workshop is conducted as a sub-field of Text Analytics Conference (TAC, 2012) which is currently under the supervision of NIST.

According to ACE, an entity in the text is a representation for naming a real object. Exemplary entities include the names of persons, locations, facilities and organizations. A sentence including these entities can express the semantic relationships in between them. For example, in the sentence "*President Clinton was in Washington today,*" there is the "*Located*" relation between "*Clinton*" and "*Washington*". In the sentence "*Steve Balmer, CEO of Microsoft, said…*" the relation of "*Role* (*CEO*, *Microsoft*)" can be extracted.

Many relation extraction techniques have been developed in the framework of various technological workshops mentioned above. Most relation extraction methods developed so far are based on supervised learning that requires learning collections. These methods are clas-

1 Defense Advanced Research Projects Agency of the U.S.

2 National Institute of Standards and Technology of the U.S.

sified into feature-based methods, semi-supervised learning methods, bootstrapping methods, and kernel-based methods (Bach & Badaskar, 2007; Choi, Jeong, Choi, and Myaeng, 2009). Feature-based methods rely on classification models for automatically specifying the category where a relevant feature vector belongs. At that, surrounding contextual features are used to identify semantic relations between the two entities in a specific sentence and represent them as a feature vector. The major drawback of the supervised learning-based methods, however, is that they require learning collections. Semi-supervised learning and bootstrapping methods, on the other hand, use a large corpora or web documents, based on reduced learning collections that are progressively expanded to overcome the above disadvantage. Kernel-based methods (Collins & Duffy, 2001), in turn, devise kernel functions that are most appropriate for relation extraction and apply them for learning in the form of a kernel set optimized for syntactic analysis and part-of-speech tagging. The kernel function itself is used for measuring the similarity between two instances, which are the main objects of machine learning. General kernel-based models will be discussed in detail in Section 3.

As one representative approach of the feature-based methods, (Kambhatla, 2004) combines various types of lexical, syntactic, and semantic features required for relation extraction by using maximum entropy model. Although it is based on the same type of composite features as that proposed by Kambhatla (2004), Zhou, Su, Zhang, and Zhang (2005) make the use of support vector machines for relation extraction that allows flexible kernel combination. Zhao and Grishman (2005) have classified all features available by that point in time in order to create individual linear kernels, and attempted relation extraction by using composite kernels made of individual linear kernels. Most feature-based methods aim at applying feature engineering algorithms for selecting optimal features for relation extraction, and application of syntactic structures was very limited.

Exemplary semi-supervised learning and bootstrapping methods are Snowball (Agichtein & Gravano, 2000) and DIPRE (Brin, 1999). They rely on a few learning collections for making the use of bootstrapping methods similar to the Yarowsky algorithm (Yarowsky, 1995) for gathering various syntactic patterns that denote relations between the two entities in a large web-based text corpus. Recent developments include KnowItAll (Etzioni, et al., 2005) and TextRunner (Yates, et al., 2007) methods for automatically collecting lexical patterns of target relations and entity pairs based on ample web resources. Although this approach does not require large learning collections, its disadvantage is that many incorrect patterns are detected through expanding pattern collections, and that only one relation can be handled at a time.

Kernel-based relation extraction methods were first attempted by Zelenco, et al. (2003). Zelenco, et al., devised contiguous subtree kernels and sparse subtree kernels for recursively measuring similarity of two parse trees in order to apply them to binary relation extraction that demonstrated relatively high performance. After that, a variety of kernel functions for relation extraction have been suggested, e.g., dependency parse trees (Culotta and Sorensen, 2004), convolution parse tree kernels (Zhang, Zhang and Su, 2006), and composite kernels (Choi et al., 2009; Zhang, Zhang, Su and Zhou, 2006), which show even better performance.

In this chapter, case analysis was carried out for kernel-based relation extraction methods, which are considered to be the most successful approach so far. Of course, some previous

http://dx.doi.org/10.5772/51005

survey papers based on the importance and effect of the methodology have been published (Bach and Badaskar, 2007; Moncecchi, Minel and Wonsever, 2010). However, they fail to fully analyze particular functional principles or characteristics of the kernel-based relation extraction models announced so far, and just cite the contents of individual articles or describe limited analysis. Although the performance of most kernel-based relation extraction methods has been demonstrated on the basis of ACE evaluation collections, comparison and analysis of the overall performance has not been made so far.

This chapter, unlike existing case studies, makes a close analysis of operation principles and individual characteristics of five kernel-based relation extraction methods starting from Zelenco, et al. (2003) which is the source of kernel-based relation extraction studies, to the composite kernel, which is considered the most advanced kernel-based relation method (Choi, et al., 2009; Zhang, Zhang, Su, et al., 2006). The focus will be laid on the ACE collection to compare the overall performance of each method. We hope this study will contribute to further research of kernel-based relation extraction of even higher performance and to high-level general kernel studies for linguistic processing and text mining.

Section 2 outlines supervised learning-based relation extraction methods and in section 3 we discuss kernel-based machine learning. Section 4 closely analyzes five exemplary kernel-based relation extraction methods. As mentioned above, Section 5 also compares the performance of these methods to analyze advantages and disadvantages of each method. Section 6 draws a conclusion.

2. Supervised learning-based relation extraction

As discussed above, relation extraction methods are classified into three categories. The difference between feature-based and kernel-based methods is shown in the following Figure 1. With respect to machine learning procedure, these two are different from semi-supervised learning methods.

On the left of Figure 1, individual sentences that make up a learning collection have at least two entities (black square) of which the relation is manually extracted and predefined. Since most relation extraction methods studied so far work with binary relations, learning examples are modified for convenient relation extraction from the pair of entities by preprocessing the original learning collection. These modified learning examples are referred to as *relation instance*. The relation instance is defined as an element of the learning collection modified so that it can be efficiently applied to the relevant relation extraction methods on the basis of specific sentence that contains at least two entities.

The aforementioned modification is closely related to feature information used in relation extraction. Since most supervised learning-based methods use both the entity itself and the contextual information about the entity, it is important to collect contextual information efficiently for improving performance. Linguistic processing (part-of-speech tagging, base phrase recognition, syntactic analysis, etc.) for individual learning sentences in the

pre-processing step contributes to making a base for effective feature selection and extraction. For example, when one sentence shown in the above Figure 1 goes through syntactic analysis, one relation instance is composed of a parse tree and the locations of entities indicated in the parse tree (Fundel, Küffner, & Zimmer, 2007; Zhang, et al., 2008; Zhou, Zhang, Ji, and Zhu, 2007). A single sentence can be represented as a feature vector or a syntactic graph (Jiang and Zhai, 2007; W. Li, Zhang, Wei, Hou, and Lu, 2008; Zhang, Zhang, and Su, 2006). The type of such relation instances depends on the relation extraction methods, and can involve various preprocessing tasks as well (D. P. T. Nguyen, Matsuo, and Ishizuka, 2007; Zhang, Zhang, and Su, 2006).

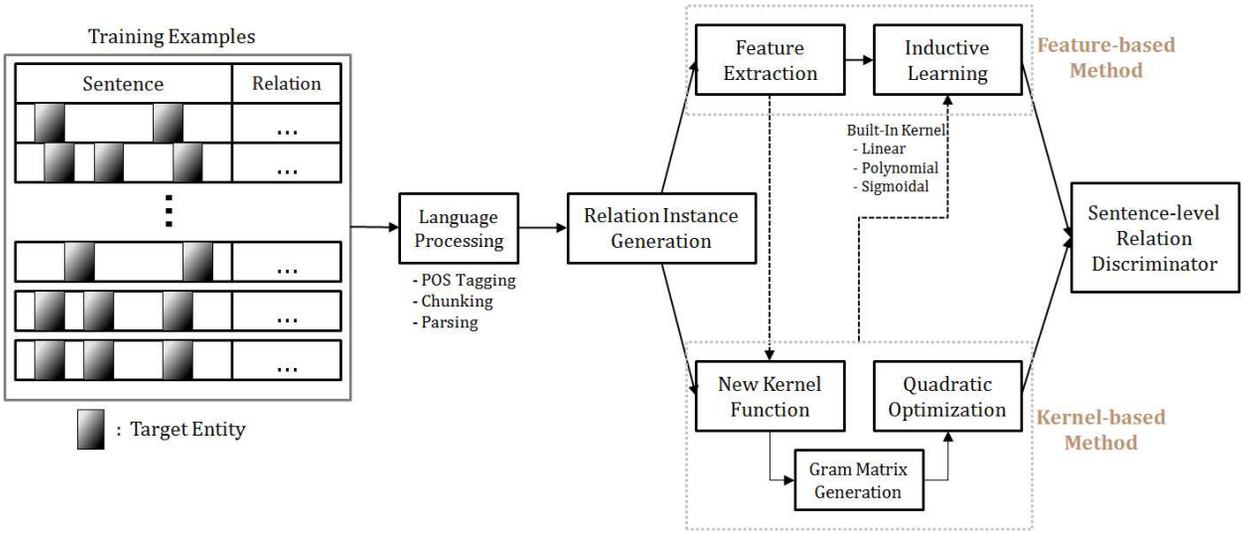

Figure 1. Learning process for supervised learning-based relation extraction.

In general, feature-based relation extraction methods follow the procedures shown in the upper part of Figure 1. That is, feature collections that "*can express individual learning examples the best*" are extracted (feature extraction.) The learning examples are feature-vectorized, and inductive learning is carried out using selected machine learning models. On the contrary, kernel-based relation extraction shown in the lower part of Figure 1 devises a kernel function that "*can calculate similarity of any two learning examples the most effectively*" to replace feature extraction process. Here, the measurement of similarity between the learning examples is not general similarity measurement in a general sense. That is, the function for enhancing similarity between the two sentences or instances that express the same relation is the most effective kernel function from the viewpoint of relation extraction. For example, two sentences "*Washington is in the U.S.*" and "*Seoul is located in Korea*" use different object entities but feature the same relation ("*located*.") Therefore, an efficient kernel function would detect a high similarity between these two sentences. On the other hand, since the sentences "*Washington is the capital of the United States*" and "*Washington is located in the United States*" express the same object entities but different relations, the similarity between them should be determined as very low. As such, in kernel-based relation

http://dx.doi.org/10.5772/51005

extraction methods, the selection and creation of kernel functions are the most fundamental part that affects the overall performance.

As shown in the Figure 1, kernel functions (linear, polynomial, and sigmoid) can be used in feature-based methods as well. They are, however, functions applied only to instances expressed with vectors. On the other hand, kernel-based methods are not limited in terms of the type of the instance, and thus can contain various kernel functions.

3. Overview of kernel-based machine learning methods

Most machine learning methods are carried out on a feature basis. That is, each instance to which an answer (label) is attached, is modified into a feature sequence or *N*-dimensional vector f_1, f_2, …, f_N to be used in the learning process. For example, important features for identifying the relation between two entities in a single sentence are entity type, contextual information between, before and after entities' occurrence, part-of-speech information for contextual words, and dependency relation path information for the two entities (Choi et al., 2009; Kambhatla, 2004; W. Li, Zhang, et al., 2008; Zhang, Zhang, and Su, 2006). These data are selected as a single feature, respectively, to be represented with a vector for automatic classification of the relation between the entities.

In section 2, we discussed that the essence of feature-based methods is to create a feature vector that can best express individual learning examples. However, in many cases, it is not possible to express the feature vector reasonably. For example, a feature space is required for expressing syntactic information[3] of a specific sentence as a feature vector, and it is almost impossible to express it as a feature vector in a limited space in some cases (Cristianini and Shawe-Taylor, 2000). Kernel-based methods are for learning by calculating kernel functions between two examples while keeping the original learning example without additional feature expression (Cristianini and Shawe-Taylor, 2000). The kernel function is defined as the mapping $K : \boldsymbol{X} \times \boldsymbol{X} \rightarrow [0, \infty)$ from the input space $\boldsymbol{X}$ to the similarity score $\phi(x) \cdot \phi(y) = \sum_i \phi_i(x)\phi_i(y)$. Here, $\phi(x)$ is the mapping function from learning examples in the input space υ to the multidimensional feature space. The kernel function is symmetric, and exhibits positive semi-definite features. With the kernel function, it is not necessary to calculate all features one by one, and machine learning can thus be carried out based only on similarity between two learning examples. Exemplary models where learning is carried out on the basis of all similarity matrices between learning examples include Perceptron (Rosenblatt, 1958), Voted Perceptron (Freund and Schapire, 1999), and Support Vector Machines (Cortes and Vapnik, 1995) (Moncecchi, et al., 2010). Recently, kernel-based matching learning methods draw increasingly more attention, and are widely used for pattern recognition, data and text mining, and web mining. The per-

3 Dependency grammar relation, parse tree, etc. between words in a sentence.

formance of kernel methods, however, depends to a great extent on kernel function selection or configuration (J. Li, Zhang, Li, and Chen, 2008).

