CS224N: Final Project

Information Extraction of Seminar Information

Magnus Almgren
magnus.almgren@stanford.edu

Jenny Berglund
jenny.berglund@stanford.edu

Abstract

Email is used for many types of communications today, even for tasks it was not originally designed for. One such example is announcements of different types. In many environments, seminar announcements, meeting announcements and, in the case of graduate students, free-food announcement and parties, flood the inbox of users.

A significant amount of time is spent for each user to read the message, determine if he has seen it before, and then update his calendar. From personal experience, we note that many users just leave important announcements in the inbox and then miss them.

Some systems offer an integration of announcements sent by email into calendar software (notably Microsoft Outlook). However, this requires all users to run the same type of software which is difficult to enforce even within the same company.

In the following report we describe a complete system we have built, which will extract information from seminar announcements messages sent via email. It also has the capability of sending emails containing this extracted information, so that one each day can get a reminder of the next day's events.

The information extraction is done with both a pattern matching approach and a Hidden Markov Model.

1 Introduction

Any user in a company or in the academic environment can attest to the flood of email during a day. Emails are not only used to communicate, but also to announce meetings and other types of events.

We argue that email is not suited to the latter kind of messages. The user has to read the message and then manually update some sort of calendar, may it be a Palm, some type of software or an old paper-based calendar. This may be very time consuming.

However, as each seminar message may be sent several times, the situation is worse. A talk may be rescheduled, canceled, change rooms, change times. Many times there are also reminder messages.

Speaking from personal experience, most users do not seem to do any manual updating of a calendar. They simply leave the messages in the inbox, and trust that they will remember it.

As more and more communication will happen via email in the future, more and more events will be scheduled via email and it is more likely that a saved message will simply be forgotten among all other messages.

We propose an semi-automatic system to alleviate this problem. In Figure 1 we show an overview of the system.

Our system focuses on seminar announcements, but can easily be extended to other types of events.

As an email message arrives, an *Email Classifier* classifies it as being either a seminar announcement or not. This classifier is described in Section 4. If the email is considered to be a seminar announcement, another program, described in Section 5, tries to extract relevant information, such as date, time and location.

If the extraction is successful, the information is inserted into a web-based calendar, and a couple of lines describing what information was extracted are added to the email. This way, the user can check whether the extraction was correct, and otherwise manually update the calendar.

Another feature of the system is the ability to send *summarizing emails*, e.g. to each day at a certain time send an email containing the scheduled events for that day.

These aspects of the systems are discussed in Section 6 and 7.

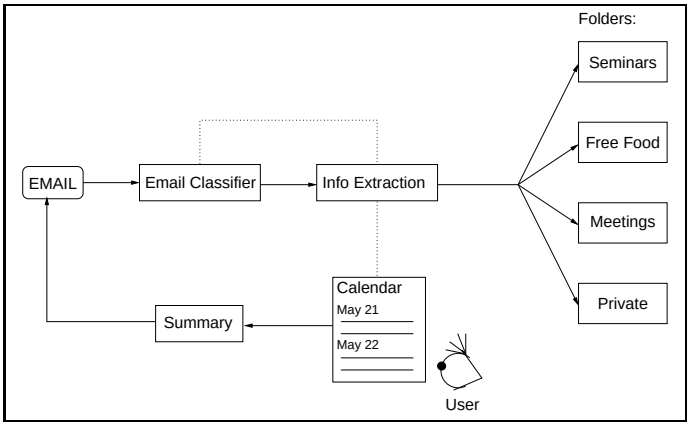

Figure 1: The complete information extraction system. Email is read by a classification system, and then fed to the correct information extraction model. The IR model extracts some type of information and updates a calendar. This calendar can be browsed by the user, and every day a special tool summarizes the events of the day.

2 Related Work

There have been other attemts at solving this problem. Below we will discuss a representative, yet not exhaustive, list of similar research and compare it with the current paper.

