Learning to Understand Web Site Update Requests

William W. Cohen
Center for Automated Learning & Discovery
Carnegie Mellon University
wcohen@cs.cmu.edu

Einat Minkov
Language Technologies Institute
Carnegie Mellon University
einat@cs.cmu.edu

Anthony Tomasic
Institute for Software Research
Carnegie Mellon University
tomasic@cs.cmu.edu

Abstract

Although Natural Language Processing (NLP) for *requests* for information has been well-studied, there has been little prior work on understanding requests to update information. In this paper, we propose an intelligent system that can process natural language website update requests semi-automatically. In particular, this system can analyze requests, posted via email, to update the factual content of individual tuples in a database-backed website. Users' messages are processed using a scheme decomposing their requests into a sequence of entity recognition and text classification tasks. Using a corpus generated by human-subject experiments, we experimentally evaluate the performance of this system, as well as its robustness in handling request types not seen in training, or user-specific language styles not seen in training.

1 Introduction

In this paper, we present a natural language system that helps a webmaster maintain the web site for an organization. Specifically, we describe a system for understanding certain natural-language requests to change the factual content on a website. We will assume that the website is based on a database, and focus on requests to update specific facts in this database.

To motivate this, we note that although NLP for requests to *deliver* information(i.e., question-answering) has been well-studied, there has been little prior work on NLP for requests to *update* information. However, NLP for update requests is an attractive research problem, in part because a user can more easily detect an imperfectly-processed utterance.

As a concrete example of update requests, we consider here requests for web-site updates. Such a system would be practically useful, as many organizations maintain a single large database-backed web site that includes information that can be contributed or corrected by many individuals. Since individual users, each of whom may only contribute a few database changes a year, may be reluctant to learn how to interface with the database to make their occasional updates, in many orginazations users submit update requests via email in natural language to a human webmaster. Frequently, there is a waiting period before the human webmaster can incorporate corrections, leading to long processing times, and a web site that is not up to date.

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.

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

In this paper, we describe an intelligent system that can process website update requests semi-automatically. First, natural language processing is used to analyze an incoming request. Based on the analysis, the system then constructs an executable version of the proposed change, which is represented as a pre-filled instance of a form. By examining the form, the end user can efficiently determine whether the analysis step was correctly accomplished, and, if necessary, override the results of the agent's analysis by changing values in the form. Prior experiments with human subjects have shown that this process is an effective means of reducing human effort, even if the initial analysis step is imperfect [2004].

This paper focuses on the natural-language processing part of this system. As is typical of informal text like email, users' messages are often ungrammatical, use capitalization patterns inconsistently, use many abbreviations and include typos (as illustrated in Figure 1). As a consequence, standard shallow-NLP tools such as part-of-speech tagging and noun-phrase chunking, which are preliminary steps for text parsing, are quite unreliable. We therefore suggest here a learning approach, where rather than parse the text into a framework of pre-modeled domain knowledge, we decompose the general task into a sequence of entity extraction and classification sub-tasks. All of these sub-tasks can be learned from incoming messages, improving system performance over time.

We will first describe a scheme for decomposing request-understanding into a sequence of learning tasks. Next, we describe the corpus of requests that is used for performance evaluation. We then describe each of the learning sub-tasks

in detail, along with experimental results. We also present experimental results on the robustness of the system – in particular, how the system will perform on request types not seen in training, or on user-specific language usage not seen in training. Finally, we evaluate the end-to-end system's performance, to determine what fraction of messages can be processed completely without errors. We conclude with a review of related work and our conclusions.

2 Understanding Update Requests

2.1 Analysis procedure

Figure 1 gives some example web site update requests that are addressed by the given analysis procedure. General requests that are not for factual update (e.g., "The animated gif in the logo doesn't flash properly when I view it from my home PC") will simply be flagged and forwarded to the real human webmaster.

The analysis procedure contains the following steps.

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

Named entity recognition (NER). Another step of the analysis is to identify all *entity names* in a request. Figure 2 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 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 *keyEntity* if it serves to identify the database tuple which is to be modified. In the figure, key entities are marked with a superscript K. An example is the entity "Freddy Smith" in the sentence "please delete Freddy Smith's phone number". (b) An entity is a *newEntity* (marked with a superscript N) if it is a value to be stored in the database. (c) An entity is an *oldEntity* (superscript O) if it is a value currently in the database which the user expects to be replaced with a *newEntity*. (d) Entities unrelated to the execution of the request are considered to be *noiseEntities*. In the figure, they have no superscript marking. Role-based entity classification is an *entity classification* task, in which entities produced by the earlier NER step are given an additional classification.

