Also appears as Technical Report UT-AI-05-326, Artificial Intelligence Lab, University of Texas at Austin, November 2005

A Kernel-based Approach to Learning Semantic Parsers

Rohit J. Kate
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
rjkate@cs.utexas.edu

Doctoral Dissertation Proposal

Supervising Professor: Raymond J. Mooney

Abstract

Semantic parsing involves deep semantic analysis that maps natural language sentences to their formal executable meaning representations. This is a challenging problem and is critical for developing user-friendly natural language interfaces to computing systems. Most of the research in natural language understanding, however, has mainly focused on shallow semantic analysis like case-role analysis or word sense disambiguation. The existing work in semantic parsing either lack the robustness of statistical methods or are applicable only to simple domains where semantic analysis is equivalent to filling a single semantic frame.

In this proposal, we present a new approach to semantic parsing based on string-kernel-based classification. Our system takes natural language sentences paired with their formal meaning representations as training data. For every production in the formal language grammar, a Support-Vector Machine (SVM) classifier is trained using string similarity as the kernel. Each classifier then gives the probability of the production covering any given natural language string of words. These classifiers are further refined using EM-type iterations based on their performance on the training data. Meaning representations for novel natural language sentences are obtained by finding the most probable semantic parse using these classifiers. Our experiments on two real-world data sets that have deep meaning representations show that this approach compares favorably to other existing systems in terms of accuracy and coverage.

For future work, we propose to extend this approach so that it will also exploit the knowledge of natural language syntax by using the existing syntactic parsers. We also intend to broaden the scope of application domains, for example, domains where the sentences are noisy as typical in speech, or domains where corpora available for training do not have natural language sentences aligned with their unique meaning representations. We aim to test our system on the task of complex relation extraction as well. Finally, we also plan to investigate ways to combine our semantic parser with some recently developed semantic parsers to form committees in order to get the best overall performance.

Contents

1 Introduction

Computational systems that learn to transform natural language sentences into formal meaning representations have important practical applications in enabling user-friendly natural language communication with computers. They can also provide insights into human language acquisition. However, there has been relatively little research on developing systems that learn such semantic parsers. Most of the research in natural language processing (NLP) has been focused on lower-level tasks like syntactic parsing, word-sense disambiguation, information extraction etc.(Manning & Schütze, 1999). Recently, researchers have shown interest in the task of *semantic role labeling*, a form of "shallow" semantic parsing, which involves filling out semantic roles of a predicate given a sentence (Gildea & Jurafsky, 2002), (Carreras & Marquez, 2004). In this proposal, we have considered an even more ambitious task of deep semantic parsing of sentences into their computer executable meaning representations.

Previous work on semantic parsing has focused on simple domains, primarily, ATIS (Air Travel Information Service) (Price, 1990) whose semantic analysis is equivalent to filling a single semantic frame (Kuhn & De Mori, 1995), (Miller, Stallard, Bobrow, & Schwartz, 1996), (Popescu, Armanasu, Etzioni, Ko, & Yates, 2004). There has also been work in semantic parsing in which meaning representation of the domains are more complex with richer predicates and nested structures (Zelle & Mooney, 1996), (Tang & Mooney, 2001), (Kate, Wong, & Mooney, 2005). But all these systems use deterministic rule-based learning methods and lack in robustness which is the characteristic of learning methods currently used in statistical NLP. Some previous work do not use any learning method (Androutsopoulos, Ritchie, & Thanisch, 1995), (Popescu, Etzioni, & Kautz, 2003) which make them difficult to port to other domains.

In this proposal, we present a novel kernel-based statistical approach to learning semantic parsers. Kernel methods are a powerful new approach to machine learning that have demonstrated success in a wide variety of applications (Shawe-Taylor & Cristianini, 2000). Kernelized support-vector machines (SVMs) is an effective and theoretically well-founded approach to learning non-linear classifiers (Cristianini & Shawe-Taylor, 2000). An additional advantage of kernels is that, unlike most learning methods, they are not restricted to handling feature-based data and can be defined over complex, unbounded data structures such as strings and trees typically encountered in NLP problems. In particular, string and tree kernels have recently been effectively applied to a variety of problems in text learning and NLP (Collins, 2002), (Lodhi, Saunders, Shawe-Taylor, Cristianini, & Watkins, 2002), (Zelenko, Aone, & Richardella, 2003), (Cumby & Roth, 2003), (Culotta & Sorensen, 2004), (Bunescu & Mooney, 2005b) and (Bunescu & Mooney, 2005a).

Kernel methods are particularly suitable for semantic parsing because semantic parsing involves mapping phrases of natural language (NL) sentences to semantic concepts in meaning representation language (MRL). Given that natural languages are so flexible, there could be various ways in which one can express the same semantic concept. It is difficult for rule-based methods or even statistical feature-based methods to capture the range of NL contexts which map to a semantic concept because they tend to enumerate these contexts. In contrast, kernel methods allow a convenient mechanism to implicitly work with potentially infinite number of features which can robustly capture these range of contexts.

Our system, KRISP (Kernel-based Robust Interpretation by Semantic Parsing), takes NL sentences paired with their formal meaning representations as training data. The productions of the formal MRL are treated like semantic concepts. For each of these productions, a Support-Vector Machine (SVM) classifier is trained using string similarity as the kernel. Each classifier then indicates the probability of the production covering different substrings of the sentence. This information is used to compositionally build a complete meaning representation (MR) of the sentence. We demonstrate through experiments on two real-world data sets that KRISP compares favorably to other existing systems.

We plan to extend our work in the following directions:

1. Currently our system uses a string-based kernel which does not exploit the NL syntax. We plan to use tree kernels based on syntactic trees (Collins, 2002) or dependency trees (Zelenko et al., 2003) in future.

2. We aim to test KRISP on noisy NL sentences, for e.g. as typically generated in speech, in which case we believe KRISP's robustness will be valuable.

3. We plan to investigate ways to form committees of KRISP and some recently built semantic parsers to improve performance.

4. We plan to broaden the applicability of KRISP to more realistic domains where in the data available for training the NL sentences may not aligned with their meaning representations.

5. We also want to test KRISP on the problem of Complex Relation Extraction (McDonald, Pereira, Kulick, Winters, Jin, & White, 2005a), which involves extracting related named entities from a sentence. This can be viewed as a less deeper semantic parsing on which we believe KRISP could be applied.

The remainder of this proposal is organized as follows: Section 2 describes the semantic parsing task including the related work in this area. It also briefly reviews kernel-based machine learning, SVMs and the string-subsequence kernel we use in our system. Section 3 describes our new semantic parsing algorithm, KRISP, and presents its experimental evaluations. In Section 4, we outline our plans for future research.

2 Background and Related Work

2.1 Semantic Parsing

Semantic parsing is the process of mapping natural language (NL) utterances into their computer understandable meaning representations (MRs). These MRs are expressed in formal languages which we call meaning representation languages (MRLs). We assume that all MRLs have deterministic context free grammars, which is true for almost all computer languages. This ensures that every MR will have a unique parse tree. A learning system for semantic parsing is given a training corpus of NL sentences paired with their respective MRs from which it has to induce a semantic parser which can map novel NL sentences to their correct MRs. This section describes the three application domains on which the research in semantic parsing has mainly been focused. Next, some existing approaches for learning semantic parsers are briefly described.

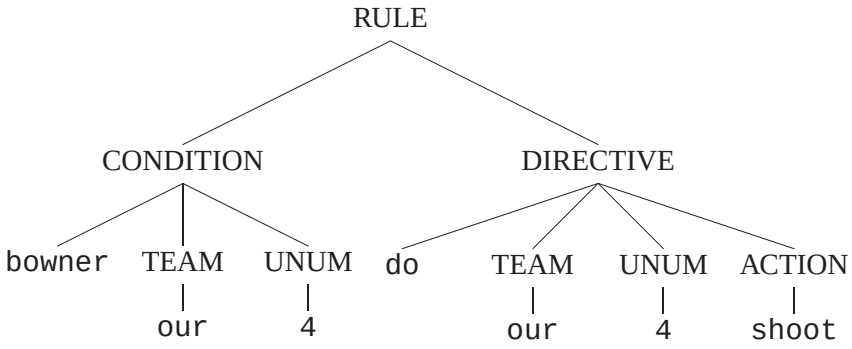
2.1.1 Application Domains

CLANG: The RoboCup Coach Language: RoboCup [1] is an international AI research initiative using robotic soccer as its primary domain. One of the several competitions organized under it is the Coach Competition where coachable soccer agents compete on a simulated soccer field. The coaching advice is given to them in a standard formal coach language called CLANG (Chen et al., 2003). CLANG is a simple declarative language with prefix notation like LISP. Figure 1 gives an example of a piece of coaching advice

[1] http://www.robocup.org/

NL: *"If our player 4 has the ball, our player 4 should shoot."*
CLANG: `((bowner our {4}) (do our {4} shoot))`

CLANG parse tree:



Non-terminals: `bowner`, `our`, `4`, `shoot`
Terminals: RULE, CONDITION, DIRECTIVE, TEAM, UNUM, ACTION
Productions:
RULE → (CONDITION DIRECTIVE)
CONDITION → (`bowner` TEAM {UNUM})
DIRECTIVE → (`do` TEAM {UNUM} ACTION)
TEAM → `our` UNUM → `4` ACTION → `shoot`

Figure 1: An example of natural language advice and its CLANG meaning representation with parse tree.

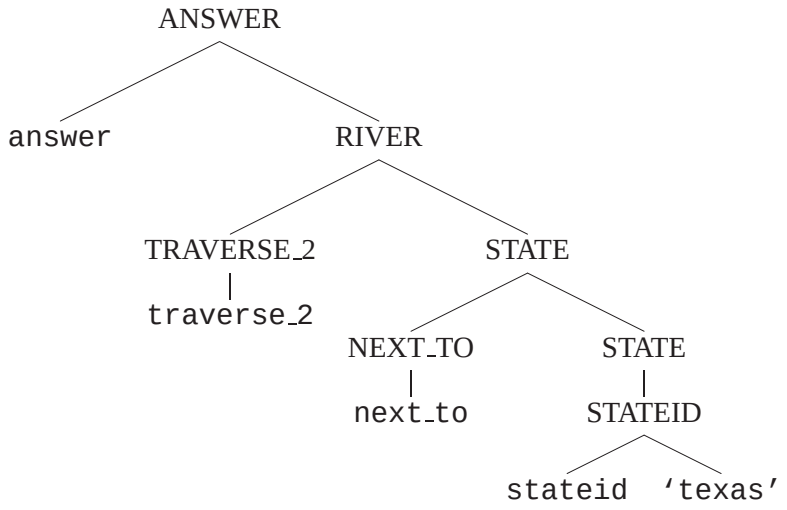
in NL with its corresponding CLANG MR. The unique parse of the MR is also shown along with the involved terminals, non-terminals and productions. In the MR, `bowner` stands for ball owner and UNUM stands for uniform numbers (players 1 through 11). Our learning algorithm exploits these MR parses and the productions.

GEOQUERY: A Database Query Application: GEOQUERY is a logical query language for a small database of about 800 U.S. geographical facts. This domain was originally chosen because of the availability of a hand-built natural language interface for comparison, GEOBASE, which came with Turbo Prolog 2.0 (Borland International, 1988). Its query language is Prolog augmented with several meta-predicates (Zelle & Mooney, 1996). These queries were converted into a functional, variable-free query language in (Kate et al., 2005) which is more convenient for some semantic parsers.

Figure 2 shows an NL query and its MR in Prolog and functional query language forms. The parse of the functional query language is also shown with the involved productions, non-terminals and terminals. This example is also used in the Section 3 to illustrate how our system does semantic parsing. The MR in the functional query language can be read as if processing a list which gets modified by various functions. The innermost expression, `stateid('texas')`, stands for the list with a single element: the state of Texas. Next, the expression `next_to(stateid('texas'))` denotes the list containing all the states next to the state of Texas. The expression, `traverse_2(next_to(stateid('texas')))`, denotes the list of all the rivers which flow through these states which are next to Texas. This list is finally returned as the answer. The unary function, `traverse_2(S)`, which returns the list of rivers traversing through states in the list S, relates to the binary predicate `traverse(A,B)` of the query language in Prolog, which is true if A flows through B. Similarly, there is a unary function, `traverse_1(R)`, in the functional query language which returns the list of the states through which the rivers in the list R traverse through.

ATIS: Air Travel Information System: ATIS is an ARPA-sponsored benchmark domain for speech

NL: *"Which rivers run through the states bordering texas?"*
Prolog: `answer(A,(river(A),traverse(A,B),state(B),next_to(B,C),const(C,stateid('texas'))))`
Functional query language: `answer(traverse_2(next_to(stateid('texas'))))`
Parse tree of the MR in functional query language:



Non-terminals: ANSWER, RIVER, TRAVERSE_2, STATE, NEXT_TO, STATEID
Terminals: `answer`, `traverse_2`, `next_to`, `stateid`, `'texas'`
Productions:
ANSWER → `answer`(RIVER)
RIVER → TRAVERSE_2(STATE)
STATE → NEXT_TO(STATE)
STATE → STATEID
TRAVERSE_2 → `traverse_2`
NEXT_TO → `next_to`
STATEID → `stateid 'texas'`

Figure 2: An example of natural language query and its meaning representation in Prolog and in functional query language with its parse tree.

