Learning To Understand Web Site Update Requests

Xxxxxx X. Xxxxxxx
Center for Xxxxx Xxxxx of Xxxxx
University of Xxxxx
xxxx@xx.xxx.edu

Yyyyyy Yyyyyyy
Department of Yyyyy
Yyyyy University
yy@yyy.yyyy.edu

Zzzzzz Zzzzzzz, Jr.
Department of Zzzzz
University of Zzzzz
xxxx@xx.xxx.edu

Abstract

We experimentally evaluate components of a system that learns to analyze natural-language requests to update information on a database-backed website. Our long-term goal is to develop a system which can adapt to changes in the distribution of requests as the underlying database schema changes—in short, a system that performs deep analysis of text in a domain of discourse that is limited, but which grows over time. We describe a scheme for decomposing request-understanding into a sequence of entity recognition and text classification tasks, each of which can be solved using standard learning methods. We then present experimental results that quantify how well these tasks can be learned.

1 Introduction

Because deep understanding of unrestricted natural language is difficult, practical NLP systems generally restrict either the domain of discourse, or the depth of analysis. In this paper, we describe a system which provides relatively deep analysis of text within a domain of discourse that is *limited*, but *extensible*. Specifically, we describe a system for understanding a certain class of database update requests over a database with a schema that changes over time. Typical changes in the schema might include the addition of new database relations or new attributes. Thus at any point in time, the domain of discourse is limited; however, the domain of discourse can grow gradually over time. This setting requires that the NLP system be able to learn how to adapt to the expanding domain of discourse.

As a concrete illustration of this task, consider an NLP system which assists a webmaster in maintaining an organizational web site. To motivate this, we note that many organizations want some sort of central control of public web sites, so as to maintain a uniform look and feel. However, the factual content which such a web site presents often concerns many smaller organizational units, and a wide range of people might want to post news items, update directory information, and so on. One possible solution to this problem is a database-backed website, with a database of information that can be updated by all users. However, this approach may not be accepted in all contexts—for instance, individual users, each of whom may only contribute a few database changes a year, may be reluctant to learn how to manipulate the database to make occasional updates.

Many organizations thus adopt a model in which users submit update requests via email in natural language to a human webmaster. Some examples of such update requests are shown in Figure 1.

In this setting, one component of the webmaster's work is processing a stream of email requests, each of which suggests a certain specific factual change to the database on which the web site is

Add the following contact to the Staff list. Arthur Scott ascott@ardra.com Rm 7992 281 1914
On the events page, delete row ”December 23 Assembly for Automotive Engineers Conference Room A21”
On the people page under Tommy Lee delete 281 2000
Please delete Kevin Smith’s phone number - thanx, Martha
Change Mike Roberts to Michael Roberts.
Please add Greg Johnson’s phone number- 281 2000

Figure 1: Representative update requests (edited slightly for space and readability)

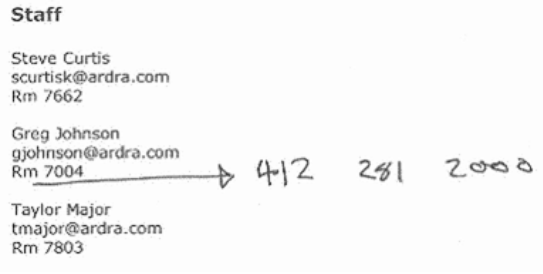

Figure 2: A pictorial description of an update task

based. The specific content of these update requests is restricted, but will change whenever new kinds of information are added to the web site.

In this paper, we will explore this “webmaster support task” in some detail. Our motivation for considering this task is partly practical, as this work is part of a larger effort to construct a robust intelligent assistant for common work-related tasks. However, we are also interested in the task as an example of an NLP problem that requires deep processing over a restricted domain that naturally changes over time.

More specifically, this paper experimentally evaluates some components of a system that (a) can *analyze* natural-language database update requests in enough detail to implement them automatically; and (b) can *learn* to perform this analysis, for any given database schema, from a moderate amount of training data. Our long-term goal is that the system also be able to (c) *acquire training data* from its users in the course of normal operation. With these capabilities, the system will be able to learn to understand new update requests when the domain of discourse expands (as a result of new kinds of information being added to the website).