Target relation classification. The second column of Figure 2 shows the relation associated with each request. For any fixed database schema, there is a fixed set of possible relations, so this is a *text classification* operation.

Target attribute classification. Given entities, the roles of entities, the target relation, and the request type, the semantics of the many tuple-based commands will be often completely determined. One type of request that may still be underspecified is the *deleteValue* request. 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 specify the value to be deleted. Hence, to complete the analysis for *deleteValue* requests, it is necessary to determine the attribute that needs to be deleted. This is again a text classification task: given a database schema, only a fixed number of attributes need to be considered as possible targets.

For pedagogical reasons, we have described these steps as if they are taken separately. However, the steps are not independent—i.e., information from each step of analysis may affect other steps. In section 6 we describe and evaluate a particular sequence, where outputs of some steps are propagated as inputs to the next steps.

3 The Experimental Corpus

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

Overall, the human-generated corpus contains a total of only 617 example requests, involving approximately 20 subjects, and about 30 different tasks.

Note that the same pictorial task descriptions were presented to multiple users. This sort of duplication can lead to undesirable behavior for a learning system: if a certain pictorial task ,demonstrating addition of a phone number to a person named Greg Johnson for example, is represented by multiple similar examples 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.

The requests in the corpus are largely factual updates concerning a single tuple in the database, so we will focus our attention on such requests. Also, the relations in the underlying database schema of the corpus do not contain two attributes or more of the same type, where "type" is defined by the output of the entity recognizer. For instance, personal details might include a home phone number and an office phone number, but our corpus has no such duplications. Duplications of this sort would require an additional entity classifier.

As mentioned, the text itself is often ungrammatical and noisy. We pre-processed the text, annotating it with a version of Brill's part-of-speech tagger [Brill, 1995] and a hand-coded noun-phrase chunker which was tuned for email (using a different corpus). In learning, however, we rely mainly on alternative features that exploit syntactic properties of the messages. These features prove to be informative for the noisy text in our corpus.

	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]**$^{N}_{\text{phone}}$	alterValue	people	phoneNum

Figure 2: Analyzed update requests.

4 Learning

Below we describe each of the individual learning tasks. Relevant experimental results are given for every component.

4.1 Entity Recognition

Named Entity Recognition (NER), or the identification of the substrings of a request that correspond to entity names, is a well-studied yet non-trivial Natural-Language Processing task. We evaluated NER performance for seven linguistic types: time, date, amount, email addresses, phone numbers, room numbers, and personal names. The data includes some mentions of additional entity types (e.g., job titles and organization names) but not in sufficient quantity for learning.

We experimented with two approaches to entity extraction: a rule-based approach, in which hand-coded rules are used to recognize entities; and learning-based extraction. The rule language we used is based on cascaded finite state machines. The learning algorithm we use here is VPHMM, a method for discriminatively training hidden Markov models using a voted-perceptron algorithm [Collins, 2002].

We found that manually constructed rules are best suited for entities such as e-mail addresses and temporal expressions. These types are based on limited vocabularies and fairly regular patterns, and are therefore relatively easy to model manually. Email addresses are an extreme example of this: a simple regular expression matches most email addresses.

Table 1(a) shows the results of extraction using hand-coded rules for email and temporal expressions. We evaluated the rules on the main corpus, which was used for generating the rules, and also on a 96-message "validation set", containing messages which were collected in a second, later series of human-subject experiments (unfortunately, no time expressions were present in this additional set.) As shown in the table, the entity F1 performance is above 95% for all cases that could be evaluated.

In Table 1(b) we show results for learning on the full set of entity types, applying the VPHMM algorithm. Here NER is reduced to the problem of sequentially classifying each token as either "inside" or "outside" the entity type to be extracted. Performance is evaluated by the F1-measure[1], where entities are only counted as correct if both start and end boundaries are correct (i.e., partially correct entity boundaries are given no partial credit.) The left-hand columns in the table (titled "5CV") show F1-measure performance on unseen examples, as estimated using 5-fold cross validation. The right-hand columns will be discussed later.