Some commercial systems have let their tools encompass both the emails, calendars and other tools. Two examples are Microsoft Outlook from Microsoft and Lotus Notes from IBM. Both systems allow a user to send *special types* of email messages, to set up meetings or announce talks. These messages can

offer buttons to accept or decline a meeting, and they may automatically insert a message into a user's calendar.

The systems are quite easy to use, but a major disadvantage is that everyone must use the same system. There is no standard between different vendors. As there are many email clients, this requirement is difficult to enforce, even within the same company[1].

One more problem with this approach is security. When one user can send an email that will update another user's calendar directly with commands in the mail there might be vulnerabilities to be exploited.

We also suggest a complete solution, but the parts can be added onto existing systems, processing the emails before the actual email clients. As the information extraction and classification is statistical in nature, and we only assume the standard mail delivery protocol this system should work on most systems.

Mitchell and Maes ([5] and [4]) separately looks at how one can automate scheduling of meetings. This is a very challenging task, and even though their results are impressing the systems cannot be fully trusted. Mitchell report that his system encounters many errors during semester shifts.

We do not try to set up meetings, but focus on presenting relevant information to the user, thus alleviating the task of keeping a calendar up-to-date. This is a somewhat easier task. Both approaches are valid and will play a major role in the future.

Other people have also looked explicitly at how one can extract information from emails. However, they have not accounted for the different types of emails as we do. In our system we can use different models based on the classification of the email. As such models will better fit the message we have a greater chance of extracting the correct information.

3 Description of Approach

We focus on the following points:

- Presenting information to the user in a friendly fashion
- Give the user direct feedback on the behavior of the system, by adding to the emails information about what additions have been made to the calendar.
- Use two statistical methods to classify emails: TF-IDF and Naive Bayes, which allows us to concentrate on the actual seminar messages.
- Use a Hidden Markov Model together with a knowledge-based pattern matching approach. The former can adapt and generalize, while the latter has a high accuracy.
- Learning of the Hidden Markov Model, but also of the knowledge-based approach[2].
- Concentration on important fields (time, location) as opposed to other fields (title, speaker). The former must be correct, because otherwise we cannot keep the information in a calendar, nor go to the speech. However, if the system fails on the title by not including all words we still get the idea.

4 Email Classifier

The first part of the system is the *email classifier*. We use a program already implemented by the authors. For a detailed description, see [3]. Here, we briefly discuss some of the core algorithms.

The classifier uses a combination of two algorithms: A text classification algorithm called TF-IDF, and a naive bayes model using certain features of an email message. The assumption is that many users sort their emails largely based on senders and recipients (which might for example be the case with mailing lists; the senders are different but the recipient is always the list), and those features would thus be much more important than just another word in the document. However, not all folders are necessarily based on such an approach, so it is also important to be able to recognize content based classes. Therefore, a combination of the two algorithms seems reasonable.

[1] One of the authors worked at IBM Research, but did not use the standard Lotus Notes. At one point he was accused of being rude because he had not accepted or declined a meeting. As it turned out, this was sent as a special message. The email was delivered with the information, and the author had sent a reply. However, this reply did not have the correct format and thus the attendance list of the meeting, automatically updated by Lotus Notes, failed to include the author.

[2] The HMM finds something which the pattern matching does not find, and can then ask the user if the information is correct. This is not automatic in the current version, but the user must manually add an entry to a text file.