NL: *"Show me flights from New York to Los Angeles."*
SQL: `SELECT flight_id FROM flight WHERE from_airport='New York' AND to_airport = 'Los Angeles'`

Figure 3: An example of natural language question and its meaning representation in SQL.

recognition and understanding (Price, 1990). Its corpus consists of spoken NL questions about air travel, their transcribed forms and their MR in SQL database query language. The questions are relatively simple and their semantic analysis is equivalent to filling a single semantic frame. But one thing that makes this corpus interesting for semantic parsing is that it was built by engaging the subjects in dialogs through speech which sometimes leads to noise in the NL sentences as well as coreferences across these sentences. Figure 3 gives a sample question and its SQL form. We haven't tried our semantic parser on this domain yet, this is part of our future work.

2.1.2 Related Work

This subsection summarizes related work in the area of semantic parsing including our previous work based on transformation rules.

- **Syntax-based Semantic Parsing:** Some approaches use syntactic and semantic annotations on the training NL sentences to learn semantic parsers. Miller et al. (1996) present an approach in which the nodes of full syntactic parse trees of the NL sentences are augmented with semantic categories. They model this type of *augmented tree parsing* by probabilistic recursive transition networks. They have tested their system on the ATIS corpus.

  Ge and Mooney (2005) present a system called SCISSOR, that learns a statistical parser that integrates syntax and semantics. It needs semantically annotated syntactic parse trees of the NL sentences for training in which each internal node has a semantic label in addition to a syntactic head word. State-of-the-art syntactic parsing model, (Collins, 1997) Model 2, is then used to learn the integrated parser. The MR can be recovered from the parse tree using a recursive procedure which allows this system to obtain MRs which are multiple levels deep (unlike (Miller et al., 1996) where the output MR is essentially flat). SCISSOR has been tested on CLANG and GEOQUERY domains.

  The approach by Zettlemoyer and Collins (2005) combines syntactic-semantic parsing using combinatory categorical grammars (CCG). While its training does not require additional syntactic or semantic annotations, it needs some hand-built rules to encode prior knowledge of syntax. Their system learns rules to construct a bilingual lexicon relating CCG syntactic categories to the lambda functions associated with the semantics. A log-linear model is used for doing probabilistic parsing of NL sentences using this lexicon.

- **Semantic Parsing by Transformation Rules:** In our previous work (Kate et al., 2005), we developed a system, SILT, which does semantic parsing by learning transformation rules to incrementally transform NL sentences into their MRs. The transformation rules associate NL patterns with MRL templates. During parsing, whenever a rule's NL pattern is found to match in a sentence, the matched pattern is replaced by the MRL template. By the end of parsing, the entire sentence gets transformed into its MR. One drawback of this system that limits its recall is that it uses hard-matching transformation rules which are sometimes too brittle to capture all the range of NL contexts. Its parsing is also done deterministically which is less robust than doing a probabilistic parsing. SILT has two

versions: a tree-based version that utilizes syntactic parse trees of the sentences and a string-based version which does not use NL syntax.

Recently, Wong (2005) has developed a system called WASP based on synchronous context-free grammars. This is an improvement over SILT. It uses a statistical word alignment model to find good transformation rules which are then used to build a probabilistic model for parsing.

- **Other Approaches:** CHILL, (Zelle & Mooney, 1996), (Tang & Mooney, 2001), is an Inductive Logic Programming (ILP) framework for learning semantic parsers. It learns rules to control the actions of a deterministic shift-reduce parser. It processes sentences one word at a time making hard parsing decisions every time, this makes the system somewhat brittle. Since it also does deterministic parsing, it may not be able to find the globally best parses of the sentences. The ILP techniques are also slow and memory-intensive and do not scale to large corpora.

  PRECISE (Popescu et al., 2003), (Popescu et al., 2004) is a system to build NL interface to databases. This system does not involve learning. It uses the notion of *semantically tractable* sentences, the sentences which can have only a unique semantic interpretation. These are the type of sentences this system can parse. Using a partly manually constructed lexicon which relates NL words to semantic types and a set of semantic constraints, it reduces the semantic parsing task to a maximum flow graph problem. The results show that over 90% of context-independent sentences in the ATIS corpus are semantically tractable while only 80% of GEOQUERY sentences are semantically tractable. This shows that GEOQUERY is more challenging domain for semantic parsing than ATIS.

  In past there have been a few more approaches for semantic parsing, mainly tested on the ATIS domain: He and Young (2003) use hidden Markov model (HMM), Papineni, Roukos, and Ward (1997) and Macherey, Och, and Ney (2001) use machine translation algorithms and Kuhn and De Mori (1995) use decision trees to translate NL questions into SQL queries.

2.2 Kernel-based Machine Learning

Traditionally, machine learning methods accept an explicit feature-based representation of the input where an example is represented by a collection of features (feature vector). But often data cannot be expressed effectively using features, especially when the data is present in some structured form as is typically true in several NLP problems. The structural information is often lost when the data is reduced to some pre-defined set of features. For e.g., when a natural language sentence (a sequence structure) is reduced to a bag of words or even a bag of bigrams or trigrams, the information about the presence of longer subsequences is lost. To avoid this, if one tries to explicitly include all possible features so that no information is lost (e.g. make all possible subsequences as features) then the number of features blow-up and it becomes computationally impractical for the learning algorithms to handle them.

Kernel-based methods (Vapnik, 1998) are an attractive alternative to feature-based methods. They allow the learning algorithms to work on potentially infinite number of features without explicitly handling them. The machine learning algorithms which use the data only to compute similarity (dot-product) between the examples can be kernelized, like Support Vector Machines (SVMs) (Cristianini & Shawe-Taylor, 2000), Perceptron (Aizerman, Braverman, & Rozonoér, 1964), Principal Component Analysis (Schölkopf, Smola, & Muller, 1999) or Nearest Neighbor. A kernel is a similarity function satisfying certain properties which maps a pair of objects to their similarity score. Formally, a kernel function K over the domain X maps two objects x,y ϵX to their similarity score, $K(x, y)$, which ranges from 0 to infinity. For all the objects $x_1, x_2, ..., x_n \in X$, the nxn matrix $(K(x_i, x_j))_{ij}$, called the Gram matrix, is required to be symmetric

and positive-semidefinite. Due to this property, kernel functions can be shown to implicitly calculate the dot-product of feature vectors of objects in some high-dimensional feature space. Hence the underlying kernelized machine learning algorithm then essentially analyzes the data in this implicit high-dimensional space.

The following subsections briefly describe SVMs, the kernelized machine learning algorithm we use in this proposal, and string-subsequence kernel, the kernel function we use with SVMs.

2.3 Support Vector Machines

Mapping data to a high-dimensional space, as is typically done through kernels, comes with a problem that learning algorithms tend to overfit the training data due to sparsity of data likely to be present because of so many dimensions (known as the "curse of dimensionality"). But Support Vector Machines (SVMs) are known to be resistant to this overfitting, hence it is the best choice for a kernelized learning algorithm.

SVMs were first introduced by Boser, Guyon, and Vapnik (1992) and have become a popular classification algorithm. Given two sets of points (like positive and negative examples), SVMs learn a separating hyperplane separating the points such that *margin*, the distance between the hyperplane and the closest point, is maximized. The points closest to the separating hyperplane are called *support vectors*. This solution which maximizes the margin has sound theoretical justification for valid generalization which is resistant to overfitting even in high dimensional spaces (Vapnik, 1998).

Since SVMs use data only to find similarity between data points, they can be kernelized. Through the kernel function the input points are implicitly mapped to a high dimensional feature space. A linear separating hyperplane with maximum margin is then found in this high dimensional space. This may correspond to some complex non-linear separating hyperplane in the original input space. For training, kernelized SVMs need kernels between every pair of training examples (i.e. the Gram matrix) and for testing they need kernels between the test example and all its support vectors (a subset of training examples).

2.4 String Subsequence Kernel

Following the framework of Lodhi et al. (2002), we define a kernel between two strings as the number of common subsequences between them. One difference, however, is that their strings are over characters while our strings are over words. The more the two strings share, the greater the similarity score will be deemed.

Formally, following the notation of (Rousu & Shawe-Taylor, 2005), let Σ be a finite alphabet, a string is a finite sequence of elements from Σ, and the set of all strings is denoted by Σ^*. For any string s, we denote $|s|$ as the length of the string $s = s_1 s_2 .. s_{|s|}$. The string $s[i..k]$ stands for the *substring* $s_i s_{i+1} .. s_k$ of s, substrings are contiguous by definition. We say that u is a *subsequence* of s, if there exists an index sequence $\mathbf{i} = (i_1 i_2 .. i_{|u|})$, with $1 \leq i_1 < .. < i_{|u|} \leq |s|$, such that $u_j = s_{i_j}$ for $j = 1, .., |u|$, and write $u = s[\mathbf{i}]$ for short. Subsequences need not be contiguous by their definition. We call the distance between the first index of $\mathbf{i}$ to its last index as its span, $span(\mathbf{i}) = i_{|u|} - i_1 + 1$. For example, consider the string $s = left_1\ side_2\ of_3\ our_4\ penalty_5\ area_6$, where the subscripted numbers are indices of the words in the string. Then $u = left\ penalty\ area$ is a subsequence of s because there is an index sequence $\mathbf{i} = (1\ 5\ 6)$ such that $u = s[\mathbf{i}]$. The span of $\mathbf{i}$, $span(\mathbf{i})$ equals $6 - 1 + 1 = 6$.

Since there can be multiple index sequences $\mathbf{i}$ for a string s, such that $u = s[\mathbf{i}]$, we define $\Phi_u(s)$ as the number of such unique index sequences, i.e. $\Phi_u(s) = |\{\mathbf{i} | s[\mathbf{i}] = u\}|$. But this definition does not take into account the sum total of all the gaps present in different index sequences. If we want to downweight the presence of gaps, we can do it through a *decay factor* $\lambda \in (0, 1]$ and redefine $\Phi_u(s)$ as:

$$\Phi_u(s) = 1/\lambda^{|u|} \sum_{\mathbf{i}:s[\mathbf{i}]=u} \lambda^{span(\mathbf{i})} \quad (1)$$

The normalization $1/\lambda^{|u|}$ ensures that only gaps and not the matches are penalized. Note that for $\lambda = 1$, the above reduces to the earlier definition which had no gap penalties. For the examples of u and s given earlier, $\Phi_u(s) = \lambda^6/\lambda^3 = \lambda^3$, which represents the total gap of 3 present in the index sequence $\mathbf{i} = (1\ 5\ 6)$ that skips over the three words $side_2\ of_3\ our_4$.

Finally, we define the kernel $K(s,t)$ between two strings s and t as:

$$K(s,t) = \sum_{u \in \Sigma^*} \Phi_u(s)\Phi_u(t) \quad (2)$$

The kernel so defined is implicitly using the space of all possible subsequences as features and computing their dot-products.

Table 1 shows an example for computation of kernel between the two strings $s = left_1\ side_2\ of_3\ our_4\ penalty_5\ area_6$ and $t = our_1\ left_2\ penalty_3\ area_4$, where the subscripted numbers are simply the indices of the words in the strings. Note that the table includes all the subsequences, u, that are common between the two strings. The chosen value for the parameter λ can be plugged in the final expression to get the numeric kernel value. Lodhi et al. (2002) give an efficient dynamic programming algorithm to compute string subsequence kernels in $O(n|s||t|)$ time where n is the maximum length of subsequences one wants to consider. Rousu and Shawe-Taylor (2005) give another algorithm which works faster when the alphabet size is large.

The kernel can be normalized to have values in the range $[0, 1]$ to remove any bias due to different string lengths:

$$K_{normalized}(s,t) = \frac{K(s,t)}{\sqrt{K(s,s)K(t,t)}} \quad (3)$$

String subsequence kernels have been previously used with success in Natural Language Processing (NLP) for Text Classification(Lodhi et al., 2002) and relational Information Extraction (Bunescu & Mooney, 2005b). We use them here for Semantic Parsing.

3 Completed Research

This section describes our novel approach to learning semantic parsers which we call KRISP, Kernel-based Robust Interpretation by Semantic Parsing. The description is followed by some implementation details and experiments.

