In *Proceedings of the Joint 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL 2006), pp. 913-920, Sydney, Australia, July 2006.*

Using String-Kernels for Learning Semantic Parsers

Rohit J. Kate
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-0233, USA
rjkate@cs.utexas.edu

Raymond J. Mooney
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-0233, USA
mooney@cs.utexas.edu

Abstract

We present a new approach for mapping natural language sentences to their formal meaning representations using string-kernel-based classifiers. Our system learns these classifiers for every production in the formal 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.

1 Introduction

Computational systems that learn to transform natural language sentences into formal meaning representations have important practical applications in enabling user-friendly natural language communication with computers. However, most of the research in natural language processing (NLP) has been focused on lower-level tasks like syntactic parsing, word-sense disambiguation, information extraction etc. In this paper, we have considered the important task of doing deep semantic parsing to map sentences into their computer-executable meaning representations.

Previous work on learning semantic parsers either employ rule-based algorithms (Tang and Mooney, 2001; Kate et al., 2005), or use statistical feature-based methods (Ge and Mooney, 2005; Zettlemoyer and Collins, 2005; Wong and Mooney, 2006). In this paper, we present a novel kernel-based statistical method for learning semantic parsers. Kernel methods (Cristianini and Shawe-Taylor, 2000) are particularly suitable for semantic parsing because it involves mapping phrases of natural language (NL) sentences to semantic concepts in a meaning representation language (MRL). Given that natural languages are so flexible, there are various ways in which one can express the same semantic concept. It is difficult for rule-based methods or even statistical feature-based methods to capture the full 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 a potentially infinite number of features which can robustly capture these range of contexts even when the data is noisy.

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, a Support-Vector Machine (SVM) (Cristianini and Shawe-Taylor, 2000) 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 (MR) of the sentence.

Some of the previous work on semantic parsing has focused on fairly simple domains, primarily, ATIS (Air Travel Information Service) (Price, 1990) whose semantic analysis is equivalent to filling a single semantic frame (Miller et al., 1996; Popescu et al., 2004). In this paper, we have tested KRISP on two real-world domains in which meaning representations are more complex with richer predicates and nested structures. Our experiments demonstrate that KRISP compares favor-

NL: *"If the ball is in our goal area then our player* 1 *should intercept it."*

CLANG:
```
((bpos (goal-area our))
 (do our {1} intercept))
```

Figure 1: An example of an NL advice and its CLANG MR.

ably to other existing systems and is particularly robust to noise.

2 Semantic Parsing

We call the process of mapping natural language (NL) utterances into their computer-executable meaning representations (MRs) as *semantic parsing*. These MRs are expressed in formal languages which we call meaning representation languages (MRLs). We assume that all MRLs have deterministic context free grammars, which is true for almost all computer languages. This ensures that every MR will have a unique parse tree. A learning system for semantic parsing is given a training corpus of NL sentences paired with their respective MRs from which it has to induce a semantic parser which can map novel NL sentences to their correct MRs.

Figure 1 shows an example of an NL sentence and its MR from the CLANG domain. CLANG (Chen et al., 2003) is the standard formal coach language in which coaching advice is given to soccer agents which compete on a simulated soccer field in the RoboCup [1] Coach Competition. In the MR of the example, `bpos` stands for "ball position".

The second domain we have considered is the GEOQUERY domain (Zelle and Mooney, 1996) which is a query language for a small database of about 800 U.S. geographical facts. Figure 2 shows an NL query and its MR form in a functional query language. The parse of the functional query language is also shown along with the involved productions. This example is also used later to illustrate how our system does semantic parsing. The MR in the functional query language can be read as if processing a list which gets modified by various functions. From the innermost expression going outwards it means: the state of Texas, the list containing all the states next to the state of Texas and the list of all the rivers which flow through these states. This list is finally returned as the answer.

