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 Technology Institute
Carnegie Mellon University
einat@cs.cmu.edu

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

ABSTRACT

In many organizations, users submit requests to update the organizational website via email to a human webmaster. In this paper, we propose an intelligent system that can process certain website update requests semi-automatically. In particular, we describe a system that can analyze requests to update the factual content of individual tuples in a database-backed website, using a particular scheme for decomposing request-understanding into a sequence of entity recognition and text classification tasks. Each of these tasks can be solved using existing learning methods. Using a corpus generated by human-subject experiments, we experimentally evaluate the components of this system, as well as various combinations of these components. We also present experimental results on the robustness of the system. In particular, we present results predicting how the system will perform on request types not seen in training; how it will perform on user-specific language usage not seen in training; and how it will perform in the absence of features specific to the database schema of the website.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; I.2.6 [**Artificial Intelligence**]: Learning

Keywords

Learning, information extraction, named entity recognition, sequential learning

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, 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 that 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.

WWW2005, May 10–14, 2005, Chiba, Japan.

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 2105

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

One possible solution to this problem is a database-backed website, with a database of information that can be updated by any user. 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 their 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 specific factual change to the database on which the web site is based.

In this paper, we will explore this "webmaster support task" in detail. Our proposed solution is 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 [15].

This paper focuses on the natural-language processing part of this system. We will first describe a scheme for decomposing request-understanding into a sequence of entity recognition and text classification tasks. We next describe the corpus of requests that is used to evaluate performance on these subtasks. We then present experimental results on performance for all of the subtasks. As will be shown, performance on these subtasks is surprisingly good, despite

the fact that the update requests we experiment with use unusual and sometimes ungrammatical language (as illustrated in Figure 1). We also present experimental results on the robustness of the system. In particular, we present results predicting how the system will perform on request types not seen in training; and how it will perform on user-specific language usage not seen in training; and how it will perform in the absence of features specific to the database schema of the website. Finally, we evaluate combinations of these components, 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. 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 focus here on messages that request a factual update to the underlying database. Requests that are not of this form 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: 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 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).* The next part 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* (in the figure, marked with a superscript N) if it is a value not currently in the database, which the user wants 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 expects to be replaced with a *newEntity*. As an example, in the request "please change Freddy Smith's phone number from 555-1111 to 555-1234", "555-1111" is an *oldEntity* and "555-1234" is a *newEntity*. (d) Entities unrelated to the execution of the request are considered to be *noiseEntities*. In the figure, they have no superscript marking. An example is the name "Martha" in the request "please delete Freddy Smith's phone number - thanx Martha".

  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.* Since each request concerns a single tuple, it necessarily also concerns a single relation. 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 often completely determined (as we show below, in Section 2.2). 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 these 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. Note that the classification is only neccesary for a subset of requests.

  In this paper, we will *not* consider another case in which requests can be underspecified: when a relation contains two or more attributes of the same type, where "type" is defined by the output of the entity recognizer. For instance, if the entity recognizer identifies phone numbers, and the "person" relation contains both office phone number and cell phone number as attributes, then the analysis of "change William's mobile nmber to 412-555-1234" will be "change $[\textbf{William}]^K_{\text{person}}$'s mobile nmber to $[\textbf{412-555-1234}]^N_{\text{phone}}$". This is ambiguous, since the new phone number entity could be stored as either a office phone number or a cell phone number. We ignore this case simply because our experimental corpus contains no examples of such ambiguity—it could be handled straightforwardly by using an additional entity classifier.

For pedagogical reasons, we have described these steps as if they are taken in the fixed order given above. 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.

2.2 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 type. Given these constraints, the steps described above are sufficient to map a request to a database update. Figure 3 summarizes the algorithm for doing this.

	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.

