A Review of Relation Extraction

Nguyen Bach
Language Technologies Institute
School of Computer Science
Canergie Mellon University
Pittsburgh, PA 15213
nbach@cs.cmu.edu

Sameer Badaskar
Language Technologies Institute
School of Computer Science
Canergie Mellon University
Pittsburgh, PA 15213
sbadaska@cs.cmu.edu

Abstract

Many applications in information extraction, natural language understanding, information retrieval require an understanding of the semantic relations between entities. We present a comprehensive review of various aspects of the entity relation extraction task. Some of the most important supervised and semi-supervised classification approaches to the relation extraction task are covered in sufficient detail along with critical analyses. We also discuss extensions to higher-order relations. Evaluation methodologies for both supervised and semi-supervised methods are described along with pointers to the commonly used performance evaluation datasets. Finally, we also give short descriptions of two important applications of relation extraction, namely question answering and biotext mining.

1 Introduction

There exists a vast amount of unstructured electronic text on the Web, including newswire, blogs, email communications, governmental documents, chat logs, and so on. How could a human be helped to understand all of this data? A popular idea is turn unstructured text into structured by annotating semantic information. However the sheer volume and heterogeneity of data make human annotation impossible. Instead, we would like to have a computer annotate all data with the structure of our interest. Normally, we are interested in relations between entities, such as person, organization, and location. Examples of relations are person-affiliation and organization-location. Current state-of-the-art named entities recognizers (NER), such as (Bikel et al., 1999) and (Finkel et al., 2005), can automatically label data with high accuracy. However, the whole relation extraction process is not a trivial task. The computer needs to know how to recognize a piece of text having a semantic property of interest in order to make a correct annotation. Thus, extracting semantic relations between entities in natural language text is a crucial step towards natural language understanding applications. In this paper, we will focus on methods of recognizing relations between entities in unstructured text.

A relation is defined in the form of a tuple $t = (e_1, e_2, ..., e_n)$ where the e_i are entities in a predefined relation r within document D. Most relation extraction systems focus on extracting binary relations. Examples of binary relations include *located-in(CMU, Pittsburgh)*, *father-of(Manuel Blum, Avrim Blum)*. It is also possible to go to higher-order relations as well. For example, in the sentence "*At codons 12, the occurence of point mutations from G to T were observed*" exists a 4-ary biomedical relation. The biomedical relationship between a type of variation, its location, and the corresponding state change from an initial-state to an altered-state can be extracted as *point_mutation(codon, 12, G, T)*.

In this review, we begin with supervised approaches which formulate the relation extraction task as a binary classification problem. Further, we discuss feature based (Kambhatla, 2004) (Zhao & Grishman, 2005) and kernel based methods of supervised relation extraction. The major advantage of kernel methods is they offer efficient solutions that allow us to explore a large (often exponential, or in some cases, infinite) feature space in polynominal computational time, without the need to explicitly represent the features. We will discuss and compare kernels like - tree kernel (Zelenko et al., 2003), subsequence kernel (Bunescu & Mooney, 2005b), and dependency tree kernel (Bunescu & Mooney, 2005a).

More recently, semi-supervised and boostrapping approaches have gained special attention. In section 3, we will review DIPRE (Brin, 1998), and Snowball (Agichtein & Gravano, 2000) systems which only require a small set of tagged seed instances or a few hand-crafted extraction patterns per relation to launch the training process. They all use a semi-supervised approach similar to the Yarowsky's algorithm in word sense disambiguation (Yarowsky, 1995). Also, KnowItAll (Etzioni et al., 2005) and TextRunner(Banko et al., 2007) propose large scale relation extraction systems which have a self-trained binary relation classifier.

In section 4, we review a method to extract higher-order relations (McDonald et al., 2005). The novelty of (McDonald et al., 2005) is to factorize complex relations into binary relations which are represented as a graph, and an algorithm to reconstruct complex relations by making tuples from selected maximal cliques in the graph. The primary advantage of their approach is that it allows for the use of binary relation classifiers which have been well studied and are often accurate.

In section 5, we discuss evaluation methods and standard evaluation datasets such as ACE (NIST, 2007) and MUC (Grishman & Sundheim, 1996). Finally, we present applications of relation extraction technology on biomedical text (Liu et al., 2007) and question answering (Ravichandran & Hovy, 2002).

2 Supervised Methods

We first present the task of relation extraction as a classification task. For the sake of simplicity and clarity, we restrict our discussion to binary relations between two entities. N-ary relation extraction will be discussed in the succeeding sections. Given a sentence $S = w_1, w_2, .., e_1.., w_j, ..e_2, .., w_n$, where e_1 and e_2 are the entities, a mapping function $f(.)$ can be given as

$$f_R(T(S)) = \begin{cases} +1 & \text{If } e_1 \text{ and } e_2 \text{ are related according to relation } R \\ -1 & \text{Otherwise} \end{cases} \tag{1}$$

where $T(S)$ are *features* that are extracted from S. Essentially the mapping function $f(.)$ decides whether the entities in the sentence are in a relation or not. Put in another way, the task of entity-relation extraction becomes that of *entity-relation detection*. If a labeled set of positive and negative relation examples are available for training, the function $f(.)$ can be constructed as a discriminative classifier like Perceptron, Voted Perceptron or Support Vector Machines (SVMs). These classifiers can be trained using a set of features selected after performing textual analysis (like POS tagging, dependency parsing, etc) of the labeled sentences. On the other hand, input to the classifiers can also take the form of rich structural representations like parse trees. Depending on the nature of input to the classifier training, supervised approaches for relation extraction are further divided into (1) feature based methods and (2) kernel methods. We discuss both these methods in appropriate detail.

2.1 Feature based Methods

Given a set of positive and negative relation examples, syntactic and semantic features can be extracted from the text. These extracted features serve as cues for deciding whether the entities in a sentence are related or not. Syntactic features extracted from the sentence include (1) the entities themselves, (2) the types of the two entities, (3) word sequence between the entities, (4) number of words between the entities and (5) path in the parse tree containing the two entities. Semantic cues include the path between the two entities in the dependency parse. Both the semantic and syntactic features extracted are presented to the classifier in the form of a feature vector, for training or classification. (Kambhatla, 2004) trains a log-linear model using the features described, for the task of entity classification. On the other hand, (Zhao & Grishman, 2005) and (GuoDong et al., 2002) use SVMs trained on these features using polynomial and linear kernels respectively for classifying different types of entity relations. Some features are good indicators of entity relations while others are not and it is important to select only those features which are relevant to the task. Feature based methods involve heuristic choices and the features have to be selected on a trial-and-error basis in order to maximize performance. Since NLP applications in general and relation extraction in particular involve structured representations of the input data, it can be difficult to arrive at an optimal subset of relevant features. To remedy the problem of selecting a suitable feature-set, specialized kernels are designed for relation extraction in order to exploit rich representations of the input data like shallow parse trees etc. These kernels explore the input representations exhaustively in an implicit manner in a higher dimensional space. We now discuss some of the kernel methods for relation extraction in detail.

