Email Event Extraction and Interpretation

Keith Ito
kito@cs.stanford.edu
CS224N Final Project

June 5, 2003

Abstract

Email frequently contains date and time information about events. Extracting and interpreting it could be useful to both applications and the end user. However, such information is often expressed in natural language, making this task more difficult than simple pattern matching. In this paper, I present a complete system using HMMs and semantic grammars which achieves both high precision and recall for this information extraction task.

1 Introduction

1.1 The problem

One of the primary uses of email is for notification of meetings, deadlines, and other time-sensitive events. However, conventional email clients leave it to the user to manually determine the date and time information from each message.

Furthermore, events and the corresponding email are often temporally separated; one may receive an email regarding an event weeks in advance. It then becomes the user's responsibility to remember this information until it is needed. This is a difficult and error-prone task that could benefit from automation.

In addition, the use of calendar software such as Microsoft Outlook or IBM Lotus Notes is widespread, particularly in the corporate environment. Users could benefit greatly from a means of automatically converting email messages into a format supported by the calendar software.

This project consists of a system that uses IE techniques, in particular, Hidden Markov Models (HMMs) and a semantic grammar, to parse date and time information from natural language in email messages and convert it into a machine-understandable representation. Once messages are annotated with this information, they can easily be browsed, queried, or imported automatically into calendar applications, although actually doing so is beyond the scope of this project.

The focus for this project is date and time information because of its prevalence in email messages, and because date and time taken together are members of a totally ordered set that is easy to reason about in isolation. Other fields, such as locations, that one may wish to extract from email simply encode naming information, and cannot be reasoned about without significant out of band knowledge regarding the entity to which the name refers.

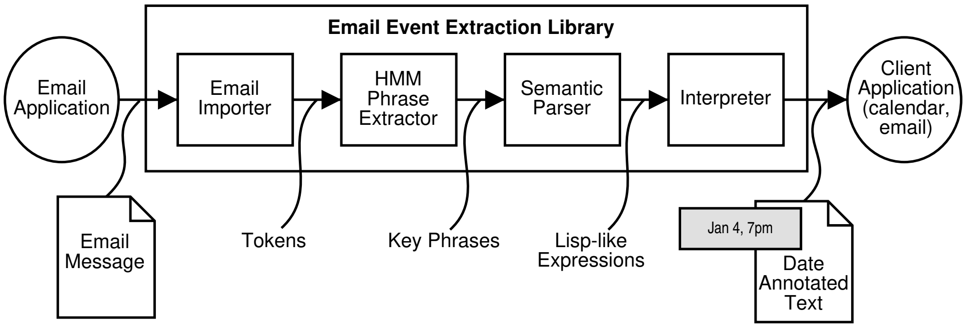

Figure 1: The data flow model for this project. The email client feeds messages into the event extraction library, which uses an HMM to extract relevant phrases. These are fed to a semantic parser to generate a Lisp-like representation which is evaluated to determine the date references in the email.

1.2 Related Work

To date, much of the work in information extraction from email has focused on tagging portions of the message with various annotations without regard to interpreting the tagged data. An example is the extraction of speaker, location, and time information from seminar announcements [5]. Work has also been done in learning templates to populate database entries from short email-like pieces of text [4]. Others have also explored extracting date information from email using pattern matching [1]. However, none of these approaches have examined the use of semantic parsing to extract date information from natural language and reason about it.

Ray and Craven [11] combine the use of HMMs and a simple parser for information extraction. In their case, they are extracting *n-ary* relations from biology journal articles. Their approach parses the sentence first to tag words with part of speech, then uses an augmented HMM to extract information, using the cross product of base HMM states with parts of speech as the state space of the new HMM. Their algorithm is essentially the reverse of this project as they use the parser as a preprocessor for the HMM, while I use the HMM as a preprocessor for a parser.