4.1 TF-IDF

TF-IDF stands for Term Frequency-Inverse Document Frequency, and is a very simple text classifying algorithm that basically works by counting words. In our setting, a message $\mathcal{M}$ is represented by a word-frequency vector $F(\mathcal{M})$, where each component $F(\mathcal{M}, w)$ represents the number of times the word w occurs in the message. A folder is represented by a weighted word-frequency vector that is defined as follows:
The *fractional frequency* $FF(\mathcal{F}, w)$ of a word w in a folder $\mathcal{F}$ is the number of times w occurs in any message in $\mathcal{F}$, divided by the total number of words in $\mathcal{F}$. The *term frequency* is then $TF(\mathcal{F}, w) = \frac{FF(\mathcal{F},w)}{FF(\mathcal{A},w)}$, where $\mathcal{A}$ represents the set of all messages. The *document frequency* $DF(w)$ of a word is defined as the fraction of folders in which the word appears at least once. Finally, the weight is computed:

$$W(\mathcal{F}, w) = \frac{TF(\mathcal{F}, w)}{DF(w)^2}$$

These weights are then used to classify a new message, represented by a word vector $F(\mathcal{M}, w)$. For each folder, a similarity measure between the folder and the message is:

$$SIM(\mathcal{M}, \mathcal{F}) = \frac{\Sigma_{w \in \mathcal{M}} F(\mathcal{M}, w) W(\mathcal{F}, w)}{min(\Sigma_{w \in \mathcal{M}} F(\mathcal{M}, w), \Sigma_{w \in \mathcal{M}} W(\mathcal{F}, w))}$$

Thus, TF-IDF would choose to classify the message into the folder with the highest similarity value, but it can also suggest more than one folder, e.g. the three with highest similarity.

Despite it's simplicity, this text classification algorithm has some advantages. For example, since we are not using a stop-list (which would mean not counting certain particularly common and uninformative words), it is language independent. This is important since a user might receive messages in several languages.

4.2 Naive Bayes

A Naive Bayes network is then used to capture some specific features of emails.

The naive BN has these features:

- the email address of the sender
- the email address of the recipient
- the n most likely folders according to the TF-IDF classification.

This way the importance of the sender and recipient can be captured, and the content of the message still be used. The Naive BN will learn the weights for these parameters given a certain folder, and will thus allow both folders where messages are saved according to sender and folders centered around a certain topic, such as given seminars.

4.3 Resulting Scheme

First, the text classifier is trained, by counting the words and computing the weights. The results of this classifier are then used as features in a second classifier, the naive bayes network.

Thus, this is a second learning scheme, where the BN parameters, i.e. the probabilities of the features given the folders, are learned from our training data. The final classification is the output of this network, i.e. the most likely folder given the sender, the recipient and the n best folders according to content.

4.4 Changes to the existing system

We could not use the system described in [3] and Section 4 directly but we had to make a few changes to it.

The original system is designed to classify an email into one of many existing folders. It achieves a very high accuracy by giving a couple of suggestions that the user can choose from. In this case, we only want to know whether an email message is a seminar announcement or not, i.e. there are only two categories. On the other hand, we cannot give the user the opportunity to select between a number of folders, but have to trust our best guess.

We also had to make a little change to the implementation of the naive bayes algorithm, as it turned out to be not as robust as it should have been when training on several thousands of emails.

Finally, the original version of the classifier as built in [3] never saved any state, but rather retrained each time it was run. We added this funcitonality, so that the training could be separated from the actual classifying. This change was necessary because of the time it took to retrain the system each time it was run.

5 Information Extraction

Given an email that we know (or believe) to be a seminar announcement, we try to extract from it the following pieces of information:

- `DATE`
- `TIME` – the start time
- `LOCATION` – the room where the talk is to be held
- `TITLE` – the title of the talk
- `SPEAKER` – name
- `AFFILIATION` – of the speaker
- `TYPE` – the type of seminar, e.g. "SRI AI Seminar Series"

These fields might not all be present in each email, but a seminar announcement usually contains most of these fields.

We combine two ways of extracting the desired information, a pattern matching approach and a Hidden Markov Model, both described below.

5.1 Pre-filtering Step

Before feeding a tagged message to the HMM for training, or an untagged message to either the HMM or the pattern matcher for information extraction, it is first preprocessed by some filters.

