The Dissertation Committee for Rohit Jaivant Kate
certifies that this is the approved version of the following dissertation:

Learning for Semantic Parsing with Kernels under Various Forms of Supervision

Committee:

Raymond J. Mooney, Supervisor

Jason Baldridge

Risto Miikkulainen

Dan Roth

Peter Stone

Learning for Semantic Parsing with Kernels under Various Forms of Supervision

by

Rohit Jaivant Kate, B.Tech.; M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2007

To my parents and sister.

Acknowledgments

Firstly and foremostly, I am thankful to my advisor, Ray Mooney, for his guidance and help throughout my Ph.D. process. I am also thankful to him for sharing with me and influencing me with his passion for AI, dedication for science and vigor of doing research, and making me a good scientist and researcher in the process.

I thank the rest of my thesis committee members: Jason Baldridge, Risto Miikkulainen, Dan Roth and Peter Stone for their insightful and helpful comments to improve this work.

Thanks to my parents and sister for their long-distance but constant love, care and support all these years. I thank my friends and fellow graduate students Ruifang, John, Razvan and Lily for all the good times. I also thank department staff members Stacy, Gloria and Katherine for smoothly taking care of all the administrative matters. Finally, I am thankful to all those at UT who enriched the experience of my stay here.

This research was supported by Defense Advanced Research Projects Agency under grant HR0011-04-1-0007 and by a Google research grant.

ROHIT JAIVANT KATE

The University of Texas at Austin

August, 2007

Learning for Semantic Parsing with Kernels under Various Forms of Supervision

Publication No. _____

Rohit Jaivant Kate, Ph.D.
The University of Texas at Austin, 2007

Supervisor: Raymond J. Mooney

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 computing systems that understand natural language input. This thesis presents a new machine learning approach for semantic parsing based on string-kernel-based classification. It 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. Meaning representations for novel natural language sentences are obtained by finding the most probable semantic parse using these classifiers. This method does not use any hard-matching rules and unlike previous and other recent methods, does not use grammar rules for natural language, probabilistic or otherwise, which makes it more robust to noisy input.

Besides being robust, this approach is also flexible and able to learn under a wide range of supervision, from extra to weaker forms of supervision. It can easily utilize extra supervision given in the form of syntactic parse trees for natural language sentences by using a syntactic tree kernel instead of a string kernel. Its learning algorithm can also take advantage of detailed supervision provided in the form of semantically augmented parse trees. A simple extension using transductive SVMs enables the system to do semi-supervised learning and improve its performance utilizing unannotated sentences which are usually easily available. Another extension involving EM-like retraining makes the system capable of learning under ambiguous supervision in which the correct meaning representation for each sentence is not explicitly given, but instead a set of possible meaning representations is given. This weaker and more general form of supervision is better representative of a natural training environment for a language-learning system requiring minimal human supervision.

For a semantic parser to work well, conformity between natural language and meaning representation grammar is necessary. However meaning representation grammars are typically designed to best suit the application which will use the meaning representations with little consideration for how well they correspond to natural language semantics. We present approaches to automatically transform meaning representation grammars to make them more compatible with natural language semantics and hence more suitable for learning semantic parsers. Finally, we also show that ensembles of different

semantic parser learning systems can obtain the best overall performance.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Building computing systems which understand and process natural languages has been a long-standing goal of artificial intelligence. Most researchers have approached this goal from what can be described as a "broad and shallow" direction. They have focused on tasks which involve analyzing open domain natural language text, but the analysis done is typically shallow which is suitable just enough for inferring some simple properties about the text or for representing it in some intermediate linguistic representation. Syntactic parsing (Collins, 1997; Charniak, 1997), information extraction (Cardie, 1997; Califf, 1999), word sense disambiguation (Ide & Jéronis, 1998), semantic role labeling (Gildea & Jurafsky, 2002; Carreras & Marquez, 2004) etc. are examples of such tasks.

Although considerable progress has been made on these tasks, but for a computing system to understand natural language at the level of processing and manipulating its intended meaning, it is necessary for it to map natural language utterances into computer manipulatable meaning representations. This is the task of semantic parsing which involves deeper analysis of natural language text. Not only this task is critical for developing computational sys-

tems which understand and process natural language, but it can also provide insights into human language acquisition.

However, doing semantic parsing on open domain text is still impractical because there is no global meaning representation language for open domain. Hence, semantic parsing is restricted to specific application domains, like answering database questions or controlling a robot, where the application entails the meaning representation language. This direction of research can thus be described as "narrow and deep". It is hoped that the two directions will successfully meet benefiting from each other and that will significantly impact the way computers will be used to process natural languages in the future.

In this thesis, we have considered the task of semantic parsing and have developed and evaluated a new framework for learning semantic parsers which works well under various forms of supervision.

1.1 Semantic Parsing

Semantic parsing is the task of mapping *natural language* (NL) sentences into formal executable *meaning representations* (MRs). These MRs are expressed in domain-specific formal *meaning representation languages* (MRLs). Given a corpus of NL sentences paired with their MRs, the task of a semantic parsing learning system is to induce a semantic parser which can map novel NL sentences to their correct MRs.

Some of the 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). Some previous work does not use any learning method (Androutsopoulos, Ritchie, & Thanisch, 1995; Popescu, Etzioni, & Kautz, 2003) which makes them difficult to port to other domains. The learning systems (Zelle & Mooney, 1996; Tang & Mooney, 2001; Kate, Wong, & Mooney, 2005) use meaning representations which are more complex with richer predicates and nested structures, but these systems use deterministic rule-based learning methods and lack in robustness which is the characteristic of learning methods currently used in statistical natural language processing. Recently, new learning systems for semantic parsing have been developed which use statistical feature-based methods (Ge & Mooney, 2005; Zettlemoyer & Collins, 2005; Wong & Mooney, 2006) and have been shown to perform better than the systems developed in the past.

In this thesis, we have developed a novel kernel-based statistical method for semantic parsing. Kernel methods are a powerful new approach to machine learning that have demonstrated success in a wide variety of applications (Shawe-Taylor & Cristianini, 2004). Unlike feature-based learning methods, kernelized learning methods offer the advantage of implicitly working with potentially infinite number of features defined over complex unbounded structures like strings and trees, typically encountered in *natural language*

processing (NLP) problems. For these reasons, kernel-based methods have recently been effectively applied to a variety of problems in text learning and NLP, like text categorization (Lodhi, Saunders, Shawe-Taylor, Cristianini, & Watkins, 2002), syntactic parsing(Collins, 2002b), relational information extraction (Culotta & Sorensen, 2004; Bunescu & Mooney, 2005b), pronoun resolution (Yang, Su, & Tan, 2006), semantic role labeling (Che, Zhang, Liu, & Li, 2006) and textual entailment (Wang & Neumann, 2007). Support-Vector Machines (SVMs; Cristianini & Shawe-Taylor, 2000) are the most commonly used kernelized learning method. SVMs find a separating hyperplane which maximizes the margin between training examples in the feature space. This intuitively appealing way of classification also has sound theoretical guarantees of generalization performance (Vapnik, 1998) even in typically high dimensional feature spaces implicitly induced by kernels.

Our system, KRISP (Kernel-based Robust Interpretation for Semantic Parsing), takes NL sentences paired with their formal meaning representations as training data. The productions of the formal MRL grammar are treated like semantic concepts. For each of these productions, an SVM classifier is trained using string similarity as the kernel (Lodhi et al., 2002). Each classifier then estimates the probability of the production covering different substrings of the sentence. This information is used to compositionally build a complete meaning representation of the sentence. Given that natural languages are so flexible, there are various ways to express the same semantic concept and kernel-based methods capture the full range of natural language expressions

better than feature-based methods, which makes this framework well suited for semantic parsing. Our experiments demonstrate that KRISP compares favorably to recent statistical learning systems for semantic parsing and is particularly robust to noise. We also extend this framework to work under various forms of supervision.

1.2 Thesis Contributions

In this section we summarize the main contributions of this thesis.

- The core contribution of this thesis is the new framework for learning for semantic parsing using kernel-based string classification. This framework draws benefits from the advantages offered by kernels to implicitly work with potentially infinite number of features and avoids any specifc feature representation. Our framework does not learn any hard-matching rules and unlike all the previous semantic parser learning frameworks, it does not use grammar rules for natural language, these two factors make it robust to noisy input. This is an important advantage because for most applications the input for semantic parsing will most likely be noisy, like spontaneous utterances with grammatical errors or output from speech recognizers etc.

- We show how extra supervisions in the form of syntactic parse trees and semantically augmented parse trees (Ge & Mooney, 2005) can be incorporated in our learning framework to improve results.

- We have also looked into how to make the best use of weaker forms of supervision for learning semantic parsers. Since unannotated data is usually easily available, we extended our basic semantic parser learning method to make use of unannotated data in the form of sentences without meaning representations to improve performance. To our best knowledge, this is the first semi-supervised learning system for semantic parsing.

- We also developed a learning framework for semantic parsing to work with ambiguous supervision in which the correct meaning representation for each sentence is not explicitly given but instead a set of possible meaning representations for each sentence is given. This weaker and more general form of supervision is more representative of a natural training environment for a language-learning system requiring minimal human supervision, like a computer system which will learn language by observing perceptual contexts while simultaneously being exposed to natural language commentary, similar to the way children learn language without being explicitly taught to analyze sentences. The experimental results show that our method learns accurate semantic parsers even under ambiguous supervision. This contribution can be regarded as a starting point for building computing systems which learn language from their perceptual contexts with little human supervision.

- If the constructs in the meaning representation language do not correspond well with natural language, then the semantic parser may not

perform well. We have also developed a framework for transforming the meaning representation grammar if it does not conform well with the natural language semantics. To our best knowledge, this is the first work to address this issue.

- KRISP and some recent semantic parser learning systems, like SCISSOR (Ge & Mooney, 2005) and WASP (Wong & Mooney, 2006), have different strengths and weaknesses. The thesis also describes some simple methods to form ensembles of semantic parser learning systems to obtain the best overall performance.

1.3 Organization of the Thesis

Following is the outline of the thesis.

- **Chapter 2, Background and Related Work:** This chapter provides some background about semantic parsing, kernel-based learning methods and briefly summarizes the related work.

- **Chapter 3, Kernel-based Approach to Learning Semantic Parsers:** We describe our new framework for learning semantic parsers in detail, and with experimental evaluations we show that our approach performs competitively with recent approaches and is more robust to noise.

- **Chapter 4, Utilizing More Supervision:** We describe the extensions we made to our learning system to utilize extra forms of supervision

in the form of syntactic trees and semantically augmented parse trees. Experimentally, we found that our approach does not benefit a lot from extra supervisions.

- **Chapter 5, Utilizing Weaker Forms of Supervision:** In this chapter we describe our semi-supervised method for learning semantic parsing which improves the performance of semantic parsing by utilizing NL sentences not paired with their MRs. We also describe a framework to learn semantic parsers under ambiguous supervision and show that it copes well with ambiguities to learn accurate parsers.

- **Chapter 6, Transforming the Meaning Representation Grammar to Improve Semantic Parsing:** This chapter describes our new approach to transform the meaning representation grammar to make it conform to natural language semantics. We experimentally show that the transformed grammars improve the performance of semantic parsing.

- **Chapter 7, Ensembles of Semantic Parsing:** We describe simple methods to form ensembles of semantic parsers and show that it achieves better performance than the individual semantic parsers. We also give some experimental analysis.

- **Chapter 8, Directions for Future Work:** We mention possibilities of some future research directions based on this work.

- **Chapter 9, Conclusions:** We summarize the contributions of this thesis.

We note that the material presented in Chapter 3 had appeared in our previous publication (Kate & Mooney, 2006), while the material presented in Chapter 5 has appeared in (Kate & Mooney, 2007b) and (Kate & Mooney, 2007a).

Chapter 2

Background and Related Work

This chapter gives background about the task of semantic parsing and kernel-based learning methods, and summarizes the related work in semantic parsing.

2.1 Semantic Parsing

Semantic parsing is the task of mapping *natural language* (NL) utterances into their computer understandable *meaning representations* (MRs) for some domain-specific application. 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.

Some early systems for semantic parsing were manually-built (Hendrix, Sacerdoti, Sagalowicz, & Slocum, 1978; Androutsopoulos et al., 1995). Building such systems requires a lot of manual effort and domain expertise, and they can not be easily ported to other domains. These systems are also usually brittle, i.e. they work well only for a narrow subset of natural language

input. The alternate to manually building semantic parsers is to use machine learning to automatically build them from training data.

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. As a learning task, this is very different from the task of learning for syntactic parsing, because the training data for learning for syntactic parsing consists of the syntactic parse trees built over the NL sentences from which statistics about the relatedness between parts of the parse trees to the parts of the NL sentences can be easily estimated. In contrast, the training data for the task of learning for semantic parsing consists of an NL sentence paired with its MR with no information about how the different parts of the NL sentence relate to different parts of its MR.

We describe the three application domains in which the research in learning for semantic parsing has mainly been focused.

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

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

sentences. Figure 2.1 gives a sample question and its SQL form.

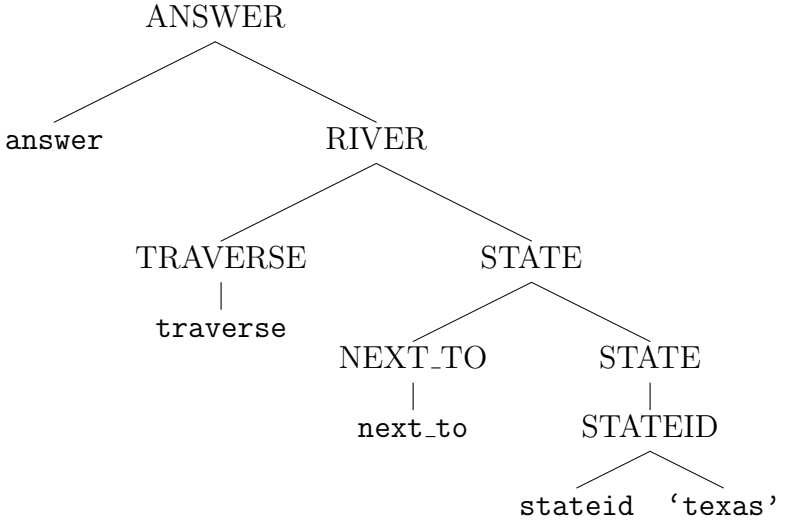
Geoquery: A Database Query Application: GEOQUERY is a logical query language for a small database of about 800 U.S. geographical facts. This domain was originally chosen because of the availability of a hand-built natural language interface for comparison, GEOBASE, which came with Turbo Prolog 2.0 (Borland 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 by Kate et al. (2005) which is more convenient for some semantic parsers.

Figure 2.2 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. Brackets are not shown in the parse to avoid clutter. Our learning algorithm exploits productions and MR parses, and this example is also used later in the Section 3.2 to illustrate how our system does semantic parsing. Throughout this thesis, the non-terminals are shown in upper-case and the terminals are shown in lower-case. 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 ele-

ment: 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(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(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.

The original GEOQUERY corpus used in semantic parsing experiments was constructed 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 call this the GEO250 corpus. We note that, on average, the queries in this corpus are more complex than those in the ATIS corpus which makes semantic parsing harder on the GEOQUERY domain. This was also shown by the results of Popescu et al. (2004). The MRs in GEOQUERY usually have deep nested structures. The GEOQUERY corpus was later 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). We call this the GEO880 corpus. The average length of sentence in this corpus is 7.48 words and the average number of tokens per MR is 6.47.

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(next_to(stateid('texas'))))`
Parse tree of the MR in functional query language:



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

Figure 2.2: An example of natural language query and its meaning representation in Prolog and in functional query language with its parse tree.

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., 2003c). CLang is a simple declarative language with prefix notation like LISP. Figure 2.3 gives an example of a piece of coaching advice 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. Brackets are again not shown to avoid clutter. In the MR, `bowner` stands for ball owner and UNUM stands for uniform numbers (players 1 through 11).

The CLang corpus used in semantic parsing experiments was constructed by randomly selecting 300 pieces of coaching advice from the log files of 2003 RoboCup Coach Competition. These formal advice instructions were translated into English by one of four annotators. On average there were 22.5 words per sentence and 13.42 tokens per MR in this corpus.

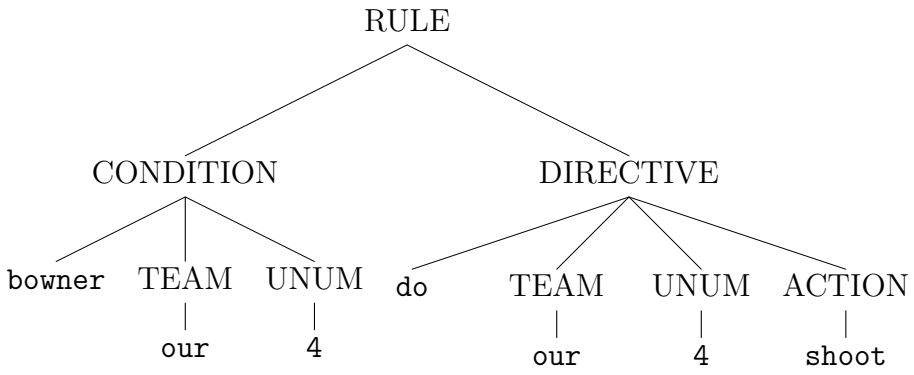
2.2 Kernel-based Machine Learning

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

[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 2.3: An example of natural language advice and its CLang meaning representation with parse tree.

using features, especially when the data is present in some structured form like strings or trees, as is typically true in several *natural language processing* (NLP) problems. The structural information is often lost when the data is reduced to some pre-defined set of features. For example, 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 (like make all possible subsequences as features) then the number of features blow-up and it becomes computationally impractical for the feature-based 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 constructing and manipulating 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, & Müller, 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 $n \times n$ 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.

Kernel-based learning methods are increasingly becoming popular in NLP. They have been applied to a variety of NLP tasks, like text classification(Lodhi et al., 2002), syntactic parsing (Collins, 2002b), relational information extraction (Culotta & Sorensen, 2004; Bunescu & Mooney, 2005b), pronoun resolution (Yang et al., 2006), semantic role labeling (Che et al., 2006) and textual entailment (Wang & Neumann, 2007). We use them in this thesis for semantic parsing.

The following subsections briefly describe SVMs, the kernelized machine learning algorithm we use in this thesis, and string-subsequence kernel, the kernel function we use with SVMs in our main semantic parser learning system.

2.2.1 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 large number of dimensions (known as the "curse of dimensionality"). But Support Vector Machines (SVMs) are known to be resistant to this overfitting, hence

they are typically 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 (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 the training examples).

SVMs have been shown to perform particularly well in text domains (Joachims, 1998) in which the data is typically represented in a very high dimensional space with a separate dimension for every word or n-gram.

2.2.2 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. Word subsequence kernels were also used by Cancedda, Gaussier, Goutte, and Renders (2003). The more the two strings share, the greater the similarity score will be deemed.

Formally, following the notation of Rousu and 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:

$$\Phi_u(s) = 1/\lambda^{|u|} \sum_{\mathbf{i}:s[\mathbf{i}]=u} \lambda^{span(\mathbf{i})} \tag{2.1}$$

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:

$$K(s, t) = \sum_{u \in \Sigma^*} \Phi_u(s)\Phi_u(t) \tag{2.2}$$

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

Table 2.1 shows a sample kernel computation 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

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

kernels in $O(n|s||t|)$ time where n is the maximum length of subsequences one wants to consider. Rousu and Shawe-Taylor (2005) present 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:

$$K_{normalized}(s,t) = \frac{K(s,t)}{\sqrt{K(s,s)K(t,t)}} \tag{2.3}$$

String subsequence kernels have been previously used with success in NLP for text classification (Lodhi et al., 2002; Cancedda et al., 2003) and relational information extraction (Bunescu & Mooney, 2005b). We use them in this thesis for semantic parsing.

2.3 Related Work

This subsection summarizes related work in the area of semantic parsing.

2.3.1 Syntax-based Semantic Parsing

Some approaches have used 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. A 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 MRs are essentially flat). SCISSOR has been tested in the CLANG and GEOQUERY domains. The approach by Nguyen, Shimazu, and Phan (2006) also uses semantically annotated syntactic parse trees. This approach uses structured SVMs (Tsochantaridis, Hofmann, Joachims, & Altun, 2004) and learns en-

sembles of semantic parsers through bagging (Breiman, 1996) and boosting (Freund & Schapire, 1996).

The approach by Zettlemoyer and Collins (2005) combines syntactic and semantic parsing using combinatory categorial grammars (CCG; Steedman, 2000). 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. Their system was further improved by relaxing the CCG grammar (Zettlemoyer & Collins, 2007).

2.3.2 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 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.

Wong and Mooney (2006) have developed a system called WASP based on synchronous context-free grammars (Aho & Ullman, 1972) that uses state-of-the-art statistical machine translation techniques for semantic parsing. It uses a statistical word alignment model to find good transformation rules which are then used to build a probabilistic model for parsing. This is an improvement over SILT. The system was extended to work when MRs are in λ-calculus (Wong & Mooney, 2007b). This framework for semantic parsing was also inverted to form a natural language generation system to map MRs into NL sentences (Wong & Mooney, 2007a).

2.3.3 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, 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 indicates that GEOQUERY is more challenging domain for semantic parsing than ATIS.

In the 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.

The approach for learning semantic parsers presented in this thesis differs from the related work in the following important ways. Our approach does not use any grammar rules for natural language sentences which makes it very flexible and robust to process a wide range of natural language input, including when corrupted with noise. The learning method used in our system is kernel-based, which can capture the full range of natural language expressions which express the same semantic concept better than feature-based methods, and does not need any feature-engineering. Finally, through simple and elegant extensions our approach also works well under a wide range of supervision.

Chapter 3

Kernel-based Approach to Learning Semantic Parsers

This chapter presents our novel approach to learning semantic parsers which we call KRISP, Kernel-based Robust Interpretation for Semantic Parsing. It learns string-kernel-based classifiers for every production of the meaning representation language grammar. Meaning representations for novel natural language sentences are obtained by finding the most probable semantic parse using these string classifiers. Our experiments on two real-world data sets show that this approach compares favorably to other existing systems and is particularly robust to noise.