3.1 Overview of KRISP

Given a set of NL sentences paired with corresponding MRs, KRISP learns the semantic parser in iterations, each iteration improving upon the parser learned in the previous iteration. In each iteration, it first collects positive and negative examples for each of the productions in the MRL grammar. In the first iteration, these positive and negative examples are simply the NL sentences based on whether the production is present in their corresponding MR parses or not. Using these examples, Support-Vector Machine (SVM) classifiers with string similarity as the kernel are trained. These classifiers are then used in the semantic parsing to generate the best parse for each of the training NL sentences. Based on whether these parses are correct

u	$\{(\mathbf{i}, span(\mathbf{i}))\|s[\mathbf{i}] = u\}$	$\{(\mathbf{i}, span(\mathbf{i}))\|t[\mathbf{i}] = u\}$	$\Phi_u(s)$	$\Phi_u(t)$	$\Phi_u(s) * \Phi_u(t)$
left	{((1), 1)}	{((2), 1)}	1	1	1
our	{((4), 1)}	{((1), 1)}	1	1	1
penalty	{((5), 1)}	{((3), 1)}	1	1	1
area	{((6), 1)}	{((4), 1)}	1	1	1
left penalty	{((1 5), 5)}	{((2 3), 2)}	λ^3	1	λ^3
left area	{((1 6), 6)}	{((2 4), 3)}	λ^4	λ	λ^5
our penalty	{((4 5), 2)}	{((1 3), 3)}	1	λ	λ
our area	{((4 6), 3)}	{((1 4), 4)}	λ	λ^2	λ^3
penalty area	{((5 6), 2)}	{((3 4), 2)}	1	λ	λ
left penalty area	{((1 5 6), 6)}	{((2 3 4), 3)}	λ^3	1	λ^3
our penalty area	{((4 5 6), 3)}	{((1 3 4), 4)}	1	λ	λ
					$K(s,t) = 4 + 3\lambda + 3\lambda^3 + \lambda^5$

Table 1: An example of computing subsequence kernel between the strings $s = left_1\ side_2\ of_3\ our_4\ penalty_5\ area_6$ and $t = our_1\ left_2\ penalty_3\ area_4$.

or not, positive and negative examples are collected which are used to train classifiers in the next iteration. Figure 4 shows this overall picture. The following two subsections describe the two main modules of KRISP: semantic parsing and training the classifiers which are used in semantic parsing.

3.2 Semantic Parsing

KRISP does semantic parsing by finding the best semantic derivation of a sentence. The following subsections describe semantic derivations and the algorithm used by KRISP to find them.

3.2.1 Semantic Derivations

We define a *semantic derivation*, D, of an NL sentence, s, as a parse tree of an MR (not necessarily the sentence's correct MR in which case the semantic derivation will also be incorrect) such that each node of the parse tree also contains a substring of the sentence in addition to a production. We denote nodes of the derivation tree by tuples $(\pi, [i..j])$, where π is its production and $[i..j]$ stands for the substring $s[i..j]$ of s, and we say that the node or its production *covers* the substring $s[i..j]$. The substrings covered by the children of a node are not allowed to overlap, and the substring covered by the parent must be the concatenation of the substrings covered by its children nodes. Figure 5 shows a semantic derivation of the NL sentence and the MR parse which were shown in figure 2. Productions are shown on the nodes of the tree instead of non-terminals to emphasize the role of productions in derivations. The substrings $s[i..j]$ covered by each production is shown by $[i..j]$ on its node.

Sometimes, the children nodes of an MR parse tree node may not be in the same order as are the substrings of the sentence they should cover in a derivation tree. For e.g., if the sentence was *"Through the states that border Texas which rivers run?"*, which has the same MR as the sentence shown in figure 5, then the order of the children of the node with the production "RIVER → TRAVERSE_2(STATE)" would need to be reversed. To accommodate this, a semantic derivation tree is allowed to contain MR parse tree nodes in which the children nodes have been permuted.

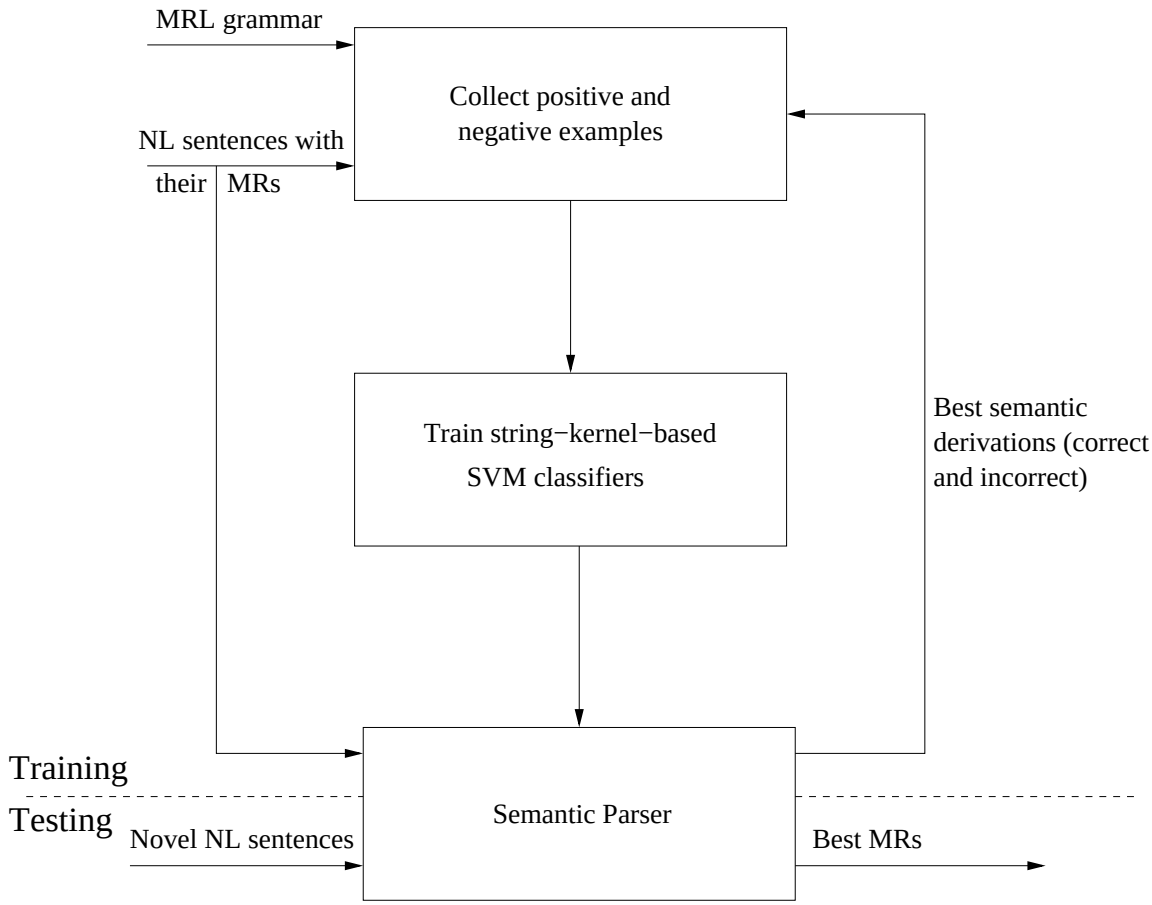


Figure 4: Overview of KRISP

Let $P_\pi(s[i..j])$ denote the probability that production π covers the substring $s[i..j]$. These probabilities are obtained by the string-kernel based SVM classifiers $P = \{P_\pi | \pi \in MRL\ grammar\ G\}$. Given these classifiers, this subsection explains how KRISP finds the most probable semantic derivations of the NL sentences, the section 3.3 will describe how KRISP trains these classifiers. From a semantic derivation it is trivial to obtain the MR: simply take the MR parse over the sentence and write it as an MR expression. Since some children nodes might have been permuted, this step also needs to permute them back to the way they should be according to the MRL productions. We refer to this procedure as $recover$ in section 3.3.

The probability of a semantic derivation D of a sentence s is simply:

$$P(D) = \prod_{(\pi,[i..j])\in D} P_\pi(s[i..j]) \tag{4}$$

The task of the semantic parser is to find the most probable derivation of sentence s. This task can be recursively performed using the notion of a *partial derivation* $E_{n,s[i..j]}$, which stands for any subtree of a derivation tree with n as the left-hand-side (LHS) non-terminal of the root production and which covers s from index i to j. For e.g., the subtree rooted at the node "(STATE $\rightarrow$ NEXT_TO(STATE),[5..9])" in the derivation shown in figure 5 is a partial derivation which would be denoted as $E_{STATE,s[5..9]}$. Note that

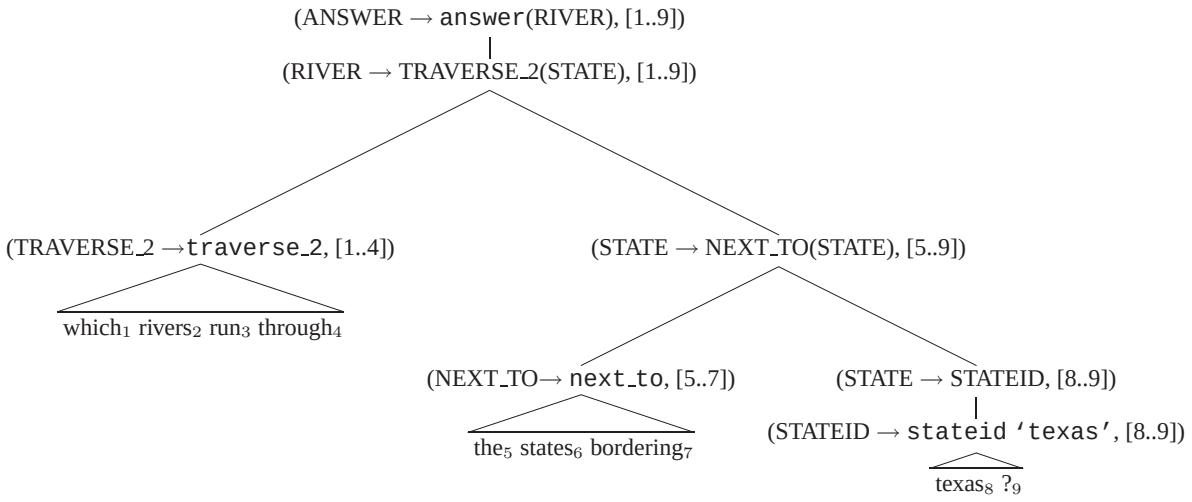


Figure 5: Semantic derivation of the NL sentence *"Which rivers run through the states bordering texas?"* which gives MR as `answer(traverse_2(next_to(stateid(texas))))`.

derivation D of sentence s is then simply $E_{start,s[1..|s|]}$, where $start$ is the start symbol of MRL's context free grammar.

Mathematically, the most probable partial derivation $E^*_{n,s[i..j]}$ is recursively defined as:

$$E^*_{n,s[i..j]} = makeTree(\underset{\pi=n\rightarrow n_1..n_t \in G,(p_1,..,p_t)\in partition(s[i..j],t)}{\text{argmax}} (P_\pi(s[i..j]) \prod_{k=1..t} P(E^*_{n_k,p_k}))) \quad (5)$$

The above equation finds the most probable partial derivation $E^*_{n,s[i..j]}$ by trying out all productions $\pi = n \rightarrow n_1..n_t$ in the MRL grammar G which have n as the LHS non-terminal, and all *partitions* with t elements of the substring $s[i..j]$ (n_1 to n_t are right-hand-side (RHS) non-terminals, terminals do not play any role in this process and are not shown for simplicity). A partition of a substring $s[i..j]$ with t elements is a $t-$tuple containing t non-overlapping substrings of $s[i..j]$ which give $s[i..j]$ when concatenated. Here $partition(s[i..j], t)$ is a function which returns the set of all partitions of $s[i..j]$ with t elements including their permutations. For, e.g. calling $partition($*"the states bordering texas ?"*$, 2)$ will return the set of partitions: {*("the", "states bordering texas ?"), ("states bordering texas ?", "the"), ("the states", "bordering texas ?"), ("bordering texas ?", "the states"), ("the states bordering", "texas ?"), ("texas ?", "the states bordering"), ("the states bordering texas", "?"), ("?", "the states bordering texas")*} . To find the most probable partial derivation $E^*_{STATE,s[5..9]}$ for the sentence shown in figure 5, the above equation will try all the productions in the grammar with STATE as the LHS non-terminal, for e.g., one of them being "STATE → NEXT_TO STATE". Then for this example production, it will try all partitions with permutations of the the substring $s[5..9]$ with two elements (shown earlier), and recursively find the most probable derivations $E^*_{NEXT\_TO,p_1}$ and E^*_{STATE,p_2}, where (p_1, p_2) denotes a partition. The recursion reaches base cases when the productions which have n on the LHS do not have any non-terminal on the RHS or when the substring $s[i..j]$ becomes smaller than the length t.

The equation finds that production π and partition $(p_1, .., p_t)$ which gives the maximum product of the probability of π covering the substring $s[i..j]$ with the probabilities of all the recursively found most probable partial derivations. The procedure $makeTree(\pi, (p_1, .., p_t))$ then constructs a partial derivation tree by making π as its root production and making the most probable partial derivation trees found through the recursion as children subtrees which cover the substrings according to the partition $(p_1, .., p_t)$.

A naive implementation of the above recursion will be computationally very expensive, but by suitably extending the well known Earley's context-free grammar parsing algorithm (Earley, 1970) it can be implemented efficiently. The above task has some resemblance to probabilistic context-free grammar (PCFG) parsing for which efficient algorithms are available (Stolcke, 1995), but we note that our task of finding the most probable semantic derivation differs from PCFG parsing in two important ways:

1. The probability of a production is not independent of the sentence but depends on which substring of the sentence it covers.

