Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Philadelphia, July 2002, pp. 71-78. Association for Computational Linguistics.

Kernel Methods for Relation Extraction

Dmitry Zelenko **Chinatsu Aone** **Anthony Richardella**
{dmitry_zelenko,chinatsu_aone,anthony_richardella}@sra.com
SRA International
4300 Fair Lakes Ct.
Fairfax VA 22033 USA

Abstract

We present an application of kernel methods to extracting relations from unstructured natural language sources. We introduce kernels defined over shallow parse representations of text, and design efficient algorithms for computing the kernels. We use the devised kernels in conjunction with Support Vector Machine and Voted Perceptron learning algorithms for the task of extracting `person-affiliation` and `organization-location` relations from text. We experimentally evaluate the proposed methods and compare them with feature-based learning algorithms, with promising results.

1 Introduction

Information extraction is an important unsolved problem of NLP. It is the problem of extracting entities and relations among them from text documents. Examples of entities are `people`, `organizations`, and `locations`. Examples of relations are `person-affiliation` and `organization-location`. The `person-affiliation` relation means that a particular `person` is affiliated with a certain `organization`. For instance, the sentence, "John Smith is the chief scientist of the Hardcom Corp.", contains the `person-affiliation` relation between the `person` "John Smith" and the `organization` "Hardcom Corp.". In this paper, we address the problem of extracting such relations from natural language text.

We propose a machine learning approach to relation extraction and a novel methodology for information extraction based on kernel methods (Vapnik, 1998; Cristianini and Shawe-Taylor, 2000).

We believe that *shallow* parsing is an important prerequisite for information extraction. Shallow parsing provides a robust mechanism for producing text representations that can be effectively used for entity and relation extraction.

Indeed, the first step of our relation extraction approach is a powerful shallow parsing component of an information extraction system (Aone and Ramos-Santacruz, 2000). The system comprises cascading finite state machines that identify names, noun phrases, and a restricted set of parts of speech in text. The system also classifies noun phrases and names as to whether they refer to `people`, `organizations` and `locations`, thereby producing *entities*. Thus, the input to the relation extraction system is a shallow parse, with noun phrases and names marked with relevant entity types.

We formalize a relation extraction problem as a shallow parse classification problem in section 4. A shallow parse is turned into an example whose label reflects whether a relation of interest is expressed by the shallow parse. The learning system uses the labeled examples to output a model that is applied to shallow parses to obtain labels, and thus extract relations.

A unique property of the kernel methodology is that we do not explicitly generate features. More precisely, an example is no longer a feature vector as is common in machine learning algorithms. Instead, examples retain their original representations (of shallow parses) and are used within learning algorithms only via computing a similarity (or kernel[1]) function between them. Such a use of examples allows our learning system to *implicitly* explore a much larger feature space than one computationally feasible for processing with feature-based learning algorithms.

We conduct an experimental evaluation of our approach in section 6. We compare our approach with the feature-based linear methods (Roth, 1999), with promising results.

2 Related Work on Information Extraction

The problem of relation extraction from natural language texts was previously addressed by Message Understanding Conferences (MUC). A number of systems were developed that relied on parsing and manual pattern development for identifying the relations of interest (see, for example, (Aone et al., 1998)). An adaptive system (Miller et al., 1998), presented under the aegis of MUC, used lexicalized probabilistic context-free grammars augmented with semantic information to produce a semantic parse of text for detecting `organization-location` relations. Among other popular probabilistic formalisms for information extraction are Hidden Markov Models (HMM) (Bikel et al., 1999), Maximum Entropy Markov Models (MEMM) (McCallum et al., 2000) and Conditional Random Fields (CRF) (Lafferty et al., 2001).

Online learning algorithms for learning linear models (e.g., Perceptron, Winnow) are becoming increasingly popular for NLP problems (Roth, 1999). The algorithms exhibit a number of attractive features such as incremental learning and scalability to a very large number of examples. Their recent applications to shallow parsing (Munoz et al., 1999) and information extraction (Roth and Yih, 2001) exhibit state-of-the-art performance. The linear models are, however, feature-based which imposes constraints on their exploiting long-range dependencies in text. In section 6, we compare the methods with our approach for the relation extraction problem.

[1]A kernel function is a similarity function satisfying certain properties, see (Cristianini and Shawe-Taylor, 2000) for details.