3.1 Overview of KRISP

KRISP learns a semantic parser from the training data of *natural language* (NL) sentences paired with their respective *meaning representations* (MRs). The key idea in KRISP is to treat the productions of the grammar of the *meaning representation language* (MRL) as semantic concepts. For every production, a Support-Vector Machine (SVM; Cristianini & Shawe-Taylor, 2000) classifier is trained using string similarity as the kernel (Lodhi et al.,

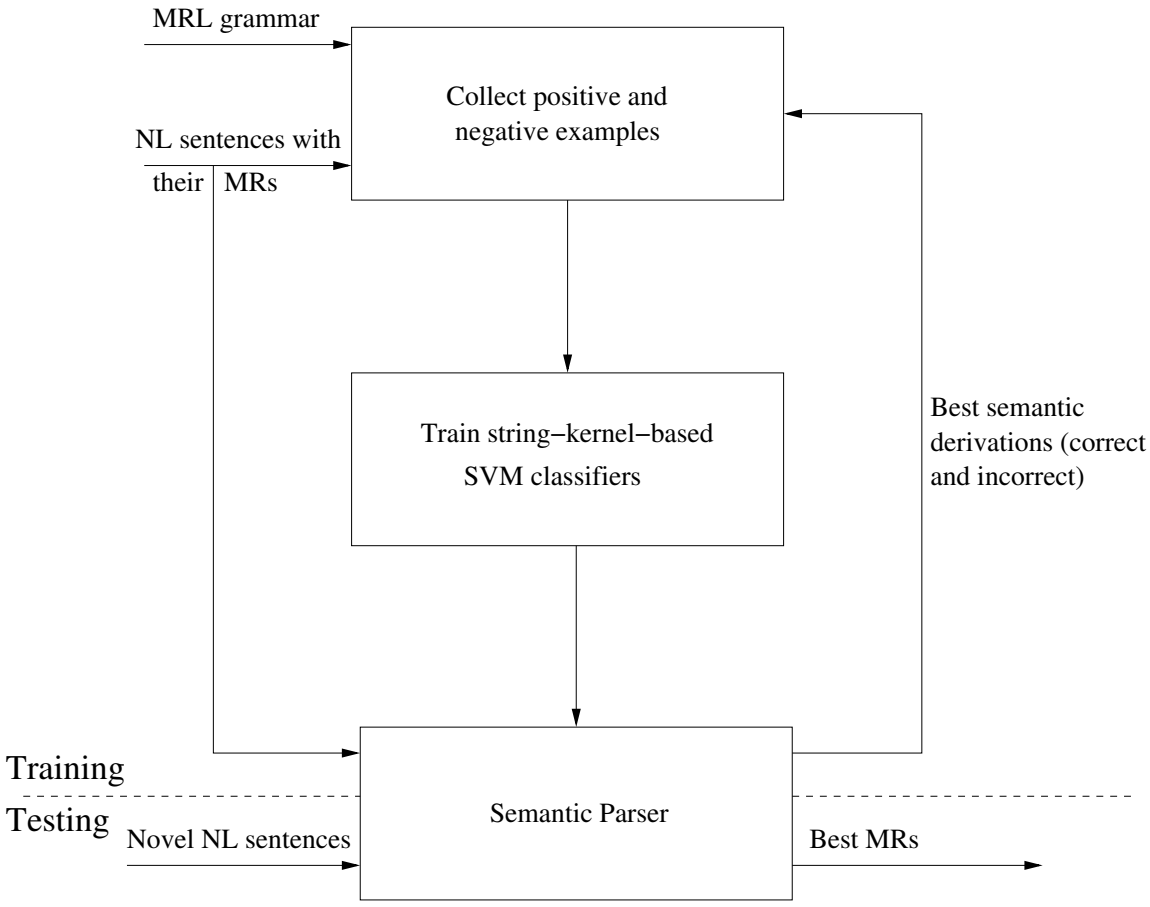


Figure 3.1: Overview of KRISP

2002). Each classifier can then estimate the probability of any NL substring representing the semantic concept for its production. During semantic parsing, the classifiers are called to estimate probabilities on different substrings of the input sentence and the most probable MR for the complete sentence is compositionally built.

KRISP trains the classifiers used in semantic parsing iteratively. Fig-

ure 3.1 shows its overview diagram. In each iteration, for every production π in the MRL grammar, KRISP collects positive and negative examples. In the first iteration, the set of positive examples for production π contains all sentences whose corresponding MRs use the production π in their parse trees. The set of negative examples includes all of the other training sentences. Using these positive and negative examples, an SVM classifier is trained for each production π using a string kernel. In subsequent iterations, the parser learned from the previous iteration is applied to the training sentences and more refined positive and negative examples, which are more specific substrings within the sentences, are collected for training. Iterations are continued until the classifiers converge. The following sections describe KRISP in more details.

3.2 KRISP's Semantic Parsing

In this section we describe how KRISP does semantic parsing using string classifiers, and in the next section we describe KRISP's training algorithm which is used to train these string classifiers.

KRISP does semantic parsing using the notion of a *semantic derivation* of an NL sentence. In the following subsections, we define the semantic derivation of an NL sentence and its probability. The task of semantic parsing then is to find the most probable semantic derivation of an NL sentence which is determined using an extended version of Earley's algorithm for context-free grammar parsing.

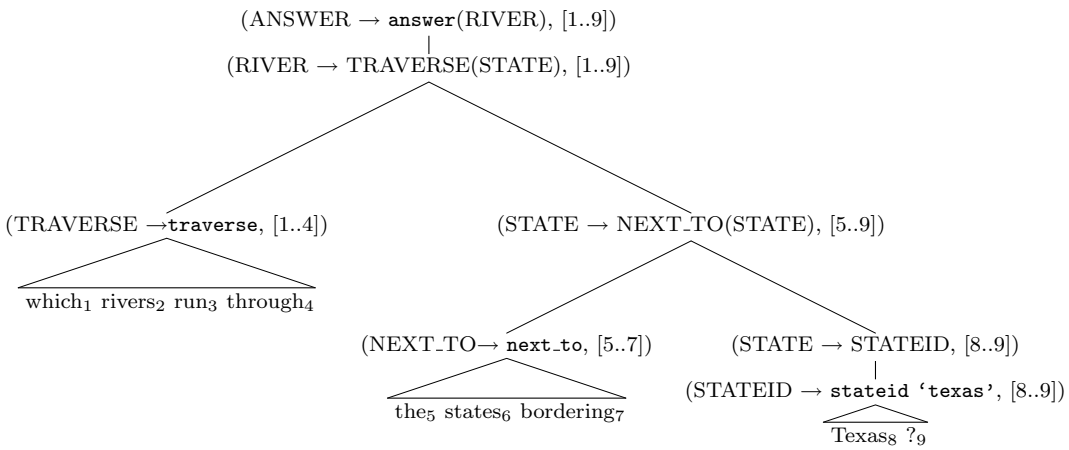


Figure 3.2: Semantic derivation of the NL sentence *"Which rivers run through the states bordering Texas?"* which gives MR `answer( traverse( next_to( stateid( 'texas' ))))`.

3.2.1 Semantic Derivation

We define a *semantic derivation*, D, of an NL sentence, s, as a parse tree of an MR (not necessarily the correct MR) 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 (i.e. the substring from the ith word to the jth word), 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. Figure 3.2 shows a semantic derivation of an NL sentence and an MR parse. The words are numbered according to their position in the sentence. Instead of non-terminals, productions are shown

in the nodes to emphasize the role of productions in semantic derivations. The substrings $s[i..j]$ covered by each production are shown by $[i..j]$ in its node.

Sometimes, the children 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 semantic derivation. For example, if the sentence was *"Through the states that border Texas which rivers run?"*, which has the same MR as the sentence in Figure 3.2, then the order of the children of the node "RIVER $\rightarrow$ TRAVERSE(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 have been permuted.

Note that given a semantic derivation of an NL sentence, it is trivial to obtain the corresponding MR which is simply the string generated by the parse. Since children nodes may be permuted, this step also needs to permute them back to the way they should be according to the MRL productions. If a semantic derivation gives the correct MR of the NL sentence, then we call it a *correct semantic derivation*, otherwise it is an *incorrect semantic derivation*.

3.2.2 Most Probable Semantic Derivation

Let $P_\pi(s[i..j])$ denote the probability that a production π of the MRL grammar covers the NL substring $s[i..j]$. In other words, the NL substring $s[i..j]$ expresses the semantic concept of a production π with probability $P_\pi(s[i..j])$. In the next section, we will describe how KRISP obtains these probabilities using string-kernel based SVM classifiers. Assuming these probabilities are

independent of each other, the probability of a semantic derivation D of a sentence s is then the product of the probabilities of all the productions in it covering their respective substrings of the sentence:

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

We note that without normalization, the above does not form a probability distribution, but since these probabilities are used only for ranking the semantic derivations, normalization is not done. The task of the semantic parser is to find the most probable semantic derivation of a sentence s. This task can be recursively performed using the notion of a *partial derivation* $E_{n,s[i..j]}$, which stands for a subtree of a semantic 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 example, the subtree rooted at the node "(STATE $\rightarrow$ NEXT_TO(STATE),[5..9])" in the derivation shown in Figure 3.2 is a partial derivation which would be denoted as $E_{STATE,s[5..9]}$. Note that the derivation D of sentence s is then simply $E_{start,s[1..|s|]}$, where *start* is the start symbol of the MRL's context free grammar, G.

Our procedure to find the most probable partial derivation $E^*_{n,s[i..j]}$ considers all possible subtrees whose root production has n as its LHS non-terminal and which cover s from index i to j. Mathematically, the most probable partial derivation $E^*_{n,s[i..j]}$ is recursively defined as:

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

where $partition(s[i..j], t)$ is a function which returns the set of all partitions of $s[i..j]$ with t elements including their permutations. 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. For example, (*"the states bordering", "Texas ?"*) is a partition of the substring *"the states bordering Texas ?"* with 2 elements. The procedure $makeTree(\pi, (p_1, .., p_t))$ 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 $(p_1, .., p_t)$.

The most probable partial derivation $E^*_{n,s[i..j]}$ is found using the above equation by trying all productions $\pi = n \rightarrow n_1..n_t$ in G which have n as the LHS, and all *partitions* with t elements of the substring $s[i..j]$ (n_1 to n_t are right-hand-side (RHS) non-terminals of π, terminals do not play any role in this process and are not shown for simplicity). The most probable partial derivation $E^*_{STATE,s[5..9]}$ for the sentence shown in Figure 3.2 is found by trying all the productions in the grammar with STATE as the LHS, for example, one of them being "STATE → NEXT_TO STATE". Then for this sample production, all partitions, (p_1, p_2), of the substring $s[5..9]$ with two elements will be considered, and the most probable derivations $E^*_{NEXT\_TO,p_1}$ and E^*_{STATE,p_2} will be found recursively. The recursion reaches base cases when the productions that 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.

According to the equation, a production $\pi \in G$ and a partition $(p_1, .., p_t) \in$

$partition(s[i..j], t)$ will be selected in constructing the most probable partial derivation. These will be the ones which maximize the product of the probability of π covering the substring $s[i..j]$ with the product of probabilities of all the recursively found most probable partial derivations consistent with 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 (1970) context-free grammar parsing algorithm it can be implemented efficiently. We describe this extended version of the algorithm in the next subsection. 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 the 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.

3.2.3 Extended Earley's Algorithm for Computing the Most Probable Semantic Derivation

Parsing a sentence s by Earley's (1970) 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 for computing the most probable semantic derivation of a sentence.

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 $({}_5\text{STATE} \rightarrow \text{NEXT\_TO} \bullet_8 \text{STATE}, 0.88)$. 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 3.3 gives the extended Earley's algorithm, EXTENDED_EARLEY, 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 $\omega = 10$ in our experiments), the algorithm will do a greedy approximation search to find the ω most probable 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.

function EXTENDED_EARLEY(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($({}_iA \rightarrow \alpha\ B\ \bullet_k\ \beta, p*q)$))
   INSERT($({}_iA \rightarrow \alpha\ B\ \bullet_k\ \beta, p*q)$, $chart[k]$)
  elseif $(r = P_{A\rightarrow\alpha\ B\ \beta}(s[i..k-1]) \geq \theta)$ **then**
   INSERT($({}_iA \rightarrow \alpha\ B\ \bullet_k\ \beta, p*q*r)$, $chart[k]$)
procedure INSERT($state$, $chart[j]$)
 if ($state$ is not already in $chart[j]$) **then**
  BEAM_PUSH($state$, $chart[j]$)

Figure 3.3: The extended Earley's algorithm for obtaining the most probable semantic derivation.

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. Out of the possible substrings the predicted state can cover, if there is no substring which can be covered with probability greater than a threshold θ by the production of predicted state, then that predicted state is not included in the chart to prevent very low probability parses. In our experiments, the value of θ was 0.5.

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 the threshold θ. 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 states are maintained for each

of the productions starting and ending at the same places in the sentence and with dots at the same positions. 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

In this section, we describe how KRISP learns the classifiers which give the probabilities $P_\pi(u)$ needed for semantic parsing as described in the previous section. Given the training corpus of NL sentences paired with their MRs $\{(s_i, m_i)|i = 1..N\}$, KRISP first parses the MRs using the MRL grammar, G. We represent the parse of MR, m_i, by $parse(m_i)$.

Figure 3.4 shows pseudo-code for KRISP's training algorithm. KRISP learns a semantic parser iteratively, each iteration improving upon the parser learned in the previous iteration. In each iteration, for every production π of G, KRISP collects positive and negative example sets. In the first iteration, the set $\mathcal{P}(\pi)$ of positive examples for production π contains all 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 of the remaining training sentences.

Using these positive and negative examples, an SVM classifier[1], C_π, is

[1]We use the LIBSVM package available at: `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

function TRAIN_KRISP(training corpus $\{(s_i, m_i)|i = 1..N\}$,
MRL grammar G)
// collect positive and negative examples for the first iteration
for each $\pi \in G$
  for $i = 1$ **to** N **do**
    if π is used in $parse(m_i)$ **then**
      include s_i in $\mathcal{P}(\pi)$
    else include s_i in $\mathcal{N}(\pi)$
for iteration $= 1$ **to** MAX_ITR **do**
  for each $\pi \in G$ **do**
    $C_\pi = trainSVM(\mathcal{P}(\pi), \mathcal{N}(\pi))$ // SVM training
  for each $\pi \in G$ $\mathcal{P}(\pi) = \Phi$ // empty the positive examples
  for $i = 1$ **to** N **do**
    $D =$EXTENDED_EARLEY(s_i, G, P) // obtain best derivations
    if $\nexists\, d \in D$ such that $parse(m_i) = getMR(d)$ **then**
      // if no correct derivation then force to find one
      $D = D \cup$ EXTENDED_EARLEY_CORRECT(s_i, G, P, m_i)
    $d^* = \arg\max_{d \in D \& getMR(d) = parse(m_i)} P(d)$
    // collect positives from maximum probability correct derivation
    COLLECT_POSITIVES(d^*)
    for each $d \in D$ **do**
      if $P(d) > P(d^*)$ **and** $getMR(d) \neq parse(m_i)$ **then**
      // collect negatives from incorrect derivation with larger
      // probability than the correct one
      COLLECT_NEGATIVES(d, d^*)
return classifiers $\mathcal{C} = \{C_\pi|\pi \in G\}$

Figure 3.4: KRISP's training algorithm

trained for each production π using a normalized string subsequence kernel. Following the framework of Lodhi et al. (2002), we define a kernel between two strings as the number of common subsequences they share. One difference, however, is that their strings are over characters while our strings are over words. A word subsequence kernel like this has been previously used by Cancedda et al. (2003) for text classification. The more the two strings share, the greater the similarity score will be. String subsequence kernels have been previously used with success in natural language processing for text classification (Lodhi et al., 2002; Cancedda et al., 2003) and relational information extraction (Bunescu & Mooney, 2005b). We use them here for semantic parsing.

Although any string classifier can be used in our system, kernel-based classifiers are particularly suitable because semantic parsing involves mapping phrases of NL sentences to semantic concepts in 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 classifiers or even statistical feature-based classifiers 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. By using a string-subsequence kernel, KRISP implicitly uses a very large number of word subsequences as features. In Chapter 4, we will describe a modification to KRISP which uses syntactic tree-kernels. We

use SVMs for kernel-based classifiers because they offer the advantage of being resistant to overfitting in such high dimensional feature spaces (Vapnik, 1998).

Normally, SVM classifiers only predict the class of the test example but one can obtain class probability estimates 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). The classifier C_π then gives us the probabilities $P_\pi(u)$. We represent the set of these classifiers by $\mathcal{C} = \{C_\pi | \pi \in G\}$.

Next, using these classifiers, the extended Earley's algorithm, denoted by EXTENDED_EARLEY in the pseudo-code, is invoked to obtain the ω best semantic derivations for each of the training sentences. The procedure $getMR$ returns the MR for a semantic derivation. At this point, for many training sentences, the resulting most-probable semantic derivation may not give the correct MR. Hence, next, the system collects more refined positive and negative examples to improve the result in the next iteration. It is also possible that for some sentences, none of the obtained ω derivations give the correct MR. But as will be described shortly, the most probable derivation that 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, EXTENDED_EARLEY_CORRECT, is invoked which also takes the correct MR as an argument and returns the best ω derivations it finds, all of which give the correct MR. This is easily done by making sure all subtrees derived in the process are present in the parse of the correct MR.

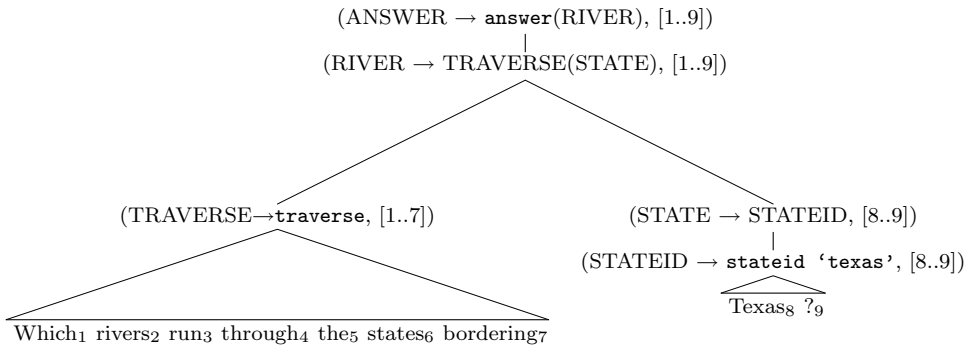


Figure 3.5: An incorrect semantic derivation of the NL sentence *"Which rivers run through the states bordering Texas?"* which gives the incorrect MR `answer(traverse(stateid('texas')))`.

From these derivations, positive and negative examples are collected for the next iteration as follows. Positive examples are collected from the most probable derivation which gives the correct MR. Figure 3.2 showed an example of a derivation which gives the correct MR. At each node in such a derivation, the substring covered is taken as a positive example for its 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 3.5 shows an example of an incorrect derivation for the same sentence. 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 3.2 and 3.5

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 used to collect negative examples. In the example, the node "(TRAVERSE→`traverse`,[1..7])" will be used to collect a 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→`traverse`) 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 and drop below the probability of the correct derivation.

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

ative examples. To further increase the number of negative examples, the positive examples of a production π are added as negative examples for all the other productions which have the same LHS non-terminal as π. Sharing same LHS non-terminal indicates that the MR parse subtrees rooted at those productions can be used in the same places in MR parse trees, hence this step prevents the rest of the productions from occurring in the semantic derivations where π should occur. After a specified number of MAX_ITR iterations, the trained classifiers from the last iteration are returned. Testing involves using these classifiers to generate the most probable derivation of a test sentence as described in the previous section, and returning its MR.

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 trained in 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.

The MRL grammar may contain productions corresponding to constants of the domain, for e.g., state names like "STATEID → `'texas'`", or river names like "RIVERID → `'colorado'`" etc. 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 example, the user may give "Texas" as the constant substring for the production "STATEID → `'texas'`. Then KRISP does not learn classifiers for these constant pro-

ductions and instead decides if they cover a substring of the sentence or not by matching it with the provided 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 substring (e.g. "RIVERID → `colorado`" and "STATEID → `colorado`"), then all of them get probability equal to 1 and the maximum probability semantic derivation gets decided based on the rest of the context. A constant production is allowed to cover extra words in the sentence with same probability,[2] but none of these extra words 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.

3.4 Experiments

3.4.1 Methodology

KRISP was evaluated on two domains: CLANG and GEOQUERY which were described in Chapter 2. The CLANG corpus has 300 NL-MR pairs and the GEO880 has 880 NL-MR pairs. Table 3.1 shows some statistics about

[2]The parent productions above the constant productions in semantic derivations, which would also cover those words, will however prevent the constant productions from covering too many extra words.

Statistic	CLang	Geo880
No. of examples	300	880
Average NL sentence length	22.52	7.48
Average MR length (tokens)	13.42	6.47
No. of non-terminals	16	44
No. of non-constant productions	102	133
No. of unique NL tokens	337	270

Table 3.1: Some statistics of the corpora used for evaluation.

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.

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 for 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:

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

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

A combined score of precision and recall, called F-measure, is obtained by taking their harmonic mean:

$$F\ measure = \frac{2 * Precision * Recall}{Precision + Recall}$$

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 output MRs (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 10, the minimum probability threshold θ was fixed to 0.5 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 6, beyond this we found that the system only overfits the training corpus and gives no benefit on testing. Since the NL sentences in the GEO880 corpus are more compact with very few extra words, than the NL sentences in the CLANG corpus, we kept the kernel parameter, λ (see Subsection 2.2.2), which penalizes gaps in the subsequences equal to 0.75 for experiments with the GEO880 corpus, and equal to 0.25 for CLANG corpus, although the performance does not change significantly by varying this parameter unless it is at its extreme values.

We compared our system's performance with the systems described briefly in the previous chapter: WASP (Wong, 2005), SCISSOR (Ge & Mooney,

2005), the system by Zettlemoyer and Collins (2007) and CHILL (with COCKTAIL ILP algorithm (Tang & Mooney, 2001)). WASP and SCISSOR also assign 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 the system by Zettlemoyer and Collins (2007) 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.

3.4.2 Results and Discussion

Figure 3.6 shows the results on the CLANG corpus. Note that precision is shown on the y-axis starting from 50%. KRISP gives slightly lower precision than WASP but gives slightly more maximum recall. The difference between their best F-measures on the precision-recall curves were not found to be statistically significant ($p > 0.05$) based on paired t-test. SCISSOR gives much higher maximum recall and its best F-measure was found significantly better ($p < 0.05$) than KRISP's best F-measure. However, we note that SCISSOR requires more supervision for the training corpus in the form of semantically annotated syntactic parse trees for the training sentences. CHILL could not be run beyond 160 examples because its Prolog implementation runs out of memory. For 160 examples, it gave 49.2% precision with 12.67% recall. Figure 3.7 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

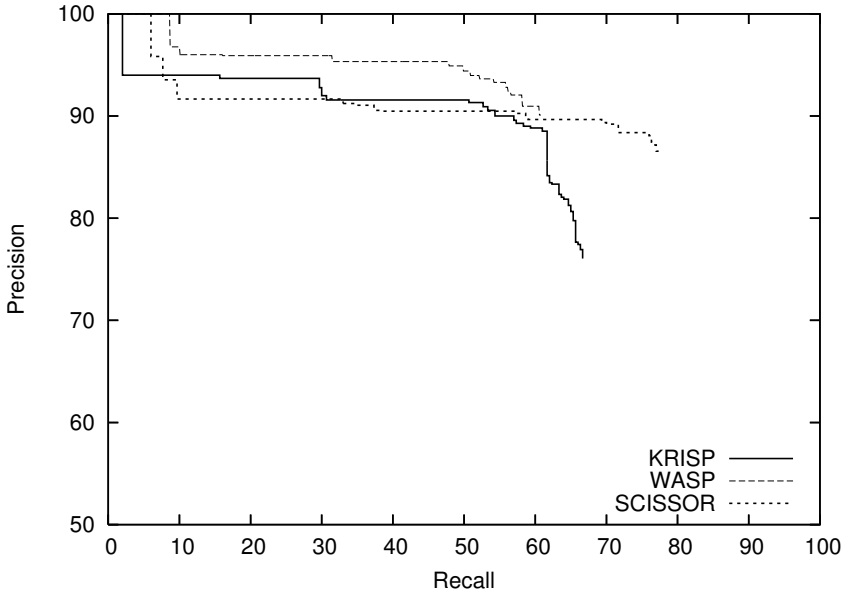


Figure 3.6: Results on the CLang corpus.