Below, we will describe a scheme for decomposing request-understanding into a sequence of entity recognition and text classification tasks, each of which can be solved using standard learning methods. We then present experimental results that quantify how well these tasks can be learned. First, however, we will describe the corpus of requests we use in our experiments.

2 The Experimental Corpus

To collect a corpus, a series of controlled human-subject experiments were performed, in which participants were given a series of tasks in pictorial form (Figure 2) and asked that they compose and send an appropriate e-mail messages to a webmaster agent. In response, the agent returned a *preview* of the updated page, and a *pre-filled template* that contained a structured representation of the user’s request. The user could correct errors by editing text in various template slots, or by choosing from pull-down menus.

The advantage of this procedure is that it provides a corpus of real user-generated update requests early in the development cycle, so that one can immediately begin studying the NLP issues. However, the procedure is relatively expensive, and hence our corpus is relatively small, containing only 617 requests. This is adequate for training purposes—after all, a system that adapts to changes too slowly is not very useful—but a larger corpus would be preferable for evaluation purposes.

Further, to simplify the procedure for experimenting with human subjects, some pictorial task descriptions were presented to several different users. This sort of duplication can lead to undesirable behavior for a learning system: for instance, if the task of Figure 2 were repeated many times in the data, then the system might learn a correlation between the phrase “Greg Johnson” and the task of adding a phone number. To address this problem, we manually replaced duplicate entity names with alternative values throughout the corpus, preserving surface features such as capitalization patterns and misspellings. We also restrict all the experi-

ments of this paper so that the messages associated with a particular task are never split between training and test sets, thus ensuring that intra-task correlations (like the "Greg Johnson-add phone number" correlation above) can not be used to improve test set performance. Stated another way, we sample the corpus so that every test message is for a novel task—not a task seen in training.

Another limitation of this corpus concerns the database schema for the web sites of the human-subject experiments. Let us first distinguish between two notions of "type". A *linguistic type* has the property that entities of the same type are linguistically similar: this is the notion of type used by entity recognition (NER) systems, which extract instances of particular linguistic types such as persons, organizations, and locations. In contrast, a *database type* is a set of values that can be legally assigned to a particular attribute. In the database schema of the corpus, no relations contained two or more attributes of the same linguistic type. Hence given a relation R and a entity e of known linguistic type we can readily determine which attribute of R corresponds to e: for instance, a phone number can only be stored in one possible column of the "person" relation. In general, however, this will not be the case—consider for instance a database containing home and work phone numbers for each person.

In this paper, we assume that no relation contains multiple attributes of the same linguistic type. Although the analysis technique we propose can be extended so as to relax this assumption, our current corpus does not support an experimental study of the necessary techniques.

Finally, we note that this text is very noisy—for instance, typos are common, and capitalization is inconsistent. This is typical of informal text like email but a consequence is that standard shallow-NLP tools such as POS tagging and NP chunking are rather unreliable. Hence care is needed to construct reliable features which exploit the syntactic properties of the message. In the experiments of this paper we used a version of Brill's POS tagger (Brill, 1995) and a hand-coded NP-chunker which was tuned for email (using a different corpus).

3 Understanding Update Requests

3.1 Analysis procedure

Figure 1 gives some example web site update requests. Although many other kinds of requests are possible (e.g., "The animated gif in the logo doesn't flash properly when I view it from my home PC") we will assume that our web site is backed by a relational database, and focus on messages that request a factual update to the underlying database. The analysis procedure we describe contains several steps.[1]

Request type classification. An informal preliminary analysis of real webmaster request logs suggested that most factual-update requests are in one of the following forms: to add a new tuple to the database; to delete an existing tuple; to delete a value from an existing tuple; or to alter (add or replace) a value of an existing tuple. One step of analysis is determining the type of request. This is a *text classification* task: each request will be mapped to one of the four categories *addTuple*, *deleteTuple*, *deleteValue*, and *alterValue*, or else mapped to *otherRequest* (for requests not falling into one of the four supported varieties).

Entity recognition. Another part of the analysis is to identify all entities in a request. Figure 3 shows the result of correctly recognizing person names, email addresses, phone numbers, room numbers, and event titles in some sample requests. The subscript after an entity indicates its linguistic type (for instance, "person" or "room number").

