Extracting Relations with Integrated Information Using Kernel Methods

Shubin Zhao **Ralph Grishman**
Department of Computer Science
New York University
715 Broadway, 7th Floor, New York, NY 10003
shubinz@cs.nyu.edu grishman@cs.nyu.edu

Abstract

Entity relation detection is a form of information extraction that finds predefined relations between pairs of entities in text. This paper describes a relation detection approach that combines clues from different levels of syntactic processing using kernel methods. Information from three different levels of processing is considered: tokenization, sentence parsing and deep dependency analysis. Each source of information is represented by kernel functions. Then composite kernels are developed to integrate and extend individual kernels so that processing errors occurring at one level can be overcome by information from other levels. We present an evaluation of these methods on the 2004 ACE relation detection task, using Support Vector Machines, and show that each level of syntactic processing contributes useful information for this task. When evaluated on the official test data, our approach produced very competitive ACE value scores. We also compare the SVM with KNN on different kernels.

1 Introduction

Information extraction subsumes a broad range of tasks, including the extraction of entities, relations and events from various text sources, such as newswire documents and broadcast transcripts. One such task, *relation detection*, finds instances of predefined relations between pairs of *entities*, such as a *Located-In* relation between the entities *Centre College* and *Danville, KY* in the phrase *Centre College in Danville, KY*. The 'entities' are the individuals of selected semantic types (such as people, organizations, countries, …) which are referred to in the text.

Prior approaches to this task (Miller et al., 2000; Zelenko et al., 2003) have relied on partial or full syntactic analysis. Syntactic analysis can find relations not readily identified based on sequences of tokens alone. Even 'deeper' representations, such as logical syntactic relations or predicate-argument structure, can in principle capture additional generalizations and thus lead to the identification of additional instances of relations. However, a general problem in Natural Language Processing is that as the processing gets deeper, it becomes less accurate. For instance, the current accuracy of tokenization, chunking and sentence parsing for English is about 99%, 92%, and 90% respectively. Algorithms based solely on deeper representations inevitably suffer from the errors in computing these representations. On the other hand, low level processing such as tokenization will be more accurate, and may also contain useful information missed by deep processing of text. Systems based on a single level of representation are forced to choose between shallower representations, which will have fewer errors, and deeper representations, which may be more general.

Based on these observations, Zhao et al. (2004) proposed a discriminative model to combine information from different syntactic sources using a kernel SVM (Support Vector Machine). We showed that adding sentence level word trigrams as global information to local dependency context boosted the performance of finding slot fillers for

Proceedings of the 43rd Annual Meeting of the ACL, pages 419–426,
Ann Arbor, June 2005.

management succession events. This paper describes an extension of this approach to the identification of entity relations, in which syntactic information from sentence tokenization, parsing and deep dependency analysis is combined using kernel methods. At each level, kernel functions (or kernels) are developed to represent the syntactic information. Five kernels have been developed for this task, including two at the surface level, one at the parsing level and two at the deep dependency level. Our experiments show that each level of processing may contribute useful clues for this task, including surface information like word bigrams. Adding kernels one by one continuously improves performance. The experiments were carried out on the ACE RDR (Relation Detection and Recognition) task with annotated entities. Using SVM as a classifier along with the full composite kernel produced the best performance on this task. This paper will also show a comparison of SVM and KNN (k-Nearest-Neighbors) under different kernel setups.

2 Kernel Methods

Many machine learning algorithms involve only the dot product of vectors in a feature space, in which each vector represents an object in the object domain. Kernel methods (Muller et al., 2001) can be seen as a generalization of feature-based algorithms, in which the dot product is replaced by a kernel function (or kernel) $\Psi(X,Y)$ between two vectors, or even between two objects. Mathematically, as long as $\Psi(X,Y)$ is symmetric and the kernel matrix formed by Ψ is positive semi-definite, it forms a valid dot product in an implicit Hilbert space. In this implicit space, a kernel can be broken down into features, although the dimension of the feature space could be infinite.

Normal feature-based learning can be implemented in kernel functions, but we can do more than that with kernels. First, there are many well-known kernels, such as polynomial and radial basis kernels, which extend normal features into a high order space with very little computational cost. This could make a linearly non-separable problem separable in the high order feature space. Second, kernel functions have many nice combination properties: for example, the sum or product of existing kernels is a valid kernel. This forms the basis for the approach described in this paper. With these combination properties, we can combine individual kernels representing information from different sources in a principled way.