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 on the Geo880 corpus are shown in Figure 3.8. On this corpus, Krisp obtains higher precision than Wasp at lower recall values but Wasp obtains higher maximum recall. Both these systems obtain performance very close to that of Scissor. The difference between their best F-measures was not found to be statistically significant ($p > 0.05$) based on paired t-test. Chill is able to obtain higher recall. The system by Zettlemoyer and Collins (2007) is able to achieve higher precision and recall, but we note that it relies on some initial hand-written rules for lexical acquisition. Figure 3.9 shows the

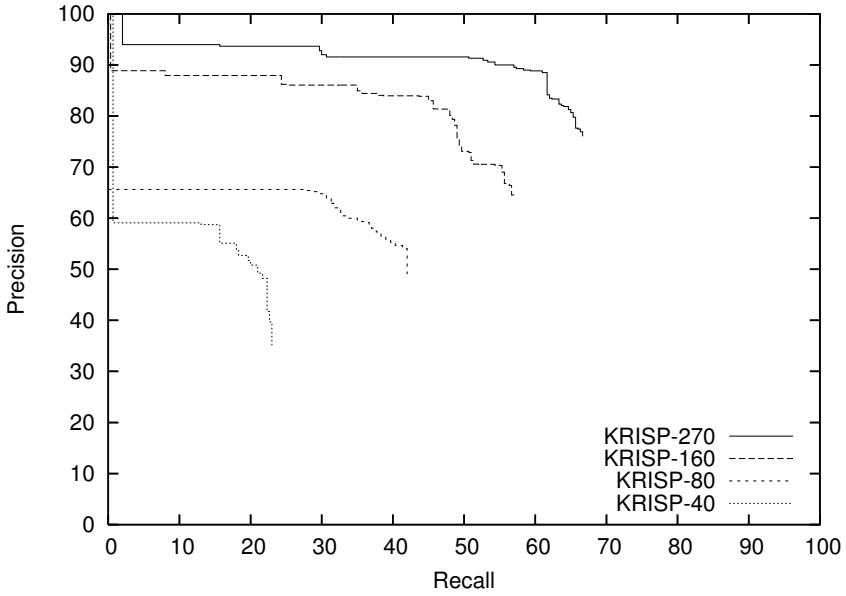


Figure 3.7: Results on the CLang corpus when trained on increasing number of training examples.

precision-recall curves for Krisp when it is trained on increasing number of training examples in each fold. It can be seen that the learning has reached very close to convergence on this corpus.

In Figure 3.10 we have shown how the performance of Krisp changes with the number of iterations it makes in its training algorithm. It can be seen that after the first iteration, in which it collects entire sentence as positive and negative examples for its classifiers, the performance is very poor. But after the second iteration, in which it collects more refined positive and negative examples, the performance shows a big improvement. With subsequent iterations, the performance improves by only a small amount.

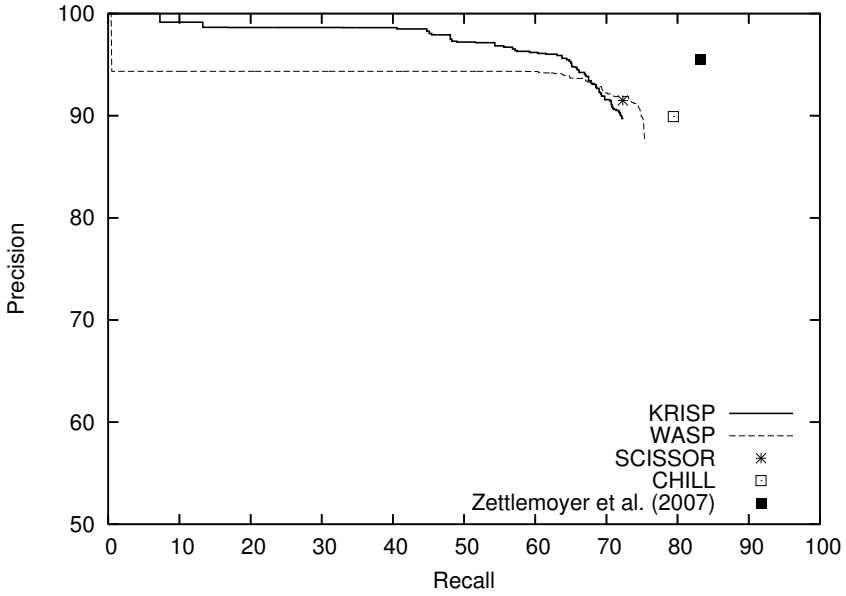


Figure 3.8: Results on the GEO880 corpus.

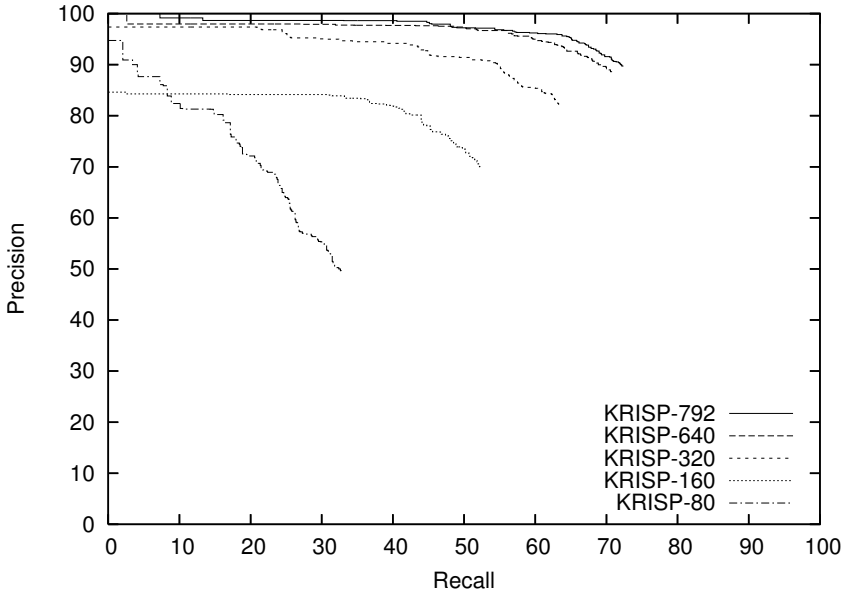


Figure 3.9: Results on the GEO880 corpus when trained on increasing number of training examples.

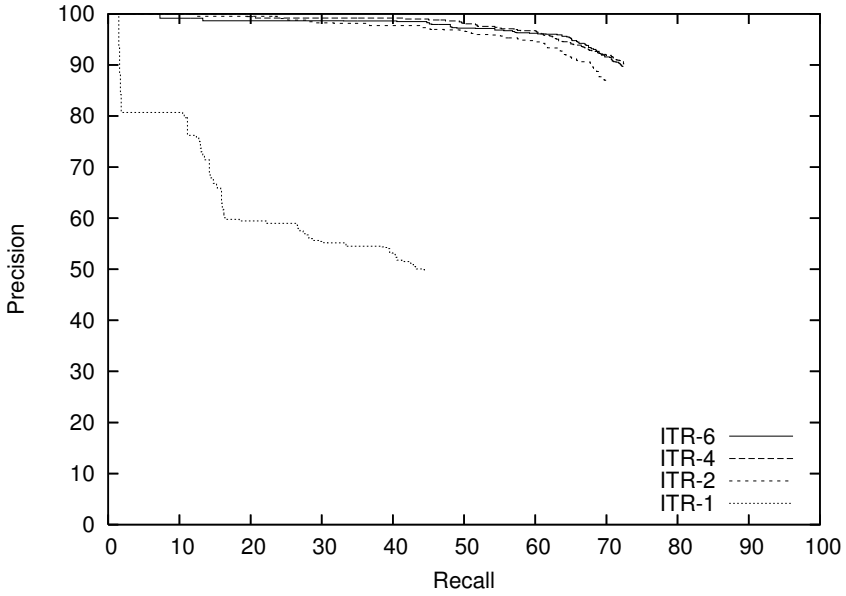


Figure 3.10: Results on the GEO880 corpus with increasing number of iterations of the of KRISP's training algorithm.

KRISP took on average 8 minutes and 26 seconds to run each fold of the GEO880 corpus, and on the CLANG corpus it took on average 45 minutes and 54 seconds on state-of-the-art cluster processors. Although GEO880 has more examples than CLANG, but it takes less time to run because KRISP spends the majority of the time in parsing the sentences which depends on the sentence length.[3] This shows that sentence length is currently the bottleneck for KRISP's efficiency and it will be easier to scale-up KRISP with the number of training examples.

[3]Earley's algorithm runs in cubic time in the length of the sentence

3.4.3 Robustness to Noise

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). As opposed to the other semantic parser learning systems, KRISP's string-kernel-based semantic parsing does not use grammar rules for natural language, probabilistic or otherwise, and does not use hard-matching rules for classification, hence it should be more flexible and robust to noise. We tested this hypothesis by running experiments on data which was artificially corrupted with simulated speech recognition errors.

The interjections, environment noise etc. are likely to be recognized as real words by a speech recognizer. To simulate this, after every word in a sentence, with some probability P_{add}, an extra word is added which is chosen with probability proportional to its word frequency found in the British National Corpus (BNC), a good representative sample of English. A speech recognizer may sometimes completely fail to detect words, so with a probability of P_{drop} a word is sometimes dropped. A speech recognizer could also introduce noise by confusing a word with a high frequency phonetically close word. We simulate this type of noise by substituting a word in the corpus by another word,

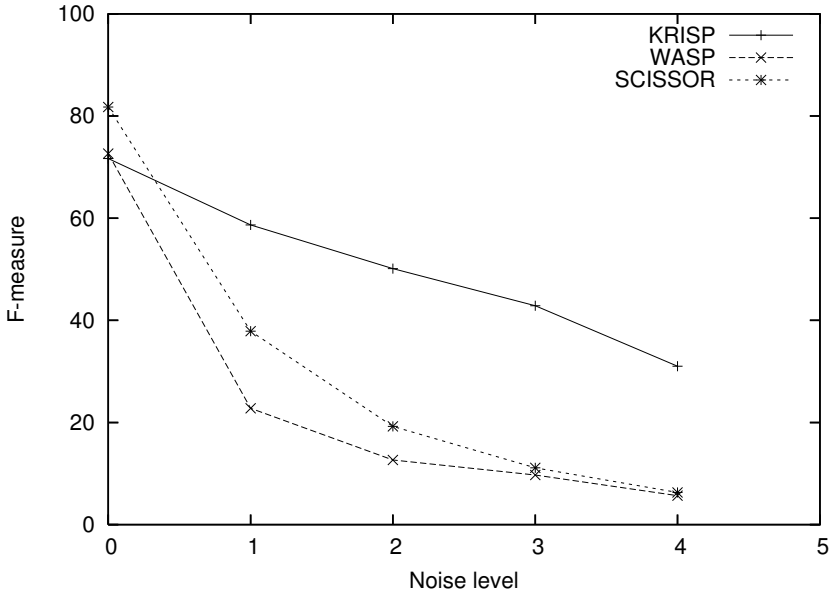


Figure 3.11: Results on the CLANG corpus with increasing amounts of noise in the test sentences.

w, with probability $p^{ed(w)} * P(w)$, where p is a parameter, $ed(w)$ is w's edit distance (Levenshtein, 1966) from the original word and $P(w)$ is w's probability proportional to its word frequency. The edit distance which calculates closeness between words is character-based rather than based on phonetics, but this should not make a significant difference in the experimental results.

Figure 3.11 shows the results on the CLANG corpus with increasing amounts of noise, from level 0 to level 4. The noise level 0 corresponds to no noise. The noise parameters, P_{add} and P_{drop}, were varied uniformly from being 0 at level 0 and 0.1 at level 4, and the parameter p was varied uniformly from being 0 at level 0 and 0.01 at level 4. We are showing the best F-measure for each system at different noise levels. As can be seen, KRISP's performance

degrades gracefully in the presence of noise while other systems' performance degrade much faster, thus verifying our hypothesis. In this experiment, only the test sentences were corrupted, we get qualitatively similar results when both training and test sentences are corrupted. The results are also similar on the GEOQUERY corpus.

3.4.4 Semantic Parsing with Different Natural Languages

We have translations of the original GEOQUERY corpus with 250 examples, which we call GEO250 corpus, in three other natural languages: Japanese, Spanish and Turkish. 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 3.12 shows results of running KRISP on other natural languages. The performance on English and Spanish are comparable. Japanese gives somewhat low precision, 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 lower recall, we believe it is because Turkish has larger number of unique tokens (36% more than English) due to its complex agglunative morphology, which makes learning from its corpus less general.

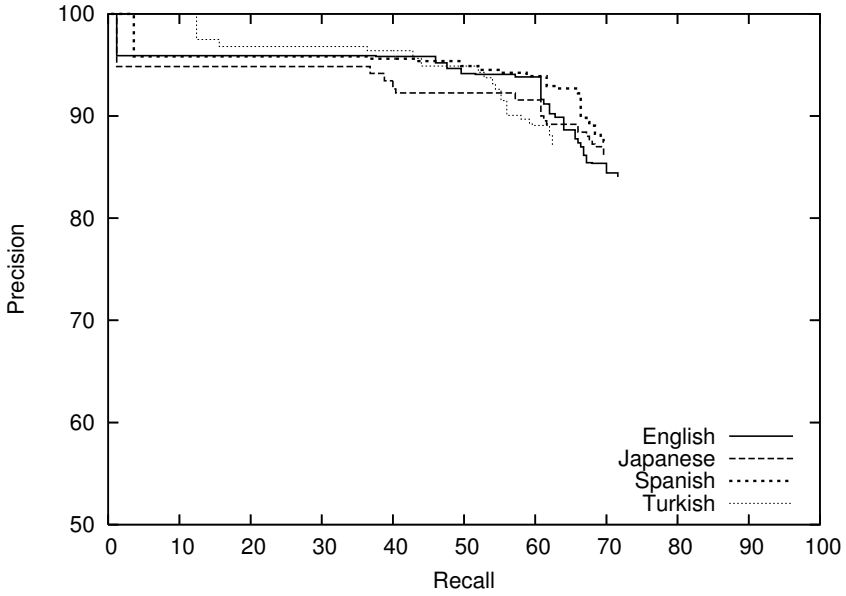


Figure 3.12: Results of KRISP on GEO250 corpus for different natural languages.

3.5 Chapter Summary

In this chapter we presented our new kernel-based approach to learn semantic parsers. It trains SVM classifiers based on string subsequence kernels 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 showed that our system compares favorably to other existing systems and is particularly robust to noise.

Chapter 4

Utilizing More Supervision

The training data for learning the semantic parsers described in the previous chapter consisted of a corpus of natural language (NL) sentences paired with their meaning representations. If the corpus-builders were willing to provide more supervision in the form of syntactic or semantic annotations for the natural language sentences, then the learning system should be able to utilize them and learn better semantic parsers. In this chapter, we show how KRISP utilizes two forms of extra supervision - syntactic parse trees and semantically augmented parse trees.

4.1 Utilizing Syntactic Parse Trees

Given that the 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 help in its semantic parsing. If syntactically annotated parse trees are provided for the training sentences then existing syntactic parsers, like that of Bikel (2004), can be trained with the given training data in addition to the Wall Street Journal (WSJ) corpus to obtain syntactic parse trees for the test sentences. This was also done in the tree-

based version of SILT (Kate et al., 2005). In order to exploit the NL syntax, the most natural extension of the string-kernel-based approach is to make it tree-kernel-based by defining a tree kernel over syntactic parse trees.

4.1.1 Tree Kernels

Syntactic-tree-kernels were first introduced by Collins and Duffy (2001) and were also used by Collins (2002a) for the task of re-ranking 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 4.1 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. In our system, we use an efficient algorithm to compute tree kernels introduced by Moschitti (2006) which runs in close to linear time in the size of the input trees.

4.1.2 Using Tree Kernels in Krisp

The string-kernel-based KRISP described in the previous chapter calculates the similarity between two NL substrings as the number of common subsequences between them. In order to use the syntactic information, we modified KRISP so that the similarity between two NL substrings takes into

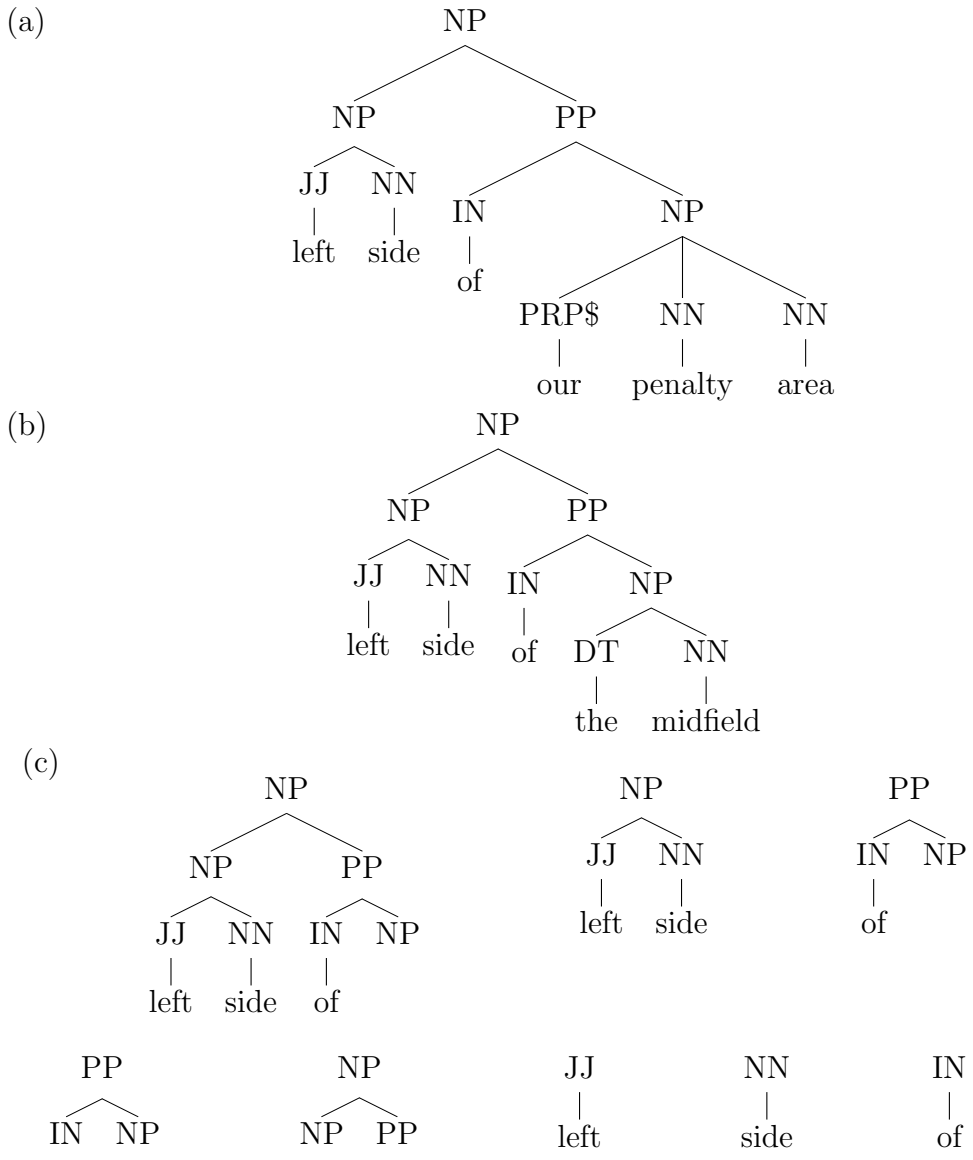


Figure 4.1: (a) and (b) are two example syntactic parse trees, (c) shows all the common subtrees between them.

account their respective syntactic structures.

The syntactic structure of an NL substring is computed from the syntactic parse of the full NL sentence in the following way. The words of the NL substring appear as a subset of leaves in the syntactic parse. First, the lowest common ancestor of these leaves is determined. Then, the smallest subtree rooted at this node which includes all the leaves of the NL substring and includes entire productions at every internal node is found. This is taken as the syntactic structure of the NL substring. For example, in Figure 4.1, the syntactic structure of the substring "left side of" of the tree shown in (a) will be the first tree shown in (c). Notice that the NP on the far right is included to cover the production for PP at its parent but none of the leaves under it is included.

Similarity between two NL substrings is then computed as the tree kernel between their syntactic structures. KRISP uses the training and test sentences only to find the similarity between various substrings, hence in order to use syntactic parses, substituting tree kernels for string kernels is the only modification needed in the entire system.

4.1.3 Experiments

Using the provided gold-standard syntactic parse trees for the training sentences and the first 21 sections of the WSJ corpus, we trained Bikel's syntactic parser (Bikel, 2004). For the test sentences, syntactic parse trees were generated using the trained Bikel's parser. Using the gold-standard syntactic

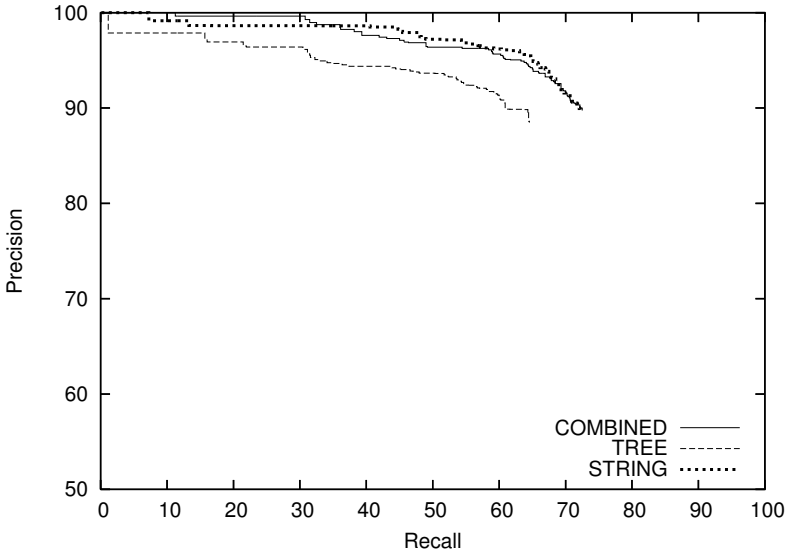


Figure 4.2: Precision-recall curves for KRISP using tree, string and combined kernels on the Geoq880 corpus.