We use an XML type of tagging, where a tag `<STATENAME>` marks the beginning of a state, and a tag `</STATENAME>` marks the end. To avoid confusion, we start by removing any tags of the same format that don't contain valid state names[3]. We also remove any `''>''` at the beginning of a line, which are commonly inserted when emails are forwarded, and remove some special characters, allowing only {`a-z A-Z 0-9 < > : , - _ @ /`} and whitespace. This is the state in which the email is fed into the pattern matcher.

For the Hidden Markov Model, some extra steps of prefiltering are made. The message is split to only have one word on each line, with a special symbol representing newline. Some changes are also made to the actual content of the message. Any occurring number (integer) is converted to one of the symbols *num0-12*, *num13-31* or *num*. Names of months or weekdays are converted into *month* and *day*, respectively, and expressions of the form

```
\d{1,2}:\d{1,2}(am|pm|AM|PM)?
```

(`perl` syntax) are converted into the symbol *time*. This makes it a lot easier for the model to find dates and times, since the vocabulary is drastically decreased and fewer events will be unseen. If any of these symbols are identified as the desired information, the actual words are retrieved from the original file.

5.2 Pattern Matching with Knowledge

Some of the fields we are trying to extract have a very distinct syntax, and are thus suited for a pattern matching approach. This is the case with both dates and times, and we also use a simple pattern matching technique to try to find the location, title and speaker.

[3] This might for example be HTML tags, which are quite common in emails.

5.2.1 Pattern Matching Dates

We allow the following formats (and variants):

- May 31, 1999 or 31 May
- May 1st, 1999
- 5/31/99, 05/31/99, 5/31, 05/31, 5/31/1999, 05/31/1999
- 5-31-99, 05-31-99, 5-31, 05-31, 5-31-1999, 05-31-1999

5.2.2 Patterns for Times

We have a simplistic view of times:

- match the numbers before am/pm
- match places where you have digits:digits

5.2.3 Pattern for Locations, Speaker and Title

These fields do not have a uniform format as the other fields described above. Here, we use the approach of looking for keywords. If we find a line starting with `room:`, `location:` or `place:`, we take the rest of that line to be the desired `LOCATION` field.

For locations, we also keep a list of already specified seminar rooms. If we don't find a keyword tag, we look through the message for rooms in that list. The first location of such an occurence is taken to be the room of the speech. We make the assumption that the important seminar information will be at the top of the email, and less important information is found later. By always taking the first occurence we also avoid the information in the signatures of the email.

For the fields `SPEAKER` and `TITLE`, we simply use the keyword approach.

5.3 Hidden Markov Model

To extract information not found by the pattern matcher we use a *Hidden Markov Model*. The model is trained on pre-tagged announcements.

The states of the HMM are the fields we are trying to extract, plus a state `OTHER`, representing anything else. The outputs are each word in the email.

5.3.1 Learning and calculating the probabilities

While training, we learn the transition probabilities (i.e. the probability of transferring to a certain state given the current state), and the emission probabilities (the probability of seeing a particular word given the state). This is done by going through all the tagged training announcements and counting the transitions between states, and the outputs in each state. This information is then later used to estimate the probabilities of certain transitions or emissions.

Since the data will be very sparse, i.e. most events will be unseen, especially emissions, *smoothing* is extremely important when estimating the probabilities. We use Lidstone's law for smoothing, and thus estimate:

- Emission probabilities (the probability of seeing word w while in state s):
  $P(w|s) = \frac{C(w,s)+\lambda}{C(s)+\lambda|V|}$
  $C(w,s)$ is the number of times word w is seen within state s, $C(s)$ is the total number of words generated by state s, λ is a smoothing parameter, which we chose to 0.01, and $|V|$ is the vocabulary size. In this case, the vocabulary size is unknown, and we use estimates that differ depending on the state, s.t. `OTHER` has the largest vocabulary size whereas `DATE` and `TIME` have smaller vocabularies.[4]
- Transition probabilities (the probability of transitioning to state s' from state s):
  $P(s'|s) = \frac{C(s',s)+\lambda}{C(s)+\lambda|V|}$
  Here, $C(s',s)$ is the number of seen transitions from s to s', and $C(s)$ is the total number of transitions from state s, which will be the same as the number of words emitted from that state[5]. The vocabulary size $|V|$ corresponds to the number of states.