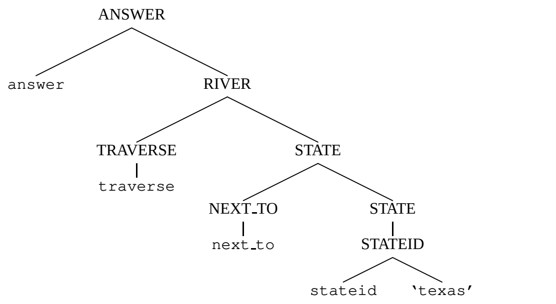
[1] `http://www.robocup.org/`

NL: *"Which rivers run through the states bordering Texas?"*

Functional query language:

`answer(traverse(next_to(stateid('texas'))))`

Parse tree of the MR in functional query language:



Productions:
ANSWER → answer(RIVER) RIVER → TRAVERSE(STATE)
STATE → NEXT_TO(STATE) STATE → STATEID
TRAVERSE → traverse NEXT_TO → next_to
STATEID → stateid('texas')

Figure 2: An example of an NL query and its MR in a functional query language with its parse tree.

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. In section 3, we describe how KRISP learns the string classifiers that are used to obtain the probabilities needed in finding the most probable semantic derivation.

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 shows a semantic derivation of the NL sentence and the MR parse which were shown in figure 2. 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.

Sometimes, the children of an MR parse tree

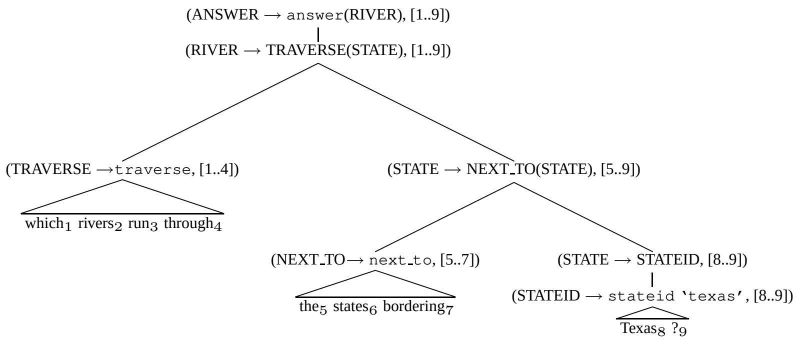


Figure 3: Semantic derivation of the NL sentence *"Which rivers run through the states bordering Texas?"* which gives MR as `answer(traverse(next_to(stateid(texas))))`.

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, then the order of the children of the node "RIVER → 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 simply as 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*.

2.2 Most Probable Semantic Derivation

Let $P_\pi(u)$ denote the probability that a production π of the MRL grammar covers the NL substring u. In other words, the NL substring u expresses the semantic concept of a production π with probability $P_\pi(u)$. In the next subsection 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:

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

The task of the semantic parser is to find the most probable 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 → NEXT_TO(STATE),[5..9])" in the derivation shown in figure 3 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 nonterminal and which cover s from index i to j. Mathematically, the most probable partial derivation $E^*_{n,s[i..j]}$ is recursively defined as:

$$E^*_{n,s[i..j]} =$$

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

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 will be 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 which have n on the LHS do not have any non-terminal on the RHS or when the substring $s[i..j]$ becomes smaller than the length t.

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 is computationally expensive, but by suitably extending the well known Earley's context-free parsing algorithm (Earley, 1970), it can be implemented efficiently. The above task has some resemblance to probabilistic context-free grammar (PCFG) parsing for which efficient algorithms are available (Stolcke, 1995), but we note that our task of finding the most probable semantic derivation differs from PCFG parsing in two important ways. First, the probability of a production is not independent of the sentence but depends on which substring of the sentence it covers, and second, the leaves of the tree are not individual terminals of the grammar but are substrings of words of the NL sentence. The extensions needed for Earley's algorithm are straightforward and are described in detail in (Kate, 2005) but due to space limitation we do not describe them here. Our extended Earley's algorithm does a beam search and attempts to find the ω (a parameter) most probable semantic derivations of an NL sentence s using the probabilities $P_\pi(s[i..j])$. To make this search faster, it uses a threshold, θ, to prune low probability derivation trees.

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 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 [2], C_π, is 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. The more the two strings share, the greater the similarity score will be.