2.2 Kernel Methods

The kernels used for relation-extraction (or relation-detection) are based on string-kernels described in (Lodhi et al., 2002). String kernels have been discussed in (Lodhi et al., 2002) in the context of text classification. However, an understanding of the workings of string-kernels is essential for interpreting the kernels used for relation extraction. Given two strings x and y, the string-kernel computes their similarity based on the number of subsequences that are common to both of them. More the number of subsequences common, greater the similarity between the two strings. Each string can be mapped to a higher dimensional space where each dimension corresponds to the presence (weighted) or absence (0) of a particular subsequence. For example, a string $x =$ **cat** can be expressed in a higher dimensional space of subsequences as follows:

$$\begin{aligned}\phi(x = cat) &= [\phi_a(x) \;..\; \phi_c(x) \;..\; \phi_t(x) \;..\; \phi_{at}(x) \;..\; \phi_{ca}(x) \;..\; \phi_{ct}(x) \;..\; \phi_{cat}(x) \;..\,] \\ &= [\;\lambda \quad .. \quad \lambda \quad .. \quad \lambda \quad .. \quad \lambda^2 \quad .. \quad \lambda^2 \quad .. \quad \lambda^2 \quad .. \quad \lambda^3 \quad ..\,] \qquad (2)\end{aligned}$$

Where $\lambda \in (0, 1]$ is a decay factor such that longer and non-contiguous subsequences are penalized. Infact, $\phi_{ct}(cat)$ is penalized more (λ^3) than $\phi_{ca}(cat)$ (λ^2) and $\phi_{at}(cat)$ (λ^2) since **ct** occurs non-contiguously in **cat**. To generalize, let u be a subsequence present at indices $i = i_1, i_2, ..., i_{|u|}$ ($i_1 \leq i_2 \leq ... \leq i_{|u|}$) inside the string x ($u = x[i]$) and let $l(i) = i_{|u|} - i_1 + 1$ be the length of u. Since u can be present inside x in more than one way, the coordinate of the string x corresponding to u in the higher dimensional space is given by

$$\phi_u(x) = \sum_{i:u=x[i]} \lambda^{l(i)} \qquad (3)$$

If U is the set of all possible ordered subsequences present in strings x and y, the kernel similarity between x and y is given by

$$\begin{aligned}K(x, y) &= \phi(x)^T \phi(y) \\ &= \sum_{u \in U} \phi_u(x)^T \phi_u(y) \qquad (4)\end{aligned}$$

A straightforward computation of (4) using (3) involves exponential complexity. In practice however, the computation of equation (4) is performed implicitly in a efficient manner using a dynamic programming technique described in (Lodhi et al., 2002). The complexity of computation of the string-kernel given in (4) is $O(|x||y|^2)$ ($|x| \geq |y|$).

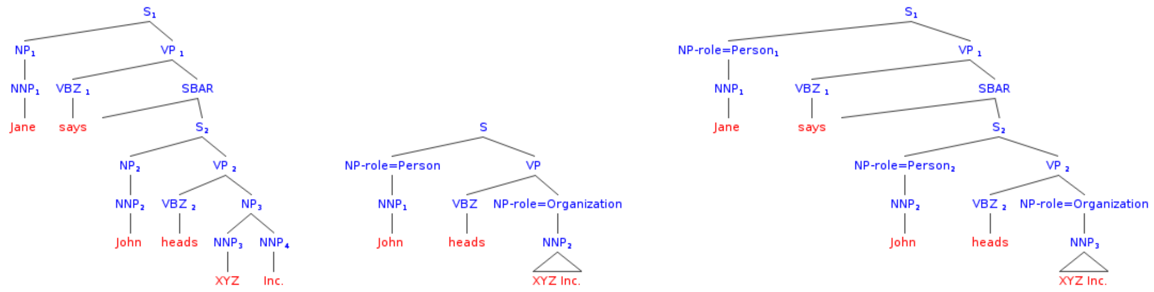
A more generalized interpretation of equation (4) is that if x and y are two objects, $K(x, y)$ computes the structural commonness between them and in doing so, computes their similarity. x and y could be objects like strings, sequences of words, parse trees etc. In relation extraction, if x^+ and x^- are objects representing positive and negative entity-relation examples respectively and if y is the test example, $K(x^+, y) > K(x^-, y)$ implies that y contains a relation or otherwise. In practice however, $K(x, y)$ is the similarity function used in classifiers like SVMs, Voted Perceptron etc. For the task of relation extraction, objects x^+, x^- and y can be represented as (1) word sequences around the entities under question or (2) parse trees containing the entities. Depending on the choice of representation, we discuss two major kernel approaches: Bag of features kernels and Tree kernels.

2.2.1 Bag of features Kernel

Consider the sentence "*The headquarters of* **Google** *are situated in* **Mountain View**". Given that **Google** and **Mountain View** are the named entities, the words **situated** and **headquarters** indicate an *organization-location* relationship between the two entities. Thus we can conclude that the context around the entities under question can be used to decide whether they are related or not. (Bunescu & Mooney, 2005b) use the word-context around the named entities for extracting protein interactions from the MEDLINE abstracts. A sentence $s = w_1, ..., e_1, ..., w_i, ..., e_2, ..., w_n$ containing related entities e_1 and e_2 can be described as $s = sb\ e_1\ sm\ e_2\ sa$. Here sb, sm and sa are portions of word-context *before*, *middle* and *after* the related entities respectively. Now given a test sentence t containing entities e'_1 and e'_2, the similarity of its *before*, *middle* and *after* portions with those of sentence s is computed using the sub-sequence kernel based on (4). The kernel computes similarity between two sequences at the word level rather than character level (as in string-kernels). In the paper of (Bunescu & Mooney, 2005b), three subkernels are defined, one each for matching the *before*, *middle* and *after* portions of the entities context and the combined kernel is simply the sum of all the subkernels. Using subsequence kernels in conjunction with SVMs improves both the precision and recall (see section 5 for evaluation metrics) of the relation extraction task (Bunescu & Mooney, 2005b) over a set of 225 MEDLINE abstracts when compared to a existing rule based system. Further, Bunescu et. al. also augment the words in the context with their respective part-of-speech (POS) tags, entity types etc to improve over the dependency tree kernel approach of (Culotta & Sorensen, 2004).