Performance is shown for two sets of features. The *base feature* set corresponds to words and capitalization templates over a window including the word to be classified, and the three adjacent words to each side. The second set of features, labeled *tuned features* in the table, is comprised of the base features plus some additional, entity-type specific features,

Type	Test Set: Full Corpus	Test Set: Validation
Time	95.7	n/a
Date	96.1	97.7
Email	100.0	100.0

(a) Rules

Type	Base f. $5CV$	Tuned f. $5CV$	Tuned features $5CV_{USR}$	Tuned features $5CV_{REQ}$
Time	87.7	91.2	88.2	93.9
Date	88.5	94.4	95.8	88.9
Amount	89.7	93.1	93.1	85.4
Phone	87.3	94.2	92.4	82.3
Room#	81.9	90.4	87.1	83.0
Person	80.9	90.3	83.6	88.3

(b) Learning

Table 1: Entity recognition results: F1 measures

[1] F1 is the geometric mean of recall and precision.

which are constructed using the same rule language used to build the hand-coded extractors. For example, in extracting dates we added an indicator as to whether a word is a number in the range 1-31; for personal names, we added an indicator for words that are in certain dictionaries of first and last names.

Overall, the level of performance for extraction – better than 90% for every entity type, using the tuned features – is very encouraging, especially considering the irregularity of the text and the relatively small amount of training data available. We found that users tend to use the terminology and formats of the website, resulting in reduced variability.

4.2 Role-based entity classification

Once an entity span has been identified, we must determine its functional role—i.e., whether it acts as a *keyEntity*, *newEntity*, *oldEntity*, or *noiseEntity* (as outlined in Section 2.1). We approach this problem as a classification task, where the extracted entities are transformed into instances to be further classified by a learner.

The features used for the learner are as follows. (a) 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. (b) The closest preceding preposition. (c) The presence or absence of a possessive marker after the entity. (d) An indication whether the entity is part of a determined NP.

The experimental results for the important classes are shown in Table 2, in the column marked "5CV". We used here an SVM learner with a linear kernel [Joachims, 2001]. We show results for each class separately, and in addition to F1 performance for each category, we also show error rate. The "Default Error" is the error obtained by always guessing the most frequent class.

Entity Role	F1/Error $5CV$	$5CV_{USR}$	$5CV_{REQ}$	Default Error
keyEntity	87.0/11.5	83.5/14.4	84.0/14.3	44.2
newEntity	88.8/ 7.5	85.0/10.6	83.4/10.7	34.4
oldEntity	81.0/ 2.5	81.3 /2.5	76.4/ 3.0	6.7

Table 2: Role-based entity classification results

The results for the role determination are almost-surprisingly good, considering the difficult, linguistic nature of this role assignment task. The set of features suggested here is small and simple, and yet very informative, supporting effective learning of roles even for semi-ungrammatical texts.

4.3 Target relation classification

To determine the target relation, we used the same SVM learner. The input features to the classifier are a "bag-of-words" representation of a request, as well as the entity types included in the request (for example, presence of a "phone number" entity in a request indicates a "people" relation, in our database schema.). Results are shown in Table 3 in the "5CV" column. As shown by these results, the task of relation determination is relatively straight-forward, provided sufficient training data.

Target Relation	F1/Error $5CV$	$5CV_{USR}$	$5CV_{REQ}$	Def. Error
people	99.7 / 0.3	99.3 / 0.8	97.3/ 3.4	38.7
budget	100.0 / 0.0	99.2 / 1.6	78.8 / 3.6	10.0
events	99.6 / 0.2	97.4 / 1.1	97.8 / 1.0	22.7
sponsors	100.0 / 0.0	98.6 / 0.2	98.6 / 0.2	6.0

Table 3: Target relation classification results

4.4 Request type 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*. This means that most request types can be determined algorithmically from the set of entity roles found in a request.

The primary need for a request-type classifier is to distinguish between *deleteValue* and *deleteTuple* requests. These types of requests are often syntactically quite similar. Consider for instance the requests "delete the extension for Dan Smith" and "delete the entry for Dan Smith". The first is a *deleteValue* for a phone number, and the second is a *deleteTuple* request. The action verb ("delete") and the included entities, however, are identical. To distinguish the two request-types, it is necessary to determine the direct object of the verb "delete"—which is difficult, since shallow parsing is inaccurate on this very noisy corpus—or else to construct features that are correlated with the direct object of the verb.