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, which we call 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

[2] 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)
 for each π ∈ G // collect positive and negative examples for the first iteration
   for i = 1 to N do
     if π is used in parse(m_i) then
       include s_i in 𝒫(π)
     else include s_i in 𝒩(π)
 for iteration = 1 to MAX_ITR do
   for each π ∈ G do
     C_π = trainSVM(𝒫(π), 𝒩(π)) // SVM training
   for each π ∈ G    𝒫(π) = Φ // empty the positive examples, accumulate negatives though
   for i = 1 to N do
     D =EXTENDED_EARLEY(s_i, G, P) // obtain best derivations
     if ∄ d ∈ D such that parse(m_i) = getMR(d) then
       D = D ∪ EXTENDED_EARLEY_CORRECT(s_i, G, P, m_i) // if no correct derivation then force to find one
     d* = arg max_{d∈D&getMR(d)=parse(m_i)} P(d)
     COLLECT_POSITIVES(d*) // collect positives from maximum probability correct derivation
     for each d ∈ D do
       if P(d) > P(d*) and getMR(d) ≠ parse(m_i) then
       // collect negatives from incorrect derivation with larger probability than the correct one
       COLLECT_NEGATIVES(d, d*)
return classifiers 𝒞 = {C_π|π ∈ G}
```

Figure 4: KRISP's training algorithm

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 which gives the correct MR is needed to collect positive and negative examples for the next iteration. Hence in these cases, a version of the extended Earley's algorithm, 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.

From these derivations, positive and negative examples are collected for the next iteration. Positive examples are collected from the most probable derivation which gives the correct MR, figure 3 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 5 shows an example of an incorrect derivation. 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 and 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. To further increase the number of negative examples, every positive example for a production is also included as a negative example for all the other productions having the same LHS. After a specified number of MAX_ITR iterations,

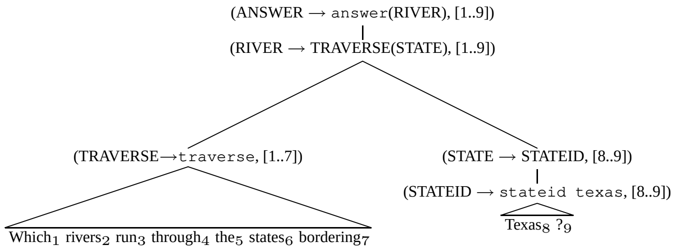


Figure 5: An incorrect semantic derivation of the NL sentence *"Which rivers run through the states bordering Texas?"* which gives the incorrect MR `answer(traverse(stateid(texas)))`.

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 subsection 2.2, and returning its MR.

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 productions and instead decides if they cover a substring of the sentence or not by matching it with the provided constant substrings.

4 Experiments

4.1 Methodology

KRISP was evaluated on CLANG and GEOQUERY domains as described in section 2. The CLANG corpus was built by randomly selecting 300 pieces of coaching advice from the log files of the 2003 RoboCup Coach Competition. These formal advice instructions were manually translated into English (Kate et al., 2005). The GEOQUERY corpus contains 880 English queries collected from undergraduates and from real users of a web-based interface (Tang and Mooney, 2001). These were manually translated into their MRs. The average length of an NL sentence in the CLANG corpus is 22.52 words while in the GEOQUERY corpus it is 7.48 words, which indicates that CLANG is the harder corpus. The average length of the MRs is 13.42 tokens in the CLANG corpus while it is 6.46 tokens in the GEOQUERY corpus.