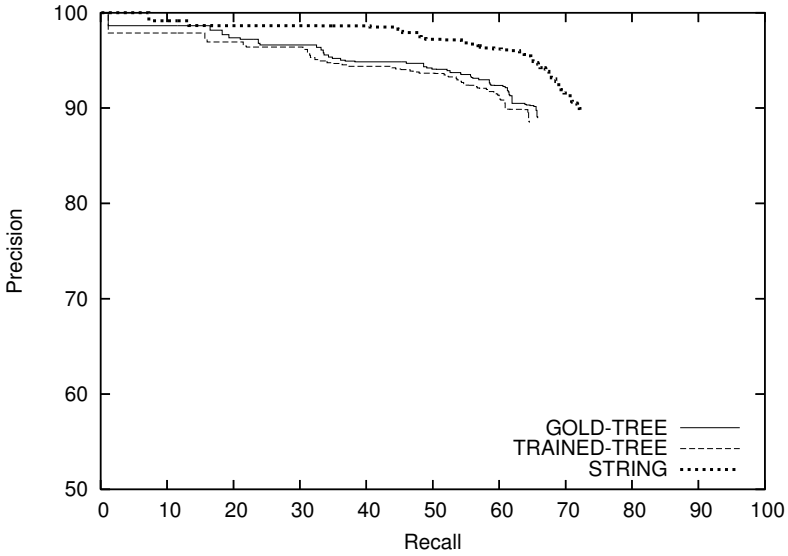


Figure 4.3: Precision-recall curves for KRISP on the Geo880 corpus using tree kernel with gold-standard syntactic parses and syntactic parses obtained from a trained syntactic parser.

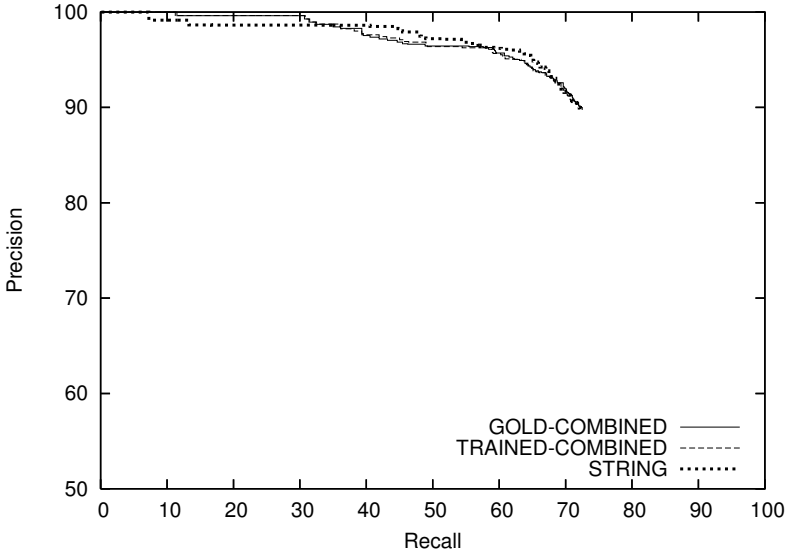


Figure 4.4: Precision-recall curves for KRISP on the Geo880 corpus using combined string and tree kernels with gold-standard syntactic parses and syntactic parses obtained from a trained syntactic parser.

parse trees of the training sentences, we trained KRISP with a tree kernel. This tree-kernel-based version of KRISP was used to semantically parse the test sentences.

Figure 4.2 shows the precision-recall curves for the Geo880 corpus generated using 10-fold cross-validation. It can be seen that tree-kernel-based KRISP performs worse than string-kernel-based KRISP. The difference betweeen the best F-measures on the precision-recall curves was found to be statistically significant ($p < 0.01$) based on paired t-test. This may be because syntactic parse trees are more detailed which makes them sparse and hence difficult to generalize from. When we combined the two kernels by simply adding the string and tree kernel values between two substrings, the

performace is close to the performance when only string-kernels are used. The combined kernel gave best F-measure of 80.3%, slightly better than 80.1%, the best F-measure given by the string kernel. Although the difference was not found to be statistically significant ($p > 0.05$).

We also ran experiments using gold-standard syntactic parse trees for the test sentences to see if the performane limitation was due to the limitations of the learned syntactic parser. Figures 4.3 and 4.4 show the precision-recall curves for the Geo880 corpus when gold standard syntactic parse trees are used for the test sentences with both tree and combined kernels. The performance thus obtained is very close to when a learned syntactic parser is used, showing that a better syntactic parser would not improve the performance substantially.

4.2 Utilizing Semantically Augmented Parse Trees

Another form of detailed supervision can be provided for training semantic parsers by semantically annotating the training sentences with semantic tags taken from the meaning representation language (MRL). For KRISP, the most useful form of such annotation would be the most suitable correct semantic derivations for the training sentences. These then can be directly used to collect positive examples during training instead of using the inferred correct semantic derivations which are improved iteratively. But we did not undertake the intensive task of manually annotating the training sentences with semantic derivations, instead we used another form of semantic annotation, called *semantically augmented parse trees*, which were already available

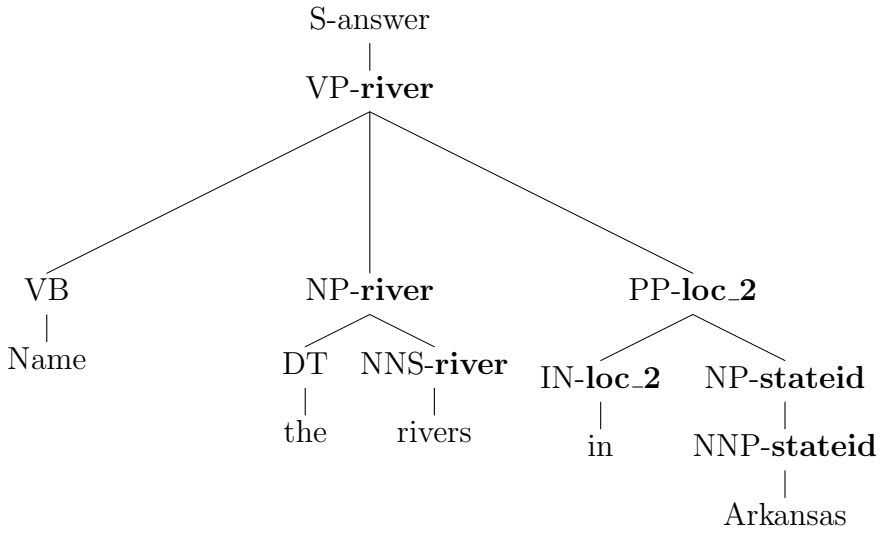


Figure 4.5: A SAPT for the sentence "Name the rivers in Arkansas" whose meaning representation is answer(river(loc_2(stateid('Arkansas')))) in the Geoquery corpus.

for our experimental corpora.

4.2.1 Semantically Augmented Parse Trees

Semantically Augmented Parse Trees (SAPTs) were introduced and used for learning semantic parsers by Ge and Mooney (2005). A SAPT for a sentence is its syntactic parse tree in which in addition to the syntactic labels, the internal nodes are augmented with semantic labels which are tokens from the meaning representation (MR) of the sentence. Figure 4.5 shows a SAPT for a sentence where the syntactic and semantic labels (in bold) on the internal nodes are separated by a hyphen symbol. Some semantically vacuous nodes do not have any semantic label.

Ge and Mooney (2005) presented an approach called SCISSOR, in which a semantic parser is trained directly with SAPTs using extensions of learning techniques used in statistical syntactic parsing. The trained semantic parser then generates SAPTs for the test sentences from which MRs are then derived. Their approach does not need MRL grammar and directly works with the MR expressions. In the next subsection, we describe an approach for using SAPTs in KRISP's training algorithm.

4.2.2 Using SAPTs in Krisp

KRISP can benefit from SAPTs by collecting positive examples in the form of substrings of the sentences for training its classifiers for the MRL productions. However, since the semantic labels used in SAPTs are tokens of the MR expressions and not the productions of the MRL, positive examples for the productions can not be directly collected. We used the following procedure to collect possible positive examples from a SAPT and the corresponding MR.

First, the leaves of the SAPT, which are words of the sentence, are numbered left to right starting from zero. These numbers are then propagated up the tree to all the nodes with semantic labels, such that each node gets a range that covers the ranges assigned to its children. Figure 4.6 (a) shows a sample SAPT along with the ranges of its nodes in square brackets. The ranges of all the semantic tokens are collected. If a semantic token occurs multiple times, the union of its ranges is taken. In the example, the token 'stateid' is assigned the range [4], 'loc_2' is assigned the range [3-4], 'river' is

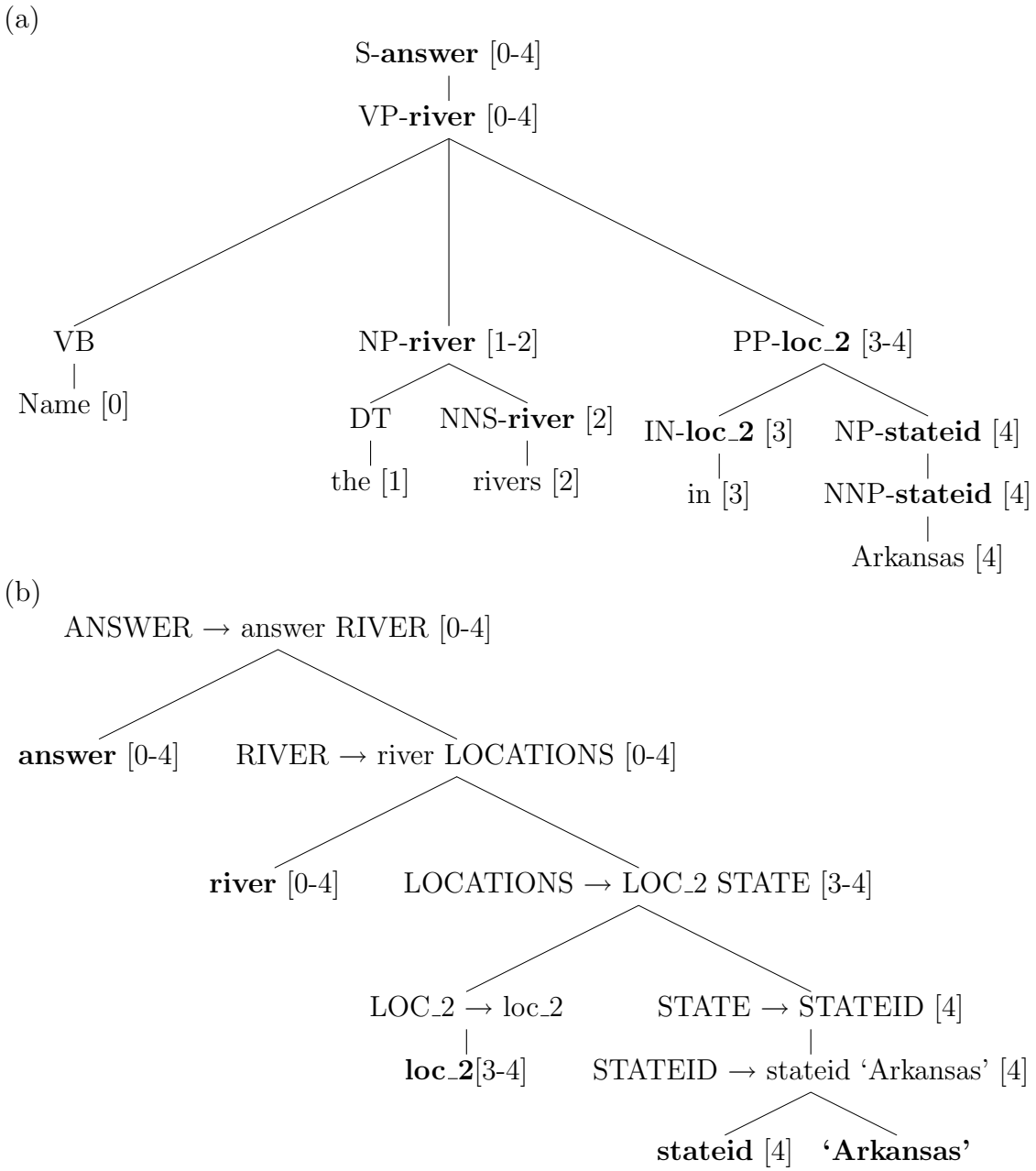


Figure 4.6: (a) A SAPT with the leaves numbered and the internal nodes with semantic labels shown with the ranges they cover. (b) An MR parse with the leaves shown with the ranges obtained from the corresponding SAPT which are propagated up the tree to collect appropriate positive examples for the productions.

assigned the range [0-4] and 'answer' is also assigned the range [0-4].

These ranges of tokens are then transfered to the tokens present at the leaves (terminals) of the corresponding MR parse tree. Figure 4.6 (b) shows the MR parse tree and the ranges at its leaves. In the figure, the MRL productions are shown in the internal nodes and the non-terminals are capitalized. These ranges are propagated up the MR parse tree, such that a parent gets the union of the ranges of its children. The positive examples for productions in the internal nodes are then collected, which are simply the substrings of the sentence with words in the associated ranges. For example, the production LOCATIONS → LOC_2 STATE will get "in Arkansas" as a positive example because its range is [3-4].

The positive examples thus collected from all of the training examples are used in the first iteration of KRISP's training algorithm, instead of using an entire sentence as a positive example for a production if the sentence's MR parse uses that production. However, as before, the negative examples in the first iteration are entire sentences.

If a token appears multiple times in an MR, then this procedure does not ensure that that the positive examples collected from the children of a node will not overlap, unlike when positive examples are collected from semantic derivations. But for the first iteration, the positive examples thus collected will still be more refined than if the entire sentence is collected as a positive example for all the productions in the MR parse, which leads to a complete overlap between all of the positive examples. It may be also noted that the

SAPTs sometimes may not have all of the MR tokens labeled. For example, in Figure 4.6 (a), the semantic token 'Arkansas' does not appear as a semantic label for any of the nodes. This may sometimes lead to undefined ranges for some productions in the MR parse, in which case the entire sentence is collected as a positive example for that production.

The positive examples collected from SAPTs are used only in the first iteration during training. For subsequent iterations, the positive examples are collected as usual from the correct semantic derivations found by the semantic parser trained from the previous iteration.

4.2.3 Experiments

We trained KRISP with semantic annotations given in the form of SAPTs for the training examples and compared its performance when it is not given SAPTs. Figure 4.7 shows the precision-recall curves for the 10-fold corss-validation on the Geo880 corpus. The performance of SCISSOR which directly trains on SAPTs is also shown for comparison. The values of the best F-measures when trained on different amounts of training data are shown in Table 4.1. It can be seen that with full training data, the performance using SAPTs is slightly worse than when not using SAPTs, although the difference between the best F-measures was not found to be statistically significant ($p < 0.05$) based on paired t-test. However, when the training data is less, giving SAPTs during training helps improve the performance. The difference between the best F-measures was found to be statistically significant ($p < 0.05$)

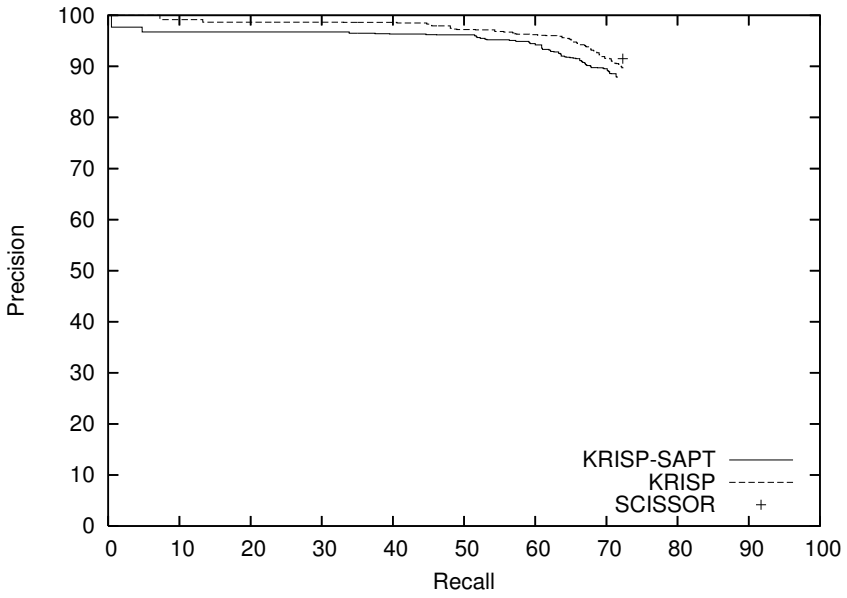


Figure 4.7: Precision-recall curves for KRISP with and without using SAPTs on the Geoq880 corpus with all the training examples in each fold. The performance of SCISSOR is also shown for comparison.

Number of training examples	40	80	160	320	640	792
KRISP's best F-measure (%)	18.05	39.48	60.00	71.74	78.71	80.13
KRISP-SAPT's best F-measure (%)	**20.07**	41.98	60.19	70.58	79.14	79.04

Table 4.1: The best-F measures obtained on the GEO880 corpus with increasing number of training examples by KRISP and when it is given SAPTs (KRISP-SAPT).

Number of iterations	1	2	4	6
KRISP's best F-measure (%)	47.06	77.57	80.53	80.12
KRISP-SAPT's best F-measure (%)	63.47	79.29	79.49	79.04

Table 4.2: The best-F measures obtained on the GEO880 corpus by KRISP and when it is given SAPTs (KRISP-SAPT) with increasing number of iterations of the training algorithm.

when the number of training examples were 40. The differences were not significant when trained with any other number of training examples.

Table 4.2 shows the best F-measures obtained with increasing number of iterations of the training algorithm. It is evident that after the first iteration, during which the system with SAPTs uses the information from SAPTs to collect better positive examples, a much better semantic parser is learned. However, with subsequent iterations, the system without SAPTs is able to learn as well as the system with SAPTs.

The increase in performance obtained using SAPTs is less than what one would have expexted, this may be due to the fact that SAPTs were designed to work with SCISSOR's training algorithm. As mentioned in the beginning of this section, annotations in the form of semantic derivations would

be more helpful to KRISP's training algorithm.

4.3 Chapter Summary

In this chapter we presented methods to utilize extra forms of supervision for learning semantic parsers. By giving KRISP more supervision in the form of syntactic or semantic annotations, its performance can be improved, although not by a large margin. This also shows that string-kernel based KRISP with no extra supervision which directly relates words and substrings to their meanings is able to internally determine the information that the extra supervision is explicitly providing.

Chapter 5

Utilizing Weaker Forms of Supervision

KRISP, and all the other learning systems for semantic parsing mentioned in chapter 2 use supervised learning methods which need annotations in the form of meaning representations (MRs) paired with natural language (NL) sentences as training data. However, it requires considerable human effort to provide such annotations. In this chapter, we consider two weaker forms of supervision: semi-supervision and ambiguous supervision, which are easier to obtain than full supervision, and describe the extensions we made to KRISP to utilize these forms of supervision.

5.1 Semi-Supervised Learning for Semantic Parsing

Semi-supervised learning methods utilize cheaply available unannotated data during training along with annotated data and often perform better than purely supervised learning methods trained on the same amount of annotated data (Chapelle, Schölkopf, & Zien, 2006). In this section we present a semi-supervised learning system for semantic parsing. The unannotated data we consider is in the form of NL sentences that are not paired with their MRs.

We modified KRISP to make a semi-supervised system we call SEMISUP-

KRISP. The key modification is the use of *transductive* Support Vector Machines (SVMs) to learn classifiers for the MRL productions instead of the normal SVMs. The next subsection gives a brief background of transductive SVMs. We then describe the learning algorithm of SEMISUP-KRISP. Finally, experiments on a real-world dataset are presented which show the improvements SEMISUP-KRISP obtains over KRISP by utilizing unannotated sentences.

5.1.1 Transductive SVMs

Given positive and negative training examples in some vector space, an SVM finds the maximum-margin hyperplane which separates them. Maximizing the margin prevents over-fitting in very high-dimensional data which is typical in natural language processing and thus leads to better generalization performance on test examples. When the unlabeled test examples are also available during training, a transductive framework for learning (Vapnik, 1998) can further improve the performance on the test examples.

Transductive SVMs were introduced in (Joachims, 1999). The key idea is to find the labeling of the test examples that results in the maximum-margin hyperplane that separates the positive and negative examples of *both* the training and the test data. Figure 5.1 shows an illustration for how the presence of unlabeled (test) examples can help in finding a better hyperplane. Such a hyperplane is found by including variables in the SVM's objective function representing labels of the test examples. Finding the exact solution for the resulting optimization problem is intractable, however Joachims (1999)

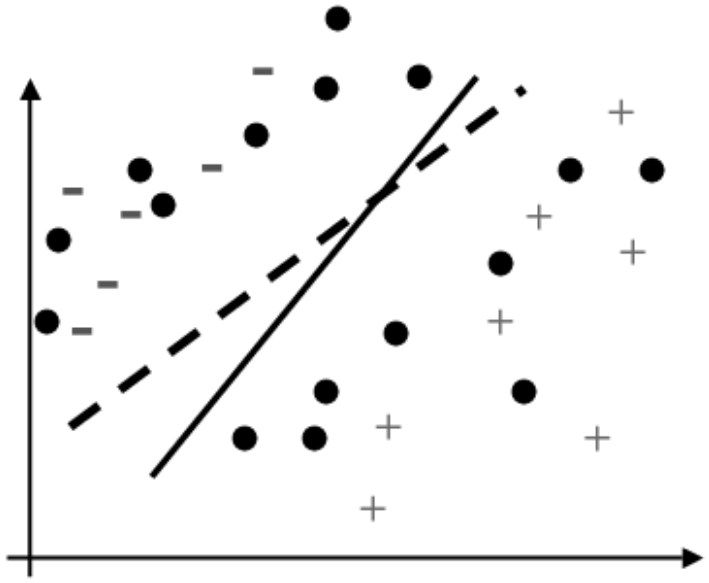


Figure 5.1: An illustration showing that the presence of unlabeled examples (dots) can help in finding a better hyperplane separating the positive (+) and negative (-) examples. An SVM will find the hyperplane shown by the solid line which maximizes the margin separating the positive and negative examples, but when the unlabeled examples are given, a transductive SVM will find the hyperplane shown by the dotted line which maximizes the margin separating all the examples.

gives an approximation algorithm for it. One drawback of that algorithm is that it requires the proportion of positive and negative examples in the test data be close to the proportion in the training data, which may not always hold, particularly when the training data is small. Chen, Wang, and Dong (2003a) present another approximation algorithm which we use in our system because it does not require this assumption. More recently, new optimization methods have been used to scale-up transductive SVMs to large data sets (Collobert, Sinz, Weston, & Bottou, 2006), however we did not face scaling problems in our current experiments.