Many classifiers can be used with kernels. The most popular ones are SVM, KNN, and voted perceptrons. Support Vector Machines (Vapnik, 1998; Cristianini and Shawe-Taylor, 2000) are linear classifiers that produce a separating hyperplane with largest margin. This property gives it good generalization ability in high-dimensional spaces, making it a good classifier for our approach where using all the levels of linguistic clues could result in a huge number of features. Given all the levels of features incorporated in kernels and training data with target examples labeled, an SVM can pick up the features that best separate the targets from other examples, no matter which level these features are from. In cases where an error occurs in one processing result (especially deep processing) and the features related to it become noisy, SVM may pick up clues from other sources which are not so noisy. This forms the basic idea of our approach. Therefore under this scheme we can overcome errors introduced by one processing level; more particularly, we expect accurate low level information to help with less accurate deep level information.

3 Related Work

Collins et al. (1997) and Miller et al. (2000) used statistical parsing models to extract relational facts from text, which avoided pipeline processing of data. However, their results are essentially based on the output of sentence parsing, which is a deep processing of text. So their approaches are vulnerable to errors in parsing. Collins et al. (1997) addressed a simplified task within a confined context in a target sentence.

Zelenko et al. (2003) described a recursive kernel based on shallow parse trees to detect *person-affiliation* and *organization-location* relations, in which a relation example is the least common subtree containing two entity nodes. The kernel matches nodes starting from the roots of two subtrees and going recursively to the leaves. For each pair of nodes, a subsequence kernel on their child nodes is invoked, which matches either contiguous or non-contiguous subsequences of node. Compared with full parsing, shallow parsing is more reliable. But this model is based solely on the out-

put of shallow parsing so it is still vulnerable to irrecoverable parsing errors. In their experiments, incorrectly parsed sentences were eliminated.

Culotta and Sorensen (2004) described a slightly generalized version of this kernel based on dependency trees. Since their kernel is a recursive match from the root of a dependency tree down to the leaves where the entity nodes reside, a successful match of two relation examples requires their entity nodes to be at the same depth of the tree. This is a strong constraint on the matching of syntax so it is not surprising that the model has good precision but very low recall. In their solution a bag-of-words kernel was used to compensate for this problem. In our approach, more flexible kernels are used to capture regularization in syntax, and more levels of syntactic information are considered.

Kambhatla (2004) described a Maximum Entropy model using features from various syntactic sources, but the number of features they used is limited and the selection of features has to be a manual process.[1] In our model, we use kernels to incorporate more syntactic information and let a Support Vector Machine decide which clue is crucial. Some of the kernels are extended to generate high order features. We think a discriminative classifier trained with all the available syntactic features should do better on the sparse data.

4 Kernel Relation Detection

4.1 ACE Relation Detection Task

ACE (Automatic Content Extraction)[2] is a research and development program in information extraction sponsored by the U.S. Government. The 2004 evaluation defined seven major types of relations between seven types of entities. The entity types are PER (Person), ORG (Organization), FAC (Facility), GPE (Geo-Political Entity: countries, cities, etc.), LOC (Location), WEA (Weapon) and VEH (Vehicle). Each mention of an entity has a mention type: NAM (proper name), NOM (nominal) or PRO (pronoun); for example *George W. Bush*, *the president* and *he* respectively. The seven relation types are EMP-ORG (Employment/Membership/Subsidiary), PHYS (Physical), PER-SOC (Personal/Social), GPE-AFF (GPE-Affiliation), Other-AFF (Person/ORG Affiliation), ART (Agent-Artifact) and DISC (Discourse). There are also 27 relation subtypes defined by ACE, but this paper only focuses on detection of relation types. Table 1 lists examples of each relation type.

Type	Example
EMP-ORG	*the CEO of Microsoft*
PHYS	*a military base in Germany*
GPE-AFF	*U.S. businessman*
PER-SOC	*a spokesman for the senator*
DISC	*many of these people*
ART	*the makers of the Kursk*
Other-AFF	*Cuban-American people*

Table 1. ACE relation types and examples. The heads of the two entity arguments in a relation are marked. Types are listed in decreasing order of frequency of occurrence in the ACE corpus.