2. The leaves of the tree are not individual terminals of the grammar but are substrings of words of the NL sentence.

3.2.2 Extended Earley's Algorithm for Semantic Derivations

Parsing a sentence s by Earley's algorithm involves a single left-to-right pass over s while filling an array called a *chart*, that has $|s| + 1$ entries. For each word position in the sentence, the chart contains a list of *states* of the subtrees derived so far. Each subtree is compactly represented in a state only once which is shared by other subtrees which need it. The possible subtrees are predicted top-down and are completed bottom-up which makes the parsing very efficient. Jurafsky and Martin (2000) present a good description of Earley's algorithm which we extend here.

A state in each chart entry contains the following information:

1. the root production of the subtree

2. where in the sentence this subtree's coverage begins

3. up to which RHS non-terminals in the production the subtree has been completed and where in the sentence its coverage ends

4. the probability of the subtree derived so far

All this information about a state can be compactly represented by a *dotted rule*, an example of which is ($_5$STATE $\rightarrow$ NEXT_TO $\bullet_8$ STATE, 0.88). Here the subscripted number 5 on the LHS non-terminal indicates that this subtree starts its coverage from the fifth word of the sentence, the dot and its subscript 8 indicates that subtree corresponding to NEXT_TO non-terminal has been completed whose coverage ends at the seventh word in the sentence but the subtree corresponding to STATE non-terminal on the RHS hasn't been completed yet, and 0.88 is the probability of this derivation subtree so far. A state is called *complete* if the dot is at the end of the production, a complete state means that the whole tree below the root production has been completed. A state is called a *base state* if its production has no non-terminal on the RHS, these correspond to"POS $\rightarrow$ word" type of productions in syntactic parsing. In order to recover the tree structures from this chart structure, each state also contains links to the completed states it is composed. This information is not shown for simplicity.

Figure 6 gives the extended Earley's algorithm, EARLEY_DERIVE, for obtaining the most probable semantic derivation of a sentence s, given the MRL grammar G and the classifiers P. It does a beam search and gives the best ω derivations it finds, where ω is a system parameter called the *beam width*. If the beam width is infinite, then this algorithm is guaranteed to find all the semantic derivations of the sentence (which will include the most probable one), but this setting is computationally impractical to run. With a smaller beam width (like $\omega = 30$ in our experiments), the algorithm will do a greedy approximation search to find the ω most probable derivations.

function EARLEY_DERIVE(sentence s, MRL grammar, classifiers P)
INSERT($({}_0NULL \rightarrow \bullet_0\ start,\ 1)$, $chart[0]$)
for i=0 **to** $|s|$ **do**
for each $state$ in $chart[0..i-1]$ **do**
if (BASE($state$) **and** INCOMPLETE($state$)) **then** SCANNER($state$, i)
for each $state$ in $chart[i]$ **do**
if (**not** BASE($state$) **and** INCOMPLETE($state$)) **then** PREDICTOR($state$)
elseif (BASE($state$) **and** INCOMPLETE($state$)) **then** SCANNER($state$,i)
else COMPLETER($state$)
return($chart$)
procedure PREDICTOR($({}_iA \rightarrow \alpha\ \bullet_j\ B\ \beta,\ p)$)
for each $(B \rightarrow \gamma)$ in MRL grammar **do**
for each permutation γ' of γ **do**
INSERT($({}_jB \rightarrow \bullet_j\ \gamma',\ 1)$, $chart[j]$)
procedure SCANNER($({}_iA \rightarrow\ \bullet_i\ \alpha,\ p)$, k)
if $(p = P_{A\rightarrow\alpha}(s[i..k]) \geq \theta)$ **then**
INSERT($({}_iA \rightarrow \alpha\ \bullet_{k+1},\ p)$, $chart[k+1]$)
procedure COMPLETER($({}_jB \rightarrow \gamma\ \bullet_k,\ p)$)
for each $({}_iA \rightarrow \alpha\ \bullet_j\ B\ \beta, q)$ in $chart[j]$ **do**
if (INCOMPLETE($({}_iA \rightarrow \alpha\ \mathbf{B}\ \bullet_k\ \beta, p*q)$))
INSERT($({}_iA \rightarrow \alpha\ \mathbf{B}\ \bullet_k\ \beta, p*q)$, $chart[k]$)
elseif $(r = P_{A\rightarrow\alpha\ B\ \beta}(s[i..k-1]) \geq \theta)$ **then**
INSERT($({}_iA \rightarrow \alpha\ \mathbf{B}\ \bullet_k\ \beta, p*q*r)$, $chart[k]$)
procedure INSERT($state$, $chart[j]$)
if ($state$ is not already in $chart[j]$) **then**
if (INCOMPLETE($state$)) **then**
PUSH($state$, $chart[j]$)
else let $state = ({}_iA \rightarrow \alpha \bullet_j,\ p)$
if BEAM($A \rightarrow \alpha$, i, j) is not full **then**
PUSH($state$, $chart[j]$)
elseif ($p >$ lowest probability of state in BEAM($A \rightarrow \alpha$, i, j)) **then**
replace it by $state$

Figure 6: The extended Earley's algorithm for obtaining the most probable semantic derivations

In the pseudo code, the Greek alphabets α, β and γ are used to represent sequences (possibly empty) of non-terminals and terminals on the RHS of productions while the capitalized alphabets stand for non-teminals. The parsing starts by inserting the dummy state $({}_0NULL \rightarrow \bullet_0\ start)$ which has the start symbol of the MRL grammar on the RHS and the subscripts tell that nothing has been parsed yet. Parsing then proceeds by examining chart entries and words of the sentence left-to-right. There are three main procedures involved: PREDICTOR, SCANNER and COMPLETER.

The PREDICTOR procedure generates states representing the top-down expectations of the parses. Since in a semantic derivation a sentence may get covered by any permutation of the RHS non-terminals, the predictor generates states corresponding to all the permutations of RHS non-terminals. If a state in the chart entry being processed is incomplete and is not a base state then PREDICTOR is called on that state. For example, when PREDICTOR is called on the state $({}_5$STATE $\rightarrow$ NEXT_TO $\bullet_8$ STATE, $q)$ it will predict the state $({}_8$STATE $\rightarrow \bullet_8$ STATEID, 1) among some other states, hoping to find a subtree for the RHS non-terminal STATE. The value 1 is a temporary placeholder probability which will get multiplied by some real probability when this state gets completed.

If a state is a base state and is incomplete, then SCANNER is called on it. SCANNER looks at the

current word in the sentence and if the substring from the beginning word of the state till this word has a good probability for getting covered by the state's production, then a new complete state is generated. This probability has to be greater than a threshold θ, which is a system parameter used to prune very low probability parses. Since the leaves of the derivation can contain any number of words, SCANNER is called for all previous chart entries first (i.e. base states being completed may have their begin word anywhere back in the sentence). As an example, when SCANNER is called on the state ($_8$STATE $\rightarrow$ $\bullet_8$ STATEID, 1) while processing the ninth word, then if the probability $p = P_{STATE \rightarrow STATEID}(s[8..9]) \geq \theta$ then the SCANNER will produce the completed state ($_8$STATE $\rightarrow$ STATEID $\bullet_{10}$, p).

If a state is complete, then COMPLETER is called on it. The COMPLETER looks at all the states in the chart which need this completed subtree and generates new states advancing them from their previous states. The probability of a new state is the product of the probabilities of its previous state and the probability of the state on which completer was called. If a new state is a complete state then it is included only if the probability r of its production covering the substring from the beginning word of the state till the end word of the state is greater than the parameter θ. The probability r is also multiplied with the current probability of the new state to get its new probability. For example, calling COMPLETER on ($_8$STATE $\rightarrow$ STATEID $\bullet_{10}$, p) will generate state ($_5$STATE $\rightarrow$ NEXT_TO STATE $\bullet_{10}$, $p * q$) from the previous state ($_5$STATE $\rightarrow$ NEXT_TO $\bullet_8$ STATE, q). Since this is also a complete state (i.e. the dot is in the end), this will be included only if ($r = P_{STATE \rightarrow NEXT\_TO\ STATE}(s[5..9]) \geq \theta$) and in that case the new probability will be also multiplied by r to get the state: ($_5$STATE $\rightarrow$ NEXT_TO STATE $\bullet_{10}$, $p * q * r$).

Finally, a procedure called INSERT inserts states into the chart. A state is included only if it is not already present in the chart entry. Also, to do the beam search, beams of only the best ω completed states for each of the productions starting and ending at the same places in the sentence are maintained. If the beam is full then the new state to be inserted replaces the lowest probability state in the beam provided the new probability is greater than that lowest probability. Since threshold θ is used to prune low probability trees, its is possible that the algorithm may not find any derivation.

3.3 KRISP's Training Algorithm

Given the training corpus of NL sentences paired with their MRs $\{(s_i, m_i)|i = 1..N\}$, KRISP first parses the MRs with the MRL grammar G. Since the MRL is a formal language with a deterministic context free grammar, this parsing can be done unambiguously. Sample MR parses were shown in figures 1 and 2. We represent the parse of MR, m_i, by $parse(m_i)$.

Figure 8 shows KRISP's training algorithm. For each production π of the MRL grammar, KRISP collects positive and negative examples sets. In the first iteration, the set $\mathcal{P}(\pi)$ of positive examples for production π contains all those sentences s_i such that $parse(m_i)$ uses the production π. The set of negative examples $\mathcal{N}(\pi)$ for production π includes all the remaining training sentences.

Using these positive and negative examples, an SVM classifier is trained for each production using a string subsequence kernel. Normally, SVM classifiers only predict the class of the test example but one can obtain probability estimate of an example's class by mapping the distance of the example from the SVM's separating hyperplane to the range [0,1] using a learned sigmoid function (Platt, 1999). This then gives us the probabilities $P_\pi(s[i..j])$ on which the semantic parsing described in the previous section depends. We represent the set of these classifiers by $P = \{P_\pi|\pi \in G\}$.

Next, using these classifiers the Earley's extended algorithm, EARLEY_DERIVE, described in the previous subsection is invoked to obtain the ω best derivations for each sentence. The procedure $recover$ returns the MR from a semantic derivation, this is a simple procedure which was also described in subsection 3.2.1. It is possible that for some sentences, none of the obtained derivations give the correct MR. But as

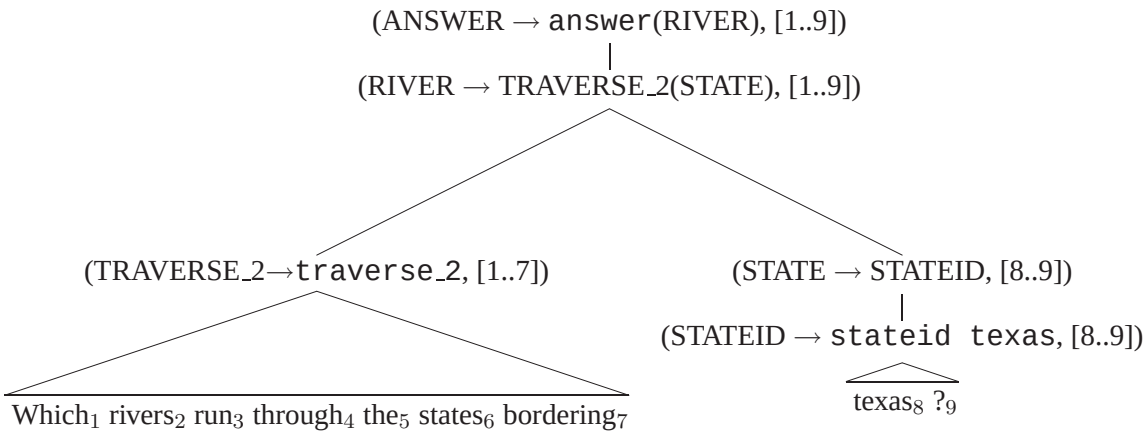


Figure 7: An incorrect semantic derivation of the NL sentence *"Which rivers run through the states bordering texas?"* which gives the incorrect MR `answer(traverse_2(stateid(texas)))`, the correct one being `answer(traverse_2(next_to(stateid(texas))))`.

is described later, the most probable derivation which gives the correct MR is needed to collect positive and negative examples for the next iteration. Hence in these cases, a version of the extended Earley's algorithm, EARLEY_DERIVE_CORRECT, is invoked which also takes the correct MR as an argument and obtains the best ω derivations, all of which give the correct MR. This is done easily by making sure that all the subtrees derived in the process are present in the parse of the correct MR.

From these derivations, positive and negative examples are collected for the next iteration. Positive examples are collected from the most probable derivation which gives the correct MR (figure 5 shows an example of a correct derivation). At each node in that derivation, the substring covered is taken as a positive example for the production. Negative examples are collected from those derivations whose probability is higher than the most probable correct derivation but which do not give the correct MR. Figure 7 shows an example of an incorrect derivation, the one shown in figure 5 being the correct one. Here the function "`next_to`" is missing from the MR it produces.