Although transductive SVMs were originally designed to improve performance on the *test* data by utilizing its availability during training, they can also be directly used in a semi-supervised setting (Bennett & Demiriz, 1999), where unlabeled data is available during training that comes from the same distribution as the test data but is not the actual data on which the classifier is eventually to be tested. This framework is more realistic in the context of semantic parsing where sentences must be processed in real-time and it is not practical to re-train the parser transductively for every new test sentence. Instead of using an alternative semi-supervised SVM algorithm, we preferred to use a transductive SVM algorithm (Chen et al., 2003a) in a semi-supervised manner, since it is easily implemented on top of an existing SVM system.

function TRAIN_SEMISUP_KRISP(Annotated corpus $\mathcal{A} = \{(s_i, m_i)|i = 1..N\}$, MRL grammar G,Unannotated sentences $\mathcal{T} = \{t_i|i = 1..M\}$)
// obtain classifiers by training KRISP
$\mathcal{C} \equiv \{C_\pi|\pi \in G\}$ = TRAIN_KRISP(A,G)
Let
$\mathcal{P} = \{p_\pi =$ Set of positive examples used in training $C_\pi|\pi \in G\}$
$\mathcal{N} = \{n_\pi =$ Set of negative examples used in training $C_\pi|\pi \in G\}$
$\mathcal{U} = \{u_\pi = \phi|\pi \in G\}$ // set of unlabeled examples, initially all empty
for $i = 1$ **to** M **do**
$\{u_\pi^i|\pi \in G\}$ =COLLECT_CLASSIFIER_CALLS(PARSE$(t_i, \mathcal{C})$)
$\mathcal{U} = \{u_\pi = u_\pi \cup u_\pi^i|\pi \in G\}$
for each $\pi \in G$ **do**
// retrain classifiers utilizing unlabeled examples
C_π =TRANSDUCTIVE_SVM_TRAIN(p_π, n_π, u_π)
return classifiers $\mathcal{C} = \{C_\pi|\pi \in G\}$

Figure 5.2: SEMISUP-KRISP's training algorithm

5.1.2 Semi-Supervised Semantic Parsing

We modified the supervised system KRISP, described in chapter 3, to incorporate semi-supervised learning. Supervised learning in KRISP involves training SVM classifiers on positive and negative examples that are substrings of the annotated sentences. In order to perform semi-supervised learning, these classifiers need to be given appropriate unlabeled examples. The key question is: Which substrings of the unannotated sentences should be given as unlabeled examples to which productions' classifiers? Giving all substrings of the unannotated sentences as unlabeled examples to all of the classifiers would lead to a huge number of unlabeled examples that would not conform to the underlying distribution of classes each classifier is trying to separate. We had

initially tried this, and as was expected, it did not give any benefit over fully supervised learning.

SEMISUP-KRISP's training algorithm, shown in Figure 5.2, selects appropriate unlabeled examples in the following way. It first runs KRISP's existing training algorithm and obtains SVM classifiers for every production in the MRL grammar. Sets of positive and negative examples that were used for training the classifiers in the last iteration are collected for each production. Next, the learned parser is applied to the unannotated sentences. During the parsing of each sentence, whenever a classifier is called to estimate the probability of a substring representing the semantic concept for its production, that substring is saved as an unlabeled example for that classifier. These substrings are representative of the examples that the classifier will actually need to handle during testing. Note that the MRs obtained from parsing the unannotated sentences do not play a role during training since it is unknown whether or not they are correct. These sets of unlabeled examples for each production, along with the sets of positive and negative examples collected earlier, are then used to retrain the classifiers using transductive SVMs. The retrained classifiers are finally returned and used in the final semantic parser.

5.1.3 Experiments

We compared the performance of SEMISUP-KRISP and KRISP in the GEOQUERY domain for semantic parsing in which the MRL is a functional language used to query a U.S. geography database (Kate et al., 2005). Its

original corpus, GEO250, contains 250 NL queries collected from undergraduate students and annotated with their correct MRs (Zelle & Mooney, 1996). We used this data as the supervised corpus. Later, 630 additional NL queries were collected from real users of a web-based interface and annotated (Tang & Mooney, 2001). We used this additional data as *unannotated* sentences in our current experiments. We also collected an additional 407 queries from the same interface, making a total of 1,037 unannotated sentences.

The systems were evaluated using standard 10-fold cross validation. All the unannotated sentences were used for training in each fold. Performance was measured in terms of precision (the percentage of generated MRs that were correct) and recall (the percentage of all sentences for which correct MRs were obtained). An output MR is considered correct if and only if the resulting query retrieves the same answer as the correct MR when submitted to the database. Since the systems assign confidences to the MRs they generate, the entire range of the precision-recall trade-off can be obtained for a system by measuring precision and recall at various confidence levels. We present learning curves for the best F-measure (harmonic mean of precision and recall) obtained across the precision-recall trade-off as the amount of annotated training data is increased. Figure 5.3 shows the results for both systems.

The results clearly show the improvement SEMISUP-KRISP obtains over KRISP by utilizing unannotated sentences, particularly when the number of annotated sentences is small. We also show the performance of a hand-built semantic parser GEOBASE (Borland International, 1988) for comparison. From

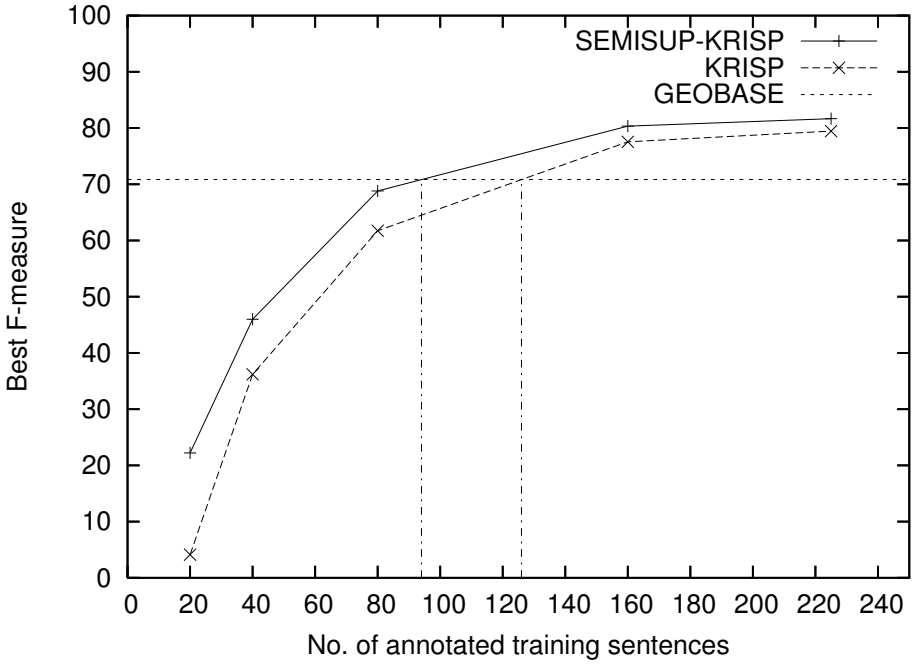


Figure 5.3: Learning curves for the best F-measures on the GEO250 corpus.

the figure, it can be seen that, on average, KRISP achieves the same performance as GEOBASE when it is given 126 annotated examples, while SEMISUP-KRISP reaches this level given only 94 annotated examples, a 25.4% savings in human-annotation effort.

5.2 Ambiguous Supervision for Semantic Parsing

Most learning systems for natural language processing require very detailed supervision in the form of human annotations such as parse trees or meaning representations for the training sentences. Ideally, a system would be able to learn language like a human child, by only being exposed to utterances

in a rich perceptual context. Frequently, the context of an utterance can be used to narrow down its interpretation to a fairly small set of reasonable alternatives. There has been some work on inferring the meanings of individual words given a corpus of sentences each paired with an ambiguous set of multiple possible meaning representations (Siskind, 1996). However, the methods developed in that work only acquire lexical semantics and do not learn how to disambiguate words and compose their meanings in order to interpret complete sentences. In this chapter, we explore the task of learning a semantic parser from ambiguous supervision, in which each sentence is annotated with an ambiguous set of multiple, alternative potential interpretations. We show how an accurate semantic parser can be learned by augmenting an existing supervised learning system to handle such ambiguous training data.

Testing such a system in a realistic setting would require a perceptual system that can construct a set of plausible meanings for a sentence from the context in which it is uttered. Since this is a difficult unsolved problem, we evaluate the system by artificially obfuscating training data previously used to assess supervised semantic-parser learners. We also evaluate our system on another artificially-created corpus that models ambiguities more realistically. Experimental results indicate that our system is able to learn accurate parsers even given such ambiguous supervision.

5.2.1 Ambiguous Supervision

We call the supervision *unambiguous* when a learning system for semantic parsing is given a corpus of NL sentences in which each sentence is paired with its respective correct MR. All the previous work on learning for semantic parsing has used only unambiguous supervision. In this type of supervision, while the learning system is not given which parts of the MRs correspond to which portions within the sentences, it is however unambiguously given which complete MRs correspond to which sentences. There are two major shortcomings with this type of detailed supervision. First, manually constructing an unambiguous corpus in which each sentence is annotated with its correct MR is a difficult task. A computer system that observes some perceptual context and is simultaneously exposed to natural language should be able to automatically learn the underlying language semantics. But since the training data available in such a setting will not consist of NL sentences unambiguously paired with their MRs, it will require human effort to build such a corpus before the learning can take place. Secondly, unambiguous supervision does not model the type of data children receive when they are learning a language. In order to learn to analyze the meaning of sentences, children have to also learn to identify the correct meaning of a sentence among the several meanings possible in their current perceptual context. Therefore, a weaker and more general form of supervision for learning semantic parsers needs to be considered.

Consider the type of supervision a system would receive when learning language semantics from perceptual contexts. We assume that the low-level

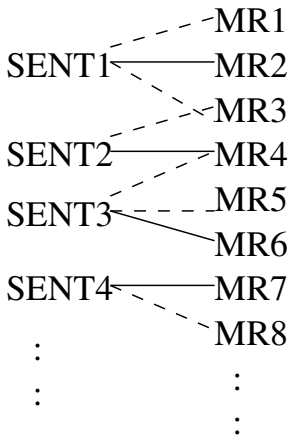


Figure 5.4: Sample Ambiguous Training Data (solid-line: correct meaning; dashed-line: possible meaning).

sensory data (real or simulated) from a system's perceptual context is first abstracted into symbolic meaning representations (MRs). Our model of supervision corresponds to the type of data that will be gathered from a temporal sequence of perceptual contexts with occasional language commentary. The training data in this model thus consists of a sequence of NL sentences and a sequence of MRs in which each NL sentence is associated with a set of one or more consecutive MRs. These associated MRs represent the general perceptual context the system was in when it registered the NL sentence. The association of multiple MRs with each NL sentence makes such a corpus ambiguous.

We assume that each NL utterance means something unique in the perceptual context, so that *exactly one* MR out of all the MRs associated with an NL sentence represents its correct meaning. Also, since different NL utterances would normally refer to different things, we assume that an MR

function TRAIN_KRISPER(Ambiguous corpus $\mathcal{A} = \{(sent_i, \mathcal{M}_i) | \mathcal{M}_i \neq \phi, i = 1..N\}$, MRL grammar G)
REDUCE_AMBIGUITIES($\mathcal{A}$) // reduce easy to resolve ambiguities
$\mathcal{U} = \phi$ // corpus for training, initially empty
for $i = 1$ **to** N **do** // collect weighted initial examples
  $\mathcal{U} = \mathcal{U} \cup \{(sent_i, mr, w) | mr \in \mathcal{M}_i, w = 1/|\mathcal{M}_i|\}$
// obtain classifiers by training KRISP
$\mathcal{C} \equiv \{C_\pi | \pi \in G\}$ = TRAIN_WEIGHTED_KRISP($\mathcal{U}$,G)
while (not converged) **do**
  $\mathcal{V} = \phi$ // collect examples with parse confidences, initially empty
  for $i = 1$ **to** N **do**
    $\mathcal{V} = \mathcal{V} \cup \{(sent_i, mr, w) | mr \in \mathcal{M}_i, w =$PARSE_ESTIMATE$(sent_i, mr, \mathcal{C})\}$
  $\mathcal{U} =$BEST_EXAMPLES($\mathcal{V}$) // find the best consistent examples
  $\mathcal{C} =$TRAIN_KRISP($\mathcal{U}$,G) // retraining
return $\mathcal{C}$ // return classifiers trained in the last iteration

Figure 5.5: KRISPER's training algorithm

can be the correct meaning of *at most one* of the sentences with which it is associated.[1] Figure 5.4 shows a small example of such a corpus. The sentences are shown connected to their possible MRs by lines. For illustration purpose, the connections between sentences and their correct MRs are shown with solid lines and the rest are shown with dotted lines. However, this distinction is obviously not included in the training data.

[1] A duplicate MR (or sentence) that reappears later in the sequence will be treated separately each time.

5.2.2 Krisper: The Semantic Parsing Learning System for Ambiguous Supervision

We extended KRISP's training algorithm to handle ambiguous supervision. We call our new system KRISPER (KRISP with EM-like Retraining). It employs an iterative approach analogous to EM (Dempster, Laird, & Rubin, 1977), where, through retraining, each iteration improves upon determining the correct MR out of the possible MRs for each sentence.

Figure 5.5 shows KRISPER's training algorithm. It takes the ambiguous corpus as input in which each NL sentence is paired with a non-empty set of MRs. Using the assumption that an MR in the corpus can be the correct meaning of at most one sentence, it first removes some easily resolved ambiguities present in the input. For example, if the input includes the examples $(sent_1, \{mr_1\})$, $(sent_2, \{mr_1, mr_2\})$ and $(sent_3, \{mr_2, mr_3, mr_4\})$, then it is clear that mr_1 can not be the correct meaning of $sent_2$, because then $sent_1$ will be left without any correct meaning, hence mr_2 must be the correct meaning of $sent_2$. This then prohibits mr_2 from being the correct meaning of $sent_3$, reducing the set of its possible MRs to $\{mr_3, mr_4\}$. In general, we use the following procedure to remove these type of ambiguities. First, we note that an ambiguous corpus forms a bipartite graph with the sentences and the MRs as two disjoint sets of vertices and the associations between them as connecting edges. The set of correct NL–MR pairs form a *matching* on this bipartite graph which is defined as a subset of the edges with at most one edge incident on every vertex (Cormen, Leiserson, & Rivest, 1990). Since all NL sentences

have a correct MR, this matching is in fact a *maximum matching* with cardinality equal to the number of sentences. In order to check whether an NL–MR pair can be in the set of the correct NL–MR pairs, our procedure removes the edge connecting the pair and the edges incident on the two vertices and sees if the maximum matching on the resulting graph includes all of the remaining NL sentences.[2] If not, then the procedure removes that NL–MR association from the corpus, because any matching that includes it will not be able to include edges to cover all of the NL sentences. All of the NL–MR pairs in the corpus are checked by this procedure, and the whole process is iterated until no additional NL–MR pairs are removed.

KRISPER then assumes that *every* MR in the set of MRs paired with each NL sentence is correct for that NL sentence and collects the resulting NL–MR paired examples. Each such example is also given a weight which is inversely proportional to the number of MRs that were associated with its NL sentence. For example, from the ambiguous input example of $(sent_3, \{mr_3, mr_4\})$, two examples, $(sent_3, mr_3)$ and $(sent_3, mr_4)$, will be collected and both will be given weight equal to 1/2. These examples are then given to KRISP's training algorithm to learn an initial semantic parser. Since the training data is noisy with many incorrect NL–MR pairs in this first iteration, the parameter of the SVM training algorithm that penalizes incorrect

[2] An $O(|V||E|)$ algorithm exists for finding a maximum matching on a bipartite graph, where $|V|$ is the number of vertices and $|E|$ is the number of edges (Cormen et al., 1990). Our procedure finds a maximum matching separately for every maximally connected subgraph of the graph which makes this step even more efficient.

classifications is kept low. This parameter is increased exponentially with each subsequent iteration. Although there will be many incorrect NL–MR pairs in the first iteration, the parser is expected to learn some regularities due to the presence of the correct pairs. The weighting procedure ensures that the more an NL–MR pair is likely to be incorrect, the less weight it receives. KRISP's existing training algorithm, however, does not accept weighted examples as input so we modified it as follows. All of the positive and negative SVM training examples that KRISP extracts from an input NL–MR pair are given the same weight as the input example. Then, a version of an SVM which takes weighted input examples is used for training. We used the tool "Weights for data instances" available in the LIBSVM package.[3]

Next, for each NL sentence in the training data, KRISPER estimates the confidence of generating each of the MRs in the sentence's set of possible MRs by calling the learned parser's PARSE_ESTIMATE function (this is essentially the same function as the EXTENDED_EARLEY_CORRECT function described in chapter 3, given an MR and a sentence, this function computes the confidence that the MR is the correct one for that sentence). For the purpose of training the parser in the next and subsequent iterations, it pairs the NL sentence with only one MR from its set, the one with the highest confidence.[4] But since the sets of possible MRs of NL sentences could overlap, the new NL–

[3] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools`

[4] We found that this works better than pairing each NL sentence with all of its associated MRs with weights proportional to their confidences because after the first iteration the system usually has a good idea of the correct NL–MR pairs and the remaining pairs only increase the noise.

MR pairs should be consistently chosen so that an MR does not get paired with more than one sentence. The problem of consistently selecting the best pairs is an instance of the *maximum weight assignment* problem on a bipartite graph which can be solved using the *Hungarian Algorithm* (Munkres, 1957) in $O(|V|^3)$ time, where $|V|$ is the number of vertices. The pairs found by this algorithm are then given to KRISP's training algorithm to learn a better semantic parser. Since now only one MR is associated with each NL sentence, the weights of all examples are set to 1. The algorithm terminates when the NL–MR pairs for an iteration differ by less than 5% (a parameter) from the pairs in the previous iteration. The parser trained on these NL–MR pairs is then returned as the final learned parser. Although we do not have a theoretical proof of convergence for our method, but empirically we observed that it converges rapidly. It never needed more than six iterations in our experiments.

5.2.3 Corpora Construction

This subsection describes how the corpora were created to experimentally evaluate KRISPER. To our best knowledge, there is no real-world ambiguous corpus available for learning semantic parsers. Hence, to evaluate our system, we constructed two ambiguous corpora: AMBIG-GEOQUERY and AMBIG-CHILDWORLD. The AMBIG-GEOQUERY corpus was constructed by artificially obfuscating the existing real-world unambiguous GEOQUERY corpus, while the AMBIG-CHILDWORLD corpus was constructed completely artificially but attempts to more accurately model real-world ambiguities.

Ambig-Geoquery Corpus

We used the unambiguous GEOQUERY corpus to artificially construct the AMBIG-GEOQUERY corpus which conforms to the model of ambiguous supervision described earlier with a sample shown in Figure 5.4. Using the MRs and NL sentences present in the unambiguous corpus, an ambiguous corpus was formed in which each NL sentence was paired with a set of multiple possible MRs. First, a sequence of randomly permuted MRs from the GEOQUERY corpus, which we call *base* MRs, was formed. Next, random MRs, chosen from the same corpus, were inserted between every pair of adjacent base MRs. The number of MRs inserted between any two base MRs was randomly chosen uniformly between 0 and α (a parameter). Next, each NL sentence from the GEOQUERY corpus was paired with a set of MRs that formed a window in the sequence centered at the sentence's correct base MR. The width of the window in either direction from the base MR was randomly chosen uniformly between 0 and β (another parameter). Since the window is around the correct MR, this ensures that there will always be a correct meaning for each sentence in its set of possible MRs.

By varying the parameters α and β, we generated three levels of ambiguity, which we call levels 1, 2 and 3. In level 1, both parameters were set to 1 and this resulted in training data that on average has 24.8% sentences associated with only one MR, 50.1% with two and 25.1% with three MRs. In level 2, both parameters were set to 2 and this resulted in an average of 11.2% sentences associated with one MR, 22.3% with two, 33.7% with three, 22%

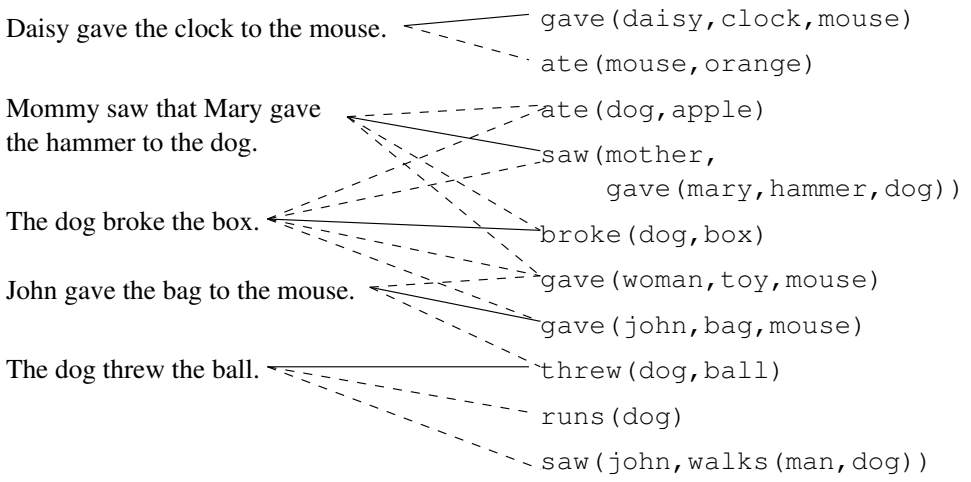


Figure 5.6: A sample of the AMBIG-CHILDWORLD corpus corresponding to a perceptual context.

with four and 10.8% with five MRs. Finally, in level 3, both parameters were set to 3 and there were on average 6% sentences associated with only one MR, 12.5% with two, 19.4% with three, 25.9% with four, 18.4% with five, 11.8% with six and 6% with seven MRs.

Ambig-ChildWorld Corpus

Although the AMBIG-GEOQUERY corpus uses real-world NL sentences and MRs, it does not model the ambiguities realistically because the MRs associated with a sentence may have nothing in common, but in a real-world perceptual context, the potential candidate meanings for a sentence will usually be related. Hence, we created another corpus, AMBIG-CHILDWORLD, which models ambiguities more realistically. It tries to mimic the type of language data that would be available to a child while learning a language.

We first constructed a synchronous context-free grammar (Aho & Ullman, 1972) to simultaneously generate simple NL sentences and their correct MRs which are in predicate logic but without quantification. The synchronous grammar relates some simple NL verbs to logic predicates and some simple NL nouns (namely people, animals and things) to the arguments of those predicates. There are 15 verbs and 37 nouns in the grammar. It can also generate a few complex sentences. The corpus is generated to model occasional language commentary on a series of perceptual contexts. A perceptual context is modelled as a collection of events happening in a room involving a few people, animals and things. The next perceptual context will involve different people, animals and things.

The data corresponding to a perceptual context was created in the following manner. First, a random subset of the synchronous grammar was extracted. Since people, animals and things are part of the grammar, this process selects random subsets of them as well. Next, a random sequence of NL sentences and their correct MRs was generated using this subset of the synchronous grammar, but only a few of the NL sentences were retained. These were chosen in such a way that the number of skipped sentences between a retained sentence and the next retained sentence in the sequence was randomly chosen uniformly between 0 and α, where α is a parameter.[5] Retaining only a few sentences models the fact that only a few events happening in the room will receive NL commentary. Next, each NL sentence was associated with a

[5]It plays the same role here as in the construction of the AMBIG-GEOQUERY corpus.

set of MRs which form a window in the sequence centered at the sentence's correct MR. The width of the window in either direction of the MR was again randomly chosen uniformly between 0 and β. This models the fact that the sentence might be referring to one of the multiple events that were happening while the sentence was being spoken.

The process of generating data then continues with a different subset of the grammar representing a different perceptual context. Since the MRs generated from a perceptual context will involve the same people, animals and things, they will usually be related, thus modelling ambiguities more realistically. From each perceptual context, 5 to 10 NL sentences with their associated MR sets were created. A sample of the AMBIG-CHILDWORLD corpus corresponding to one perceptual context is shown in Figure 5.6. For illustration, the correct NL–MR pairs are shown with solid lines and the remaining pairs with dotted lines. In this corpus there were on average 5.47 words in a sentence.