2.2.2 Tree Kernels

Compared to the previous approach, (Zelenko et al., 2003) replace the strings in the kernel of equation (4) with structured shallow parse trees built on the sentence. They use the tree kernels and plug them into SVM and Voted Perceptron for the task of extracting *person-affiliation* and *organization location* relations from text. Thus, the kernel is designed to compute the similarity between two entity-augmented shallow parse tree structures. Given a sentence, its shallow parse is constructed first. The rationale for using shallow parse trees instead of full parse trees is that they tend to be more robust and reliable. Apart from phrasal information, the shallow parser augments the tree with entity information. Now given a shallow parse, positive examples are constructed by enumerating the lowest subtrees which cover both the related entities. If a subtree does not cover the related entities, it is assigned a negative label.



(a) Parse Tree (b) Positive Relation Example (c) Negative Relation Example

Figure 1: These figures are only illustrative of the process and do not depict actual shallow-parse trees.

Figure. 1(a) gives an example of a parse tree for the sentence *Jane says John heads XYZ Inc*. The subtree rooted at S_2 covers both the related entities (*John* and *XYZ Inc*) and hence gets labeled positive as shown in Figure 1(b). On the other hand, 1(c) is a negative example since it is not the lowest subtree covering the related entities and the entity *Jane* is not in a relation with the other entities. Each node T in a positive or negative example tree has three attributes: entity role ($T.role$ ={person, organization, none}), chunk type ($T.chunk$ ={NP, VP, etc.}) and the text it covers ($T.text$). Given a set of positive and negative examples and a suitable kernel function, training can be done using a classifier like SVM, Voted perceptron etc.

The kernel function described in (Zelenko et al., 2003) is a modification of the kernel specified in equation (4). Given two shallow parse trees the kernel computes the structural commonality between them. This is analogous to the string-kernels which compute similarity between two strings in terms of the number of character subsequences common to them. In other words, the tree kernel computes a weighted sum of the number of subtrees that are common between the two shallow-parse trees. The kernel similarity is computed recursively as follows:

For two subtrees rooted at T_1 and T_2, $K(T_1, T_2)$ is given by

1. Match the attributes of nodes T_1 and T_2. If the attributes do not match, return a similarity of zero.
2. If the attributes match, add a score of 1 and compare child-sequences of T_1 and T_2. If $children(T_1)$ and $children(T_2)$ are the child-sequences of T_1 and T_2 respectively, their similarity is computed in terms of the number of child subsequences that are common between them. The process of computing child-sequence similarity is analogous to the computation of string-kernels given by equation (4).

If, m and n are the number of nodes in the shallow parse trees, the complexity of kernel computation is shown to be $O(mn^3)$. The kernels used in (Culotta & Sorensen, 2004) are very similar to (Zelenko et al., 2003). The unique feature of their work is that they use a dependency tree instead of shallow parse trees. Also, every node of the dependency tree also contains more information like word identity, POS tag, generalized-POS, phrase tag (NP, VP etc) entity type, entity level, relation argument etc. The argument of (Culotta & Sorensen, 2004) is that they use a richer structured representation which leads to significant performance gains when compared to a bag-of-words kernel.

To sum up, (Culotta & Sorensen, 2004) and (Zelenko et al., 2003) use rich structural information in the form of trees in order to obtain a decent performance in the relation extraction task. In contrast, (Bunescu & Mooney, 2005a) make an interesting observation that the shortest path between the two entities in a dependency parse encodes sufficient information to extract the relation between them. If e_1 and e_2 are two entities in a sentence and p their predicate, then the shortest path between e_1 and e_2 passes through p. This is because, e_1 and e_2 are the arguments of p. Given a sentence S, a dependency tree is first extracted. Then the shortest path between the entity pairs in a sentence are computed. Let one such path be $P = e_1 \rightarrow w_1 \rightarrow .. \rightarrow w_i \leftarrow .. \leftarrow w_n \leftarrow e_2$. Here, w_i are the words in the shortest path and arrows indicate the direction of dependency as extracted from the tree. Using P alone as a feature would lead to bad performance due to data sparsity. Therefore word-classes like POS are extracted from each of the words and the feature vectors is given by a Cartesian product as

$$x = [\genfrac{}{}{0pt}{}{e_1}{C(e_1)}] \times [\rightarrow] \times [\genfrac{}{}{0pt}{}{w_1}{C(w_1)}] .. \times [\rightarrow] \times [\genfrac{}{}{0pt}{}{w_i}{C(w_i)}] \times [\leftarrow] .. \times [\leftarrow] \times [\genfrac{}{}{0pt}{}{w_n}{C(w_n)}] \times [\leftarrow] \times [\genfrac{}{}{0pt}{}{e_2}{C(e_2)}]$$

where $C(w_i)$ are the classes of word w_i which can be the POS, generalized-POS, entity type etc. The kernel over this feature space is defined as

$$K(x, y) = \begin{cases} 0 & \text{If } |x| \neq |y| \\ \prod_{i=1}^{|x|} f(x_i, y_i) & \text{Otherwise} \end{cases} \tag{5}$$

where $f(x_i, y_i)$ is the number of word-classes common to x_i and y_i. The advantage of this kernel over the ones proposed by (Zelenko et al., 2003) and (Culotta & Sorensen, 2004) is that its computation requires linear time apart from having a simplified feature space. At the same time, the performance of the relation extraction is shown to be better than Culotta's tree based approach discussed before. In fact the recall of the shortest-path dependency kernel is found to be significantly better than Culotta's tree based kernel while having a comparable precision.

