Machine Learning, 60, 11–39, 2005
2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

Support Vector Learning for Semantic Argument Classification*

SAMEER PRADHAN spradhan@cslr.colorado.edu
KADRI HACIOGLU hacioglu@cslr.colorado.edu
VALERIE KRUGLER† krugler@cslr.colorado.edu
WAYNE WARD whw@cslr.colorado.edu
JAMES H. MARTIN martin@cslr.colorado.edu
DANIEL JURAFSKY‡ jurafsky@cslr.colorado.edu
The Center for Spoken Language Research, University of Colorado, Boulder, Co 80303

Editors: Dan Roth and Pascale Fung

Abstract. The natural language processing community has recently experienced a growth of interest in domain independent shallow semantic parsing—the process of assigning a WHO did WHAT to WHOM, WHEN, WHERE, WHY, HOW etc. structure to plain text. This process entails identifying groups of words in a sentence that represent these semantic arguments and assigning specific labels to them. It could play a key role in NLP tasks like Information Extraction, Question Answering and Summarization. We propose a machine learning algorithm for semantic role parsing, extending the work of Gildea and Jurafsky (2002), Surdeanu et al. (2003) and others. Our algorithm is based on Support Vector Machines which we show give large improvement in performance over earlier classifiers. We show performance improvements through a number of new features designed to improve generalization to unseen data, such as automatic clustering of verbs. We also report on various analytic studies examining which features are most important, comparing our classifier to other machine learning algorithms in the literature, and testing its generalization to new test set from different genre. On the task of assigning semantic labels to the PropBank (Kingsbury, Palmer, & Marcus, 2002) corpus, our final system has a precision of 84% and a recall of 75%, which are the best results currently reported for this task. Finally, we explore a completely different architecture which does not requires a deep syntactic parse. We reformulate the task as a combined chunking and classification problem, thus allowing our algorithm to be applied to new languages or genres of text for which statistical syntactic parsers may not be available.

Keywords: shallow semantic parsing, support vector machines

1. Introduction

Automatic, accurate and wide-coverage techniques that can annotate naturally occurring text with semantic argument structure can facilitate the discovery of patterns of information in large text collections (Wallis & Nelson, 2001; Hearst, 1999). Shallow semantic parsing

*This research was partially supported by the ARDA AQUAINT program via contract OCG4423B and by the NSF via grant IIS-9978025.
†Currently at Stanford University, Stanford, CA, 94305 (vkrugler@stanford.edu)
‡Currently at Stanford University, Stanford, CA, 94305 (Jurafsky@stanford.edu)

produces such an annotation, assigning a simple WHO did WHAT to WHOM, WHEN, WHERE, WHY, HOW, etc. structure to sentences in text. More specifically, when presented with a sentence, a parser should, for each predicate in the sentence, identify and label the predicate's semantic arguments. This process entails identifying groups of words in a sentence that represent these semantic arguments and assigning specific labels to them. This notion of shallow semantic parsing, or case role analysis, has a long history in the computational linguistics literature. For a short review of this history, the reader is referred to Jurafsky and Martin (2000).

In recent work, a number of researchers have cast this problem as a tagging problem and have applied various supervised machine learning techniques to it (Gildea, & Jurafsky 2000; Blaheta & Charniak 2000; Gildea & Jurafsky, 2002; Gildea & Palmer 2002; Gildea & Hockenmaier 2003; Surdeanu et al. 2003; Chen & Rambow 2003; Fleischman & Hovy 2003; Hacioglu & Ward 2003; Thompson Levy, & Manning, 2003; Pradhan et al., 2003, 2004). In this paper, we report on a series of experiments exploring this approach.

For our initial experiments, we adopted the approach described by Gildea and Jurafsky (2002) and evaluated a series of modifications to improve performance. We first replaced their statistical classification algorithm with one that uses Support Vector Machines and compared performance using the original set of features. We then evaluated the utility of a series of additional features. All of the experiments we report were conducted on PropBank, an approximately 300k-word corpus annotated with semantic arguments of verbs. We evaluated the system on three main tasks: (1) identifying phrases that represent semantic arguments, (2) independently classifying phrases known to be semantic arguments into the specific categories and (3) the combined task of identifying the arguments and then assigning respective labels to them. We evaluated results using both hand-corrected syntactic parses, and actual parses produced by a syntactic parser (Charniak, 2001).

The Gildea and Jurafsky algorithm assumes the existence of a full syntactic parse. Many of the features for classification are derived from the parse, and the algorithm proceeds by classifying the nodes of the parse tree. The requirement of processing by a syntactic parser limits the applicability of the system to languages and genres of text for which syntactic parsers are available. We reformulated the semantic role annotation task as a combined chunking and classification problem. Retaining the same classifier, we replaced features derived from the hierarchical structure of the syntactic parse with ones derived from a flat chunked representation.

Finally, we conducted an experiment to test the sensitivity of both sets of features to changes in genre of the text. Both the syntactic parser and semantic classifier were trained on a corpus from the Wall Street Journal. We manually tagged about 400 sentences from the AQUAINT corpus (LDC, 2002), which is a collection of text from the New York Times Inc., Associated Press Inc., and Xinhua News Service. We evaluated the systems trained on the WSJ corpus on the test data from AQUAINT.

Section 2 introduces the semantic annotation schema and corpora used for the task. In Section 3 we discuss the semantic parsing system that makes use of features extracted from a full syntactic parse, and which assumes that the semantic argument boundaries align with those of the syntactic parser. In Section 4 we discuss in detail the system that does

Table 1. Argument labels associated with the predicate *operate* (sense: work) in the PropBank corpus.

Tag	Description
ARG0	Agent, operator
ARG1	Thing operated
ARG2	Explicit patient (thing operated on)
ARG3	Explicit argument
ARG4	Explicit instrument

not depend on a full syntactic parse, by formulating the semantic parsing problem as a chunking and classification task.

2. Semantic annotation and corpora

Two corpora are available for developing and testing semantic argument annotation—FrameNet[1] (Baker, Fillmore, & Lowe, 1998) and PropBank[2] (Kingsbury, Palmer, & Marcus, 2002). FrameNet uses predicate specific labels such as JUDGE and JUDGEE. PropBank uses predicate independent labels—ARG0, ARG1, etc. In this paper, we will be reporting on results using PropBank, a 300k-word corpus in which predicate argument relations are marked for almost all occurrences of non-copula verbs in the Wall Street Journal (WSJ) part of the Penn TreeBank (Marcus et al., 1994). The arguments of a verb are labeled sequentially from ARG0 to ARG5, where ARG0 is the PROTO-AGENT (usually the subject of a transitive verb) ARG1 is the PROTO-PATIENT (usually its direct object), etc. PropBank does attempt to treat semantically related verbs consistently. In addition to these "core arguments," additional "adjunctive arguments," referred to as ARGMs are also marked. Some examples are ARGM-LOC, for locatives, and ARGM-TMP, for temporals. We will refer to these as ARGMs. Table 1 shows the argument labels associated with the predicate *operate* in PropBank.

Following is an example structure extracted from the PropBank corpus. The syntax tree representation along with the argument labels is shown in Figure 1.

[$_{\textbf{ARG0}}$ It] [$_{\textbf{predicate}}$ operates] [$_{\textbf{ARG1}}$ stores] [$_{\textbf{ARGM-LOC}}$ mostly in Iowa and Nebraska].

The adjunctive arguments are shown in Table 2. The PropBank annotation scheme assumes that a semantic argument of a predicate aligns with one or more nodes in the hand-corrected (hence TreeBank hand-corrected) parses. Although, most frequently the arguments are identified by one node in the tree, there can be cases where the arguments are discontinuous and more than one nodes are required to identify parts of the arguments as in Figure 2.

Sometimes the tree can have *trace* nodes which refer to another node in the tree, but do not have any words associated with them. These can also be marked as arguments. As traces are not reproduced by a syntactic parser, we decided not to consider them in our experiments—whether or not they represent arguments of a predicate. PropBank also contains arguments that are coreferential.

Table 2. List of adjunctive arguments in PropBank—ARGMS.