We next introduce a class of kernel machine learning methods and apply them to relation extraction.

3 Kernel-based Machine Learning

Most learning algorithms rely on feature-based representation of objects. That is, an object is transformed into a collection features $f_1, \ldots, f_N$, thereby producing a N-dimensional vector.

In many cases, data cannot be easily expressed via features. For example, in most NLP problems, feature based representations produce inherently local representations of objects, for it is computationally infeasible to generate features involving long-range dependencies.

Kernel methods (Vapnik, 1998; Cristianini and Shawe-Taylor, 2000) are an attractive alternative to feature-based methods. Kernel methods retain the original representation of objects and use the objects in algorithms only via computing a kernel (or similarity) function between a pair of objects.

In many cases, it may be possible to compute a similarity function in terms of certain features without enumerating all the features. An excellent example is that of subsequence kernels (Lodhi et al., 2002). In this case, the objects are strings of characters, and the similarity (kernel) function computes the number of common subsequences of characters in two strings. Despite the exponential number of features (subsequences), it is possible to compute the subsequence kernel in polytime. We therefore are able to take advantage of long-range features in strings without enumerating the features explicitly. In section 5, we will extend the subsequence kernel to operate on shallow parses for relation extraction.

Another pertinent example is that of parse tree kernels(Collins and Duffy, 2001), where objects represent trees and the kernel function computes the number of common subtrees in two trees. The tree kernel used within the Voted Perceptron learning algorithm (Freund and Schapire, 1999) was shown to deliver excellent performance in improving Penn Treebank parsing. There are a number of learning algorithms that can operate only using kernels of examples. The models produced by the learning algorithms are also expressed using only examples' kernels. The algorithms that process examples only via computing their kernels are sometimes called *dual* learning algorithms.

The Support Vector Machine(SVM) (Cortes and Vapnik, 1995) is a learning algorithm that not only allows for a dual formulation, but also provides a rigorous rationale for resisting overfitting (Vapnik, 1998). After discovery of the kernel methods, several existing learning algorithms were shown to have dual analogues. For instance, the Perceptron learning algorithm can be easily represented in the dual form (Cristianini and Shawe-Taylor, 2000). A variance-reducing improvement of Perceptron, Voted Perceptron (Freund and Schapire, 1999), is a robust and efficient learning algorithm that is very easy to implement. It has been shown to exhibit performance comparable to that of SVM. In section 6, we experimentally evaluate SVM and Voted Perceptron for relation extraction.

We next show how to formalize relation extraction as a learning problem.

4 Problem Formalization

Let us consider the sentence, "John Smith is the chief scientist of the Hardcom Corp.". The shallow parsing system produces the representation of the sentence shown in Figure 1.

We convert the shallow parse tree into one or more examples for the `person-affiliation` relation. This type of relation holds between a `person` and an `organization`. There are three nodes in the shallow parse tree in Figure 1 referring to people, namely, the "John Smith" node with the type "Person", and the "PNP" nodes[2]. There is one "Organization" node in the tree that refers to an organization. We create an example for the `person-affiliation` relation by taking a `person` node and an `organization` node in the shallow parse tree and assigning attributes to the nodes specifying the role that a node plays in the `person-affiliation` relation. The person and organization under consideration will receive the *member* and *affiliation* roles, respectively. The rest of the nodes will receive *none* roles reflecting that they do not participate in the relation. We then attach a label to the example by asking the question whether the node with the role of *member* and the node with the role of *affiliation* are indeed (semantically) affiliated, according to the sentence. For the above sentence, we will then generate three positive examples, shown in Figure 2.

[2]Note that after the tree is produced, we do not know if the "Person" and the "PNP" nodes refer to the same person.

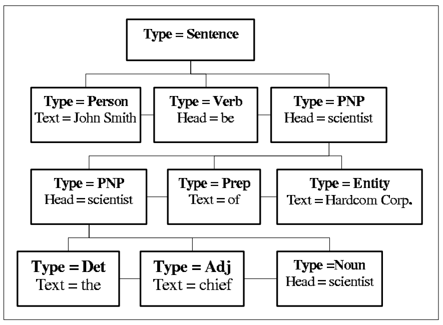

Figure 1: The shallow parse representation of the the sentence "John Smith is the chief scientist of the Hardcom Corp.".The types "PNP", "Det", "Adj", and "Prep" denote "Personal Noun Phrase", "Determiner", "Adjective", and "Preposition", respectively.