The following procedure is used to collect negative examples from incorrect derivations. The incorrect derivation and the most probable correct derivation are traversed simultaneously starting from the root using breadth-first traversal. The first nodes where their productions differ is detected, and all of the words covered by the these nodes (in both derivations) are marked. In the correct and incorrect derivations shown in figures 5 and 7 respectively, the first nodes where the productions differ are "(STATE → NEXT_TO(STATE), [5..9])" and "(STATE → `stateid`, [8..9])". Hence, the union of words covered by them: 5 to 9 ("*the states bordering texas?"*), will be marked. For each of these marked words, the procedure considers all of the productions which cover it in the two derivations. The nodes of the productions which cover a marked word in the incorrect derivation but not in the correct derivation are taken as negative examples. In the example, the node "TRAVERSE_2→`traverse_2`, [1..7]" will be taken as negative example (i.e. the words 1 to 7 ''*which rivers run through the states bordering"* will be a negative example for the production TRAVERSE_2→`traverse_2`) because the production covers the marked words *"the"*, *"states"* and *"bordering"* in the incorrect derivation but not in the correct derivation. With this as a negative example, hopefully in the next iteration, the probability of this derivation will decrease significantly to go below the probability of the correct derivation.

In each iteration, the positive examples from previous iteration are first removed so that new positive

function TRAIN_KRISP(training corpus $\{(s_i, m_i)|i = 1..N\}$, MRL grammar G)
for each $\pi \in G$ // collect positive and negative examples for the first iteration
  for $i = 1$ **to** N **do**
    if π is used in $parse(m_i)$ **then**
      include π in $\mathcal{P}(\pi)$
    else include π in $\mathcal{N}(\pi)$

for iteration $= 1$ **to** MAX_ITR **do**
  for each $\pi \in G$ **do**
    $P_\pi = trainSVM(\mathcal{P}(\pi), \mathcal{N}(\pi))$ // SVM training
  for each $\pi \in G$ $\mathcal{P}(\pi) = \Phi$ // empty the positive examples, accumulate negatives though
  for $i = 1$ **to** N **do**
    $D =$EARLEY_DERIVE(s_i, G, P) // obtain best derivations
    if $\nexists\, d \in D$ such that $parse(m_i) = recover(d)$ **then**
      $D = D \cup$ EARLEY_DERIVE_CORRECT(s_i, G, P, m_i) // if no correct derivation then force to find one
    $d^* = \text{argmax}_{d \in D \& recover(d) = parse(m_i)}\, P(d)$
    COLLECT_POSITIVES(d^*) // collect positives from maximum probability correct derivation
    for each $d \in D$ **do**
      if $P(d) > P(d^*)$ **and** $recover(d) \neq parse(m_i)$ **then**
      // collect negatives from incorrect derivation with larger probability than the correct one
      COLLECT_NEGATIVES(d, d^*)
return classifiers $P = \{P_\pi|\pi \in G\}$

Figure 8: KRISP's training algorithm

examples which lead to better correct derivations can take their place. However, negative examples are accumulated across iterations for better accuracy because negative examples from each iteration only lead to incorrect derivations and it is always good to include them. Moreover, since the extended Earley's algorithm does a limited beam search and may not find all the derivations, in each iteration it may miss some incorrect derivations from which negative examples could have been collected. Hence accumulating them across iterations only helps in collecting more negative examples.

After a specified number of MAX_ITR iterations, the trained classifiers from the last iteration are returned. The testing involves generating the most probable derivation of the test sentence and returning its MR.

3.4 Implementation

This section lists some details about KRISP's implementation.

1. **Dealing with Constants:** The MRL grammar may contain productions corresponding to constants of the domain, for e.g., "STATEID → `'new york'`", "RIVERID → `'colorado'`" etc. in GEOQUERY and "NUM → 2", "STRING → "DR4C10" etc. in CLANG. Our system allows the user to specify such productions as *constant productions* giving the NL substrings, called *constant substrings*, which directly relate to them. For e.g, user may give "texas" as the constant substring for the production "STATEID → `'texas'`. Then KRISP does not learn classifiers for these constant productions and instead decides if they cover a substring or not by matching it with the constant substrings. Whenever a constant substring is found in the NL sentence, KRISP takes the probability of the corresponding production covering this substring as 1. If n productions have the same constant

(e.g. "RIVERID → `colorado`" and "STATEID → `colorado`"), then all of them get probability equal to $1/n$ and the maximum probability derivation gets decided based on the rest of the context. If in a derivation, a constant production covers more words besides its constant substring, say a total of n words taking the constant substring as one word (for e.g. production "STATEID → `'texas'`" covers two words ,"texas ?", in the derivation shown in figure 5), then the probability is taken as $1/n$ to discourage constant productions from covering more than their constant substrings. Also, none of these extra words being covered should be another constant substring otherwise the derivation will miss the other corresponding constant production. If a constant substring found in the sentence corresponds to only one constant production then the constant substring in the sentence is replaced by the LHS non-terminal (e.g. "texas" in the sentence will be replaced by STATEID) to facilitate more generalization when learning classifiers for other productions.

2. **Computing Kernels:** While finding the best semantic derivation of a sentence, KRISP computes the probability of productions on several substrings of the sentence, which requires computing kernels between all of these substrings and the positive and negative examples. When the dynamic programming algorithm by Lodhi et al. (2002) compute kernels between two strings $s[1..|s|]$ and $t[1..|t|]$, in the process it also finds kernels between all the substrings $s[1..k]$ and $t[1..l]$ where $k < |s|$ and $l < |t|$. Hence, by running their algorithm for computing kernels between strings $s[j..|s|]$ for $1 \leq j \leq |s|$ and $t[1..|t|]$, we get kernels between t and all the substrings of s. This way, kernels between all substrings of a sentence and an example are computed efficiently and stored in a table which are used as needed.

3. **Faster Semantic Parsing:** In order to make semantic parsing faster, productions whose probabilities of covering the complete sentence are very low, i.e. less than the threshold θ according to the classifiers from the first iteration, are not considered for obtaining best semantic derivations even in latter iterations. This reduces the number of productions to be considered in the extended Earley's algorithm which significantly improves training as well as testing time.

4. **SVM Package:** We use the LIBSVM package [2] for SVMs, specifically its tool "SVM with Precomputed Kernel Matrices". This package is known to be fast and easy to use.

3.5 Experiments

3.5.1 Methodology

KRISP was evaluated on two domains: CLANG and GEOQUERY which were described in section 2.1.1. The CLANG corpus was built by randomly selecting 300 pieces of coaching advice from the log files of the 2003 RoboCup Coach Competition. These formal advice instructions were translated into English by one of four annotators. The GEOQUERY corpus was built by collecting 250 questions by asking undergraduate students to generate English queries for the given database. These queries were then manually translated into logical form (Zelle & Mooney, 1996). We note that the queries in this corpus are more complex than those in the ATIS corpus (described in section 2.1.1) which makes the GEOQUERY problem harder. This was also shown by the results in (Popescu et al., 2004). The GEOQUERY corpus was expanded to 880 sentences by collecting more queries, some of them from real users of the web-based interface to the database (Tang & Mooney, 2001). Table 2 shows some statistics about these corpora. The average length of an NL sentence in the CLANG corpus is 22.52 words while in the GEOQUERY corpus it is less than 8 words, this indicates that CLANG is the harder corpus. The average length of the MRs is also larger in the CLANG corpus.

[2] http://www.csie.ntu.edu.tw/ cjlin/libsvm/

Statistic	CLANG	GEO250	GEO880
No. of examples	300	250	880
Avg. NL sentence length	22.52	6.76	7.48
Avg. MR length (tokens)	13.42	6.20	6.47
No. of non-terminals	16	44	44
No. of productions	102	133	133
No. of unique NL tokens	337	159	270

Table 2: Some statistics of the corpora used for evaluation.

KRISP was evaluated using standard 10-fold cross validation. Since KRISP uses a threshold θ to prune low probability parses, it may fail to return any complete MR of a test sentence. Hence we computed the number of test sentences for which KRISP produced complete MRs, and the number of these MRs that were correct. For CLANG, an output MR is considered correct if it exactly matches the correct MR, up to reordering of the arguments of commutative operators like and. For GEOQUERY, an output MR is considered correct if the resulting query retrieves the same answer as the correct MR when submitted to the database. Then the performance was measured in terms of precision and recall defined as follows:

$$Precision = \frac{\text{Number of correct MRs}}{\text{Number of test sentences with complete output MRs}} \tag{6}$$

$$Recall = \frac{\text{Number of correct MRs}}{\text{Number of test sentences}} \tag{7}$$

KRISP gives probabilities for its semantic derivations which can be taken as confidences in the corresponding MRs. These confidences can be used to plot precision-recall curves by first sorting the best MR for each sentence (from all the folds) by their confidences and then finding precision for every recall value. In our experiments, the beam width parameter ω was fixed to 30, the minimum probability threshold θ was fixed to 0.05 and the maximum length of the string subsequences used for computing kernels was fixed to 3. These parameters were found through pilot experiments. The maximum number of iterations, MAX_ITR, required were only 2, beyond this we found that the system only overfits the training corpus and gives no benefit on testing.

We compared our system's performance with the systems described briefly in the Related Work subsection: the string and tree versions of SILT(Kate et al., 2005), WASP (Wong, 2005), SCISSOR (Ge & Mooney, 2005), system by Zettlemoyer and Collins (2005) and CHILL (with COCKTAIL ILP algorithm (Tang & Mooney, 2001)). WASP and SCISSOR also give confidences to the MRs they generate which are used to plot precision-recall curves. The results of the other systems are shown as points on the precision-recall graph. The results of Zettlemoyer and Collins (2005) are available only for the GEO880 corpus. Their experimental set-up also differs from ours, they explicitly set aside 600 GEOQUERY examples for training and used the remaining 280 for testing. Their experiment was repeated twice and the average statistics were reported. We also compared KRISP with GEOBASE (Borland International, 1988), a hand-built NL interface for the GEOQUERY domain. Its results are available only for the GEO250 corpus.

3.5.2 Results and Discussion

Figure 9 shows the results on the CLANG corpus. KRISP performs better than either version of SILT in both precision and recall. It performs comparable to WASP but gives slightly less maximum recall. Although

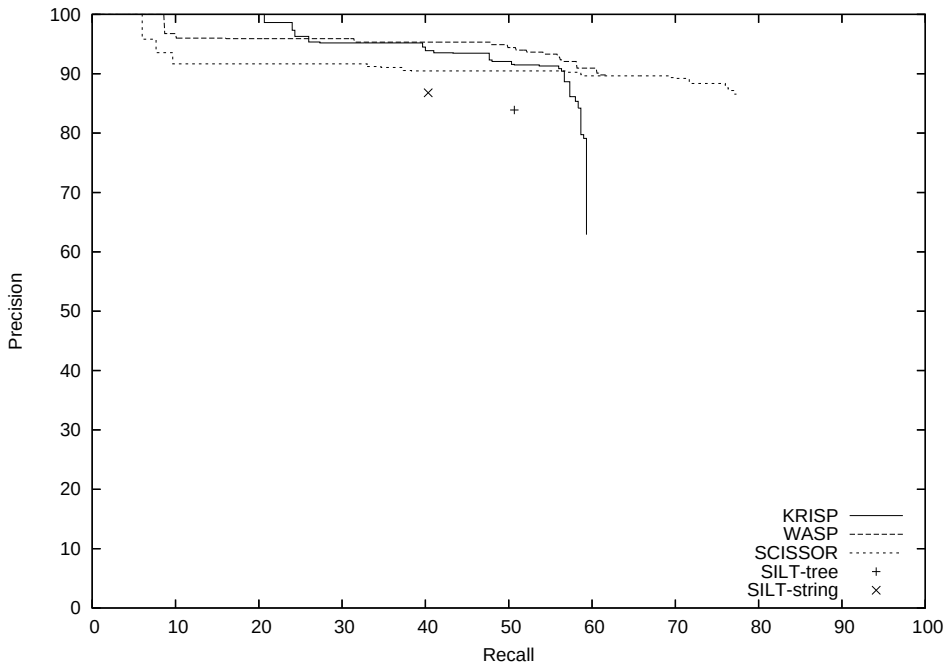


Figure 9: Results on CLANG corpus.

SCISSOR gives less precision at lower recall values, it gives much higher maximum recall. However, we note that it requires more supervision for the training corpus. CHILL could not be run beyond 160 training examples because Prolog, in which it has been implemented, runs out of memory. For 160 training examples it gave 49.2% precision with 12.67% recall. Its poor performance is due to two main reasons. First, it does not exploit the formal grammar of the MRL which guides the parsing process in other systems. Second, CHILL uses a deterministic shift-reduce parsing framework to parse a sentence from left to right. This type of parsing is restrictive because if the system fails to first parse the words on the left side of the sentence then it will fail to parse the entire sentence. Figure 10 shows the precision-recall curves for KRISP when it is trained on increasing number of training examples in each fold. The graph shows that increasing the amount of training data results in significant improvement in performance even between the last two curves. This indicates that the performance can be improved further on this corpus if more training data is provided.