Kernel-based learning methods are used also for natural language processing. Linear, polynomial and Gaussian kernels are typical in simple feature vector-based machine learning. Convolutional kernel (Collins & Duffy, 2001) is used for efficient learning of structural data such as trees or graphs. The convolution kernel is a type of kernel function featuring the idea of sequence kernels (Lodhi, Saunders, Shawe-Taylor, Cristianini, & Watkins, 2002), tree kernels (Culotta & Sorensen, 2004; Reichartz, Korte, & Paass, 2009; Zelenco, et al., 2003; Zhang, Zhang, & Su, 2006; Zhang, et al., 2008), and graph kernels (Gartner, Flach, & Wrobel, 2003). The convolutional kernel can measure the overall similarity by defining "sub-kernels" for measuring similarity between the components of an individual entity and calculating similarity convolution among the components. For example, the sequence kernel divides the relevant sequence into subsequences for measuring similarity of two sequences to calculate overall similarity by means of similarity measurement between the subsequences. Likewise, the tree kernel divides a tree into its sub-trees to calculate similarity between them and then it calculates the convolution of these similarities.

As described above, another advantage of the kernel methods is that learning is possible as a single kernel function for input instance collections of different type. For example, (Choi et al., 2009; Zhang, Zhang, Su, et al., 2006) have demonstrated a composite kernel for which the convolutional parse tree kernel is combined with the entity kernel for high-performance relation extraction.

4. Kernel-based relation extraction

The most prominent characteristic of the relation extraction models derived so far is that linguistic analysis is used to carefully identify relation expressions and syntactic structures directly and indirectly expressed in specific sentences. In this section, five important research results are discussed and analyzed. Of course, there are many other important studies that have drawn much attention due to their high performance. Most of approaches, however, just modify or supplement the five basic methods discussed below. Therefore, this study can be an important reference for supplementing existing research results in the future or studying new mechanisms for relation extraction, by intuitively explaining the details of major studies. Firstly, tree kernel methods originally proposed by Zelenco, et al. (2003) are covered in detail. Then, the method proposed by Culotta & Sorensen (2004) is covered where the dependency tree kernel was used for the first time. Also, kernel-based relation extraction (Bunescu & Mooney, 2005) using dependency path between two entities in a specific sentence on the basis of similar dependency trees is discussed. Additionally, the subsequence kernel-based relation extraction method proposed by (Bunescu & Mooney (2006) is explained. Finally, the relation extraction models (Zhang, Zhang, Su, et al., 2006) based on the composite kernel for which various kernels are combined on the basis of the convolution parse tree kernel proposed by Collins & Duffy (2001) are covered in detail.

4.1. Tree kernel-based method (Zelenco, et al., 2003)

This study is known as the first application of kernel method to relation extraction. The parse trees derived from shallow parsing are used for measuring similarity between the sentences containing entities. In their study, the REES (Relation and Event Extraction System) is used to analyze part-of-speeches and types of individual words in a sentence, as well as the syntactic structure of the sentence in question. Here, the REES is a relation and event extraction system developed by Aone & Ramos-Santacruz (2000). The Figure 2 below shows example result from the REES.

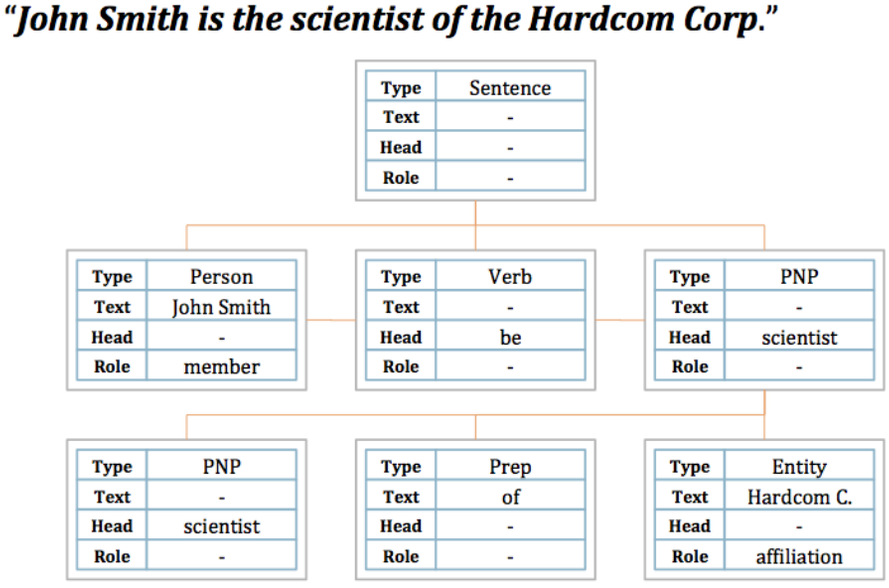

Figure 2. Exemplary shallow parsing result for relation extraction.

As shown in Figure 2, when the syntactic analysis of the input sentence is completed, particular information for words of the sentence is analyzed and extracted. Four types of attribute information are attached to all words or entities other than articles and stop words. Type represents the part-of-speech or entity type of the current word. While "*John Smith*" is the "*Person*" type, "*scientist*" is specified as "PNP" representing a personal noun. Head represents the presence of key words of composite nouns or preposition phrases. Role represents the relation between the two entities. In Figure 2, "*John Smith*" is the "*member*" of "*Hardcom C.*" In its turn, "*Hardcom C.*" is an "*affiliation*" of "*John Smith*".

As one can see, it is possible to identify and extract the relation between the two entities in a specific sentence at a given level if the REES is used. Since the system, however, was developed on the basis of rules, it has some limitations in terms of scalability and generality (Zelenco et al., 2003). Zelenco et al. (2003) have constructed tree kernels on the basis of the REES analysis result with a view of better relation extraction so as to overcome the limitations of machine learning and low performance. The kernel function defined in this study for measuring the similarity of a pair of shallow parsing trees consists of the following chain of equations. The comparison function for each individual configuration node in the trees is as follows.

$$t(P_1.p,\ P_2.p)=\begin{cases}1, & \text{if } P_1.Type=P_2.Type \text{ and } P_1.Role=P_2.Role\\ 0, & \text{otherwise}\end{cases} \quad (1)$$

In this equation, $P_i.p$ represents a specific parent node in the parsing tree; $P_i.Type$ represents word and entity type information; and $P_i.Role$ represents relation information between the two entities. Equation 1 is called *matching function*, and is used for comparing between the part-of-speeches, entity type and relation type information for each node. If both type and role are the same as in binary comparison, 1 is returned and otherwise 0.

$$k(P_1.p,\ P_2.p)=\begin{cases}1, & \text{if } P_1.Text=P_2.Text\\ 0, & \text{otherwise}\end{cases} \quad (2)$$

Equation 2 represents the function for deciding whether two nodes comprise the same words or entities. (Zelenco et al., 2003) named this *similarity function*. The recursive kernel function K_c for the child node of a specific parent node, is defined as follows on the basis of the two functions. Although all the functions above do not use "Head" field in each node for simplicity in (Zelenco et al., 2003), it would be valuable to use the field for better performance.

$$\begin{aligned}
&K_c(P_1.c,\ P_2.c)=\sum_{i,j,l(\boldsymbol{i})=l(j)} SSK(P_1.c,\ P_2.c,\ \boldsymbol{i},\ \boldsymbol{j})\\
&SSK(P_1.c,\ P_2.c,\ \boldsymbol{i},\ \boldsymbol{j})=\lambda^{d(\boldsymbol{i})}\lambda^{d(\boldsymbol{i})}K(P_1[\boldsymbol{i}],\ P_2[\boldsymbol{j}])\prod_{s=1,\ldots,l(\boldsymbol{i})} t(P_1[i_s],\ P_2[j_s])\\
&K(P_1[\boldsymbol{i}],\ P_2[\boldsymbol{j}])=\textstyle\sum_{s=1,\ldots,l(\boldsymbol{i})} K(P_1[i_s],\ P_2[j_s])\\
&\boldsymbol{i}=\{i_1,\ i_2,\ \ldots,\ i_n \mid i_1\le i_2\le\ldots\le i_n\}\\
&d(\boldsymbol{i})=i_n-i_1+1\\
&l(\boldsymbol{i})=n
\end{aligned} \quad (3)$$

In Equation 3, $P_i.c$ represents the child nodes of the specific node ($P_i.p$). $SSK(P_1.c, P_2.c, \mathrm{i}, \mathrm{j})$ is the function for calculating subsequence similarity between child nodes ($P_i.c$) of $P_1.p$ and $P_2.p$. Here, i is the index representing all the subsequences of the child nodes of $P_i.p$. $^{d(i)}$, $0<$ <1 is the weight factor depending on the length of the child node subsequences. This variable determines how many subsequences of the specific child node are contained in the entire child nodes in order to lower the similarity in case of multiple matching subsequences. d(i) represents the distance between the first and last nodes of the currently processing subsequence i, and l(i) represents the number of the nodes of i. The kernel function between the two trees is defined as follows.

$$K(P_1,\ P_2)=\begin{cases}0, & \text{if } t(P_1.p,\ P_2.p)=0\\ k(P_1.p,\ P_2.p)+K_c(P_1.c,\ P_2.c), & \text{otherwise}\end{cases} \quad (4)$$

P_i represents the tree to be compared. The similarity between the two trees is calculated by adding up the similarity function k (Equation 2) for the current node (parent node) and the similarity calculation function K_c (Equation 3) between the child nodes. The kernel calculation process for the following parse trees will be described in detail later for intuitive understanding of the kernel function.

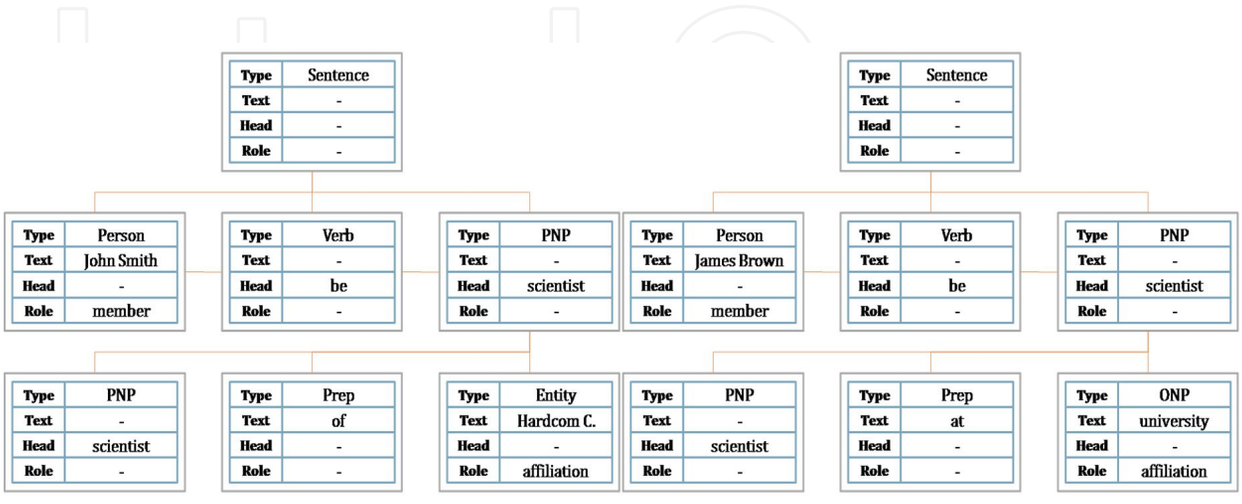

Figure 3. Two sample parse trees for illustrating kernel calculation process.