Thus, we used the following as features. (a) The counts of *keyEntities*, *oldEntities*, and *newEntities* in a request. (b) The action verbs appearing in a 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. (d) Nouns from the previous step that also appear in a dictionary of 12 common attribute names (e.g., "phone", "extension", "room", "office", etc).

The results are shown in Table 4. With these features, one can distinguish between these request types quite accurately.

Request Type	F1/Error $5CV$	$5CV_{USR}$	$5CV_{REQ}$	Def. Error
deleteTuple	93.1 / 2.4	92.6 / 2.6	74.7 / 9.2	18.0
deleteValue	82.9 / 3.1	86.0 / 2.4	57.5 / 6.0	9.1

Table 4: Request type classification results

4.5 Target attribute classification

The classification of requests by target attributes is very similar to request type classification, 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 do not specify an *oldEntity* value.

Here in fact we learn a vocabulary for attributes names. A simple bag-of-words feature works quite well for this task, as is shown by the results in Table 5 in the "5CV" column. The vocabulary used in the corpus to describe each attribute is fairly small: e.g., phone is usually described as "phone", "line" or "extension". Perhaps this is because users tend to use the terminology of the website, or because the relevant vocabularies are limited by nature.

Request Type	F1/Error $5CV$	$5CV_{USR}$	$5CV_{REQ}$	Def. Error
personal name	77.3 / 2.8	66.7 / 4.2	17.5 / 7.6	7.0
phone#	92.7 / 1.0	92.9 / 1.0	8.2 / 7.3	9.1
room#	87.0 / 2.9	91.5 / 1.0	56.9 / 9.1	18.0
publication	79.6 / 3.7	81.2 / 2.1	44.8 / 5.2	9.1
photo	93.1 / 2.4	78.6 / 3.9	71.3 / 5.3	18.0
CV	82.9 / 3.1	86.5 / 0.8	- / -	9.1
amount	93.1 / 2.4	92.5 / 1.0	92.7 / 1.0	18.0

Table 5: Attribute classification results

5 Robustness Issues

One practically important question is how robust this automated webmaster is to changes in the distribution of users and/or requests. To investigate such questions, one can use a different sampling strategy in performing cross-validation. For instance, to determine how robust the system is to queries from new users, we grouped all the examples generated by each subject into a single set, and then performed a cross-validation constrained so that no set was split between training and test. In other words, in every test fold, all of the example requests were from subjects that had not contributed to the training set. This split thus estimates performance of a system that is used for a very large pool of users. Cross validation by user results are given in the results tables, in the columns marked as "USR".

In the corpus, users usually have some personal stylistic quirks—for instance, a user might consistently give dates, names etc. in a particular format. Thus one would expect that performance with this sort of split will be worse than performance with the default uniform splits. As can be seen from the results, the F1 for most NER task drops only slightly, and is above 80 for all entity types. Slight drops in performance are also seen on two of the three entity-role tasks, and noticible drops are seen on two of the seven attribute-classification tasks (person name and photo). Overall, performance seems to be affected only slightly in this setting.

Similarly, to determine how robust the system is to future requests that are quite different from requests encountered during training, we grouped together examples for the same request type (including all requests generated from a particular pictorial task), and then again performed a cross-validation constrained so that no set was split between training and test. In this scenario, all of the example requests in every test fold are for tasks that were not encountered in the training set. The results of this split are given in the columns titled as "REQ".

To summarize the results, the loss in performance for NER problems is moderate, but larger than that seen when splitting by users. Entity-role classification drops off only slightly, and performance for target-relation classification also remains excellent for most relations. However, performance for request-type classification does drop off noticibly. This drop is almost certainly due to lack of appropriate training data: there are only a handful of tasks updating the "budget" relation, and also only a relatively small number of tasks requiring request-type classification. Similarly, the task of classification by attribute name is practically infeasible for some attribute types in this settings, due to the small number of attribute names mentions in the corpus. However, provided that the system is given sufficient training data for the relevant relation and attribute, it should perform well on different requests.