[4]The vocabularies for DATE and TIME are significantly reduced by the symbol matching done in the prefiltering step, described in Section 5.1.

[5]A transition is made for each word, but will quite often be back to the same state

5.3.2 Computing the most likely path

For an untagged email, we then use the learned model and the *Viterbi* algorithm (see [1]) to compute the most likely state sequence given this particular text. If there are words that according to this most likely sequence belong to states other than `OTHER`, they are extracted as the information corresponding to that state.

5.4 Combining the two approaches

We let both the pattern matcher and the HMM try to extract information from each message. If the pattern matcher finds a certain field, we use that information, otherwise we use the information found by the Hidden Markov Model.

6 Browsing the Calendar

We wanted to include the functionality to easily browse the calendar, but not spend too much time on a fancy interface. For this reason, we let the calendar file be stored in HTML. We have a few advantages with this system:

- Platform-independent representation as almost all platforms have a browser.
- Presentation of the calendar can be done more effectively with formatting tags in HTML compared to text.
- Simplicity of HTML makes it easy for a user to go in and change some information if necessary. As each seminar email contains the information extracted, the user can make sure that the model worked correctly and change the information if necessary[6].

Clearly, we can choose to represent the calendar file in text format, but then we cannot highlight certain items. We can use a more powerful language, such as postscript, but then the user cannot manually change the file. HTML gives enough expression while being very easy.

6.1 Generating Summaries

The summaries where generated by a very simple Perl program. This program was scheduled with `Cron`[7]to run at 6am every day. It parses the calendar file in HTML. The days activities are summarized in a text message that is mailed to the user.

7 Connecting the Pieces

The following discussion has little to do with the statistical approaches to classification and extraction. However, we feel a need to describe how we connected the pieces.

This discussion focus on the system the authors had access to. It is simpler to install it on a regular unix system.

New mail is fetched with the program `fetchmail`. In the file `.fetchmail` we specify that we want to sort the mails with `procmail`. It seemed this was the only way to run our filter. We then gave `procmail` directions to start our filter, `seminarFilter` and give the whole mail to this program.

This program will first run the classifier, and then launch the extraction programs if the classifier says it is a seminar message. For all other messages, there will be no extraction. However, it is easy to extend the system with several types of extraction models and based on the classification we launch the correct program.

If the extraction system manages to extract a valid time and date, we will insert the entry into the calendar. The program `seminarFilter` also adds information to the actual email, so the user immediately can compare the information saved in the calendar with the information in the email.

[6]We envision a future version, where the user can simply *reply* to an email with the correct information, to make the changes.
[7]This is a daemon run on unix systems, so that a user can schedule jobs to be run regularly at different times.

8 Extending the Knowledge Base using the Markov Model

Seminars are usually held iat the same time and in the same room each week. For a certain user, one can assume that the rooms the seminars will be held in will not change very much. For this reason, the pattern matching technique looks for actual room descriptions in the email.

However, a new seminar, or a rescheduling of a meeting, may be in a new room not recognized by the system. Thus, we would like the knowledge-based approach to integrate new rooms as they are mentioned in the emails.

When the Markov Model extracts a location, and the pattern matching approach fails, the user has the option of extending the knowledge-base by adding the room (if it is actually a room). In the current system, the user must manually add the room to a text file, but in a future version the system should be able to update itself.

9 Implementation

The email classifier and the Hidden Markov Model are both implemented in C++, whereas the prefiltering of the messages and the pattern matching is done in perl.