Figure 1 shows a sample newswire sentence, in which three relations are marked. In this sentence, we expect to find a PHYS relation between *Hezbollah forces* and *areas*, a PHYS relation between *Syrian troops* and *areas* and an EMP-ORG relation between *Syrian troops* and *Syrian.* In our approach, input text is preprocessed by the Charniak sentence parser (including tokenization and POS tagging) and the GLARF (Meyers et al., 2001) dependency analyzer produced by NYU. Based on treebank parsing, GLARF produces labeled deep dependencies between words (syntactic relations such as logical subject and logical object). It handles linguistic phenomena like passives, relatives, reduced relatives, conjunctions, etc.

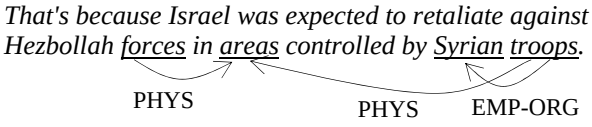

Figure 1. Example sentence from newswire text

4.2 Definitions

In our model, kernels incorporate information from

[1] Kambhatla also evaluated his system on the ACE relation detection task, but the results are reported for the 2003 task, which used different relations and different training and test data, and did not use hand-annotated entities, so they cannot be readily compared to our results.

[2] Task description: http://www.itl.nist.gov/iad/894.01/tests/ace/
ACE guidelines: http://www.ldc.upenn.edu/Projects/ACE/

tokenization, parsing and deep dependency analysis. A relation candidate *R* is defined as

$R = (arg_1, arg_2, seq, link, path)$,

where arg_1 and arg_2 are the two entity arguments which may be related; $seq=(t_1, t_2, \ldots, t_n)$ is a token vector that covers the arguments and intervening words; $link=(t_1, t_2, \ldots, t_m)$ is also a token vector, generated from *seq* and the parse tree; *path* is a dependency path connecting arg_1 and arg_2 in the dependency graph produced by GLARF. *path* can be empty if no such dependency path exists. The difference between *link* and *seq* is that *link* only retains the "important" words in *seq* in terms of syntax. For example, all noun phrases occurring in *seq* are replaced by their heads. Words and constituent types in a stop list, such as time expressions, are also removed.

A token *T* is defined as a string triple,

$T = (word, pos, base)$,

where *word*, *pos* and *base* are strings representing the word, part-of-speech and morphological base form of *T*. Entity is a token augmented with other attributes,

$E = (tk, type, subtype, mtype)$,

where *tk* is the token associated with *E*; *type*, *subtype* and *mtype* are strings representing the entity type, subtype and mention type of *E*. The subtype contains more specific information about an entity. For example, for a GPE entity, the subtype tells whether it is a country name, city name and so on. Mention type includes NAM, NOM and PRO.

It is worth pointing out that we always treat an entity as a single token: for a nominal, it refers to its head, such as *boys* in *the two boys*; for a proper name, all the words are connected into one token, such as *Bashar_Assad*. So in a relation example *R* whose *seq* is $(t_1, t_2, \ldots, t_n)$, it is always true that $arg_1=t_1$ and $arg_2=t_n$. For names, the base form of an entity is its ACE type (person, organization, etc.). To introduce dependencies, we define a dependency token to be a token augmented with a vector of dependency arcs,

$DT=(word, pos, base, dseq)$,

where $dseq = (arc_1, \ldots, arc_n)$. A dependency arc is

$ARC = (w, dw, label, e)$,

where *w* is the current token; *dw* is a token connected by a dependency to *w*; and *label* and *e* are the role label and direction of this dependency arc respectively. From now on we upgrade the type of *tk* in arg_1 and arg_2 to be dependency tokens. Finally, *path* is a vector of dependency arcs,

$path = (arc_1, \ldots, arc_l)$,

where *l* is the length of the path and arc_i $(1 \le i \le l)$ satisfies $arc_1.w=arg_1.tk$, $arc_{i+1}.w=arc_i.dw$ and $arc_l.dw=arg_2.tk$. So *path* is a chain of dependencies connecting the two arguments in *R*. The arcs in it do not have to be in the same direction.