Tag	Description	Examples
ARGM-LOC	Locative	*the museum, in Westborough, Mass.*
ARGM-TMP	Temporal	*now, by next summer*
ARGM-MNR	Manner	*heavily, clearly, at a rapid rate*
ARGM-DIR	Direction	*to market, to Bangkok*
ARGM-CAU	Cause	*In response to the ruling*
ARGM-DIS	Discourse	*for example, in part, Similarly*
ARGM-EXT	Extent	*at $38.375, 50 points*
ARGM-PRP	Purpose	*to pay for the plant*
ARGM-NEG	Negation	*not, n't*
ARGM-MOD	Modal	*can, might, should, will*
ARGM-REC	Reciprocals	*each other*
ARGM-PRD	Secondary Predication	*to become a teacher*
ARGM	Bare ArgM	*with a police escort*
ARGM-ADV	Adverbials	*(none of the above)*

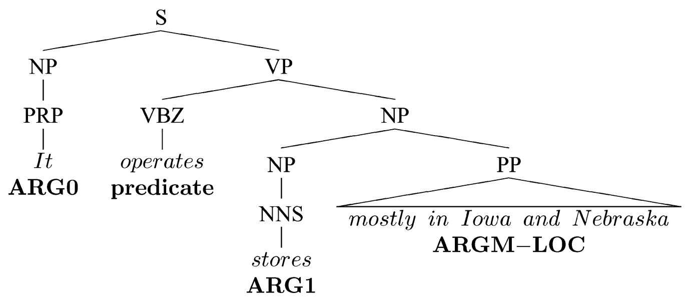


Figure 1. Syntax tree for a sentence illustrating the PropBank tags.

Most of the experiments in this paper are performed on the July 2002 release of PropBank. For these set of experiments, we treated discontinuous arguments as two instances of the same argument. We also do not differentiate between coreferential and non-coreferential arguments. Recently, a newer, larger, completely adjudicated Feb 2004 version of PropBank was released. We also report our best system performance on this dataset. In this dataset, we treat discontinuous and coreferential arguments in accordance to the CoNLL 2004 shared task on semantic role labeling. The first part of a discontinuous argument is labeled as it is, while the second part of the argument is labeled with a prefix "`C-`"appended to it. All coreferential arguments are labeled with a prefix "`R-`" appended to them.

We follow the standard convention of using Section 02 to Section 21 as the training set, Section-00 as the dev-test set and Section-23 as the test set. For the July 2002 release of PropBank, the training set comprises about 51,000 sentences, instantiating about 132,000

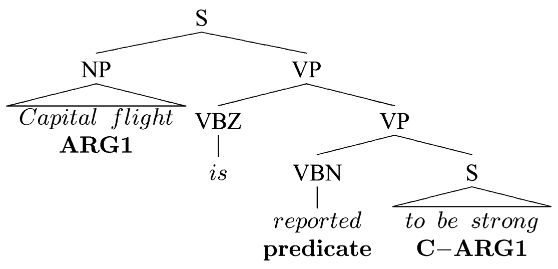


Figure 2. Syntax tree for a sentence illustrating a continuation argument.

arguments, and the test set comprises 2,700 sentences instantiating about 7,000 arguments. The Feb 2004 release training set comprises about 85,000 sentences instantiating about 250,000 arguments and the test set comprises 5,000 sentences instantiating about 12,000 arguments.

3. Constituent-by-Constituent semantic parsing

3.1. Problem description

The problem of shallow semantic parsing can be viewed as a two step process. First, is the process of identifying the predicate whose arguments we plan to identify. In our case, this is any non-copula verb in the sentence. The second step is to identify and classify words or phrases that represent the semantic arguments of that predicate. This process can be evaluated on three distinct tasks:

- *Argument Identification*—This is the process of identifying parse constituents in the sentence that represent valid semantic arguments of a given predicate.
- *Argument Classification*—Given constituents known to represent valid arguments of a predicate, assign the appropriate argument labels to them.
- *Argument Identification and Classification*—A combination of the above two tasks, where the system first identifies probable constituents that represent arguments of a predicate, and then assigns them the most likely argument labels.

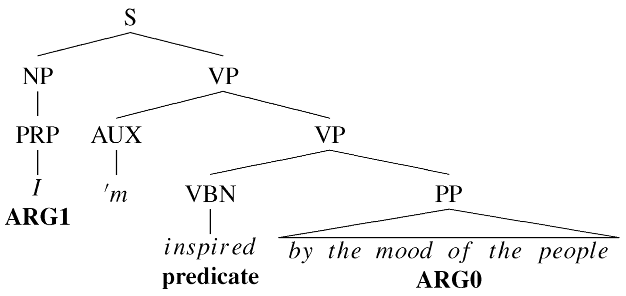


Figure 3. A sample sentence from the PropBank corpus.

Each node in the parse tree can be classified as either one that represents a semantic argument (i.e., a NON-NULL node) or one that does not represent any semantic argument (i.e., a NULL node). The NON-NULL nodes can then be further classified into the set of argument labels. For example, in the tree of Figure 3, the PP that encompasses "*by the mood of the people*" is a NON-NULL node because it corresponds to a semantic argument—ARG 0. The node NP that encompasses "*the people*" is a NULL node, since it does not correspond to a semantic argument.

3.2. Baseline features

Our baseline system uses the same set of features introduced by Gildea and Jurafsky (2002): predicate, path, phrase type, position, voice, head word, and verb sub-categorization. All these features (except one—the predicate) are extracted from the syntactic parse tree of the sentence. Some of the features, viz., predicate, voice and verb sub-categorization are shared by all the nodes in the tree. All the others change with the constituent under consideration. For evaluation purposes only predicates for which the arguments were annotated in PropBank were considered. In the actual system, all non-copula verbs are considered as predicates.

- *Predicate*: The predicate lemma is used as a feature.
- *Path:* The syntactic path through the parse tree from the parse constituent to the predicate being classified.
  For example, in Figure 4, the path from ARG0—"The lawyers" to the predicate "went", is represented with the string NP$\uparrow$S$\downarrow$VP$\downarrow$VBD. $\uparrow$ and $\downarrow$ represent upward and downward movement in the tree respectively.
- *Phrase Type*: This is the syntactic category (NP, PP, S, etc.) of the phrase corresponding to the semantic argument.
- *Position*: This is a binary feature identifying whether the phrase is before or after the predicate.
- *Voice*: Whether the predicate is realized as an active or passive construction. We run a set of hand-written `tgrep` expressions on the syntax tree to identify passive voiced predicates. In approximately 11% of the sentences in PropBank, the predicate had a passive instantiation.

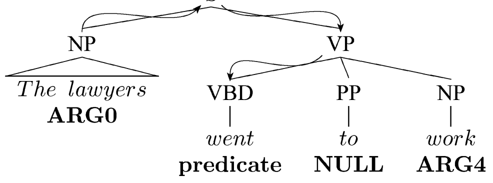


Figure 4. Illustration of path NP$\uparrow$S$\downarrow$VP$\downarrow$VBD.

- *Head Word*: The syntactic head of the phrase. This is calculated using a head word table described by Magerman (1994) and modified by Collins (1999) [Appendix. A]. This feature is not case sensitive.
- *Sub-categorization*: This is the phrase structure rule expanding the predicate's parent node in the parse tree. For example, in Figure 4, the sub-categorization for the predicate "went" is VP $\rightarrow$ VBD-PP-NP.

3.3. Classifier

Support Vector Machines (SVMs) (Vapnik, 1998; Burges, 1998) have been shown to perform well on text classification tasks, where data is represented in a high dimensional space using sparse feature vectors (Joachims, 1998; Kudo & Matsumoto, 2000; Lodhi et al., 2002). We formulate the parsing problem as a multi-class classification problem and use a Support Vector Machine (SVM) classifier (Hacioglu et al., 2003; Pradhan et al., 2003). However, SVMs are binary classifiers. There are two common approaches for extending SVMs to multi-class classification problems (Allwein, Schapire, & Singer, 2000). The first is known as the PAIRWISE approach, where a separate binary classifier is trained for each of the class pairs and their outputs are combined to predict the classes. This approach requires the training of $\frac{N(N-1)}{2}$ binary classifiers. The second, known as the ONE *vs* ALL (OVA) approach, involves training n classifiers for a n-class problem. The classifiers are trained to discriminate between examples of each class, and those belonging to all other classes combined. During testing, the classifier scores on an example are combined to predict its class label.

While some experiments have been reported that the pairwise approach outperforms the OVA approach (Kressel, 1999), our initial experiments show better performance for OVA. Therefore, in this paper, all results are using the OVA classification approach.

3.4. System implementation

The system can be viewed as comprising two stages—the training stage and the testing stage. We will first discuss how the SVM is trained for this task. Since the training time taken by SVMs scales exponentially with the number of examples, and about 90% of the nodes in a syntactic tree have NULL argument labels, we found it efficient to divide the training process into two stages:

1. Filter out the nodes that have a very high probability of being NULL. A binary NULL *vs* NON-NULL classifier is trained on the entire dataset. A sigmoid function is fitted to the raw scores to convert the scores to probabilities as described by Platt (2000). All the training examples are run through this classifier and the respective scores for NULL and NON-NULL assignments are converted to probabilities using the sigmoid function. Nodes that are most likely NULL (probability > 0.90) are pruned from the training set. This reduces the number of NULL nodes by about 90%. This is accompanied by a very negligible (about 1%) pruning of nodes that are NON-NULL.