As Almgren and Berglund point out, many commercial products integrate email and calendaring. However, such products make no efforts to integrate their mail and calendar functionality [1]. Other attempts at integrating natural language and calendars include Mitchell's learning personal assistant [10]. This approach restricted the domain to the user answering questions posed by the system, thus placing the initiative with the system and simplifying the problem. In addition, it did not attempt to do sophisticated parsing of date information.

2 Techniques

2.1 Overview

This project can be divided into four modules, as illustrated in Figure 1. The first module imports and tokenizes the email. This involves no significant natural language processing and is discussed only in the implementation details (Section 4.1). The second module extracts the text containing time and date information from the message using a Hidden

Markov Model. The third module takes these fragments of text and interprets them using a semantic grammar. A modified CKY parser is used to convert the phrases into a Lisp-like intermediate form. The final module takes this semantic representation and interprets it based on its context in the email in order to generate a date/time annotation.

2.2 Hidden Markov Model

Upon entering the system, each message is processed so that every word in the subject and body is represented as a separate token. This token list is passed to an HMM to find the tokens that are part of date or time phrases.

The model architecture is a fully-connected two-state HMM. The two states are conceptually labeled "DATE" and "BACKGROUND". The DATE state models phrases that contain date and time information, while BACKGROUND models everything else.

The HMM parameters are estimated on a training set in which portions containing date information have been previously marked with the DATE annotation by hand. Maximum likelihood estimation is used, with smoothing performed using Lidstone's Law:

Emission probabilities, $e_k(b)$ are estimated as:

$$e_k(b) = \frac{E_k(b) + \lambda}{\sum_{b'} E_k(b') + |V|}$$

where $E_k(b)$ is the number of times that token b is observed to be emitted in state k in the training set. λ is a constant, $0 < \lambda < 1$, which can be used to vary the degree of discounting, and $|V|$ is the number of distinct word types in the training set.

Transition probabilities, a_{kl} are estimated as:

$$a_{kl} = \frac{A_{kl} + \lambda}{\sum_{l'} A_{kl} + |S|}$$

where A_{kl} is the number of times that the transition from state k to state l is observed in the training data. $|S|$ is the number of states in the HMM, in this case, two.

The Viterbi algorithm is used to compute the sequence of states that is most likely for a given email. This is an inductive algorithm where the probability, δ_l, of the most likely path ending in state l at time $t+1$ with emission x_{t+1} is given by:

$$\delta_l(t+1) = e_l(x_{t+1}) \max_k \delta_k(t) a_{kl}$$

The Viterbi algorithm can be implemented such that it stores backtrace pointers at every step in the computation to the cell from which it made the most probable transition. This path is read back to produce the most likely assignment of output tokens to states.

Each contiguous region of tokens emitted in the DATE state is stored into a token list and parsed by the semantic grammar as described below.

2.3 Semantic Grammar

The output from the HMM is a list of zero or more phrases (token sequences) that are likely to contain date information based on the model's parameters. Other IE approaches have often stopped here, assuming that once the date information is tagged, the tool's job is

Rules	
S: $merge(\alpha)(\beta)$	$\rightarrow$ DATE: α TIME: β
S: $merge(\beta)(\alpha)$	$\rightarrow$ TIME: α DATE: β
DATE: $merge(\alpha)(\beta)$	$\rightarrow$ MD: α YEAR: β
DATE: $merge((year\ \alpha)\ \beta)$	$\rightarrow$ ADJ: α MD: β
MD: $merge(\alpha)(\beta)$	$\rightarrow$ MONTH: α DAY: β
MONTH: (α)	$\rightarrow$ MONTHNAME: α
DAY: $(dayofmonth\ \alpha)$	$\rightarrow$ NUMBER1-31: α
YEAR: $(year\ \alpha)$	$\rightarrow$ NUMBER1-3000: α
ADJ: $(number\ +1)$	$\rightarrow$ next
MONTHNAME: $(month\ 6)$	$\rightarrow$ june
NUMBER1-31: $(number\ 1)$	$\rightarrow$ 1
NUMBER1-31: $(number\ 2)$	$\rightarrow$ 2
NUMBER1-3000: $(number\ 2003)$	$\rightarrow$ 2003