2.3 Discussion

The supervised relation extraction methods described so far have concentrated mainly on kernel methods. Starting with the definition of string kernel, we have described methods that use variations

Algorithm 1 Yarowsky's algorithm in general form (McDonald, 2004)

```
Input: a set of unlabeled data $D$ and a set of seed examples $S$
repeat
    Train a classifier $C$ on $S$
    Label $D$ using $C$
    $N$ = top $n$ labels that $C$ was highly confident
    $S = S \cup N$
    $D = D \backslash N$
until convergence criteria is reached
```

of the string kernels to extract relations. The discussion on kernels is organized in an increasing order of richness of the input and corresponding increase in the complexity of the kernel. While the bag-of-features kernel discussed in section 2.2.1 is relatively simple, the tree kernels of (Zelenko et al., 2003) and (Culotta & Sorensen, 2004) require richer representations of the input in the form of shallow-parse trees, dependency trees etc. In contrast to the kernel methods, the feature-based methods described in (Kambhatla, 2004) and (Zhao & Grishman, 2005) use a set of carefully extracted features (heuristics) for the relation extraction task. In problems related to NLP, it is often difficult to generate optimal feature representations from the input data and kernels offer a convenient solution by exploring the input implicitly in a much higher dimensional space. Thus, using kernels changes the problem from that of feature engineering to that of kernel design. While the tree kernels of (Zelenko et al., 2003) and (Culotta & Sorensen, 2004) use structured input and outperform feature based methods, they are computationally burdensome. On the other hand, the shortest dependency-path kernels described in (Bunescu & Mooney, 2005a) offer the advantage of both linear time kernel similarity computation and relatively better performance than (Culotta & Sorensen, 2004). With the exception of (Bunescu & Mooney, 2005a), none of the other papers make a performance comparison with each others work. In so far, the dependency-path kernels emerge as the winner among all the kernel methods.

In general, we observe that supervised methods have some limitations.

1. These methods are difficult to extend to new entity-relation types for want of labeled data.
2. Extensions to higher order entity relations are difficult as well.
3. They are relatively computationally burdensome and do not scale well with increasing amounts of input data.
4. We see that most of the methods described require pre-processed input data in the form of parse tree, dependency parse trees etc. Thus, the pre-processing stage is error prone and can hinder the performance of the system.

3 Semi-supervised Methods

To summarize so far, we have seen that there are advances on relation extraction using labeled data. How about semi-supervised/boostrapping relation extraction approach? Semi-supervised learning has become an important topic in computational linguistics. For many language-processing tasks including relation extraction, there is an abundance of unlabeled data, but labeled data is lacking and too expensive to create in large quantities, therefore making bootstrapping techniques desirable.

We focus on (Yarowsky, 1995) and (Blum & Mitchell, 1998) algorithms since the semi-supervised relation extraction methods we are reviewing use these algorithms. The main idea of both algorithms is to use the output of the weak learners as training data for the next iteration. Co-training (Blum & Mitchell, 1998) is a weakly supervised paradigm that learns a task from a small set of labeled data and a large pool of unlabeled data using separate, but redundant views of the data (i.e. using disjoint feature subsets to represent the data). To ensure provable performance guarantees, the co-training algorithm assumes as input a set of views that satisfies two fairly strict conditions. First, each view must be sufficient for learning the target concept. Second, the views must be conditionally

Algorithm 2 DIPRE (Brin, 1998)

```
1. Use the seed examples to label some data
2. Induce patterns from the labeled examples
3. Apply the patterns to data, to get a new set of author/title pairs
4. Return to step 2, and iterate until convergence criteria is reached
```

independent of each other given the class.

The general framework of the Yarowsky algorithm is shown in Algorithm 1. Yarowsky applies this algorithm to the word sense disambiguation task. Yarowsky algorithm in fact is a special case of the co-training algorithm. (Abney, 2004) presents a theoretical analysis for Yarowsky algorithm. As we explore the family of the technique, we will keep two primary questions in mind: 1) How to obtain seed automatically? and 2) What is a good seed?

3.1 DIPRE (Brin, 1998)

DIPRE (Dual Iterative Pattern Relation Expansion) is a relation extraction system proposed by (Brin, 1998). The relation of interest is $(author, book)$ from the Web. DIPRE starts with a small set of $(author, book)$ pairs which are also called seeds. Assume that our current seed set has only one seed (*Arthur Conan Doyle, The Adventures of Sherlock Holmes*). The system crawls the Internet to look for pages containing both instances of the seed. To learn patterns DIPRE uses a tuple of 6 elements $[order, author, book, prefix, suffix, middle]$ where $order$ is 1 if the author string occurs before the book string and 0 otherwise, $prefix$ and $suffix$ are strings contain the 10 characters occurring to the left/right of the match, $middle$ is the string occurring between the author and book.

For example, if the web crawler found pages which have

"*Read The Adventures of Sherlock Holmes by Arthur Conan Doyle online or in you email*"

"*know that Sir Arthur Conan Doyle wrote The Adventures of Sherlock Holmes, in 1892*"

"*When Sir Arthur Conan Doyle wrote the adventures of Sherlock Holmes in 1892 he was high ...*"

then the extracted tuples are

[*0, Arthur Conan Doyle, The Adventures of Sherlock Holmes, Read, online or, by*]

[*1, Arthur Conan Doyle, The Adventures of Sherlock Holmes, now that Sir, in 1892, wrote*]

[*1, Arthur Conan Doyle, The Adventures of Sherlock Holmes, When Sir, in 1892 he, wrote*]

After extracting all tuples, the system groups tuples by matching $order$ and $middle$. For each group of tuples, the longest common prefix of suffix strings and the longest common suffix of prefix strings are extracted, thus each group induces a pattern in the form

[*longest-common-suffix of prefix strings, author, middle, book, longest-common-prefix of suffix strings*].

For example the pattern would be [*Sir, Arthur Conan Doyle, wrote, The Adventures of Sherlock Holmes, in 1892*]. The next step is to generalize the pattern with a wild card expression, and the pattern will be [*Sir, .*?, wrote, .*?, in 1892*]. DIPRE uses this pattern to search the Web again, and extract relations, from instances it extracts a new relation (*Arthur Conan Doyle, The Speckled Band*). DIPRE adds the new relations to the seed set and repeat the procedure again until some stopping criteria, such as no new seed relations are extracted or the number of relations over some threshold. The general form of DIPRE algorithm is shown in algorithm 2.

How is DIPRE similar to Yarowsky algorithm? Both algorithms initialize with a seed example set. The classifier used by DIPRE is pattern matching which is trained iteratively by extracting patterns from seed relations. Given a string, if it matches to one of pattern then the string is classified as positive and used to extract new relations, otherwise negative. New relations are added to the seed set to retrain the classifier which means extract more patterns in DIPRE. DIPRE is an instance of application of Yarowsky algorithm to relation extraction.