Table 3. Baseline performance on all the three tasks using hand-corrected parses.

PROPBANK ARGUMENTS	P	R	F_1	A
Identification	91.2	89.6	90.4	
Classification	–	–	–	87.1
Identification + Classification	83.3	78.5	80.8	

2. The remaining training data is used to train a OVA classifier that contains all the classes along with the NULL class.

With this strategy only one classifier (NULL *vs* NON-NULL) has to be trained on all of the data. The remaining OVA classifiers are trained on the nodes passed by the filter (approximately 20% of the total), resulting in a considerable savings in training time.

In the testing stage, we do not perform any filtering of NULL nodes in a first pass. Instead, all the nodes are classified directly as NULL or one of the arguments using the classifier trained in step 2 above. We observe a slight decrease in recall if we filter the test examples using a NULL *vs* NON-NULL classifier in a first pass, as we do in the training process.

For our experiments, we used TinySVM[3] along with YamCha[4] (Kudo & Matsumoto, 2000,2001) as the SVM training and test software. The system uses a polynomial kernel with degree 2; the cost per unit violation of the margin, $C=1$; and, tolerance of the termination criterion, $e = 0.001$.

3.5. Baseline system performance

Table 3 shows the baseline performance numbers on all the three tasks mentioned earlier in Section 3.1 using the PropBank corpus with hand-corrected parses.[5] The features listed in Section 3.2 were used.

For the argument identification and the combined identification and classification tasks, we report the precision (P), recall (R) and the F_1[6] scores, and for the argument classification task we report the classification accuracy (A). This test set and all test sets, unless noted otherwise are Section-23 of PropBank.

3.6. System improvements

3.6.1. Disallowing overlaps. The system as described above might label two constituents even if they overlap in words. This is a problem since we would like to have each word to belong to at most one semantic argument as constrained by the PropBank labeling process. We propose to choose among the overlapping constituents by retaining the one for which the SVM has the highest confidence, and labeling the others NULL. Since we are dealing with strictly hierarchical trees, nodes overlapping in words always have a ancestor-descendant relationship, and therefore the overlaps are restricted to subsumptions only. The probabilities obtained by applying the sigmoid function to the raw SVM scores are used as the measure of confidence. Table 4 shows the performance improvement on the task of

Table 4. Improvements on the task of argument identification and classification using hand-corrected parses.

SYSTEM	P	R	F_1
Baseline	83.3	78.5	80.8
No. overlaps	85.4	78.1	*81.6

identifying and labeling semantic arguments using the hand-corrected parses. In this system, the overlap-removal decisions are taken independently of each other. For this experiment and all other experiments where we report on performance improvement, we conduct a χ^2 test of significance to determine whether the difference in number of responses over all the confusion categories (correct, wrong, false positive and false negative) are statistically significant at $p = 0.05$. All the statistically significant improvements are marked with an *.

3.6.2. New features. We tested several new features, two of which were obtained from the literature—named entities in constituents and head word part of speech. The others are novel features.

Named entities in constituents. Surdeanu et al. (2003) reported a performance improvement on classifying the semantic role of the constituent by relying on the presence of a named entity in the constituent. We expect that some of these entities such as location and time are particularly important for the adjunctive arguments ARGM-LOC and ARGM-TMP. We evaluated a similar use of named entities in our system. We tagged 7 named entities (PERSON, ORGANIZATION, LOCATION, PERCENT, MONEY, TIME, DATE) using IdentiFinder (Bikel, Schwartz, & Weischedel, 1999) and added them as 7 binary features.

Head word part of speech. Surdeanu et al. (2003) showed that adding the part of speech of the head word of a constituent as a feature in the task of argument identification gave a significant performance boost to their decision tree based system. We added this feature to our system.

Verb clustering. The predicate is one of the most salient features in predicting the argument class. Since our training data is relatively limited, any real world test set will contain predicates that have not been seen in training. In these cases, we can benefit from some information about the predicate by using predicate cluster as a feature. The distance function used for clustering is based on the intuition that verbs with similar semantics will tend to have similar direct objects. For example, verbs such as "eat", "devour", "savor", will tend to all occur with direct objects describing food. The clustering algorithm uses a database of verb-direct-object relations extracted by Lin (1998). The verbs were clustered into 64 classes using the probabilistic co-occurrence model of Hofmann and Puzicha (1998). We then use the verb class of the current predicate as a feature.

Generalizing the path feature. As we will see in Section 3.11, for the argument identification task, path is one of the most salient features. However, it is also the most data sparse feature. To overcome this problem, we tried generalizing the path in three different ways:

Table 5. Argument labels associated with the two senses of predicate *talk* in PropBank corpus.

	Sense 1: speak		Sense 2: persuade/dissuade	
Talk	Tag	Description	Tag	Description
	ARG0	Talker	ARG0	Talker
	ARG1	Subject	ARG1	Talked to
	ARG2	Hearer	ARG2	Secondary action

1. Compressing sequences of identical labels into one following the intuition that successive embedding of the same phrase in the tree might not add additional information.
2. Removing the direction in the path, thus making insignificant the point at which it changes direction in the tree, and
3. Using only that part of the path from the constituent to the lowest common ancestor of the predicate and the constituent—"Partial Path". For example, the partial path for the path illustrated in Figure 4 is NP↑S.

Verb sense information. The arguments that a predicate can take depend on the word sense of the predicate. Each predicate tagged in the PropBank corpus is assigned a separate set of arguments depending on the sense in which it is used. Table 5 illustrates the argument sets for the predicate *talk*. Depending on the sense of the predicate *talk*, either ARG1 or ARG2 can identify the *hearer*. Absence of this information can be potentially confusing to the learning mechanism.

We added the oracle sense information extracted from PropBank, to our features by treating each sense of a predicate as a distinct predicate.

Head word of prepositional phrases. Many adjunctive arguments, such as temporals and locatives, occur as prepositional phrases in a sentence, and it is often the case that the head words of those phrases, which are always prepositions, are not very discriminative, e.g., "in the city", "in a few minutes", both share the same head word "in" and neither contain a named entity, but the former is ARGM-LOC, whereas the latter is ARGM-TMP. Therefore, we tried replacing the head word of a prepositional phrase, with that of the first noun phrase inside the prepositional phrase. We retained the preposition information by appending it to the phrase type, eg., "PP-in" instead of "PP".

First and last word/POS in constituent. Some arguments tend to contain discriminative first and last words so we tried using them along with their part of speech as four new features.

Ordinal constituent position. In order to avoid false positives of the type where constituents far away from the predicate are spuriously identified as arguments, we added this feature which is a concatenation of the constituent type and its ordinal position from the predicate.

Constituent tree distance. This is a finer way of specifying the already present position feature.

Constituent relative features. These are nine features representing the phrase type, head word and head word part of speech of the parent, and left and right siblings of the constituent in focus. These were added on the intuition that encoding the tree context this way might add robustness and improve generalization.

Temporal cue words. There are several temporal cue words that are not captured by the named entity tagger and were considered for addition as a binary feature indicating their presence.

Dynamic class context. In the task of argument classification, these are dynamic features that represent the hypotheses of at most previous two nodes belonging to the same tree as the node being classified.

3.6.3. Alternative pruning strategies. In the baseline system architecture, we opted for the two step training strategy mentioned in Section 3.4. To evaluate possible performance degradation accompanying the savings in training time, we tested two other pruning strategies. The three strategies in order of increased training time are:

- **Two-pass hard-prune** strategy, which uses a NULL *vs* NON-NULL classifier trained on the entire data in a first pass. All nodes labeled NULL are filtered. And then, ONE *vs* ALL classifiers are trained on the data containing *only* NON-NULL examples. There is no NULL *vs* ALL classifier in the second stage.
- **Two-pass soft-prune** strategy (baseline), uses a NULL *vs* NON-NULL classifier trained on the entire data, filters out nodes with high confidence (probability > 0.90) of being NULL in a first pass and then trains ONE *vs* ALL classifiers on the remaining data *including* the NULL class.
- **One-pass no-prune** strategy that uses all the data to train ONE *vs* ALL classifiers—one for each argument including the NULL argument. This has considerably higher training time as compared to the other two.

Table 6 shows performance on the task of identification and classification of PropBank arguments. These performance numbers are computed over the reference examples, and not just the ones that pass the filter. There is no statistically significant difference between the *two-pass soft-prune* strategy, and the *one-pass no-prune* strategy. However, both are better

Table 6. Comparing pruning strategies on the task of argument identification and classification using hand-corrected parses.