Table 1: A simplified subset of the semantic grammar. Following each symbol is an expression that is used to build up the semantic representation of the parse as the parse tree is created. When the first rule is invoked, for example, the semantic representation of S symbol is the function *merge* applied to the representation of DATE and TIME symbols.

done - the extraction of actual meaning from the date is left for either human intervention or pattern matching.

Such behavior would be sufficient if all date information was expressed in a limited number of formats. However, this does not tend to be the case in email. For example, sentences one might typically find in a message are: "The meeting is tomorrow at 3 in the afternoon." or "I'll be in town next Thursday." Clearly, a pattern matching scheme would be insufficient to capture the semantics of such expressions. Thus, an approach that attempts to model the language generating such phrases is necessary.

The problem of creating such a semantic grammar [8] for the recognition and interpretation of dates and times is a nontrivial one, yet it is still simple enough that a relatively small grammar can capture nearly all common usages.

I designed an augmented semantic grammar with rules specifically designed for processing dates and times. A simplified subset of the rules in the grammar is presented in Table 2.3. The complete grammar is located in the MeeConfig.properties file that is included in the project. The complete grammar contains 36 nonterminal and 128 terminal productions[1].

One observation about the grammar is that there are only two productions from the start symbol, even in the complete version. As a result, it is impossible to represent many valid dates with parse trees rooted in the start symbol. However, since partial parses are legal and have valid semantic representations, it is not necessary for a parse to be rooted in the start symbol in order to derive semantic information from it.

2.4 CKY Parser

Parsing the grammar described in the previous section is accomplished using a CKY parser. This bottom-up dynamic programming technique begins with all parses of length 1, then

[1]In addition to these are the implicit productions for numbers which are handled separately by the lexicon

generates parses of incrementally increasing sizes by connecting subparses.

Conventional CKY parsing has the restriction that the grammar must be in Chomsky Normal Form (CNF). However, due to my implementation of lexical lookup of numbers, it would have been difficult to perform the unit production elimination step necessary to convert the grammar into CNF. Instead, I wrote my parser implementation to handle unit productions by performing a recursive unit production expansion step following the generation of a cell in the WFST. All elements in the cell are checked to see if they are the sole nonterminal on the right hand side of a rule in the grammar, and if so, the left hand side of that rule is also added to the cell.

For efficiency, the parser keeps a hash index over the right-hand side of each grammar production such that the reverse lookups used in parsing can be done in constant time.

The CKY algorithm makes it trivial to find all legal subparses, even in sentences which are not fully parsable by the grammar. This is essential to this project since the phrases returned by the HMM often contain a few words of "junk" at either end that is not date or time related information and would not be recognized if there was a requirement that all symbols be part of the parse.

The semantic meaning of the parse is built up in parallel with the parse tree, producing a Lisp-like expression representing the semantics of each subparse. When parsing is complete, the parser outputs both a parse tree and the semantic representation of the sentence.

To give the reader a feel for this output, a few examples are presented. The first example draws only on the subset of the grammar listed in Table 2.3. The remaining two examples use the full semantic grammar.