Three levels of ambiguity were created for this corpus by varying the parameters α and β similar to the way in which the three levels of ambiguity for the AMBIG-GEOQUERY corpus were created. The distributions of the number of MRs associated with NL sentences in the three levels of this corpus are also similar to the distributions in the corresponding levels of the AMBIG-GEOQUERY corpus. On the two corpora, the first step of KRISPER's training algorithm, which removes easily resolvable ambiguities, removed on average 8% of the NL–MR pairs for level 1, 3.2% for level 2 and 1.3% for level 3.

5.2.4 Experiments

Methodology

KRISPER was evaluated using standard 10-fold cross validation. For the AMBIG-GEOQUERY corpus, the original unambiguous GEOQUERY corpus was divided into ten equal parts. For each fold, one part was used for testing the accuracy of the learned semantic parser and the remaining nine parts were used to construct the ambiguous training data by the method described in the previous subsection. Since AMBIG-CHILDWORLD is artificially created, there was no scarcity of training data, hence the training data for each fold was generated separately. The testing data for each fold was also generated separately using the entire synchronous grammar, but with no ambiguity added.

There are some constants in the GEOQUERY domain, like state and city names, which appear in the NL sentences as well as in their corresponding MRs. This information is normally exploited when training KRISP, but if it is exploited in the obfuscated data then the introduced ambiguities can often be trivially resolved because most of the time only one MR out of the possible candidate MRs will have the constants which are present in the associated sentence. Therefore, we prevented KRISPER from exploiting such matching constants, making the learning task even harder because the parser now has to learn the meanings of the constants as well (which is analogous to learning the names of specific items in the world). This is also true with the AMBIG-CHILDWORLD corpus.

We used GEO250, the original GEOQUERY corpus containing 250 NL

queries annotated with their correct MRs (Zelle & Mooney, 1996) as the unambiguous corpus. Since learning accurate parsers from ambiguous data obviously requires more examples than when using unambiguous data, we also tried artificially increasing the size of the AMBIG-GEOQUERY training set by replicating examples in the training set after changing their constants to other constants of the same type (e.g. changing a state name to another state name). The other constants were always chosen from within the training corpus being replicated. In our experiments, the corpus was replicated two, three and four times for each of the three levels of ambiguities. We also created the training data for the AMBIG-CHILDWORLD corpus for each ambiguity level in the same four sizes: each fold containing 225, 450, 675 and 900 examples. The testing corpus was same for all the sizes, each fold containing 25 examples.

For higher levels of ambiguity when the training size is large, the number of NL–MR pairs generated in the first iteration of KRISPER's training algorithm becomes very large. To save running time, we divide the training data to result in size of around 500 NL–MR pairs in each part. Then the first iteration of the algorithm is run on each part separately. However, this is not done in the subsequent iterations and the algorithm is run on the entire training data. In our experiments, KRISPER's training algorithm did not require more than six iterations to converge.

Performance of semantic parsing was again measured in terms of precision (the percentage of generated MRs that were correct) and recall (the percentage of all sentences for which correct MRs were obtained). For AMBIG-

GEOQUERY corpus, an output MR is considered correct if the resulting query retrieves the same answer as the correct MR when submitted to the database, and for AMBIG-CHILDWORLD corpus, an output MR is considered correct if it exactly matches the correct MR.

Since KRISPER assigns confidences to the MRs it outputs, an entire range of the precision-recall trade-off can be obtained by varying the confidence threshold for accepting a parse. We present the results in the form of learning curves for the best F-measure (harmonic mean of precision and recall) across the precision-recall tradeoff.

Results and Discussion

Figure 5.7 shows the results obtained by training KRISPER on the AMBIG-GEOQUERY training data with ambiguity levels 1, 2 and 3. The results obtained when the training data is unambiguous are also shown for comparison. When smaller number of training examples are given, the performance on ambiguous training data for all the levels is worse than the performance on unambiguous training data. But as the amount of ambiguous training data is increased, despite the weak form of supervision, KRISPER starts to learn as accurate a parser as with the same amount of unambiguous training data. The learning curve for level 3 shows a large performance gain but it has not yet converged.

Figure 5.8 shows the results obtained when training KRISPER on the AMBIG-CHILDWORLD data with the three ambiguity levels. The results for

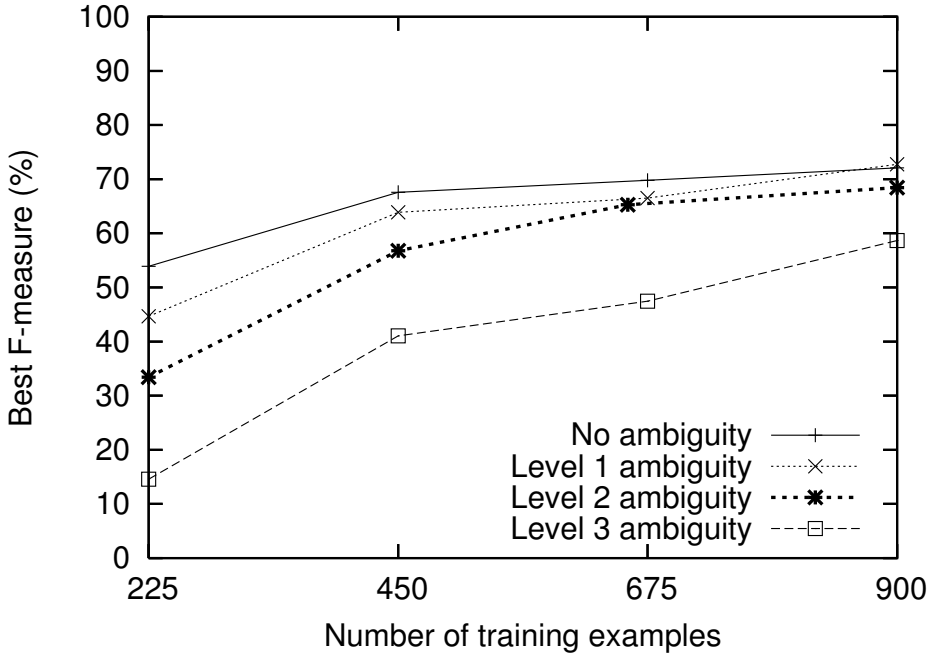


Figure 5.7: Learning curves for KRISPER on the AMBIG-GEOQUERY corpus with various levels of ambiguities.

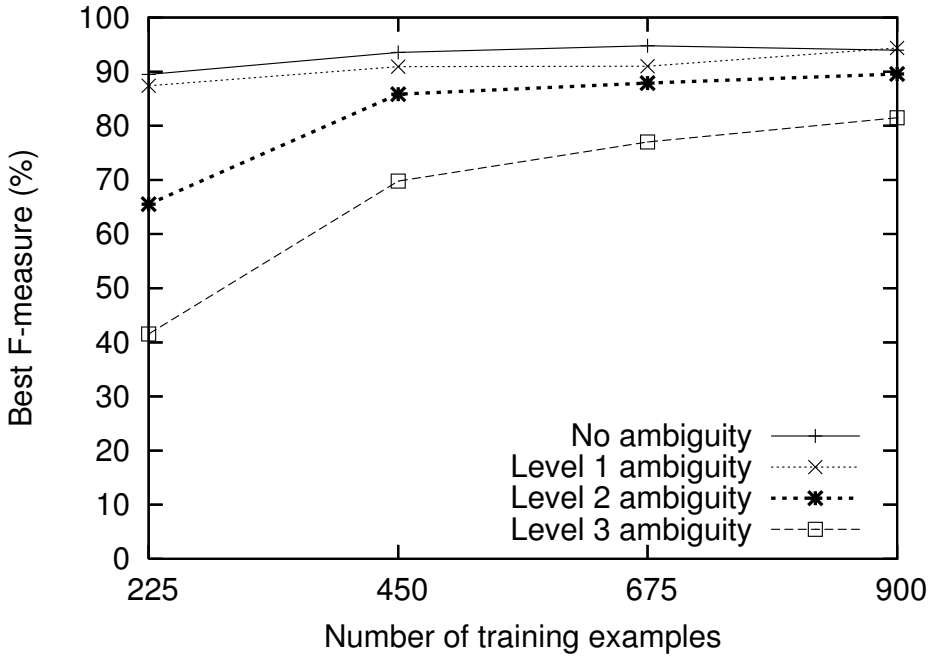


Figure 5.8: Learning curves for KRISPER on the AMBIG-CHILDWORLD corpus with various levels of ambiguities.

unambiguous training are also shown for comparison. As the training-set size is increased, KRISPER is able to overcome the ambiguities and learn almost as accurate a semantic parser as with no ambiguity on this corpus as well. Since weaker ambiguous supervision is cheaper to obtain than unambiguous supervision, it is reasonable to expect availability of higher amounts of ambiguous training data than unambiguous training data in practice.

Although artificially-constructed ambiguous training data was used in our experiments, the results indicate a promising potential for using KRISPER to learn language semantics from real-world ambiguous training data.

5.3 Chapter Summary

In this chapter we described two weaker forms of supervision for semantic parsing which are easier to obtain than full supervision. We extended our semantic parsing learning algorithm to utilize these forms of supervision. Experimentally, we showed that the performance of a semantic parser can be improved by utilizing unlabeled data using semi-supervised learning. We also showed that accurate semantic parsers can be learned even when the given supervision is ambiguous.

Chapter 6

Transforming the Meaning Representation Grammar to Improve Semantic Parsing

6.1 Motivation

The framework for learning semantic parsers described in the previous chapters assumes that along with the training data, a context-free grammar (CFG) for parsing the meaning representations (MRs) is given, which we will call the meaning representation grammar (MRG). There can, however, be several different CFGs that accept the same meaning representation language (MRL). Given that KRISP uses the productions of the provided MRG as semantic concepts, and learns how well different natural language substrings probabilistically represent them, it is essential for learning a good semantic parser that the productions of the provided MRG are closely associated with the underlying semantics of the natural language.

However, an MRL and its MRG are typically designed to best suit the application in which the MRs will be used with little consideration for how well they correspond to the semantics of a natural language. This can result in an MRG whose productions do not correspond well with the semantics of the natural language.

In the experiments reported in the previous chapters, for KRISP as well as for the other systems which use MRGs, the MRGs for CLANG and GEOQUERY were first manually modified to make them more compatible with the natural language. In the original MRG for CLANG provided by the ROBOCUP community (Chen et al., 2003b), there were several constructs which did not correspond well with their meanings in the natural language. For example, the MR expression of the rectangle "(rec (pt -32 -35) (pt 0 35))", whose parse according to the original MRG is shown in Figure 6.1, represents "our midfield". As can be seen, the numbers as well as the productions for the POINTs in the MR expression do not correspond to anything in its natural language utterance. It is also impossible to derive a semantic derivation of the MR expression over this natural language utterance because there are not enough words in it to cover all the productions present in the MR parse at the lowest level. A new MRG was manually created to make it correspond better with the natural language by replacing such long MR expressions for regions by shorter expressions like "(midfield our)". Note that the names for the new tokens introduced were chosen for readability and their similarity to the natural language words is inconsequential for learning semantic parsers. This new MRG was used in all the previous work which uses the CLANG corpus.

The MRL for GEOQUERY, which is a variable-free functional query language, was constructed from the original Prolog expressions of its MRs. From this MRL, the MRG was then manually designed so that its productions were compatible with the semantics of the natural language. This MRG was

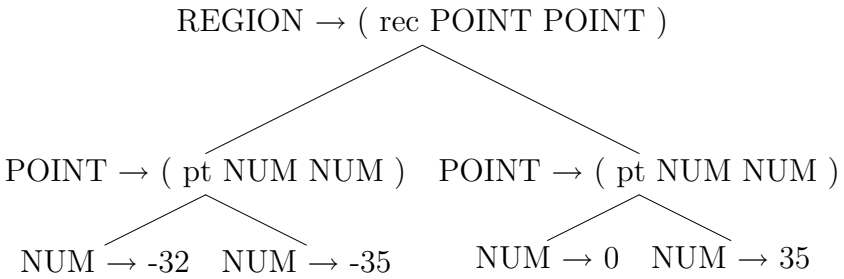


Figure 6.1: The parse for the CLang expression "(rec (pt -32 -35) (pt 0 35))" corresponding to the natural language utterance "our midfield" using its original MRG.

different from some simple MRG one would otherwise design for the MRL. Since the system Wasp (Wong & Mooney, 2006) does not use productions as semantic concepts the way Krisp does, a different MRG was used for its experiments that was more suitable for its learning algorithm. We found that Krisp does not learn a good semantic parser if it is given a simple MRG or the MRG used by Wasp. Figure 6.2 (a) shows the parse tree obtained using a simple MRG for the MR "answer(longest(river(loc_2(stateid('Texas')))))" corresponding to the natural language sentence "Which is the longest river in Texas?". The MR parse obtained using the simple MRG is more like a linear chain which means that each production will have to cover the entire sentence in a correct semantic derivation. But ideally, different productions should correspond to the meanings of different substrings of the sentence. Figure 6.2 (b) shows a parse tree obtained using the manually designed MRG in which the productions "$QUALIFIER \rightarrow longest$" and "$LOC\_2 \rightarrow loc\_2$" would cover the "longest" and "in" substrings of the NL sentence with possibly other words in a good semantic derivation. Note that another simple MRG in which

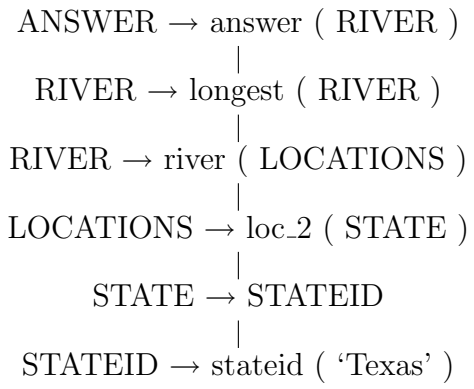
a separate production is added for every terminal, like "$LOC\_2 \rightarrow loc\_2$" may not always be good because if there are some superfluous terminals in the MRL then it will force the semantic parser to learn their meanings as well.

Manually engineering an MRG to work well for semantic parsing is a tedious task and requires considerable domain expertise in the meaning representation language. In this chapter, we present approaches to automatically transform a given MRG to make it more suitable for learning semantic parsers. We first describe four operators which transform an MRG into another MRG that accepts the same MRL. We next describe how these operators are applied to transform the given MRG in an error-driven manner using KRISP's semantic parsing learning algorithm. We also describe another way of transforming an MRG through *meaning representation macros*. These are able to directly introduce the transformations needed to eliminate the type of incompatibilities in the MRGs shown in Figure 6.1. We present experimental results to show how these grammar transformations improve the performance on semantic parsing.

6.2 Using Operators to Transform a Meaning Representation Grammar

The meaning representation languages used for semantic parsing are always assumed to be context-free. There has been some work in learning context-free grammars (CFGs) for a language given several examples of its expressions (Lee, 1996). Most of the approaches directly learn a grammar

(a)



(b)

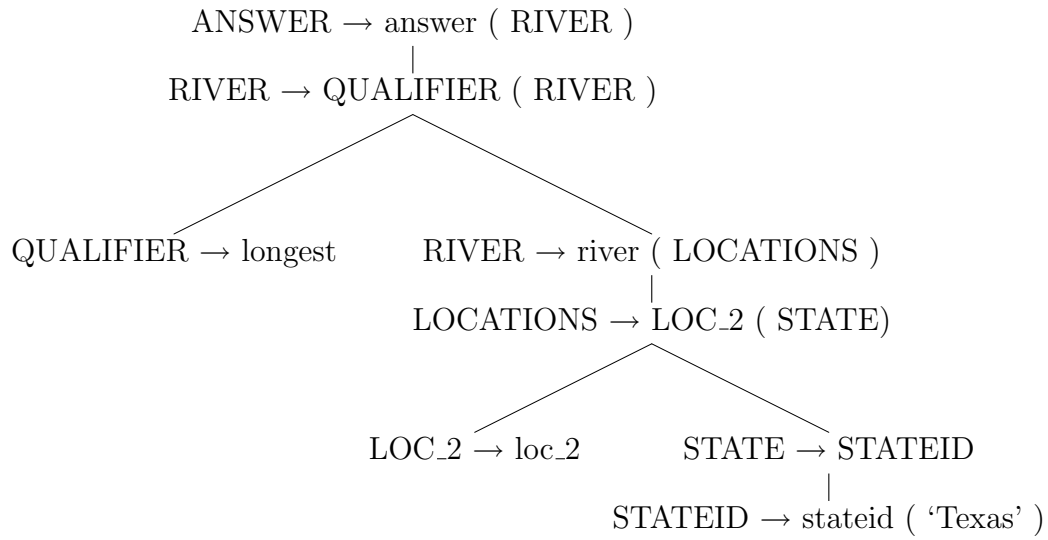


Figure 6.2: Different parse trees obtained for the MR "answer(longest(river(loc_2(stateid('Texas')))))" corresponding to the NL sentence "Which is the longest river in Texas?" using (a) a simple MRG (b) a manually designed MRG.

from the expressions but there also have been approaches that first start with a naive grammar and then transform it using suitable operators to a better grammar (Langley & Stromsten, 2000). The goodness for a grammar is typically measured in terms of its simplicity and coverage. For the task of learning semantic parsers, since an initial MRG is always given, there is no need to first learn it from its MRs. The initial MRG is transformed to a better MRG using the operators described in the next subsection. Our criteria for goodness of an MRG is the performance of the semantic parser learned using the grammar. Subsection 6.2.2 describes how and when the operators are applied to transform the MRG.

6.2.1 Transformation Operators

We describe four CFG transformation operators which are used to transform an MRG. Each of these operators preserves the coverage of the grammar, i.e. after application of the operator, the transformed grammar accepts the same language that the previous grammar accepted.

1. **Create Non-terminal from a Terminal (CreateNT):** Given a terminal symbol t in the grammar, this operator adds a new production $T \rightarrow t$ to it and replaces all the occurrences of the terminal t in all the other productions by the new non-terminal T.

   In the context of KRISP's semantic parsing learning algorithm, this operator introduces a new semantic concept the previous grammar was not explicit about. For example, this operator may introduce a production

(a semantic concept) $LONGEST \rightarrow longest$ to the simple grammar whose parse was shown in Figure 6.2 (a). This is close to the production $QUALIFIER \rightarrow longest$ of the manual grammar used in the parse shown in Figure 6.2 (b).

2. **Merge Non-terminals (MergeNT):** This operator merges n non-terminals T_1, T_2, ...T_n, by introducing n productions $T \rightarrow T_1$, $T \rightarrow T_2$, ...$T \rightarrow T_n$ where T is a new non-terminal. All the occurrences of the merged non-terminals on the right-hand-side (RHS) of all the remaining productions are then replaced by the non-terminal T. If this leads to multiple copies of some production then only one copy is kept. In order to ensure that this operator preserves the coverage of the grammar, before applying it, it is made sure that if one of these non-terminals, say T_1, occurs on the RHS of a production π_1 then there also exist productions π_2, ...π_n which are same as π_1 except that T_2, ...T_n respectively occur in them in place of T_1. If this condition is violated for any production of any of the n non-terminals then this operator is not applicable.

   This operator enables generalization of some non-terminals which occur in similar contexts leading to generalization of productions in which they occur on the RHS. For example, this operator may generalize non-terminals $LONGEST$ and $SHORTEST$ in GEOQUERY MRG to form $QUALIFIER \rightarrow LONGEST$ and $QUALIFIER \rightarrow SHORTEST$ productions. This can lead to generalization of productions $RIVER \rightarrow LONGEST\ RIVER$ and $RIVER \rightarrow SHORTEST\ RIVER$ into a

single production $RIVER \rightarrow QUALIFIER\ RIVER$. This would then increase the number of positive examples that can be collected from the training data for this production, all of them representing some superlative quality of rivers. The concepts of "longest" and "shortest" will be learned separately by the productions $LONGEST \rightarrow longest$ and $SHORTEST \rightarrow shortest$ respectively.

3. **Combine Two Non-terminals (CombineNT):** This operator combines two non-terminals T_1 and T_2 into one new non-terminal T by introducing a new production $T \rightarrow T_1\ T_2$. All the instances of T_1 and T_2 occurring adjacent in this order on the RHS (with at least one more non-terminal) of all the other productions are replaced by the new non-terminal T. For example, the production $A \rightarrow a\ B\ T_1\ T_2$ will be changed to $A \rightarrow a\ B\ T$. Without the presence of an extra non-terminal on the RHS, this change will merely add redundancy to the parse trees which would use this production. This operator will not eliminate other occurrences of $T1$ and $T2$ on the RHS of other productions in which they do not occur adjacent to each other. In the context of semantic parsing, this operator only adds an extra level in the MR parses which does not seem useful in itself, but later if the non-terminals T_1 and T_2 get eliminated (by the application of the operator described next), this operator will be combining the concepts represented by the two non-terminals.

4. **Delete Production (DeleteProd):** This last operator deletes a production and replaces the occurrences of its left-hand-side (LHS) non-

terminal with its RHS in the RHS of all the other productions. For example, if the production $T \rightarrow a\ B\ C$ is deleted by this operator then a production $A \rightarrow T\ c$ will be changed to $A \rightarrow a\ B\ C\ c$. But if there is another production (say, $T \rightarrow t$) with the same LHS as the production being deleted (T), then the original copies of the altered productions are also kept. In this example, the original production $A \rightarrow T\ c$ will be kept in addition to the changed production $A \rightarrow a\ B\ C\ c$. This is necessary because the grammar has an alternate way of generating expressions from T which could be used in the original production $A \rightarrow T\ c$. This operator is not applicable if the LHS non-terminal is the start symbol of the grammar or if its LHS non-terminal also occurs on its RHS (i.e. it is a recursive production).

In terms of semantic parsing, this operator eliminates the need to learn a semantic concept. It can undo the transformations obtained by the previous operators by deleting the new productions they introduce.

Instead of creating an operator to delete a non-terminal, we chose this operator which deletes a production because deleting a non-terminal can drastically change the grammar which may reduce its chances of finding a good MRG for semantic parsing. A non-terminal can, however, get deleted from the grammar if this operator is applied to all the productions in which that non-terminal occurs on the LHS.

We note that the CombineNT and MergeNT operators are same as the two operators used by Langley and Stromsten (2000) to search a good CFG

for natural language sentences from the space of its possible CFGs. We also note that the applications of CreateNT and CombineNT operators can reduce a CFG to its Chomsky normal form (in which all the productions are of the form $A \rightarrow a$ and $A \rightarrow B\ C$), and conversely, because of the reverse transformations achieved by the DeleteProd operator, a Chomsky normal form of a CFG can be converted into any other CFG which accepts the same language.

6.2.2 Applying Transformation Operators

In order to transform an MRG to improve semantic parsing, a simple hill-climbing approach will be to search the space of all possible MRGs by applying one of the applicable transformation operators at a time and measuring the performance on semantic parsing task. If the performance does not improve then undo that transformation and continue searching. This search method will be computationally very intensive because it involves re-training the semantic parser for each application of an operator. We next describe a faster but a less thorough heuristic search method which typically applies several operators before re-training the semantic parser to evaluate its performance.