The results for the GEO250 corpus are shown in figures 11. KRISP gets better recall than SILT and GEOBASE. It is able to extend the recall further than WASP's maximum recall but it gives slightly less precision than WASP at lower recall values. KRISP's maximum recall is about same as CHILL's recall but KRISP has higher precision at that recall value. SCISSOR does particularly well in precision on this corpus. Figure 12 shows the precision-recall curves of KRISP with increasing amounts of training data.

Figure 13 shows the results for the GEO880 corpus. The confidence values of SCISSOR's output MRs were not available, hence its result is shown as a point. On this corpus, KRISP does better than SILT but lags behind the other systems, particularly in precision. While we know that GEO880 is harder and less cleaner that GEO250 corpus, we plan to investigate the reasons for this lower performance. The precision-recall curves of KRISP on this corpus with increasing amounts of training data are shown in figure 14.

We have translations of GEO250 corpus in three other natural languages: Spanish, Turkish and Japanese.

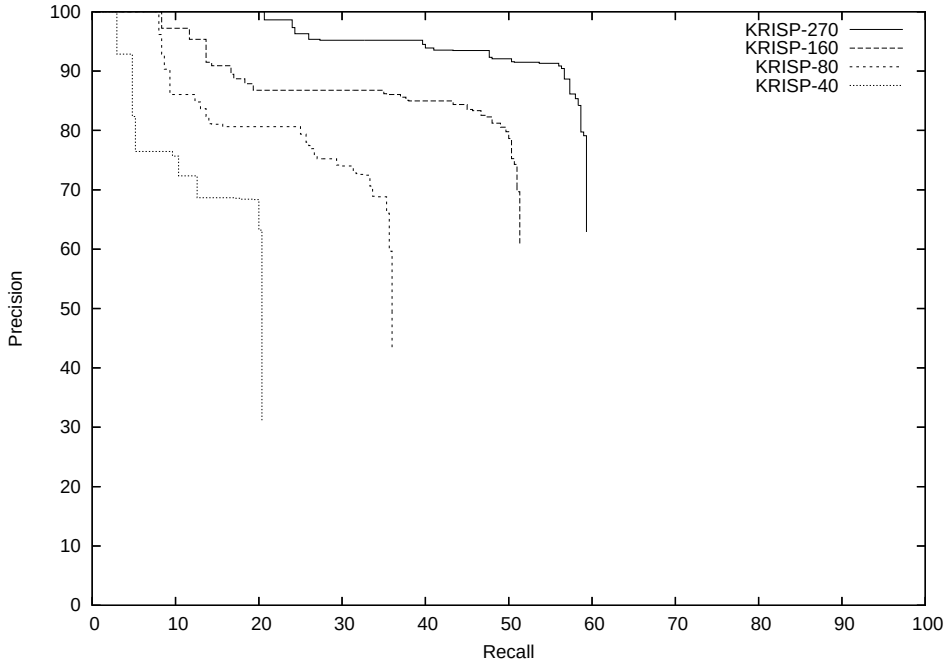


Figure 10: Results of KRISP on the CLANG corpus when trained on increasing number of training examples.

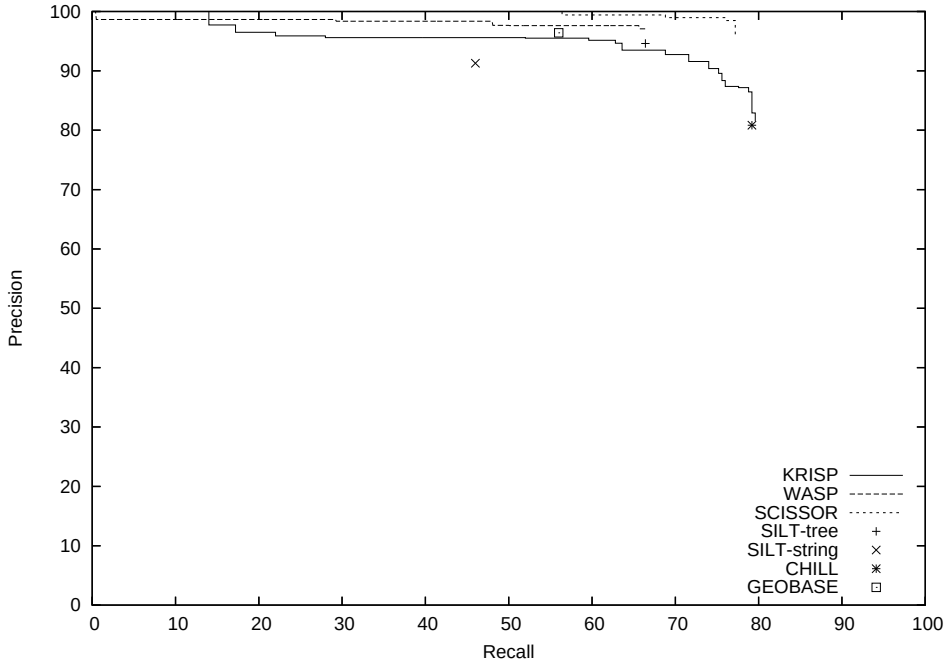


Figure 11: Results on GEO250 corpus.

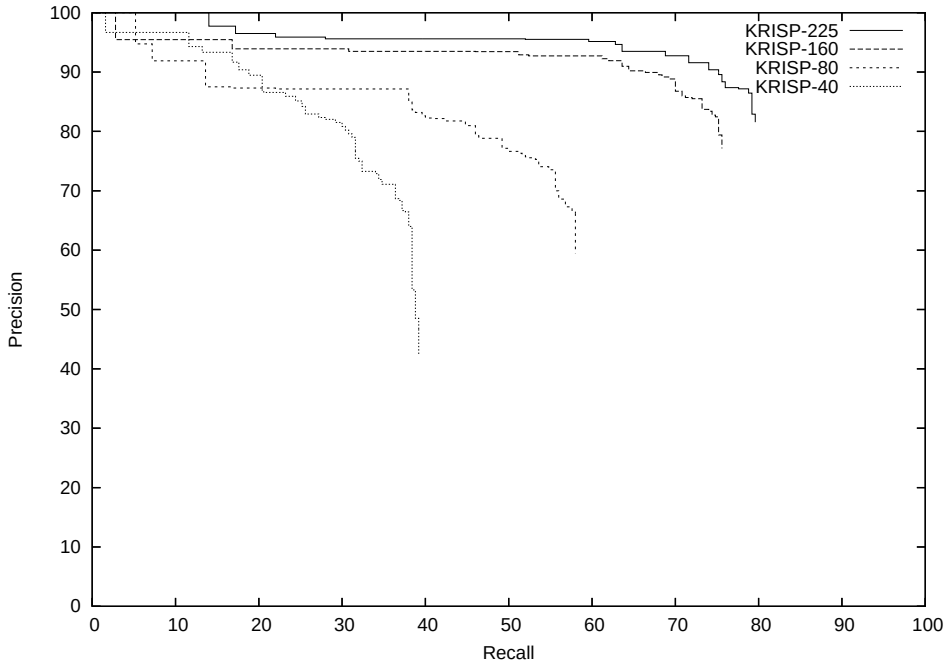


Figure 12: Results of KRISP on the GEO250 corpus when trained on increasing number of training examples.

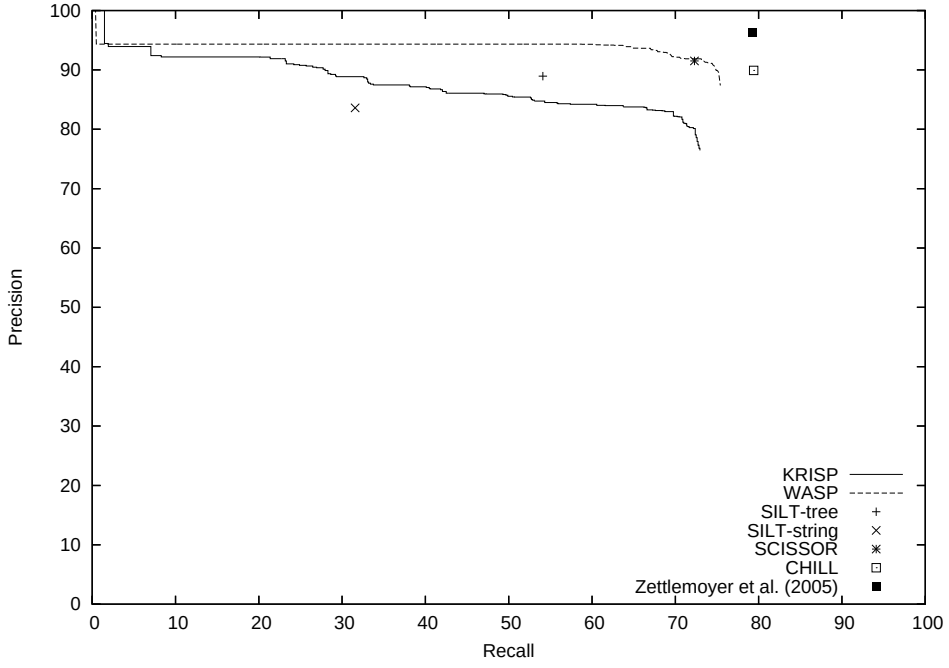


Figure 13: Results on GEO880 corpus.

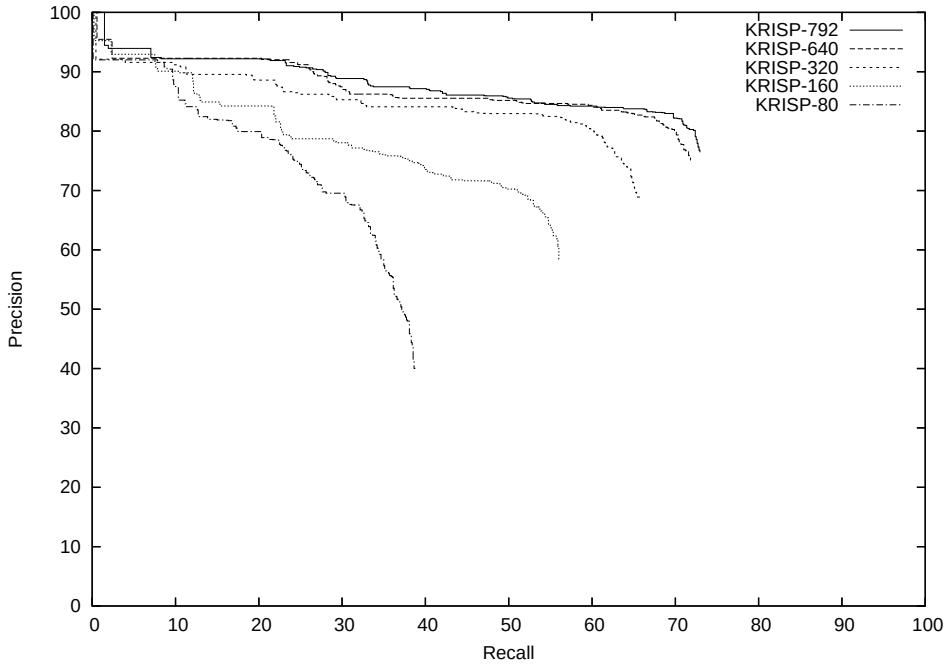


Figure 14: Results of KRISP on the GEO880 corpus when trained on increasing number of training examples.

Corpus	Training	Testing
GEO250	1.44	0.05
GEO880	18.1	0.65
CLANG	58.85	3.18

Table 3: Average training and testing time per fold in minutes taken by KRISP on the three corpora.

Since KRISP's learning algorithm does not use any natural language specific knowledge, it is directly applicable to other natural languages. Japanese uses different names for the names of the places (e.g. Tekisasu for Texas, Nyuu Yooku for New York etc.), we provide KRISP this information through the constant substrings. Figure 15 shows results of running KRISP on other natural languages. The performance on English and Spanish are comparable. Japanese gives the lowest precision and recall, we suspect it is because in Japanese words are formed by joining morphemes and there could have been confusion brought by breaking these into tokens in our corpus. Turkish gives slightly better precision but lower recall, we believe it is because Turkish has larger number of unique tokens (36% more than English) which makes learning from its corpus more precise but less general.

Table 3 shows the average training and testing time in minutes KRISP takes to run in one fold on each of the three corpora. The machine used had Intel Pentium 4, 2.6 GHz processor and 2 Gb RAM. The time taken mainly depends on the number of examples and the length of the NL sentences in the corpus.