	No overlaps		
Pruning strategy	P	R	F_1
Two-pass hard-prune	84	80	*81.9
Two-pass soft-prune	86	81	83.4
One-pass no-prune	87	80	83.3

than the *two-pass hard-prune* strategy. Our initial choice of training strategy was dictated by the following efficiency considerations: (i) SVM training is a convex optimization problem that scales exponentially with the size of the training set, (ii) On an average about 90% of the nodes in a tree are NULL arguments, and (iii) We have to optimize only one classifier on the entire data. We continued to use the *two-pass soft-prune* strategy.

3.7. Argument sequence information

In order to improve the performance of their statistical argument tagger, Gildea and Jurafsky used the fact that a predicate is likely to instantiate a certain set of arguments. We use a similar strategy, with some additional constraints: (i) argument ordering information is retained, and (ii) the predicate is considered as an argument and is part of the sequence. We achieve this by training a trigram language model on the argument sequences, so unlike Gildea and Jurafsky, we can also estimate the probability of argument sets not seen in the training data. We first convert the raw SVM scores to probabilities using a sigmoid function. Then, for each sentence being parsed, we generate an argument lattice using the n-best hypotheses for each node in the syntax tree. We then perform a Viterbi search through the lattice using the probabilities assigned by the sigmoid as the observation probabilities, along with the language model probabilities, to find the maximum likelihood path through the lattice, such that each node is either assigned a value belonging to the PROPBANK ARGUMENTS, or NULL.

The search is constrained in such a way that no two NON-NULL nodes overlap with each other. To simplify the search, we allowed only NULL assignments to nodes having a NULL likelihood above a threshold. While training the language model, we can either use the actual predicate to estimate the transition probabilities in and out of the predicate, or we can perform a joint estimation over all the predicates. We implemented both cases considering two best hypotheses, which always includes a NULL (we add NULL to the list if it is not among the top two). On performing the search, we found that the overall performance improvement was not much different than that obtained by resolving overlaps as mentioned earlier. However, we found that there was an improvement in the CORE ARGUMENT accuracy on the combined task of identifying and assigning semantic arguments, given hand-corrected parses, whereas the accuracy of the ADJUNCTIVE ARGUMENTS slightly deteriorated. This seems to be logical considering the fact that the ADJUNCTIVE ARGUMENTS are not linguistically constrained in any way as to their position in the sequence of arguments, or even the quantity. We therefore decided to use this strategy only for the CORE ARGUMENTS. Although, there was an increase in F_1 score when the language model probabilities were jointly estimated over all the predicates, this improvement is not statistically significant. However, estimating the same using specific predicate lemmas, showed a significant improvement in accuracy. The performance improvement is shown in Table 7.

3.8. Best system performance

It was found that a subset of the features helped argument identification and another subset helped argument classification. We will look at the individual feature contribution in

Table 7. Improvements on the task of argument identification and classification using hand-corrected parses, after performing a search through the argument lattice.

CORE ARGS/ Hand-corrected parses	P	R	F_1
Baseline w/o overlaps	90.5	87.4	88.9
Common predicate	91.2	86.9	89.0
Specific predicate lemma	91.7	87.2	*89.4

Table 8. Best system performance on all three tasks using hand-corrected parses.

		HAND-CORRECTED PARSES			
Classes	Task	P	R	F_1	A
ALL	Id.	95.2	92.5	93.8	
ARGS	Classification	–	–	–	91.0
	Id. + Classification	88.9	84.6	86.7	
CORE	Id.	96.2	93.0	94.6	
ARGS	Classification	–	–	–	93.9
	Id. + Classification	90.5	87.4	88.9	

Section 3.11. The best system is trained by first filtering the most likely nulls using the best NULL *vs* NON-NULL classifier trained using all the features whose argument identification F_1 score is marked in bold in Table 11, and then training a ONE *vs* ALL classifier using the data remaining after performing the filtering and using the features that contribute positively to the classification task—ones whose accuracies are marked in bold in Table 11. Table 8 shows the performance of this system.

3.9. *Using automatic parses*

Thus far, we have reported results using hand-corrected parses. In real-word applications, the system will have to extract features from an automatically generated parse. To evaluate this scenario, we used the Charniak parser (Charniak, 2001) to generate parses for PropBank training and test data. We chose Charniak parser over Collins' parser for two reasons: (i) At the time the source code only for Charniak parser was available and it could be modified to accept data from standard input required for the interactive parsing application that we were developing, and (ii) Preliminary experiments indicated that the Charniak parser was faster than the Collins' parser. The predicate lemma was extracted using the XTAG morphology database[7] (Daniel et al., 1992).

In these generated parses, there are about 6% arguments whose boundaries do not align exactly with any of the hand-generated phrase boundaries. We tried to recover training information in such cases by automatically correcting the slight misalignment caused by a class of parser errors—as done before by Gildea & Jurafsky (2002). In

Table 9. Performance degradation when using automatic parses instead of hand-corrected ones.

		AUTOMATIC PARSES			
CLASSES	TASK	*P*	*R*	F_1	*A*
ALL	Id.	89.3	82.9	86.0	
ARGS	Classification	–	–	–	90.0
	Id. + Classification	84.0	75.3	79.4	
CORE	Id.	92.0	83.3	87.4	
ARGS	Classification	–	–	–	90.5
	Id. + Classification	86.4	78.4	82.2	

Table 10. Best system performance on all three tasks on the latest PropBank data, using hand-corrected parses.

ALL ARGS	TASK	*P*	*R*	F_1	*A*
HAND	Id.	96.2	95.8	96.0	
	Classification	–	–	–	93.0
	Id. + Classification	89.9	89.0	89.4	
AUTOMATIC	Classification	–	–	–	90.1

order to align the automatically generated constituents with the hand generated ones, words were dropped one by one from the right end of a hand-generated constituent until it matched one of the automatically generated constituents, with which it was then aligned. Table 9 shows the performance degradation when automatically generated parses are used.

3.10. Using latest Propbank Data

Owing to the Feb 2004 release of much more and completely adjudicated PropBank data, we have a chance to report our performance numbers on this data set. Table 10 shows the same information as in previous Tables 8 and 9, but generated using the new data. Owing to time limitations, we could not get the results on the argument identification task and the combined argument identification and classification task using automatic parses.

3.11. System analysis

Feature performance. Table 11 shows the effect each feature has on the argument classification and argument identification tasks, when added individually to the baseline. Addition of named entities to the NULL vs NON-NULL classifier degraded its performance. We attribute this to a combination of two things: (i) there are a significant number of constituents that contain named entities, but are not arguments of a particular predicate (the parent of an argument node will also contain the same named entity) therefore this provides a noisy

Table 11. Effect of each feature on the argument classification task and argument identification task, when added to the baseline system.

	Argument Classification	Argument Identification		
Features	*A*	*P*	*R*	F_1
Baseline	87.9	93.7	88.9	91.3
+ Named entities	**88.1**	93.3	88.9	91.0
+ Head POS	**88.6**	94.4	90.1	**92.2**
+ Verb cluster	**88.1**	94.1	89.0	**91.5**
+ Partial path	**88.2**	93.3	88.9	91.1
+ Verb sense	**88.1**	93.7	89.5	**91.5**
+ Noun head PP (only POS)	**88.6**	94.4	90.0	**92.2**
+ Noun head PP (only head)	**89.8**	94.0	89.4	**91.7**
+ Noun head PP (both)	**89.9**	94.7	90.5	**92.6**
+ First word in constituent	**89.0**	94.4	91.1	**92.7**
+ Last word in constituent	**89.4**	93.8	89.4	**91.6**
+ First POS in constituent	**88.4**	94.4	90.6	**92.5**
+ Last POS in constituent	**88.3**	93.6	89.1	91.3
+ Ordinal const. pos. concat.	87.7	93.7	89.2	**91.4**
+ Const. tree distance	**88.0**	93.7	89.5	**91.5**
+ Parent constituent	**87.9**	94.2	90.2	**92.2**
+ Parent head	85.8	94.2	90.5	**92.3**
+ Parent head POS	**88.5**	94.3	90.3	**92.3**
+ Right sibling constituent	**87.9**	94.0	89.9	**91.9**
+ Right sibling head	**87.9**	94.4	89.9	**92.1**
+ Right sibling head POS	**88.1**	94.1	89.9	**92.0**
+ Left sibling constituent	**88.6**	93.6	89.6	**91.6**
+ Left sibling head	86.9	93.9	86.1	89.9
+ Left sibling head POS	**88.8**	93.5	89.3	**91.4**
+ Temporal cue words	**88.6**	–	–	–
+ Dynamic class context	**88.4**	–	–	—

feature for NULL vs. NON-NULL classification. We then tried using this as a feature solely in the task of classifying constituents known to represent arguments, using features extracted from hand-corrected parses. On this task, overall classification accuracy increased from 87.1% to 88.1%. As expected, the most significant improvement was for adjunct arguments like temporals (ARGM-TMP) and locatives (ARGM-LOC) as shown in Table 12. Since the number of locative and temporal arguments in the test set is quite low (<10%) as compared to the core arguments, the increase in performance on these does not boost the overall performance significantly.