First, using the provided MRG and the training data, a semantic parser is trained using KRISP. The trained semantic parser is applied to each of the training sentences. Next, for each production π, two values $total_\pi$ and $incorrect_\pi$ are computed. The value $total_\pi$ counts how many training examples use production π in their MR parses. The value $incorrect_\pi$ counts the

number of training examples for which the semantic parser incorrectly uses the production π during parsing, i.e. it either did not include the production π in the parse of the MR it produces when the correct MR's parse included it, or it included the production π when it was not present in the correct MR's parse. These two statistics for a production indicate how well the semantic parser was able to use the production in semantic parsing. If it was not able to use a production π well, then the ratio $incorrect_\pi/total_\pi$, which we call $mistakeRatio_\pi$, will be high indicating that some change needs to be made to that production.

After computing these values for all the productions, any of the four procedures described below corresponding to the four operators can be invoked to transform the MRG. Figure 6.3 shows how these procedures are invoked in our MRG transformation algorithm.

1. **Apply CreateNT:** For each terminal t in the grammar, $total_t$ and $incorrect_t$ values are computed by summing up the corresponding values for all the productions in which t occurs on the RHS with at least one non-terminal (without a non-terminal on the RHS, the operator will only add a redundant level to the parses using this production). If $total_t$ is greater than β (a parameter) and $mistakeRatio_t = incorrect_t/total_t$ is greater than α (another parameter), then the CreateNT operator is applied, provided the production $T \rightarrow t$ is not already present. A high $mistakeRatio_t$ indicates that the semantic parser incorrectly used most

of the productions in which t occurs on the RHS, perhaps because an additional semantic concept was being learned under each of those productions, and it is hoped that this will be corrected by introducing a new semantic concept in the form of the production $T \rightarrow t$ which will be learned separately.

2. **Apply MergeNT:** All the non-terminals occurring on the RHS of all those productions π are collected whose $mistakeRatio_{\pi}$ value is greater than α and whose $total_{\pi}$ value is greater than β. These non-terminals are then partitioned such that the criteria for applying the MergeNT is satisfied by the non-terminals in each partition with size at least two. The MergeNT operator is then applied to the non-terminals in each partition with size at least two. Merging the non-terminals will generalize the productions in which the non-terminals were separately occurring, and hopefully the semantic parser will then learn better to use these generalized productions correctly.

3. **Apply CombineNT:** For every non-terminal pair T_1T_2, $total_{T_1T_2}$ and $incorrect_{T_1T_2}$ values are computed by summing their values for the productions in which the two non-terminals are adjacent in the RHS in the presence of at least one more non-terminal. If $mistakeRatio_{T_1T_2} = incorect_{T_1T_2}/total_{T_1T_2}$ is greater than α and $total_{T_1T_2}$ is greater than β, then the CombineNT operator is applied to these two non-terminals. The order in which instances of this operator is applied is the order

in which the pairs of the non-terminals were collected. If the productions in which the two non-terminals alway occur adjacent are mostly incorrectly used, then this could be an indication that the two concepts represented by the two non-terminals are the same and should not be learned separately.

4. **Apply DeleteProd:** The DeleteProd operator is applied to all the productions π where it is applicable and whose $mistakeRatio_\pi$ is greater than α and $total_\pi$ is greater than β. The order in which they are applied is the order in which they are in the grammar. This step simply deletes the productions which are mostly incorrectly used.

Figure 6.3 shows the complete algorithm we used for applying the operators. The procedure for applying one of the transformation operators is called after collecting the *total* and *incorrect* values of all the productions. After the grammar is transformed, the MRs in the training data are re-parsed and the semantic parser is re-trained. Next, the procedure for applying the next transformation operator is called. The whole process repeats for a specified number of iterations. If the transformations done by the first three operators did not help, then they could be undone by the DeleteProd operator. This order was decided to allow merging and combining of the of the new non-terminals which may get created, and to allow deletion of the newly created productions which were not found helpful. However, since the the process proceeds in iterations, the operators are applied in a cyclic fashion and hence their order is not very

Train semantic parser on training examples
Repeat for MAX_ITERATIONS
    Collect *total* and *incorrect* values for all productions
    Apply CreateNT
    Retrain using the transformed grammar

    Collect *total* and *incorrect* values for all productions
    Apply MergeNT
    Retrain using the transformed grammar

    Collect *total* and *incorrect* values for all productions
    Apply CombineNT
    Retrain using the transformed grammar

    Collect *total* and *incorrect* values for all productions
    Apply DeleteProd
    Retrain using the transformed grammar

Return the semantic parser trained with the final grammar

Figure 6.3: The MRG transformation algorithm to improve performance on semantic parsing.

critical. In the experiments, we found that the performance does not improve much after two iterations. For the experiments, we fixed the α parameter to be 0.75 and β parameter to be 5.

6.2.3 Experiments

We tested our MRG transformation approach to improving semantic parsing on the GEO880 corpus by giving it the simple MRG and the MRG used in the WASP system mentioned in the Section 6.1, and by comparing its performance with the semantic parser learned using the manually-built MRG.

Figure 6.4 shows that the performance of the semantic parser that KRISP learns when trained on the simple MRG is not good. But when that MRG is transformed using the approach described in the last section, the performance of the semantic parser dramatically improves and is very close to the performance obtained by the manually-engineered grammar. A similar trend was observed when using WASP's MRG, as shown in Figure 6.5. In this case, the final performance is only slightly worse than the performance obtained using the manual grammar.

We found that most of the the performance gain is obtained because of the CreateNT and DeleteProd operators. The MergeNT operator was only occasionally used and the CombineNT operator was never used. One reason for this is that the conditions for applying these operators are very restrictive, for example, very few non-terminals in the MRGs we considered met the conditions for applying the MergeNT operator, and there were very few productions

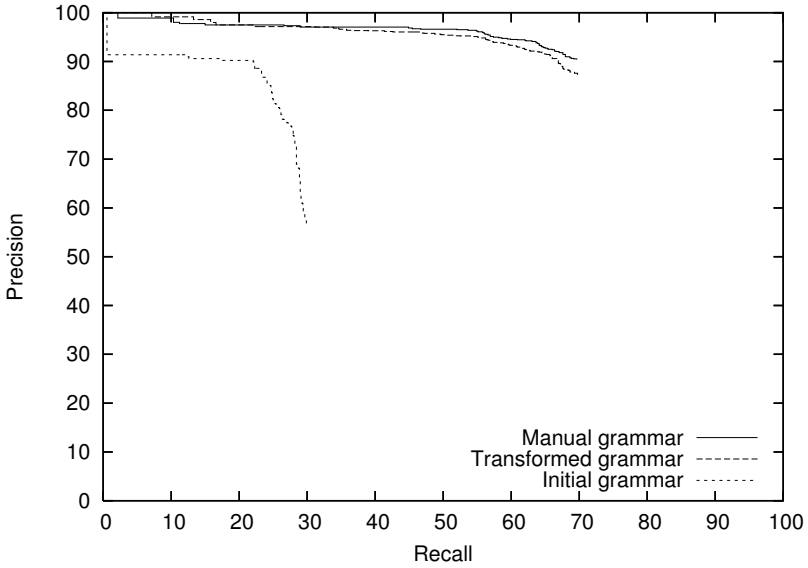


Figure 6.4: The results comparing the performances of the learned semantic parsers on the GEO880 corpus using different grammars: a simple initial MRG, the MRG obtained after transforming it and and the manually designed grammar.

with more than two non-terminals on the RHS (a condition for applying the CombineNT operator). However, we believe that for some different types of grammars, these operators could be more helpful. When the MRG transformation algorithm was applied to the manually-engineered grammar, we found that there was not much change in the performance.

6.3 Using Meaning Representation Macros

As described in Section 6.1, sometimes there can be large parses for MR expressions which do not correspond well with their semantics in the natural language. Figure 6.1 showed an example for this. While it is possible

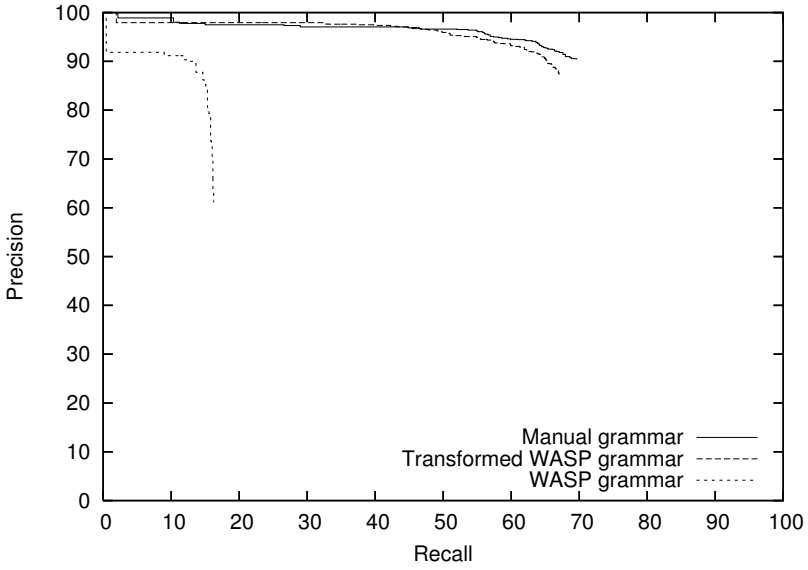


Figure 6.5: The results comparing the performance of learned semantic parsers on the GEO880 corpus using different grammars: the MRG used in WASP, the MRG obtained after transforming it and the manually designed grammar.

to transform the MRG using the operators described in the previous section to reduce the whole parse to just one production which will then correspond directly to its meaning in the natural language, it will require a particular sequence of transformation operators to achieve this which may rarely happen during the heuristic search used in the MRL transformation algorithm. In this section, we describe a more direct way of obtaining such transformations using MR macros.

6.3.1 Meaning Representation Macros

A meaning representation macro for an MRG is a production with only terminals on its RHS which form a legal expression in the MRL and which is derivable from its LHS non-terminal. For example, for the CLang example shown in Figure 6.1, the production $REGION \rightarrow (rec(pt\ -32\ -35)(pt\ 0\ 35))$ is a meaning representation macro. From an MR parse drawn with non-terminals at the internal nodes (instead of productions) and terminals at the leaves, a macro can be derived from a subtree rooted at any of the internal nodes by making its root non-terminal as the LHS non-terminal and the left-to-right sequence formed by the terminal leaves as the RHS. We use the following error-driven procedure to introduce macros in the MRG in order to improve the performance of semantic parsing.

6.3.2 Learning Meaning Representation Macros

A semantic parser is first learned from the training data using KRISP and the given MRG. The learned semantic parser is then applied to the training sentences and if the system can not produce any parse for a sentence then the parse tree of its corresponding MR is included in a set called *failed parse trees*. Common subtrees in these failed parse trees are likely to be good candidates for introducing macros. A set of *candidate trees* is created as follows. This set is initialized to the set of failed parse trees. The largest common subtree[1] of every pair of trees in the candidate trees is then also included in the set if it is not an empty tree. The process continues with the newly added trees until no new tree can be included. This process is similar to the repeated bottom-up generalization of clauses in the inductive logic programming system GOLEM (Muggleton & Feng, 1992). Next, the trees in this set are sorted based on the number of failed parse trees of which they are a subtree. The trees which are part of fewer than β subtrees are removed. Then in highest to lowest order, the trees are selected one-by-one to form macros, provided their height is greater than two (otherwise it will be an already existing MRG production) and an already selected tree is not its subtree. These are the trees which are present frequently as subtrees in the failed parse trees, which is an indication that they do not correspond well with the natural language and hence introducing macros for them may help. A macro is formed from a tree by making the non-terminal root of the tree as its LHS non-terminal and the left-to-right

[1] All the leaves of a subtree must be terminals.

sequence of terminals at the leaves as its RHS. An example macro for CLANG could be $REGION \rightarrow (rec(pt\ -32\ -35)(pt\ 0\ 35))$ formed from the tree shown in Figure 6.1.

These newly formed macros (productions) are then included in the MRG. Although this makes the grammar ambiguous because there are now two ways to parse the MR expressions from which the macros were derived (using the macro productions and using the earlier parses), but by inserting the macros at the front of the list of MRG productions, the parsing is made unambiguous because the MRL parser we use prefers earlier productions to resolve ambiguities during parsing. Alternatively, one can introduce a new terminal for the entire RHS (similar to new terminals like "midfield" that were introduced by the manual CLANG grammar). This will involve changing the MR expressions as well.

The MRs in the training data are re-parsed and the semantic parser is re-trained using the new MRG. In order to delete the macros which were not found useful, a procedure similar to the application of DeleteProd is used. The *total* and *incorrect* values for all the macros are computed as in the Subsection 6.2.2. The macros for which $mistakeRatio = total/incorrect$ is found greater than α and *total* is greater than β are removed. No further change needs to be done to the grammar since there already is an alternate way to parse each of the expressions for which the macros were used.

This whole procedure is repeated a specified number of iterations. In order to prevent re-introducing the deleted previously deleted, the list of deleted

Train semantic parser on training examples
Repeat for MAX_ITERATIONS
    Collect *failed parse trees* from the training examples
    Form a set of *candidate trees* and expand it by repeated generalization
    Form macros from the best candidate trees
    Retrain using the transformed grammar

    Collect *total* and *incorrect* values for all productions
    Delete unsuccessful macros
    Retrain using the transformed grammar

Return the semantic parser trained with the final grammar

Figure 6.6: The macro transformation algorithm to improve performance of semantic parsing.

macros is maintained across iterations and a new macro is not introduced if it is present in this list. Figure 6.6 shows the complete macro transformation algorithm for introducing and removing macros. We found that two iterations are usually sufficient. In the experiments, the value for α was 0.75 and for β was 5.

6.3.3 Experiments

We tested our macro transformation algorithm on the CLang corpus using the original MRG in which all the chief regions of the soccer field were in the form of numeric MR expressions which do not correspond to their meanings in the natural language. We compared the performance of the semantic parser trained with Krisp using the original MRG, trained with the new MRG obtained from the original MRG using the macro transformation algorithm de-

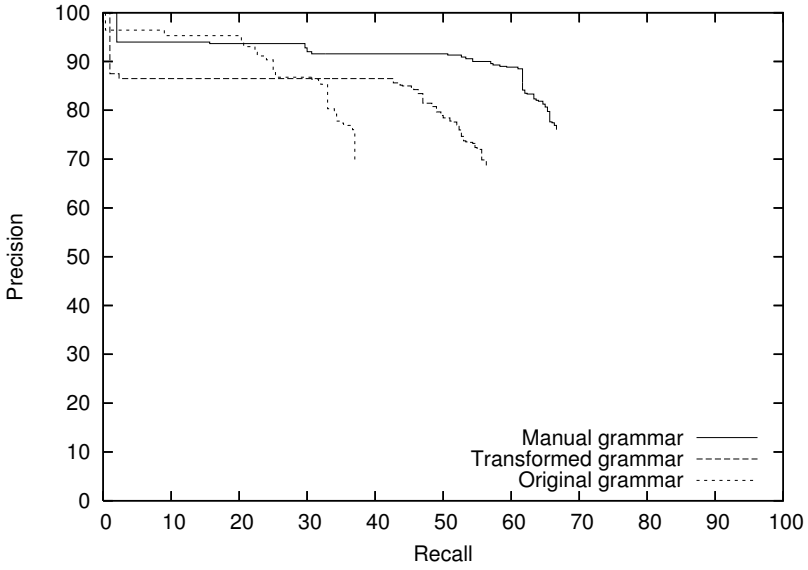


Figure 6.7: The results comparing the performances of the learned semantic parsers on the CLANG corpus using different grammars: the original CLANG MRG, the MRG obtained after applying macro transformations and the manually designed MRG.

scribed in the previous subsection, and trained using the manually-built MRG.

Figure 6.7 shows the results. After applying the macro transformations the performance improved by a large margin. Although the precision was lower for low recall values, the recall increased by a large quantity and the best F-measure improved from 50% to 63%. But the performance still lagged behind that obtained using the manually-engineered MRG. The most likely reason for this is that the manual MRG introduced some domain specific expressions, like left, right, left-quarter etc., which correspond directly to their meanings in the natural language. For example, the expression "(left (midfield our))" corresponds to the utterance "left of our midfield", and because the expression

"(midfield our)" corresponds to the utterance "our midfield", the concept of "left" can easily be learned. On the other hand, the only way to specify "left" of a region using the original CLang MRG is by specifying the coordinates of the left region, like "(rec(pt -32 -35)(pt 0 0))" is the left of "(rec (pt -32 -35) (pt 0 35))". It is not easy to learn the concept of "left" from such expressions even with the use of macros.

When we applied transformation operators on the original CLang MRG and macro transformations on the Geoquery MRGs used in Section 6.2.3, it did not lead to any performance gain. This shows that the productions in the two types of grammars do not conform with natural language semantics in different ways.

6.4 Chapter Summary

An MRG whose productions do not correspond well with the natural language can lead to a bad performance by a semantic parser that uses the MRG productions. But by using the transformation operators and macros to transform the MRG to better suit the natural language, the performance on semantic parsing task can be improved, often close to the performance obtained using manually engineered grammars.

Chapter 7

Ensembles of Semantic Parsers

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). In this chapter, we consider forming ensembles of semantic parsers. We combine KRISP with two other competitive semantic parser learning systems developed in our research group: WASP (Wong & Mooney, 2006) and SCISSOR (Ge & Mooney, 2005), and show that the resulting ensemble achieves the best overall performance.

Previously, Nguyen et al. (2006) have used ensembles of semantic parsers. Their semantic parsers are learned using the same learning algorithm from different training data employing Bagging (Breiman, 1996) and Boosting (Freund & Schapire, 1996) techniques. Their system then selects one final output from the outputs of their semantic parsers using a classifier which when given an output, classifies whether it is correct or not. This classifier is trained using boosting with subtrees as weak decision stump functions (Kudo & Matsumoto, 2004). Such a classifier works in their system because just like their training data, the outputs of their semantic parsers are *semantically augmented parse*

trees (SAPTs; Ge & Mooney, 2005), in which parts of the meaning representation are tied with parts of the natural language sentence, and hence a classifier can be trained using the training data to classify the output as correct or incorrect. But for two of our systems, KRISP and WASP, the training data and the outputs are not in the form of SAPTs and hence we can not use such a classifier.

In the following section, we describe two simple methods we used to combine the outputs of our semantic parsers.

7.1 Combining Outputs of Semantic Parsers

We first considered the following *simple majority* algorithm to combine the outputs of the three semantic parsers - KRISP, WASP and SCISSOR. If all the three systems output the same meaning representation (MR) for a natural language (NL) sentence then that is also the output of the ensemble. If however, two of the systems output an MR which is different from the third system's output MR, then the output of those two systems is the output of the ensemble. If all the three systems output different MRs or one of them fails to produce an output, then the tie is broken according to a pre-specified order among the three semantic parsers. In these cases, the output of the first system (usually the best system according to internal cross-validation) is the output of the ensemble, but if this system failed to output any MR then the output of the second system is the output of the ensemble. Finally, if both these systems fail to output an MR then the output of the third system will

be the output of the ensemble.

Another way of combining the outputs of semantic parsers is to combine components of their outputs. This is analogous 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 and exploit the fact that the leaves of all the output syntactic parse trees are same, namely, the words of the sentence. This is not true for MR parse trees.

We used the following *subtree majority* algorithm to combine components of the outputs of the three semantic parsers. The algorithm can be easily generalized if there are more than three semantic parsers. The output MRs are first parsed using the MRL grammar and their MR parse trees are obtained. The algorithm constructs the final MR parse tree by combining the common subtrees which are in majority in these MR parse trees. The intuition is that while a single semantic parser may not have gotten all the subtrees in its MR parse right, but if majority of the semantic parsers' output MR parses share some subtrees, then they are more likely to be correct and should be included in the final output.

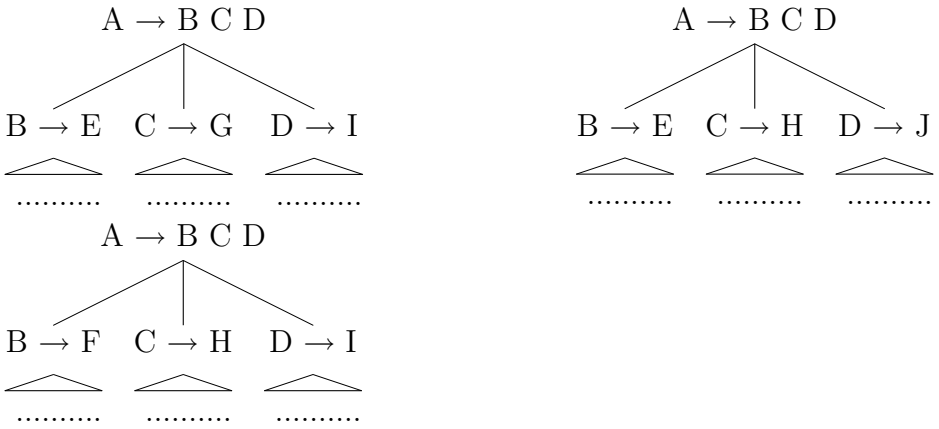
The subtree majority algorithm proceeds as follows. First, if the trees don't have the same production at the root, then the output parse tree of the ensemble is obtained by the simple majority algorithm described earlier. This is done, because if the root production is different then there is no elegant way

to combine the subtrees underneath the three roots.[1] But if the trees have the same production at the roots, then the output tree for the ensemble is created in the following manner. The common production is first included as its root. Next, for each right-hand-side (RHS) non-terminal in this production, a subtree is constructed by invoking the subtree majority algorithm recursively and is added as a child to the root. Figure 7.1 shows an illustration of this algorithm. Since the roots of the three output trees have the same production ($A \rightarrow B\ C\ D$), this production is included at the root of the ensemble's output tree. This algorithm is then recursively called for constructing subtrees under each of the non-terminals B, C and D. Since the productions under B non-terminal are different in the three trees, the simple majority algorithm is called on all the subtrees rooted under B. This then selects the tree which has $B \rightarrow E$ as the root production (assuming the triangles underneath it are same). Similarly subtrees with $C \rightarrow H$ and $D \rightarrow I$ as the root productions will be included under the C and D non-terminals respectively to complete the ensemble output tree. Note that the resulting tree is different from any of the initial trees.

A different approach to forming ensembles of semantic parsers is to first learn which parser performs best on which types of sentences and then simply choose the output MR for a test sentence from the best parser for that

[1] If more than three trees were to be combined, then at this step it will be reasonable to include the root production which is in majority, but doing this with only three trees will degenerate the problem to finding ensemble of the two trees which have the common production.

(a) Output trees of the semantic parsers:



(b) Output tree of the ensemble:

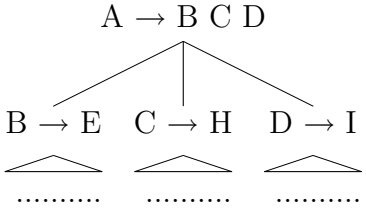


Figure 7.1: An example of the subtree majority algorithm to combine trees. (a) Output MR parse trees obtained from the given semantic parsers. (b) Output MR parse tree of the ensemble obtained using the subtree majority algorithm.

type of sentence. This is similar to the "Parser Switching" approach presented by Henderson and Brill (1999) for combining syntactic parses from different parsers. We, however, did not try this approach. Another direction for future work could be to combine partial outputs from the semantic parsers. If the semantic parsers fail to generate complete MRs but can generate partial MRs, then it should be possible in some cases to form a complete MR out of these partial MRs.