Role-based entity classification. We distinguish between four different *roles* for an entity in an update request. (a) An entity is a *key entity* if it serves to identify the database tuple which is to be modified. In the figure, key entities are marked with a superscript K. (b) An entity is a *newEntity* (in the figure, marked with a superscript N) if it is a value not currently in the database, which the user desires to be stored in the database. (c) An entity is an *oldEntity* (marked with superscript O) if it is a value currently in the database which the user

[1] For pedagogical reasons, we will describe these steps as if they are taken in a fixed order. However, the steps are not independent—i.e., information from each step of analysis affects all other steps—and it may be computationally preferable to re-order or interleave the decisions associated with each step.

	Request	Request Type	Target Relation	Target Attribute
1	Add the following contact to the Staff list. **[Arthur Scott]**$^{N}_{\text{person}}$ **[ascott@ardra.com]**$^{N}_{\text{email}}$ Rm **[7992]**$^{N}_{\text{room}}$ **[412 281 1914]**$^{N}_{\text{phone}}$	addTuple	people	–
2	On the events page, delete row "**[December 23]**$^{K}_{\text{date}}$ **[Assembly for Automotive Engineers Conference]**$^{K}_{\text{eventTitle}}$ Room **[A21]**$^{K}_{\text{room}}$"	deleteTuple	events	–
3	On the people page under **[Tommy Lee]**$^{K}_{\text{person}}$ delete **[412 281 2000]**$^{O}_{\text{phone}}$	deleteValue	people	phoneNum
4	Please delete **[Freddy Smith's]**$^{K}_{\text{person}}$'s phone number - thanx, **[Martha]**$_{\text{person}}$	deleteValue	people	phoneNum
5	Change **[Mike Roberts]**$^{K}_{\text{person}}$ to **[Michael Roberts]**$^{N}_{\text{person}}$ on the People page.	alterValue	people	personName
6	Please add **[Greg Johnson]**$^{K}_{\text{person}}$'s phone number- **[412 281 2000]**$_{\text{phone}}$	alterValue	people	phoneNum

Figure 3: Analyzed update requests.

expects to be replaced with a *newEntity*. (d) Entities unrelated to the execution of the request are called *noiseEntities*. In the figure, they have no superscript marking.

Target relation classification. Since each request concerns a single tuple, it necessarily also concerns a single relation. The second column of Figure 3 shows the relation associated with each request. For any fixed database schema, there are fixed set of possible relations, so this is another *text classification* operation.

Target attribute classification. Given entities, the roles of entities, the target relation, and the request type, the semantics of the tuple-based commands will be completely determined.[2] However, *deleteValue* requests may still be underspecified. As an example consider request 4 in the figure: the previous analysis tells us we should delete some attribute value from the tuple of the "person" relation with the key value of "Tommy Lee", but does not tell us *which* attribute to delete.

Hence, to complete the analysis for these requests, it is necessary to determine the attribute that needs to be deleted. This is again a text classification task: given a fixed database schema, only a fixed number of attributes need to be considered as possible targets.

[2]However, *addTuple* requests would not be completely determined if there were two or more attributes of the same linguistic type.

Let R be the *target relation*; let T be the *request type*. If $T \neq addTuple$ then let τ be the tuple in R that best matches the *keyEntities* in the request. If $T = alterValue$ or $T = deleteValue$, let a be the *target attribute*.

If $T = addTuple$ then build a tuple τ' containing all the *newEntities* in the request, and propose the update: "add τ' to R".

Otherwise, if $T = deleteTuple$ then propose the update: "delete τ from R".

Otherwise, if $T = deleteValue$ then propose the update: "delete the value $\tau.a$ in R".

Otherwise, if $T = alterValue$ and there are *newEntity* in the request, then let E_N be the *newEntities* in the request, and let E_O be the *oldEntities*. If the database schema specifies that $R.a$ is filled by a set of values, propose the update: "set $\tau.a = \tau.a \cup E_N - E_O$". Otherwise, if the database schema specifies that R.a is filled by a single value propose the update: "set $\tau.a = e$", where e is the highest-confidence element of E_N.

Figure 4: The procedure for building a database update from an analyzed request.