The original sentence of the parse tree on the left side is "*John Smith is a chief scientist of Hardcom C.*", and the original sentence of the tree on the right side is "*James Brown is a scientist at the University of Illinois*." For convenience, the left-side tree is referred to as P_1, and the right-side tree as P_2. For easy explanation, "*chief*" and "*University of Illinois*" are removed or abbreviated. The kernel function for the two trees is primarily expressed as in the following.

$$\begin{aligned} K(P_1, P_2) &= k(P_1.Sentence.p, P_2.Sentence.p) \\ &\quad + K_c([P_1.Person, P_1.Verb, P_1.PNP], [P_2.Person, P_2.Verb, P_2.PNP]) \\ &= k(P_1.Sentence.p, P_2.Sentence.p) \\ &\quad + \sum_{i,j,l(i)=l(j)} SSK([P_1.Person, P_1.Verb, P_1.PNP], [P_2.Person, P_2.Verb, P_2.PNP], \boldsymbol{i}, \boldsymbol{j}) \end{aligned} \tag{5}$$

Equation 4 is used to calculate the tree kernel between the two trees P_1 and P_2, Here $P_i.Sentence.p$ represents the root node of i-th tree. Equation 3 is used to calculate K_c for each child node of the root node to expand it into the sum of the *SSK* function. The Figure 4 below shows the process of calculating the kernel of the *SSK* function.

Figure 4 shows the process of calculating the kernel function between the second-level child nodes of the two trees. Since all nodes at this level have unexpectedly the matching node type as shown in Figure 3, kernel similarity between the subsequences of each matching node as shown in the equations on the right side of Figure 3. Since matching only between subsequences of the same length is implemented as in Equation 3, non-matching subsequen-

ces are excluded from kernel calculation through conformity check among subsequences of which the length is 1, 2 and 3 respectively. The result of kernel calculation is as follows.

$$
\begin{aligned}
K(P_1,P_2) &= k(P_1.Sentence.p, P_2.Sentence.p) \\
&\quad + K_c([P_1.Person, P_1.Verb, P_1.PNP],[P_2.Person, P_2.Verb, P_2.PNP]) \\
&= \lambda^2\{K(P_1.Person, P_2.Person) + K(P_1.Verb, P_2.Verb) + K(P_1.PNP, P_2.PNP)\} \quad \rightarrow (l(\mathbf{i})=1, d(\mathbf{i})=1) \\
&\quad + \lambda^4\{K(P_1.Person, P_2.Person) + 2K(P_1.Verb, P_2.Verb) + K(P_1.PNP, P_2.PNP)\} \quad \rightarrow (l(\mathbf{i})=2, d(\mathbf{i})=2) \\
&\quad + \lambda^6\{K(P_1.Person, P_2.Person) + K(P_1.PNP, P_2.PNP)\} \quad \rightarrow (l(\mathbf{i})=2, d(\mathbf{i})=3) \\
&\quad + \lambda^6\{K(P_1.Person, P_2.Person) + K(P_1.Verb, P_2.Verb) + K(P_1.PNP, P_2.PNP)\} \quad \rightarrow (l(\mathbf{i})=3, d(\mathbf{i})=3) \\
&= \lambda^2\{0+1+K(P_1.PNP, P_2.PNP)\} + \lambda^4\{0+2+K(P_1.PNP, P_2.PNP)\} \\
&\quad + \lambda^6 K(P_1.PNP, P_2.PNP) + \lambda^6\{1+K(P_1.PNP, P_2.PNP)\} \\
&= \lambda^2 + 2\lambda^4 + \lambda^6 + \{\lambda^2 + \lambda^4 + 2\lambda^6\} K(P_1.PNP, P_2.PNP)
\end{aligned}
\tag{6}
$$

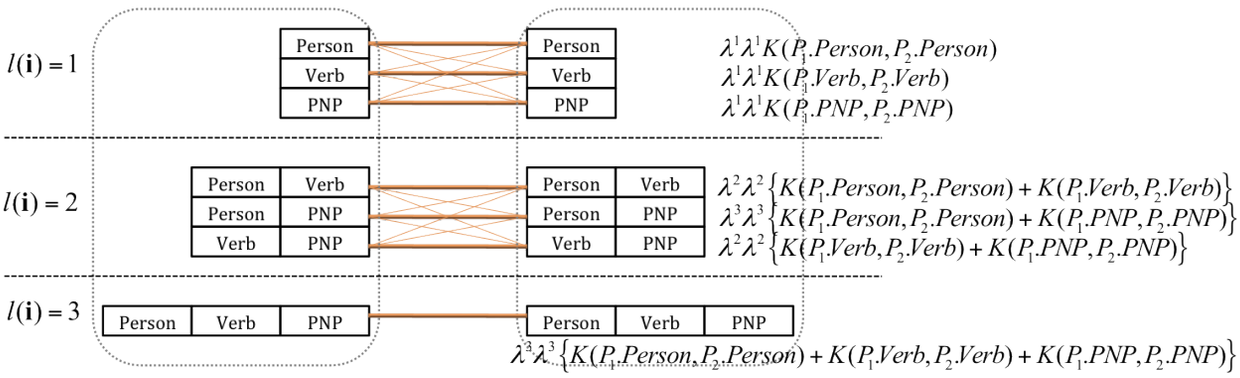

Figure 4. Executing Subsequence Similarity Kernel (SSK) function.

As Equation 6 shows, only the Equation expressed on the basis of is left, other than $K(P_1.PNP, P_2.PNP)$. The kernel function recursively compares child node subsequences at the third level to calculate resulting kernel similarity.

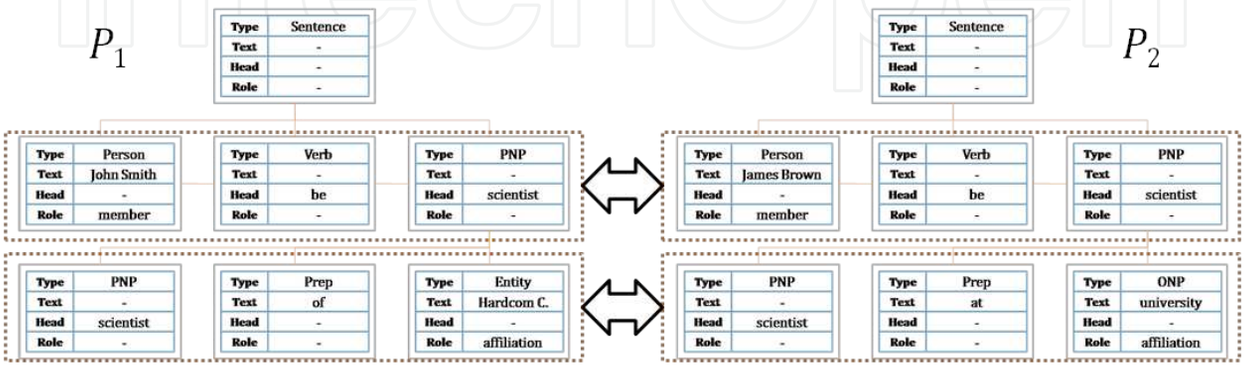

Figure 5. Process and method of calculating the tree kernel.

http://dx.doi.org/10.5772/51005

Figure 5 shows the process and method of calculating the kernel function. Basically, for the tree kernel, *Breadth First Search* is carried out. In calculating similarity between the two trees, trees to be compared are primarily those with the same node type and role. Since the kernel value is 0 when the text is different, nodes with the same text are substantially compared. In Figure 5, these are "*be*" and "*scientist*" nodes.

Zelenco, et al., (2003) divides tree kernel calculation into two types. One is the *sparse subtree kernel* described above, and the other one is the *continuous subtree kernel*, which is discussed below. First, the sparse subtree kernel includes two node subsequences for comparison, although two node subsequences are not continuously connected. For example, "*Person*, *PNP*" on the left side and "*Person*, *PNP*" on the right side in the middle of Figure 4 are node sequences, separated in the parse tree. The sparse subtree kernel includes such subsequences for comparison. On the contrary, the continuous subtree kernel does not approve such subsequences and excludes them from comparison. The Figure 6 below shows additional exemplary sentence for comparison and describing the effect of two tree kernels.

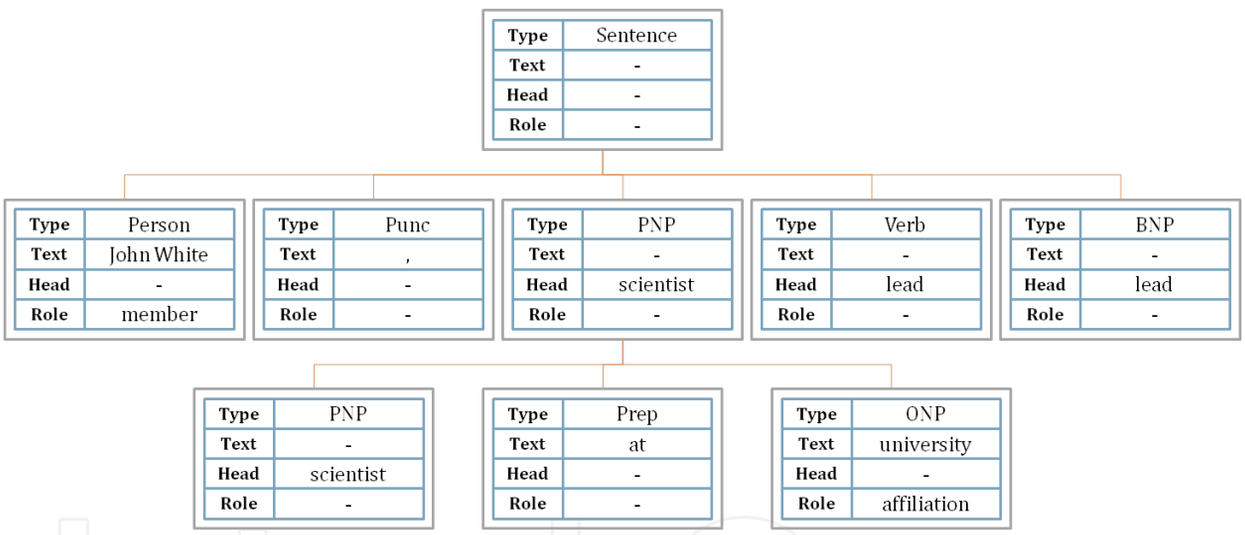

Figure 6. Additional sample sentence and parsing result.

Figure 6 shows the parsing result for "*John White, a well-known scientist at the University of Illinois, led the discussion*." Unlike the sentences discussed above, this one has an independent inserted phrase in apposition, and comprises the same contents as the second sentence of Figure 3 "*James Brown is a scientist at the University of Illinois*." If these two sentences are compared by means of the continuous subtree kernel, a very low kernel similarity will be derived because there is almost no continuous node on the parse tree although they include similar contents. The sparse subtree kernel is used overcome this deficiency. Figure 7 shows a part of the process of calculating kernel values for two sentences.

Figure 7 shows the process of calculating *K*([*Person*, *Verb*, *PNP*], [*Person*, *Punc*, *PNP*, *Verb*, *BNP*]) by means of sparse subtree kernel. When continuous subtree kernel is used, the similarity be-

tween the two sentences is very low. A better similarity value is revealed by two pairs of matching subsequences in the subsequence of which the length is 2, as shown in Figure 7.

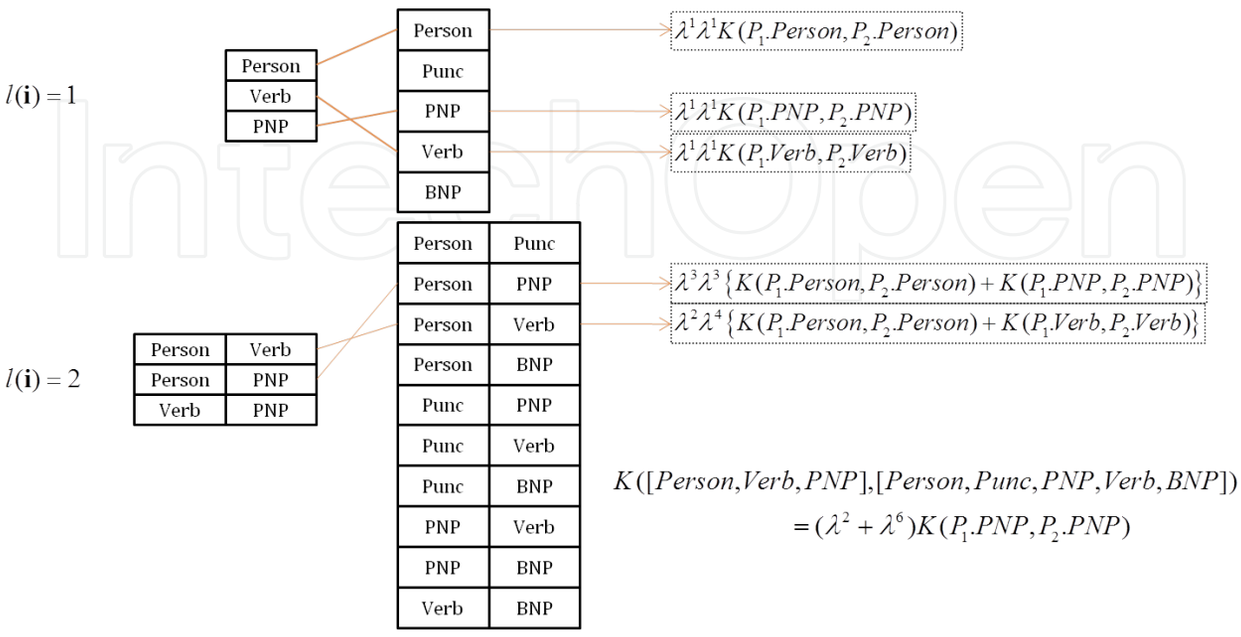

Figure 7. Process of calculating K([Person, Verb, PNP], [Person, Punc, PNP, Verb, BNP]).