Note that in generating the examples between the "PNP" and the "Organization" we eliminated the nodes that did not belong to the least common subtree of "Organization" and "PNP", thereby removing irrelevant subtrees.

To summarize, a relation example is shallow parse, in which nodes are augmented with the role attribute, and each node of the shallow parse belongs to the least common subtree comprising the relation entities under consideration.

We now formalize the notion of relation example. We first define the notion of the example node.

Definition 1 *A node p is a set of attributes $\{a_1, a_2, \ldots\}$. The attributes are named.*

We use $p.a$ to denote the value of attribute with the name a in the node p, e.g., $p.Type = Person$ and $p.Role = member$.

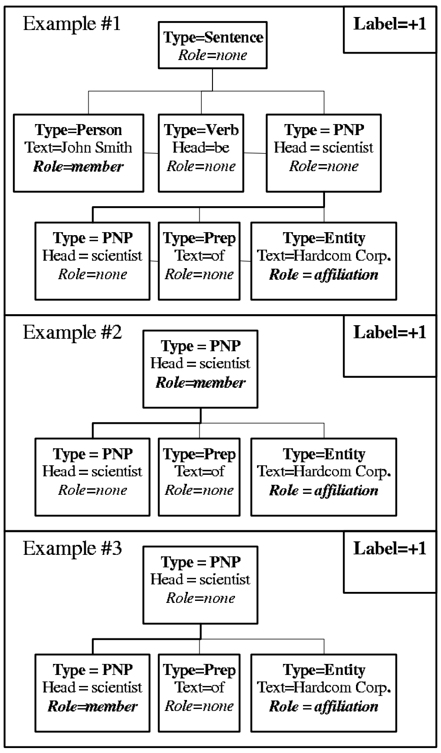

Figure 2: The three `person-affiliation` examples generated from the shallow parse in Figure 1. The "Label=+1" means that the examples do express the relation.

Definition 2 *An (unlabeled) relation example is defined inductively as follows:*

- *Let p be a node, then the pair $P = (p, [])$ is a relation example, where by $[]$ we denote an empty sequence.*

- *Let p be a node, and $[P_1, P_2, \ldots, P_l]$ be a sequence of relation examples. Then, the pair $P = (p, [P_1, P_2, \ldots, P_l])$ is a relation example.*

We say that p is the parent of $P_1, P_2, \ldots, P_l$, and P_i's are the children of p. We denote by $P.p$ the first element of the example pair, by $P.c$ the second element of the example pair, and use the shorthand $P.a$ to refer to $P.p.a$, and $P[i]$ to denote P_i.

A labeled relation example is an unlabeled relation example augmented with a label $l \in \{-1, +1\}$. An example is positive, if $l = +1$, and negative, otherwise.

We now define kernels on relation examples that represent similarity of two shallow parse trees.

5 Kernels for Relation Extraction

Kernels on parse trees were previously defined by (Collins and Duffy, 2001). The kernels enumerated (implicitly) all subtrees of two parse trees, and used the number of common subtrees, weighted appropriately, as the measure of similarity between two parse trees. Since we are operating with shallow parse trees, and the focus of our problem is relation extraction rather than parsing, we use a different definition of kernels.

We first define a matching function $t(\cdot, \cdot) \in \{0, 1\}$ and a similarity function $k(\cdot, \cdot)$ on nodes. The matching function defined on nodes determines whether the nodes are matchable or not. For example, the nodes may be matchable only if their types and roles match. That is, if two nodes have the same roles, and compatible types[3], then their node matching function is equal to 1; otherwise, it is equal to 0. The similarity function on nodes is computed in terms of the nodes' attributes.

Then, for two relation examples P_1, P_2, we define the similarity function $K(P_1, P_2)$ in terms of similarity function of the parent nodes and the similarity function K_c of the children. Formally (o.w. means "otherwise"),

$$K(P_1,P_2)=\begin{cases} 0, \text{ if } t(P_1.p,P_2.p)=0 \\ k(P_1.p,P_2.p)+K_c(P_1.c,P_2.c), \text{ o.w.} \end{cases} \tag{1}$$

Different definitions of the similarity function K_c on children give rise to different K's. We now give a general definition of K_c in terms of similarities of children subsequences. We first introduce some helpful notation (similar to (Lodhi et al., 2002)).