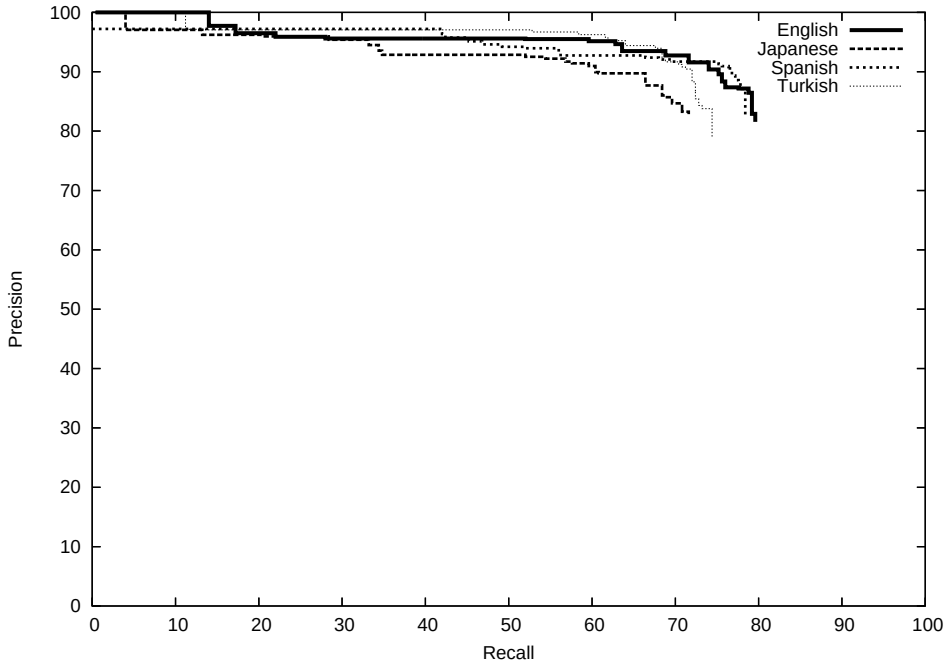


Figure 15: Results of KRISP on GEO250 corpus for different natural languages.

4 Proposed Research

The future work we propose to do are categorized under short-term and long-term future work. Under short-term future work, we plan to improve the KRISP algorithm by exploiting NL syntax, test it on noisy NL sentences and combine it with other semantic parsers to form committees. Under long-term future work, we plan to extend our system to also work on non-parallel corpora and try it on the task of complex relation extraction. The following subsections describe these plans.

4.1 Short-term Future Work

4.1.1 Exploiting Natural Language Syntax

To do semantic parsing of a sentence, KRISP currently uses only the word order of the sentence and not its syntax. While this certainly has the advantage that it makes the system directly portable to other natural languages, given that semantic interpretation of a sentence largely depends on how the words are combined according to the NL grammar, using the syntax of the sentence should certainly help in its semantic parsing. In order to exploit the NL syntax, the most natural extension of our string-kernel-based approach will be to make it tree-kernel-based. Existing syntactic parsers, like that of Bikel (2004), can be trained on our syntactically annotated corpora in addition to Wall Street Journal (WSJ) corpus to obtain syntactic parse trees of the sentences, this was also done in the tree-based version of SILT (Kate et al., 2005). One can then define a tree kernel over these syntactic parse trees and use it instead of the string kernel in the rest of our algorithm.

Syntactic-tree-kernels were first introduced by Collins and Duffy (2001) and were also used by Collins

and Duffy (2002) for the task of reranking syntactic parse trees. They define a kernel between two trees as the number of subtrees shared between them. A subtree is defined as any subgraph of the tree which includes more than one node, with the restriction that entire productions must be included at every node. Figure 16 shows two syntactic parse trees and all the common subtrees between them. The kernel defined this way captures most of the structural information present in the syntactic parse trees in the form of tree fragments which the kernelized learning algorithms can then implicitly use as features. Collins and Duffy (2001) also give an efficient algorithm to compute this kernel which runs in close to linear time in the size of the input trees.

Often the syntactic information needed for a task is present in the dependency trees (Hudson, 1984) alone and full syntactic parse trees are not needed which may in fact lead to over-fitting because of the large number of irrelevant features they may produce. Cumby and Roth (2003) have shown the advantage of a syntactically determined restricted kernel based on dependency trees over the "all subtrees" kernel from (Collins & Duffy, 2002) on the task of named entity recognition. Recently, various kernels defined over dependency trees and their variants have also shown success in the task of relational information extraction like (Zelenko et al., 2003), (Culotta & Sorensen, 2004) and (Bunescu & Mooney, 2005a). Moreover, there also has been progress in learning dependency tree parsers (McDonald, Pereira, Ribarov, & Hajič, 2005b). Dependency trees capture important aspects of functional relationship between words in a sentence which can directly help in its semantic interpretation. We plan to investigate the use of dependency tree-kernels in our system. There has not been any direct comparison between string-kernels with syntactic-parse-tree-kernels or with dependency-tree-kernels to our knowledge and it will be interesting to see the comparison on the task of semantic parsing.

4.1.2 Noisy NL Sentences

Any real world application in which semantic parsers would be used to interpret natural language of a user is likely to face noise in the input. If the user is interacting through spontaneous speech and the input to the semantic parser is coming form the output of a speech recognition system then there are many ways in which noise could creep in the NL sentences: interjections (like um's and ah's), environment noise (like door slams, phone rings etc.), out-of-domain words, grammatically ill-formed utterances etc.(Zue & Glass, 2000). These types of noises are also present in the ATIS corpus (Ward, 1990). Even when the user is interacting through typing, noise could be present because of typos, out-of-domain words etc.. The semantic parser needs to be robust to the noise in these situations.

The performance of systems like SILT (Kate et al., 2005) and WASP (Wong, 2005) could degrade in presence of such noise because it will obstruct the application of their hard-matching transformation rules. SCISSOR's performance could also degrade because it tries to build complete syntactic-semantic parse trees of the sentences but the presence of noise would obstruct this process. In contrast, KRISP's semantic parsing is more flexible and hence robust to noise. The presence of extra tokens or corrupted tokens may decrease the kernel values by some amount but it will not affect the semantic parsing in any hard way. Hence we expect KRISP's performance will degrade gracefully in the presence of noise.

We plan to first do some preliminary experiments on our existing corpora after artificially corrupting them with noise. Then we plan to get a real world noisy corpus (like ATIS) and compare KRISP with other systems on it.

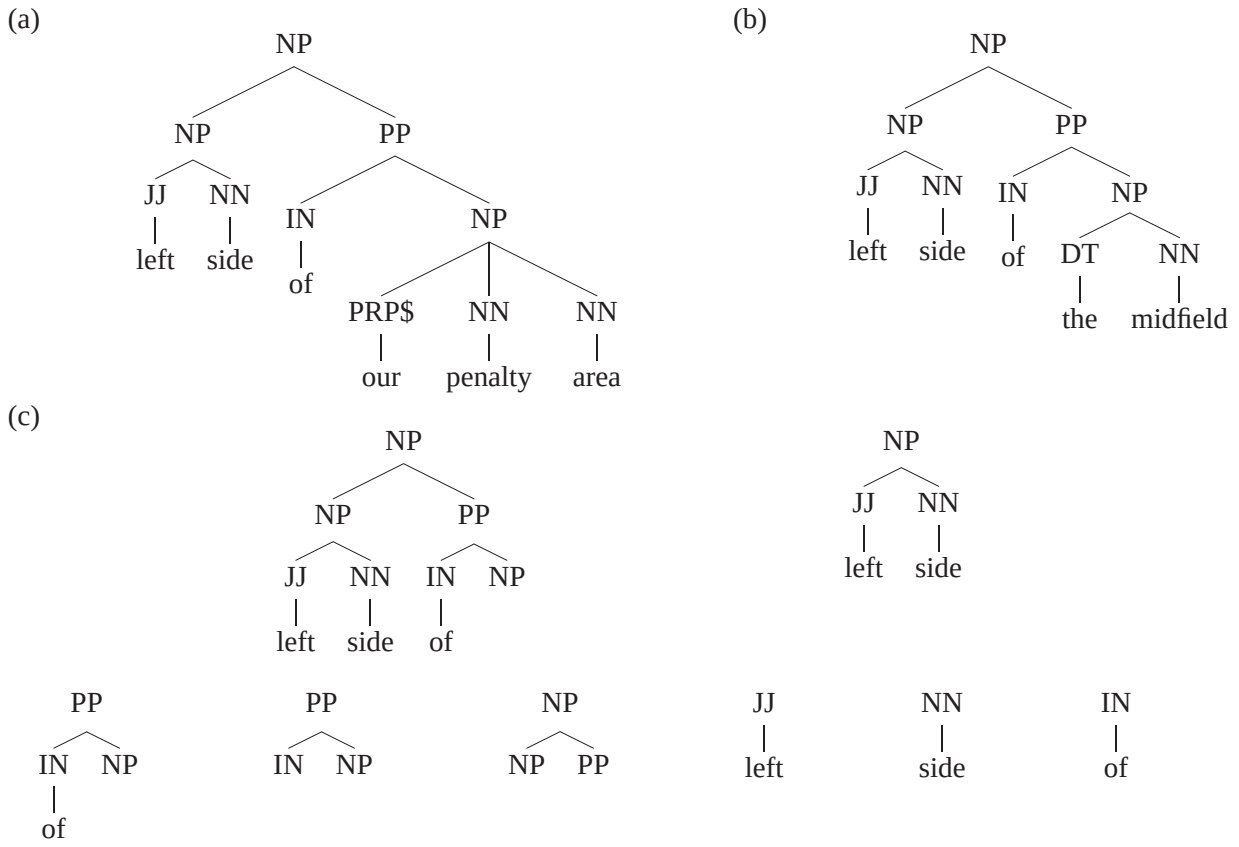


Figure 16: (a) and (b) are two example syntactic parse trees, (c) shows all the common subtrees between them.

4.1.3 Committees of Semantic Parsers

We have three competitive semantic parsers, KRISP, WASP (Wong, 2005) and SCISSOR (Ge & Mooney, 2005), developed within our research group. The next natural step would be to form a committee in order to get the best overall performance. Forming committees or ensembles of learned systems is known to improve performance provided each individual system performs well enough and these systems are diverse, i.e. they make different types of errors (Dietterich, 2000).

Table 4 shows the number of correct answers each semantic parser gets on the CLANG corpus which has total 300 examples and the maximum number of correct answers one can get by combining their correct answers (i.e. if an oracle chooses the output MR from the output MRs of the individual semantic parsers). The numbers clearly show that the three semantic parsers often make different errors and there is a lot of scope for improvement if we form a committee.

There are two general approaches in which a committee of semantic parsers can combine the output MRs from different semantic parsers. The first approach is to learn which parser performs best on which types of sentences and then for a test sentence simply choose the output MR from the best parser for that sentence. This is similar to the "Parser Switching" approach presented by Henderson and Brill (1999) for combining syntactic parses from different parsers. The second approach is to look into the output MRs generated by different parsers and combine their best components to output the best MR. This is analogous

System or Committee	No. of Correct Answers out of 300
KRISP	178
WASP	185
SCISSOR	232
KRISP + WASP	223
KRISP + SCISSOR	253
WASP + SCISSOR	246
KRISP + WASP + SCISSOR	259

Table 4: Upper bounds on the number of correct answers one can get on the CLANG corpus by combining the correct answers of different semantic parsers.

to the "Parse Hybridization" method of Henderson and Brill (1999) and the committee-based probabilistic partial parsing method of Inui and Inui (2000). But we note that their methods are not directly applicable to our problem because their methods are designed specifically for syntactic parsing. The second approach of combining output MRs will be particularly useful in cases where none of the semantic parsers generate a complete MR but each generates some of its components. We plan to investigate various ways of combining the output MRs of different semantic parsers under each of the two general approaches.

4.2 Long-term Future Work

4.2.1 Non-parallel Training Corpus

In our work so far, the training data consisted of NL sentences aligned with their respective MRs, $\{(m_i, s_i)|i = 1..N\}$. Building such corpora requires human annotation. However, there are scenarios where vast amount of data is available in which NL sentences and MRs are both present but they are not aligned. For e.g., in ROBOCUP commentary task, commentaries of simulated games are generated in natural language (André, Binsted, Tanaka-Ishii, Luke, Herzog, & Rist, 2000). When this commentary is paired with the symbolic description of events happening in the entire field then we get the scenario where NL sentences and MRs are present but they are not aligned. This is known as *referential ambiguity* problem where it is not known what portion of the NL description refers to which symbolic description. There may even be sentences which do not have their corresponding MRs present in the data (e.g. commentator deviates to some other topic) and there may be MRs which are not described in the NL (e.g. commentator skips describing less important events). This is also a more realistic context in which language acquisition occurs (Pinker, 1995). Let us represent such a corpus as $\{(M_i, S_i)|i = 1..N\}$, where each M_i and S_i are sets of MRs and NL sentences respectively. For each paired (M_i, S_i), the correspondence between the MRs in M_i and NL sentences in S_i is unknown.

While there has been some work in solving referential ambiguity problem for learning semantic lexicons (Siskind, 1996), there has not been any work in solving this for the bigger task of complete semantic parsing. Learning a semantic parser from such a corpus will first involve finding out the correspondences between NL sentences and their respective MRs and then learning semantic parsers form them. One can view this type of referential ambiguity of which sentence corresponds to which MR as one level higher than the referential ambiguity of which portion of the sentence corresponds to which production of the MR parse, something our system is already designed to resolve. Hence we believe that extending our system accordingly to one level up would help it learn semantic parser from such a corpus. Following is an outline of how we plan to

do this:

1. Collect all pairs of MRs and NL sentences from each (M_i, S_i) to form a training set T_0, i.e. $T_0 = \{(m_j, s_k) | m_j \in M_i \; and \; s_k \in S_i \; for \; i = 1..N\}$.