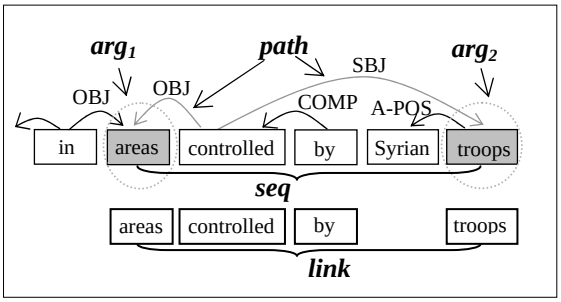

Figure 2. Illustration of a relation example *R*. The *link* sequence is generated from *seq* by removing some unimportant words based on syntax. The dependency links are generated by GLARF.

Figure 2 shows a relation example generated from the text "… *in areas controlled by Syrian troops*". In this relation example *R*, $\boldsymbol{arg_1}$ is *(("areas", "NNS", "area",* ***dseq****), "LOC", "Region", "NOM")*, and $\boldsymbol{arg_1.dseq}$ is *((OBJ, areas, in, 1), (OBJ, areas, controlled, 1))*. $\boldsymbol{arg_2}$ is *(("troops", "NNS", "troop",* ***dseq****), "ORG", "Government", "NOM")* and $\boldsymbol{arg_2.dseq}$ *= ((A-POS, troops, Syrian, 0), (SBJ, troops, controlled, 1))*. ***path*** is *((OBJ, areas, controlled, 1), (SBJ, controlled, troops, 0))*. The value 0 in a dependency arc indicates forward direction from *w* to *dw*, and 1 indicates backward direction. The *seq* and *link* sequences of *R* are shown in Figure 2.

Some relations occur only between very restricted types of entities, but this is not true for every type of relation. For example, PER-SOC is a relation mainly between two person entities, while PHYS can happen between any type of entity and a GPE or LOC entity.

4.3 Syntactic Kernels

In this section we will describe the kernels designed for different syntactic sources and explain the intuition behind them.

We define two kernels to match relation examples at surface level. Using the notation just defined, we can write the two surface kernels as follows:

1) Argument kernel

$$\psi_1(R_1, R_2) = \sum_{i=1,2} K_E(R_1.\arg_i, R_2.\arg_i),$$

where K_E is a kernel that matches two entities,

$$K_E(E_1, E_2) = K_T(E_1.tk, E_2.tk) + I(E_1.type, E_2.type) + I(E_1.subtype, E_2.subtype) + I(E_1.mtype, E_2.mtype)$$

$$K_T(T_1, T_2) = I(T_1.word, T_2.word) + I(T_1.pos, T_2.pos) + I(T_1.base, T_2.base)$$

K_T is a kernel that matches two tokens. *I(x, y)* is a binary string match operator that gives 1 if x=*y* and 0 otherwise. Kernel Ψ_1 matches attributes of two entity arguments respectively, such as type, subtype and lexical head of an entity. This is based on the observation that there are type constraints on the two arguments. For instance PER-SOC is a relation mostly between two person entities. So the attributes of the entities are crucial clues. Lexical information is also important to distinguish relation types. For instance, in the phrase *U.S. president* there is an EMP-ORG relation between *president* and *U.S.*, while in *a U.S. businessman* there is a GPE-AFF relation between *businessman* and *U.S.*

2) Bigram kernel

$$\psi_2(R_1, R_2) = K_{seq}(R_1.seq, R_2.seq),$$

where

$$K_{seq}(seq, seq') = \sum_{0 \le i < seq.len} \sum_{0 \le j < seq'.len} (K_T(tk_i, tk'_j) + K_T(< tk_i, tk_{i+1} >, < tk'_j, tk'_{j+1} >))$$

Operator $<t_1, t_2>$ concatenates all the string elements in tokens t_1 and t_2 to produce a new token. So Ψ_2 is a kernel that simply matches unigrams and bigrams between the *seq* sequences of two relation examples. The information this kernel provides is faithful to the text.

3) Link sequence kernel

$$\psi_3(R_1, R_2) = K_{link}(R_1.link, R_2.link) = \sum_{0 \le i < \min\_len} K_T(R_1.link.tk_i, R_2.link.tk_i),$$

where *min_len* is the length of the shorter link sequence in R_1 and R_2. Ψ_3 is a kernel that matches token by token between the link sequences of two relation examples. Since relations often occur in a short context, we expect many of them have similar link sequences.