10 Experiments and Results

10.1 Email Classifier

The interested reader should refer to [3] for a more complete discussion of the accuracy of the email classifier. We made a very limited experiment for this report, as the preconditions are different[8].

We let the classifier train on 2915 emails classified as `OTHERS` and 100 emails classified as `Seminars`. There are two reasons the numbers are so different:

1. We did not have that many seminar messages, but quite a few other emails.
2. We envision the system to actually see many more of the former category than the latter and the training should reflect this fact.

We then let the system classify another 67 seminar messages, and 72 other messages.

The system managed to correctly classify 64 of the 67 Seminar messages as `Seminars`, thus having an accuracy of 95.5%. However, it classified 15 of the 72 non-seminar messages as `Seminars` (20.5%).

Clearly, it is very important that all seminar messages goes through the system. As shown above, 95.5% is an impressing number. This leaves little extra work for the user.

That some of the other messages were wrongly classified as `Seminars` is not as severe. Firstly, they will probably not contain the correct information and if the extraction system cannot find a valid time and date the email will be ignored. Secondly, it is better to insert an invalid entry (to be ignored) than to miss important seminars.

The `OTHERS` group constists of many different types of emails, and some may look like seminars. Their format is not well defined, and some are emails that are only received one time. Thus, it is not surprising that a few of these messages were wrongly classified.

10.2 The Information Extraction

For testing the information extraction performance of the Hidden Markov Model and the Pattern Matcher, we used a set of 95 tagged seminar announcements. These were taken from the authors' mailboxes, and tagged by hand, since there was no existing data set containing the fields we are interested in. Of these, 75 were used for training (of the HMM), and 20 for testing. The assessment of the correctness of the extracted information was done manually. For the results reported in Tables 1 and 2, "Totally correct" means the extracted information was exactly the desired informaion, "Nearly correct" means that the extracted information was very close to what we were looking for, typically having one or a few additional words, or possibly missing a few unsignificant words, and "Has information" means that the extracted information contains the desired information, but possibly also a lot of unrelated text.

[8] We only use a binary classification, and we use quite a large number of emails.

Field	Totally correct	Nearly correct	Has information
DATE	95 %	95 %	95 %
TIME	75 %	80 %	80 %
TITLE	35 %	70 %	85 %
SPEAKER	45 %	70 %	80 %
LOCATION	15 %	15 %	75 %
AFFILIATION	25 %	35 %	80 %
TYPE	55 %	85 %	100 %

Table 1: The results achieved when using the HMM for information extraction. For an explanation of the criteria used, see Section 10.2.

Field	Totally correct	Nearly correct
DATE	80 %	85 %
TIME	80 %	80 %
TITLE	15 %	35 %
SPEAKER	0 %	35 %
LOCATION	30 %	30 %

Table 2: The results achieved when using only the pattern matcher for information extraction. For an explanation of the criteria used, see Section 10.2.

10.2.1 The Hidden Markov Model

We trained the model on the 75 messages in the training set, and then tested it using the 20 test emails. The ability of the model to extract the different fields is reported in Table 1.

10.2.2 The Pattern Matcher

The pattern matcher was tested on the same 20 test emails as the HMM.[9] The results achieved are shown in Table 2.

10.2.3 Combined

Table 3 shows the results achieved when the HMM and the pattern matcher were combined and tested on the 20 test emails.

10.3 Performance of the whole system

We were unable to try the performance of the whole system, because it is based on *received emails* where the header of the email must be correct. It is thus not possible to send "fake" seminar announcements, since they are likely to be classified as non-seminar emails because of the sender and recipient. However, we can use seminar announcements that we already have and run them through both the classifier and the information extraction program, and get a fairly accurate picture of the system's performance.