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

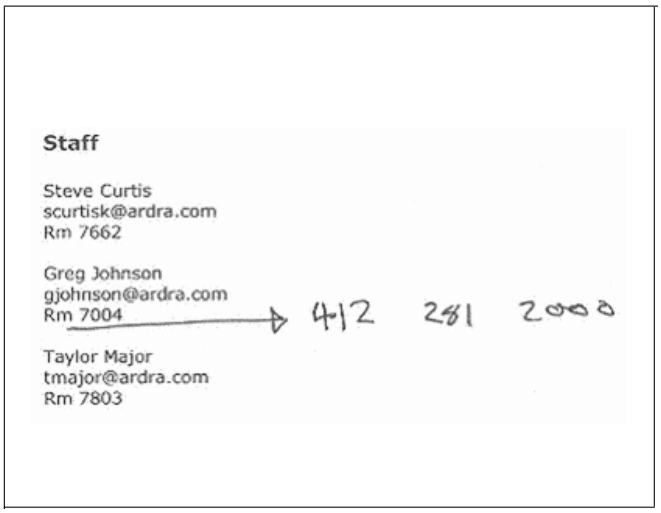

Figure 4: A pictorial description of an update task

The analysis produces an proposed conceptual update to the user's view of the database. (In general, this single conceptual update must be converted to a transaction for the underlying database, this conversion may be non-trivial; these issues are discussed elsewhere [15].) Given this update, the system selects an appropriate form, fills in all known values. and then presents the form to the user for correction or verification.

Even if the user (or webmaster) needs to correct some errors in the analysis, the update process is faster, on average, than manually changing a database (as has been experimentally verified in other human-subject experiments [15]). However, it is clearly preferable for the analysis to be as accurate as possible. Below we will experimentally evaluate the different steps of the analysis process. First, however, we will describe the corpus used in the analysis.

3. THE EXPERIMENTAL CORPUS

To evaluate performance of the individual steps of the analysis, it is necessary to use a corpus of user requests. 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. Figure 4 shows an example of the pictorial description shown to users—a possible request that might be associated with this might be "add 412-281-2000 as Greg Johnson's phone #, please." In response to the user's request, the agent returned a *preview* of the updated page, and also *pre-filled 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.

The advantage of this procedure is that it provides a corpus of real user-generated update requests, suitable for experimental study of NLP-related issues. However, the procedure is relatively expensive, and hence our corpus is rela-

tively small, containing a total of only 617 example requests.

To simplify the procedure for experimenting with human subjects, the same 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 4 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.

By design, the requests in the corpus are restricted: in each case, the intended request is a factual update concerning a single tuple in the database. The corpus thus contains no instances of requests that affect multiple database tuples, like "add these phone extensions to the site: 2213 - for William C., 2275 - Einat M., and 2248 - Anthony T.", or "change the office number of all LTI master's students directly funded by the RADAR project to A201". These sort of requests may be addressed by future research. The corpus does contain some requests that are not factual updates (such as, changing font format).

The text itself is quite noisy. Typos are frequent. Also, as is typical of informal text like email, the messages in our corpus are often ungrammatical, and often use capitalization patterns inconsistently. As a consequence, standard shallow-NLP tools such as POS tagging and NP chunking (being constructed for formal text such as newswire) are somewhat unreliable. We annotated the text with a version of Brill's POS tagger [1] and a hand-coded NP-chunker which was tuned for email (using a different corpus). In the experiments, however, we rely mainly on alternative features that exploit syntactic properties of a message.

4. EXPERIMENTS

We use the 617 request messages in our corpus to evaluate how well can every component of the analysis procedure be learned. Below we describe in detail the experimental setting and results for each system 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 natural-language processing task. We evaluated NER 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 of extraction methods.

We experimented with three approaches to entity extraction: a rule-based approach, in which hand-coded rules are used to recognize entities; and two learning-based approaches. The rule language we used is based on cascaded finite state machines. The two learning algorithms are VPHMMs (a method for discriminatively training hidden Markov models using a voted-perceptron algorithm [2]) and CRFs, or conditional random fields [7, 14]. VPHMMs are an example of a margin-based learning approach to entity recognition; CRFs are an example of a probabilistic approach. The implementations used here are the ones provided in the Minorthird package[12]. Performance is evaluated by F1-measure[1], and 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.) Table 1 shows results for entity recognition on the corpus.

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

(a) Rules

	Base features		Tuned features	
Type	*VPHMM*	*CRF*	*VPHMM*	*CRF*
Time	87.7	87.9	91.2	95.1
Date	88.5	76.5	94.4	95.3
Amount	89.7	94.1	93.1	**95.1**
Phone	87.3	86.1	**94.2**	92.8
Room#	81.9	76.0	90.4	**92.2**
Person	80.9	74.2	**90.3**	89.7

(b) Learning

Table 1: Entity recognition results: F1 measures using 5CV

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 some of the entities. We evaluated the rules on the main corpus, which was used also 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. The column labeled "Validation" shows performance on the 96-element "validation" set. (Unfortunately the linguistic entity types were somewhat different in the later round of experiments, and 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 more complex entity types. The table shows F1-measure performance on unseen examples, as estimated using 5-fold cross validation (5CV). Here NER was reduced to the problem of sequentially classifying each token as either "inside" or "outside" the entity type to be extracted; the VPHMM was executed for 10 epochs, and the CRF optimization method was likewise limited to making 10 passes over the data. (Preliminary experiments suggested that more iterations or more complex tagging schemes do not substantially improve performance for this task.)