We denote by $\mathbf{i}$ a sequence $i_1 < i_2 < \ldots < i_n$ of indices, and we say that $i \in \mathbf{i}$, if i is one of the sequence indices. We also use $d(\mathbf{i})$ for $i_n - i_1 + 1$, and $l(\mathbf{i})$ for length of the sequence $\mathbf{i}$. For a relation example P, we denote by $P[\mathbf{i}]$ the sequence of children $[P[i_1], \ldots, P[i_n]]$.

[3]Some distinct types are compatible, for example, "PNP" may be compatible with "Person".

For a similarity function K, we use $K(P_1[\mathbf{i}], P_2[\mathbf{j}])$ to denote $\sum_{s=1,\ldots,l(\mathbf{i})} K(P_1[i_s], P_2[j_s])$. Then, we define the similarity function K_c as follows

$$K_c(P_1.c,P_2.c)= \sum_{\substack{\mathbf{i},\mathbf{j}\\ l(\mathbf{i})=l(\mathbf{j})}} \lambda^{d(\mathbf{i})+d(\mathbf{j})} K(P_1[\mathbf{i}],P_2[\mathbf{j}]) T(\mathbf{i},\mathbf{j}) \quad (2)$$

where

$$T(\mathbf{i},\mathbf{j})=\prod_{s=1,\ldots,l(\mathbf{i})} t(P_1[i_s].p,P_2[j_s].p)$$

The formula (2) enumerates all subsequences of relation example children with matching parents, accumulates the similarity for each subsequence by adding the corresponding child examples' similarities, and decreases the similarity by the factor of $\lambda^{d(\mathbf{i})}\lambda^{d(\mathbf{i})}$, $0 < \lambda < 1$, reflecting how spread out the subsequences within children sequences. Finally, the similarity of two children sequences is the sum all matching subsequences similarities.

The following theorem states that the formulas (1) and (2) define a kernel, under mild assumptions (the proof is omitted for lack of space).

Theorem 1 *Let $k(\cdot,\cdot)$ and $t(\cdot,\cdot)$ be kernels over nodes. Then, K as defined by (1) and (2) is a kernel over relation examples.*

We first consider a special case of K_c, where the subsequences $\mathbf{i}$ and $\mathbf{j}$ are assumed to be *contiguous* and give a very efficient algorithm for computing K_c. In section 5.2, we address a more general case, when the subsequences are allowed to be sparse (non-contiguous).

5.1 Contiguous Subtree Kernels

For contiguous subtree kernels, the similarity function K_c enumerates only children contiguous subsequences, that is, for a subsequence $\mathbf{i}$ in (2), $i_{s+1} = i_s + 1$ and $d(\mathbf{i}) = l(\mathbf{i})$. Since then $d(\mathbf{i}) = d(\mathbf{j})$ as well, we slightly abuse notation in this section by making λ stand for λ^2 in formula (2). Hence, (2) becomes

$$K_c(P_1.c,P_2.c)= \sum_{\substack{\mathbf{i},\mathbf{j}\\ l(\mathbf{i})=l(\mathbf{j})}} \lambda^{l(\mathbf{i})} K(P_1[\mathbf{i}],P_2[\mathbf{j}]) T(\mathbf{i},\mathbf{j}) \quad (3)$$

The core of the kernel computation resides in the formula (3). The formula enumerates all contiguous subsequences of two children sequences. We now give a fast algorithm for computing K_c between P_1 and P_2, which, given kernel values for children, runs in time $O(mn)$, where m and n is the number of children of P_1 and P_2, respectively.

Let $C(i,j)$ be the K_c computed for suffixes of children sequences of P_1 and P_2, where every subsequence starts with indices i and j, respectively. That is,

$$C(i,j)= \sum_{\substack{\mathbf{i},\mathbf{j},i_1=i,j_1=j\\ l(\mathbf{i})=l(\mathbf{j})}} \lambda^{l(\mathbf{i})} K(P_1[\mathbf{i}],P_2[\mathbf{j}] T(\mathbf{i},\mathbf{j})$$