KRISP was evaluated using standard 10-fold cross validation. For every test sentence, only the best MR corresponding to the most probable semantic derivation is considered for evaluation, and its probability is taken as the system's confidence in that MR. Since KRISP uses a threshold, θ, to prune low probability derivation trees, it sometimes may fail to return any MR for a test sentence. We computed the number of test sentences for which KRISP produced MRs, and the number of these MRs that were correct. For CLANG, an output MR is considered correct if and only if it exactly matches the correct MR. For GEOQUERY, 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. 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).

In our experiments, the threshold θ was fixed to 0.05 and the beam size ω was 20. These parameters were found through pilot experiments. The maximum number of iterations (MAX_ITR) required was only 3, beyond this we found that the system only overfits the training corpus.

We compared our system's performance with the following existing systems: the string and tree versions of SILT (Kate et al., 2005), a system that learns transformation rules relating NL phrases to MRL expressions; WASP (Wong and Mooney, 2006), a system that learns transformation rules using statistical machine translation techniques; SCISSOR (Ge and Mooney, 2005), a system that learns an integrated syntactic-semantic parser; and CHILL (Tang and Mooney, 2001) an ILP-based semantic parser. We also compared with the CCG-based semantic parser by Zettlemoyer et al. (2005), but their results are available only for the GEO880 corpus and their experimental set-up is also different from ours. Like KRISP, WASP and SCISSOR also give confidences to the MRs they generate which are used to plot precision-recall curves by measuring precisions and recalls at vari-

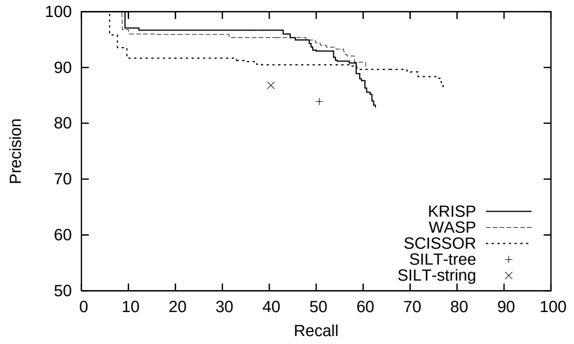


Figure 6: Results on the CLANG corpus.

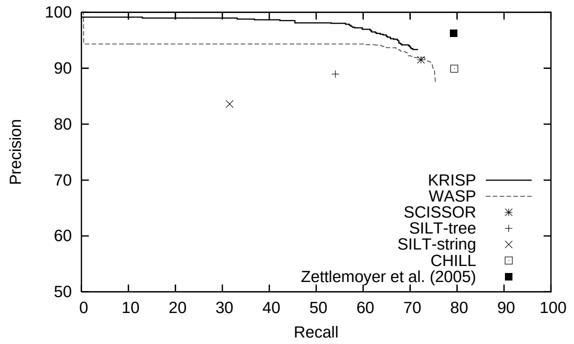


Figure 7: Results on the GEOQUERY corpus.

ous confidence levels. The results of the other systems are shown as points on the precision-recall graph.

4.2 Results

Figure 6 shows the results on the CLANG corpus. KRISP performs better than either version of SILT and performs comparable to WASP. Although SCISSOR gives less precision at lower recall values, it gives much higher maximum recall. 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 training examples because its Prolog implementation runs out of memory. For 160 training examples it gave 49.2% precision with 12.67% recall.

Figure 7 shows the results on the GEOQUERY corpus. KRISP achieves higher precisions than WASP on this corpus. Overall, the results show that KRISP performs better than deterministic rule-based semantic parsers like CHILL and SILT and performs comparable to other statistical semantic parsers like WASP and SCISSOR.

4.3 Experiments with Other Natural Languages

We have translations of a subset of the GEOQUERY corpus with 250 examples (GEO250 corpus) in

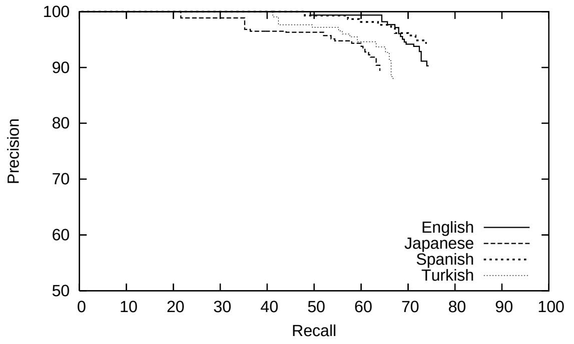


Figure 8: Results of KRISP on the GEO250 corpus for different natural languages.

three other natural languages: Spanish, Turkish and Japanese. Since KRISP's learning algorithm does not use any natural language specific knowledge, it is directly applicable to other natural languages. Figure 8 shows that KRISP performs competently on other languages as well.

4.4 Experiments with Noisy NL Sentences

Any real world application in which semantic parsers would be used to interpret natural language of a user is likely to face noise in the input. If the user is interacting through spontaneous speech and the input to the semantic parser is coming form the output of a speech recognition system then there are many ways in which noise could creep in the NL sentences: interjections (like um's and ah's), environment noise (like door slams, phone rings etc.), out-of-domain words, grammatically ill-formed utterances etc. (Zue and Glass, 2000). As opposed to the other systems, KRISP's string-kernel-based semantic parsing does not use hard-matching rules and should be thus 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 sim-