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

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

of feature, labeled *tuned features* in the table, is comprised of the base features plus some additional, entity-type specific features, 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 names. The most complex feature-tuning was done for personal names, and these features were tuned on a separate corpus [11].

Overall, the level of performance for extraction is very encouraging, especially considering the informal nature of the text and the relatively small amount of training data available. For every entity type, better than 90% F1 measure is obtained by the best extractor (shown in boldface in the table), and five of the seven types can be extracted with an F1 of more than 94%.

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", "be" etc. (b) The closest preceding preposition. (c) The presence or absence of a possessive marker after the entity. (d) The closest preceding word w which is either a preposition, an action verb, or a determiner. Feature (d) is intended to detect whether the entity is part of a determined NP.

The experimental results for the important classes are shown in Table 2. We use here a maximum entropy learner, as well as an SVM learner with a linear kernel [6]. The SVM learner is representative of the margin-based approach to learning, and the maximum entropy learner is representative of the probabilistic approach. 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 SVM	F1/Error $MaxEnt$	Default Error
keyEntity	**87.0** / 11.5	85.5 / 13.2	44.2
newEntity	**88.8** / 7.5	88.2 / 8.5	34.4
oldEntity	**81.0** / 2.5	79.5 / 3.1	6.7

Table 2: Role-based entity classification results: F1 measures and percent error

The results for the role determination are overall quite promising, indicating that the features used are quite informative. We note that as yet, we are not using one possibly important feature—similarity to an entity in the existing database. One would expect that this would be a good indicator of the *oldEntity* role.

4.3 Target relation classification

To determine the target relation, we used the same learners, and a bag-of-words representation of a request for this task. The results are shown in Table 3. Even this simple representation for requests performs quite well, especially for the more frequently updated relations.

We also explored using the types of previously-identified entities as features for this task; for example, presence of a "phone number" entity in a request indicates a "people" relation, in our database schema. This expanded feature set shows a further improvement in performance. Note, however, that these results are based on using the true values for the entity extractors, rather than predicted values from a necessarily imperfect entity recognizer. The results for this augmented settings appear in the column marked with an asterisk.

Target Relation	F1/Error SVM	F1/Error $MaxEnt$	F1/Error SVM^*	Def. Error
people	**98.7** / 1.6	98.3 / 2.1	99.7 / 0.3	38.7
budget	95.8 / 0.8	**98.4** / 0.3	100.0 / 0.0	10.0
events	**98.2** / 8.1	97.2 / 1.3	99.6 / 0.2	22.7
sponsors	86.6 / 1.5	**88.9** / 1.3	100.0 / 0.0	6.0

Table 3: Target relation classification results: F1 measures and percent error

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, to distinguish *deleteTuple* and *deleteValue*, 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.

4.5 Target attribute classification

Request Type	F1/Error SVM	$MaxEnt$	Def. Error
deleteTuple	**93.1** / 2.4	92.7 / 2.8	18.0
deleteValue	**82.9** / 3.1	72.3 / 0.6	9.1

Table 4: Request type classification 5-CV results: F1 measures and percent error

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 contain no *oldEntity*.

A simple bag-of-words feature works quite well for this task, as is shown by the results in Table 5. This is somewhat surprising, since there are relatively few positive examples of each of these concepts. In the corpus, however, the vocabulary used to describe each attribute is fairly small: e.g., phone is usually described as "phone", "line" or "extension". Perhaps this is because user requests refer to an existing website, and users tend to use the terminology of the website.