3.2 Using the analysis

To summarize, we have assumed thus far that requests affect a single database, and that no database relation contains more than one attribute of any single linguistic type. We will also assume that the set of linguistic types present in the database is fixed (so that learning is not needed to construct entity recognizers). Given these constraints, the steps described above are sufficient to map a request to a database update. Figure 4 sum-

marizes the algorithm for doing this.[3]

Recall that in our expected setting, the system will verify changes before executing them, and the user can make corrections in the system's analysis of a request. Even if the user (or webmaster) needs to correct some errors in the analysis, the update process is often faster than manually changing a database (as has been experimentally verified in other human-subject experiments (Ano, 2004)). Since partially correct analyses are useful, we will focus our experiments on evaluation on the individual steps of the analysis.

4 Experiments

The long-term goal of this research project is to allow the system to continually retrain on data collected from users, and thus adapt to domain changes over time. In other words, learning is part of what the the NLP system does; this is in contrast to most empirical work in computational linguistics, which uses learning to build a non-adaptive NLP component.

However, the validation of such a system is similar to the validation of an empirically developed NLP system: to assess learning, we should measure performance on held-out test data.

4.1 Experimental procedure

Recall that our corpus contains 617 request messages, elicited from human users by presenting them with pictorial descriptions of certain update tasks (as in Figure 2). In all the experiments reported below, we split the data so that no test message concerns a task represented in the training set. This sampling scheme estimates performance in an environment in which many different tasks are encountered.

4.2 Entity Recognition

We evaluated entity recognition performance for seven linguistic types: time, date, amount, email addresses, phone numbers, room numbers, and person names. Other entity types are present in the data (e.g., job titles and organization names) but not in sufficient quantity to support systematic evaluation.

[3]Note the output of the system is a "proposed" update on the user's view of the database. In general performing this single intended change will result in multiple updates to the underlying database, and converting the proposed update to database commands that have the intended effect may be non-trivial.

Linguistic Type	Test Set	
	Full Corpus	Validation
Rules		
Time	95.7	n/a
Amount	92.6	n/a
Room#	87.8	(100.0) 59.1
Phone	92.6	(98.2) 52.4
Date	96.1	97.7
Email	100.0	100.0
Person	81.2	73.9
	Train/Test	
	50/50	80/20
VP-HMM		
Time	90.0	84.9
Date	76.3	85.0
Amount	82.3	100.0
Email	0.0	100.0
Phone	90.0	84.9
Room#	80.0	81.6
Person	78.0	77.6
Person+	89.3	84.6

Figure 5: Entity recognition results: F1 measures

Table 5 shows results for entity recognition using hand-coded rules (cascaded FSTs) on this task. We evaluate them on the main corpus, and also on a set of 96 additional update-request messages which were collected in a second, later series of human-subject experiments. Unfortunately the linguistic entity types were somewhat different in the later round of experiments: no time or amount entities were present in this "validation" set, and room and phone numbers which followed different naming conventions. The column labeled "Validation" shows performance on the 96-element "validation" set. For room and phone numbers also we show (in parentheses) the performance of slightly modified hand-coded rules which cover the naming conventions.

We also show results for learning, using the method of Collins (2002) for discrimatively training HMMs, training on either 50% or 80% of the available data, and testing on the remainder. A

Entity Role	Train/Test 50% 5-CV	5-CV	Def. Error
keyEntity	79.8 / 17.1	84.3 / 14.1	44.2
newEntity	77.5 / 14.7	86.4 / 8.8	48.5
oldEntity	67.4 / 4.0	80.4 / 2.5	6.8

Figure 6: Role-based entity classification results: F1 measures and percent error

Entity Role	Train/Test 50% 5-CV	5-CV	Def. Error
people	92.9 / 9.1	95.7 / 5.5	38.7
budget	56.3 / 6.8	67.3 / 5.5	10.0
events	80.2 / 8.1	79.5 / 8.6	22.7
sponsors	88.2 / 1.3	81.3 / 1.9	6.0

Figure 7: Target relation identification results: F1 measures and percent error

Request Type	Train/Test 50% 5-CV	5-CV	Def. Error
deleteTuple	18.7 / 19.8	65.1 / 12.3	18.0
deleteValue	50.0 / 12.6	27.3 / 10.4	9.1
without nouns			
deleteTuple	7.2 / 20.7	57.1 / 14.1	18.0
deleteValue	51.3 / 12.0	33.3 / 9.1	9.1
with dictionary			
deleteTuple	25.7 / 16.9	81.3 / 6.5	18.0
deleteValue	60.6 / 10.5	85.2 / 2.6	9.1

Figure 8: Request type identification results: F1 measures and percent error

single feature set, similar to that used by Collins, was used for all the entities, and the row labeled "Person+" uses some additional engineered features, derived in part from a large name lexicon.