Let $L(i,j)$ be the length of the longest sequence matching states in the children of P_1 and P_2 starting with indices i and j, respectively:

$$L(i,j)=\max\{l:\prod_{s=0,\ldots,l} t(P_1[i+s].p,P_2[j+s].p)=1\}$$

Then, the following recurrences hold:

$$L(i,j)=\begin{cases} 0, \text{ if } t(P_1[i].p,P_2[j],p)=0 \\ L(i+1,j+1)+1, \text{ otherwise} \end{cases} \quad (4)$$

$$C(i,j)=\begin{cases} 0, \text{ if } t(P_1[i].p,P_2[j],p)=0 \\ \frac{\lambda-\lambda^{L(i,j)+1}}{1-\lambda} K(P_1[i],P_2[j])+\lambda C(i+1,j+1) \\ \qquad \text{otherwise} \end{cases} \quad (5)$$

The boundary conditions are:

$$L(m+1,n+1)=0 \quad \text{and} \quad C(m+1,n+1)=0$$

The recurrence (5) follows from the observation that, if $P_1[i]$ and $P_2[j]$ match, then every matching pair (c_1, c_2) of sequences that participated in computation of $C(i+1, j+1)$ will be extended to the matching pair $([P_1[i], c_1], [P_2[j], c_2])$. Now we can easily compute $K_c(P_1.c, P_2.c)$ from $C(i, j)$.

$$K_c(P_1.c,P_2.c)=\sum_{i,j} C(i,j) \quad (6)$$

The time and space complexity of K_c computation is $O(mn)$, given kernel values for children. Hence, for two relation examples the complexity of computing $K(P_1, P_2)$ is the sum of computing K_c for the matching internal nodes (assuming that complexities of $k(\cdot,\cdot)$ and $t(\cdot,\cdot)$ are constant).

5.2 Sparse Subtree Kernels

For sparse subtree kernels, we use the general definition of similarity between children sequences as expressed by (2).

As in the previous section, we give an efficient algorithm for computing K_c between P_1 and P_2. The algorithm runs in time $O(mn^3)$, given kernel values for children, where m and n ($m \geq n$) is the number of children of P_1 and P_2, respectively.

Derivation of an efficient programming algorithm for sparse subtree computation will be presented in a full version of the paper, for lack of space. Below we list the recurrences for computing K_c.

$$
\begin{aligned}
K_c &= \textstyle\sum_{q=1,\ldots,\min(m,n)} K_{c,q}(m,n) \\
K_{c,q}(i,j) &= \lambda K_{c,q}(i,j-1)+\textstyle\sum_{s=1,\ldots,i} \big[t(P_1[s].p,P_2[j].p)\cdot \\
&\quad \lambda^2 C_{q-1}(s-1,j-1,K(P_1[s],P_2[j]))\big] \\
C_q(i,j,a) &= aC_q(i,j)+\textstyle\sum_{r=1,\ldots,q} C_{q,r}(i,j) \\
C_q(i,j) &= \lambda C_q(i,j-1)+C'_q(i,j) \\
C'_q(i,j) &= t(P_1[i],P_2[j])\lambda^2 C_{q-1}(i-1,j-1)+\lambda C'_q(i,j-1) \\
C_{q,r}(i,j) &= \lambda C_{q,r}(i,j-1)+C'_{q,r}(i,j) \\
C'_{q,r}(i,j) &= \begin{cases} \lambda C'_{q,r}(i,j-1)+ \\ t(P_1[i],P_2[j])\lambda^2 C_{q-1,r}(i-1,j-1), \text{ if } q\neq r \\ \lambda C'_{q,r}(i,j-1)+t(P_1[i],P_2[j])\cdot \\ \quad \lambda^2 K(P_1[i],P_2[j])C_{q-1}(i-1,j-1), \text{ o.w.} \end{cases}
\end{aligned}
$$

The boundary conditions are

$$
\begin{aligned}
K_{c,q}(i,j) &= 0, \text{ if } q>\min(i,j) \\
C_q(i,j) &= 0, \text{ if } q>\min(i,j) \\
C_0(i,j) &= 1, \\
C'_q(i,j) &= 0, \text{ if } q>\min(i,j) \\
C_{q,r}(i,j) &= 0, \text{ if } q>\min(i,j) \text{ or } q<r \\
C'_{q,r}(i,j) &= 0, \text{ if } q>\min(i,j) \text{ or } q<r
\end{aligned}
$$

As can be seen from the recurrences, the time complexity of the algorithm is $O(mn^3)$ (assuming $m \geq n$). The space complexity is $O(mn^2)$.