Table 12. Improvement in classification accuracies after adding named entity information.

	ARGM-LOC			ARGM-TMP		
	P	*R*	F_1	P	*R*	F_1
Baseline	61.8	56.4	59.0	76.4	81.4	78.8
With named entities	70.7	67.0	*68.8	81.1	83.7	*82.4

Table 13. Performance of various feature combinations on the task of argument classification.

Features	Accuracy
All	91.0
All except Path	90.8
All except Phrase Type	90.8
All except HW and HW-POS	90.7
All except All Phrases	*83.6
All except Predicate	*82.4
All except HW and FW and LW info.	*75.1
Only Path and Predicate	74.4
Only Path and Phrase Type	47.2
Only Head Word	37.7
Only Path	28.0

Adding head word POS as a feature significantly improves both the argument classification and the argument identification tasks. We tried two other ways of generalizing the head word: (i) adding the head word cluster as a feature, and (ii) replacing the head word with a named entity if it belonged to any of the seven named entities mentioned earlier. Neither method showed significant improvement. Table 11 also shows the contribution of replacing the head word and the head word POS separately in the feature where the head of a prepositional phrase is replaced by the head word of the noun phrase inside it. Apparently, a combination of relative features seem to have a significant improvement on either or both the classification and identification tasks, and so do the first and last words in the constituent.

Feature salience. In analyzing the performance of the system, it is useful to estimate the relative contribution of the various feature sets used. Table 13 shows the argument classification accuracies for combinations of features on the training and test set for all PropBank arguments, using hand-corrected parses.

In the upper part of Table 13 we see the degradation in performance by leaving out one feature at a time. The features are arranged in the order of increasing salience. Removing all head word related information has the most detrimental effect on the performance. The lower part of the table shows the performance of some feature combinations by themselves.

Table 14. Performance of various feature combinations on the task of argument identification.

Features	P	R	F_1
All	95.2	92.5	93.8
All except HW	95.1	92.3	93.7
All except Predicate	94.5	91.9	93.2
All except HW and FW and LW info.	91.8	88.5	*90.1
All except Path and Partial Path	88.4	88.9	*88.6
Only Path and HW	88.5	84.3	86.3
Only Path and Predicate	89.3	81.2	85.1

Table 14 shows the feature salience on the task of argument identification. As opposed to the argument classification task, where removing the path has the least effect on performance, on the task of argument identification, removing the path causes the convergence in SVM training to be very slow and has the most detrimental effect on the performance.

Size of training data. One important concern in any supervised learning method is the amount of training examples required for the near optimum performance of a classifier. To address this issue, we trained the classifier on varying amounts of training data. The resulting plots are shown in Figure 5. We approximate the curves by plotting the accuracies at seven data sizes. The first curve from the top indicates the change in F_1 score on the task of argument identification alone. The third curve indicates the F_1 score on the task of argument identification and classification. Both the curves run almost parallel to each other indicating that there is constant loss due to classification errors throughout the data range. We also plotted the precision and recall values for the combined task of identification and classification.

Performance tuning. The expectations of a semantic parser by a real-world application would vary considerably. Some applications might favor precision over recall, and others vice-versa. Since we have a mechanism of assigning confidence to the hypothesis generated by the parser, we can tune its performance accordingly. Table 15 shows the achievable increase in precision with some drop in recall for the task of identifying and classifying semantic arguments using the hand-corrected parses. The arguments that get assigned probability below the confidence threshold are assigned a null label.

3.12. Comparison with similar systems.

We evaluated our system in a number of ways. First, we compare it against 4 other shallow parsers in the literature. In comparing the systems, results are reported for the three types of tasks mentioned earlier. Not all the systems report performance on all these tasks, so we could only compare against tasks for which performance numbers are reported by the different systems. This results in some of the cells in the comparison tables being empty.

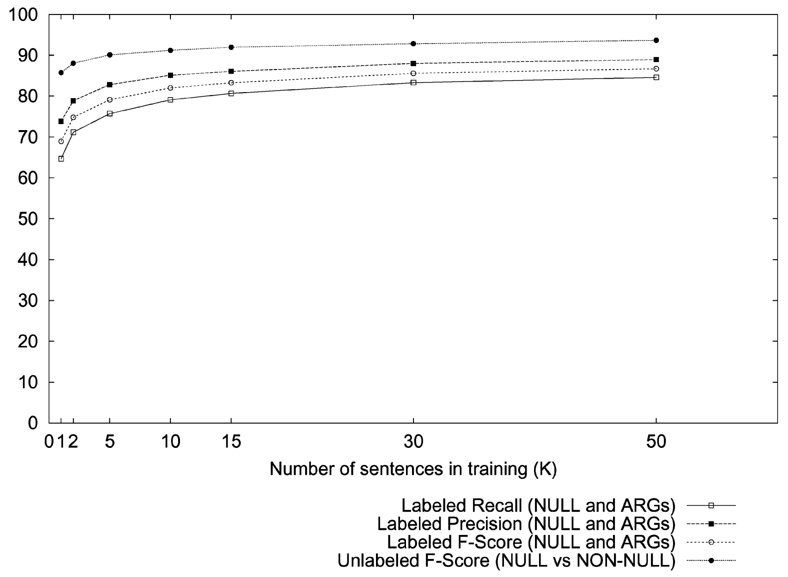


Figure 5. Learning curve for the task of identifying and classifying arguments using hand-corrected parses.

The Gildea and Palmer (G&P) system. Gildea & Palmer (2002) use the same features used by Gildea and Jurafsky (2002), which are also our initial set of features. Two of the features–Head Word and Path—are found to be the most discriminative for argument identification. Maximum likelihood probabilities that the constituent is an argument, based on these two features–$P(is\_argument|Path, Predicate)$ and $P(is\_argument|Head, Predicate)$, are interpolated to calculate the probability that the constituent under consideration represents an argument.

In the second step, the constituents that are retained are assigned a normalized probability that is calculated by interpolating distributions conditioned on various sets of features using a backoff lattice. The most probable argument is selected. They report results on the December 2001 release of PropBank.

Table 15. Precision/Recall table for the combined task of argument identification and classification using hand-corrected parses.

Confidence Threshold	*P*	*R*
0.10	88.9	84.6
0.25	89.0	84.6
0.50	91.7	81.4
0.75	96.7	65.2
0.90	98.7	32.3
0.95	98.5	12.3

The Surdeanu et al. system. Surdeanu et al. (2003) report results for a system that uses the same features as Gildea and Jurafsky (2002) (Surdeanu System I). They then report performance gains by adding additional features (Surdeanu System II) All the feature are combined with a decision tree classifier—C5 (Quinlan, 1986, 2003). Using the built-in boosting capabilities of this classifier show a slight improvement on their baseline performance. They use the July 2002 release of PropBank.

The additional features used by Surdeanu System II are:

- **Content Word:** This is a lexicalized feature that selects the most informative word from a phrase using some heuristics.
- **Part of Speech of the Content Word**
- **Named Entity class of the Content Word**
- **Boolean Named Entity Flags—**The value of these features are set to true or false depending on whether the corresponding named entity is present in the phrase.
- **Phrasal verb collocations—**This feature comprises frequency statistics related to the verb and the immediately following preposition.

The Gildea and Hockenmaier (G&H) system. This system is similar to the Gildea and Palmer (2002) system, but uses features extracted from a CCG grammar, which is a dependency grammar, instead of a phrase structure grammar. The features used are:

- **Path:** This was replaced with the concatenation of the category that the head word belongs to, the slot that it fills and an indicator of whether the word is a categorical functor. When the category information is unavailable, the path through the binary tree from the constituent to the predicate is used.
- **Phrase type:** This is the maximal projection of the PropBank argument's head word in the CCG parse tree.
- **Voice**
- **Head Word**

Gildea and Hockenmaier (2003) report on both the core arguments and the adjunctive arguments on the November 2002 release of the PropBank, which contains approximately 20% more data, so we cannot directly compare to their system. However, since 80% of the data is shared by the two system, and we use the same test set, a comparison is valuable. We will refer to this as "G&H System I".