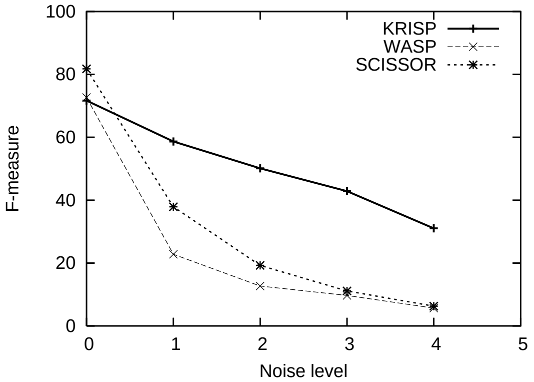


Figure 9: Results on the CLANG corpus with increasing amounts of noise in the test sentences.

ulate this type of noise by substituting a word in the corpus by another word, 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 9 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 (harmonic mean of precision and recall) 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.

5 Conclusions

We presented a new kernel-based approach to learn semantic parsers. SVM classifiers based on string subsequence kernels are trained for each of the productions in the meaning representation language. These classifiers are then used to compositionally build complete meaning representations of natural language sentences. We evaluated our system on two real-world corpora. The results showed that our system compares favorably to other existing systems and is particularly robust to noise.

Acknowledgments

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

References

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

Nello Cristianini and John Shawe-Taylor. 2000. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

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

R. Ge and R. J. Mooney. 2005. A statistical semantic parser that integrates syntax and semantics. In *Proc. of 9th Conf. on Computational Natural Language Learning (CoNLL-2005)*, pages 9–16, Ann Arbor, MI, July.

R. J. Kate, Y. W. Wong, and R. J. Mooney. 2005. Learning to transform natural to formal languages. In *Proc. of 20th Natl. Conf. on Artificial Intelligence (AAAI-2005)*, pages 1062–1068, Pittsburgh, PA, July.

Rohit J. Kate. 2005. A kernel-based approach to learning semantic parsers. Technical Report UT-AI-05-326, Artificial Intelligence Lab, University of Texas at Austin, Austin, TX, November.

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

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444.

Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. 1996. A fully statistical approach to natural language interfaces. In *Proc. of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pages 55–61, Santa Cruz, CA.

John C. Platt. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In Alexander J. Smola, Peter Bartlett, Bernhard Schölkopf, and Dale Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 185–208. MIT Press.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proc. of 20th Intl. Conf. on Computational Linguistics (COLING-04)*, Geneva, Switzerland, August.

Patti J. Price. 1990. Evaluation of spoken language systems: The ATIS domain. In *Proc. of 3rd DARPA Speech and Natural Language Workshop*, pages 91–95, June.

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

L. R. Tang and R. J. Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proc. of the 12th European Conf. on Machine Learning*, pages 466–477, Freiburg, Germany.

Yuk Wah Wong and Raymond J. Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proc. of Human Language Technology Conf. / North American Association for Computational Linguistics Annual Meeting (HLT/NAACL-2006)*, New York City, NY. To appear.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proc. of 13th Natl. Conf. on Artificial Intelligence (AAAI-96)*, pages 1050–1055, Portland, OR, August.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proc. of 21th Conf. on Uncertainty in Artificial Intelligence (UAI-2005)*, Edinburgh, Scotland, July.

Victor W. Zue and James R. Glass. 2000. Conversational interfaces: Advances and challenges. In *Proc. of the IEEE*, volume 88(8), pages 1166–1180.