3.2 Snowball (Agichtein & Gravano, 2000)

Snowball has similar system architecture as DIPRE. The task is to identify (*organization, location*) relation on regular text. Snowball also starts with a seed set of relations and attaches a confidence of 1 to them. The classifier in Snowball is a pattern matching system like DIPRE, however, Snowball does not use exact matching. Snowball represents each tuple as a vector and uses a similarity function to group tuples. Snowball extracts tuples in form $[\mathit{prefix}, \mathit{orgnization}, \mathit{middle}, \mathit{location}, \mathit{suffix}]$. Prefix, suffix, and middle are feature vectors of tokenized terms occurring in the pair. For example, if (*CMU, Pittsburgh*) is a known pair, then for the string "*... go to CMU campus in Pittsburgh to meet ...*" the system will extract

$$[(w_1, \text{go}), (w_2, \text{to}), \text{ORG}, (w_1, \text{campus}), (w_2, \text{in}), \text{LOC}, (w_1, \text{to}), (w_2, \text{meet})]$$

The limit on the prefix and suffix feature vectors is 2 in this case. Each w_i is a term weight which is computed by the normalized frequency of that term in a given position. For example, term weight of token meet in the suffix is

$$\text{weight}(\textit{meet}, \text{suffix}) = \frac{\text{frequency of } \textit{meet} \text{ in suffix}}{\text{number of all word in suffix}}$$

These term weights are updated in successive iterations as more tuples are added. To group tuples which have the same (organization, location) representation but differ in prefix, suffix, and middle Snowball introduce the similarity function

$$Match(tuple_i, tuple_j) = (\mathit{prefix}_i.\mathit{prefix}_j) + (\mathit{suffix}_i.\mathit{suffix}_j) + (middle_i.middle_j)$$

After grouping tuples into classes, Snowball induces a single tuple pattern P for each class which is a centroid vector tuple.

Each pattern P is assigned a confidence score which measure the quality of a newly-proposed pattern

$$Confidence(P) = \frac{P_{positive}}{P_{positive} + P_{negative}}$$

where $P_{positive}$ is the number of times the new pattern recovers an (organization, location) pair seen at a previous iteration of training; and $P_{negative}$ is number of times it recovers a pair where the organization has been seen with a different location at a previous iteration.

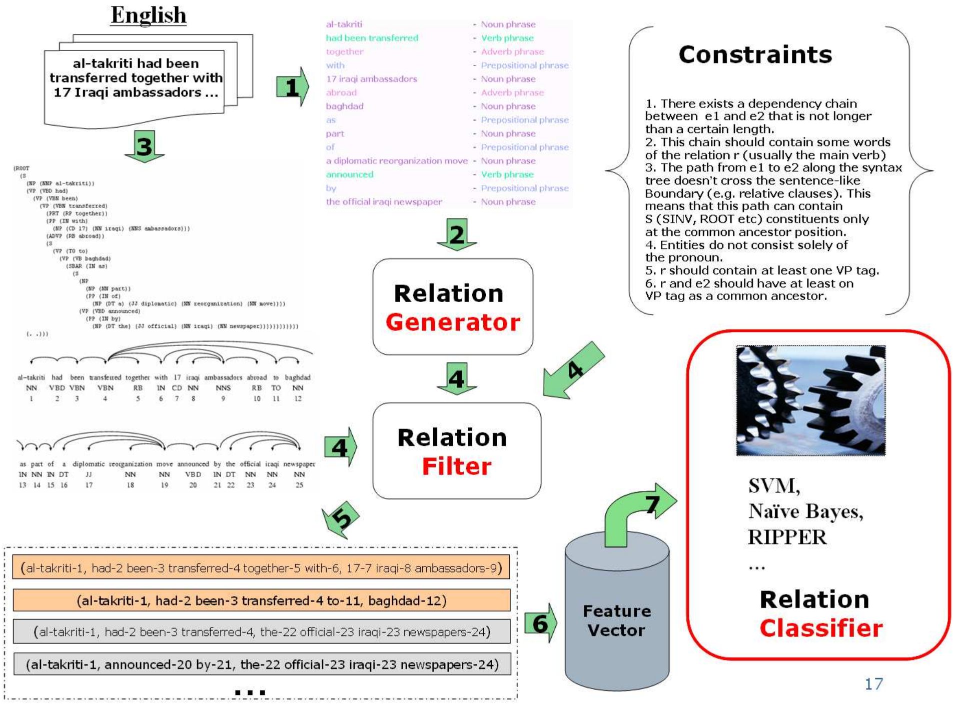
To label new data, Snowball first runs a named-entity recognizer over the data to identify all location and organization entities. Within a sentence, for each (organization, location) relation pair the system forms a tuple. Therefore, a popular pair will have a set of tuples associated with it, and new patterns are induced. The system matches each relation candidate with all patterns and only keeps candidates that have similarity score greater than some threshold. Next, Snowball assigns a high confidence score for a relation candidate when the candidate is matched by many tuples with high similarity to pattern that the system confident in. Finally, the new relation is added to the seed set and the process is repeated iteratively.

Compared to DIPRE, Snowball has a flexible matching system. Instead of having exact surface text matches, Snowball's metrics allows for slight variations in token or punctuation. What is a good seed? DIPRE handles this by favoring long patterns, thus making matches less likely. Snowball handles by first removing patterns that induced small number of tuples (unproductive patterns), second a confidence score on patterns, and third a confidence score on each seed.

3.3 KnowItAll (Etzioni et al., 2005) and TextRunner (Banko et al., 2007)

Unlike DIPRE and Snowball, KnowItAll (Etzioni et al., 2005) is a large scale Web IE system that labels its own training examples using only a small set of domain independent extraction patterns. When instantiated for a particular relation, these generic patterns yield relation-specific extraction rules which are then used to learn domain-specific extraction rules. The rules are applied to web pages, identified via search-engine queries, and the resulting extractions are assigned a probability using pointwise mutual information (PMI) derived from search engine hit counts. For example, KnowItAll utilized generic extraction patterns like "<NP1> such as <NP2>" to suggest instantiations of NP2 as candidate members of the class NP1. Next, the system used frequency information to identify which instantiations are most likely to be members of the class. Finally, KnowItAll learns a set of relation specific extraction patterns, for example "capital of <country>", that led it to extract additional cities and so on.

DIPRE, Snowball, and KnowItAll are all relation-specific systems. The set of relations of interest has to be named by the human user in advance. TextRunner (Banko et al., 2007) is proposed to overcome this issue. Instead of requiring relations to be specified in its input, TextRunner learns the relations, classes, and entities from the text in its corpus in a self-supervised fashion.