For measuring the performance of the proposed two tree kernels, (Zelenco et al., 2003) used 60% of data manually constituted as a learning collection, and carried out 10-fold cross validation. Only two relations, that is, "*Person-Affiliation*" and "*Organization-Location*", were tested. The test revealed that the kernel-based method offers better performance than the feature-based method, and the continuous subtree kernel excels the sparse subtree kernel. In particular, the tree kernel proposed in their study inspired many new tree kernel researchers.

Their study is generally recognized as an important contribution for devising kernels for efficient measuring of similarity between very complex tree-type structure entities to be used later for relation extraction. Since various information other than the syntactic information is still required, the method highly depends on the performance of the REES system for creating parse trees. Because the quantity of data was not enough for the test and the binary classification test for only two types of relation was carried out, scalability or generality of the proposed kernel was not analyzed in detail.

4.2. Dependency tree kernel-based method (Culotta & Sorensen, 2004)

As the ACE collection was constituted and distributed from 2004, relation extraction has been fully studied. Culotta and Sorensen (2004) proposed a kernel-based relation extraction method, which uses the dependency parse tree structure on the basis of the tree kernel proposed by Zelenco, et al., (2003) described in section 4.1. This special parse tree, called an *Augmented Dependency Tree*, uses MXPOST (Ratnaparkhi, 1996) to carry out parsing, and then modifies some syntactic rules on the basis of the result. Exemplary rules include "*sub-*

http://dx.doi.org/10.5772/51005

jects are dependent on verbs", "*adjectives*" are dependent on nouns they describe, etc. In order to improve analysis result, however, this study uses even more complex node features than (Zelenco et al., 2003). The hypernym extracted from WordNet (Fellbaum, 1998) is applied to expand the matching function between nodes. The composite kernel is constructed to apply it to relation extraction for the first time.

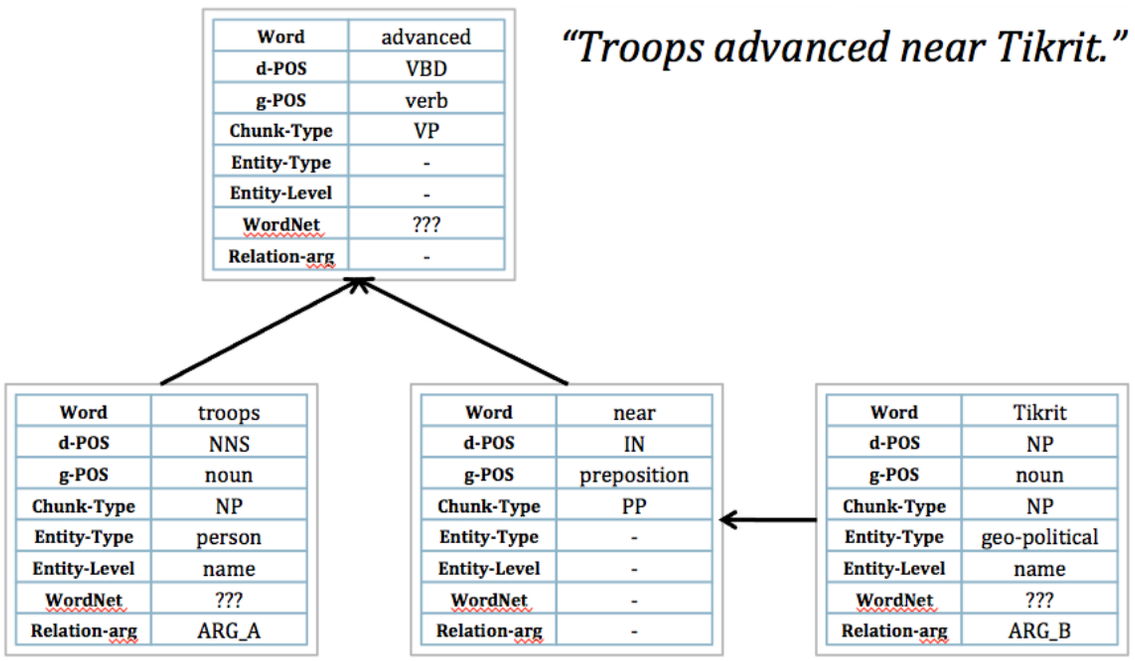

Figure 8. Sample augmented dependency tree ("*Troops advanced near Tikrit*").

Figure 8 shows a sample augmented dependency tree used in this study. The root of the tree is the verb "*advanced*", and the resultant subject and the preposition are the child nodes of the root. The object of the preposition "*Tikrit*" is the child node of the preposition "*near*". Each node has 8 types of node feature information. The Table below outlines each node feature.

Feature	Example
Words	troops, Tikrit
Detailed POS (24)	NN, NNP
General POS (5)	Noun, Verb, Adjective
Chunking Information	NP, VP, ADJP
Entity Type	person, geo-political-entity
Entity Level	name, nominal, pronoun
WordNet hypernyms	social group, city
Relation Argument	ARG_A, ARG_B

Table 1. Node features in the augmented dependency tree.

In Table 1, the first four features (words, part-of-speech information, and phrase information) are the information obtained from parsing, and the rest are named entity features from the ACE collection. Among them, the WordNet hypernym is the result of extracting the highest node for corresponding word from the WordNet database.

As discussed above, the tree kernel defined by (Zelenco et al., 2003) is used in this method. Since the features of each node are added, the matching function (Equaiton 1) and the similarity function (Equation 2) defined by Zelenco, et al., (2003) are accordingly modified into and applied. In detail, the features to be applied to the matching function and the features to be applied to the similarity function from among 8 features were dynamically divided to devise the following models to be applied.

$$
\begin{aligned}
&t_i : \text{feature vector representing the node i.}\\
&t_j : \text{feature vector representing the node j.}\\
&t_i^m : \text{subset of } t_i \text{ used in matching function}\\
&t_i^s : \text{subset of } t_i \text{ used in similarity function}\\
&m(t_i, t_j) = \begin{cases} 1 & \text{if } t_i^m = t_j^m \\ 0 & \text{otherwise} \end{cases}\\
&s(t_i, t_j) = \sum_{v_q \in t_i^s} \sum_{v_r \in t_j^s} C(v_q, v_r)
\end{aligned}
\tag{7}
$$

Where, m is the matching function; s is the similarity function; and t_i is a feature collection showing the node i. $C(\blacksquare, \blacksquare)$ is a function for comparing two feature values on the basis of approximate matching, not simple perfect matching. For example, recognition of "*NN*" and "*NP*" in the particular part-of-speech information of Table 1 as the same part-of-speeches is implemented by modifying the internal rule of the function. Equations 3 and 4 in section 4.1 are applied as tree kernel functions for comparing the similarity of two augmented dependency trees on the basis of two basic functions.

For the evaluation, the initial ACE collection version (2002) released in 2003 was used. This collection defines 5 entity types and 24 types of relations. Culotta & Sorensen (2004) tested relation extraction only for the higher 5 types of relation collections ("*AT*", "*NEAR*", "*PART*", "*ROLE*", "*SOCIAL*"). The tested kernels were the sparse subtree kernel (K_0), the continuous subtree kernel (K_1), and the bag-of-words kernel (K_2). In addition, two composite kernels for which the tree kernel was combined with the bag-of-word kernel, that is $K_3 = K_0 + K_2$, $K_4 = K_1 + K_2$, were further constituted. The test consisting of two steps of relation detection[4] and relation classification[5] revealed that all tree kernel methods, including the composite kernel, show better performance than the bag-of-words kernel. Unlike the evaluation result by (Zelenco et al., 2003), although the performance of continuous subtree kernel

4 Binary classification for identifying possible relation between two named entities.

5 Relation extraction for all instances with relations in the result of relation identification.

is higher, the reason for that was not clearly described, and the advantage of using the dependency tree instead of the full parse tree was not demonstrated in the experiment.

4.3. Shortest path dependency tree kernel method (Bunescu & Mooney, 2005)

In section 4.2, we have discussed relation extraction using dependency parse trees for the tree kernel proposed by Zelenco, et al., (2003). Bunescu & Mooney (2005) have studied the dependency path between the two named entities in the dependency parse tree with a view to proposing the shortest possible path dependency kernel for relation extraction. There is always a dependency path between two named entities in a sentence and Bunescu & Mooney (2005) argued that the performance of relation extraction is improved by using the syntactic paths. The Figure 9 below shows the dependency graph for a sample sentence.

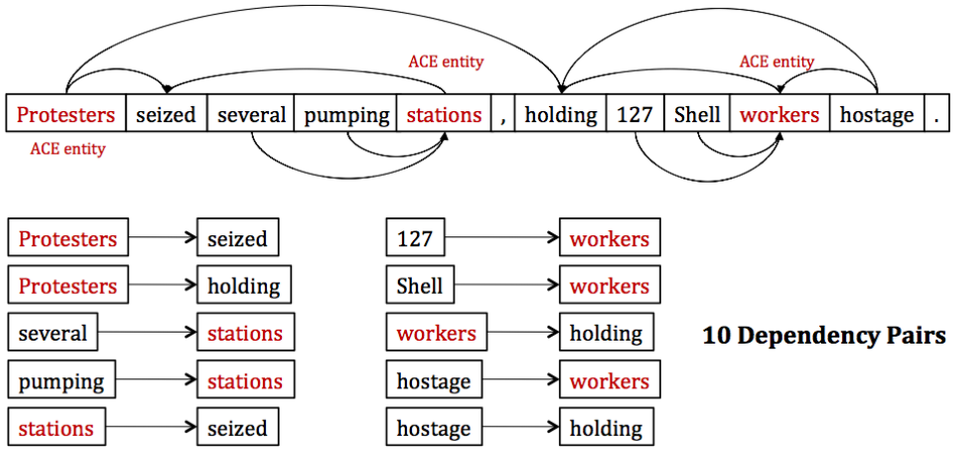

Figure 9. Dependency graph and dependency syntax pair list for the sample sentence.

The red node in Figure 9 represents the named entity specified in the ACE collection. Separation of the entire dependency graph results in 10 dependency syntax pairs. It is possible to select pairs, which include named entities in the syntax pairs to construct the dependency path as shown in Figure 10.

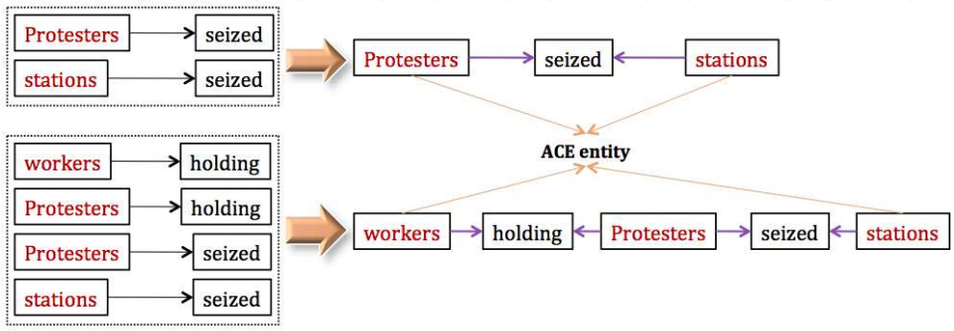

Figure 10. Extracting dependency path including named entities from dependency syntax pair collection.