The Chen and Rambow (C&R) system. Chen and Rambow (2003) also report results using decision tree classifier C4.5 (Quinlan, 1986). They report results using two different sets of features: (i) Surface syntactic features much like the Gildea and Palmer (2002) system, (ii) Additional features that result from the extraction of a Tree Adjoining Grammar (TAG) from the Penn TreeBank. They chose a Tree Adjoining Grammar because of its ability to address long distance dependencies in text. The additional features they introduced are:

- **Supertag Path:** This is the same as the feature "Path" that we saw earlier, but derived from a TAG rather than from a CFG.

Table 16. Argument classification accuracies using same features but different classifiers.

System	Accuracy
SVM	87
Decision tree (Surdeanu et al., 2003)	79
Lattice backoff (Gildea & Palmer, 2002)	77

- **Supertag:** This can be the tree-frame corresponding to the predicate or the argument.
- **Surface syntactic role**.
- **Surface sub-categorization frame**.
- **Deep syntactic role**.
- **Deep sub-categorization frame**.
- **Semantic sub-categorization frame**.

We compare results to the system based on surface syntactic features (C&R System I), and the one that uses deep syntactic features (C&R System II). Unlike all the other systems we discuss, they also report results on core arguments (ARG0-5). Furthermore, owing to the absence of some semantic annotations in the Penn TreeBank, they could extract features for about 87% of the test set. Their numbers are not therefore directly comparable to ours, but since 87% of the test sets are overlapping, a rough comparison is still valuable.

3.13. Comparing classifiers.

Since two systems, in addition to ours, report results using the same set of features on the same data, we can directly assess the influence of the classifiers. Gildea & Hockenmaier, (2003) system estimates the posterior probabilities using several different feature sets and interpolate the estimates, while Surdeanu et al., (2003) use a decision tree classifier. Table 16 shows a comparison between the three systems for the task of argument classification. Keeping the same features and changing only the classifier produces a 8% absolute increase in performance on the same test set.

3.14. Argument identification.

Table 17 compares results on the argument identification task. As in the argument classification task, we see that the SVM system performs better than "Surdeanu System I" using the same set of features. It also outperforms the "Surdeanu System II", which includes some more informative features.

3.15. Argument classification.

Table 18 compares the argument classification accuracies of the systems.

Table 17. Argument identification.

		Hand			Automatic		
Classes	System	P	R	F_1	P	R	F_1
ALL	SVM	95	92	94	89	83	86
ARGS	Surdeanu System II	–	–	89	–	–	–
	Surdeanu System I	85	84	85	–	–	–

Table 18. Argument classification.

Classes	System	HAND Accuracy	AUTOMATIC Accuracy
ALL	SVM	91	90
ARGS	G & P	77	74
	Surdeanu System II	84	–
	Surdeanu System I	79	–
CORE	SVM	93.9	90.5
ARGS	C & R System II	93.5	–
	C & R System I	92.4	–

Table 19. Argument identification and classification.

		Hand			Automatic		
Classes	System	P	R	F_1	P	R	F_1
ALL	SVM	89	85	87	84	75	79
ARGS	G & H System I	76	68	72	71	63	67
	G & P	71	64	67	58	50	54
CORE	SVM System	90	87	89	86	78	82
ARGS	G & H System I	82	79	80	76	73	75
	C & R System II	–	–	–	65	75	70

3.16. Argument identification and classification.

Table 19 shows the results for combined argument identification and classification task.

4. Word-by-Word semantic parsing

4.1. Problem description

In the Constituent-by-Constituent (C-by-C) classification approach the candidate chunks are provided by the full syntactic parse of a sentence. So the segmentation process is performed separately from the classification task. In this method the classification is done on constituents. It is also possible to classify at the word-level. In Ramshaw & Marcus, (1995), chunking was proposed as a tagging task. Here, each word in a sentence is labeled with a tag; I means that the word is inside a chunk, O means that the word is outside a chunk, and B means that the word is the beginning of a chunk. Using a variant of IOB representation, known as IOB2 (Sang & Veenstra, 1999), the words in the semantic chunks for the following example can be tagged as shown:

[$_{\mathbf{ARG1}}$I] 'm [$_{\mathbf{predicate}}$inspired] by [$_{\mathbf{ARG0}}$the mood of the people].
[$_{\mathbf{B-ARG1}}$ I] [$_{\mathbf{O}}$ 'm] [$_{\mathbf{O}}$ inspired] [$_{\mathbf{O}}$ by] [$_{\mathbf{B-ARG0}}$ the][$_{\mathbf{I-ARG0}}$ mood] [$_{\mathbf{I-ARG0}}$ of] [$_{\mathbf{I-ARG0}}$ the] [$_{\mathbf{I-ARG0}}$ people]

In this scheme, N arguments would result in $2 * N + 1$ classes. Experiments using this method have been reported in Hacioglu and Ward (2003) and Hacioglu et al. (2003).

4.2. *Features*

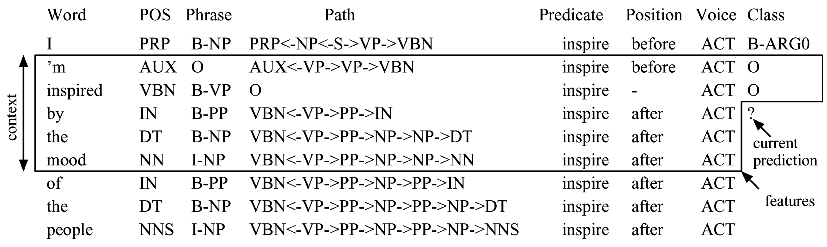
The features are very similar to the features used in the C-by-C system, except the features are derived for each word, not for each constituent. The features are

- **Predicate:** The verb in focus.
- **Part of Speech:** The part-of-speech category of the word.
- **Phrase Position:** The position of the word in a phrase based on IOB representation (e.g. B-NP, I-NP, O etc.)
- **Word Position:** The position of the word with respect to the predicate. It has two values as "before" and "after".
- **Voice:** Indicates whether the sentence is in active or passive form with respect to the predicate.
- **Path:** This is same as the path feature used in the C-by-C system, except the path is relative to the word and not the constituent.

For each word to be tagged, a set of features is created from a fixed size context that surrounds the word. A 5-word sliding window centered at the current word defines the set of features. In addition to the above features, we also use semantic IOB tags that have already been assigned to the words that precede the word in focus and appear in the context. Figure 6 clarifies the notion of context and illustrates the features used. It should be noted that some of the features attached to the words are sentence-level features.

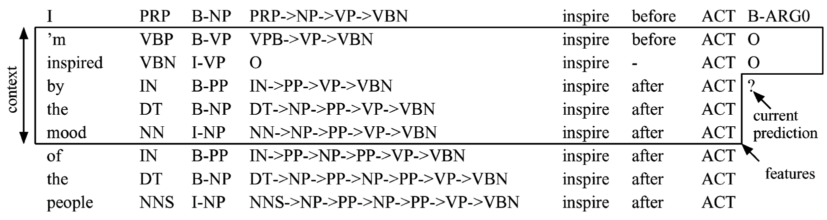
4.3. *System implementation*

As in the C-by-C classification framework we have used SVMs in the ONE *vs* ALL setup. All SVM classifiers are realized using the TinySVM with the polynomial kernel of degree 2 and the general purpose SVM based chunker YamCha as distributed, without any changes. Figures 8 and 9 illustrate two implementations for feature extraction. As in the baseline


context
current prediction
features

Word	POS	Phrase	Path	Predicate	Position	Voice	Class
I	PRP	B-NP	PRP<-NP<-S->VP->VBN	inspire	before	ACT	B-ARG0
'm	AUX	O	AUX<-VP->VP->VBN	inspire	before	ACT	O
inspired	VBN	B-VP	O	inspire	-	ACT	O
by	IN	B-PP	VBN<-VP->PP->IN	inspire	after	ACT	?
the	DT	B-NP	VBN<-VP->PP->NP->NP->DT	inspire	after	ACT	
mood	NN	I-NP	VBN<-VP->PP->NP->NP->NN	inspire	after	ACT	
of	IN	B-PP	VBN<-VP->PP->NP->PP->IN	inspire	after	ACT	
the	DT	B-NP	VBN<-VP->PP->NP->PP->NP->DT	inspire	after	ACT	
people	NNS	I-NP	VBN<-VP->PP->NP->PP->NP->NNS	inspire	after	ACT	