11 Discussion

As can be seen in the results tables, the information extraction system has a high performance for all the fields. `DATE` and `TIME` are the most critical fields, since the entry in the calendar will depend highly on them, and here we are exactly correct in 85% of the test cases. The other fields are not as sensitive to a little bit of noise (as long as the information is there, it doesn't really matter if the extracted title is a couple of words too long), and for all these fields we find the desired information (and possibly other irrelevant information too) in at least 80% of the test cases, and sometimes in all the test cases.

[9] Naturally, no training was done of the pattern matcher.

Field	Totally correct	Nearly correct	Has information
DATE	85 %	90 %	90 %
TIME	85 %	100 %	100 %
TITLE	50 %	75 %	90 %
SPEAKER	20 %	85 %	90 %
LOCATION	45 %	45 %	85 %
AFFILIATION	25 %	35 %	80 %
TYPE	55 %	85 %	100 %

Table 3: The results achieved when combining the HMM and the pattern matcher for information extraction. For an explanation of the criteria used, see Section 10.2.

The HMM has remarkably good performance on the `DATE` field, and also very good performance on the `TIME` field. This is probably greatly due to the symbol mapping described in Section 5.1.

The `DATE` field performance actually goes down when the pattern matching information is also considered. Since the performance of the HMM is so good on this particular field, one could consider reversing the "priority order" for this field, so that the pattern matching result doesn't override the HMM.

One reason for the good performance of the information extraction system is that it is highly specialized to deal with seminar announcements. Specifically, the HMM is trained using only such messages, and the patterns in the pattern matcher are handcrafted.

However, one might also be interested in extracting information from other types of emails, maybe parties or meeting announcements. Then, many of the fields used here would be totally irrelevant. Our system is easily extendable to deal with such a situation. The email classifier can be retrained to handle these extra categories, and thus we can start a different information extraction program depending on what kind of email it is. Therefore, we would be able to have different HMMs and pattern matchers for each such category, using only the fields relevant to that particular category and trained only on such messages, and would be able to achieve a much higher accuracy in the information extraction part than if trying to develop one model to deal with several different categories of announcements.

12 Conclusion

In this report, we have described a complete system for managing event information, and showed the results when using the system on seminar announcements. The system achieves a high accuracy at finding the interesting email messages and extracting relevant information, and also allows for easy detection of any mistakes.

A Programs and Files Used

The *seminarSystem* uses quite a lot of files and programs installed in different parts. Below we give a short description of the major programs.

- `fetchmailrc`
  This should be stored in the home directory and makes fetchmail (Stanford mail tool) run the filter `procmail`.
- `procmail`
  This file is stored in the users home directory, and contains filtering rules for procmail. Our version has two rules, one that starts the *SeminarSystem*, and another which stores the resulting email into the home folder. The last rule is not necessary on more regular mail systems.
- `seminarFilter.pl`
  This program is the main script for the seminar system and it should be stored in the users home directory.. It will be started by `procmail`, and given the mail via `STDIN`. It then starts the email classifier, and depending on the classification it starts the correct knowledge-based program and the correct Hidden Markov Model. If these succeed to extract information, the program also starts `insertEntry.pl` to insert the message into the HTML calendar file.