As one can see from Figure 10, it is possible to construct the dependency path between the named entities, "*Protesters*" and "*stations*," and the dependency path between "*workers*" and "*stations*". As discussed above, the dependency path for connecting two named entities in this sentence can be extended infinitely. Bunescu & Mooney (2005) estimated that the shortest path among them contributes the most to establishing the relation between two entities. Therefore, it is possible to use kernel-based learning for estimating the relation between the two named entities connected by means of dependency path. For example, for estimating the relation for the path of "*protesters* ▪ *seized* ▪ *stations*", the relation is estimated that the PERSON entity ("*protesters*") did a specific behavior ("*seized*") for the FACILITY entity ("*stations*"), through which PERSON ("*protesters*") is located at FACILITY ("*stations*") ("LOCATED_AT"). At another complex path of "*workers* ▪ *holding* ▪ *protesters* ▪ *seized* ▪ *stations*", it is possible to estimate the relation that PERSON ("*workers*") is located at FACILITY ("*stations*") ("LOCATED_AT") if PERSON ("*protesters*") did some behavior ("*holding*") to PERSON ("*workers*"), and PERSON ("*protesters*") did some behavior ("*seized*") to FACILITY ("*stations*"). As such, with the dependency relation path, it is possible to identify semantic relation between two entities more intuitively.

For learning, Bunescu & Mooney (2005) have extracted the shortest dependency paths, including two entities from individual training instances as shown in the Table below.

Relations	Relation Instances
LOCATED_AT	**protesters** ▪ seized ▪ **stations**
LOCATED_AT	**workers** ▪ holding ▪ **protesters** ▪ seized ▪ **stations**
LOCATED_AT	**detainees** ▪ abusing ▪ **Jelisic** ▪ created ▪ at ▪ **camp**

Table 2. Shortest path dependency tree-based sample relation extraction instance.

As shown in Table 2, each relation instance is expressed as a dependency path whose both ends are named entities. In terms of learning, however, it is not easy to extract sufficient features from such instances. Therefore, as discussed in section 4.2, various supplementary information is created, such as part-of-speech, entity type and WordNet synset. As a result, individual nodes which make up the dependency path comprise a plurality of information elements, and a variety of new paths are finally created as shown in Figure 11.

$$\begin{bmatrix} \text{protesters} \\ \text{NNS} \\ \text{Noun} \\ \text{Person} \end{bmatrix} \times [\rightarrow] \times \begin{bmatrix} \text{seized} \\ \text{VBD} \\ \text{Verb} \end{bmatrix} \times [\leftarrow] \times \begin{bmatrix} \text{stations} \\ \text{NNS} \\ \text{Noun} \\ \text{Facility} \end{bmatrix}$$

Figure 11. New dependency path information created in a single instance.

As shown in Figure 11, with more information available for individual nodes, new 48 dependency paths can be created through Cartesian product of the node values. Here, relation extraction is carried out by applying the dependency path kernel for calculating redundancy of the information included in each node rather than comparing all newly created paths, as shown below.

$$\begin{aligned} &\boldsymbol{x} = x_1 x_2 \cdots x_m, \quad \boldsymbol{y} = y_1 y_2 \cdots y_n \\ &K(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 0, & m \neq n \\ \prod_{i=1}^{n} c(x_i, y_i) & m = n \end{cases} \\ &c(x_i, y_i) = |x_i \cap y_i| \end{aligned} \tag{8}$$

In the above Equation 8, x and y represent extended individual instances; m and n denote the lengths of the dependency path; $K(\blacksquare, \blacksquare)$ presents the dependency path kernel; and $c(\blacksquare, \blacksquare)$ is a function for calculating the level of information element redundancy between the two nodes. The Figure below shows the process of calculating the kernel value on the basis of Equation 8.

- '***his*** *actions in* ***Brcko***' ➔ **his** → actions ← in ← **Brcko**
- '***his*** *arrival in* ***Beijing***' ➔ **his** → arrival ← in ← **Beijing**

his PRP Person	→	actions NNS Noun	←	in IN	←	Brcko NNP Noun Location	
his PRP Person	→	arrival NN Noun	←	in IN	←	Beijing NNP Noun Location	
3	1	1	1	2	1	3	18

Figure 12. Calculating dependency path kernel.

As shown in Figure 12, the process of comparing two dependency paths is very simple. If the length of two paths is different, the kernel function simply returns zero (0). Otherwise, the level of information redundancy is then calculated for each node with respect to two paths. Since all the corresponding values are identical in the first node ("his", "PRP" and "Person"), the output is set to 3. As one matches in the second node, 1 is returned. By exponentiating all the calculated values, the kernel value is found to be 18.

On the basis of the same test environment as the collection used by Culotta & Sorensen (2004), two parsing systems, the CCG parser (Hockenmaier & Steedman, 2002) and the CFG parser (Collins, 1997) have been used to construct the shortest dependency path. The test included K_4 (bag-of-words kernel + continuous subtree kernel) that demonstrated the best performance by Culotta & Sorensen (2004) for comparing performance. The test revealed

that the CFG-based shortest dependency path kernel offers better performance by using the CFG parser than the CCG parser for the same kernel.

With regard to relation extraction, the shortest dependency path information is considered to be very useful, and is highly likely to be used in various fields. However, the kernel structure is too simple. Yet another limitation is that only the paths of the same length are included in calculating similarity of two dependency paths.

4.4. Subsequence kernel-based method (Bunescu & Mooney, 2006)

The tree kernel presented by Zelenco, et al., (2003) is to compare two sibling nodes basically at the same level and uses the subsequence kernel. Bunescu & Mooney (2006) introduced the subsequence kernel and attempted relation extraction only with the base phrase analysis (chunking), without applying the syntactic structure. Since kernel input is not of a complex syntactic structure, but base phrase sequences, the assumption was that the feature space can be divided into 3 types to comprise maximum 4 words for each type of features as follows, by using the advantage of easy selection of contextual information essential for relation extraction.

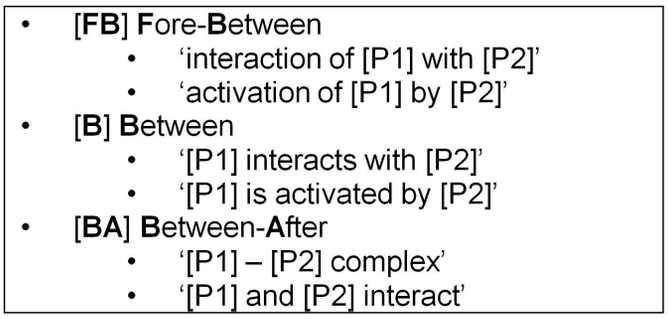

Figure 13. Contextual location information for feature extraction.

In Figure 13, [FB] represents the words positioned before and between the two entities; [B] means only the word between them; and [BA], accordingly, means word collections between and after. The 3 types of feature collections can accept individual relation expressions, respectively. Furthermore, various types of supplementary word information (part-of-speech, entity type, WordNet synset, etc.) are used to expand them as in the methods described above.

Zelenco et al., (2003) described how to calculate the subsequence kernel, which will be described in detail again later. The kernel calculation function $K_n(s, t)$ is defined as shown below based on all n-length subsequences included in two sequences s, t.

$$K_n(s, t, \lambda) = \sum_{i:|i|=n} \sum_{j:|j|=n} \lambda^{l(i)+l(j)} \prod_{k=1}^{n} c(s_{i_k}, t_{j_k}) \tag{9}$$

Where, i and j represent subsequences contained in s, t respectively; $c(\blacksquare, \blacksquare)$ is a function for deciding the homogeneity of the two inputs; and λ is a weight given to matching subsequences. l(i) and l(j) are the values indicating how far each relevant subsequence is posi-

tioned from the entire sequences. In order to calculate the weighted length, equation 9 selects among s and t only n-length subsequences, which exist in both sequences. For easy description, the following two sentences and base phrase analysis result will be used to explain the process of calculating the kernel value.

Figure 14. Two sentences and base phrase analysis result to illustrate the process of calculating subsequence kernel.

As shown in Figure 14, each of the nodes that consist of analysis result has 8 types of lexical information (word, part-of-speech, base phrase type, entity type, etc.) The kernel value, $K_3(s, t, 0.5)$, of two analysis sequences is calculated according to the process shown in Figure 15, with features of the subsequence of which the length is 3.

There are three subsequence pairs which are decided that the node of all subsequence is at least 0 by means of the homogeneity decision function $c(\blacksquare, \blacksquare)$, among the subsequences in s and t. Scores for each of matching subsequences are derived by calculating the cumulative factorial of $c(\blacksquare, \blacksquare)$ for each of them and then multiplying it by the weight. For example, the similarity of 0.84375 is obtained for "*troops advanced near*" and "*forces moved toward*". On the contrary, the similarity of "*troops advanced …Tikrit*" and "*forces moved …Bagdhad*" is 0.21093. This results from the lowered weight because the two subsequences are positioned apart. At last, the similarities of the subsequences are introduced to Equation 8 that gives 1.477.

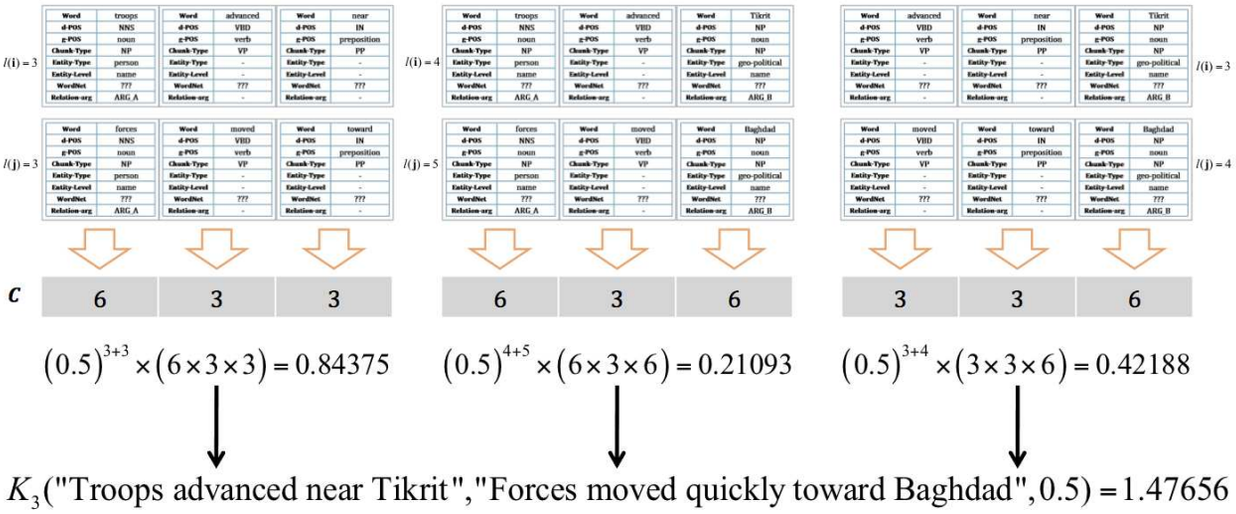

Figure 15. Process of calculating $K_3(s, t, 0.5)$.

As described above, it is possible to construct the subsequence kernel function based on the subsequences of all lengths by using contextual location information and Equation 8 as described above. Figure 16 shows surrounding contextual location where named entities occur in the sentence.

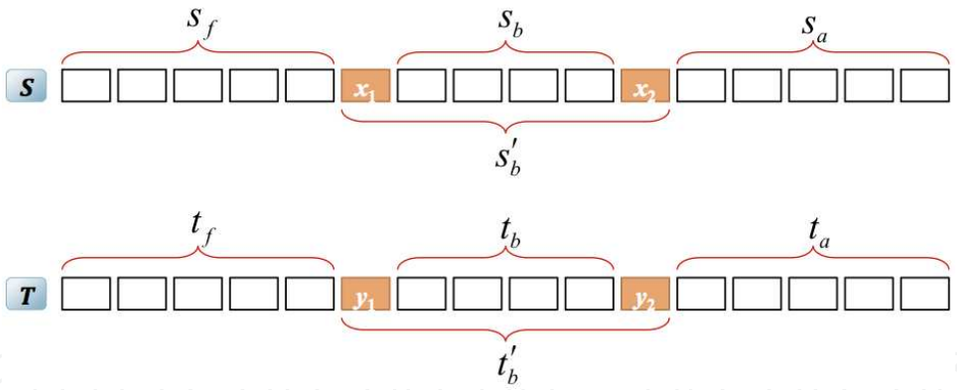

Figure 16. Specifying contextual information depending on named entity location and defining variables.

$$
\begin{aligned}
K(s,\ t) &= K_{fb}(s,\ t) + K_b(s,\ t) + K_{ba}(s,\ t) \\
K_{b,i}(s,\ t) &= K_i(s_b,\ t_b,\ 1) \cdot c(x_1,\ y_1) \cdot c(x_2,\ y_2) \cdot \lambda^{l(s'_b)+l(t'_b)} \\
K_{fb}(s,\ t) &= \sum_{i\geq 1,\ j\geq 1,\ i+j\, fb_{\max}} K_{b,i}(s,\ t) \cdot K'_j(s_f,\ t_f) \\
K_b(s,\ t) &= \sum_{1\leq i\leq b_{\max}} K_{b,i}(s,\ t) \\
K_{ba}(s,\ t) &= \sum_{i\geq 1,\ j\geq 1,\ i+j\, ba_{\max}} K_{b,i}(s,\ t) \cdot K'_j(\bar{s}_f,\ \bar{t}_f)
\end{aligned}
\tag{10}
$$

In Figure 16, x_i and y_i represent named entities; s_f and t_f denote the word lists before named entities; and s_a and t_a presents contextual word collections after two entities. s_b' and t_b' represent contextual information including two entities. Thus, the subsequence kernel for two sequences s and t is defined with the following Equation 10.

