Information Extraction of Seminars

- Entity tagging -

1. Data preprocessing

Data preprocessing was required in order to split the header and the body of the e-mail. Keeping the code cleaner and well organized, a separate Python program was created named “parser.py” which implements the Parser class. This will return a tuple of lists, first one consisting of all main headers in the mail(e.g.: ”Abstract”, “Speaker” etc.) and the second list consisting of content of the mail(e.g.: text after Abstract header etc.). Knowing that python dictionaries change the inserting order, the tuple method was preferred instead of using dictionaries. This will help when the new tagged e-mail will be written in order to maintain the same order as in untagged version of the e-mail.

2. Paragraphs

Paragraphs recognition was implemented using the split method after finding two newlines characters. Hence, the code will be simply reduced as:

content.split("\n\n")

where content will be the abstract of the current considering e-mail.

3. Sentences

For sentences recognition, the nltk tokenize module was used:

tokenize.sent_tokenize(paragraph)

which will return a list of sentences extracted from the current paragraph. Extracting sentences procedure is based on the above procedure on splitting the text into paragraphs.

4. Extracting “stime” and “etime”

The header of an e-mail can contain valuable pieces of information about starting time and ending time of the seminar. As data is already stored in a data structure, tuple of lists, it’s easy to reach the header’s pieces. Time was extracted using regular expressions, but keeping in mind that there are more shapes that can suggest time, 2 regex were developed:

- **Only starting time**

  time_pattern = "(\d{1,2}\:?(\d{2})?\s*(([AaPp])[Mm])?)"
- **Both starting time and ending time**

  time_pattern + r'\s*-\s*' + time_pattern

Note the use of "?" as there can be derived a lot of patterns for expressing a specific time (e.g.: containing "am" or "pm", containing minutes or not etc.) and all of these are covered in the above regex. More than that, only the time referred to the seminar is searched in the data structure as firstly the program search for:

"([Tt][Ii][Mm][Ee]|[Ww][Hh][Ee][Nn])"

tab to not retrieve other times as posting time or so.

5. Extracting speaker and location

For seminar's speaker a new Python program was created, "nertagger.py" where a Named Entity Recognition was created in order to find both speaker and location of the seminar. This is because it's not necessary to be only speaker's name or location associated with the header tab, so this will remove the additional information. Firstly, in the Parser class a new function was implemented, ***get_training_sents***, which returns a dictionary with keys as ***"speaker"*** and ***"location"*** and values as the ones retrieved from the handtagged given data. This dictionary will be then used as training in the custom ner tagger. Moreover, for speaker, nltk.corpus.names was used as training data.

When a speaker header is found:

"([Ww][Hh][Oo]|[Ss][Pp][Ee][Aa][Kk][Ee][Rr])"

Or a location header is found:

"([Ww][Hh][Ee][Rr][Ee]|[Pp][Ll][Aa][Cc][Ee]|[Ll][Oo][Cc][Aa][Tt][Ii][Oo][Nn]"

The values is passed to the tagger.

There it is tagged using the nltk pos tagger:

nltk.pos_tag(word_tokenize(content))

This will be helpful later when the text will be split by nnp. The program will try to find the entity only if the chunk is nnp. Consecutive pairs of nnps will be considered together and also punctuation(e.g.:"," etc.) will be left as it is as it may be a useful hint.

The program checks first if the given data (which can be one nnp, two or so) was evaluated in the past. If it was, then it is evaluated again in the same manner.

In other case, text will be split in sets (e.g.: given the string "James Andrew Arthur", it will split it in ["James", "Andrew", "Arthur", "James Andrew", "Andrew Arthur"]) and will check again in the training data. Here, it will count the number of appearance in the training data for both location and person tags, and the most number will be the final tag (e.g.: if the sets scored 10 for person and 5 for location, it will be tagged as a person etc.).

If no appearance up to this point, there will be a ***wikification*** for every singleton from list and again count the appearance. This was implemented using ***dbpedia*** and also importing ***requests*** module. Here the number of person and location tags will be counted and the biggest one will decide the final tag.

If no entity found by now, then a ***grammar*** is there to help. Using hadtagged data, some patterns were selected (e.g.: "is presenting" etc.) and will be used in abstract to find the seminar's entities.

6. Evaluation of the program

Given the correct tagged mails, the "test_seminar.py" program was created which will output in "seminar_tagger_results.txt" the evaluation of the program consisting of precision, recall, f1 measure and accuracy. This was done by counting the true positive, true negative, false positive and false negative classification and apply the proper formulas.

	Stime	Etime	Sentence	Speaker	Location
Precision	67,45%	99,13%	67,91%	81,65%	25,49%
Recall	44,79%	62,16%	73,29%	26,25%	4,51%
F1 measure	53,83%	76,41%	70,5%	39,73%	7,66%
Accuracy	**45,47%**	**67,57%**	**54,53%**	**32,33%**	**10,31%**

Given the results, an overall accuracy of 42%, but in reality the tagger is much more efficient for identifying times and speaker as if there are other formats than the found one, it won't be tagged.

7. Conclusion

First of all, the ner tagger should have more training data to be more precise especially for location. Also, a grammar can be implemented as well for finding location when it's not in the header. More, given a time, all formats of that time should be tagged.

In conclusion, having more data for both speaker and location, ner tagger should be used as well in the abstract to find entities and also to be efficient. Because of lack of data, it is used now only for header.