Example Phrase 1: *June 2 2003*

Parse:

DATE[MD[MONTH[MONTHNAME] DAY[NUMBER1-31]] YEAR[NUMBER1-3000]]

Semantic Expression:

(merge (merge ((month 6)) ((dayofmonth 2))) ((year 2003)))

Example Phrase 2: *next Sept. 11*

Parse:

DATE[ADJ MD[MONTH[MONTH[MONTHNAME] PUNCT] DAY[NUMBER1-31]]]

Semantic Expression:

merge (year (+1)) (merge (((month 9))) ((dayofmonth 11))))

Example Phrase 3: *extraneous Saturday at six in the evening text*

Parse:

S[DATE[NN-DATE[DAYOFWEEK]] TIME[P TIME[UNQ-TIME[HOUR[NUMBER0-23]] AMPM[P AMPM-N[DET AMPM-N]]]]]

Semantic Expression:

(merge (((dayofweek 7))) ((merge (((hour 6))) (((hour +12)))))))

Note that in Example 3, the phrase contains some extraneous text surrounding both sides of the date expression. Since the parser returns the longest subparse in the table, this is not a problem, and the correct expression is derived from the phrase regardless.

2.5 Parse selection

The output of the CKY parser described in the previous section is a list of all possible subparses of the input phrase, along with their semantic representations. Before this data can be sent to the interpreter, it is necessary to trim it down to contain only the non-redundant parses that provide the strongest information about the phrase.

One potential way of resolving this would be to take the parse corresponding to the longest token list from the parser and sending it to the interpreter. However, this approach does not take into account the varying likelihoods of parses based on their content. In addition, it will not be able to send multiple parses to the interpreter as may be necessary if two separate date phrases are merged into one due to the low interstate transition probabilities in the HMM. Such a situation can also arise when a single date phrase is split into two subparses that may not connect due to the presence of extraneous text not generated by the grammar.

An alternative is to take the parse or parses that maximizes the log odds ratio of being in the target (DATE) state. This parse can be selected by taking:

$$\arg\max_{\{x_1,\ldots,x_k\}\in 2^S} \sum_{i=1}^{k}\sum_{j=1}^{n} \log \frac{P(x_{ij}|DATE)}{P(x_{ij}|BACKGROUND)}$$

where S is the set of terminal strings corresponding to subparses of the phrase, and x_{ij} is the jth symbol in the ith string. $x_1, \ldots, x_k$ are also subject to the constraint that they do not overlap. The parse selector performs a greedy search for the set of sentences with the maximum odds ratio and sends these as a list to the interpreter[2].

2.6 Interpreter

To evaluate the expressions generated by the previous step, I implemented a simplified Lisp interpreter. It simply splits the top level input into a function and arguments, then recursively calls itself on each argument, stopping when the argument is atomic. After the arguments have been evaluated, the function is applied to the arguments and the result is returned. This is equivalent to doing a depth-first traversal of the tree represented by the expression.

There are two types of non-function entities: numbers and dates. Numbers can be either absolute values such as "4", or relative values such as "+1". There are only two classes of functions, date generators and merges. Table 2.6 describes these two classes and their effects.

The result of the interpretation step is a single date expression representing the phrase. This date expression is either an absolute date or a differential from the *base* date when the email was sent. In the latter case, the differential is applied to the *base* date to derive an absolute date which is then assigned as an annotation to the email. Multiple dates may be assigned to messages in which the HMM or parse selector identified multiple phrases. In this case, a stack is used to collect date expressions that have already been seen, and non-conflicting dates are merged with previously seen dates before being assigned to the message. The rationale behind doing this is that dates appearing early in the message may be referenced by dates later in the message. For example, the date of an event may appear

[2]The actual quantity computed by the implementation is the minimum of the odds ratio for being not in the target state, which is equivalent

Function	Arguments	Effects
$merge$	date $d1$, date $d2$	for each $field$, if $d1.field$ or $d2.field$ is a relative value, add $d1.field$ and $d2.field$. otherwise, replace $d1.field$ with $d2.field$
$year$, $month$, $dayofmonth$, $dayofweek$, day, $hour$, $minute$	number n	create a new date with the field corresponding to the function name set to n

Table 2: Interpreter functions

early in the message, with specific times appearing much later. If the new date expression conflicts with the element on top of the stack, then elements are popped off until one that does not conflict (or the empty stack) is reached.