The subsequence kernel $K(s, t)$ consists of the sum of the contextual kernel before and between entity pairs, K_{fb}, the intermediate contextual kernel K_b of the entities, and the contextual kernel between and after entity pairs K_{ba}. fb_{max} is the length of the target context "Fore-Between" and b_{max} is the length of "Between" context. Also, ba_{max} is the target length of the context "Between-After" as seen in the figure 13. $\bar{s}$ and $\bar{t}$ are the reverse versions of strings of s and t respectively. The definition of the individual contextual kernel is described from the third to the fifth line of Equation 10. Here, K'_n is the same as K_n, with the exception that it specifies the length of the relevant subsequence from the location where the subsequence starts to the end of the entire sequence, and is defined as follows.

$$K'_n(s, t, \lambda) = \sum_{\boldsymbol{i}:|\boldsymbol{i}|=n} \sum_{\boldsymbol{j}:|\boldsymbol{j}|=n} \lambda^{|s|+|t|-i_1-j_1+2} \prod_{k=1}^{n} c(s_{i_k}, t_{j_k}) \tag{11}$$

In Equation 11, i_1 and j_1 represent the starting positions of subsequences i and j respectively. The individual contextual kernel calculates the similarity between the two sequences for the subsequences in the location divided on the basis of the locations specified in Figure 16, with Equation 10, and totalizes the kernel values to calculate resulting kernel values.

The performance evaluation in the same test environment as used in Sections 4.2 and 4.3 shows increased performance, even without complicated pre-processing, such as parsing, and without any syntactic information. In conclusion, the evaluation shows that this method is very fast in terms of learning speed and is an approach with a variety of potentials for improving performance.

4.5. Composite kernel-based method (Zhang, Zhang, Su, et al., 2006)

The release of ACE 2003 and 2004 versions contributed to full-scale study on relation extraction. In particular, the collection is characterized by even richer information for tagged entities. For example, ACE 2003 provides various entity features, e.g., entity headwords, entity type, and entity subtype for a specific named entity, and the features have been used as an important clue for determining the relation between two entities in a specific sentence. In this context, Zhang, Zhang, Su, et al., (2006) have built a composite kernel for which the convolution parse tree kernel proposed by Collins & Duffy, (2001) is combined with the entity feature kernel. Equation 11 gives the entity kernel definition.

$$\begin{aligned} K_L(R_1, R_2) &= \textstyle\sum_{i=1,2} K_E(R_1.E_i, R_2.E_i) \\ K_E(E_1, E_2) &= \textstyle\sum_i C(E_1.f_i, E_2.f_i) \\ C(f_1, f_2) &= \begin{cases} 1, & \text{if } f_1 = f_2 \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{12}$$

In Equation 12, R_i represents the relation instance; and $R_i.E_j$ are the j-th entity of R_i. $E_i.f_j$ represents the j-th entity feature of entity E_i; and C(▪ , ▪) is a homogeneity function for the two entities. It is possible to calculate the entity kernel K_L by summation on the basis of feature redundancy decision kernel K_E for a pair of entities.

Second, the convolution parse tree kernel expresses one parse tree as an occurrence frequency vector of a subtree as follows so as to measure the similarity between the two parse trees.

$$\phi(T)=\left(\#subtree_1(T),\ ...,\ \#subtree_i(T),\ ...,\ \#subtree_n(T)\right) \tag{13}$$

In Equation 13, $\#subtree_i(T)$ represents the occurrence frequency of the i-th subtree. All parse trees are expressed with the vector as described above, and the kernel function is calculated as the inner product of two vectors as follows.

$$K(T_1,\ T_2)=\langle\phi(T_1),\ \phi(T_2)\rangle \tag{14}$$

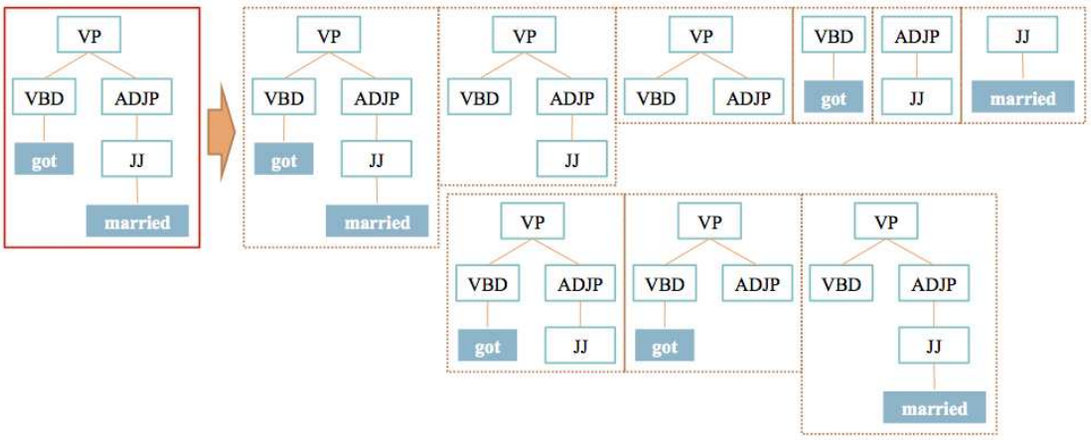

Figure 17. Parsing tree and its subtree collection.

Figure 17 shows all subtrees of a specific parse tree. There are nine subtrees in the figure altogether, and each subtree is an axis of the vector, which expresses the left side parse tree. If the number of all unified parse trees that can be extracted for *N* parse trees is *M*, each of extracted subtrees can be expressed as an *M*-dimension vector.

As shown in Figure 17, there are two constraints for a subtree of a specific parse tree. First, the number of nodes of the subtree must be at least 2, and the subtree should comply with production rules used by syntactic parser to generate parse trees of sentences (Collins & Duffy, 2001). For example, [VP ▪ VBD ▪ "*got*"] cannot become a subtree.

It is necessary to investigate all subtrees in the tree T, and calculate their frequency so as to build the vector for each of parse trees. This process is quite inefficient, however. Since we need only to compute the similarity of two parse trees in the kernel-based method, we can come up with indirect kernel functions without building the subtree vector for each parse

http://dx.doi.org/10.5772/51005

tree as with Equation 13. The following Equation 15, proposed by Collins & Duffy (2001), is used to calculate efficiently the similarity of two parse trees

$$
\begin{aligned}
&K(T_1, T_2)=\langle \phi(T_1), \phi(T_2)\rangle \\
&=\sum_i \#subtree_i(T_1)\cdot \#subtree_i(T_2) \\
&=\sum_i \left(\sum_{n_1\in N_1} I_{subtree_i}(n_1)\right)\cdot\left(\sum_{n_2\in N_2} I_{subtree_i}(n_2)\right) \\
&=\sum_{n_1\in N_1}\sum_{n_2\in N_2}\Delta(n_1, n_2) \\
&N_1, N_2 \rightarrow \text{the set of nodes in trees } T_1 \text{ and } T_2. \\
&I_{subtree_i}(n)=\begin{cases}1 & \text{if } ROOT(subtree_i)=n\\ 0 & \text{otherwise}\end{cases} \\
&\Delta(n_1, n_2)=\sum_i I_{subtree_i}(n_1)\cdot I_{subtree_i}(n_2)
\end{aligned}
\tag{15}
$$

In Equation 15, T_i represents a specific parse tree, and N_i represents all node collections of the parse tree. $I_{st}(n)$ is the function for checking whether the node n is the root node of the specific subtree st. The most time-consuming calculation in Equation 15 falls on calculating $\varnothing(n_1, n_2)$. To enhance this, Collins & Duffy (2001) came up with the following algorithm.

1. If CFP Production rules of n_1 and n_2 are different each other,
$$\Delta(n_1, n_2) = 0$$
2. If both n_1 and n_2 are pre-terminals (POS-tag),
$$\Delta(n_1, n_2) = 1\times\lambda$$
3. Otherwise,
$$\Delta(n_1, n_2) = \lambda\prod_{j=1}^{nc(n_1)}\left(1+\Delta\left(ch(n_1, j), ch(n_2, j)\right)\right)$$
$nc(n_1)\rightarrow$ the child number of n_1,
$ch(n, j)\rightarrow j^{th}$ child of node n
$\lambda\rightarrow$ decay factor

Figure 18. Algorithm for calculating $\varnothing(n_1, n_2)$

The function $\varnothing(n_1, n_2)$ defined in (3) of Figure 18 compares the child nodes of the input node to calculate the frequency of subtrees contained in both parse trees and the product thereof, until the end conditions defined in (1) and (2) are satisfied. In this case, the decay factor , which is a variable for limiting large subtrees so as to address the issue that larger subtrees among the subtrees of the parse tree comprise another subtrees therein, can be applied repeatedly for calculating the inner product of the subtree vector.

Two kernels built as described above, that is, the entity kernel and the convolution parse tree kernel, are combined in the following two manners.

$$K_1(R_1, R_2)=\alpha \cdot \frac{K_L(R_1, R_2)}{\sqrt{K_L(R_1, R_1) \cdot K_L(R_2, R_2)}} + (1-\alpha)\frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1) \cdot K(T_2, T_2)}} \quad (16\text{-}1)$$

$$K_2(R_1, R_2)=\alpha \cdot \left(\frac{K_L(R_1, R_2)}{\sqrt{K_L(R_1, R_1) \cdot K_L(R_2, R_2)}} + 1\right)^2 + (1-\alpha)\frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1) \cdot K(T_2, T_2)}} \quad (16\text{-}2)$$

In the above equations 16-1 and 16-2, K_L represents the entity kernel and K stands for the convolution parse tree kernel. Equation 16-1 shows the composite kernel being a linear combination of the two kernels, and Equation 16-2 defines the composite kernel constructed using quadratic polynomial combination.

Furthermore, Zhang, Zhang, Su, et al., (2006) proposed the method for pruning relation instance by leaving a part of the parse tree and removing the rest, so as to improve similarity measurement performance of the kernel function, and to exclude unnecessary contextual information in learning.

Tree Pruning Methods	Details
Minimum Complete Tree (MCT)	Minimum complete sub-tree encompassing two entities
Path-enclosed Tree (PT)	Sub-tree belong to the shortest path in between two entities
Chunking Tree(CT)	Sub-tree generated by discarding all the internal nodes excpet nodes for base phrases and POS from PT
Context-sensitive PT(CPT)	Sub-tree generated by adding two additional terminal nodes outside PT
Context-sensitive CT(CCT)	Sub-tree generated by adding two additional terminal nodes outside CT
Flattened PT(FPT)	Sub-tree generated by discarding all the nodes having only one paraent and child node from PT
Flattened CPT(FCPT)	Sub-tree generated by discarding all the nodes having only one paraent and child node from CT

Table 3. Relation instance pruning (Zhang, Zhang, Su, et al., 2006; Zhang et al., 2008).

For the evaluation, Zhang, Zhang, Su, et al., (2006) used both ACE 2003 and ACE 2004. They parsed all available relation instances with Charniak's Parser (Charniak, 2001), and on the basis of the parsing result carried out instance conversion using the method described in Table 3. To this end, Moschitti (2004) has developed a kernel tool, while SVMLight (Joachims, 1998) was used for learning and classification.