6 Overall Evaluation

In this section, we complement the component-level evaluations with an evaluation of the entire end-to-end process. We executed the tasks in the following order: NER is run for each entity type; then, roles of the extracted entities are assigned; finally, relation and request types are assigned. Note that the noisy predicted entities (i.e., entities extracted by a NER model) were used as input to the entity-role classifier, as well as to the relation to the request-type classifiers. Here we used VPHMMs and hand-coded rules for extraction, and a non-sequential multi-class voted perceptron [Freund and Schapire, 1998] for classification.[2]

From the user's perspective, it is interesting to note what percentage of the requests can be successfully processed at a message level, at different levels of automation. In the experiments, 79.2% of the messages got both their relation and request type classified correctly. In these cases the user would have received the correct form, with some entries filled out incorrectly. In more than half of the cases (53.4%), the user would have received the correct form, with all entities correctly extracted, but with some entity roles mislabeled. In 39.5% of the messages, the automatic processing encountered no errors at all.

Note that in the end-to-end scenario errors from the entity recognition phase are propagated to role classification task. Also, in order for a message to be considered fully correct, assignments must be accurate for each one of the multiple entities included in this message. That is, many correct decisions must be made for perfect performance per request. Overall, we find these results to be very promising, considering the limited size of our corpus.

7 Related work

Lockerd *et. al* [2003] propose an automated Webmaster called "Mr. Web" which has a similar email-based interface.

[2]The voted perceptron is another margin-based classifier. For implementation reasons, it was more convenient to use in these experiments than an SVM, although its performance was generally quite not as good.

They manually analyzed 325 update requests to assess their linguistic regularity, but do not describe any algorithm for processing the requests.

Our system addresses a fairly general natural-language processing task: learning to understand database updates. As such it might be compared to other systems that use learning in NLP. Previous NLP systems have generally either performed deep semantic analysis using hand-coded grammars 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 use learning to perform a complete 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 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. A second difference is that our learning method uses primarily 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. 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]).

8 Conclusions

We have described and experimentally evaluated a scheme for processing email requests for certain website facutal updates using a sequence of entity recognition and classification tasks. We showed that the noisy informal email text can be successfully processed applying a learning approach, using relatively small sets of syntactic features. Experimental results show that also with limited amount of data, the system reaches a promising rate of 40% of messages processed perfectly. We expect this rate to improve as the examples set grows. Further, human-subject experiments have also shown that partially correct results are useful in settings described here [Tomasic *et al.*, 2004]. Thus the work in this paper is a realistic evaluation of components of an efficient, adaptive, automatic webmaster assistant.

Open questions that remain to be resolved by future research include relaxing the restriction that each request concerns the update of a single tuple per email and evaluating more complex entity types for the entity recognition component. Improving on entity recognition will both enable expansion of system coverage, as well as boost its overall performance.

Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA), or the Department of Interior-National Business Center (DOI-NBC).

References

[Brill, 1995] Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 1995.

[Collins, 2002] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *EMNLP*, 2002.

[Fillmore *et al.*, 2000] C. J. Fillmore, F. C. Baker, and H. Sato. The framenet database and software tools. In *LREC*, 2000.

[Freund and Schapire, 1998] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. In *Computational Learning Theory*, 1998.

[Gildea and Jurafsky, 2002] D. Gildea and D. Jurafsky. Automated labeling of semantic roles. *Computational Linguistics*, 2002.

[Joachims, 2001] Thorsten Joachims. A statistical learning model of text classification with support vector machines. In *SIGIR*, 2001.

[Lockerd *et al.*, 2003] Andrea Lockerd, Huy Pham, Taly Sharon, and Ted Selker. Mr.web: An automated interactive webmaster. In *CHI*, 2003.

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

[Miller *et al.*, 1996] S. Miller, D D. Stallard, R. Bobrow, and R. Schwartz. A fully statistical approach to natural language interfaces. In *ACL*, 1996.

[Mooney, 2004] Ray Mooney. Learning semantic parsers: An important but under-studied problem. In *Working notes of the AAAI spring symposium on language learning*, 2004.

[Tomasic *et al.*, 2004] Anthony Tomasic, William Cohen, Susan Fussell, John Zimmerman, Marina Kobayashi, Einat Minkov, Nathan Halstead, Ravi Mosur, and Jason Hum. Learning to navigate web forms. In *IIWEB*, 2004.

[Zelle and Mooney, 1996] J. M. Zelle and R. J. Mooney. Learning database queries using inductive logic programming. In *AAAI*, 1996.