Interpreting expressions in a separate module rather than combining the interpreter with the parser allows the semantic grammar rules to contain expressions. For example, the rule:

DATE:$merge(year\ \alpha)(\beta) \rightarrow$ ADJ:α MONTH:β

expresses that adjectives modifying month actually apply to the year. For example, in the phrase "next January", the adjective "next" corresponds to increasing the year by 1, not increasing the month, which would be the result of a straightforward merge.

3 Results

The system described in the previous section takes as input a set of email messages and associates a list of zero or more dates with each message.

To evaluate system performance, a number of email messages were annotated by hand and the system response was compared to the human-generated annotations. Hand annotating emails is a time-consuming task, and due to time constraints on this project, I was only able to annotate approximately 140 messages. These were split into a training set of 100 messages and a test set of approximately 40 messages.

This data was derived from personal email that I received during two distinct time periods. The training set contains mail received from December 2002 through April 2003, and the test set contains email received from March 2002 through November 2002. Messages are spread across these time periods to minimize the number of references to the same events in order to provide a wider variety of training and test cases to the system. In addition, email that was clearly spam, as well as email with an HTML body was removed from both the test and training sets. Although these steps were conducted by hand, they could have just as easily been done using machine classification techniques.

3.1 HMM evaluation

Although the Hidden Markov Model only produces output which is used internally to the system, it is still interesting to evaluate its performance on the test data. The HMM was evaluated on a subset of 20 messages from the test set which had a total of 38 fragments

Hit Criterion	Precision	Recall	F-Score
(a) Exact	0.806	0.795	0.800
(b) Overlap	0.944	0.919	0.932
(c) Window	0.972	0.946	0.959
(d) Date Only	0.973	0.973	0.973

Table 3: Complete system results for the four match criteria described in Section 3.2.

of event information in them. A recall of 0.927 and precision of 0.322 was observed (F-score = 0.478). While Freitag and McCallum were able to attain F-scores[3] as high as 0.991 for their HMM which recognized seminar times from announcements, the task here is not comparable since their HMM was trained on very specific date formats, a task which essentially reduces to pattern matching. A more comparable task from their experiments would be the recognition of speaker names, which also exhibit a short duration and variety of contexts. The F-score of 0.478 obtained here is comparable to their F-scores for such a task, which were around 0.5.

Still, the low precision score seen for this HMM is somewhat surprising, and may be due to the fact that dates often appear without any related context information, making it difficult to differentiate them from non-date phrases. Regardless of the cause, the poor precision is not a serious impediment since the HMM is used solely to generate input phrases to the parser which will further discard inputs that it is unable to recognize. The high recall, on the other hand, is essential to the success of the semantic parsing step since the only information it receives as input is from the HMM.

3.2 Complete system evaluation

The system was tested with the complete test set, consisting of 43 messages, all of which contained at least one date reference. Precision, recall, and F-score for the data set are reported in Table 3.2.

The actual precision and recall rates depend on the definition of a “hit” and a “miss”. Depending on the application using the date extraction system, this definition could vary. For example, a daily task list application might only care about having the correct date and not the time. A constraint visualizer might only care about the relative ordering of events, not their exact times. In order to address a wide variety of applications, precision and recall rates, as well as F-scores are reported for four different “hit” definitions: a) Exact match: the date and time must match the starting time of the event down to the minute; b) Overlap match: the date must match exactly, and the time must fall some time during the duration of the event if it is known, or within 2 hours if it is not; c) Window Match: the date must match exactly, and the time must fall within 2 hours on either side of the event time; d) Date Only Match: the date must match exactly, and there are no restrictions on the time.

Additionally, all scenarios except for the exact match consider a miss by exactly 12 hours due to missing an am/pm distinction to be a match.

As a baseline comparison, the date pattern matcher in [1] would have had precision of 1.0, recall of 0.095, and an F-score of 0.173 on the same data set with the most lenient

[3] F-score is the harmonic mean of the precision and recall, $\frac{2}{\frac{1}{P}+\frac{1}{R}}$

(Date Only) criterion[4].