- `insertEntry.pl`
  This program updates the calendar file written in HTML. Currently, it cannot handle duplicates but this is quite easy functionality to add.
- `summary.pl`
  This program is started as a `cron` job every day, and reads through the calendar file. It will summarize all the events for the day in an email that is sent to the user. This program requires the user to add an entry to the cronfile with the program `crontab`.
- `rules.pl`
  This program implements the rule-based system in Perl. Each new rule must be added explicitly by coding. However, the program has a small ability to learn. For example, the user can add *rooms* to a text file to extend the functionality. The plan is that when the Hidden Markov Model extracts information not in the knowledge base, the user is presented with the option of extending the knowledge base by adding the new instance.
- `tfidf`
  The Email classifier uses some extra files to store its state. To use it, you first need to train the program on a set of mails and then it is ready to classify new mails. The current version is restarted for each new mail that arrives. A more efficient version would be to have it running in the background, and just send a signal to it to classify a new mail.
- `Viterbi`
  The Hidden Markov Model trying to extract the information. This program also uses some files to store state. These are not explicitly described here, since they are automatically generated.
- `Email/parseFolders.pl`
  This is a Perl script that can read a standard mailbox in UNIX. It will read the mail folder, and write each message to a separate file. This way we can let both the email classification program and the hidden markov model train on the emails. We also have a variant of this program to understand the mail folders from Eudora.
- `HMM/parseFolders.pl`
  This is a third variant of the above script. It will take a normal Perl message, and only keep the *body* of the message as this is the only information used for the extraction system.
- `mergeData.pl`
  This program compares the information found by both the expert system, and the hidden markov model and chooses what should be the final version. The rules in the current version is quite simple and we trust the expert system more than the hidden markov model.
- `parseMessage.pl`
  This program takes a single mail message, and writes it in a format understandable by the email classifier program, i.e. it removes the headers except the `To`, `From` and `Subject` part.
- `preFilter.pl`
  This program filters the emails, and removes unnecessary information. We basicly only leave letters, numbers and white space.
- `markovFilter.pl`
  This filter takes the filtered body of a message, and writes it into a proper format for the hidden markov model (compare it with parseMessage.pl for the email classification).
- `removeTags.pl`
  This filter simply removes the tags added to all the email messages to train the hidden markov model.
- `typeFilter.pl`
  This filter converts numbers and dates to tags, such as `|month|`. We used this program to see if we could increase the accuracy of the hidden markov model. Also, this method gives us less dependency on the training data. Even if we have never had a speech on a Thursday, the model can still say it is a date.
- `sumres.pl`
  This program takes the output of the hidden markov model and converts the date and the time to a standard format. Without this conversion, we can not extract the information automatically later, with the program `summary.pl`.

- `testinstance.pl`
  This programs runs the hidden markov model for extraction, with all necessary filters applied automatically. The input should be the body of a normal mail.
- `globfiles.pl`
  To increase the accuracy of the email classification system, it is better to have fewer categories. For example, instead of having `Seminars`, `Private`, `School`, etc, it is enough to use `Seminars` and `OTHERS`. This program will take directories with email files and put them together so that the email classification program only trains on two categories.

References

[1] Manning, C. and Schütze, H. (1999) Foundations of Statistical Natural Language Processing. The MIT Press, England.

[2] Lecture notes from the course `cs224n` at Stanford University, spring quarter 2000 with teacher Manning.

[3] Almgren, M. and Berglund, J. (2000) Email Classification with TF-IDF in a Naive Bayesian Framework with Handcrafted Features, *http://www/ almgren/cs229/report.html*

[4] Maes, P. (1994) Agents that reduce work and information overload, *Communications of the ACM*, 37(7):30–40

[5] Mitchell, T. M., Caruana, R., Freitag D., McDermott, J., and Zabowski, D. (1994) EExperience with a learning personal assistant, *Communications of the ACM*, 37(7):80–91

[6] Leek, T.R. (1997) Information Extraction Using Hidden Markov Models, *Master of Science Thesis, University of California, San Diego*

[7] Seymore, K., McCallum, A. and Rosenfeld, R. (1999) Learning Hidden Markov Model Structure for Information Extraction, *AAAI'99 Workshop on Machine Learning for Information Extraction. 1999.*

[8] Freitag, D. and McCallum, A. (1999) Information extraction using HMMs and shrinkage, *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*

[9] Freitag, D (1998) Multistrategy learning for information extraction, *ICML-98*

[10] Freitag, D. (1998) Information extraction from HTML: application of a general machine learning approach, *AAAI-98*

[11] Freitag, D. (1997) Using grammatical inference to improve precision in information extraction, *ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition, Nashville, July, 1997*