English
al-takriti had been transferred together with 17 Iraqi ambassadors ...
1
2
3
4
4
4
5
6
7
Relation Generator
Relation Filter
Relation Classifier
Feature Vector
SVM,
Naïve Bayes,
RIPPER
...
17
al-takriti had been transferred together with 17 iraqi ambassadors abroad to baghdad
NN VBD VBN VBN RB IN CD NN NNS RB TO NN
1 2 3 4 5 6 7 8 9 10 11 12
as part of a diplomatic reorganization move announced by the official iraqi newspaper
IN NN IN DT JJ NN NN VBD IN DT NN NN NN
13 14 15 16 17 18 19 20 21 22 23 24 25

al-takriti	- Noun phrase
had been transferred	- Verb phrase
together	- Adverb phrase
with	- Prepositional phrase
17 iraqi ambassadors	- Noun phrase
abroad	- Adverb phrase
baghdad	- Noun phrase
as	- Prepositional phrase
part	- Noun phrase
of	- Prepositional phrase
a diplomatic reorganization move	- Noun phrase
announced	- Verb phrase
by	- Prepositional phrase
the official iraqi newspaper	- Noun phrase

```
(ROOT
  (S
    (NP (NNP al-takriti))
    (VP (VBD had)
      (VP (VBN been)
        (VP (VBN transferred)
          (PRT (RP together))
          (PP (IN with)
            (NP (CD 17) (NN iraqi) (NNS ambassadors)))
          (ADVP (RB abroad))
          (S
            (VP (TO to)
              (VP (VB baghdad)
                (SBAR (IN as)
                  (S
                    (NP
                      (NP (NN part))
                      (PP (IN of)
                        (NP (DT a) (JJ diplomatic) (NN reorganization) (NN move))))
                    (VP (VBD announced)
                      (PP (IN by)
                        (NP (DT the) (JJ official) (NN iraqi) (NN newspaper)))))))))))
    (. .)))
```

Constraints

1. There exists a dependency chain between e1 and e2 that is not longer than a certain length.
2. This chain should contain some words of the relation r (usually the main verb)
3. The path from e1 to e2 along the syntax tree doesn't cross the sentence-like Boundary (e.g. relative clauses). This means that this path can contain S (SINV, ROOT etc) constituents only at the common ancestor position.
4. Entities do not consist solely of the pronoun.
5. r should contain at least one VP tag.
6. r and e2 should have at least on VP tag as a common ancestor.

(al-takriti-1, had-2 been-3 transferred-4 together-5 with-6, 17-7 iraqi-8 ambassadors-9)

(al-takriti-1, had-2 been-3 transferred-4 to-11, baghdad-12)

(al-takriti-1, had-2 been-3 transferred-4, the-22 official-23 iraqi-23 newspapers-24)

(al-takriti-1, announced-20 by-21, the-22 official-23 iraqi-23 newspapers-24)

...

Figure 2: Self-supervised training of Learner module in TextRunner.

TextRunner defines relations in the form of a tuple $t = (e_1, r, e_2)$ where e_1 and e_2 are strings meant to denote entities or noun phrases, and r is a string meant to denotes a relationship between e_1 and e_2. The system contains three components which are 1) self-supervised Learner ; 2) single-pass Extractor; and 3) redundancy-based Assesor.

The Learner first automatically labeled its own training data as positive or negative, then it uses this labeled data to train a binary classifier which is used by Extractor. The Extractor generates one or more candidate relations from each sentence, then runs a classifier and retains the ones labeled as trustworthy relations. The Assesor assigns a probability to each retained tuple based on a probabilistic model of redundancy in text introduced in (Downey et al., 2005).

Figure 2 shows 7-step training process for the English relation classifier in the Learner. Given an English web page, for each sentence a noun phrase chunker (step 1) is called and then the relation candidator (step 2) will generate possible relations. In the following steps (3, 4 and 5), a syntactic parser and dependency parser are run and the relation filter will use parse trees, dependency trees, and set of constraints to label trustworthy and untrustworthy relations (aka positive and negative examples). Once the system had labeled relations, it maps each relation to a feature vector representation (step 6). All features are domain independent and can be evaluated at extraction time without the use of a parser. Following feature extraction, the system uses this set of automatically labeled feature vector as the training set for a binary classifier, such as SVM, Naive Bayes, and so on (step 7). The classifier is language-specific but contains no relation-specific or lexical features. Thus, the classifier can be used in a domain independent manner.

Parsers are only used once when the Learner performs its self-training process for the classifier. The key idea of the system is the Extractor extracts relations from a large amount of text without running the dependency parser. For example, we are interested in extracting relations from English Giga words corpus. Instead of running parsers for the whole corpus, we only need to parse one million words and use the labeled data to train the binary classifier.

3.4 Discussion

The major disadvantage of DIPRE is the hard pattern matching system. For example, two patterns in DIPRE are different if they only differ from a single punctuation. Snowball has a softer pattern matching system but it relies on a named entity recognizer. Current state-of-the-art NER systems are supervised systems (Bikel et al., 1999) and (Finkel et al., 2005) require a lot of annotated training data. It may be not a problem for standard entities such as person, organization, and location, however, Snowball will have a problem when we want to extend to new entity and relation types. TextRunner has the same problem when the system heavily relies on a dependency parser to self annotate its own training data. DIPRE can be ad to extract relations in new language since DIPRE does not depend on any natural language processing tools, such as parser, chunker, NER and so on, which are trained for a specific language.

Furthermore, Snowball, KnowItAll, and TextRunner rely on a large number of input parameters. The definition parameters are clear, and provide options for users to balance their requirements of the system. However, none of these systems mention how they can select optimal parameters and it seems that those parameters are selected by experience and hard coded.

A major limitation of all systems studied here is that they are extracting relation on the sentence level. In fact, relations can span over sentences and even cross documents. It is not straightforward to modify algorithms described here to capture long range relations.

4 Beyond Binary Relations

A major deficiency of all systems studied here is that they focus primarily on binary relations. Semi-supervised systems such as TextRunner claim that their system can deal with n-ary relations but we are not very clear about the algorithmic changes that are required. Recently (McDonald et al., 2005) proposed a framework for extracting complex relations (tuples) between entities in the text. Their experiments are on extracting 4-ary relations from biomedical abstract text. An instance in a relation is a list of entities $(e_1, e_2, ..., e_n)$ where e_i is entity type. For example, we are interested in the ternary relation (*organizer, conference, location*) that relates an organizer to a conference at a particular location. For a sentence "ACL-2010 will be hosted by CMU in Pittsburgh", the system should extract (*CMU, ACL-2010, Pittsburgh*).