As the experiments show, both hand-coded rules and learned extractors perform reasonably well on this task, in spite of the informal nature of the text. Many of the learned extractors require a relatively small amount of training data.

In spite of the success of learning at this task, however, we still propose to assume a fixed set of linguistic types, as it is still unclear if the data required for extraction learning can be collected using the procedure described in Section 3.2.

4.3 Role-based entity classification

The language used to convey the role of an entity may depend on the database schema. For instance, a person-name *newEntity* might be described differently when it is added to a project relation ("The ACQUAINT project has a new member, Tom Edison - please update the web page") than when it is added to a calendar relation ("My 2pm meeting on July 2nd is with Tom Edison"). Therefore, we use a learned classifier to classify entities by roles. The results are shown in Table 6. Here we used 5-fold cross-validation (5-CV) to measure performance, and also report results of a 5-CV experiment in which only 50% the available training data (i.e., 40% of the full corpus) is used on each fold.

The learner we is the version of AdaBoost described by Schapire and Singer (1998), used to boost decision stumps for 1000 iterations. The features used for the learner are as follows. (a) The closest preceding preposition. (b) The presence or absence of a possessive marker after the entity. (c) The closest preceding "action verb". An action verb is one of a few dozen words generally used to denote an update, such as "add", "delete", etc. (d) The closest preceding word w which is either a preposition, an action verb, or a determiner. This is intended to detect whether the entity is part of a determined NP.

4.4 Target relation classification

Determining the target relation is clearly a schema-dependent step. We used the same learner and a bag-of-words representation of a request for this task. The results are shown in Table 7. Even this simple representation for requests performs quite well, especially for the more frequently updated relations.

4.5 Request type classification and target attribute classification

In many cases the type of a request can be determined from the roles of the entities in the request. For instance, an *addTuple* request has no *keyEntities* but may have multiple *newEntities*; conversely a *deleteTuple* request has *keyEntities*, but no *newEntities*; and only an *alterValue* request can have both *keyEntities* and *newEntities*.

A remaining ambiguity is between *deleteValue* and *deleteTuple* requests, which are surprisingly hard to distinguish. These requests are often syntactically quite similar. Consider for instance the requests "delete the extension for Dan Smith" and "delete the line for Dan Smith". The first is probably a *deleteValue* for a phone number, but the second might be a *deleteTuple* instead.

To distinguish *deleteTuple* and *deleteValue*, we used the following as features. (a) The counts of the number of *keyEntities*, *oldEntities*, and *newEntities* in a request. (b) The action verbs appearing in request. (c) The nouns that appear in an NP immediately following an action verb, or that appear in NPs before an action verb in passive form.

The results are shown in Table 8. To give some additional insight into this problem, we also give results without feature (c)—i.e., without the nouns that are the probable object of the action verb–and with an additional "dictionary" feature: do any selected nouns appear in a dictionary of 12 common attribute names (e.g., "phone", "extension", "room", "office", etc).

Note that the addition of a small amount of *schema-dependent* information (in the form of the dictionary) leads to a substantial improvement in performance. As a practical matter, it may be desirable to extend the learning approach described here and allow some types of schema-dependent knowledge (such as a list of words that might be used for a new attribute) to be injected into the learning process when the schema changes. We note also that there are relatively few useful examples in the corpus—over 80% of the requests are neither *deleteTuple* or *deleteValue*.

The classification of requests by target attributes is a very similar prediction problem, except that rather than determining *if* a delete request concerns an attribute, one must determine *which* attribute the request concerns. Given our assumptions, this step need only be performed for *deleteValue* requests that contain no *oldEntity*, and hence there is very little relevant training data. The feature set above can also be used to predict target attributes fairly reliably for attributes which used in this context moderately frequently, but details are omitted due to space limitations.