Request Type	F1/Error SVM	$MaxEnt$	Def. Error
personal name	**77.3** / 2.8	76.7 / 3.2	7.0
phone#	**92.7** / 1.0	91.8 / 1.1	9.1
room#	87.0 / 2.9	**87.3** / 3.2	18.0
publication	79.6 / 3.7	**82.8** / 2.4	9.1
photo	**93.1** / 2.4	84.1 / 3.2	18.0
CV	82.9 / 3.1	**84.2** / 1.0	9.1
amount	93.1 / 2.4	**96.4** / 0.5	18.0

Table 5: Attribute classification 5-CV results: F1 measures and percent error

5. ROBUSTNESS ISSUES

5.1 Performance for new users and novel requests

Above, we presented a particular decomposition of the natural-language processing problem associated with a webmaster support task. We showed that this NLP problem can be broken down into a a cascade of classification and extraction tasks, and also showed that each subtask can be solved reliably using existing learning methods. All our evaluations were conducted on a corpus constructed by performing human-subject experiments. These human-subject experiments involved approximately 20 users, who were asked to generate natural-language requests corresponding to approximately 30 different database updates.

One practically important question is how robust such a system 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 that was 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 train/test split thus estimates performance of a system that is used for a very large pool of users—a pool so large that it is unlikely to see multiple requests from the same user.

In the corpus, users usually have some personal stylistic quirks—for instance, a user might consistently give dates in a particular format. Thus one would expect that performance with this sort of split will be worse than performance with the default splits. The degree to which performance is actually changed on various representative subproblems shown in Table 6. The results presented are for VPHMMs (for NER tasks) and linear-kernel SVMs (for classification tasks).

Linguistic Type	Base features $5CV$	$5CV_{usr}$	Tuned features $5CV$	$5CV_{usr}$
Time	87.7	78.0	91.2	88.2
Date	88.5	89.4	94.4	95.8
Amount	89.7	90.2	93.1	93.1
Phone	87.3	87.3	94.2	92.4
Room#	81.9	76.8	90.4	87.1
Person	80.9	72.1	90.3	83.6

(a) Entity recognition: F1 measure

Entity Role	F1/Error $5CV$	$5CV_{req}$	Def. Error
keyEntity	87.0 / 11.5	83.5 / 14.4	44.2
newEntity	88.8 / 7.5	85.0 / 10.6	34.4
oldEntity	81.0 / 2.5	81.3 / 2.5	6.7

(b) Entity-role classification: F1 measure and error rate

Target Attribute	F1/Error $5CV$	$5CV_{usr}$	Def. Error
personal name	77.3 / 2.8	66.7 / 4.2	7.0
phone#	92.7 / 1.0	92.9 / 1.0	9.1
room#	87.0 / 2.9	91.5 / 1.9	18.0
publication	79.6 / 3.7	81.2 / 2.1	9.1
photo	93.1 / 2.4	78.6 / 3.9	18.0
CV	82.9 / 3.1	86.5 / 0.8	9.1
amount	93.1 / 2.4	92.5 / 1.0	18.0

(c) Attribute classification: F1 measure and error rate

Table 6: Estimated performance with a large pool of users

Although the performance on extracting personal names drops from an F1 of 90.3 to an F1 of 83.3, the F1 every other NER task drops only slightly, and most F1-measures remain in the upper 80's to mid 90's. 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, the experimental corpus was limited in that it contained only about 30 different *request species*. Here we define a *request species* to include all requests generated from a particular task image, such as that shown in Figure 4. Recall that specific entity names in the requests were manually replaced with alternatives in the corpora we use; thus requests of the same "species" are isomorphic except for the

specific entities that appear in them. An example of two requests of the same species might be "add the phone number 412-281-2000 for greg johnson, please" or "Please note, g. jonson's phone number is missing. It should be 281-2000".

To determine how robust the system is to requests that are quite different from requests encountered during training, we grouped together examples for the same request species, 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. Again, one would expect that performance would be lower in this more difficult condition, which simulates a scenario in which many different varieties of requests are encountered, and hence every request has a completely novel structure. This scenario probably overestimates the difficulty encountered by a real system.