6 Experiments

In this section, we apply kernel methods to extracting two types of relations from text: `person-affiliation` and `organization-location`.

A `person` and an `organization` are part of the `person-affiliation` relation, if the `person` is a member of or employed by `organization`. A company founder, for example, is defined not to be affiliated with the company (unless, it is stated that (s)he also happens to be a company employee).

An `organization` and a `location` are part of the `organization-location` relation, if the `organization`'s headquarters is at the `location`. Hence, if a single division of a company is located in a particular city, the company is not necessarily located in the city.

The nuances in the above relation definitions make the extraction problem more difficult, but they also allow to make fine-grained distinctions between relationships that connect entities in text.

6.1 Experimental Methodology

The (text) corpus for our experiments comprises 200 news articles from different news agencies and publications (Associated Press, Wall Street Journal, Washington Post, Los Angeles Times, Philadelphia Inquirer).

We used the existing shallow parsing system to generate the shallow parses for the news articles. We generated relation examples from the shallow parses for both relations, as described in section 4. We retained only the examples, for which the shallow parsing system did not make major mistakes (90% of the generated examples). We then labeled the retained examples whether they expressed the relation of interest, whereby we obtained 3524 (1262 positive) examples for the `person-affiliation` relation and 1915 (506 positive) examples for the `org-location` relation.

For each relation, we randomly split the set of examples into a training set (60% of the examples) and a testing set (40% of the examples). We obtained the models by running learning algorithms (with kernels, where appropriate) on the training set, testing the models on the test set, and computing performance measures. In order to get stable performance estimates, we averaged performance results over 10 random train/test splits.

We report the standard performance estimates (precision, recall, and F-measure) for each experiment.

We now describe the experimental setup of the algorithms used in evaluation.

6.2 Kernel Methods Configuration

We evaluated two kernel learning algorithms: Support Vector Machine (SVM) (Cortes and Vapnik, 1995) and Voted Perceptron (Freund and Schapire, 1999). For SVM, we used the SVM^{Light} (Joachims, 1998) implementation of the algorithm, with custom kernels incorporated therein. We implemented the Voted Perceptron algorithm as described in (Freund and Schapire, 1999).

We implemented both contiguous and sparse subtree kernels and incorporated them in the kernel learning algorithms. For both kernels, λ was set to 0.5. The only domain specific information in the two kernels was encapsulated by the following matching $t(\cdot,\cdot)$ and similarity $k(\cdot,\cdot)$ functions on nodes.[4]

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

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

We also normalized the computed kernels before their use within the algorithms. The normalization corresponds to the standard unit norm normalization of examples in the feature space corresponding to the kernel space (Cristianini and Shawe-Taylor, 2000):

$$K(P_1,P_2)=\frac{K(P_1,P_2)}{\sqrt{K(P_1,P_1)K(P_2,P_2)}}$$

6.3 Linear Methods Configuration

We evaluated two feature-based algorithms for learning linear discriminant functions: Naive-Bayes (Duda and Hart, 1973) and Winnow (Littlestone, 1987).

[4]The function $Class$ combines some types into a single equivalence class: $Class(PNP) = Person$, $Class(ONP) = Organization$, $Class(LNP) = Location$, and $Class(Type) = Type$ for other types.

We implemented the two algorithms in the spirit of the SNOW system (Roth, 1999). The algorithms learn models that, given an example, produce a score for each label (+1 and -1), the predict the label corresponding to the larger score.

Since the algorithms are feature-based, we designed features for the relation extraction problem. The features are conjunctions of conditions involving "Text", "Type", "Role" attributes for neighboring example nodes. We do not list features herein for lack of space.

6.4 Experimental Results

The performance results for relation extraction are shown in Table 1.

The results indicate that kernel methods do exhibit excellent performance and fare better than feature-based algorithms in relation extraction. The results also highlights importance of kernels: algorithms with the sparse subtree kernels are always significantly better than their contiguous counterparts.

The results show that performance of the Voted Perceptron is much less accurate, compared with other algorithms, for the `organization-location` relation than for the `person-affiliation` relation. This phenomenon can be probably attributed to the fact that the `organization-location` examples are more noisy, with more boundary cases present. For such a training set, regularization performed by SVM is crucial; it is more noise-tolerant than the Perceptron voting mechanism. Performance of Naive Bayes for `organization-location` relation is notable, since it performs as good as or better than more elaborate algorithms.