5 Related work

Previous NLP systems have generally either performed deep semantic analysis in a restricted domain, or else a shallower analysis in a broader domain. While learning has been an important tool for developing broad-coverage NLP components such as POS taggers, parsers, and named entity recognition systems, there have surprisingly few attempts to learn to perform deep semantic analysis. Notable exceptions are the CHILL system (Zelle and Mooney, 1996), which learns to parse database queries into a meaning representation language, and the work by Miller *et. al* (1996) on using a combination of generative models to extract facts from text. Work in learning such "semantic parsers" is surveyed and motivated elsewhere (Mooney, 2004).

There are several important differences between the work described in this paper and prior efforts. One is that our learning method uses only data which can plausibly collected from user feedback. In contrast, Zelle & Mooney's system learns from sentence/query pairs, and Miller *et. al.* use a variety of sources for training data including POS-tagged text, parsed sentences, and semantically annotated text.

A second difference is that we consider understanding update requests, rather than understanding queries (like Zelle & Mooney) or declaratively stated facts (like Miller *et al*). One advantage of the update-request task is that a partially correct analysis is still useful, and furthermore, is likely to elicit user feedback which can be used for training. In contrast, it is unclear how useful it is to answer an imperfectly analyzed database query, or what could be learned from such an episode.

On the other hand, we limit ourselves to conceptually simple database updates, while Zelle & Mooney consider complex structured queries. There are also numerous smaller differences stemming from the nature of the task and corpus.

Although the purpose and scope of our research is different, the entity role classification step we consider above is broadly similar to recent work on semantic role analysis (Fillmore et al., 2000; Gildea and Jurafsky, 2002), and earlier work on case-role assignment (*e.g.*, (Miikkulainen

and Dyer, 1991)).

Lockerd *et. al* (2003) propose an automated Webmaster called "Mr. Web" which has a similar email-based interface. They manually analyzed 325 update requests to assess their linguistic regularity, but they do not describe any algorithm for processing the requests, and do not consider issues of adaptation.

6 Conclusions

We have described and experimentally evaluated a scheme for decomposing request-understanding into a sequence of entity recognition and text classification tasks. One interesting aspect of this decomposition is that it enables a large amount of the request-understanding system to be learned from data; further, much of this data can be plausibly collected from interactions with end users.

Somewhat surprisingly, current methods learn to perform each of the steps in our analysis reliably—even from our (modest-sized) corpus. F1 performance in eighties can be obtained for extracting seven entity types; identifying three of four target relations; and *entity role classification*. Only for *request type identification* is it necessary to exploit any schema-dependent knowledge to obtain reasonable performance, and this knowledge consists of only a 12-word dictionary.

Acknowledgements

We thank Xxxx Xxxxx for work on the hand-coded person name extractor, POS tagger, and NP-chunker used in these experiments. We also wish to thank Xxxx Xxxxx, Xxxxx Xxxxx, Xxxxx Xxxx, Xxxxx Xxxxxxxx, Xxxxx Xxxxxx and Xxxxx Xxxxxx for providing us with the experimental corpus.

References

2004. Evaluation of a human-agent dialog interface for an intelligent webmaster assistant. Author and venue not disclosed to preserve anonymity.

Eric Brill. 1995. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 24(1):543–565.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Empirical Methods in Natural Language Processing (EMNLP)*.

C. J. Fillmore, F. C. Baker, and H. Sato. 2000. The framenet database and software tools. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC)*, pages 1157–1160.

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

Andrea Lockerd, Huy Pham, Taly Sharon, and Ted Selker. 2003. Mr.web: An automated interactive webmaster. In *Extended abstracts on Human factors in Computer Systems (CHI'03)*, Ft. Lauderdale, Florida, April.

R. Miikkulainen and M. G. Dyer. 1991. Natural language processing with modular PDP networks and distributed lexicon. *Cognitive Science*, 15:343–399.

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

Ray Mooney. 2004. Learning semantic parsers: An important but under-studied problem. In *Working notes of the AAAI spring symposium on language learning*, pages 39–44, Palo Alto, CA, March. AAAI Press.

Robert E. Schapire and Yoram Singer. 1998. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 80–91.

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