Given a sentence, a possible way for extracting complex relations of interest would be first to list all possible tuples. Using all these tuples to train a binary classifier to distinguish valid instances from invalid ones. However, the problem here is the size of possible candidates will grow exponentially, for example a relation type with n entity elements, each element has m possible way then there are $O(m^n)$ possible complex relation candidates. Instead of trying to classify all possible relation instances, the key ideas of (McDonald et al., 2005) are

1. Start by recognizing binary relation instances that appear to be arguments of the relation of interest.

2. Extracted binary relations can be treated as the edges of graph with entity mention as nodes.

3. Reconstruct complex relations by making tuples from selected maximal cliques in the graph.

To train a classifier, they first create the set of all positive and negative pairs in the data. The positive instances are all pairs that occur together in a valid complex relation come from an annotated corpus. Negative examples are instances which have entity elements never occur together in a valid relation. This leads to a large set of positive and negative binary relations. The binary classifier was trained using standard methods such as maximum entropy and conditional random fields.

After identifying all binary relations, the next step is to create an entity graph so that two entities in the graph have an edge if the binary classifier believes they are related. They reconstruct the complex relations instances by finding maximal cliques.[1] To assign confidence score for each possible complex relations they assign weight $w(e)$ to edge e which is equal to the probability that two entities in e are related according to the classifier. The weight of a clique $w(C)$ is defined as the mean weight of the edges in the clique. Since edge weights represent probabilities, they use the geometric mean

$$w(C) = \left(\prod_{e \in E_C} w(e)\right)^{1/|E_C|}$$

If $w(C) \geq 0.5$ then clique C is a valid relation. The system finds all maximal cliques but only considers those where $w(C) \geq 0.5$. The number 0.5 is selected by running experiments. However, a problem is that there are exponentially cliques since the graph is fully connected. They use a pruning strategy that prunes out all edges that would force any clique C to have $w(C) \leq 0.5$.

There are two major advantages of factoring complex relation into binary relations. First it allows for the use of almost any binary relation classifier which have been well studied and are often accurate. Second, the number of possible binary relations is much smaller than the number of possible complex relations.

Similar to Snowball, (McDonald et al., 2005) rely on a name-entity recognizer since the method assumes that entities and their types are known. Another restriction is the arity of complex relation must be know in advance. The binary classifier they used is a feature-based classifier. We learned that kernel methods are very powerful for classifying binary relations, so it is interesting to see experiments of combining kernel methods with this method.

5 Evaluating Relation-Extraction

The evaluation of entity extraction depends on the nature of the method applied (supervised or unsupervised) and the kind of dataset used. First, we briefly describe some of the datasets that are available for entity-relation extraction task.

Dataset	Domain	Language(s)	Year	Task(s)
MUC	Military and News Reports	English	1987−1997	Named Entity Extraction (NER)
ACE	Newswire, Broadcast News, Speech Transcripts, Weblogs	English, Arabic, Chinese	1999−Present	NER, Relation Extraction, Event Detection
MEDLINE	Medical Publications	English	1950−Present	Protein-Protein Interaction and Gene Binding

Table 1: Datasets available for relation extraction evaluation.

5.1 Datasets

Table 1 shows some of the commonly used datasets. The Message Understanding Conference (Grishman & Sundheim, 1996) was a program started by DARPA to facilitate research on new methods of information extraction. The primary tasks for evaluation were Named Entity Recognition (NER) and co-reference resolution. Though the dataset addresses a different challenge, we feel that it can still be extended for evaluating relation-extraction task. NIST initiated the Automatic Content Extraction (NIST, 2007) program in order to develop technologies for extracting and characterizing meanings of human languages in the form of text. Some of the evaluation tasks of this program are shown in Table 1. Some of the relation-types that the annotated data contains are, *organization-location*, *organization-affiliation*, *citizen-resident-religion-ethnicity*, *sports-affiliation* etc. The ACE corpus is most widely used for evaluation and indeed, most of the supervised methods discussed in this paper make use of the same corpus. The Wikipedia [2] also appears to be a rich source of data for evaluation as it contains hyperlinked entities in most of its pages and has been used for relation extraction in (Culotta et al., 2006) and (Nguyen et al., 2007). The MEDLINE (PubMed, 2007) corpus is a collection of articles in the medical domain. (Bunescu & Mooney, 2005b) have used a subset of articles from MEDLINE for their experiments on detecting protein-protein interaction using subsequence kernels.

5.2 Evaluation in Supervised Methods

In the supervised methods setting, relation extraction is expressed as a classification task and hence, metrics like Precision, Recall and F-Measure are used for performance evaluation. These metrics are defined as follows:

$$\text{Precision } P = \frac{\text{Number of correctly extracted entity relations}}{\text{Total number of extracted entity relations}} \tag{6}$$

$$\text{Recall } R = \frac{\text{Number of correctly extracted entity relations}}{\text{Actual number of extracted entity relations}} \tag{7}$$

$$\text{F-Measure } F1 = \frac{2PR}{P+R} \tag{8}$$

5.3 Evaluation of Semi-supervised Methods

In the absence of labeled test data, evaluating semi-supervised methods is a slightly different procedure though the underlying metrics remain the same (Precision, Recall and F-Measure). Semi-supervised methods of relation extraction are typically applied on large amounts of data resulting often in the discovery of a large number of new patterns and relations. Therefore, getting an exact

[1] A clique C of $G = (V, E)$ is a subgraph of an undirected graph G in which there is an edge between every pair of vertices. A maximal clique of G is a clique $C = (V_C, E_C)$ such that there is no other cliques $C' = (V'_C, E'_C)$ such that $V_C \in V'_C$

[2] http://wikipedia.org/

measure of precision and recall is difficult. A small sample drawn randomly from the output is treated as a representative of the output and manually checked for actual relations. Then, an approximate estimate of the precision is calculated using the definition in section 5.2. This procedure for evaluation has been used by (Brin, 1998) and (Banko et al., 2007). Since the actual number of entity relations are difficult to obtain from large amounts of data, it is difficult to compute recall to evaluate semi-supervised methods.

6 Applications

The world wide web is a big storehouse of information which is often unstructured. Structuring this information involves among other things identifying the identities present relational structure between them. For example, it would be possible to extract the entire family-tree of a prominent personality using a resource like Wikipedia. In a way, relations describe the semantic relationships among the entities involves which is useful for a better understanding of human language. In this section we describe two important applications of relation extraction namely: automatic question-answering and bio-text mining.