Figure 6. Illustration of the context and the features used to classify a word—Chunker-1


context
current prediction
features

I	PRP	B-NP	PRP->NP->VP->VBN	inspire	before	ACT	B-ARG0
'm	VBP	B-VP	VPB->VP->VBN	inspire	before	ACT	O
inspired	VBN	I-VP	O	inspire	-	ACT	O
by	IN	B-PP	IN->PP->VP->VBN	inspire	after	ACT	?
the	DT	B-NP	DT->NP->PP->VP->VBN	inspire	after	ACT	
mood	NN	I-NP	NN->NP->PP->VP->VBN	inspire	after	ACT	
of	IN	B-PP	IN->PP->NP->PP->VP->VBN	inspire	after	ACT	
the	DT	B-NP	DT->NP->PP->NP->PP->VP->VBN	inspire	after	ACT	
people	NNS	I-NP	NNS->NP->PP->NP->PP->VP->VBN	inspire	after	ACT	

Figure 7. Illustration of the context and the features used to classify a word—Chunker-2.

system, the first system extracts features from a full syntactic parser. Even though classification is done on a word-by-word basis, features are extracted from a full parse. This system is referred to as W-by-W Chunker-1. In the second system, the full-syntactic parser is replaced by a part of speech tagger followed by a syntactic phrase chunker. The POS tagger and syntactic chunker were both implemented using YamCha. They were trained on Sections 15 through 18 of the Penn TreeBank. The POS tagger used the words, word prefixes/suffixes, word types (e.g numeric, capitalized etc.) and previous tag decisions as features. The syntactic chunker used the words, POS tags and previous chunk decisions as features. In both systems, the features were created using a sliding window of size 5 centered at the word to be tagged. The second system is referred to as W-by-W Chunker-2. As mentioned earlier, the motivations for using a chunking parser instead of a full syntactic parser are: (i) full parsing is computationally expensive than chunking, (ii) chunkers can be developed more easily for new languages and genre of text. The features created for these two systems are shown in Figure 7 for the example exhibited in Figure 6. There are slight differences in POS and phrase position features and major differences in the path features. The path feature for W-by-W Chunker 2 is uni-directional (from the word in focus to the predicate) and created using the flat structure:

[**NP** (**PRP** I)] [**VP** (**VBP** 'm) (**VBN** inspired)] [**PP** (**IN** by)] [**NP** (**DT** the) (**NN** mood)] [**PP** (**IN** of)] [**NP** (**DT** the) (**NNS** people)]

The flat path is defined as a chain of phrase chunk labels terminated with the POS tags of the word in focus and predicate. Consecutive chunks with identical labels are

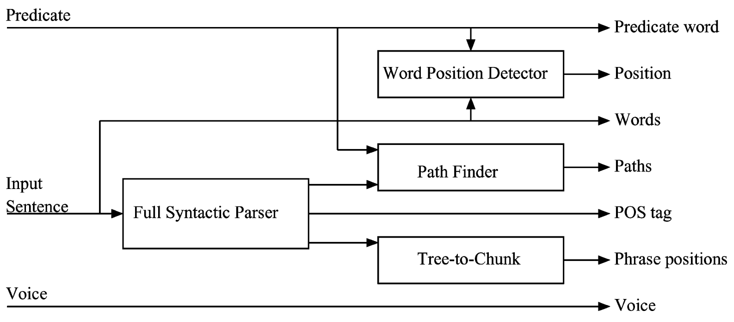


Figure 8. Chunker-1 architecture.

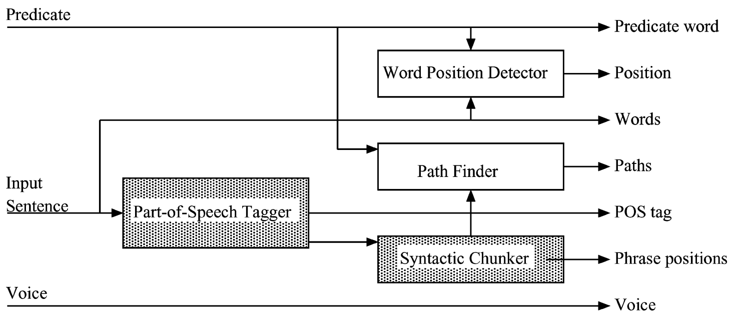


Figure 9. Chunker-2 architecture.

collapsed into one. For example, the path from *people* to the predicate *inspired* is NNS→NP→PP→NP→PP→VP→VBN.

We are experimenting with both implementations to determine trade-offs between coverage, efficiency and accuracy. We present some preliminary results in the next section that compare the accuracy of Chunker-2 to Chunker-1, and also we compare their performance to the C-by-C parser described in Section 3.

4.4. Experimental results

Since training SVM on the entire PropBank corpus takes a long time, these experiments were carried out by training both C-by-C and W-by-W systems on 10,000 randomly selected sentences from the training data. As always, Section-23 was used as the test set. We used the Charniak parser to obtain syntactic parse trees. The results are shown in Table 20.

The entire training set has approximately 1.5 million words. Training SVMs using such data is still prohibitive. Unfortunately, this kept us from carrying out extensive experiments with the W-by-W approach using the entire training set, additional features and other system settings.

When features are derived from a flat chunked parse, instead of a syntactic parse tree, we observed a significant drop in performance. The major difference between these systems

Table 20. Comparison of C-by-C and W-by-W classifiers.

System	P	R	F_1
C-by-C	80.6	67.1	73.2
Chunker-1, W-by-W	70.7	60.5	65.2
Chunker-2, W-by-W	66.2	54.9	60.0

Table 21. Comparison of W-by-W classifiers without the path feature

System	P	R	F_1
Chunker-1, W-by-W	60.4	51.4	55.6
Chunker-2, W-by-W	59.8	50.8	54.9

is the derivation of the path feature. To further illustrate the effect, we ran both systems without using the path feature. These results are shown in Table 21. As can be seen the performance of these systems is very similar, indicating the salience of the path feature. These experiments have also shown that the performance obtained by the W-by-W paradigm fell short of that obtained by the C-by-C paradigm. We think that an important step towards bridging this gap would be to adopt a two pass approach in the W-by-W paradigm, analogous in the C-by-C paradigm. That is, first chunk the sentence into arguments (NON-NULLS) and non-arguments(NULLS), and then label the NON-NULLS with the respective arguments. We are currently working along those directions to improve the performance.

5. Generalization to a new domain

Thus far, all experiments have been tested on the same genre of data that they have been trained on. In order to see how well the features transfer to a new domain, we used both the W-by-W and the C-by-C classifiers trained on PropBank to test data drawn from the AQUAINT corpus (LDC, 2002). We annotated 400 sentences from the AQUAINT corpus with PropBank arguments. This is a collection of text from the New York Times Inc., Associated Press Inc., and Xinhua News Service (PropBank by comparison is drawn from Wall Street Journal). The results of the C-by-C classifier are shown in Table 22.

There is a significant drop in the precision and recall numbers for the AQUAINT test set (compared to the precision and recall numbers for the PropBank test set which were 84% and 75% respectively). One possible reason for the drop in performance is relative coverage of the features on the two test sets. The head word, path and predicate features all have a large number of possible values and could contribute to lower coverage when moving from one domain to another. Also, being more specific, they might not transfer well across domains. All the other features are less likely to have been a source of the problem since they can take small enough values that the amount of training data is sufficient to estimate their statistics.

Table 23 shows the coverage for features on the PropBank test set. The tables show feature coverage for constituents that were Arguments and constituents that were NULL.

Table 22. Performance on the AQUAINT test set.

CLASSES	TASK	P	R	F_1	A
ALL	Id.	75.8	71.4	73.5	–
ARGS	Classification	–	–	–	83.8
	Id. + Classification	65.2	61.5	63.3	–
CORE	Id.	88.4	74.4	80.8	–
ARGS	Classification	–	–	–	84.0
	Id. + Classification	75.2	63.3	68.7	–

Table 23. Feature Coverage on PropBank test set using parser trained on PropBank training set.

Features	Arguments	Non-Arguments
Predicate, Path	87.6	2.9
Predicate, Head Word	48.9	26.5
Cluster, Path	96.3	5.0
Cluster, Head Word	83.8	60.1
Path	99.1	15.1
Head Word	93.0	90.6

Table 24. Coverage of features on AQUAINT test set using parser trained on PropBank training set.

Features	Arguments	Non-Arguments
Predicate, path	62.1	4.7
Predicate, head word	30.3	17.4
Cluster, path	87.2	10.7
Cluster, head word	65.8	45.4
Path	96.5	29.3
Head word	84.6	83.5

About 99% of the predicates in the AQUAINT test set were seen in the PropBank training set. Table 24 shows coverage for the same features on the AQUAINT test set. We believe that the drop in coverage of the more predictive feature combinations explains part of the drop in performance. We also ran the Word-by-Word chunking algorithms on the AQUAINT test set. The results are shown in Table 25. This would compare to the Table 20. We also observed a degradation in the F_1 score for this condition, from 65.2% to 52.2% for Chunker-1 and 60.0% to 45.7% for Chunker-2.