The test shows that the composite kernel features better performance than a single syntactic kernel. The combination of quadratic polynomial type shows performance between the two kernels. This means that flat feature (entity type feature) and structural feature (syntactic feature) can be organically combined as a single kernel function. In consideration that the Path-enclosed Tree method shows the best performance among all relation instance pruning methods, it is possible to achieve the effect only with core related syntactic information, so as to estimate the relation of two entities in a specific sentence.

4.6. Other recent studies

Choi, et al., (2009) have constructed and tested a composite kernel where various lexical and contextual features are added by expanding the existing composite kernel. In addition to the syntactic feature, called flat feature, they extended the combination range of lexical feature from the entity feature to the contextual feature in order to achieve high performance. Mintz, Bills, Snow, and Jurafsky (2009) proposed a new method of using Freebase, which is a semantic database for thousands relation collections, to gather exemplary sentences for a specific relation and making relation extraction on the basis of the obtained exemplary sentences. In the test, the collection of 10,000 instances corresponding to 102 types of relations has shown the accuracy of 67.6%. In addition, T.-V. T. Nguyen, Moschitti, and Riccardi (2009) have designed a new kernel, which extends the existing convolution parse tree kernel, and Reichartz, et al., (2009) proposed a method, which extends the dependency tree kernel. As described above, most studies published so far are based on the kernels described in Sections 4.1to 4.5.

5. Comparison and analysis

In the previous section, five types of kernel-based relation extraction have been analyzed in detail. Here, we discuss the results of comparison and analysis of these methods. Section 5.1 will briefly describe the criteria for comparison and analysis of the methods. Section 5.2 compares characteristics of the methods. Section 5.3 covers performance results in detail. Section 5.4 sums up the advantages and disadvantages of each of the method.

5.1. Criteria for comparison and analysis

Generally, a large variety of criteria can be used for comparing kernel-based relation extraction methods. The following 6 criteria, however, have been selected and used in this study. First, (1) linguistic analysis and pre-processing method means the pre-processing analysis methods and types for individual instances which are composed of learning collections and evaluation collections, e.g., the type of parsing method or the parsing system used. (2) The level of linguistic analysis, which is the criterion related to the method (1), is a reference to what level the linguistic analysis will be carried out in pre-processing and analyzing instances. Exemplary levels include part-of-speech tagging, base phrase analysis, dependency parsing or full parsing. In addition, (3) the method of selecting a feature space is a reference for deciding if the substantial input of the kernel function is an entire sentence or a part

thereof. Also, (4) the applied lexical and supplementary feature information means various supplementary feature information used for addressing the issue of sparse data. (5) The relation extraction method is a practical relation extraction method based on learning models already constituted. Exemplary relation extraction methods include multi-class classification at a time and a single mode method of separating instances with relations from those without relations by means of processing multiple classifications at a time or binary classification, then to carry out relation classification only for the instances with relations. (6) The manual work requirement is a reference to decide if the entire process is fully automatically carried out or manual work is required only at some step. The aforementioned 6 criteria were used to analyze the kernel-based relation extraction methods and the result of the analysis is shown in the following Table 6.

In addition, for the purpose of describing the characteristics of the kernel function, the description in section 4 will be summarized, and each structure of factors to be input of the kernel function will be described. Modification of the kernel function for optimized speed will be included in the analysis criteria and described. For performance comparison of the individual methods, types and scale of the tested collections and tested relations will be analyzed and described in detail. The following Table 4 describes the ACE collection generally used among the tested collections for relation extraction developed so far.

Items	ACE-2002	ACE-2003	ACE-2004
# training documents	422	674	451
# training relation instances	6,156	9,683	5,702
# test documents	97	97	N/A
# test relation instances	1,490	1,386	N/A
# entity types	5	5	7
# major relation types	5	5	7
# relation sub-types	24	24	23

Table 4. Description of ACE Collection.

As shown in Table 4, the ACE collection is generally used and can be divided into 3 types. ACE-2002, however, is not widely used because of consistency and quality problems. There are 5 to 7 types of entities, e.g., Person, Organization, Facility, Location, Geo-Political Entity, etc. For relations, all collections are structured to be at two levels, consisting of 23 to 24 types of particular relations corresponding to 5 types of Role, Part, Located, Near, and Social. As the method of constructing those collections is advanced and the quality thereof is improved, the tendency is that the scale of training instances is reduced. Although subsequent collections have already been constituted, they are not publicized according to the principle of non-disclosure, which is the policy of ACE Workshop. This should be improved for active studies.

5.2. Comparison of characteristics

Table 5 summarizes each of kernel functions with respect to the concept, before analyzing the kernel-based relation extraction method, in conformity to 6 types of comparison and analysis criteria described in 5.1.

Kernel types	Description of concept
Tree Kernel (TK)	Compares each node which consists of two trees to be compared.
Dependency Tree Kernel (DTK)	Based on BFS, applies subsequence kernel to the child nodes located at the same level. The decay factor is adjusted and applied to the similarity depending on the length of subsequences at the same level or on the level itself.
Shortest Path Dependency Kernel (SPDK)	Compares each element of the two paths to cumulatively calculate the common values for the elements in order to compute the similarity by multiplying all values. Similarity is 0 if the length of two paths is different.
Subsequence Kernel (SK)	For the example of measuring similarity of two words, extracts only the subsequences which exist in both words, and expresses the two words with a vector (Φ(x)) by using the subsequences, among all of the subsequences which belong to two words and of which the length is n. Afterwards, obtains the inner product of the two vectors to calculate the similarity. Generalizes and uses SSK to compare planar information of the tree kernel (sibling node).
Composite Kernel (CK)	Finds all subtrees in the typical CFG-type syntactic tree, and establishes them as a coordinate axis to represent the parse tree as a vector (Φ(x)). In this case, the following constraints hold true: (1) the number of nodes must be at least 2, and (2) the subtree should comply with the CFG creation rule. Since there can be multiple subtrees, each coordinate value can be at least 1, and similarity is calculated by obtaining the inner product of the two vectors created as such.

Table 5. Summary of kernel-based relation extraction methods.

Table 6 shows comparison and analysis of the kernel-based relation extraction methods. As it is closer to the right side, the method is more recent one. The characteristic found in all of the above methods is that various feature information in addition to the syntactic information is used as well. Such heterogeneous information was first combined and used in a single kernel function, but is separated from the composite kernel and applied.

With respect to selecting the feature space, most of sentences or a part of the parse tree are applied other than the tree kernel. Manual work was initially required for extracting relation instances and building the parse tree. The recently developed methods, however, offer full

automation. In the relation extraction methods, multi-class classification is used, in which the case with no relation is included as one relation.

	TK	DTK	SPDK	SK	CK
Language Processor	Shallow Parserx (REES)	Statistical Parserx (MXPOST)	Statistical Parser (Collins' Parser)	Chunker (OpenNLP)	Statistical Parser (Charniak's Parser)
Level of Language Processing	PLO Tagging Parsing	Parsing	Parsing	Chunking	Parsing
Feature Selection Methods	Extracts features from parse trees manually	Selects small sub-tree including two entities from the entire dependency tree	Selects the shortest dependency path which starts with one entity and ends with the other one	Before Entities After Entities Between Entities	MCT PT CPT FPT FCPT
Features used in Kernel Computation	Entity Headword Entity Role Entity Text	Word POS Chunking Info. Entity Type Entity Level WordNet Hypernym Relation Parameter	Word POS Chunking Info. Entity Type	Word POS Chunking Info. Entity Type Chunk Headword	Entity Headword Entity Type Mention Type LDC mention Type Chunking Info.
Relation Extraction Methods	Single Phase (Multiclass SVM)	Cascade Phase (Relation Detection and Classification)	Single Phase Cascade Phase	Single Phase (Multiclass SVM)	Single Phase (Multiclass SVM)
Manual Process	Necessary (Instance Extraction)	Necessary (Dependency Tree)	Necessary (Dependency Path)	N/A	N/A

Table 6. Comparison of characteristics of kernel-based relation extraction.

5.3. Comparison of performance

Table 7 shows the parameter type of each kernel function and computation complexity in calculating the similarity of two inputs. Most of them show complexity of $O(N^2)$, but SPDK exceptionally demonstrates the complexity of the order of O(N) and can be considered as the most efficient kernel.

Kernels	Parameter Structure	Time Complexity
TK	Shallow parse trees	CSTK : $O(\|N_{i,1}\|*\|N_{i,2}\|)$
DTK	Dependency trees	SSTK : $O(\|N_{i,1}\|*\|N_{i,2}\|^3)$ $\|N_1\|$: #(1st input's nodes in level i) $\|N_2\|$: #(2nd input's nodes in level i)
SPDK	Dependency paths	$O(\|N_1\|)$ $\|N_1\|$: #(1st input's nodes)
SK	Chunking results	$O(n*\|N_1\|*\|N_2\|)$ n : subsequence length $\|N_1\|$: #(1st input's nodes) $\|N_2\|$: #(2nd input's nodes)
CK	Full parse trees	$O(\|N_1\| *\|N_2\|)$ $\|N_1\|$: #(1st input's nodes) $\|N_2\|$: #(2nd input's nodes)

Table 7. Parameter structure and calculation complexity of each kernel.

It should be noted that the complexity shown in Table 7 is just kernel calculation complexity. The overall complexity of relation extraction can be much higher when processing time for parsing and learning is also considered.

Articles	Year	Methods	Test Collection	F1
(Zelenco et al., 2003)	2003	TK	200 News Articles (2-relations)	**85.0**
(Culotta & Sorensen, 2004)	2004	DTK	ACE-2002 (5-major relations)	**45.8**
(Kambhatla, 2004)	2004	ME	ACE-2003 (24-relation sub-types)	**52.8**
(Bunescu & Mooney, 2005)	2005	SPDK	ACE-2002 (5-major relations)	**52.5**
(Zhou et al., 2005)	2005	SVM	ACE-2003 (5-major relations)	**68.0**
			ACE-2003 (24-relation sub-types)	**55.5**
(Zhao & Grishman, 2005)	2005	CK	ACE-2004 (7-major relations)	**70.4**
(Bunescu & Mooney, 2006)	2006	SK	ACE-2002 (5-major relations)	**47.7**
(Zhang, Zhang, Su, et al., 2006)	2006	CK	ACE-2003 (5-major relations)	**70.9**
			ACE-2003 (24-relation sub-types)	**57.2**
			ACE-2004 (7-major relations)	**72.1**
			ACE-2004 (23-relation sub-types)	**63.6**
(Zhou et al., 2007)	2007	CK	ACE-2003 (5-major relations)	**74.1**
			ACE-2003 (24-relation sub-types)	**59.6**
			ACE-2004 (7-major relations)	**75.8**
			ACE-2004 (23-relation sub-types)	**66.0**
(Jiang & Zhai, 2007)	2007	ME/SVM	ACE-2004 (7-major relations)	**72.9**

Table 8. Comparison of performance of each model of kernel-based relation extraction.

Articles and Approaches		Test Collection																				
		200 News			ACE-2002						ACE-2003						ACE-2004					
Relation Sets		PLO-2			Main-5			Sub-24			Main-5			Sub-24			Main-7			Sub-23		
Precision/Recall/F-measure		P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
(Zelenco et al., 2003)	TK	91.6	79.5	85.0																		
(Culotta & Sorensen, 2004)	DTK				67.1	35.0	45.8															
(Kambhatla, 2004)	ME													63.5	45.2	52.8						
(Bunescu & Mooney, 2005)	SPDK				65.5	43.8	52.5															
(Zhou et al., 2005)	SVM										77.2	60.7	68.0	63.1	49.5	55.5						
(Zhao & Grishman, 2005)	CK																69.2	70.5	70.4			
(Bunescu & Mooney, 2006)	SK							73.9	35.2	47.7												
(Zhang, Zhang, Su, et al., 2006)	CK										77.3	65.6	70.9	64.9	51.2	57.2	76.1	68.4	72.1	68.6	59.3	63.6
(Zhou et al., 2007)	CK										80.8	68.4	74.1	65.2	54.9	59.6	82.2	70.2	75.8	70.3	62.2	66.0
(Jiang & Zhai, 2007)	ME/ SVM																74.6	71.3	72.9			

Table 9. Comparison of performance of each kernel-based relation extraction method.

http://dx.doi.org/10.5772/51005

ACE-2002, which is the first version of ACE collection, had the issue with data consistency. In the subsequent versions this problem has been continuously addressed, and finally resolved in version ACE-2003. Starting from 52.8% achieved by Kambhatla (2004) on the basis of the performance of ACE-2003 with respect to 24 relation collections, the performance was improved up to 59.6% recently announced by Zhou, et al., (2007). Similarly, the maximum relation extraction performance for 23 particular relations on the ACE-2004 collection is currently 66%.