6.1 Question Answering (QA)

If a query to a search engine is *"When was Gandhi born ?"*, then the expected answer would be *"Gandhi was born in 1869"*. The template of the answer is *<PERSON> born-in <YEAR>* which is nothing but the relational triple born_in(PERSON, YEAR) where PERSON and YEAR are the entities. To extract the relational triples, a large database (ex: web) can be queried using a small initial question-answer set (ex: *"Gandhi 1869"*). The best matching (or most confident) patterns are then used to extract answer templates which in turn can be used to extract new entities from the database. The new entities are again used to extract newer answer templates and so on till convergence. This bootstrapping based method for QA is described in (Ravichandran & Hovy, 2002).

6.2 Mining Biotext

Relation extraction methods are useful in discovering protein-protein interactions, and gene-binding conditions. Patterns like "Protein X binds with Protein Y" are often found in biomedical texts where the protein names are entities which are held together by the "bind" relation. Such protein-protein interactions are useful for applications like drug discovery etc. Other relations of interest are, a protein's location inside an organism. Such ternary relationships are extracted using linear kernels computed over features in (Liu et al., 2007). Cancer researchers can use inferences like *"Gene X with mutation Y leads to malignancy Z"* in order to isolate cancerous genes. These information patterns can be pieced together by extracting ternary relations between genes, mutations and malignancy conditions in a large corpus of biotext.

7 Conclusions

So far, we have reviewed all the aspects of the entity-relation extraction problem starting with the algorithms, discussing the evaluation criteria finally culminating with a discussion of some important applications. Among the supervised approaches, dependency path kernels proposed by (Bunescu & Mooney, 2005a) stand out as the best both in terms of computational complexity and performance. Surprisingly the tree kernel of (Zelenko et al., 2003) has not been comparatively evaluated with other kernels which leaves room for speculation. It is clear that kernel methods outperform feature-based approaches for supervised relation extraction. Semi-supervised approaches seem to be well suited for open domain relation extraction systems since they can easily scale with the database size and can extend to new relations easily. Supervised approaches on the other hand can do well when the domain is more restricted like the case of biotext mining. The problem of N-ary relation extraction is often factored in sets of binary relation-extraction problems which can be suboptimal. It would be interesting to investigate approaches that handle higher order relations efficiently without factorizing them.

References

Abney, S. (2004). Understanding the yarowsky algorithm. *Comput. Linguist.* (pp. 365–395). Cambridge, MA, USA: MIT Press.

Agichtein, E., & Gravano, L. (2000). Snowball: Extracting relations from large plain-text collections. *Proceedings of the Fifth ACM International Conference on Digital Libraries*.

Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., & Etzioni, O. (2007). Open information extraction from the web. *IJCAI '07: Proceedings of the 20th International Joint Conference on Artificial Intelligence*. Hyderabad, India.

Bikel, D. M., Schwartz, R. L., & Weischedel, R. M. (1999). An algorithm that learns what's in a name. *Machine Learning*, *34*, 211–231.

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers* (pp. 92–100).

Brin, S. (1998). Extracting patterns and relations from the world wide web. *WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT '98*.

Bunescu, R. C., & Mooney, R. J. (2005a). A shortest path dependency kernel for relation extraction. *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing* (pp. 724–731). Vancouver, British Columbia, Canada: Association for Computational Linguistics.

Bunescu, R. C., & Mooney, R. J. (2005b). Subsequence kernels for relation extraction. *Neural Information Processing Systems, NIPS 2005, Vancouver, British Columbia, Canada*.

Culotta, A., McCallum, A., & Betz, J. (2006). Integrating probabilistic extraction models and data mining to discover relations and patterns in text. *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics* (pp. 296–303). New York, New York: Association for Computational Linguistics.

Culotta, A., & Sorensen, J. (2004). Dependency tree kernels for relation extraction. *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics* (p. 423). Morristown, NJ, USA: Association for Computational Linguistics.

Downey, D., Etzioni, O., & Soderland, S. (2005). A probabilistic model of redundancy in information extraction. *IJCAI* (pp. 1034–1041).

Etzioni, O., Cafarella, M., Downey, D., Popescu, A. M., Shaked, T., Soderland, S., Weld, D. S., & Yates, A. (2005). Unsupervised Named-Entity Extraction from the Web: An Experimental Study. *Artificial Intelligence* (pp. 191–134).

Finkel, J. R., Grenager, T., & Manning, C. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics* (pp. 363–370). Morristown, NJ, USA: Association for Computational Linguistics.

Grishman, R., & Sundheim, B. (1996). Message understanding conference - 6: A brief history. *Proceedings of the 16th conference on Computational Linguistics* (pp. 466–471).

GuoDong, Z., Jian, S., Jie, Z., & Min, Z. (2002). Exploring various knowledge in relation extraction. *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics* (pp. 419–444).

Kambhatla, N. (2004). Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. *Proceedings of the ACL 2004*.

Liu, Y., Shi, Z., & Sarkar, A. (2007). Exploiting rich syntactic information for relationship extraction from biomedical articles. *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers* (pp. 97–100). Rochester, New York: Association for Computational Linguistics.

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

McDonald, R. (2004). Extracting relations from unstructured text. *UPenn CIS Technical Report*.

McDonald, R., Pereira, F., Kulick, S., Winters, S., Jin, Y., & White, P. (2005). Simple algorithms for complex relation extraction with applications to biomedical ie. *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics* (pp. 491–498). Ann Arbor, Michigan.

Nguyen, D. P., Matsuo, Y., & Ishizuka, M. (2007). Subtree mining for relation extraction from Wikipedia. *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers* (pp. 125–128). Rochester, New York: Association for Computational Linguistics.

NIST (2007). The ace 2007 (ace07) evaluation plan. *http://www.nist.gov/speech/tests/ace/ace07/doc/ace07-evalplan.v1.3a.pdf*.

PubMed (2007). Medline. *PubMed Home, http://www.ncbi.nlm.nih.gov/sites/entrez*.

Ravichandran, D., & Hovy, E. (2002). Learning surface text patterns for a question answering system. *In proceedings of the ACL Conference*.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. *Proceedings of the 33rd conference on Association for Computational Linguistics* (pp. 189–196). NJ, USA.

Zelenko, D., Aone, C., & Richardella, A. (2003). Kernel methods for relation extraction. *Journal of Machine Learning Research*.

Zhao, S., & Grishman, R. (2005). Extracting relations with integrated information using kernel methods. *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics* (pp. 419–426).