These results show that the semantic grammar approach significantly outperforms pattern matchers for deriving meaning from date information. Furthermore, if the user or application is willing to tolerate an error of several hours on either side of the event, an F-score over 0.95 can be attained.

3.3 Issues

Several factors decreased the effectiveness of this system. They are described in this section.

3.3.1 Distinction between past and future

Many words used in describing dates can refer to the past or future depending on their context. For example, the word "Wednesday" can refer to a Wednesday in the current week, in a past week, or in a future week. While the semantic grammar used in this project is able to account for expressions such as "last Wednesday" or "next Wednesday," it does not attempt to use verb tense information to determine the correct sense when "Wednesday" is used in isolation, and instead assumes that it is referring to the Wednesday immediately following the timestamp of the message. Perhaps a more sophisticated semantic grammar would take tense information into account adjust the date calculation.

3.3.2 Separating events in nested discourses

Because the interpreter uses a simple stack to track the current date context, it can become confused when one event is nested in another. One test message in the data set can be paraphrased as, "Come to the show on Thursday because Friday's show is sold out. The show starts at 8:30." The author begins by providing date information about an event, diverges to discuss a different date, then returns to give the time information associated with the first date. The simple algorithm used for event separation incorrectly associates the time "8:30" with the date "Friday". It is unclear how to resolve this problem without a much deeper understanding of the semantics than the simple parser used for this project has.

3.3.3 Implied am/pm distinction

The system occasionally extracts the am/pm distinction in times incorrectly. This is largely due to the fact that the necessary information for determining this is often omitted. In the sentence, "I'll see you at 9 tomorrow!", it is left unspecified whether the "9" refers to 9:00 am or 9:00 pm. If the sentence is referring to a meeting at work, the meaning 9:00 am is implied. If, on the other hand, the sentence is referring to a party, 9:00 pm is implied. A human interpreter can usually immediately determine which is the case by looking at the context of the email and using logical reasoning to determine that, for example, email discusses a party, and parties are more likely to occur at night than in the morning, so the message must be implicitly referring to 9:00 pm.

While it may be possible to extract context information in an automated fashion, building a framework on which to reason about it is difficult, and can be impossible if the necessary context is communicated out of band.

[4]Since the pattern matcher only finds exact matches, it never generates false positives and thus has a precision of 1.0

3.3.4 Errors in spelling and consistency

Email is a relatively informal means of communication, and email messages tend to have more errors, both in terms of syntax and semantics than, say, the Wall Street Journal corpus. For example, in one test email, the author spelled tomorrow as "tommorow", which prevented the semantic grammar from properly recognizing it. In another message, the author referred to the same event as occurring on "Friday" and "Nov. 21", when November 21 of that year was, in fact, a Thursday. While running the data through a spell checker would mitigate errors of the first type, it would be more difficult to resolve inconsistencies of the second type. Perhaps the best action in this case is to actually return both dates and let the user sort out the inconsistency, which is what the system currently does.

4 Implementation Details

This section discusses some of the details of this project that were not necessary for understanding the algorithms, but are nonetheless important components in the process.

4.1 Extracting email messages

Nearly all email clients either store their messages in the mbox format [12] or have the capacity to export messages to the format. To be compatible with this popular format, the software for this project uses the GNU Classpathx Knife library [3], which contains a JavaMail provider for handling mbox mail stores. Since this project uses the generic JavaMail interface to the library, it should work with minor modifications with POP3, IMAP, and NNTP data stores, as well as any other format for which a JavaMail provider exists.