Table 25. Identification and classification performance of the chunking systems on the AQUAINT test set.

Classes	System	*P*	*R*	F_1
All	Chunker-1, W-by-W	54.2	50.4	52.2
Arguments	Chunker-2, W-by-W	49.5	42.4	45.7

6. Conclusions

We have described an algorithm which significantly improves the state-of-the-art in shallow semantic parsing. Like previous work, our parser is based on a supervised machine learning approach. It achieves the best published performance on the task of identifying and labeling semantic arguments in PropBank. The performance of the parser was analyzed along several dimensions: the statistical classifier used, features used, the architecture of the system and the genre of the text. Use of an SVM classifier, compared to the backoff algorithm used by Gildea and Jurafsky, resulted in a significant performance improvement (8% absolute) using the same set of features for classification. New features were added to the system and feature performance was analyzed. We found that path is the most salient feature for argument identification, but is difficult to generalize. Only the partial-path generalization seemed to have any benefit. For the task of classification, head word and predicate were the most salient features, but may be difficult to estimate because of data sparsity. Adding features that are generalizations of the more specific features (named entities, head word part of speech and verb clusters) resulted in some improvement.

We reported on a series of experiments using a different architecture which does not require a syntactic parser. This architecture performed significantly worse than our parser-based architecture, but may offer advantages for new languages or genres in which syntactic parsers are not available.

In analyzing the portability of the system to new genres of data, we found a significant drop in performance for a text source different than the one the system was trained on.

Acknowledgments

We would like to thank Ralph Weischedel and Scott Miller of BBN Inc. for letting us use their named entity tagger–IdentiFinder; Martha Palmer for providing us with the PropBank data and useful advice; Daniel Gildea for helpful discussions, and all the anonymous reviewers for their comments and suggestions.

Notes

1. `http://www.icsi.berkeley.edu/~framenet/`
2. `http://www.cis.upenn.edu/~ace/`
3. `http://cl.aist-nara.ac.jp/~taku-ku/software/TinySVM/`
4. `http://cl.aist-nara.ac.jp/~taku-ku/software/yamcha/`
5. We report on hand-corrected parses because it is easier to compare with results from the literature; we will also report on performance using an actual errorful parser.

6. $F_1 = \frac{2 \times P \times R}{P+R}$
7. `ftp://ftp.cis.upenn.edu/pub/xtag/morph-1.5/morph-1.5.tar.gz`

References

Allwein, E. L., Schapire, R. E., & Singer, Y. (2000). Reducing multiclass to binary: A unifying approach for margin classifiers. In *Proceedings of the 17th International Conference on Machine Learning* (pp. 9–16). San Francisco, CA: Morgan Kaufmann.

Baker, C. F., Fillmore, C. J., & Lowe, J. B. (1998). The Berkeley Framenet Project. In *Proceedings of the International Conference on Computational Linguistics (COLING/ACL-98)*. (pp. 86–90). Montreal.

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

Blaheta, D., & Charniak, E. (2000). Assigning function tags to parsed text. In *Proceedings of the 1st Annual Meeting of the North American Chapter of the ACL(NAACL)* (pp. 234–240). Seattle, Washington.

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery, 2:2*, 121–167.

Charniak, E. (2001). Immediate-head parsing for language models. In *Proceedings of the 39th Annual Conference of the Association for Computational Linguistics (ACL-01)*. Toulouse, France.

Chen, J., & Rambow, O. (2003). Use of deep linguistics features for the recognition and labeling of semantic arguments. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Sapporo, Japan.

Collins, M. J. (1999) Head-driven statistical models for natural language parsing. Ph.D. thesis, University of Pennsylvania, Philadelphia.

Daniel, K., Schabes, Y., Zaidel, M., & Egedi, D.(1992). A freely available wide coverage morphological analyzer for English. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*. Nantes, France.

Fleischman, M., & Hovy, E. (2003). A maximum entropy approach to framenet tagging. In *Proceedings of the Human Language Technology Conference*. Edmonton, Canada.

Gildea, D., & Hockenmaier, J. (2003). Identifying semantic roles using combinatory categorial grammar. In*Proceedings of the Conference on Empirical Methodsin Natural Language Processing*. Sapporo, Japan.

Gildea, D., & Jurafsky, D. (2000). Automatic labeling of semantic roles. In *Proceedings of the 38th Annual Conference of the Association for Computational Linguistics (ACL-00)* (pp. 512–520). Hong Kong.

Gildea, D. & Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational Linguistics, 28:3*, 245–288.

Gildea, D., & Palmer, M. (2002). The necessity of syntactic parsing for predicate argument recognition. In *Proceedings of the 40th Annual Conference of the Association for Computational Linguistics (ACL-02)*. Philadelphia, PA.

Hacioglu, K., Pradhan, S., Ward, W., Martin, J., & Jurafsky, D. (2003). Shallow semantic parsing using support vector machines. Technical Report TR-CSLR-2003-1, Center for Spoken Language Research, Boulder, Colorado.

Hacioglu, K., & Ward, W. (2003). Target word detection and semantic role chunking using support vector machines. In *Proceedings of the Human Language Technology Conference*. Edmonton, Canada.

Hearst, M. (1999). Untangling text data mining. In *Proceedings of the 37th Annual Meeting of the ACL* (pp. 3–10). College Park, Maryland.

Hofmann, T., & Puzicha, J. (1998). Statistical models for co-occurrence data. Memo, Massachusetts Institute of Technology Artificial Intelligence Laboratory.

Joachims, T. (1998) Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning (ECML)*.

Kingsbury, P., Palmer, M., & Marcus, M. (2002). Adding semantic annotation to the Penn Treebank. In *Proceedings of the Human Language Technology Conference*. San Diego, CA.

Kressel, U. H. G. (1999). Pairwise classification and support vector machines. In B. Scholkopf, C. Burges, & A. J. Smola (Eds.), *Advances in kernel methods*. The MIT Press.

Kudo, T., & Matsumoto, Y. (2000). Use of support vector learning for chunk identification. In *Proceedings of the 4th Conference on CoNLL-2000 and LLL-2000* (pp. 142–144).

Kudo, T., & Matsumoto, Y. (2001). Chunking with support vector machines. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-2001).*

LDC: (2002). The AQUAINT Corpus of English News Text, Catalog no. LDC2002T31.

Lin, D. (1998). Automatic retrieval and clustering of similar words. In *Proceedings of the International Conference on Computational Linguistics (COLING/ACL-98).* Montreal, Canada.

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

Magerman, D. (1994). Natural language parsing as statistical pattern recognition. Ph.D. thesis, Stanford University, CA.

Marcus, M., Kim, G., Marcinkiewicz, M.A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., & Schasberger, B. (1994). The Penn treebank: Annotating predicate argument structure.

Platt, J. (2000). Probabilities for support vector machines. In A. Smola, P. Bartlett, B. Scholkopf, & D. Schuurmans (Eds.), *Advances in large margin classifiers*. Cambridge, MA: MIT press.

Pradhan, S., Hacioglu, K., Ward, W., Martin, J., & Jurafsky, D. (2003). Semantic role parsing: Adding semantic structure to unstructured text. In *Proceedings of the International Conference on Data Mining (ICDM 2003).* Melbourne, Florida.

Pradhan, S., Ward, W., Hacioglu, K., Martin, J., & Jurafsky, D. (2004). Shallow Semantic parsing using support vector machines. In *Proceedings of the Human Language Technology Conference/North American chapter of the Association of Computational Linguistics (HLT/NAACL)*. Boston, MA.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1:1*, 81–106.

Quinlan, R. (2003). Data Mining Tools See5 and C5.0. http://www.rulequest.com.

Ramshaw, L. A., & Marcus, M. P. (1995). Text chunking using transformation-based learning. In *Proceedings of the Third Annual Workshop on Very Large Corpora* (pp. 82–94).

Sang, E. F. T. K., & Veenstra, J. (1999). Representing text chunks. In *Proceedingsof the EACL*. (pp. 173–179).

Surdeanu, M., Harabagiu, S., Williams,J., & Aarseth, P. (2003). Using predicate-argument structures for information extraction. In *Proceedings of the 41stAnnual Meeting of the Association for Computational Linguistics*. Sapporo, Japan.

Thompson, C. A., Levy, R., & Manning, C. D. (2003). A generative model for semantic role labeling. In *Proceedings of the European Conference on Machine Learning (ECML).*

Vapnik, V. (1998). *Statistical learning theory* New York: John Wiley and Sons Inc.

Wallis, S., & Nelson, G. (2001). Knowledge discovery in grammatically analysed corpora. *Data Mining and Knowledge Discovery, 5:4*, 305–335.

Received October 8, 2003
Revised March 31, 2004
Accepted May 24, 2004