7.2 Experiments

We formed ensembles of the three semantic parser learning systems, KRISP, WASP and SCISSOR, using simple majority and subtree majority algorithms and compared their performance against the individual systems on the GEO880 and CLANG corpora. To break the ties in the simple majority algorithm, we pre-specified the order of systems as SCISSOR followed by KRISP followed by WASP. We kept SCISSOR in front because it generally obtains the best performance. The order of KRISP and WASP was chosen arbitrarily and was not found to make a big difference in performance. We found that our two ensemble algorithms gave exactly the same output on both the corpora. This implies that there was no example for which combining the majority subtrees generated a different tree from the one obtained by just choosing the majority tree. But in general, they can be different.

Tables 7.1 and 7.2 show the results obtained using the individual semantic parsers and their ensemble (same result by either method) on the

	Ensemble	KRISP	SCISSOR	WASP		Oracle
Precision(%)	87.82	90.33	92.98	87.67		100.00
Recall(%)	84.20	71.47	73.41	75.80		88.52
F-measure(%)	**85.96**	79.72	81.95	81.27		93.91

Table 7.1: Results obtained on the GEO880 corpus by the individual semantic parser learning systems and their ensemble using simple majority. The maximum performance achievable by an oracle ensemble is also shown.

GEO880 and CLANG corpora respectively. The shown precisions, recalls and F-measures were obtained by averaging them across the folds of 10-fold cross-validation. Since the confidence values for the output MRs from different semantic parsers are not calibrated, confidence values were not assigned to the outputs of the ensemble. Hence, we could not plot PR curves or determine the best F-measures.

As can be seen from Table 7.1, on the GEO880 corpus, the ensemble obtains a much higher recall than the individual systems with some drop in precision. Its F-measure was found to be statistically significantly better (with $p < 0.01$ based on paired t-test) than the F-measures obtained by each of the individual systems. In the last column, we also show the maximum performance an oracle ensemble can obtain which always chooses the correct output if at least one is correct. The oracle ensemble achieves around 94% F-measure while the current ensemble achieves around 86% which shows that the performance of the ensemble could be further improved.

The results obtained on the CLANG corpus are shown in Table 7.2. On this corpus, the ensemble obtains higher recall than the individual systems, but

	Ensemble	KRISP	SCISSOR	WASP		Oracle
Precision(%)	83.21	76.11	88.13	87.26		100.00
Recall(%)	80.67	66.67	75.00	61.00		85.00
F-measure(%)	**81.91**	70.99	**80.93**	71.47		91.89

Table 7.2: Results obtained on the CLANG corpus by the individual semantic parser learning systems and their ensemble using simple majority. The maximum performance achievable by an oracle ensemble is also shown.

	All MRs same	Only two MRs same	No two MRs same		
Examples	366	242	KRISP 133	SCISSOR 120	WASP 169
Precision	99.18%	92.98%	63.91%	73.33%	59.76%

Table 7.3: Analysis of the outputs obtained from the three semantic parser learning systems on the GEO880.

its F-measure increases only marginally over SCISSOR's. This improvement was not found to be statistically significant ($p > 0.05$). Although the F-measures obtained by the ensemble and SCISSOR were statistically higher (with $p < 0.05$) than those obtained by KRISP and WASP. The oracle ensemble obtains an F-measure close to 92% while the current ensemble obtains an F-measure close to 82% showing that there is still sufficient scope to improve the ensemble.

We did some analysis of the outputs given by the three semantic parsers to get some insight into the simple majority ensemble. Tables 7.3 and 7.4 summarize this analysis for the GEO880 and CLANG corpora respectively. The first column in both the tables shows for how many examples all the three systems output the same MR. For GEO880, out of total 880 examples,

	All MRs same	Only two MRs same	No two MRs same		
Examples	142	75	KRISP 59	SCISSOR 44	WASP 19
Precision	98.59%	86.67%	24.42%	59.09%	15.79%

Table 7.4: Analysis of the outputs obtained from the three semantic parser learning systems on the CLANG corpus which has total 300 examples.

for 366 examples all three systems output the same MR, and for CLANG, from total 300 examples, for 142 examples they output the same MR. For both the corpora, we found that when all the three systems output the same MR, the precision is very close to 100%. This shows that from these three individual systems, a very accurate ensemble can be constructed which outputs an MR only when all three system output the same MR. However, the recall of this ensemble will be very low, about 42% for GEO880 and about 47% for CLANG. This also shows that when the semantic parsers generate incorrect outputs, they almost never all generate the same incorrect output, indicating that they make different types of errors.

The second columns of Tables 7.3 and 7.4 show for how many examples exactly two systems output the same MR which is different from what the third system outputs (which could be empty if it fails to generate an MR). On the GEO880 corpus, this happens with 242 examples, and giving the majority MR as output obtains close to 93% precision. And on the CLANG corpus, this happens with 75 examples and achieves a precision near 87%. This shows that occasionally two of the semantic parsers make similar errors and generate the

same incorrect output.

Finally, the third columns show how the individual parsers perform when no two of them output the same MR (i.e. when the conditions of the first two columns are not met). On the GEO880 corpus, out of the remaining 272 examples, KRISP generates outputs for 133 examples and fails to generate outputs for the rest of them. The corresponding numbers for SCISSOR and WASP are 120 and 169 respectively. The precisions obtained on these examples by each system are low. On the CLANG corpus, out of the remaining 83 examples, KRISP generates outputs for 59 examples, SCISSOR generates for 44 examples and WASP generates for 19 examples. The precisions of the systems are again very low on these examples, particularly for KRISP and WASP. This shows that when the systems completely disagree on an output, they are likely to be incorrect. This also indicates that there are some hard examples in the corpora for which all the systems have difficulty generating the correct MRs. For an ensemble, it is difficult to generate the correct outputs in these cases. A possible solution would be to learn to identify which semantic parser is likely to generate the correct MR for the given type of input sentence. For example, the analysis by Ge (2006) showed that for longer sentences (which are more common in CLANG), SCISSOR is generally more accurate.

In conclusion, we obtained the best overall performance by forming a simple majority ensemble from the three semantic parser learning systems. The analysis showed that when the semantic parsers agree on the output, the output is very likely to be correct, and when they disagree they are more

likely to be incorrect. We also noted that there is still scope to improve the performance that can be obtained by an ensemble.

Chapter 8

Directions for Future Work

This chapter describes some future research directions based on this work to improve semantic parsing and to broaden its scope.

8.1 Improving Krisp's Semantic Parsing Framework

There are a number of directions in which Krisp's semantic parsing framework can be improved. For computing the probability of a semantic derivation, Krisp makes an independence assumption that the probability of productions covering different substrings of the sentence are independent (Subsection 3.2.2). This is clearly a simplification, and using a better way to compute the probability of a semantic derivation without making this independence assumption could lead to improvement in performance. Particularly, the probabilities of parent productions should not be considered independent of the probabilities of their children productions in a semantic derivation, because parents always cover the substrings covered by the children. However, such an approach will also increase the complexity of the system resulting in slower parsing speed.

Krisp defines a semantic derivation in a restrictive way by not allowing

the words covered by sibling nodes to overlap. However, it is possible that some words may be relevant to two or more productions (semantic concepts) which are at sibling nodes in a semantic derivation. A more relaxed framework which allows for overlap between the covered words could be beneficial.

8.2 Better Kernels for Semantic Parsing

In this thesis, we used a string-subsequence kernel in our main semantic parser learning system. We also used a syntactic tree kernel in Chapter 4. It should be possible to further improve results by using even better kernels.

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, 2002a) 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 those by Zelenko, Aone, and Richardella (2003), Culotta and Sorensen (2004) and Bunescu and Mooney (2005a). Moreover, there also has been progress in learning dependency tree parsers (McDonald, Pereira, Ribarov, & Hajič, 2005b). Dependency trees capture important aspects of the functional relationships between words in a sentence, and using similarity between them as the kernel should help in semantic parsing.

Using word categories in a learning algorithm can often alleviate the data sparsity in training data arising due to infrequent use of words, and obtain better generalization performance. Kernels based on word categories have been shown to be beneficial in relation extraction task (Bunescu & Mooney, 2005b). Constructing kernels based on word categories from the WordNet (Fellbaum, 1998), or from statistically-determined word clusters (Baker & McCallum, 1998) or from the application domain specific word ontology should help improve the performance on the semantic parsing task.

It was noted in in Subsection 3.4.3 that in any semantic parsing application, the presence of noise is very likely. We also showed that our kernel-based approach to semantic parsing is robust to noise. It should be possible to make it more robust by designing special kernels which handle noise. For example, if a noise model is given or learned that indicates the likeliness of a word to be noise, then this information could be easily used to down-weight the contribution of these words in computing the kernel value. Designing kernels which are resistant to noise is a fruitful avenue for future research.

8.3 Extending to 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 classifiers for extracting the binary relation *"protein-protein interaction"*. This can be viewed as learning for an MRL grammar which has only one production: "INTERACTION →

PROTEIN PROTEIN". Hence we believe KRISP can be extended for use in relation extraction, particularly in complex n-ary relation extractions, which require a level of semantic analysis which is intermediate between the levels required for semantic parsing and for single entity information extraction.

A complex relation is defined as an n-ary relation among $n > 2$ typed entities (McDonald, Pereira, Kulick, Winters, Jin, & White, 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 which would be useful in applications like automatic database generation, intelligent document searching etc. To use KRISP in extracting complex relations, it will require building an appropriate meaning representation grammar, that treats 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 process of extracting the complex relation can be treated as semantic parsing. Figure 8.1 shows a possible semantic derivation of the example

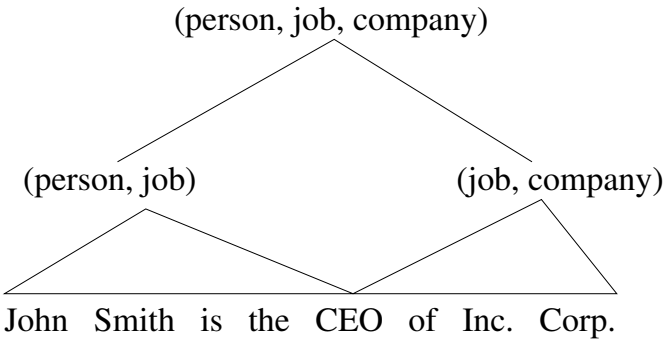


Figure 8.1: Relation `(person,job,company)` derived over the given sentence.

sentence from which the complex relation `(person, job, company)` could be extracted.

8.4 Learning from Perceptual Contexts

In Chapter 5, we presented an approach to learn semantic parsers from ambiguous supervision. Although our experimental results were on artificially-built corpora, the results showed a promising potential in using the approach to learn language semantics from real-world ambiguous data. One of the future directions is to build such corpora and test our approach on them. Our system directly uses symbolic meaning representations to represent perceptual contexts, a longer term future research direction would be to combine such a system with a vision-based system which can map real-world perceptual context into symbolic meaning representations. This could lead to a system which would learn language from perceptual contexts. There has been some work in this direction (Bailey, Feldman, Narayanan, & Lakoff, 1997; Roy, 2002; Yu

& Ballard, 2004), but the complexity of the natural language used has been limited.

8.5 Interactive Natural Language Communication with Computers

While semantic parsing can help the computer understand users' natural language utterances, for an interactive natural language communication with users, the computer should also be able to communicate back in natural language. For instance, if the computer fails to understand a natural language utterance, then instead of reporting a failure, it should be able to appropriately converse with the user in natural language for clarification. This then adds the interesting aspect of discourse processing (Lochbaum, Grosz, & Sidner, 1999; Ortiz & Grosz, 2002) to the problem. The computer should also be able to interpret new utterances in the context of the past communication. It will be interesting to explore how the current natural language generation and discourse processing techniques can be used in conjunction with semantic parsing to enable interactive communication.

8.6 Broadening Towards Open Domain

A longer term future direction would be to broaden the applicability of "narrow and deep" semantic parsing to open domain text. This will be helpful in several applications including question-answering systems, creating personalized software assistants, better natural language communication with

robots etc. While it is difficult to construct one global meaning representation language (MRL) for open domain text, but based on what actions the computer is expected to perform in response to the natural language input, one can narrow down or "ground" the meaning of natural language in terms of the computer's actions. Once this is done, a semantic parser can be trained to convert NL sentences into their meanings in this MRL. To successfully process open domain NL sentences, a semantic parser will need to employ the techniques developed from the "broad and shallow" direction like syntactic parsing, word sense disambiguation, anaphora resolution etc. to first reduce the ambiguities present in the input. This will also need training data in the form of open domain corpora annotated with appropriate semantics, like FrameNet (Fillmore, Baker, F., & Sato, 2002) and OntoNotes (Hovy, Marcus, Palmer, Pradhan, Ramshaw, & Weischedel, 2006).

Chapter 9

Conclusions

This thesis presented a new machine learning framework for semantic parsing. Using the productions of the meaning representation language as semantic concepts, it learns string classifiers for them which estimate the probability of a natural language substring to represent the semantic concepts. This information is used to parse the sentence into its meaning representation. The string classifiers are learned using sound statistical machine learning technique of Support-Vector machines (SVMs) using string subsequence kernels. This framework was experimentally shown to perform competitively with recent statistical learning methods. Unlike other methods, this method does not use grammar rules for natural language, probabilistic or otherwise, and does not use hard-matching rules for classification, which makes it flexible and robust to noisy input.

We showed that our approach is capable of learning under a wide range of supervision. It can utilize extra supervision provided in the form of syntactic parse trees or semantically augmented parse trees. Although experimentally, we found that extra supervision is not very useful because our basic learning algorithm without extra supervision usually determines the information

provided by the extra supervision.

The thesis also presented a semi-supervised learning algorithm for semantic parsing which uses transductive SVMs. Experiments showed that using unannotated sentences can help improve the performance of semantic parsing, particularly when training data is limited. The thesis also introduced a new method for learning from ambiguous supervision in which more than one meaning representation can be associated with each training sentence. This form of supervision is better representative of the supervision a learning system would receive when learning from its perceptual contexts with minimal human supervision. Our experiments demonstrated that our method can cope with ambiguities to learn accurate semantic parsers given sufficient training data.

If the meaning representation grammar does not conform to natural language semantics, then it could degrade the performance of a semantic parser. Manually transforming a meaning representation grammar is a non-trivial task and requires considerable expertise in the meaning representation language. In this thesis, we also investigated ways to automatically transform meaning representation grammars if they do not conform with natural language semantics. We showed the effectiveness of our proposed methods through experimental evaluation. Finally, the thesis also presented simple ensembles of semantic parsers which were shown to achieve the best overall performance.

Overall, the thesis contributed in improving the state-of-the-art in semantic parsing by providing a robust learning framework which works under

various forms of supervision. It also investigated previously unexplored aspects of semantic parsing: learning under ambiguous supervision and transforming the meaning representation grammar. It is hoped that this work will help in the development of semantic parsers for practical applications as well as stimulate further research in semantic parsing.

Bibliography

Aho, A. V., & Ullman, J. D. (1972). *The Theory of Parsing, Translation, and Compiling*. Prentice Hall, Englewood Cliffs, NJ.

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.

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

Bailey, D., Feldman, J., Narayanan, S., & Lakoff, G. (1997). Modeling embodied lexical development. In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*.

Baker, L. D., & McCallum, A. K. (1998). Distributional clustering of words for text classification. In *Proceedings of 21st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 96–103, Melbourne, Australia.

Bennett, K., & Demiriz, A. (1999). Semi-supervised support vector machines. *Advances in Neural Information Processing Systems*, *11*, 368–374.

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.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

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 (HLT/EMNLP-05)*, pp. 724–731, Vancouver, BC.

Bunescu, R. C., & Mooney, R. J. (2005b). Subsequence kernels for relation extraction. In Weiss, Y., Schölkopf, B., & Platt, J. (Eds.), *Advances in Neural Information Processing Systems 18*, Vancouver, BC.

Califf, M. E. (Ed.). (1999). *Papers from the AAAI-1999 Workshop on Machine Learning for Information Extraction*, Orlando, FL. AAAI Press.

Cancedda, N., Gaussier, E., Goutte, C., & Renders, J. M. (2003). Word sequence kernels. *Journal of Machine Learning Research, Special Issue on Machine Learning Methods for Text and Images*, *3*, 1059–1082.

Cardie, C. (1997). Empirical methods in information extraction. *AI Magazine*, *18*(4), 65–79.

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.

Chapelle, O., Schölkopf, B., & Zien, A. (Eds.). (2006). *Semi-Supervised Learning*. MIT Press, Cambridge, MA.

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 598–603, Providence, RI.

Che, W., Zhang, M., Liu, T., & Li, S. (2006). A hybrid convolution tree kernel for semantic role labeling. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL-06)*, pp. 73–80, Sydney, Australia.

Chen, Y., Wang, G., & Dong, S. (2003a). Learning with progressive transductive support vector machine. *Pattern Recognition Letters*, *24*, 1845–1855.

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

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

Collins, M. (2002a). 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. (2002b). 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. 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.

Collobert, R., Sinz, F., Weston, J., & Bottou, L. (2006). Large scale transductive SVMs. *Journal of Machine Learning Research*, *7(Aug)*, 1687–1712.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to Algorithms*. MIT Press, Cambridge, MA.

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.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, *39*, 1–38.

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.

Fellbaum, C. D. (1998). *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA.

Fillmore, C. J., Baker, F., C., & Sato, H. (2002). The FrameNet database and software tools. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC)*, pp. 1157–1160, Las Palmas.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on*

Machine Learning (ICML-96), pp. 148–156, Bari, Italy. Morgan Kaufmann.

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.

Ge, R. (2006). Learning semantic parsers using statistical syntactic parsing techniques. Doctoral Dissertation Proposal, University of Texas at Austin.

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.

Hendrix, G. G., Sacerdoti, E., Sagalowicz, D., & Slocum, J. (1978). Developing a natural language interface to complex data. *ACM Transactions on Database Systems*, *3*(2), 105–147.

Hovy, E. H., Marcus, M., Palmer, M., Pradhan, S., Ramshaw, L., & Weischedel, R. (2006). OntoNotes: The 90% solution. In *Proceedings of Human Language Technology Conference / North American Chapter of the Association for Computational Linguistics Annual Meeting (HLT-NAACL-06)*, New York, NY.

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

Ide, N. A., & Jéronis, J. (1998). Introduction to the special issue on word sense disambiguation: The state of the art. *Computational Linguistics*, *24*(1), 1–40.

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.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the Tenth European Conference on Machine Learning (ECML-98)*, pp. 137–142, Berlin. Springer-Verlag.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99)*, pp. 200–209, Bled, Slovenia.

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., & Mooney, R. J. (2006). Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL-06)*, pp. 913–920, Sydney, Australia.

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.

Kate, R. J., & Mooney, R. J. (2007a). Learning language semantics from ambiguous supervision. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI 2007)*, pp. 895–900, Vancouver, Canada.

Kate, R. J., & Mooney, R. J. (2007b). Semi-supervised learning for semantic parsing using support vector machines. In *Proceedings of Human Language Technologies: The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT-07)*, pp. 81–84, Rochester, NY.

Kudo, T., & Matsumoto, Y. (2004). A boosting algorithm for classification of semi-structured text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP-04)*, pp. 301–308, Barcelona, Spain.

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.

Langley, P., & Stromsten, S. (2000). Learning context-free gramamr with a simplicity bias. In *Proceedings of the 11th European Conference on Machine Learning (ECML-01)*, pp. 220–228, Barcelona, Spain.

Lee, L. (1996). Learning of context-free languages: A survey of the literature. Technical report TR-12-96, Center for Research in Computing Technology, Harvard University.

Levenshtein, V. I. (1966). Binary codes capable of correcting insertions and reversals. *Soviet Physics Doklady*, *10*(8), 707–710.

Lochbaum, K., Grosz, B. J., & Sidner, C. (1999). Discourse structure & intention recognition. In Dale, R., Moisl, H., & Somers, H. (Eds.), *A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text*. Marcel Dekker.

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.

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 (HLT/EMNLP-05)*, 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.

Moschitti, A. (2006). Making tree kernels practical for natural language learning. In *Proceesings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-06)*, pp. 113–120, Trento, Italy.

Muggleton, S., & Feng, C. (1992). Efficient induction of logic programs. In Muggleton, S. (Ed.), *Inductive Logic Programming*, pp. 281–297. Academic Press, New York.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society of Industrial and Applied Mathematics*, *5(1)*, 32–38.

Nguyen, L., Shimazu, A., & Phan, X. (2006). Semantic parsing with structured SVM ensemble classification models. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pp. 619–626, Sydney, Australia.

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

Ortiz, C. L., & Grosz, B. (2002). Interpreting information requests in context: A collaborative web interface for distant learning. *Journal of Autonomous Agents and Multi-Agent Systems*, *5*, 429–465.

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.

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 pars-

ing 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.

Roy, D. (2002). Learning visually grounded words and syntax for a scene description task. *Computer Speech and Language*, *16*(3), 353–385.

Schölkopf, B., Smola, A., & Müller, 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. (2004). *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.

Steedman, M. (2000). *The Syntactic Process*. MIT Press, Cambridge, MA.

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.

Tsochantaridis, I., Hofmann, T., Joachims, T., & Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *Proceedings of 21st International Conference on Machine Learning (ICML-2004)*, pp. 104–112, Banff, Canada.

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

Wang, R., & Neumann, G. (2007). Recognizing textual entailment using a subsequence kernel method. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI 2007)*, pp. 937–942, Vancouver, Canada.

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

Wong, Y., & Mooney, R. J. (2006). Learning for semantic parsing with statistical machine translation. In *Proceedings of Human Language Technology Conference / North American Chapter of the Association for Computational Linguistics Annual Meeting (HLT-NAACL-06)*, pp. 439–446, New York City, NY.

Wong, Y., & Mooney, R. J. (2007a). Generation by inverting a semantic parser that uses statistical machine translation. In *Proceedings of Human Language Technologies: The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT-07)*, pp. 172–179, Rochester, NY.

Wong, Y., & Mooney, R. J. (2007b). Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-07)*, pp. 960–967, Prague, Czech Republic.

Yang, X., Su, J., & Tan, C. L. (2006). Kernel-based pronoun resolution with structured syntactic knowledge. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL-06)*, pp. 41–48, Sydney, Australia.

Yu, C., & Ballard, D. H. (2004). On the integration of grounding language and learning objects. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-2004)*, pp. 488–493.

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.

Zettlemoyer, L. S., & Collins, M. (2007). Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL-07)*, pp. 678–687, Prague, Czech Republic.

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

Vita

Rohit Jaivant Kate was born on 14th July, 1978 in Wardha, Maharashtra state in India. He grew up in New Delhi. He completed his bachelor's degree in Computer Science and Engineering from the Indian Institute of Technology, Delhi in the year 2000. He then came to the Univeristy of Texas at Austin where he obtained Master's degree in Computer Sciences in 2002 and continued pursuing doctoral degree.

Permanent address: Department of Computer Sciences
Taylor Hall 2.124
University of Texas at Austin
Austin, TX 78712
USA

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.