2. Learn classifiers P for the productions of the MRL grammar G using KRISP, i.e. $P =$ TRAIN_KRISP(T_0, G)

3. Using these classifiers P and the procedure EARLEY-DERIVE-CORRECT described in section 3.3, find out for each (M_i, S_i) which sentence $s_k \in S_i$ corresponds best to $m_j \in M_i$. Form a training set set T_t of these pairs (a sentence should be paired with at most one MR), do not include a pair if its derivation probability is too low.

4. Learn classifiers P from training set T_t, i.e. $P =$ TRAIN_KRISP(T_t, G). Repeat steps 3-4 till T_t remains the same over an iteration, then return P.

The first step above assumes that any (m_j, s_k) pair from (M_i, S_i) is equally likely since there is no more information about it. Then KRISP is used to learn classifiers for productions from this training set in step 2. Although there will be many false positives but the negatives will be all true negatives and hopefully KRISP will find correspondences between presence of productions and portions of NL sentences. Using these classifiers, in step 3, MRs in each M_i are paired with the sentences in S_i which give their most probable derivations. Each sentence is paired at most once. There may be sentences which do not give the most probable derivation for any MR. A pair is dropped if the probability of the derivation is too low, this essentially means no sentence in the set S_i corresponds to the MR in consideration. KRISP is used to learn classifiers again from this new training set and this process is repeated till the training set does not change. The final classifiers are returned which can then be used in testing to convert novel NL sentences to their MRs.

We plan to obtain or build a real-world non-parallel corpus to test our ideas. But before that, we will do preliminary experiments with our existing corpora by artificially making them non-parallel. This can be done by first partitioning the training data into N sets and then ignoring the correspondences between the sentences and their MRs within each set. The algorithm outlined above should be able to re-pair the sentences with their corresponding MRs and learn semantic parser from this.

4.2.2 Complex Relation Extraction

KRISP obtains deep semantic parses of sentences by using string-based kernels to learn classifiers for MRL grammar productions. Bunescu and Mooney (2005b) have used string-based kernels to learn classifier for extracting the binary relation *"protein-protein interaction"*. This can be viewed as learning for an MRL grammar which has only one production: "INTERACTION $\rightarrow$ PROTEIN PROTEIN". Hence we believe KRISP can be used in relation extraction, particularly in complex n-ary relation extractions.

A complex relation is defined as an n-ary relation among n typed entities (McDonald et al., 2005a). It is defined by the *schema* $(t_1, ..., t_n)$ where $t_i \in T$ are entity types. An instance of a complex relation is a list of entities $(e_1, ...e_n)$ such that $type(e_i) = t_i$. An example of a ternary relation schema is `(person, job, company)` that relates a person to their job at a particular company. From the sentence *"John Smith is the CEO of Inc. Corp."*, an instance *(John Smith, CEO, Inc. Corp.)* of the above relation can be extracted. Some entities are also allowed to be missing in complex relations.

Most of the work in relation extraction has mainly focused on identifying binary relations like *employee-of*, *located-at* etc. (NIST, 2000). Relatively little work has been done in extracting complex n-ary relations

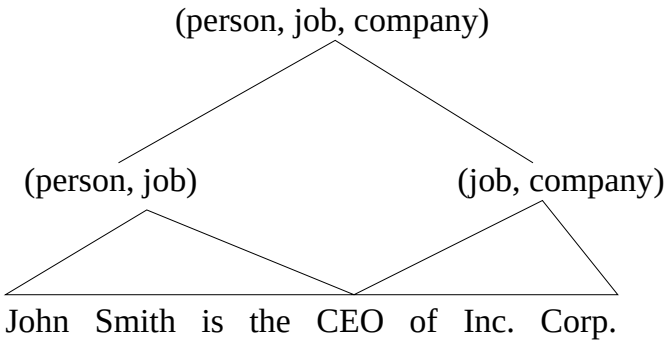


Figure 17: Relation `(person,job,company)` derived over the given sentence.

which would be useful in applications like automatic database generation, intelligent document searching etc. McDonald et al. (2005a) do complex relation extraction from Biomedical documents. They first extract all the binary relations within the complex relation (e.g `(person,job)`, `(job,company)` & `(person,company)` are all the binary relation for the complex relation `(person,job,company)`). Then graphs of all the entities are constructed in which edges are added between two entities if they are instances of a binary relation. The complex relations are extracted as the maximal cliques form these graphs. They show that this approach is better than extracting complex relations directly, which is a harder learning task. However, we note that when their approach extracts complex relation from the entity graph, the underlying sentence does not play any role.

We believe KRISP can be applied to extract complex relations. We plan to do this by treating the complex relation as a higher level production composed of lower level productions which correspond to the less complex relations (like binary relations). Then the extracting the complex relation process can be treated as semantic parsing. Figure 17 shows the semantic derivation of the example sentence from which the complex relation `(person-job-company)` has been extracted. We note that given the way KRISP does semantic parsing, the sentence will be used in the process of extracting the component binary relations as well as the final complex relation.

5 Conclusions

In this proposal, we presented a new kernel-based approach to learn semantic parsers. SVM classifiers based on string-subsequence kernels are trained for each of the productions in the meaning representation language. These classifiers are then used to compositionally build complete meaning representations of natural language sentences. We evaluated our system on two real-world corpora. The results show that our system performs better than deterministic rule-based semantic parsers and performs comparable to some recently developed statistical semantic parsers. We plan to extend this work by using NL syntax-based kernels, broaden the scope of its applications and form committees of semantic parsers.

References

Aizerman, M., Braverman, E., & Rozonoér, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, *25*, 821–837.

André, E., Binsted, K., Tanaka-Ishii, K., Luke, S., Herzog, G., & Rist, T. (2000). Three RoboCup simulation league commentator systems. *AI Magazine*, *21*(1), 57–66.

Androutsopoulos, I., Ritchie, G. D., & Thanisch, P. (1995). Natural language interfaces to databases: An introduction. *Journal of Natural Language Engineering*, *1*(1), 29–81.

Bikel, D. M. (2004). Intricacies of Collins' parsing model. *Computational Linguistics*, *30*(4), 479–511.

Borland International (1988). *Turbo Prolog 2.0 Reference Guide*. Borland International, Scotts Valley, CA.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152, Pittsburgh, PA. ACM Press.

Bunescu, R. C., & Mooney, R. J. (2005a). A shortest path dependency kernel for relation extraction. In *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing 2005 (HLT/EMNLP 2005)*, pp. 724–731, Vancouver, BC.

Bunescu, R. C., & Mooney, R. J. (2005b). Subsequence kernels for relation extraction. In *Advances in Neural Information Processing Systems*, Vancouver, BC. To appear.

Carreras, X., & Marquez, L. (2004). Introduction to the CoNLL-2004 shared task: Semantic role labeling. In *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004)*, Boston, MA.

Collins, M. (2002). Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-2002)*, pp. 489–496, Philadelphia, PA.

Collins, M., & Duffy, N. (2001). Convolution kernels for natural language. In *Proceedings of Neural Information Processing Systems (NIPS 14)*.

Collins, M., & Duffy, N. (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-2002)*, pp. 263–270, Philadelphia, PA.

Collins, M. J. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL-97)*, pp. 16–23.

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

Culotta, A., & Sorensen, J. (2004). Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pp. 423–429, Barcelona, Spain.

Cumby, C., & Roth, D. (2003). On kernel methods for relational learning. In *Proceedings of 20th International Conference on Machine Learning (ICML-2003)*, pp. 107–114.

Dietterich, T. (2000). Ensemble methods in machine learning. In Kittler, J., & Roli, F. (Eds.), *First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, pp. 1–15. Springer-Verlag.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, *6*(8), 451–455.

Ge, R., & Mooney, R. J. (2005). A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pp. 9–16, Ann Arbor, MI.

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

He, Y., & Young, S. (2003). Hidden vector state model for hierarchical semantic parsing. In *Proceedings of the 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-03)*, pp. 268–271, Hong Kong.

Henderson, J. C., & Brill, E. (1999). Exploiting diversity in natural language processing: Combining parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-99)*, pp. 187–194, College Park, MD.

Hudson, R. (1984). *Word Grammar*. Blackwell.

Inui, T., & Inui, K. (2000). Committee-based decision making in probabilistic partial parsing. In *Proceedings of the Eighteenth International Conference on Computational Linguistics*, pp. 348–354, Saarbrücken, Germany.

Jurafsky, D., & Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, Upper Saddle River, NJ.

Kate, R. J., Wong, Y. W., & Mooney, R. J. (2005). Learning to transform natural to formal languages. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-2005)*, pp. 1062–1068, Pittsburgh, PA.

Kuhn, R., & De Mori, R. (1995). The application of semantic classification trees to natural language understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *17*(5), 449–460.

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

Macherey, K., Och, F. J., & Ney, H. (2001). Natural language understanding using statistical machine translation. In *Proceedings of the 7th European Conference on Speech Communication and Technology (EuroSpeech-01)*, pp. 2205–2208, Aalborg, Denmark.

Manning, C. D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.

McDonald, R., Pereira, F., Kulick, S., Winters, S., Jin, Y., & White, P. (2005a). Simple algorithms for complex relation extraction with applications to biomedical IE. In *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL-05)*, pp. 491–498, Ann Arbor, MI.

McDonald, R., Pereira, F., Ribarov, K., & Hajič, J. (2005b). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing 2005 (HLT/EMNLP 2005)*, pp. 523–530, Vancouver, BC.

Miller, S., Stallard, D., Bobrow, R., & Schwartz, R. (1996). A fully statistical approach to natural language interfaces. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pp. 55–61, Santa Cruz, CA.

NIST (2000). ACE – Automatic Content Extraction. http://www.nist.gov/speech/tests/ace.

Papineni, K. A., Roukos, S., & Ward, R. T. (1997). Feature-based language understanding. In *Proceedings of the 5th European Conference on Speech Communication and Technology (EuroSpeech-97)*, pp. 1435–1438, Rhodes, Greece.

Pinker, S. (1995). Language acquisition. In Gleitman, L. R., & Liberman, M. (Eds.), *Language* (2nd edition)., Vol. 1 of *An Invitation to Cognitive Science*, pp. 135–182. MIT Press, Cambridge, MA.

Platt, J. C. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In Smola, A. J., Bartlett, P., Schölkopf, B., & Schuurmans, D. (Eds.), *Advances in Large Margin Classifiers*, pp. 185–208. MIT Press.

Popescu, A.-M., Armanasu, A., Etzioni, O., Ko, D., & Yates, A. (2004). Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the Twentieth International Conference on Computational Linguistics (COLING-04)*, Geneva, Switzerland.

Popescu, A.-M., Etzioni, O., & Kautz, H. (2003). Towards a theory of natural language interfaces to databases. In *Proceedings of the 2003 International Conference on Intelligent User Interfaces (IUI-2003)*, pp. 149–157, Miami, FL. ACM.

Price, P. J. (1990). Evaluation of spoken language systems: The ATIS domain. In *Proceedings of the Third DARPA Speech and Natural Language Workshop*, pp. 91–95.

Rousu, J., & Shawe-Taylor, J. (2005). Efficient computation of gapped substring kernels on large alphabets. *Journal of Machine Learning Research*, *6*, 1323–1344.

Schölkopf, B., Smola, A., & Muller, K. R. (1999). Kernel principal component analysis. In Schölkopf, B., Burges, C. J. C., & Smola, A. J. (Eds.), *Advances in Kernel Methods - Support Vector Learning*, pp. 327–352. MIT Press.

Shawe-Taylor, J., & Cristianini, N. (2000). *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Siskind, J. M. (1996). A computational study of cross-situational techniques for learning word-to-meaning mappings. *Cognition*, *61*(1), 39–91.

Stolcke, A. (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, *21*(2), 165–201.

Tang, L. R., & Mooney, R. J. (2001). Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning*, pp. 466–477, Freiburg, Germany.

Chen et al., M. (2003). Users manual: RoboCup soccer server manual for soccer server version 7.07 and later.. Available at `http://sourceforge.net/projects/sserver/`.

Vapnik, V. N. (1998). *Statistical Learning Theory*. John Wiley & Sons.

Ward, W. (1990). The CMU Air Travel Information Service: Understanding spontaneous speech. In *Proceedings of a Workshop on Speech and Natural Language*, pp. 127–129, Hidden Valley, PA.

Wong, Y. W. (2005). Learning for semantic parsing using statistical machine translation techniques. Doctoral Dissertation Proposal, University of Texas at Austin.

Zelenko, D., Aone, C., & Richardella, A. (2003). Kernel methods for relation extraction. *Journal of Machine Learning Research, 3*, 1083–1106.

Zelle, J. M., & Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pp. 1050–1055, Portland, OR.

Zettlemoyer, L. S., & Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of 21th Conference on Uncertainty in Artificial Intelligence (UAI-2005)*, Edinburgh, Scotland.

Zue, V. W., & Glass, J. R. (2000). Conversational interfaces: Advances and challenges. In *Proceedings of the IEEE*, Vol. 88(8), pp. 1166–1180.