After messages are extracted from the email file, all headers are removed except for "Date" and "Subject". Date is used to parse out the date and time that the email was sent (the *base* date for the interpreter), and Subject is prepended to the message body. Message bodies are converted to lower case and tokenized, using whitespace and punctuation characters to separate tokens. For example, the phrase "Friday, April 3rd" would be converted into 5 tokens: "friday", ",", "april", "3", and "rd". In addition, the data was "cleaned up", by removing characters such as the leading ">" denoting quoted email text and stripping lines of the form "On *date*, *person* wrote," which is often used in a reply and is, for our purposes, irrelevant noise.

4.2 Tagging training data

Training data for the HMM step was generated by manually marking all date information from the training examples in bold text using a text editor. This file was then saved in the RTF format and run through a script to convert the RTF representation of the start and end of bold text to the format used by the HMM code.

4.3 Representation of semantic grammar

For ease of both specifying the grammar and implementing the interpreter, lambda calculus expressions were converted to a simplified representation in which the actual λ symbols are

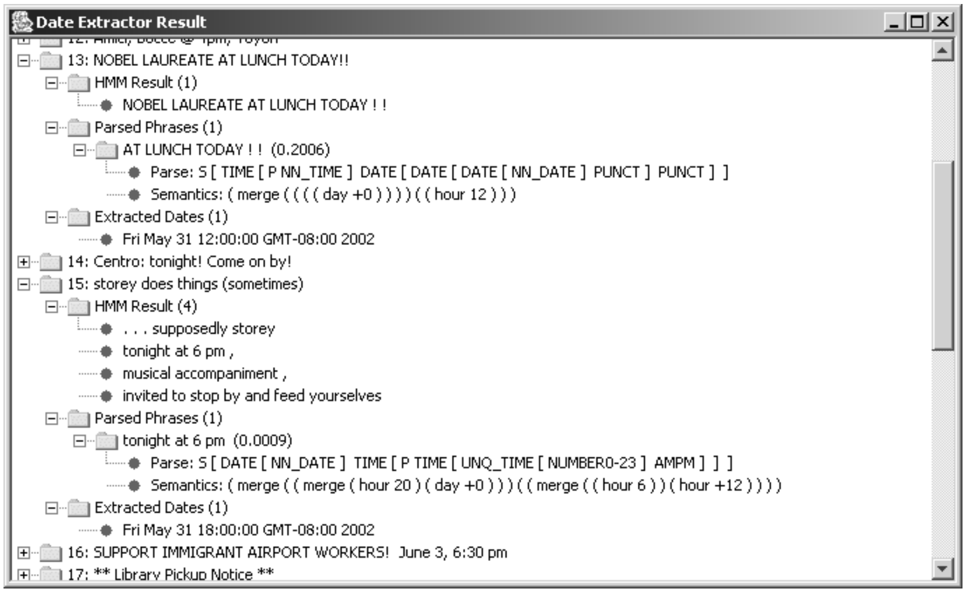

Figure 2: A screen shot of the results of extracting dates from a set of email messages. Two of the message results are expanded so that the HMM result, parses, semantics, and extracted dates are visible. This is not intended as an end user interface, but rather as a tool which demonstrates the system.

implied, and where the semantics of rules can map to function applications in addition to β reductions. For example, one rule in the grammar is:

S $\rightarrow$ DATE TIME | *merge* :0 :1

denoting that when this rule is applied, the function *merge* should be applied to the first and second symbols on the right side of the rule in that order. This is equivalent to the semantic grammar rule:

S: $merge(\alpha)(\beta) \rightarrow$ DATE: α TIME: β.

It is straightforward to map between the two representations. To convert from the standard format to the format used in this project, one simply needs to move the expression on the left hand side to the end, and replace the symbol corresponding to the first nonterminal on the right hand side with ":0" and the second nonterminal with ":1".

4.4 Demonstration application

In addition to implementing the core date extraction engine, I also wrote a simple demo application that displays the results. This application is not intended as an end user interface, but is instructive in showing the data flow at various steps in the date recognition process.