7 Conclusions and Further Work

We presented an approach for using kernel-based machine learning methods for extracting relations from natural language sources. We defined kernels over shallow parse representations of text and designed efficient dynamic programming algorithms for computing the kernels. We applied SVM and Voted Perceptron learning algorithms with the kernels incorporated therein

	person-affiliation			org-location		
	Recall	Precision	F-measure	Recall	Precision	F-measure
Naive Bayes	75.59	91.88	82.93	71.94	90.40	80.04
Winnow	80.87	88.42	84.46	75.14	85.02	79.71
Voted Perceptron (contig.)	79.58	89.74	84.34	64.43	92.85	76.02
SVM (contig.)	79.78	89.9	84.52	71.43	92.03	80.39
Voted Perceptron (sparse)	81.62	90.05	85.61	71	91.9	80.05
SVM (sparse)	82.73	91.32	**86.8**	76.33	91.78	**83.3**

Table 1: Relation extraction performance (in percentage points)

to the tasks of relation extraction. We also compared performance of the kernel-based methods with that of the feature methods, and concluded that kernels lead to superior performance.

In the future, we intend to apply the kernel methodology to other sub-problems of information extraction. For example, the shallow parsing and entity extraction mechanism may also be learned, and, perhaps, combined in a seamless fashion with the relation extraction formalism presented herein. Furthermore, the real-world use of extraction results requires discourse resolution that collapses entities, noun phrases, and pronouns into a set of equivalence classes. We plan to apply kernel methods for discourse processing as well.

8 Acknowledgements

Our work was supported through the DARPA Evidence Extraction and Link Discovery Program.

References

C. Aone and M. Ramos-Santacruz. 2000. REES: A large-scale relation and event extraction system. In *Proceedings of ANLP-2000.*

C. Aone, L. Halverson, T. Hampton, and M. Ramos-Santacruz. 1998. SRA: Description of the IE2 system used for MUC-7. In *Proceedings of MUC-7.*

D. Bikel, R. Schwartz, and R. Weischedel. 1999. An algorithm that learns what's in a name. *Machine Learning*, 34(1-3):211–231.

M. Collins and N. Duffy. 2001. Convolution kernels for natural language. In *Proceedings of NIPS-2001.*

C. Cortes and V. Vapnik. 1995. Support-vector networks. *Machine Learning*, 20(3):273–297.

N. Cristianini and J. Shawe-Taylor. 2000. *An Introduction to Support Vector Machines (and Other Kernel-Based Learning Methods).* CUP

R. O. Duda and P. E. Hart. 1973. *Pattern Classification and Scene Analysis.* John Wiley, New York.

Y. Freund and R. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

T. Furey, N. Cristianini, N. Duffy, D. Bednarski, M. Schummer, and D. Haussler. 2000. Support vector machine classification and validation of cancer tissue samples using microarray expression. *Bioinformatics*, 16.

D. Haussler. 1999. Convolution kernels on discrete structures. UC Santa Cruz Technical Report UCS-99-10.

T. Joachims. 1998. Text categorization with support vector machines: learning with many relevant features. *Proceedings of ECML-98.*

T. Joachims. 2002. *Learning Text Classifiers with Support Vector Machines.* Kluwer Academic Publishers, Dordrecht, NL.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML-2001.*

N. Littlestone. 1987. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. 2002. Text classification using string kernels. *Journal of Machine Learning Research.*

A. McCallum, D. Freitag, and F. Pereira. 2000. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of International Conference on Machine Learning, 2000.*

S. Miller, M. Crystal, H. Fox, L. Ramshaw, R. Schwartz, R. Stone, and R. Weischedel. 1998. Algorithms that learn to extract information - BBN: Description of the SIFT system. In *Proceedings of MUC-7.*

M. Munoz, V. Punyakanok, D. Roth, and D. Zimak. 1999. A learning approach to shallow parsing. TR-2087, University of Illinois at Urbana-Champaign.

D. Roth and W. Yih. 2001. Relational learning via propositional algorithms: An information extraction case study. In *Proceedings of IJCAI-01.*

D. Roth. 1999. Learning in natural language. In *Proceedings of IJCAI-99.*

V. Vapnik. 1998. *Statistical Learning Theory.* John Wiley.