4) Dependency path kernel

$$\psi_4(R_1, R_2) = K_{path}(R_1.path, R_2.path),$$

where

$$K_{path}(path, path') = \sum_{0 \le i < path.len} \sum_{0 \le j < path'.len} ((I(arc_i.label, arc'_j.label) + K_T(arc_i.dw, arc'_j.dw)) \times I(arc_i.e, arc'_j.e)$$

Intuitively the dependency path connecting two arguments could provide a high level of syntactic regularization. However, a complete match of two dependency paths is rare. So this kernel matches the component arcs in two dependency paths in a pairwise fashion. Two arcs can match only when they are in the same direction. In cases where two paths do not match exactly, this kernel can still tell us how similar they are. In our experiments we placed an upper bound on the length of dependency paths for which we computed a non-zero kernel.

5) Local dependency

$$\psi_5(R_1, R_2) = \sum_{i=1,2} K_D(R_1.\arg_i.dseq, R_2.\arg_i.dseq),$$

where

$$K_D(dseq, dseq') = \sum_{0 \le i < dseq.len} \sum_{0 \le j < dseq'.len} (I(arc_i.label, arc'_j.label) + K_T(arc_i.dw, arc'_j.dw)) \times I(arc_i.e, arc'_j.e)$$

This kernel matches the local dependency context around the relation arguments. This can be helpful especially when the dependency path between arguments does not exist. We also hope the dependencies on each argument may provide some useful clues about the entity or connection of the entity to the context outside of the relation example.

4.4 Composite Kernels

Having defined all the kernels representing shallow and deep processing results, we can define composite kernels to combine and extend the individual kernels.

1) Polynomial extension

$$\Phi_1(R_1, R_2) = (\psi_1 + \psi_3) + (\psi_1 + \psi_3)^2 / 4$$

This kernel combines the argument kernel Ψ_1 and link kernel Ψ_3 and applies a second-degree polynomial kernel to extend them. The combination of Ψ_1 and Ψ_3 covers the most important clues for this task: information about the two arguments and the word link between them. The polynomial extension is equivalent to adding pairs of features as

new features. Intuitively this introduces new features like: the subtype of the first argument is a country name and the word of the second argument is *president*, which could be a good clue for an EMP-ORG relation. The polynomial kernel is down weighted by a normalization factor because we do not want the high order features to overwhelm the original ones. In our experiment, using polynomial kernels with degree higher than 2 does not produce better results.

2) Full kernel

$$\Phi_2(R_1,R_2) = \Phi_1 + \alpha\psi_4 + \beta\psi_5 + \chi\psi_2$$

This is the final kernel we used for this task, which is a combination of all the previous kernels. In our experiments, we set all the scalar factors to 1. Different values were tried, but keeping the original weight for each kernel yielded the best results for this task.

All the individual kernels we designed are explicit. Each kernel can be seen as a matching of features and these features are enumerable on the given data. So it is clear that they are all valid kernels. Since the kernel function set is closed under linear combination and polynomial extension, the composite kernels are also valid. The reason we propose to use a feature-based kernel is that we can have a clear idea of what syntactic clues it represents and what kind of information it misses. This is important when developing or refining kernels, so that we can make them generate complementary information from different syntactic processing results.

5 Experiments

Experiments were carried out on the ACE RDR (Relation Detection and Recognition) task using hand-annotated entities, provided as part of the ACE evaluation. The ACE corpora contain documents from two sources: newswire (nwire) documents and broadcast news transcripts (bnews). In this section we will compare performance of different kernel setups trained with SVM, as well as different classifiers, KNN and SVM, with the same kernel setup. The SVM package we used is SVM^{light}. The training parameters were chosen using cross-validation. One-against-all classification was applied to each pair of entities in a sentence. When SVM predictions conflict on a relation example, the one with larger margin will be selected as the final answer.

5.1 Corpus

The ACE RDR training data contains 348 documents, 125K words and 4400 relations. It consists of both nwire and bnews documents. Evaluation of kernels was done on the training data using 5-fold cross-validation. We also evaluated the full kernel setup with SVM on the official test data, which is about half the size of the training data. All the data is preprocessed by the Charniak parser and GLARF dependency analyzer. Then relation examples are generated based these results.