The application can be run using the "*demo*" script available with this project. It takes two command line arguments: the first is name of an mbox file containing the training data, and the second is the name of an mbox file containing the test data. After date extraction has taken place, the results are shown in a window. Messages can be expanded to show the results from the HMM recognizer, parse trees of the recognized phrases, semantic representations of the phrases, and the dates extracted from the message. A screenshot from this application with two of the message results expanded is shown in Figure 2.

5 Future Work

The HMM used in phrase recognition is incredibly simplistic. It has only two states and thus cannot model context information adequately. Perhaps implementing an HMM with pre and post context states as described in [5] would improve the HMM performance. In addition, their "shrinkage" methods of combining states for maximum likelihood estimation would probably be helpful for the models with additional context states.

My parser implementation also leaves much to be desired. A probabilistic parser could be used to model a more flexible set of grammatical rules, and could also be used for parse selection, rather than decoupling this from the parsing step as is currently done.

The semantic grammar used in this project was also written rather hastily, and there are some redundant rules and less elegant parse trees often arise due to poor grammar design. With additional time, it would be nice to try to improve this grammar and extend it to cover more expressions than it currently does.

To fully evaluate the system implemented in this project, it would be beneficial to generate a larger corpus of annotated training data. It could be the case that the sparsity of data used for the training in the experiments here was a limiting factor in performance.

Finally, if the end goal of this project is to produce something that is beneficial to ordinary end users, the UI becomes an important point. Numerous issues that are critical to this goal were not even considered in this project. For example, the user interface issues of how date-annotated email should be displayed to the end user, and what to do about the inevitable mistakes that the system will make are both important issues that need to be addressed prior to any sort of deployment.

6 Conclusions

In this project, I present a complete system for extracting and interpreting date and time information from natural language in email messages. By using an HMM to detect phrases that are then input to a semantic parser, this system exploits both the ability of HMMs to find general motifs and the capacity for semantic parsers to convert natural language into a logical representation.

The results, with F-scores exceeding 0.95 when the exact match requirements are relaxed, show that refinements of the methods presented here may be of significant value to users and applications wishing to obtain date and time information from email messages.

References

[1] Almgren, M. and J. Berglund. Information Extraction of Seminar Information. *CS224N Final Project, Spring 2000.* http://nlp.stanford.edu/coursese/cs224n/2000/berglund/report.pdf

[2] Amini, M., H. Zaragoza, P. Gallinari. Learning for Sequence Extraction Tasks. http://citeseer.nj.nec.com/411576.html.

[3] Burdess, Chris. Knife JavaMail library. GNU Classpathx Project. http://www.gnu.org/software/classpathx

[4] Califf, M.E. and R. J. Mooney. Relational Learning of Pattern-Match Rules for Information Extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*. Menlo Park: AAAI Press, 1998.

[5] Freitag, D. and A. K. McCallum. Information extraction with HMMs and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*. 1999.

[6] Manning, C. D. and H. Schutze. *Foundations of Statistical Natural Language Processing* Cambridge: The MIT Press, 1999.

[7] Manning, C. An Informal but Respectable Approach to Computational Semantics. CS224N/Lin237 Lecture Notes. May 23, 2000. Revised 2003.

[8] Manning, C. Tabular Parsing CS224N/Lin237 Lecture Notes. April 2003.

[9] McCallum, A., D. Freitag, F. Pereira. Maximum Entropy Markov Models for Information Extraction and Segmentation. In *Proc. 17th International Conf. on Machine Learning* 2000.

[10] Mitchell, T., R. Caruana, D. Freitag, J. McDermott, and D. Zabowski. Experience with a Learning Personal Agent. *Communications of the ACM*. July 1994.

[11] Ray, S. and M. Craven. Representing Sentence Structure in Hidden Markov Models for Information Extraction. In *Proceedings of the 17th International Join Conference on Artificial Intelligence*. 2001.

[12] Resnick, P. Internet Message Format. RFC 2822. April 2001.