	Tuned features		
Type	$5CV$	$5CV_{req}$	$5CV_{usr}$
Time	91.2	93.9	88.2
Date	94.4	88.9	95.8
Amount	93.1	85.4	93.1
Phone	94.2	82.3	92.4
Room#	90.4	83.0	87.1
Person	90.3	88.3	83.6

(a) Named entity recognition

Entity Role	F1/Error	
	$5CV$	$5CV_{req}$
keyEntity	87.0 / 11.5	84.0 / 14.3
newEntity	88.8 / 7.5	83.4 / 10.7
oldEntity	81.0 / 2.5	76.4 / 3.0

(b) Entity-role classification

Target Relation	F1/Error			Def. Error
	$5CV$	$5CV_{req}$	$5CV^*_{req}$	
people	98.7 / 1.6	95.0 / 6.3	97.3 / 3.4	38.7
budget	95.8 / 0.8	70.1 / 4.7	78.8 / 3.6	10.0
events	98.2 / 8.1	85.8 / 5.7	97.8 / 1.0	22.7
sponsors	86.6 / 1.5	77.4 / 2.3	98.6 / 0.2	6.0

(c) Target-relation classification

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

(d) Request-type classification

Table 7: Performance with a very diverse pool of requests.

Table 7 shows the result of performing this type of cross-validation for four representative tasks: NER, entity role classification, target relation classification, and request type classification. For target relation classification, the column marked $5CV^*_{req}$ uses entity-roles as well as words as features.

To summarize the results, the loss in performance for NER problems is moderate, but larger than that seen when the pool of users is large. Entity-role classification drops off only slightly, and performance for target-relation classification also remains excellent for three of the four relations considered. However, performance for request-type classification does drop off noticibly. This drop in performance 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.

5.2 Changes to the database schema

A final issue to discuss is the robustness of the system to changes is the database schema. Typically a website will change over time, perhaps by the addition of new database relations (e.g., "presentations") or new attributes (e.g., "instant messenger id"). When this happens, the distribution of requests will also change. One advantage of the architecture presented here is that it is often straightforward to collect additional training data from user interactions with the system.

Specifically, training data can be automatically collected for any classification decision made by the system. Automatically collecting training data suitable for training an extraction system is somewhat more difficult, and may be addressed by future research; for now we note that if the database schema changes, but the set of primitive entity types remains the same, then data can be automatically collected for all necessary phases of the analysis. The question of whether the system as a whole is robust to changes in the database schema thus reduces to a question of the robustness of the features used in learning: to what extent are these features dependent on the database schema used?

Request Type	F1/Error		Def. Error
	$5CV$	$5CV_{req}$	
all features			
deleteTuple	93.1 / 2.4	74.7 / 9.2	18.0
deleteValue	82.9 / 3.1	57.5 / 6.0	9.1
without dict			
deleteTuple	90.1 / 3.6	62.6 / 13.9	18.0
deleteValue	81.4 / 3.4	44.7 / 7.6	9.1
without nouns or dict			
deleteTuple	86.8 / 4.7	58.8 / 15.2	18.0
deleteValue	79.6 / 3.7	42.7 / 8.3	9.1

Table 8: Predicting request type without schema-dependent features.

In the experiments above, we have generally attempted to avoid use of features that are database schema-dependent. The exception to this is the dictionary of attributes used in predicting request type. Table 8 shows the results obtained on request-type classification when this schema-dependent information is not available. Performance is still acceptable. However, the schema-independent system is considerably less robust to novel requests (as is shown by the column labeled $5CV_{req}$).

5.3 Discussion

These experiments show that the final system will be more robust to changes associated with a new user's stylistic quirks than to changes in languages associated with new and different types of requests. This suggests that if future human-subject experiments are used to collect data, it would be more desirable to have fewer subjects generate more requests each. More specifically, it would be useful to have

more *deleteValue* and *deleteTuple* requests in the training set, since this is the current performance bottleneck when the pool of requests becomes more diverse.

6. OVERALL EVALUATION

In sections 4 and 5 we examined performance on each subtask of the request-analysis process. In this section, we will complement these component-level evaluations with evaluations of larger sections of the request-analysis process, including an evaluation of the entire end-to-end process.

As noted above, the ordering of the subtasks can be varied. We adopted the following procedure. First, NER is run for each entity type. Next, the roles of the extracted entities are assigned. Next, relation and request types are assigned. In evaluating this process, five-fold cross-validation was used. In each fold, predicted entities (i.e., entities extracted by a NER model learned on the same training fold), rather than true entities, were used as input to the entity-role classifier. Similarly, NER-predicted entities were used as input to the relation classifier, and the request-type classifier.