5.2 Results

Table 2 shows the performance of the SVM on different kernel setups. The kernel setups in this experiment are incremental. From this table we can see that adding kernels continuously improves the performance, which indicates they provide additional clues to the previous setup. The argument kernel treats the two arguments as independent entities. The link sequence kernel introduces the syntactic connection between arguments, so adding it to the argument kernel boosted the performance. Setup F shows the performance of adding only dependency kernels to the argument kernel. The performance is not as good as setup B, indicating that dependency information alone is not as crucial as the link sequence.

	Kernel	Performance		
		prec	recall	F-score
A	Argument (Ψ_1)	52.96%	58.47%	55.58%
B	***A + link*** ($\Psi_1+\Psi_3$)	58.77%	71.25%	64.41%*
C	***B-poly*** (Φ_1)	66.98%	70.33%	68.61%*
D	***C + dep*** ($\Phi_1+\Psi_4+\Psi_5$)	69.10%	71.41%	70.23%*
E	***D + bigram*** (Φ_2)	69.23%	70.50%	70.35%
F	***A + dep*** ($\Psi_1+\Psi_4+\Psi_5$)	57.86%	68.50%	62.73%

Table 2. SVM performance on incremental kernel setups. Each setup adds one level of kernels to the previous one except setup F. Evaluated on the ACE training data with 5-fold cross-validation. F-scores marked by * are significantly better than the previous setup (at 95% confidence level).

Another observation is that adding the bigram kernel in the presence of all other level of kernels improved both precision and recall, indicating that it helped in both correcting errors in other processing results and providing supplementary information missed by other levels of analysis. In another experiment evaluated on the nwire data only (about half of the training data), adding the bigram kernel improved F-score 0.5% and this improvement is statistically significant.

Type	KNN ($\Psi_1+\Psi_3$)	KNN (Φ_2)	SVM (Φ_2)
EMP-ORG	75.43%	72.66%	77.76%
PHYS	62.19 %	61.97%	66.37%
GPE-AFF	58.67%	56.22%	62.13%
PER-SOC	65.11%	65.61%	73.46%
DISC	68.20%	62.91%	66.24%
ART	69.59%	68.65%	67.68%
Other-AFF	51.05%	55.20%	46.55%
Total	67.44%	65.69%	70.35%

Table 3. Performance of SVM and KNN (k=3) on different kernel setups. Types are ordered in decreasing order of frequency of occurrence in the ACE corpus. In SVM training, the same parameters were used for all 7 types.

Table 3 shows the performance of SVM and KNN (k Nearest Neighbors) on different kernel setups. For KNN, k was set to 3. In the first setup of KNN, the two kernels which seem to contain most of the important information are used. It performs quite well when compared with the SVM result. The other two tests are based on the full kernel setup. For the two KNN experiments, adding more kernels (features) does not help. The reason might be that all kernels (features) were weighted equally in the composite kernel Φ_2 and this may not be optimal for KNN. Another reason is that the polynomial extension of kernels does not have any benefit in KNN because it is a monotonic transformation of similarity values. So the results of KNN on kernel $(\Psi_1+\Psi_3)$ and Φ_1 would be exactly the same. We also tried different k for KNN and k=3 seems to be the best choice in either case.

For the four major types of relations SVM does better than KNN, probably due to SVM's generalization ability in the presence of large numbers of features. For the last three types with many fewer examples, performance of SVM is not as good as KNN. The reason we think is that training of SVM on these types is not sufficient. We tried different training parameters for the types with fewer examples, but no dramatic improvement obtained.

We also evaluated our approach on the official ACE RDR test data and obtained very competitive scores.[3] The primary scoring metric[4] for the ACE evaluation is a 'value' score, which is computed by deducting from 100 a penalty for each missing and spurious relation; the penalty depends on the types of the arguments to the relation. The value scores produced by the ACE scorer for nwire and bnews test data are 71.7 and 68.0 repectively. The value score on all data is 70.1.[5] The scorer also reports an F-score based on full or partial match of relations to the keys. The unweighted F-score for this test produced by the ACE scorer on all data is 76.0%. For this evaluation we used nearest neighbor to determine argument ordering and relation subtypes.

The classification scheme in our experiments is one-against-all. It turned out there is not so much confusion between relation types. The confusion matrix of predictions is fairly clean. We also tried pairwise classification, and it did not help much.