Although each model has different performance in differently sized relation collections, the composite kernel generally shows better results. In particular, Zhou, et al., (2007) have demonstrated high performance for all collections or relation collections based on extended models initially proposed by Zhang, Zhang, Su, et al., (2006). As described above, it is considered that various features for relation extraction, that is, the syntactic structure and the vocabulary, can be efficiently combined in a composite kernel for better performance.

Although the described research results do not represent all studies on relation extraction, there are many parts not evaluated yet although a lot of study results have been derived so far as seen in Table 9. It is necessary to carry out evaluation on the basis of various collections for comprehensive performance evaluation of a specific relation extraction model, but this is a challenge that more studies should be done. In particular, the key issue is to check whether the performance of relation extraction is achieved as high as described in the above without the characteristics of ACE collections, in that they provide supplementary information (entity type information) of a considerable scale for relation extraction.

5.4. Comparison and analysis of advantages and disadvantages of each method

In Section 5.4, advantages and disadvantages of five kernel-based relation extraction methods are discussed and outlined in Table 10.

Method	Advantage	Disadvantage
Feature-based SVM/ME	Applies the typical automatic sentence classification without modification. Performance can be further improved by applying various feature information. Relatively high speed	A lot of effort is required for feature extraction and selection. Performance can be improved only through feature combination.
Tree Kernel (TK)	Calculates particular similarity between shallow parse trees. Uses both structural (parenthood) and planar information (brotherhood). Optimization for speed improvement.	Very limited use of structural information (syntactic relations) Slow similarity calculation speed in spite of optimized speed
Dependency TK (DK)	Addressed the issue of insufficient use of structural information, which is a disadvantage of TK.	Predicates and key words in a dependency tree are emphasized only by means of decay factors (low emphasis capability)

Method	Advantage	Disadvantage
	Uses key words which are the core of relation expression in the sentence, as feature information, on the basis of the structure that the predicate node is raised to a higher status, which is a structural characteristic of a dependency tree.	Slow similarity calculation speed
Shortest Path DTK (SPTK)	Creates a path between two named entities by means of dependency relation to reduce noise not related to relation expression. Shows very fast computation speed because the kernel input is not trees, but paths, different from previous inputs. Adds various types of supplementary feature information to improve the performance of similarity measurement, thanks to the simple structure of paths.	Too simple structure of the kernel function Too strong constraints because the similarity is 0 if the length of two input paths is different.
Subsequence Kernel (SK)	Very efficient because syntactic analysis information is not used. Adds various supplementary feature information to improve the performance.	Can include many unnecessary features
Composite Kernel (CK)	Makes all of constituent subtrees of a parse tree have a feature, to perfectly use structure information in calculating similarity. Optimized for improved speed	Comparison is carried out only on the basis of sentence component information of each node (phrase info.) (kernel calculation is required on the basis of composite feature information with reference to word class, semantic info, etc.)

Table 10. Analysis of advantages and disadvantages of kernel-based relation extraction.

As one can see in Table 10, each method has some advantages and disadvantages. A lot of efforts are required for the process of feature selection in general feature-based relation extraction. The kernel-based method does not have this disadvantage, but has various limitations instead. For example, although the shortest dependency path kernel includes a variety of potentials, it showed low performance due to the overly simple structure of the kernel function. Since the composite kernel constitutes and compares subtree features only on the basis of part-of-speech information and vocabulary information of each node, generality of similarity measurement is not high. A scheme to get over this is to use word classes or semantic information.

A scheme can be suggested for designing a new kernel in order to overcome the above shortcomings. For example, a scheme may be used for interworking various supplementary feature

information (WordNet Synset, thesaurus, ontology, part-of-speech tag, thematic role information, etc.), so as to ensure general comparison between subtrees in the composite kernel. The performance can be improved by replacing the current simple linear kernel with the subsequence or another composite kernel and applying all sorts of supplementary feature information in order to address the shortcomings of the shortest path dependency kernel.

6. Conclusion

In this chapter, we analyzed kernel-based relation extraction method, which is considered the most efficient approach so far. Previous case studies did not fully covered specific operation principles of the kernel-based relation extraction models, just cited contents of individual studies or made an analysis in a limited range. This chapter, however, closely examines operation principles and individual characteristics of five kernel-based relation extraction methods, starting from the original kernel-based relation extraction studies (Zelenco et al., 2003), to composite kernel (Choi et al., 2009; Zhang, Zhang, Su, et al., 2006), which is considered the most advanced kernel-based method. The overall performance of each method was compared using ACE collections, and particular advantages and disadvantages of each method were summarized. This study will contribute to researchers' kernel study for relation extraction of higher performance and to general kernel studies of high level for linguistic processing and text mining.

Author details

Hanmin Jung*, Sung-Pil Choi, Seungwoo Lee and Sa-Kwang Song

*Address all correspondence to: jhm@kisti.re.kr

Korea Institute of Science and Technology Information, Korea

References

[1] ACE. (2009). *Automatic Content Extraction*, Retrieved from, http://www.itl.nist.gov/iad/mig//tests/ace/.

[2] Agichtein, E., & Gravano, L. (2000). Snowball: extracting relations from large plain-text collections. *Proceedings of the fifth ACM conference on Digital libraries*, 85-94, New York, NY, USA, ACM, doi:10.1145/336597.336644.

[3] Bach, N., & Badaskar, S. (2007). A Survey on Relation Extraction. *Literature review for Language and Statistics II*.

[4] Brin, S. (1999). Extracting Patterns and Relations from the World Wide Web. Lecture Notes in Computer Science, , 1590, 172-183.

[5] Bunescu, R., & Mooney, R. J. (2005). A Shortest Path Dependency Kernel for Relation Extraction. *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 724-731.

[6] Bunescu, R., & Mooney, R. J. (2006). Subsequence Kernels for Relation Extraction. *Proceeding of the Ninth Conference on Natural Language Learning (CoNLL-2005)*, Ann Arbor, MI, Retrieved from, http://www.cs.utexas.edu/users/ai-lab/pub-view.php?PubID=51413.

[7] Charniak, E. (2001). Immediate-head Parsing for Language Models. *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*.

[8] Choi, S.-P., Jeong, C.-H., Choi, Y.-S., & Myaeng, S.-H. (2009). Relation Extraction based on Extended Composite Kernel using Flat Lexical Features. *Journal of KIISE : Software and Applications*, 36(8).

[9] Collins, M. (1997). Three Generative, Lexicalised Models for Statistical Parsing. Madrid. *Proceedings of the 35th Annual Meeting of the ACL (jointly with the 8th Conference of the EACL)*.

[10] Collins, M., & Duffy, N. (2001). Convolution Kernels for Natural Language. *NIPS-2001*.

[11] Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, 20(3), 273-297, Hingham, MA, USA, Kluwer Academic Publishers, doi:10.1023/A:1022627411411.

[12] Cristianini, N., & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press.

[13] Culotta, A., & Sorensen, J. (2004). Dependency Tree Kernels for Relation Extraction. *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*.

[14] Etzioni, O., Cafarella, M., Downey, D., Popescu, A., , M., Shaked, T., Soderland, S., Weld, D. S., et al. (2005). Unsupervised named-entity extraction from the Web: An experimental study. *Artificial Intelligence*, 165(1), 91-134, Retrieved from, http://www.sciencedirect.com/science/article/pii/S0004370205000366.

[15] Fellbaum, C. (1998). WordNet: An Electronic Lexical Database. MIT Press Cambridge, MA

[16] Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine learning*, 37(3), 277, 296, Retrieved from, http://www.springerlink.com/index/q3003163876k7h81.pdf.

[17] Fundel, K., Küffner, R., & Zimmer, R. (2007). RelEx-Relation extraction using dependency parse trees. *Bioinformatics*, 23(3), 365-371.

[18] Gartner, T., Flach, P., & Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. *Learning Theory and Kernel Machines*, 129-143.

[19] Hockenmaier, J., & Steedman, M. (2002). Generative Models for Statistical Parsing with Combinatory Categorial Grammar. Philadelphia, PA. *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*.

[20] Jiang, J., & Zhai, C. (2007). A Systematic Exploration of the Feature Space for Relation Extraction. *NAACL HLT*.

[21] Joachims, T. (1998). Text Categorization with Support Vecor Machine: learning with many relevant features. *ECML-1998*.

[22] Kambhatla, N. (2004). Combining lexical, syntactic and semantic features with Maximum Entropy models for extracting relations. *ACL-2004*.

[23] Li, J., Zhang, Z., Li, X., & Chen, H. (2008). Kernel-based learning for biomedical relation extraction. *Journal of the American Society for Information Science and Technology*, 59(5), 756-769, Wiley Online Library.

[24] Li, W., Zhang, P., Wei, F., Hou, Y., & Lu, Q. (2008). A novel feature-based approach to Chinese entity relation extraction. *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, 89-92.

[25] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, 2, 419-444.

[26] MUC. (2001). *The NIST MUC Website*, Retrieved from, http://www.itl.nist.gov/iaui/894.02/related_projects/muc/.

[27] Mintz, M., Bills, S., Snow, R., & Jurafsky, D. (2009). Distant supervision for relation extraction without labeled data. *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, 2, 1003-1011, Stroudsburg, PA, USA, Association for Computational Linguistics, Retrieved from, http://dl.acm.org/citation.cfm?id=1690219.1690287.

[28] Moncecchi, G., Minel, J. L., & Wonsever, D. (2010). A survey of kernel methods for relation extraction. *Workshop on NLP and Web-based technologies (IBERAMIA 2010)*.

[29] Moschitti, A. (2004). A Study on Convolution Kernels for Shallow Semantic Parsing. *ACL-2004*.

[30] Nguyen, D. P. T., Matsuo, Y., & Ishizuka, M. (2007). Exploiting syntactic and semantic information for relation extraction from wikipedia. *IJCAI Workshop on Text-Mining & Link-Analysis (TextLink 2007)*.

[31] Nguyen, T.-V. T., Moschitti, A., & Riccardi, G. (2009). Convolution kernels on constituent, dependency and sequential structures for relation extraction. *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 3, 1378-1387.

[32] Ratnaparkhi, A. (1996). A Maximum Entropy Part-Of-Speech Tagger. *Proceedings of the Empirical Methods in Natural Language Processing Conference*, Retrieved from, http://onlinelibrary.wiley.com/doi/10.1002/cbdv.200490137/abstract.

[33] Reichartz, F., Korte, H., & Paass, G. (2009). Dependency tree kernels for relation extraction from natural language text. *Machine Learning and Knowledge Discovery in Databases*, 270-285, Springer.

[34] Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6), 386-408.

[35] TAC. (2012). *Text Analysis Conference*, Retrieved from, http://www.nist.gov/tac/.

[36] Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, 189-196, Morristown, NJ, USA, Association for Computational Linguistics, doi:10.3115/981658.981684.

[37] Yates, A., Cafarella, M., Banko, M., Etzioni, O., Broadhead, M., & Soderland, S. (2007). TextRunner: open information extraction on the web. Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations Stroudsburg, PA, USA: Association for Computational Linguistics Retrieved from http://dl.acm.org/citation.cfm?id=1614164.1614177 , 25-26.

[38] Zelenco, D., Aone, C., & Richardella, A. (2003). Kernel Methods for Relation Extraction. *Journal of Machine Leanring Research*, 3, 1083-1106.

[39] Zhang, M., Zhang, J., & Su, J. (2006). Exploring syntactic features for relation extraction using a convolution tree kernel. *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, 288-295, Stroudsburg, PA, USA, Association for Computational Linguistics, doi:10.3115/1220835.1220872.

[40] Zhang, M., Zhang, J., Su, J., & Zhou, G. (2006). A Composite Kernel to Extract Relations between Entities with both Flat and Structured Features. *21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, 825-832.

[41] Zhang, M., Zhou, G., & Aiti, A. (2008). Exploring syntactic structured features over parse trees for relation extraction using kernel methods. *Information processing & management*, 44(2), 687-701, Elsevier.

[42] Zhao, S., & Grishman, R. (2005). Extracting Relations with Integrated Information Using Kernel Methods. *ACL-2005*.

[43] Zhou, G., Su, J., Zhang, J., & Zhang, M. (2005). Exploring Various Knowledge in Relation Extraction. *ACL-2005*.

[44] Zhou, G., Zhang, M., Ji, D., & Zhu, Q. (2007). Tree Kernel-based Relation Extraction with Context-Sensitive Structured Parse Tree Information. *The 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 728-736.