We made two slight simplifications of the process: requests containing less-frequent entity types (like event titles) were discarded; and we did not evaluate the target-attribute classification step, which is only necessary for a handful of requests. We used VPHMMs and hand-coded rules for extraction, and a non-sequential multi-class voted perceptron [4] for classification.[2]

NER (all)	Relation	Request Type	Entity Role	% Messages Correct
X				84.5
O	X			98.9
O		X		80.1
O			X	66.8

(a) Evaluating Single Components

O	X	X		79.2
X	X	X		53.4
X	X	X	X	39.5

(b) Evaluating Multiple Components

Table 9: Percent of messages correctly processed. An "X" indicates a subtask being evaluated, and an "O" indicates the output of a subtask is used as a feature for another task.

The results are shown in Table 9. The first part of the table shows the percentage of messages which are completely correct with respect to a single component. Here an "X" indicates a subtask that is being evaluated directly, and an "O" indicates that a subtask that is used indirectly, to construct features for some task being evaluated. For instance, the first line indicates that on 84.5% of the messages, *all* named entities were extracted correctly. In almost 99% of the messages, the correct relation was determined; note that this decision is based partly on the (noisy) extracted entities. The most difficult step seems to be assigning functional roles to the extracted entities. We note that entity-role classification is closely related to semantic role analysis, which is known to be a difficult NLP problem [3, 5].

The bottom part of the table shows the percentage of messages that are completely correct with respect the composition of several subtasks. Nearly 80% of the messages have 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, the user would have received the correct form, with all entities correctly extracted, but with some entity roles mislabeled. Almost 40% of the messages were processed perfectly, and would require no further intervention from the user.

7. RELATED WORK

Lockerd *et. al* [8] 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.

Our system addresses a fairly general natural-language processing task: learning to understand single-tuple 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 [16], which learns to parse database queries into a meaning representation language, and the work by Miller *et. al* [10] on using a combination of generative models to extract facts from text. Work in learning such "semantic parsers" is surveyed and motivated elsewhere [13].

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 [3, 5], and earlier work on case-role assignment (*e.g.*, [9]).

From NLP perspective, this work is an example of pro-

[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.

cessing of noisy "informal" text. Most such text cannot be reliably analyzed using off-the-shelf NLP techniques such as POS tagging and parsing; however, outputs from these systems can still be useful as inputs to a learner.

8. 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. Human-subject experiments have also shown that partially correct results are useful in settings described here [15]. Thus the work in this paper is a realistic evaluation of components of an efficient, adaptive, automatic webmaster assistant.

Many open questions remain to be resolved by future research. One issue is relaxing the restriction that each request concerns the update of a single tuple per email. Another issue is automatically collecting training data and re-training extractors.

Acknowledgements

We thank Richard Wang for work on the POS tagger and NP-chunker used in these experiments. We also wish to thank John Zimmerman, Susan Fussell, Ellen Ayoob, Aaron Spaulding, Marina Kobayashi and Kyle Cunningham for providing us with the experimental corpus.

9. REFERENCES

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

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

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

[4] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. In *Computational Learning Theory*, pages 209–217, 1998.

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

[6] T. Joachims. A statistical learning model of text classification with support vector machines. In *Proceedings of SIGIR-01, 24th ACM International Conference on Research and Development in Information Retrieval*, 2001.

[7] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning (ICML-2001)*, Williams, MA, 2001.

[8] A. Lockerd, H. Pham, T. Sharon, and T. Selker. Mr.web: An automated interactive webmaster. In *Extended abstracts on Human factors in Computer Systems (CHI'03)*, Ft. Lauderdale, Florida, April 2003.

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

[10] S. Miller, D. D. Stallard, R. Bobrow, R., and Schwartz. 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, 1996.

[11] E. Minkov, R. Wang, and W. W. Cohen. Extracting personal names from emails: Applying named entity recognition to informal text. In preparation, draft available from http://wcohen.com., 2004.

[12] Minorthird: Methods for identifying names and ontological relations in text using heuristics for inducing regularities from data. http://minorthird.sourceforge.net, 2004.

[13] R. Mooney. 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 2004. AAAI Press.

[14] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *In Proceedings of HLT-NAACL*, 2003.

[15] A. Tomasic, W. Cohen, S. Fussell, J. Zimmerman, M. Kobayashi, E. Minkov, N. Halstead, R. Mosur, and J. Hum. Learning to navigate web forms. In *Workshop on Information Integration on the Web (IIWEB-2004)*, 2004. Toronto, Canada.

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