6 Discussion

In this paper, we have shown that using kernels to combine information from different syntactic sources performed well on the entity relation detection task. Our experiments show that each level of syntactic processing contains useful information for the task. Combining them may provide complementary information to overcome errors arising from linguistic analysis. Especially, low level information obtained with high reliability helped with the other deep processing results. This design feature of our approach should be best employed when the preprocessing errors at each level are independent, namely when there is no dependency between the preprocessing modules. The model was tested on text with annotated entities, but its design is generic. It can work with

[3] As ACE participants, we are bound by the participation agreement not to disclose other sites' scores, so no direct comparison can be provided.

[4] http://www.nist.gov/speech/tests/ace/ace04/software.htm

[5] No comparable inter-annotator agreement scores are available for this task, with pre-defined entities. However, the agreement scores across multiple sites for similar relation tagging tasks done in early 2005, using the value metric, ranged from about 0.70 to 0.80.

noisy entity detection input from an automatic tagger. With all the existing information from other processing levels, this model can be also expected to recover from errors in entity tagging.

7 Further Work

Kernel functions have many nice properties. There are also many well known kernels, such as radial basis kernels, which have proven successful in other areas. In the work described here, only linear combinations and polynomial extensions of kernels have been evaluated. We can explore other kernel properties to integrate the existing syntactic kernels. In another direction, training data is often sparse for IE tasks. String matching is not sufficient to capture semantic similarity of words. One solution is to use general purpose corpora to create clusters of similar words; another option is to use available resources like WordNet. These word similarities can be readily incorporated into the kernel framework. To deal with sparse data, we can also use deeper text analysis to capture more regularities from the data. Such analysis may be based on newly-annotated corpora like PropBank (Kingsbury and Palmer, 2002) at the University of Pennsylvania and NomBank (Meyers et al., 2004) at New York University. Analyzers based on these resources can generate regularized semantic representations for lexically or syntactically related sentence structures. Although deeper analysis may even be less accurate, our framework is designed to handle this and still obtain some improvement in performance.

8 Acknowledgement

This research was supported in part by the Defense Advanced Research Projects Agency under Grant N66001-04-1-8920 from SPAWAR San Diego, and by the National Science Foundation under Grant ITS-0325657. This paper does not necessarily reflect the position of the U.S. Government. We wish to thank Adam Meyers of the NYU NLP group for his help in producing deep dependency analyses.

References

M. Collins and S. Miller. 1997. *Semantic tagging using a probabilistic context free grammar*. In Proceedings of the 6th Workshop on Very Large Corpora.

N. Cristianini and J. Shawe-Taylor. 2000. *An introduction to support vector machines*. Cambridge University Press.

A. Culotta and J. Sorensen. 2004. *Dependency Tree Kernels for Relation Extraction*. In Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics.

D. Gildea and M. Palmer. 2002. *The Necessity of Parsing for Predicate Argument Recognition*. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics.

N. Kambhatla. 2004. *Combining Lexical, Syntactic, and Semantic Features with Maximum Entropy Models for Extracting Relations*. In Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics.

P. Kingsbury and M. Palmer. 2002. *From treebank to propbank*. In Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC-2002).

C. D. Manning and H. Schutze 2002. Foundations of *Statistical Natural Language Processing*. The MIT Press, page 454-455.

A. Meyers, R. Grishman, M. Kosaka and S. Zhao. 2001. *Covering Treebanks with GLARF*. In Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics.

A. Meyers, R. Reeves, Catherine Macleod, Rachel Szekeley, Veronkia Zielinska, Brian Young, and R. Grishman. 2004. *The Cross-Breeding of Dictionaries*. In Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC-2004).

S. Miller, H. Fox, L. Ramshaw, and R. Weischedel. 2000. *A novel use of statistical parsing to extract information from text*. In 6th Applied Natural Language Processing Conference.

K.-R. Müller, S. Mika, G. Ratsch, K. Tsuda and B. Scholkopf. 2001. *An introduction to kernel-based learning algorithms*, IEEE Trans. Neural Networks, 12, 2, pages 181-201.

V. N. Vapnik. 1998. *Statistical Learning Theory*. Wiley-Interscience Publication.

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

Shubin Zhao, Adam Meyers, Ralph Grishman. 2004. *Discriminative Slot Detection Using Kernel Methods*. In the Proceedings of the 20th International Conference on Computational Linguistics.